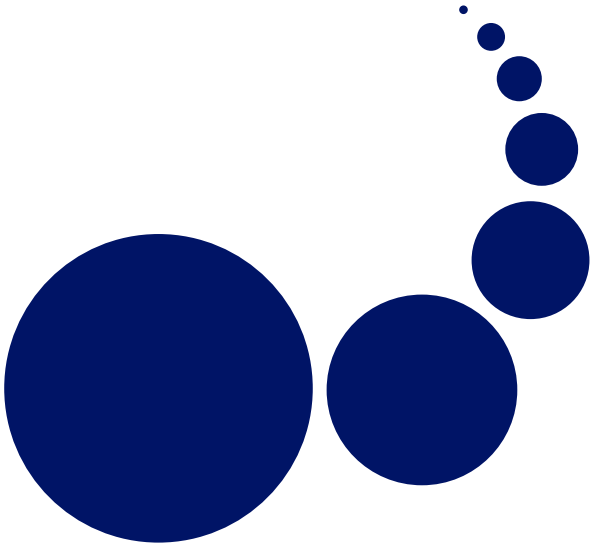
SCALABLE COMPUTING
Practice and Experience

Special Issue: Grid and Cloud Computing and their Application

Editors: Ewa Deelman, Norbert Meyer, Dana Petcu, and Marcin Paprzycki



Volume 11, Number 2, June 2010

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 11, Number 2, June 2010

TABLE OF CONTENTS

Special Issue Papers:

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. i–ii. http://www.scpe.org
ISSN 1895-1767

INTRODUCTION TO THE SPECIAL ISSUE: GRID AND CLOUD COMPUTING AND THEIR APPLICATIONS

Dear SCPE Reader,

We present a Special Issue on Grid and Cloud Computing and their Applications (http://www.scpe.org/?a=cfp_si&id=8). The first six papers included in this issue are modified and extended versions of papers presented at the 8th International Conference on Parallel Processing and Applied Mathematics, PPAM 2009 (http://www.ppam.pl), which took place on September 13–16, 2009 in Wrocław, Poland. These papers have been selected from contributions originally accepted in the frame of two workshops: the 4th Grid Applications and Middleware Workshop, GAMW'2009 (http://www.ppam.pl/?page=gamw) and the 5th Workshop on Large Scale Computations on Grids, LaSCoG'09 (http://lascog09.info.uvt.ro/). The remaining four papers were selected from the responses to an Open Call for Papers.

The selected papers are addressing a large variety of current research topics related to the call: distributed file systems, scheduling, load balancing, data mining, simulations, mobile agents, scientific portals, workflows, service negotiations or distributed visualization.

The first paper, "VOFS: A Secure Churn-Tolerant Grid File System" presents a specially designed secure file system that allows the members of a virtual organization to share files. A decentralized common file namespace is proposed to avoid a single point of failure. The proposed software stack includes a P2P system of file servers and can operate in a dynamic Grid environment

The second paper, "Matching Jobs With Resources: an Application-Driven Approach" proposes a distributed matchmaker, named GREEN, which provides Grid users with features for easy submission of job execution requests containing performance requirements. GREEN relies on a two-level benchmarking methodology: resources are characterized by means of their performance evaluated through the execution of low-level and application-specific benchmarks.

In the third paper, "Prediction and Load Balancing System for Distributed Storage," the application of a common mass storage system model in a national distributed storage system has been described. The prediction and load balancing subsystem, which provides advanced monitoring functionalities is discussed. The proposed system makes use of replication techniques to increase availability and performance of data access.

The fourth paper "Distributed Data Integration and Mining Using Admire Technology" presents the data integration engine for environmental data. The proposed software is being developed in the scope of the ADMIRE project. The proposed platform allows for integration of data from distributed, heterogeneous resources. It also allows users to construct reusable application processing elements specified in a DMIL, a language for data mining and integration.

The fifth paper "Ultra-Fast Carrier Transport Simulation on the Grid. Quasi-Random Approach" studies quasi-random number generation in a Grid-enabled package named Stochastic ALgorithms for Ultra-fast Transport in sEmiconductors (SALUTE). The performance of the corresponding algorithms on the Grid is also discussed. A large number of tests are reported on the EGEE and the SEEGRID Grid infrastructures.

In the sixth paper, "Management of High Performance Scientific Applications Using Mobile Agents Based Services," an explanation of how programmers can extend their applications to exploit services on heterogeneous and distributed platforms is provided. A native console is implemented, using mobile agents to control the application life-cycle. Moreover, software agents implement a mobile service that supports check-pointing, suspension, resuming, cloning and migration of managed applications.

The seventh paper "Vine Toolkit—Towards Portal Based Production Solutions For Scientific and Engineering Communities With Grid-Enabled Resources Support" addresses the challenge of synchronization of distributed workflows, and establishing a community driven Grid environment for the seamless results sharing and collaboration. The proposed toolkit offers user interface web components to be embedded in the existing portals, integration with a workflow engine, Grid security, and a built-in meta-scheduling mechanism allowing automatic load balancing among data centers to meet peak demands.

The eight paper "Fast Multi-Objective Rescheduling of Workflows to Constrained Resources Using Heuristics and Memetic Evolution" describes GORBA, a global optimising resource broker and allocator, which is designed to be used in a static planning environment. Several heuristics for rescheduling are introduced and their contribution to the overall planning process is studied.

The ninth paper "VieSLAF Framework: Facilitating Negotiations in Clouds by Applying Service Mediation and Negotiation Bootstrapping" presents a novel framework for the specification and management of service level

agreement (SLA) mappings and meta-negotiations facilitating service mediation, negotiation and bootstrapping in Cloud computing environments. The users may specify, manage, and apply SLA mappings without a-priori knowledge about negotiation protocols, required security standards or negotiated terms.

Finally, the tenth paper "Large Scale Problem Solving Using Automatic Code Generation and Distributed Visualization" presents a new approach to solving four important scalability challenges: programming productivity, scalability to large numbers of processors, I/O bandwidth, and interactive visualization of large data. The approach uses the Cactus framework, automated code generation, and numerical methods. A demonstration of the proposed system was awarded first place in the IEEE SCALE 2009 Challenge.

We would like to express our gratitude to all referees who have worked to help authors to improve the quality of papers selected to be published in this Special Issue.

Dana Petcu,
Ewa Deelman,
Norbert Meyer,
Marcin Paprzycki.
Special Issue Editors

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 99–108. http://www.scpe.org
ISSN 1895-1767

VOFS: A SECURE CHURN-TOLERANT GRID FILE SYSTEM*†

LEIF LINDBÄCK‡, VLADIMIR VLASSOV‡, SHAHAB MOKARIZADEH‡, AND GABRIELE VIOLINO§

Abstract. A Grid computing environment allows forming Virtual Organizations (VOs) to aggregate and share resources. We present a VO File System (VOFS) which is a secure VO-aware distributed file system that allows VO members to share files within a VO. VOFS supports access and location transparency by maintaining a common file namespace, which is decentralized to avoid a single point of failure in order to improve robustness of the file system. VOFS includes a P2P system of file servers, a VO membership service and a policy and role based security mechanism that protects the VO files from unauthorized access. VOFS can be mounted to a local file system in order to access files using a standard POSIX file API. VOFS can operate in a dynamic Grid environment (e.g. desktop Grids) since it is able to tolerate unplanned resource arrival and departure (churn) while maintaining a single uniform namespace. It supports transparent disconnected operations that allow the user to work on cached files while being disconnected. Furthermore, VOFS is a user level technique, and the current WebDAV-based VOFS prototype can operate under any operating system that has WebDAV mount support.

Key words: grid file system, virtual organization, peer-to-peer, security, namespace

1. Introduction. A Grid computing environment allows forming Virtual Organizations (VOs). A VO is a virtualised collection of users or institutions that pools their resources into a single virtual administrative domain, for some common purpose. A VO File System (VOFS) aggregates data objects (files, directories and disk space) exposed by VO members. *Expose* here means a VOFS operation to assign a data object (a directory or a file on a VO member's computer) a logical name in the VOFS namespace and make it accessible via a VOFS server.

One major challenge in such a file system is namespace management. The namespace should allow uniform and globally unique path names to be associated with data objects wherever they are located in the Grid [1]. Uniform here means access and location transparency of exposed data objects, and the same view of the file system at all nodes. This requires mapping a *logical name* of a file in VOFS namespace to its physical location. The global nature of grids enforces logical names to be uniform across different administrative domains.

In this work we consider ad-hoc grids built of resources voluntarily donated by VO members. VOFS contains different types of data objects exposed by VO members to be shared within a VO. This paper proposes a user-level solution for implementation of VOFS that allows exposing data objects, transparent access to the objects, and maintains the uniform namespace in the presence of resource churn (node leaves, joins and failures). The proposed VOFS has the following features that make it useful in ad-hoc Grids to create and maintain work spaces by exposing and sharing data objects by different applications and VO members.

1. VOFS includes a security mechanism that protects exposed data objects from unauthorized access. It supports VO membership management, authentication and role-based authorization according to VO policies including validity periods in access rights;
2. VOFS maintains a uniform namespace despite of unplanned resource churn;
3. The user-level technique of VOFS allows ordinary applications (file clients) to access the VOFS using a standard POSIX file API, i. e. the applications do not need to be modified to access files exposed to VOFS;
4. VOFS is easy to use for non-experienced users;
5. VOFS can operate under any operating system that has WebDAV [2] mount support, e.g. MS Windows, Linux, Mac OS X;
6. VOFS supports transparent disconnected operations that allow the user to work offline on cached files while being disconnected.

2. Overview. This work builds on our previous work presented in [3] that proposed three ways of maintaining the namespace: a centralized name service; a distributed directory; and a DHT-based name service. In [3] we have presented VOFS with the centralized name service that has the major disadvantage to induce a

*This paper is an extended version of the paper presented at GAMW 2009 in conjunction with the PPAM 2009 [17].

†This work was supported by the FP6 Project Grid4All funded by the European Commission (Contract IST-2006-034567).

‡School of Information and Communication Technology (ICT), Royal Institute of Technology (KTH), P.O. Box Forum 120, SE-164 40 Kista, Sweden. E-mails: leifl, vladv, shahabm@kth.se

§Net Result AB, Stockholm, Sweden. E-mail: gabriele.violino@gmail.com. Gabriele Violino was at the Royal Institute of Technology (KTH) while doing this work.

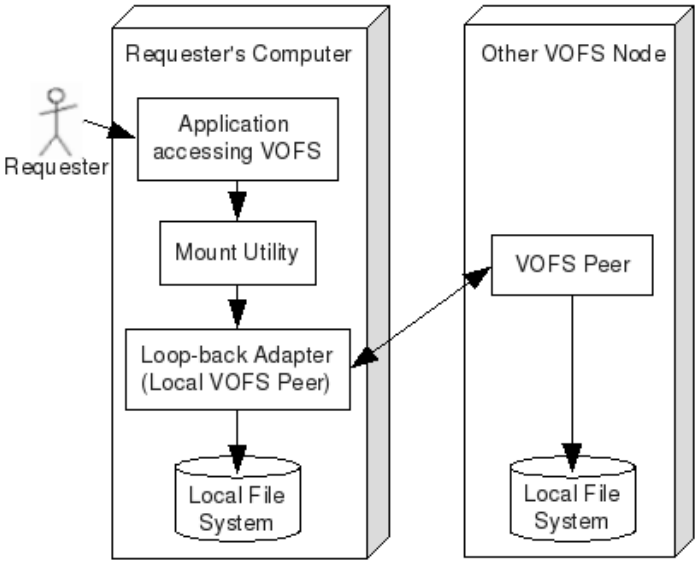


FIG. 2.1. *Schematic view of VOFS architecture*

single point of failure and a potential performance bottleneck. In this paper, we propose to build VOFS with a namespace maintained as a *distributed directory* where the namespace information is distributed among peers so that a peer knows location of *at least* those remote files which are exposed under directories hosted by the peer. In this VOFS design every peer can potentially learn the entire namespace (i. e. location of exposed data objects) via a gossiping mechanism.

In the current VOFS design we consider files and directories as data objects. The data objects can be exposed to any path in VOFS. An exposed directory offers disk space which is used by VO members to create new objects.

Exposing of a file or directory from the local node makes the data object accessible for VO members. Each peer runs a file server that provides and controls access to data objects exposed from the local node, see figure 2.1. Access to the exposed objects is achieved by mounting the local VOFS peer to a mount point, e.g. a local path. We use the WebDAV protocol [2] to access and transfer files between peers. Use of WebDAV allows accessing VOFS through any mount utility supporting WebDAV, e.g davfs2 [4] which offers a POSIX complaint API. Once mounted, access to VOFS is no different from access to local file system.

3. VOFS Namespace and File Tree. VOFS is formed as an ordinary hierarchical file tree by exposing data objects in to the VOFS tree, i. e. by assigning them paths in VOFS. The VOFS namespace is a set of mappings of logical names to physical locations. When a user exposes[1] a data object (a file or a directory) to the VOFS namespace, hence the VOFS file tree, the exposed file is assigned a logical name, which is a path in VOFS. The path may include names of virtual directories. A *virtual directory* is not hosted by any peer, i. e. it does not really exist. Thus, VOFS consists of exposed real data objects (directories and files) and virtual directories that may contain other virtual directories and exposed real data objects.

Initially, the VOFS tree contains only the root, which is initially virtual. The VOFS namespace, hence the VOFS tree, is formed explicitly and gradually as a result of exposing and unexposing data objects.

Virtual directories help to maintain the namespace, namely, to avoid possible namespace partitioning that might be caused by unexpose operations. If to assume that all directories in the VOFS tree are real (i. e. physically exist), then unexposing a real directory may cause partitioning of the tree as the data objects under the unexposed directory can not be properly identified by a path in the single-rooted VOFS tree. This motivates introducing virtual directories. The unexposed real directory becomes virtual; and names of all objects under it remain unchanged.

[1]The expose operation is described in Section 4.1

When looking up location of an object given its fully-specified VOFS path, a longest prefix match is done. The object can be accessed if the exposing peer is online despite of whether other peers are online or not.

Mappings of logical names to physical locations are the major metadata of VOFS. The metadata associates exported data objects with paths in the VOFS namespace. The same metadata are kept at every node in two tables: remote.db, which stores location information of data objects exposed by other peers; and local.db that stores location information of objects exposed by this peer. When a data object is exposed, the exposing peer adds a pair of local file system path and VOFS path to the local.db table while all other peers adds a pair of VOFS path and physical host address to their remote.db table. When a data object is unexposed, this information is removed from all peers. The namespace changes only when peers perform expose or unexpose operations. Peers communicate metadata by gossiping as explained below. All peers know the entire namespace, i.e which data objects are exposed and who exposes them.

3.1. Design Options for Avoiding Namespace Partitioning. When designing VOFS, we have considered the following three possible design options to manage objects (and their metadata) located under an unexposed directory in VOFS.

1. Unexpose all descendant data objects located under the unexposed directory;
2. Unexpose all data objects located under the unexposed directory that belong to the owner of the directory, keep objects of other owners;
3. Unexpose all data objects located under the unexposed directory that belong to the owner of the directory, and keep the unexposed directory as a virtual directory in the VOFS tree, if there are objects that belong to other owners.

Supporting the first design option (i. e. unexposing all objects under the unexposed directory), might be rather expensive, and might also cause violation of ownership of objects located under the unexposed directory. Note that in the second and the third option, all data objects under the unexposed directory, which do not belong to the owner of the directory, remain. However, the second design option results in a partitioned VOFS tree, i. e. a forest of trees, that complicates maintaining of the file system and its namespace. Remind that VOFS must tolerate frequent changes in its resources like node joins, leaves, and failures, and support frequent exposing and unexposing data objects in a rather convenient way without partitioning of the VOFS tree. Based on the above considerations, we have chosen the third above option, i. e. to keep an unexposed directory as virtual when needed (i. e. when it contains data objects that should remain in the tree) in order to avoid VOFS tree partitioning.

3.2. Algorithm for Namespace Updates. To transfer namespace updates between peers we use a gossip algorithm based on the *lazy probabilistic broadcast algorithm* described in [6]. When a peer updates the namespace it sends an *update* message to all or some of its neighbors. Each peer that receives an *update* message forwards it to all or some of its neighbours. There will be no loops since a peer never sends the same message twice.

There are no acknowledgements; instead the following recovery mechanism is used when messages are lost. Original sender id and a sequence number are attached to each message. Since there is FIFO delivery of messages, if a peer receives a message with a sequence number larger than the previous number plus one, it knows that some messages were lost. It will then send a *require* message to a subset of its neighbours. The *require* message indicates which message was lost and which peer is requiring it. A peer, which receives the *require* message checks if it has the required message. If yes, it sends the required *update* message to the requiring peer. If not, it forwards the *require* message to a subset of its neighbours. *Require* messages are forwarded only a specified number of times. Each peer maintains information about transmitted messages on its hard disk.

Note that the gossip algorithm described above is used only for namespace updates. All other communication, e.g. file transfer, involve only two peers.

Due to gossiping, there is no need to search for data objects since each peer maintains its own view of the namespace. The namespace view is almost the same as views of other peers even though there might be some inconsistencies between views caused by update latency.

4. VOFS Peers. Each user who exposes data objects must run a VOFS peer on her computer; while a user accessing VOFS does not need to run a VOFS peer. However, in the latter case, the user must know an address of any VOFS peer to be able to mount it and to access the VOFS. If the user runs a VOFS peer, then that local peer, *loopback adapter*, can be mounted to become the entry point to VOFS. In this case, there is no need to keep addresses of well-known mount points like in for example AFS [5]. Every of the VOFS peers

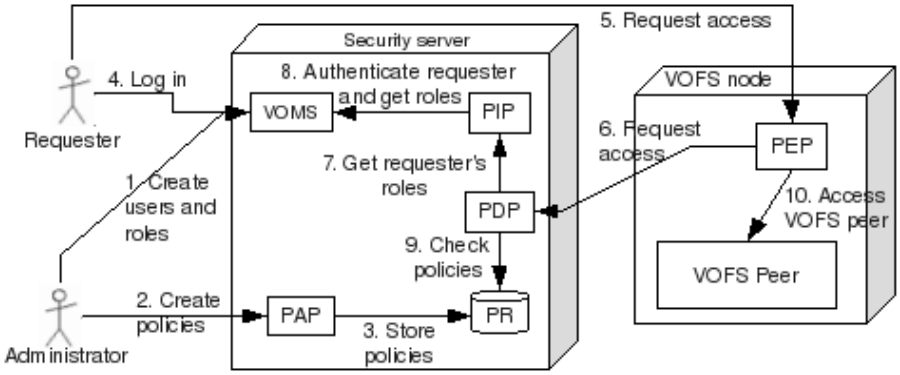


FIG. 5.1. *Interaction between security components*

provides the same set of the services that includes (un)expose, join, mount, cache. The services can be accessed by the user through the GUI of the VOFS peer. The services are described below.

4.1. (Un)Expose. A user (un)exposes data objects using an (un)expose client provided with a GUI in the current VOFS prototype. When exposing, the user defines a data object to be exposed and specifies its VOFS path. The expose service stores the logical-to-physical name mapping in the local table and initiates the update gossip algorithm. If the specified path does not exist, virtual directories are introduced in order to allow traversing the tree from root to the exposed data object. The root of VOFS is always `/`. It always exists at least virtually, but may also be mapped to a real directory. Name collision occurs when the user tries to assign a VOFS name which is already taken. In the current VOFS, the name collision is resolved as follows: if the data object to be exposed is a file, its mapping overrides the mapping of the object previously exposed with the same name; in case of directories exposed with the same name, their contents are merged.

4.2. Join. When a user starts a VOFS peer, the peer joins the P2P VOFS system. At startup, the peer downloads a list of all VO peers from the VO Membership Service (VOMS)[2]. Then the peer connects to some other peers selected from the list. The chosen peers and the new peer become neighbours. In the current VOFS prototype, selection of neighbours is random, but it could be done in a sophisticated way. They also exchange their VOFS views stored in their local and remote metadata tables described earlier. It is possible for the user to manually edit a peer's neighbour list through the GUI of the VOFS peer.

4.3. Mount. The user can mount VOFS with any mount utility supporting WebDAV used in the current VOFS; therefore we have not developed any special mount utility; instead, we use davfs [4] on Linux and NetDrive [7] on MS Windows. VOFS has not been tested on other OSs but Mac OS X has WebDAV support built in.

Once the VOFS is mounted, all POSIX file API is supported for manipulating data objects (provided the mount utility offers a POSIX API). The mount utility will translate the POSIX calls to WebDAV calls to the VOFS peer.

4.4. Cache. Each VOFS peer maintains a file cache. Read and write latency over network is compensated by the caching mechanism, which also allows offline work. VOFS uses *last write wins* reconciliation policy (a traditional file system policy for concurrent writes), which, if needed, can be replaced by a more sophisticated reconciliation policy implemented using, for example, Telex [8]. The cached copy is checked for update (compared to the master copy) when the file is read. When a file is written the new content is both stored in the cache and sent to the exposing peer, which informs all other peers who cached the file about the update. Also directory listings are cached, but unlike files they have an expiry time.

5. VOFS Security. VOFS includes a policy-based security, which ensures that only VO members can access files in VOFS. Access rights in VOFS confirm to VO policies set by resource (file and directory) owners.

[2]described in Section 5.1

The security infrastructure used in VOFS to protect exposed data objects from unauthorized access is based on the XACML authorization model [9]. The VOFS security allows to define and to enforce VO access control policies. Its goal is to provide authentication and authorization according to VO security policies. When authenticating the user's credentials are checked and the user gets a token which can be used to prove her identity in authorization checks. Authorization guarantees that users can only access resources to which they have right according to VO policies. Authorization is policy-based, policies are expressed in XACML.

5.1. Security Components. The VOFS security infrastructure is built of the following components.

Virtual Organization Membership Service, VOMS keeps a database of users and roles in the VO. It has a web based management interface for updating this data. This interface is protected by a PEP. The VOMS is also responsible for authenticating users.

Policy Enforcement Point, PEP protects a resource (VOFS peer, VOMS, PAP). Each resource has a local PEP, which is called whenever access rights shall be checked. On each request, the PEP forms an authorization request to PDP (or to its local cache) that includes the following three parameters.

1. *Subject*, which is the single sign-on identifier of the user accessing the data object. This identifier was returned by the VOMS when the user signed on;
2. *Action*, which specifies a name of an action (e.g. open) to be performed on the resource;
3. *Resource*, which is the target resource (file or directory) identifier in the form of a file path.

The PEP sends authorization requests to the PDP, and, upon receiving an authorization response from PDP, it enforces the authorization decision that can be either *Permit* or *Deny*. PEP caches the answers from PDP for further use. Caching of PDP responses at PEPs reduces security overhead. The PEP cache is invalidated by PDP when the access policies are changed. In order to further improve performance, the PDP answers not only to the request sent by the PEP, but to requests with the same subject and resource with all existing actions.

Policy Decision Point, PDP evaluates requests from PEPs according to the policies in PR, it makes the authorization decisions, and returns them to the requesting PEPs as authorization responses. A PDP response includes one of the following possible results:

1. *Permit*, this means that access is granted;
2. *Deny*, this means that access is rejected;
3. *NotApplicable*, this means that there was no matching policy;
4. *Indeterminate*, this means that no decision could be taken. For example there might be several contradicting policies.
5. *Error*, which means that policies could not be checked because of some exception, for example the communication link might be broken.

For efficiency, the PDP maintains a cache of policies (policy objects), which are loaded from the Policy Repository. Invalidation of the PDP's cache also invalidates all PEP's caches.

Policy Information Point, PIP contacts VOMS to validate the requester's identifying token and get the requester's roles. The answer from VOMS is cached, together with the lifetime of the token.

Policy Repository, PR stores the policies as XACML files.

Policy Administration Point, PAP is a server that makes updates to PR. The PAP is protected by a PEP. VOFS prototype includes a PAP client that allows the user to set an access control list for a given data object (directory) in a way similar to AFS [5], i. e. by issuing the `setact` command or via the client GUI for a given directory for a given role (user). The access rights set by the PAP client are applied to all data objects (files and directories) under the specified directory. The PAP client allows also specifying time and date for validity periods in access rights. The access rights set by the PAP client are stored in XACML policy files in the Policy Repository accessed by PDP. For example, the following command

```
setacl -dir @/se/kth -acl teacher rwid -time 01:00:00 13:00:00
```

gives the specified role `teacher` permissions to read, write, insert and delete (specified as `rwid`) under the directory `@/se/kth`. The permissions are valid from 1 AM to 1 PM.

We suppose that except for PEP there will be only one instance of each component per VO. Each PEP should be placed on the same host as the resource the PEP protects.

5.2. Scenario of Interaction with Security Components. A typical scenario of interactions between security components and VOFS peers is depicted in Figure 5.1. We distinguish four different phases: creating users and roles, creating security policies, authentication and access control.

Creating users and roles (1) The VO administrator uses the VO Membership Service, VOMS to create users and roles.

Setting policies (2) The administrator uses the Policy Administration Point, PAP to create policies. (3) The PAP stores the policies in the Policy Repository, PR. The PAP will invalidate the Policy Decision Point, PDP's cache. It can be specified in a policy when it is valid. This can be specified as time, date and day of week ranges and any combination of these.

Authentication (4) The requester logs in to the VOMS, using a web based interface. If the requester is authenticated, VOMS returns a token that is stored on the requesters's computer.

VOFS access (5) The requester uses an application that accesses VOFS. The mount utility sends the token along with the call to VOFS. The call is intercepted by the PEP which protects the VOFS peer. (6) The PEP asks the PDP whether the requester is allowed to access the peer. (7) The PDP asks the Policy Information Point, PIP for the requesters's roles. (8) The PIP contacts the VOMS to check if the token is valid and to get the user's roles. (9) The PDP evaluates the policies stored in PR. (10) If access was granted, the call is let through to the VOFS peer.

5.3. Secure Communication. The goals of secure communication are

1. To guarantee that PEPs get answers from the correct PDP;
2. To guarantee that PDP gets answer from the correct VOMS;
3. To guarantee that the token identifying a user is not stolen. If it is stolen it can be used to impersonate that user.

The first two goals can be met using certificates to identify PDP and VOMS. Regarding the third goal, there are the following risks that the token is stolen:

1. During transfer (this risk is eliminated with encrypted communication);
2. From the user's computer;
3. By a malicious node pretending to be a VOFS peer;
4. By another VOFS peer.

The second risk can be reduced if the VOMS encrypts the token with the user's public key. Before the token is passed to another peer it is decrypted with the user's private key. This means it is not possible to steal the token from a file in the user's local file system, unless also the user's private key is stolen.

The third risk is that someone writes a program that is not a VOFS peer but can issue correct commands to join the VOFS. If other peers believe it is part of the VOFS and communicates with it, it will get tokens of other users. This risk is eliminated if peers only communicate with other peers that can prove they are allowed by VOMS to take part in VOFS. To achieve this it is necessary that all peers can prove their identity using a certificate signed by a trusted certificate authority, CA. Such a certificate will contain the host address of the peer and will be issued by the VOMS that runs the trusted CA. Each peer will get its certificate from the VOMS at startup, VOMS maintains a list of allowed peers.

The fourth risk is that a trusted VOFS peer is compromised by a malicious user that changes it to report calling peer's tokens. This risk can not be eliminated since the purpose of passing the token *is* to let the receiving peer impersonate the user of the calling peer. The risk can be reduced in the same way as it is reduced using proxy certificates [10], by restricting the life time of the token and by delegating only a subset of the delegator's rights.

None of the above solutions require the user to be aware that certificates are used. This makes the VOFS easy to use also for non-experienced users.

6. Implementation of VOFS prototype. The VOFS prototype is implemented using Java Servlets, hence it can be executed on all platforms supporting Java Servlets. The prototype is bundled with Apache Tomcat. All that is needed to start it is to specify the PDP location and to start tomcat.

Figure 6.1 shows main components of the VOFS prototype briefly described below.

PEP is a servlet filter that intercepts all incoming requests. It translates the WebDAV method of the call to a VOFS operation and calls PDP (not shown in the figure) to check if the operation is permitted. If not, an HTTP 403 (forbidden) code is returned.

WebdavServlet is the access point for remote peers and the local mount utility.

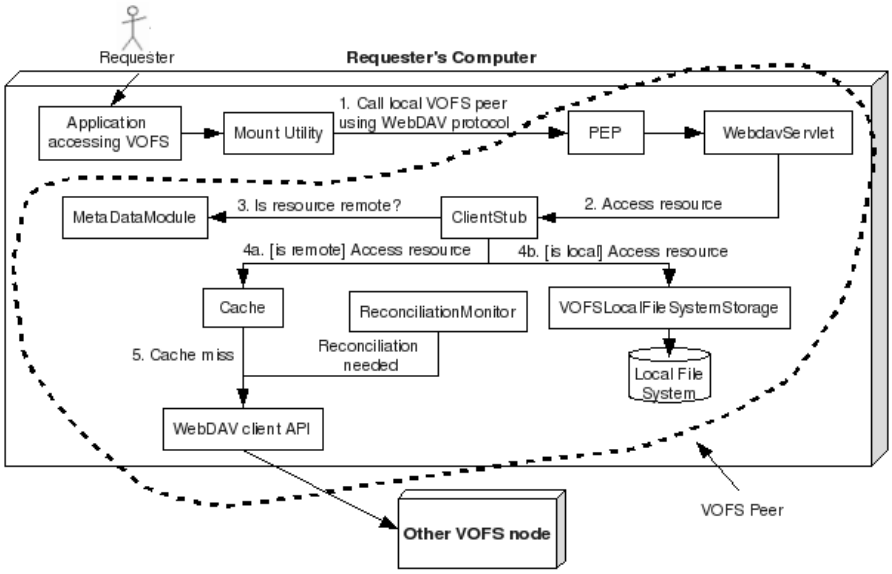


FIG. 6.1. *VOFS implementation*

ClientStub is a component that receives requests from WebdavServlet and forwards them to the correct component.

MetaDataModule keeps meta-data, see section 3, and offers the longest prefix matching engine.

LocalFileSystemStorage is a component that provides access to exposed files and directories.

Cache caches remote data objects. If a searched object is not in the cache the call is forwarded to the remote peer hosting it. The returned object is cached.

WebDAV client API is used (responsible) for contacting remote peers to read or write data objects.

ReconciliationMonitor is a component that continuously monitors the cache to see if an item in the cache is newer than the master, if so updates the master.

7. Related Work. Sprite Network File System [11] is a distributed file system similar in some aspects to VOFS. Meta-data (location information) in VOFS with decentralized name service is handled in a similar way to Sprite. However; the scopes of the two file systems are different: Sprite is designed to operate within LANs; whereas VOFS should operate over WANs. A main difference between VOFS and Sprite is that Sprite does not handle partitioning of the file tree since lookup for a file starts from root and proceeds downwards; whereas in VOFS longest prefix match is done on the entire path. VOFS allows virtual directories for keeping VOFS operational while at least one real object is in the tree. This feature and support for disconnected operation makes VOFS churn tolerant. Moreover, in Sprite every node exports resources under predefined prefixes and specific sub-directories in the tree while in VOFS a node can expose anywhere in the tree.

There exist peer-to-peer (P2P) file systems, e.g. OceanStore [12], which were developed as a file storage (data store) for file sharing. A typical P2P file system is used to store/retrieve files without support for neither POSIX file API access (i. e. the systems are not mountable), nor security. Grid file systems in contrast to P2P file sharing systems strongly require authentication and authorization to protect files from unauthorized access. VOFS allows the VO members to define and set VO security policies to be enforced by the VOFS security infrastructure.

Examples of Grid file systems include gLite file catalogs [13], Gfarm [14], and Distributed File Services, DFS [15]. The gLite *file catalogue service* [13] is used to maintain location information about files and their replicas. In contrast to VOFS, gLite catalogue service is centralized and is a single point of failure. The Gfarm file system [14] uses a virtual tree and virtual directories mapped to physical files by a metadata server, like the *centralized* solution described in [3]. Gfarm is designed to be very scalable; however, its metadata server can become a bottleneck and is a single point of failure since it is not replicated in contrast to VOFS. DFS [15] is a P2P file and storage system that can be integrated with a Grid security mechanism. DFS, in contrast to the presented VOFS, has no hierarchical namespace, but instead offers two P2P networks: one for storage space

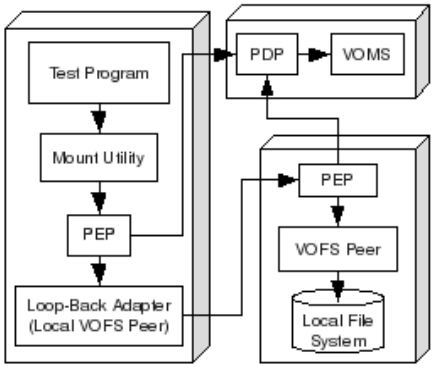


FIG. 8.1. *Setup of performance test*

and one for names and metadata. DFS is implemented using FUSE [16] that limits its usage only to Linux, while the WebDAV-based VOFS can run on multiple (if not all) operating system, e.g. MS Windows, Linux, Mac OS X.

Recently a number of cloud service providers and developers of cloud environments offer storage cloud solutions, which, in particular, provide storage (e.g. Amazons S3 [18]) and storage services based on that storage (e.g. Dropbox [19]) that allow to store, access and share files for many users for a reasonable price at any time from anywhere, with high-quality of service guarantees. The cloud-based storage is an attractive storage solution for end-users, and it can also be integrated and used in Grids. However there are certain issues (common for any cloud solutions) to be considered when storing data on a cloud environment, namely, trust and a single vendor lock-in, as data are stored in the cloud storage provided by a cloud provider, rather than on the computers of data owners or data sharers. In contrast to a cloud-based storage service, e.g. Dropbox, VOFS is formed of and builds on resources (storage) donated by VO members, and therefore one can expect that members of the VO trust each other in providing a secure file system. From security perspective, Amazon S3 API provides both bucket (fine grained) and object level (coarse grained) access controls, while VOFS does not support the notion of bucket level data management, hence; only provides object level access. Obviously, security mechanism in VOFS is not comparable with Amazon S3 API as file system space in VOFS is provided by volunteer collaboration between VO members which can not prevent back-door (unauthorized) access to data stored in a host machine or cached on a VOFS peer. However this undesired situation can be alleviated as we consider an implicit trust relationship between VO members. Also, it should be clear that a VO member without permission to see a particular file will never be able to download that file. Hence, files are only stored on computers of members who are allowed to see the files. Finally, an advantage of the VOFS security model is that files are never stored on a server. Files will only exist on the computers of the file owner and the VO members who are allowed to see the file. It is also worth noting that VOFS is open and free (no vendor lock-in and only the members resources are used,) and that the VOFS is a file system, not just a storage area, that is the VOFS maintains a file tree built by its users data owners.

8. Performance. In order to evaluate performance of the VOFS prototype, we have performed a number of evaluation experiments on the setup shown in Figure 8.1. The nodes used in the evaluation experiments are PCs with 1.86 GHz Intel Centrino CPUs and 1 GB RAM on a dedicated 100 Mbps LAN. We have evaluated performance of namespace updates, file transfer and file lookup.

We have done two measurements of the namespace update algorithm (see Fig. 8.2). The first measurement concerns updates without lost messages. It shows how long time it takes for an update message to reach a node that is one, two, four and eight network hops away from the updating node. The second measurement shows recovery of missed namespace update messages due to a node being disconnected from the node performing the updates. Figure 8.2 shows how long it takes to get information about all namespace updates performed while the node was disconnected. This is measured with one, two, four and eight missed update messages. The time is reduced if all lost messages are required and resent with one message, now there is one require and one resend per lost message. Figure 8.2 shows that the algorithm scales well.

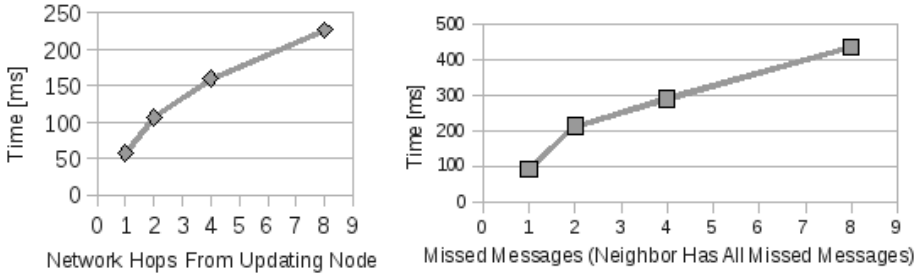


FIG. 8.2. *Namespace update performance.*

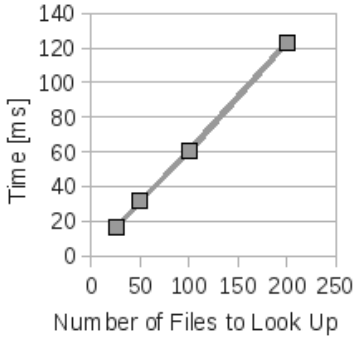


FIG. 8.3. *Lookup performance.*

Figure 8.3 shows how long time it takes to find out which node exposes a given file. This is a local operation, since all nodes have information about the entire namespace. The lookup time is about 0.7 ms per file no matter how many files are looked up.

The read test copies 100 files from a remote peer to the local file system (outside VOFS). The file cache is big enough to contain all files. The results are presented in Fig. 8.4. The figure also depicts timings for copying files within the local file system in order to compare performances of the local file system and VOFS. Bandwidth when transferring smaller files is lower because overhead takes proportionally more time. The overhead is mainly due to that the mount utility (davfs2) reads file properties before transferring files.

The write test copies 100 files from the local file system (outside VOFS) to a remote peer. Results of write test are in Fig. 8.4. Cache does not speed up performance since file content is written both to cache and to remote peer.

9. Conclusion. We have presented a churn tolerant VOFS that maintains a uniform namespace in a dynamic environment, that is when nodes frequently join or leave the VOFS. The VOFS provides a shared workspace for VO members and it is easy to use. It includes VO membership management, authentication and authorization. The VOFS Prototype is available at `http://www.isk.kth.se/~leifl/vofs/`.

Acknowledgments. Special thanks to Chen Xing (chenxing@kth.se) who implemented the first version of the VOMS and the gossip based protocol.

REFERENCES

[1] O.T. Anderson et al: Global namespace for files, IBM systems Journal Vol 43, No 4 (2004)
[2] WebDAV Community, `http://www.webdav.org/`
[3] Hamid Reza Mizani, Liang Zheng, Vladimir Vlassov, Konstantin Popov: Design and Implementation of Virtual Organization File System for Dynamic VOs. Proceedings of the 2008 11th IEEE International Conference on Computational Science and Engineering - Workshops - Volume 00, pp 77-82 (2008)
[4] davfs2, mount utility for WebDAV on Linux, `http://dav.sourceforge.net/`
[5] Howard, John H: An Overview of the Andrew File System. Winter 1988 USENIX Conference Proceedings, pp. 23-26 (1988)
[6] Rachid Guerraoui, Luís Rodrigues: Introduction to Reliable Distributed Programming. Springer-Verlag, Berlin Heidelberg (2006)

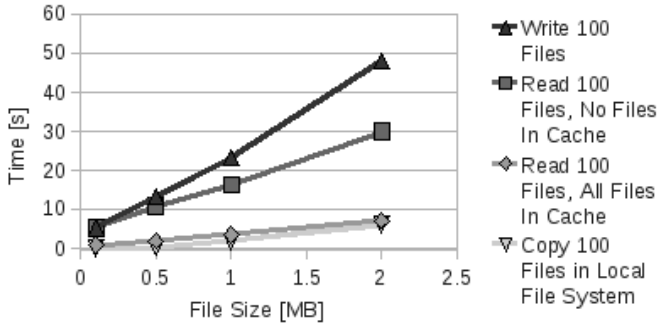


FIG. 8.4. *File transfer performance.*

[7] NetDrive, mount utility for WebDAV and FTP on MS Windows, `http://www.netdrive.net/`
[8] Lamia Benmouffok, Jean-Michel Busca, Joan Manuel Marqus, Marc Shapiro, Pierre Sutra, Georgios Tsoukalas: Telex: Principled System Support for Write-Sharing in Collaborative Applications. Research Rapport, INRIA RR-6546 (2008)
[9] Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, Tim Freeman: A Multipolicy Authorization Framework for Grid Security. in Proceedings of the Fifth IEEE Symposium on Network Computing and Application, pp 269-272 (2006)
[10] Von Welch, Ian Foster, Carl Kesselman, Olle Mulmo, Laura Pearlman, Steven Tuecke, Jarek Gawor, Sam Meder, Frank Siebenlist: X.509 Proxy Certificates for Dynamic Delegation. In Proceedings of the 3rd Annual PKI R&D Workshop (2004)
[11] Brent Welch, John Ousterhout: Prefix Tables: A Simple Mechanism for Locating Files in a Distributed System. Report No. UCB/CSD 56/261, Computer Science Division, University of California, Berkeley, California (1985)
[12] Sean Rhea, Patrick Eaton, Dennis Geels, Hakim Weatherspoon, Ben Zhao, and John Kubiatowicz: Pond: the OceanStore Prototype. Proceedings of the 2nd USENIX Conference on File and Storage Technologies, San Francisco, pp 1-14 (2003)
[13] gLite, middleware for grid computing, `http://glite.web.cern.ch/glite/`
[14] Osamu Tatebe, Satoshi Sekiguchi, Youhei Morita, Noriyuki Soda, Satoshi Matsuoka: GFARM V2: A Grid File System that Supports High-Performance Distributed and Parallel Data Computing. Computing in High Energy Physics and Nuclear Physics, Interlaken, Switzerland, pp.1172 (2004)
[15] Antony Chazapis, Georgios Tsoukalas, Georgios Verigakis, Kornilios Kourtis, Aristidis Sotiropoulos, Nectarios Koziris: Global-scale peer-to-peer file services with DFS. Grid Computing, 2007 8th IEEE/ACM International Conference on, pp 251-258 (2007)
[16] FUSE: Filesystem in Userspace, `http://fuse.sourceforge.net/`
[17] Leif Lindback, Vladimir Vlassov, Shahab Mokarizadeh, and Gabriele Violino: Churn Tolerant Virtual Organization File System for Grids, the 4th Grid Applications and Middleware Workshop (GAMW'2009) in conjunction with PPAM 2009, Wroclaw, Poland, September 13-16, 2009—to appear.
[18] Amazon Simple Storage Service (Amazon S3), `http://aws.amazon.com/s3/`
[19] Dropbox - Home - Online backup, file sync and sharing made easy, `https://www.dropbox.com/`

Edited by: Ewa Deelman
Received: March 30, 2010
Accepted: May 25, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 109–120. http://www.scpe.org
ISSN 1895-1767

MATCHING JOBS WITH RESOURCES: AN APPLICATION-DRIVEN APPROACH

A. CLEMATIS[†], A. CORANA[‡], D. D'AGOSTINO[†], A. GALIZIA[†], AND A. QUARATI[†]

Abstract. We present a distributed matchmaking methodology based on a two-level (low-level and application-level) benchmarking, that allows the specification of both syntactic and performance requirements. In particular, we point out how the use of application-level benchmarks gives a more accurate characterization of resources, so enabling a better exploitation of Grid power. The proposed methodology relies on the use of standard description languages at both application and resource sides, to foster interoperability. Moreover, the proposed tool is independent of the underlying middleware, and its distributed structure supports scalability.

Key words: grid platforms, benchmark-driven resource matchmaking, job submission languages extensions, interoperability

1. Introduction. Grid platforms supply users with a very large number of different resources to execute demanding applications. To exploit at best Grid power, efficient query and discovery tools are needed, able to provide a good matching of user requirements with resource characteristics. Unfortunately, Grid middleware offer only basic services for the retrieving of information about single resources, and thus they are often inadequate to describe more detailed and specific user requirements. So, usually, a matchmaking component (e.g. broker, matchmaker) manages over the middleware this supply-demand coupling process [1].

Some general criteria must be followed to provide a suitable and effective Grid matchmaker: a concise but as complete as possible description of application needs and resource properties, grounded on a common and shared basis, to assure interoperability; the management of both syntactic and performance requirements; the independence of the underlying middleware; a distributed structure, to allow scalability.

During past years we developed the tool GEDA (Grid Explorer for Distributed Applications), based on a distributed approach for Grid resource discovery, which combines a structured view of resources (single machines, homogeneous and heterogeneous clusters) at the Physical Organization (PO) level with an overlay network connecting the various POs [2, 3]. The GEDA architecture is modular and independent of the particular Grid middleware, although we worked with Globus Toolkit 4 [4]. The system is particularly suitable for discovering resources for structured parallel applications on large Grids.

To enhance the efficiency of the tool we develop a methodology to improve the matchmaking process based on information about performance of resources. Our aim is to supplement the basic information available via the Grid Information and Monitoring services by annotating resources with both low-level and application-specific performance metrics. These relevant aspects of resources could be examined by a broker to filter out the solutions that best fit application requirements.

Indeed, benchmarking is a widespread method to measure and evaluate performance of computer platforms [5]. Particularly, application-specific benchmarks are widely acknowledged tools in the High-Performance Computing (HPC) domain, to measure the performance of resources stressing simultaneously several aspects of the system. Notwithstanding, so far application benchmarks have not been extensively considered on the Grid, owing to various problems, such as very diversified types of applications, architectural complexity, dynamic Grid behavior, and heavy computational costs [6].

On this basis, we design GREEN (GRid Environment ENabler), a Grid service which represents an enhanced version of GEDA, whose main improvement is the management of benchmarks for a more precise characterization of resources. GREEN is a distributed matchmaker which complies with the above specifications, useful both for Grid administrators and users. It assists administrators in the insertion of benchmark information related to every PO composing the Grid, and provides users with features which a) facilitate the submission of job execution requests, by specifying both syntactic and performance requirements on resources; b) support the automatic discovery and selection of the most appropriate resources. The aim of GREEN is the discovery of the resources that satisfy user requirements and their ordering by performance ranking. The selection phase is left to a (meta)scheduler, allowing to apply the preferred scheduling policies to meet specific purposes.

An important point of our work is the use of both low-level and application-level benchmarks. Indeed, often it can occur that the rankings of resources based on low-level and application-level benchmarks are different,

[†]IMATI-CNR, Via De Marini 6, 16149 Genova, Italy
[‡]IEIIT-CNR, Via De Marini 6, 16149 Genova, Italy

with the second one usually closer to the effective performance obtainable by the user application. In this sense the use of application-level benchmarks allows a better exploitation of Grid resources.

Another important design goal of GREEN is interoperability. To this end, a unique standard language, namely JSDL (Job Submission Description Language) [7], is used to express job submission requirements, and an internal translation to the job submission languages used by the various middleware is performed. Middleware independence is pursued through an extension of JSDL in conformity with the GLUE (Grid Laboratory for a Uniform Environment) schema v. 2.0 [8]. Moreover, since we are interested in the execution of parallel applications, we borrowed from JSDL SPMD [9] some extensions to JSDL related to concurrency aspects.

This paper summarizes design principles, and provides an extended and modified version of the work [10]; main extensions regard the presentation of some experimental data on two different high performance platforms, and some preliminary results highlighting the usefulness of the proposed approach in a Grid environment.

The paper is organized as follow. Section 2 gives a brief overview on the state of the art about matchmaking and benchmarking on the Grid; Section 3 discusses the main contributions in the field of job and resource characterization languages. Section 4 briefly outlines the two-level benchmarking methodology, while Section 5 reports some preliminary experimental data collected on two parallel machines enlightening the usefulness of our approach. Section 6 gives a proof of concept of its adoption. Section 7 describes the design issues of GREEN, and an analysis of the extensions operated to existing languages. Section 8 gives some concluding remarks.

2. Related Works. The implementation of an efficient and automatic mechanism for the effective discovery of the resource that best suits a user job is one of the major problems in present Grids.

An important requirement is scalability, that is assured avoiding centralized structures; for example in [11] the Vigne tool is proposed, whose main features are a simple abstract view of resources, an application manager which selects resources using a resource allocator based on scalable and distributed discovery, and a decentralized overlay network. However, the tool does not support benchmark information.

The Globus toolkit does not provide a resource matchmaking/brokering as a core service, but the GridWay metascheduler [12, 13] was included as an optional high-level service since June 2007. GridWay provides dynamic scheduling, performance slowdown detection, opportunistic and on request migration, and fault recovery mechanisms. The main drawback of GridWay is that it allows users to specify only a fixed and limited set of resource requirements, most of them related to the queue policies of the underlying batch job systems. This choice limits the ranking of resources, and benchmarks are not considered at all.

On the contrary, gLite has a native matchmaking/brokering service that takes into account a richer set of requirements, including benchmark values [14]. However, this service is based on a semi-centralized approach, and may result in long waiting time in the job execution. Moreover, at the moment only the SPEC benchmark suite is considered, which mainly evaluates CPU performance; thus, the description of resources is partial, and can be inadequate to specific application requirements.

Work Binder [15] is a tool developed for the gLite middleware, based on the use of pilot jobs and aimed at assuring to incoming applications a fast access to computing resources; the tool is specifically designed to support interactive applications and on-demand computing, and can be adapted for different middleware.

A way to improve the efficiency of resource discovery, is to drive the search towards resources that shown good performance in the execution of jobs with similar or known behaviour. As explained in Section 3, the characterization of Grid resources based on pre-computed benchmarks seems a valid strategy to follow. The importance of benchmarking to evaluate resources in a Grid environment is largely acknowledged together with the criticalities that this task implies [16]. Actually, besides the set of interesting parameters to measure (e.g. CPU speed, memory size) different factors have to be taken into account when considering the execution of a benchmark suite on a Grid.

Several works proposed tools to manage and execute benchmarking on Grid. The Grid Assessment Probes [17] attempt to provide an insight into the stability, robustness, and performance of the Grid. The probes are designed to serve as simple Grid application exemplars and diagnostic tools. They test and measure performance of basic Grid functions, including file transfer, remote execution, and Grid Information Services (ISs) response.

The GridBench [18] is a modular tool aimed at exploring large-scale Grids in a interactive manner, taking into account performance aspects, adding new metrics to the basic ones supplied by middleware. It provides a graphical interface to define, execute and administrate benchmarks, also considering interconnection performance and resource workload. GridBench makes use of plug-ins to assure interoperability with the various middleware (currently Globus and gLite).

The NAS Grid Benchmark (NGB) suite [19] is defined by NASA, and represents typical activities of Computational Fluid Dynamics applications. It provides a set of computationally intensive benchmarks representative of scientific, post-processing and visualization workloads, and tests the Grid capabilities to manage and execute distributed applications. It uses four kinds of data-flow graphs according to parallel paradigms extracted from real applications in NASA.

All tools described above do not provide mechanisms for the submission of jobs and for their matching with resources. A brokering mechanism based on benchmarking of Grid resources is proposed in [20]. However, the scope of that broker is focused on the ARC middleware and the NorduGrid and SweGrid production environments, and it adopts xRSL, an extension of RSL (Resource Specification Language), to submit user's jobs. As a consequence, this approach lacks in generality and interoperability.

3. Resource and Job Characterization. To accomplish the matchmaking task, a proper description of resources is required at resource/owner and job/user side. To this end, different projects and research groups have proposed different languages.

At the resource side, adequate information is required to advertise resource's static (e.g. OS, number of processors) and dynamic (e.g. number of executing tasks, amount of free memory) properties. Actually, the main efforts in the direction of a standard resource description language come from the GLUE Working Group, which deployed the GLUE schema [8]. It is a conceptual model of Grid entities comprising a set of information specifications for Grid resources; an implementation through an XML Schema is given in [21]. As the schema has evolved during years, different versions have been used by various middleware, leading to the GLUE 2.0 specification. It allows the benchmarking characterization of resources by specifying the `Benchmark_t` complex type referencing benchmarks of type defined by `BenchmarkType_t`. Through the latter, declared as an open and extensible enumeration type, it is possible to specify a benchmark amongst a list of six values (e.g. specint2000, specfp2000, cint2006). However, other values compatible with the string type and with the recommended syntax are allowed.

At the user side, a job submission request expressed via a Job Submission Language (JSL), in addition to stating the application-related attributes (e.g. name and location of source code, input and output files), should express syntactic requirements (e.g. number of processors, main memory) and ranking preferences (if any) to guide and constraint the matching process on resources.

The Job Description Document (JDD) [22], introduced by Globus Alliance with the Web Services versions of the Globus Toolkit, defines an XML language closer to the XMLish dialects used in the Web Services Resource Framework (WSRF) family. The main purpose of a JDD document is to set the parameters for the correct execution of a job. The selection of the facilities to use has to be performed in advance by interacting with the WS MDS services of the available resources. In the JDD schema, it is possible to specify only few requirements, as the minimum amount of memory, or to set useful information as the expected maximum amount of CPU time. It is however possible to extend the schema with user-defined elements.

The European Data Grid Project proposed the Job Description Language (JDL), afterwards adopted by the EGEE project [23]. A JDL document contains a flat list of argument-value pairs, specifying two classes of job properties: job specific attributes and resources-related properties (e.g. `Requirements` and `Ranks`) used to guide the matching process towards the most appropriate resources. These values can be arbitrary expressions using the fields published by the resources in the MDS, and are not part of the predefined set of attributes for the JDL, as their naming and meaning depend on the adopted Information Service schema. In this way, JDL is independent of the resources information schema adopted.

The Job Submission Description Language (JSDL) developed by the JSDL- Working Group [7] of the Global Grid Forum, aims to synthesize consolidated and common features available in other JSLs, obtaining a standard language for the Grid. JSDL contains a vocabulary and normative XML Schema facilitating the declaration of job requirements as a set of XML elements. Likewise JDL, job attributes may be grouped in two classes. The `JobIdentification`, `Application` and `DataStaging` elements describe job-related properties. The `Resources` element lists some of the main attributes used to constraint the selection of the feasible resources (e.g. `CPUArchitecture`, `FileSystem`, `TotalCPUTime`). Since only a rather reduced set of these elements is stated by the JSDL schema, an extension mechanism is foreseen. Examples of JSDL extensions able to capture a more detailed description of the degree of parallelism of jobs are presented in [9, 24].

In Section 7 we present our proposal aimed at extending GLUE and JSDL with elements capable of accounting for the association of benchmarks data at both user and resource sides.

4. A Two-Level Benchmarking Methodology. To describe Grid resources, we propose a two-level methodology aimed at giving a useful enriched description of resources, and at facilitating the matchmaking process. Our methodology considers two approaches: I) the use of micro-benchmarks to supply a basic description of resource performance; II) the deployment of application-driven benchmarks to get a closer insight into the behavior of resources under more realistic conditions of a class of applications. Application-driven benchmarks can consist of:

a) the application itself, often in a light version obtained choosing a reference input data set and specific parameters to avoid long executions, obtaining in the meantime a representative run of the real application;
b) a suitable benchmark or benchmark suite belonging to the same class of the application of interest (e.g. the LINPACK benchmark for the class of linear algebra applications).

Through application-driven benchmarks, it is possible to add an evaluation of the resources on the basis of the system indicators that are more stressed by an application. Our present aim is to provide a proper description of each Grid resource in isolation, i. e. without considering complexity aspects of Grid environments. Future developments of our work foreseen to address more complex scenarios.

4.1. Micro-Benchmarks. In order to supply a basic resource characterization, mainly based on low-level performance capacity, we consider the use of traditional micro-benchmarks. To this aim, a reasonable assumption is that the performance of a machine mainly depends on the CPU, the memory and the cache, and on the interconnection network [25]; therefore, we choose a concise number of parameters to evaluate in order to provide an easy-to-use description of the various nodes. Table 4.1 shows resource properties and related metrics measured by the micro-benchmarks we employed.

TABLE 4.1
Low-level benchmarks and related metrics.

Resource Capability	CPU	Memory	Memory-Cache	Interconnection	I/O
Metric	MFLOPS	MB/sec	MB/sec	MB/sec	MB/sec
Benchmark	Flops	Stream	CacheBench	Mpptest	Bonnie

Flops provides an estimate of peak floating-point performance (MFLOPS); Stream is the standard benchmark for measuring sustained memory bandwidth, as it works with datasets much larger than the available cache; CacheBench is designed to characterize the performance of possibly multiple levels of cache present on the processor; Mpptest measures the performance of some basic MPI communication routines; Bonnie performs a series of tests on a file of known size (default 100 MB).

The micro-benchmarks used in this phase generally return more than one value; so, to obtain results easily usable in the matchmaking process, we considered for each benchmark synthetic parameters or the most significant value. They are used to characterize resources by populating the benchmark description managed by GREEN.

4.2. Application-Specific Benchmarks. Micro benchmarks are a good solution when the user has little information about the job she/he is submitting, and for applications that are not frequently executed. Indeed, very often the participants to a Virtual Organization have similar aims, and therefore it is possible to identify a set of the most used applications. In these cases the most suitable approach is to evaluate system performance through application-specific benchmarks that approximate at best the real application workload. This kind of benchmarks represent the second level of our methodology.

As case studies for this level we considered some applications of our interest, i. e. image processing, isosurface extraction, and linear algebra. For the first two classes of applications, we choose a light version code aiming to emphasize precise aspects of the considered metrics. With respect to image processing, we selected a compute intensive elaboration applied to a reference image of about 1 MB; in this way CPU metrics are mainly stressed. The isosurface extraction application provides a more exhaustive performance evaluation of the system, as also I/O operations are heavily involved. In this case, we considered the processing of a small 3D data set of 16 MB, producing a mesh made by 4 million triangles. On the contrary, to represent the class of applications based on linear algebra, we used the well known Linpack benchmark [26]. For application-driven benchmarks, the metric considered to characterize resources is execution time. Similarly to the micro-benchmarks case, the results are stored in the internal data structures of GREEN.

5. Benchmarking resources. To evaluate the effectiveness of our methodology in resource characterization, we performed some experiments on several resources normally used to deploy and execute our applications. For sake of simplicity, here we focus on two specific high-performance resources: 1) the Michelangelo system,

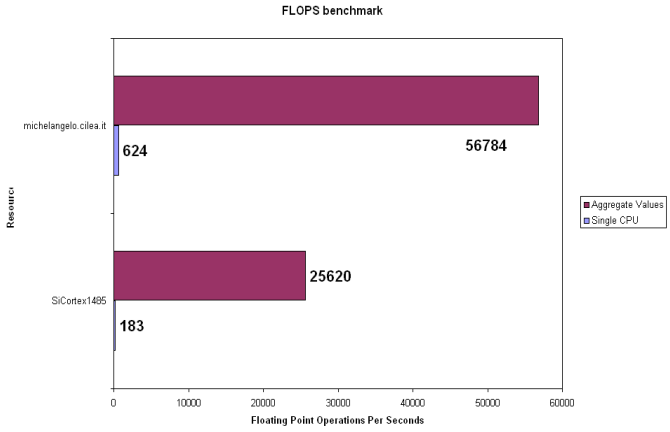


FIG. 5.1. *Comparison between resources according to FLOPS benchmark.*

made up of 53 nodes interconnected by a Gigabit switched Ethernet. As a whole, the system provides 212 AMD Opteron 275 dual core with a clock rate of 2.2 GHz. Each CPU is equipped with 2 GB RAM, and the total shared storage amounts to 30 TB [27]. 2) the SiCortex SC1458 system with 243 SiCortex node chips, each equipped with six cores, and linked by a proprietary interconnection network supporting a large message bandwidth of 4 GBytes/sec. This system pursues the Green Computing guidelines, through extremely low energy consumption [28].

By a quick comparison clearly emerges that the two resources greatly differ both in terms of the total number of CPUs and in terms of single CPU performance. In fact, SC1458 has a greater number of CPUs than the Michelangelo cluster, but the latter has faster CPUs. Despite these technical differences from which one may infer consequent performance results, this expectation is contradicted by our experiments as shown by the following discussion.

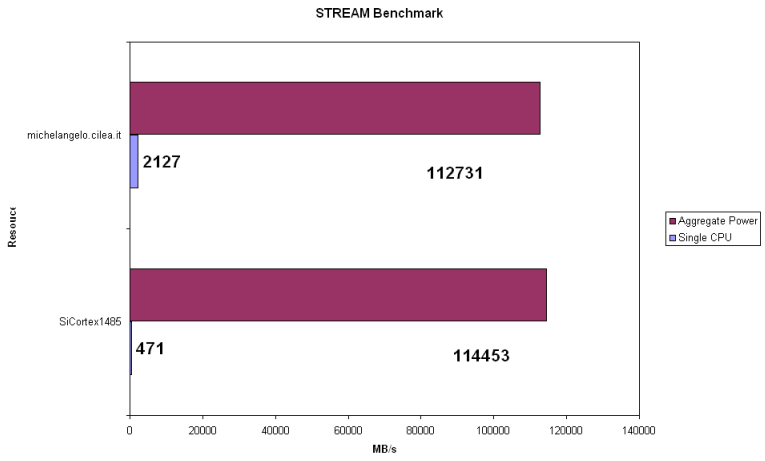


FIG. 5.2. *Comparison between resources according to STREAM benchmark.*

Starting from micro-benchmark results, the SC1458 achieves better performance in almost all cases and parameters evaluated, when considering aggregate computing power. However, its single cores have relatively low performance compared with the single CPU of the Michelangelo cluster, and the actual power of the resource

derives from the high number of provided cores and the native fast connection among processes. To outline CPU performance we depicted in Figures 5.1 and 5.2 the results obtained with FLOPS and STREAM benchmarks.

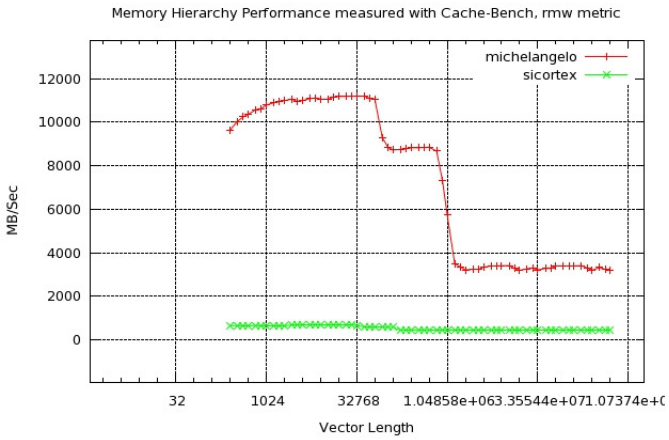


FIG. 5.3. *Comparison between resources according to CacheBench.*

Both benchmarks have been run on a CPU/core independently, and then the aggregated results are gathered to represent the performance of the whole parallel resources [16].

With respect to the cache evaluation, Figure 5.3 shows that Michelangelo performs better for all vector lengths. On the contrary, with respect to interconnection evaluation, the SC1458 achieved definitely better performance, as reported in Figure 5.4. We tested point-to-point communication performance, through the MPPTest benchmark; results are expressed in MB/Sec. As mentioned above, the Michelangelo Cluster employs a Gigabit switched Ethernet, while SC1458 has a proprietary interconnection network that performs significantly better.

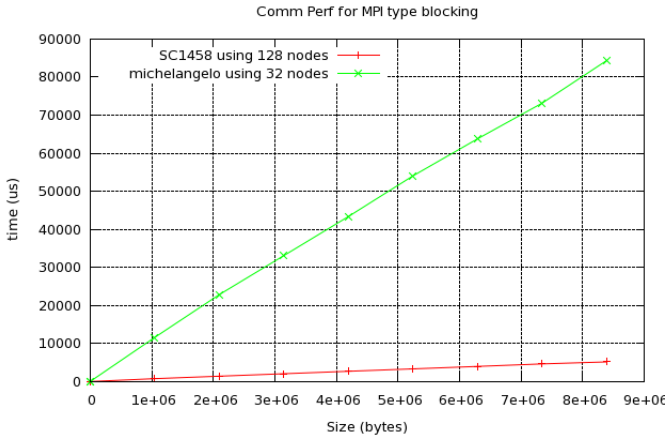


FIG. 5.4. Comparison between resources according to MPPTest benchmark.

Considering the second level of benchmark, the situation is quite different. In fact, depending on the application domain, best results are achieved alternatively by the two resources.

We conducted our tests considering the execution times (Wall Clock Time) as metric to evaluate performance. The results are normalized according to a base value; to this end, we adopted the values returned from the Michelangelo cluster. Tables 5.1 and 5.2 report the values obtained for Image Processing (IP), Isosurface Extraction (IE) and High-Performance Linpack (HPL) benchmarks. In the latter case, we examined separately the use of different sets of processors (32, 64 and 128 for both). Due to the chosen metric, lower values correspond to better execution times.

TABLE 5.1
Application-level benchmarks, execution time normalized with respect to Michelangelo.

	Michelangelo	**SC1458**
Isosurface Extraction	1	6.1
Image Processing	1	3.2

TABLE 5.2
Application-level benchmarks, execution time normalized with respect to Michelangelo.

	Michelangelo			**SC1458**		
	32p	**64p**	**128p**	**32p**	**64p**	**128p**
HPL	1	0.3	0.2	0.44	0.13	0.08

Table 5.1 shows that Michelangelo cluster performed significantly better considering the Image Processing and the Isosurface Extraction applications. Instead, Table 5.2 reporting the results related to the HPL benchmark highlights that SC1458 outperforms Michelangelo up to a factor 3, when increasing the number of processes. This behaviour depends on the different requirements of the various applications. As to Image Processing and Isosurface Extraction resulted that they benefit from fast single CPU and cache memory, while HPL tests the entire system and benefits from high number of processes linked with fast connections. Starting from these remarks, it is quite evident that the Michelangelo cluster is faster in the execution of IP and IE, while it poorly performs with respect to HPL. On the contrary, with respect to HPL, SC1458 outdoes the Michelangelo Cluster, but it does not achieve good results on the considered image processing operations and isosurface extraction.

Following our methodology, the differences in the performance of both resources in each level of benchmark clearly emerge. SC1458 definitely outperforms Michelangelo with respect to almost all micro-benchmarks. However, considering the second level of benchmark, the Michelangelo cluster appears as the suitable choice for the execution of specific applications. This performance divergence also occurred in other similar comparisons we conducted for the other benchmarks executed against the resources normally used to deploy and execute our applications. This behaviour testifies the appropriateness of our approach.

6. A proof of concept. To exemplify the potentialities of our methodology, we introduce a simplified evaluation that highlights the benefits of adopting a benchmark aware matchmaker. Let us consider a simplified Grid scenario setting three jobs (J1, J2 and J3) belonging to the Image Processing and Linear Algebra classes, that are executed against SC1458 and Michelangelo, denoted R1 and R2 respectively.

Table 6.1 reports the relative spped of resources R1 and R2 with respect to the two classes of applications. As shown in the previous Section, Michelangelo performs three times better than SC1458 for Image Processing applications, while it is about two times slower than SC1458 for Linear Algebra applications.

TABLE 6.1
Relative speed of resources with respect to the applications.

Applications	**R1**	**R2**
Image Processing	3	1
Linear Algebra	0.43	1

Table 6.2 lists the jobs. In particular, J1 and J3 are two Image Processing jobs, and J3 is computationally heavier since it processes a larger image. J2 is a Linear Algebra job. For each job the computational time required on the two resources is reported, expressed in seconds; the first column also shows the temporal instant at which the job is submitted.

In Figure 6.1-(a) we depicted the case in which our benchmark-driven methodology does not apply, no matchmaker operates and jobs are scheduled on the first available resource in a FIFO order. In this case, when J1 arrives at time 0, it is mapped to R1 which employs 60 seconds to process it. At time 10, J2 arrives and is mapped to R2, which executes it in 80 seconds. When J3 arrives at time 20 no resources are available, hence J3 is queued until one is released. This happens at time 60, when J1 terminates and J3 is assigned to R1 which takes 120 seconds to run it, ending at time 180.

TABLE 6.2
Job characteristics.

Job	Comp. Time on R1 (sec)	Comp. Time on R2 (sec)	Application
J1(0)	60	20	Image Processing
J2(10)	35	80	Linear Algebra
J3(20)	120	40	Image Processing

Let us now consider the case in which our methodology is applied and benchmark information is used to describe resources, see Figure 6.1-(b). Now the matchmaker is at work and, when asked for a suitable resource to execute the Image Processing job J1, it returns R2, since it ranks better than R1 on that job class. Running three times faster, R2 employs just 20 seconds to execute J1. When J2 arrives at time 10, R1 is selected and it takes about 35 seconds to complete the Linear Algebra job. Finally, J3 arrives and it is assigned to R2 that in the meantime becomes free, and the job ends after 40 seconds. The overall computation in this case ends at time 60.

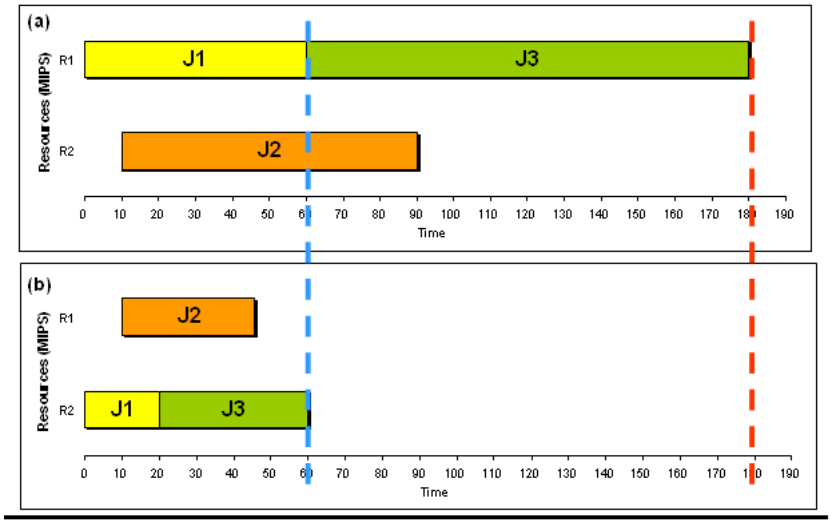


FIG. 6.1. *(a) scheduling of three jobs without any matchmaker, (b) improvement of execution times when a benchmark-driven matchmaker is applied.*

From our example it clearly emerges that the use of benchmark information could be adopted to improve scheduling strategy, raising the performance of the overall execution.

7. Benchmark-Driven Matchmaking. Due to the huge gap separating users and resources, tools that allow the two parts to better come to an agreement are highly useful. In [2, 3] we presented GEDA, a Grid service based on a distributed and cooperative approach for Grid resource discovery. It supplies users with a structured view of resources (single machines, homogeneous and heterogeneous clusters) at the PO level, and leverages on an overlay network infrastructure which connects the various POs constituting a Grid. For each PO, a GEDA instance is deployed to keep updated information about the state of all PO's resources, and to exchange them with other GEDA instances in the discovery phase.

In the present work, we describe an advanced version of GEDA, called GREEN, able to characterize Grid resources through benchmark evaluations. In this context, acting as a distributed matchmaker, GREEN manages and compares the enriched view of resources with user-submitted jobs, with the goal of selecting the most appropriate resource(s). Operating at intermediate level between applications (e.g. schedulers) and Grid middleware, GREEN aims to discover the whole set of resources satisfying user requirements ordered by ranks. The selection of a particular resource is left to a (meta)scheduler to which the resources set is forwarded; so it is possible to apply the preferred scheduling policies to optimize Grid throughput or other target functions

(e.g. response times, QoS). Once the "best" resource is chosen, GREEN will be re- invoked to carry-out the submission of the job on it, via the Execution Environment (EE).

7.1. Benchmarking Grid Resources. GREEN supplies Grid administrators with the facility of submitting, executing benchmarks (both micro and application-related) against the resources belonging to a certain administrative domain (PO), and storing results.

To reflect the underlying view of Grid resources offered by the GLUE 2.0 specification language, and to support the matching mechanism (i. e. the comparison with resources information contained in the previously acquired XML) the benchmark-value copies are directly represented as GLUE entities according to the XML reference realizations of GLUE 2.0. By employing the openness of `BenchmarkType_t` (as recalled in Section 3), the set of recognized benchmarks is extensible without any change in the document schema. An example of a benchmark document related to the execution of micro-benchmark Flops against the cluster identified by the IP 150.145.8.160, resulting in 480 MFlops is:

```
<Benchmark>
      <LocalID>150.145.8.160</LocalID>
      <Type>MFlops</Type>
      <Value>480</Value>
      <BenchLevel>micro</BenchLevel>
</Benchmark>
```

Through the use of the extension mechanism defined in GLUE specification, we enriched the `Benchmark_t` type by adding the element `BenchLevel` which specifies the benchmark level (by accepting the two string values `micro` and `application`) according to our two-level methodology.

Once a benchmark is executed and its results collected, an XML fragment, similar to the one reported above, is created for each resource and inserted in an XML document (namely *Benchmark image*), managed by GREEN, which collects all benchmark evaluations for the PO.

7.2. Extending JSDL. The counterpart of benchmarking resources is the ability for users submitting a job to express their preferences about the performance of target machines. Resources are then ordered according to performance values (ranks). As explained in Section 3, both JDD and JSDL do not provide any construct able to express some preferential ordering on selected resources. We add the element `Rank` (of complex type `Rank_Type`) devoted to this task, which embeds a sub-element `BenchmarkType_t` corresponding to the one contained in our extension of the GLUE schema. In the context of JSDL, the `Value` sub-element (see list below) is to be intended as a threshold to be satisfied by the corresponding `Value` (related to the benchmark stated by `Type`) contained in the `Benchmark` element of any resource to be selected by the matchmaker before the ranking takes place.

As we are interested in the execution of parallel applications, we borrowed from SPMD [9] an extension to JSDL that supports users with a rich description set of applications and resources related to concurrency aspects (e.g. number of processes, processes per host). The following is an example of an extended JSDL document, containing information related to parallel requirements, along with our extension to rank resources on benchmark specification. The document is requesting for nodes able to execute the application-level "IsoSurface_Benchmark" in no more than 300 time units. Note how the `Rank` element has been located inside the `Resource` one, according to the extension mechanism included by JSDL schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
xmlns:jsdl-spmd="http://schemas.ogf.org/jsdl/2007/02/jsdl-spmd"
xmlns:jsdl-
rank="http://saturno.ima.ge.cnr.it/ima/PONG/jsdl/2009/01/jsdl-rank">
  <jsdl:JobDescription>
    <jsdl:Application>
      <jsdl:ApplicationName>ParIsoExtrctn</jsdl:ApplicationName>
      <jsdl-spmd:SPMDApplication>
        <jsdl-posix:Executable>parisoextraction</posix:Executable>
```

```
        <jsdl-posix:Argument> inputvolume.raw</posix:Argument>
        <jsdl-posix:Argument>200</posix:Argument>
        <jsdl-posix:Output>isosurface.raw</posix:Output>
        <jsdl-spmd:NumberOfProcesses>4</spmd:NumberOfProcesses>
        <jsdl-spmd:ProcessesPerHost>2</spmd:ProcessesPerHost>
        <jsdl-spmd:SPMDVariation>http://www.ogf.org/jsdl/2007/02/
                                  jsdl-spmd/MPICH2</>
      </jsdl-spmd:SPMDApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <jsdl:OperatingSystemType>
        <jsdl:OperatingSystemName>LINUX</jsdl:OperatingSystemName>
      </jsdl:OperatingSystemType>
      <jsdl-rank:Rank>
        <jsdl-rank:Type>IsoSurface_Benchmark</rank:Type>
        <jsdl-rank:Value>300</rank:Value>
        <jsdl-rank:BenchLevel>application</rank:BenchLevel>
      </jsdl-rank:Rank>
    </jsdl:Resources>
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

7.3. Distributed Matchmaking Process. The main components of a GREEN instance along with some of their interactions with other middleware services, notably IS and EE, by considering a Grid composed of several POs are shown in Figure 7.1. In the following, we summarise their roles and the behaviours.

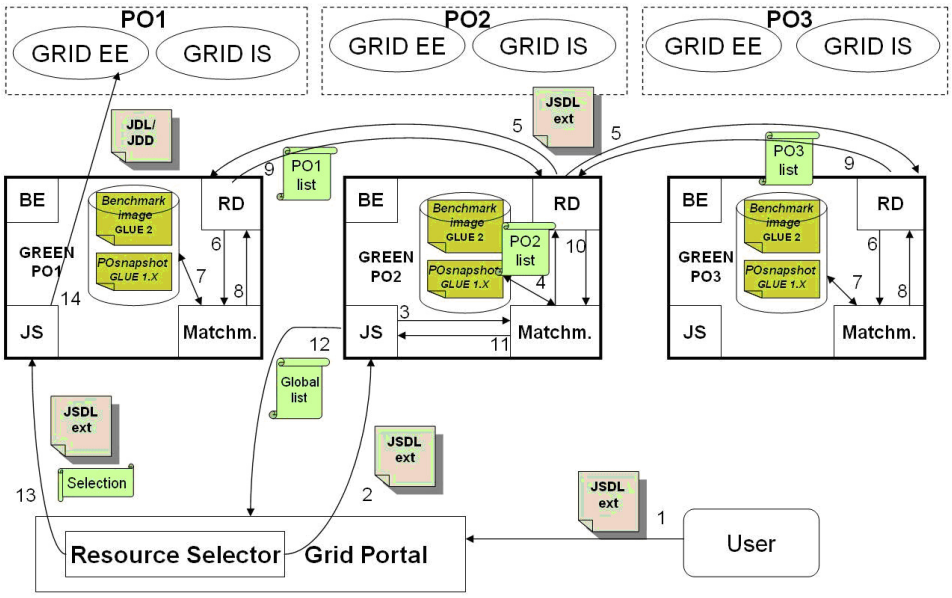


FIG. 7.1. Example of the matching phase with various GREEN instances.

The Job Submission (JS) component receives requests of jobs submission initiated by users; depending on the activation mode it behaves just like a messages dispatcher or as a translator of JSL documents, carrying out their subsequent submission to the EE. The Benchmark Evaluation (BE) supports administrators in the performance-based characterization of PO resources. The Resource Discovery (RD) is in charge of feeding GREEN with the state of Grid resources. RD operates both locally and globally by carrying out two tasks: 1) to discover the state of the PO resources; 2) to dispatch requests to other GREEN instances. As to the

first task, RD dialogues with the underlying IS (e.g. MDS, gLite IS) that periodically reports the state of the PO in the form of an XML file conformed to the GLUE version adopted by the underlying middleware. This document (namely the *PO snapshot*) is stored, as it is, and managed by GREEN to answer to external queries issued by various clients (e.g. other GREEN instances, meta-schedulers). To accomplish the dispatching task, RD handles the so-called neighbors view. Depending on the number of POs, i. e. GREEN instances running, their management could consider different strategies, whose description is beyond the scope of the paper.

To deal with different underlying middleware transparently to Grid users and applications, the syntactic differences among the various versions of GLUE are managed by GREEN through a conversion mapping at matching time. The Matchmaker performs the matching among resources in the Grid, and their subsequent ranking, with the requirements expressed by the users through the application submission document.

More in detail, let us consider the case in which a user submits an extended JSDL document through a Grid portal (1). The document is managed by the Resource Selector component, which initiates the distributed matchmaking by forwarding it to the JS component of a randomly selected GREEN instance (2) (e.g. PO2). JS activates the Matchmaker (3). This instance of matchmaker, namely the Master Matchmaker (MM), is responsible to provide the set of candidate resources to the Resource selector for this specific request. MM through RD forwards the document to all the other known GREEN instances and contemporaneously checks its local memory (4-5). All the matchmakers filter their *PO snapshot* selecting the set of PO resources satisfying the query. By analyzing the pre-computed *Benchmark image*, the satisfying resources with a `Value` element (for the chosen benchmark) that fulfils the threshold fixed in the corresponding `Rank` element of the JSDL document are extracted. The resources identifiers and their corresponding benchmark values are included in a list, called *PO list* which is returned to MM (6-10). MM merges these lists with its own PO list, producing a Global List ordered on the ranking values. The Global list is passed to JS (11) which returns it back to RS (12). Besides applying the selection policy to determine the resource to use, the Resource Selector calls the JS of the GREEN responsible of the PO owning the selected machine (GREEN PO1's instance in our case), by sending it the extended JSDL document along with the data identifying the selected resource (13). JS translates the information regarding the job execution of the original JSDL document in the format proper of the specific PO middleware, stating the resource on which the computation takes place. In particular, it will produce a JDD document for GT4 resources or a JDL document for the gLite ones. Finally, it activates the Execution Environment in charge of executing the job represented in the translated document (14).

8. Conclusions. The efficient matchmaking of application requirements with characteristics of resources is a very important issue in Grid computing, and developing a satisfactory solution may greatly improve the usefulness of Grid platforms for a large class of potential applications. However this is not an easy task, owing to the high abstraction level of Grid platforms which causes a semantic gap between application requirements and resource properties, the large number of possible Grid applications, the large number of available resources, the dynamicity of the Grid environment.

To reduce this gap we designed GREEN, a distributed matchmaker which provides Grid users with features to make easier the submission of job execution requests containing performance requirements, in order to support the automatic discovery and selection of the most suitable resource(s). GREEN relies on a two-level benchmarking methodology: resources are characterized by means of their performance evaluated through the execution of low-level and application-specific benchmarks. According to our methodology, every resource of a PO is tagged with the results obtained through the execution of the two levels of benchmarks and hence it is selectable, on a performance basis, during the matchmaking phase. A preliminary analysis outlined promising results; thus future efforts are planned in the direction of a deeper evaluation of our proposal in the context of a simulated Grid environment with particular emphasis on scheduling policies.

To ensure a good degree of independence from the underlying middleware, GREEN leverages on two standards such as JSDL and GLUE, that have been properly extended to manage the performance-based description of resources.

Acknowledgments. This work has been partially supported by project TECDOC-SIIT Under L.297 Miur Programme.

REFERENCES

[1] X. Bai, H. Yu, Y. Ji and D. C. Marinescu, *Resource matching and a matchmaking service for an intelligent Grid*, Int. Journal of Computational Intelligence, 1 (2004), pp. 163–171.

[2] A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi and A. Merlo, *Resource selection and application execution in a Grid: A migration experience from GT2 to GT4*, in Lecture Notes in Computer Science n. 4276 (OTM 2006), R. Meersman, Z. Tari et al., eds., Springer, Berlin, 2008, pp. 1132–1142.

[3] A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi, A. Merlo and A. Quarati, *A distributed approach for structured resource discovery on Grid*, in Proc. Int. Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008, Barcelona), IEEE Computer Society, 2008, pp. 117–125.

[4] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*, in Lecture Notes in Computer Science n. 3779 (Proc. IFIP Int. Conference on Network and Parallel Computing), Springer, Berlin, 2005, pp. 2–13.

[5] R. W. Hockney, *The science of computer benchmarking. Software, environments, tools*, SIAM, Philadelphia, 1996.

[6] F. Nadeem, R. Prodan, T. Fahringer and A. Iosup, *Benchmarking Grid Applications for Performance and Scalability Predictions*, in Grid Middleware and Services: Challenges and Solutions (CoreGRID series), D.Talia, R.Yahyapour and W.Ziegler, eds., Springer, 2008, pp. 19–37.

[7] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher and A. Savva, *Job Submission Description Language (JSDL) Specification v1.0*, Grid Forum Document GFD. 56, Open Grid Forum (OGF), 2005.

[8] S. Andreozzi et al., *GLUE Specification v. 2.0 (rev. 3)*, Open Grid Forum (OGF), 2009.

[9] A. Savva (Ed.), *JSDL SPMD Application Extension, Version 1.0*, Grid Forum Document GFD. 115, Open Grid Forum (OGF), 2007.

[10] A. Clematis, A. Corana, D. D'Agostino, A. Galizia and A. Quarati, *Performance based matchmaking on Grid*, in Lecture Notes in Computer Science n. 6068 (Proc. Int. Conf. on Parallel Processing and Applied Mathematics (PPAM 2009), Wroclaw (Poland), September 13-16, 2009), R. Wyrzykowski, ed., Springer, Berlin, 2010, pp. 174–183.

[11] E. Jeanvoine, L. Rilling, C. Morin and D. Leprince, *Using Overlay Networks to Build Operating System Services for Large Scale Grids*, Scalable Computing: Practice and Experience, 8 (2007), pp. 229-239.

[12] E. Huedo, R. S. Montero and I. M. Llorente, *A Framework for Adaptive Scheduling and Execution on Grids*, Software: Practice and Experience, 34 (2004), pp. 631–651.

[13] E. Huedo, R. S. Montero and I. M. Llorente, *The GridWay Framework for Adaptive Scheduling and Execution on Grids*, Scalable Computing: Practice and Experience, 6 (2005), pp. 1–8.

[14] *gLite 3.1 User Guide*, Doc. CERN-LCG-GDEIS-722398, January 2009 `https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.html`

[15] B. Marović, M. Potočnik and B. Čukanović, *Multi-application bag of jobs for interactive and on-demand computing*, Scalable Computing: Practice and Experience, 10 (2009), pp. 413-418.

[16] M. D. Dikaiakos, *Grid benchmarking: vision, challenges, and current status*, Concurrency and Computation: Practice and Experience, 19 (2007), pp. 89–105.

[17] G. Chun, H. Dail, H. Casanova and A. Snavely, *Benchmark probes for Grid assessment*, in 18th Int. Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe (USA), IEEE Computer Society, 2004, pp. 276.

[18] G. Tsouloupas and M. D. Dikaiakos, *GridBench: A Tool for the Interactive Performance Exploration of Grid Infrastructures*, Journal of Parallel and Distributed Computing, 67 (2007), pp. 1029–1045.

[19] M. Frumkin and R. F. Van der Wijngaart, *NAS Grid Benchmarks: A tool for Grid space exploration*, Cluster Computing, 5 (2002), pp. 315-324.

[20] E. Elmroth and J. Tordsson, *Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions*, Future Generation Computer Systems, 24 (2008), pp. 585–593.

[21] S. Andreozzi (Ed.), *GLUE v. 2.0 Reference Realizations to Concrete Data Models*, Open Grid Forum (OGF), 2008.

[22] M. Feller, I. Foster, S. Martin, *GT4 GRAM: A Functionality and Performance Study*, in Proc. 2007 TeraGrid Conference, 2007, Madison (USA) `http://www.globus.org/toolkit/docs/4.2/4.2.0/user/gtuser-execution.html`

[23] *Job Description Language Attributes Specification for the gLite Middleware*, Doc. EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8, May 2006 `https://edms.cern.ch/file/555796/1/EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8.pdf`

[24] I. Rodero, F. Guim, J. Corbal and J. Labarta, *How the JSDL can Exploit the Parallelism?*, in Sixth IEEE Int. Symposium on Cluster Computing and the Grid (CCGRID'06), 2006, pp. 275–282.

[25] G. Tsouloupas and M. Dikaiakos, *Characterization of Computational Grid Resources Using Low-level Benchmarks*, in Proc. 2nd IEEE Int. Conference on e-Science and Grid Computing, IEEE Computer Society, 2006, pp. 70.

[26] J. J. Dongarra, P. Luszczek and A. Petitet, *The LINPACK benchmark: Past, present, and future*, Concurrency and Computation: Practice and Experience, 15 (2003), pp. 1–18.

[27] Michelangelo Hardware, LITBIO Project `http://www.supercomputing.it/At_Cilea/michelangelo_eng.htm`

[28] SiCortex Home page, `http://sicortex.com/`

Edited by: Ewa Deelman
Received: March 30, 2010
Accepted: May 25, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 121–130. http://www.scpe.org
ISSN 1895-1767

PREDICTION AND LOAD BALANCING SYSTEM FOR DISTRIBUTED STORAGE

RENATA SŁOTA[†], DARIN NIKOLOW[†], STANISŁAW POLAK[†], MARCIN KUTA[†], MARIUSZ KAPANOWSKI[†], KORNEL SKAŁKOWSKI[†], MAREK POGODA[‡], AND JACEK KITOWSKI[†‡]

Abstract. National Data Storage is a distributed data storage system intended to provide high quality backup, archiving and data access services. These services guarantee high level of data protection as well as high performance of data storing and retrieval by using replication techniques. Monitoring and data access prediction are necessary for successful deployment of replication. Common Mass Storage System Model (CMSSM) is used to present a storage performance view of storage nodes in unified way for monitoring and prediction purposes. In this paper some conceptual and implementation details on using CMSSM for creating a Prediction and Load Balancing Subsystem for replica management are presented. Real system test results are also shown.

1. Introduction. National Data Storage (NDS) is a distributed data storage system intended to provide high quality backup, archiving and data access services [1]. These services are capable of providing high level of data protection, data availability and data access performance. In order to guarantee these capabilities replication techniques are used. Two problems arise with using this approach: selecting physical storage locations for newly created replicas and choosing the best replica for a given data transfer. If these problems get solved we can count on faster data access.

The client access to NDS is provided by Access Nodes (ANs). ANs spread over the country are located in national computer centers having direct links to the NDS Pionier backbone network [2]. The general idea is that client requests come via different ANs and the requested data is served by the most appropriate Storage Node (SN), selected separately for each request, being the one which can provide requested data fastest. In this way some natural load balancing is achieved depending on the client access pattern.

In order to provide the required high performance data access functionality a replica management system called Prediction and Load Balancing System (PLBS) was implemented. This paper presents some conceptual and implementation details on creating PLBS. Essential part of this research concerns replication and the development of replication policies, which should help achieving reasonable level of storage load balancing. These policies are based on CMSSM storage model [3] which allows to provide an unified layer for monitoring purposes. The potential scope of application of the proposed approach is wide, for example it could be used for multi-player on-line games [4] as well as for data storing from HEP experiments [5].

The rest of the paper is organized as follows: The next section presents the state of the art in the field of replication strategies and storage system modeling and monitoring. The third section describes the CMSSM storage model. Fourth section shows how CMSSM is adopted in the PLBS subsystem. The fifth section gives some overview of the replication strategies used in the system. The test results are presented in the sixth section and the last section concludes the paper.

The paper is an extended and modified version of the article [6] presented at the PPAM09 conference. The extension concerns: (i) description of CMSSM, (ii) showing how CMSSM is adopted in the PLBS subsystem, (iii) presenting new experimental results.

2. State of the Art. With the steadily growing users demand for data storage space and access quality the data management techniques become an important issue of any distributed computing environment [7]. In [8] we can find survey of data management techniques in various distributed systems. Scalability taxonomy of data management is also proposed. A part of the data management systems concerns data replication. There are many researches focused on replication strategies. In [9] five replication strategies for read only data are presented. The strategies have been tested using three different access patterns. The study assumes tiered network with a central data source. Similar network model having constant storage nodes locations in the network hierarchy is studied in [10]. Park et al. in [11] study replication with another network hierarchy with no central storage node, where the node distance is expressed as link bandwidth. Their technique might be better in the case when the Internet is used for data transfer.

The mentioned studies assume static hierarchy and do not take into account the dynamic changes of bandwidth and latency resulting from the load of distributed system. In [12] an attempt to cope with this problem has been made. For replica selection they propose a neural net based algorithm predicting the network

[†]Institute of Computer Science, AGH-UST, al. Mickiewicza 30, 30-059, Kraków, Poland (rena,darin,kito@agh.edu.pl)
[‡]Academic Computer Center CYFRONET-AGH, ul. Nawojki 11, 30-950 Kraków, Poland

transfer time of a replica. Another example of research on replica access time prediction based on previous data transfers measurements is [13] in which the Markov chains are used for prediction.

Authors of paper [14] propose 3 heuristic algorithms for selecting the location of new replica based on network latency parameters and number of client requests observed in a certain time interval from the past. Fair-Share replication presented in [15] for choosing new replica location takes into account previous access load of server as well as availability of storage device represented by their storage load. In this way better load balancing among the storage servers is achieved.

Tests of the proposed replication strategies in the studies mentioned by now are conducted by using simulations. Results of real implementation of proposed models and strategies using monitoring of existing storage environment are shown in [16] and in their previous studies. The presented in these papers replication algorithms embody, besides the data access cost imposed by the network, also the cost caused by storage devices capabilities. In the case when a distributed storage system uses high bandwidth network it turns out that the system bottleneck are the storage devices, which bandwidth can be additionally limited according to their actual access load.

The majority of replica selection algorithms assumes that many users access the same data sets. In the case of data storage service holding mainly private user data, users will rather access their own particular files (holding backups or archives). That is why, essential in this case is access load balancing increasing the overall system utilization and thus reducing the access cost. In the proposed solution essential part of the process of existing replica selection and the process of new replica location selection is focused on the evaluating of storage system performance and evaluating of server load. The evaluation is based on the adopted Common Mass Storage System Model (CMSSM) proposed in [17].

Using of general monitoring system like Nagios [18], Ganglia [19] or Gemini [20] for monitoring the performance of HSM systems is troublesome since these systems do not provide the necessary utilities and methods of data gathering, for example, interactive queries to the monitored system or monitoring of storage request queue. Taking into account the heterogeneity of storage systems in term of hardware and software it is important to have an unified access to the monitoring data. Some effort in this area is done by Distributed Management Task Force [21] by specifying the Common Information Model (CIM) [22] for managing computer systems. SMI-S [23] is a CIM based standard which defines interface for managing storage environments. Grid Laboratory Uniform Environment (GLUE) [24] is a conceptual, object-oriented, information model of Grid environments, and its main goal is to provide interoperability among elements of the Grid infrastructure. The above models do not accurately present HSM systems.

The review of related works shows that the problem of data management for performance and load balancing purposes in distributed system with underlying HSM systems as storage horses is not fully addressed which motivated us to start this research.

3. Common Mass Storage System Model. Storage systems used in modern grid-like computing environment are often heterogeneous for historic or economics reasons. Within a virtualized environment with dynamic changing parameters a proper performance monitoring is a hard task. The lack of an unified representation of performance related view of storage systems and especially of HSM storage system was the motivation for developing the Common Mass Storage System Model [3].

The model supports various storage systems with special attention taken to the HSM systems. In the model a set of performance related parameters are defined and grouped in appropriated classes according to type of storage subsystems or parts (e.g. tape drive, library, etc) they are related to.

The model can be used for internal object based representation of storage system for simulation purposes and is used by various monitoring services to give a performance point of view at the given storage system.

The class diagram for the model representing HSM systems is shown in Fig.3.1. The model is able to represent any HSM systems with a certain accuracy, which depends on the vendor's availability of methods for obtaining the performance related parameters specified in the model. We can see that the model can be quite accurate since parameters like queue of waiting requests with detailed information about each request are specified. This allows for very accurate performance prediction for a particular piece of data residing anywhere in the storage hierarchy since it is possible to monitor where the data resides - on disk cache or tertiary storage (which tape, block, etc), it is possible to monitor if there are free resources (for example tape drives) to fulfill the request.

Our vision of a general distributed storage system which makes use of CMSSM for data access prediction is presented in Fig.3.2. The system has well defined functional layers. On the bottom there is the storage layer

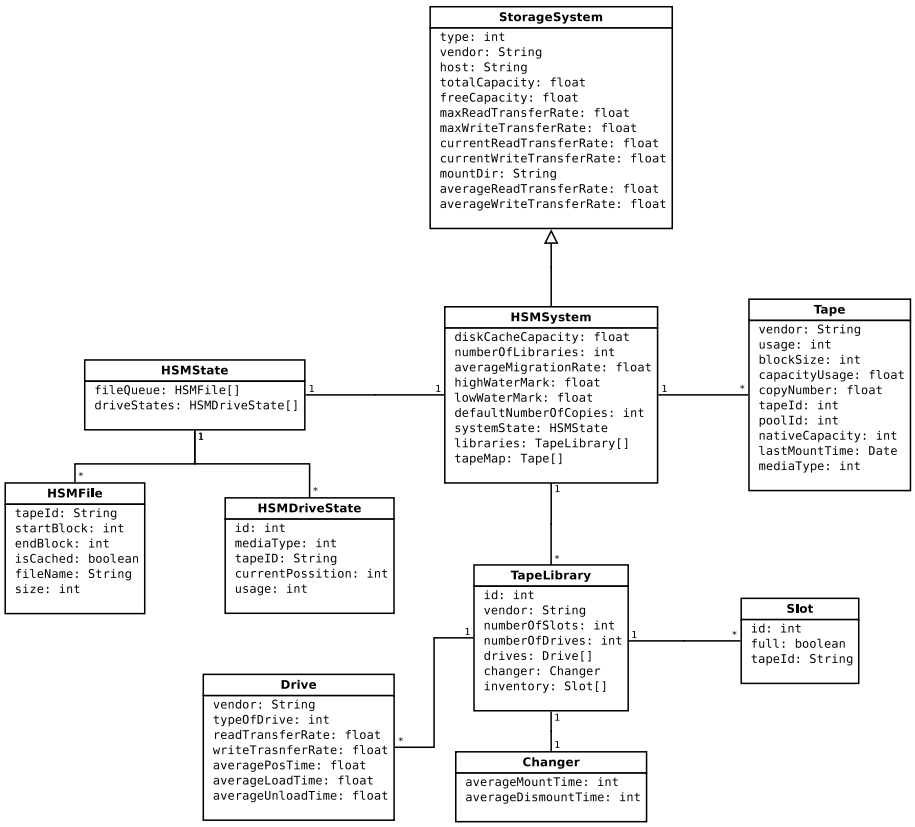


FIG. 3.1. *CMSSM's class diagram for HSM system.*

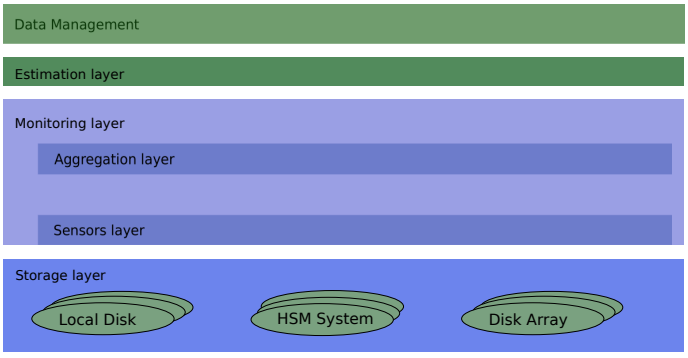


FIG. 3.2. *Layered view of distributed storage system with data access prediction using CMSSM.*

representing complete storage systems (hardware + software) like HSM systems, disk arrays, etc. Sensor layer contains pieces of software which are able to obtain one or more parameters about the given storage system. Generally, sensors are storage system dependent, but in some cases, like for instance a sensor measuring the read/write performance by doing real file I/O, one sensor can fit to more than one types of storage system. Sensors have well defined interface and can be included as plugins to a higher level monitor. Monitoring layer contains of monitors running on the monitored storage node. Monitors have well defined unified interface allowing other services to get selected parameters from them via the network. Requested parameters are sent in data format compatible with CMSSM. Estimation layer contains estimation services, which estimates the performance of storage system or time-to-complete for particular future data transfer. There can be various estimation services using different algorithms and having different estimation accuracy.

Services associated with a certain layer can use services from the lower layers and can bypass layers. For instance management layer service can use directly storage layer tools.

4. Application of CMSSM in NDS. NDS is aimed at building a national storage system providing high quality backup, archiving and data access. The system uses high bandwidth network - Pionier, as a backbone. The system consists of Access Nodes (ANs), which provide the end user interface to system, and storage nodes (SNs) with HSM systems attached. There are also a couple of management nodes. One of the goals of the project is to provide high efficiency based on replication techniques according to user profiles. In the case of data reading the best replica needs to be selected while in the case of data writing the best storage location needs to be chosen. These tasks are completed by the PLBS system briefly described below.

4.1. PLBS. Prediction and Load Balancing System (PLBS) is responsible for load balancing of data access requests among the SNs being part of the NDS. PLBS consists of three subsystems (see Fig.4.1): adopted JMX Infrastructure Monitoring System (JIMS) [25], Advanced Monitoring Database (AMDB) for keeping the values of monitored parameters and Advanced Monitoring and Prediction Daemon (AMPD).

The JIMS based monitoring system consists of Monitoring Agents (JIMS+AMT) installed on every HSM system being part of NDS, and JIMS Gateway collecting data from the agents and storing it to database. JIMS+AMT, JIMS Gateway and AMDB implement the sensors and aggregation layer (see section 3) respectively. The AMPD is responsible for proposing the best replica and location according to the chosen replication policy (see section 5) and implements the estimation layer. One of the requirements for the AMPD is that it must quickly respond, so the user requesting a storage operation does not experience system or data unavailability. Monitoring parameters are measured cyclically by background threads and are stored in the database. In this way the actual parameters (for a certain time interval) can be quickly retrieved from the database and the AMPD can return the results.

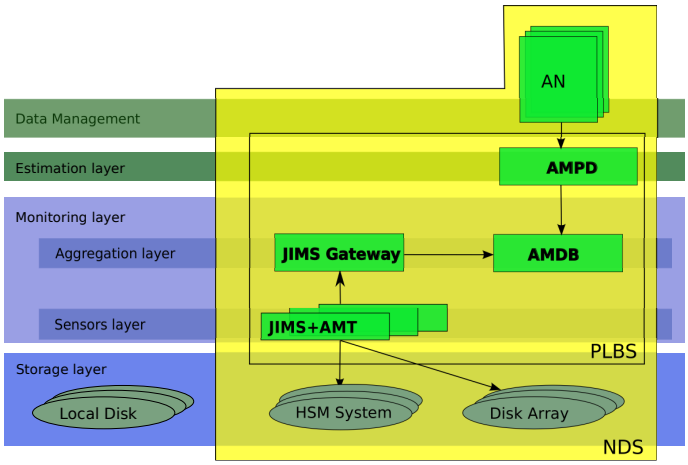


FIG. 4.1. *Application of CMSSM in NDS.*

The HSM monitoring parameters are derived from the CMSSM model described in section 3. The parameters are used to predict the system performance. The parameters are divided into two categories: static parameters changing their values rarely and dynamic parameters changing their values frequently. Part of the model used by the replication policies implemented in PLBS so far is presented below along with the database description.

4.2. The PLBS database–AMDB. The goal of the AMDB is to collect monitored parameters (from the CMSSM model) of distributed nodes in a single place. The approach to store current values of monitored parameters in a database has been chosen, because it allows to completely separate an application logic layer from a monitoring layer.

The AMDB is realized in the standard relational model and conforms to the CMSSM model. The structure of the database is derived from the structure of the monitored systems, which means that the database tables suit to essential HSM components, such as: libraries, drives, pools, tapes, disk cache, etc. The parameters, stored in the database, are divided into two groups: static parameters and dynamic parameters. A simple diagram showing relations between the PLBS database tables is shown in Fig. 4.2. The table columns specification is omitted for simplicity. The dynamic_hsm_parameters_history table stores history of changes of the dynamic

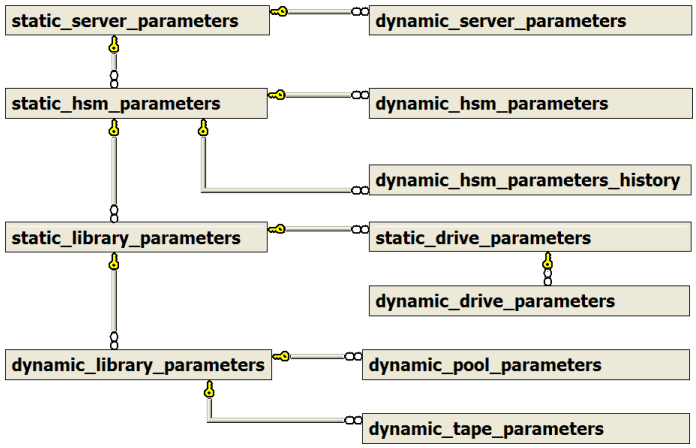


FIG. 4.2. *The AMDB diagram.*

HSM parameters. This table allows the application logic layer to make decisions based not only on the current values of parameters, but also on their statistical values.

Table 4.1 presents summary of the static parameters stored in the AMDB, which are used in the replication policies. Updates of these parameters are performed only on user's demand, for example after a HSM system reconfiguration. Some of these parameters constitutes average values (like average disk cache transfer rate), which are provided by external measurements.

TABLE 4.1
Description of static parameters used in the replication policies.

Parameter name	*Description*	*Implementation*
TotalCapacity	Estimated total capacity of a storage system installed on a single server.	The value of this parameter is estimated as a sum of disk cache capacities
TotalDCCapacity	Total capacity of a single HSM system disk cache.	The value of this parameter is received from the **df** UNIX systems command.
AverageDCReadRate	Estimated value of average disk cache read transfer rate.	The value of this parameter is measured by special benchmarks.
AverageDCWriteRate	Estimated value of average disk cache write transfer rate.	The value of this parameter is measured by special benchmarks.
NumberOfLibraries	Total number of tape libraries connected to a single server.	The value of this parameter is received from configuration files.

Table 4.2 presents summary of the dynamic parameters stored in the AMDB, which are used in the NDS replication policies. These parameters are updated periodically. The update interval is set manually in the PLBS configuration files.

5. Replication policies. The selection of SN for a given data access request is done by heuristic methods taking into account relevant monitoring parameters described in the previous section. Depending on the user profile an appropriate method (called further policy) is used. The AMPD component implements 5 replication policies: round robin—RR, reading in shortest time—R_ST, reading from the minimally loaded device—R_ML, writing replicas of big files—W_BF, writing replicas to the minimally loaded device—W_ML.

TABLE 4.2
Description of dynamic parameters, which are used in the replication policies.

Parameter name	*Description*	*Implementation*
FreeCapacity	Estimated free capacity of a storage system installed on a single server.	The value of this parameter is estimated as a sum of free tapes capacity.
FreeDCCapacity	Free space in a single HSM system disk cache.	The value of this parameter is obtained from the **df** UNIX systems command.
CurrentRate	Transfer rate value from the last measurement.	The value of this parameter is measured by periodically.
HSMLoad	Number of requests waiting or being processed by the HSM system.	The value of this parameter is received from the **dsmq** command for the Tivoli Storage Manager (TSM) systems and from the **fsejob** command for the File System Extender (FSE) systems.

The RR policy is implemented mainly for testing purposes—it does not require monitoring data and it just selects cyclically the next available SN for subsequent requests. The other four policies select the location, for which the value *Loc*, defined in equations 5.1–5.4, is maximized. The R_ST policy is defined by:

$$Loc = \alpha_1 \cdot \frac{RD}{RD_{Max}} + \alpha_2 \cdot \frac{CT}{RD} + \alpha_3 \cdot \frac{1}{1+HL}\,, \tag{5.1}$$

where RD—average disk cache read transfer rate, RD_{Max}—maximal value of average disk cache read transfer rate, taken over all locations, CT—current transfer rate, HL—hsm load, $\sum_{i\in\{1..3\}} \alpha_i = 1$, $\alpha_i > 0$. The exact meaning of these values is given in Tables 4.1 and 4.2.

Equation (5.2) expresses the R_ML policy:

$$Loc = \beta_1 \cdot \frac{ND}{ND_{Max}} + \beta_2 \cdot \frac{1}{1+HL} + \beta_3 \cdot \frac{1}{1+CL}\,, \tag{5.2}$$

where ND—number of drives, ND_{Max}—maximal value of number of drives, taken over all locations, CL—CPU load, $\sum_{i\in\{1..3\}} \beta_i = 1$, $\beta_i > 0$.

Each writing policy determines first whether enough free space is available in a HSM system. Equation 5.3 defines the W_BF policy.

$$Loc = \gamma_i \cdot \frac{FC_{DC}}{TC_{DC}} + \gamma_2 \cdot \frac{FC}{TC} + \gamma_3 \cdot \frac{WR}{WR_{Max}} + \gamma_4 \cdot \frac{1}{1+HL}\,, \tag{5.3}$$

where FC_{DC}—free disk cache capacity, TC_{DC}—total disk cache capacity, FC—free capacity, TC—total capacity, WR—average disk cache write transfer rate, WR_{Max}—maximal value of average disk cache write transfer rate, taken over all locations, $\sum_{i\in\{1..4\}} \gamma_i = 1$, $\gamma_i > 0$.

The policy W_ML is defined by equation 5.4,

$$Loc = \delta_1 \cdot \frac{FC_{DC}}{TC_{DC}} + \delta_2 \cdot \frac{WR}{WR_{Max}} + \delta_3 \cdot \frac{1}{1+HL}\,, \tag{5.4}$$

where $\sum_{i\in\{1..3\}} \delta_i = 1$, $\delta_i > 0$.

α, β, γ and δ are coefficients specifying the impact of the particular monitoring parameters being used in the above formulas. They need to be tuned for the given environment. The above policies are chosen according to the client profile making request and the type of the request. For instance, if the client has defined in the profile that it needs the data as fast as possible than the R_ML policy is chosen.

6. Test results. Four types of tests has been conducted:

- Monitoring influence tests—showing PLBS impact on the performance of the monitored HSM systems,
- Response time tests—showing how fast PLBS responds,
- Load balancing tests—showing data access requests distribution among the storage nodes in multi user and multi requests data access paradigm,
- Throughput tests—showing the total throughput of NDS for selected PLBS replication policy.

The monitoring influence tests are targeted at the JIMS+AMT module while the response time tests are targeted at AMPD module and the both concern the overhead introduced by PLBS to NDS. Load balancing and throughput tests concern the efficiency of data access which can be obtained due to deploying of PLBS to NDS and are provided for a selected policy.

The tests have been conducted using the PLATON [26] infrastructure on which the NDS described above is deployed. The testing environment consists of 5 nodes described in detail in Table 6.1 and connected via Pionier network with 1Gb links. Monitoring agents are installed on every SN while JIMS Gateway, AMPD and AMDB are installed on the kmd2 host (see Sec. 4). The results are presented in the following subsections.

Table 6.1
Test environment nodes

name	location	type	CPU	HSM	drives	HSM cache [TB]
Cyfronet	Krakow	SN	2×Xeon 3.3GHz	IBM TSM	4x LTO	2
PCSS	Poznan	SN	2×Xeon 2.8GHz	IBM TSM	3x LTO	2
WCSS	Wroclaw	SN	2×Xeon 2.8GHz	IBM TSM	2x LTO	1
TASK	Gdansk	SN	2×Xeon 2.8GHz	IBM TSM	3x LTO	0.2
kmd2	Krakow	MN	2×Xeon 2.8GHz	na	na	na

SN—Storage Node, MN—Monitoring Node

6.1. Influence tests. In order for the JIMS to retrieve monitoring data from storage nodes a monitoring agent (JIMS+AMT) (see Fig.4.1) needs to be present on these nodes. The goal of these tests is to measure the influence to performance of storage system when the JIMS+AMT is running on the same node. These tests were performed on the Cyfronet SN (see Table 6.1). This HSM system is in production and the measurements were done during periods of low activity. The main disk storage of the server resides on HP EVA8000 disk array and is attached via 2 FC 2Gb/s links. It is used as disk cache for the HSM system. Repeated patterns of simulated users activities were generated by ftp transfers from other hosts (HSM clients). The JIMS+AMT performed measurements every 10 minutes. Disk reads and writes generated by the measurements had little impact (maximum 5%) on overall execution times of data transfers to and from clients. An example test result is shown in Fig. 6.1

The most influence of JIMS+AMT activity on users data transfers occurs in short periods when the agent measures disk write performance used to calculate AverageDCWriteRate (see Table 4.1). The system utilization statistics come from `sar` program. The user data rates were taken from network traffic statistics as there was no other network traffic on the server during the tests.

6.2. Response tests. Response tests measure the time of processing prediction requests to AMPD. Table 6.2 presents test results for the implemented replication policies. Each value is taken as an average over 5000 requests. We distinguished two cases: (1) the client is on the same machine that AMPD, (2) the client is located remotely to the AMPD component.

We can see that the response times are acceptable for all policies and they do not exceed 102 ms for remote clients and 65 ms for local ones. The network overhead has great influence on the final response times - without it the processing time is shorter by about 37 ms.

6.3. Load balancing test. Load balancing test shows how the requests get distributed among the storage nodes. One monitoring node and four storage nodes have taken part in this test (see Table 6.1). The testing procedure is as follows: First, a set of 100 files has been written to the storage nodes in such way that all files are replicated to all four storage nodes. The file size is 1GiB. Next, a script requesting storage nodes performance prediction and reading data from the appropriate replica is run. The script starts new requests

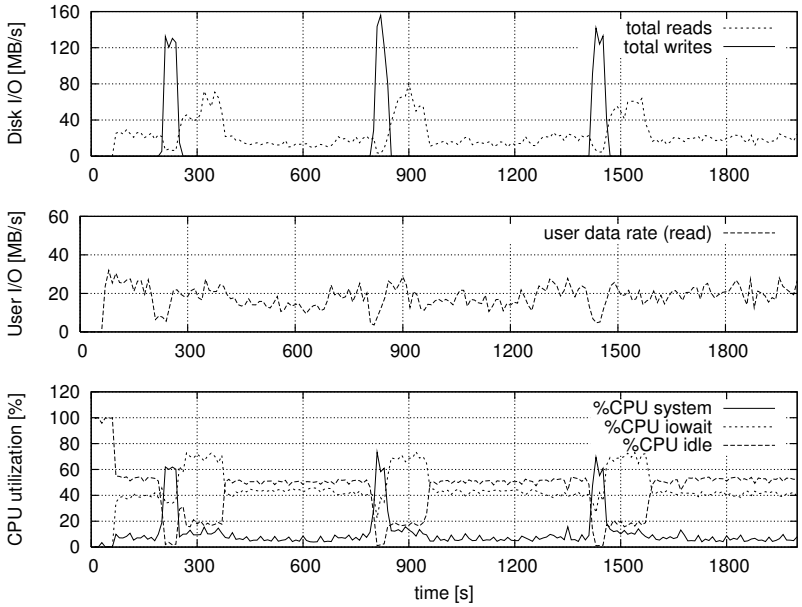


FIG. 6.1. *An example result of monitoring influence test.*

TABLE 6.2
Time of serving prediction requests

Replication policy	Response time [ms]	
	local client	remote client
Reading in shortest time	64.5	101.6
Reading from the minimally loaded device	18.6	55.1
Writing big files	14.7	51.3
Writing to the minimally loaded device	13.3	50.0
Round Robin	11.8	48.8

until 8 concurrent transfers get present. When a transfer is over another request is started. For each test run 2000 requests have been done. The number of requests has been chosen big enough to allow at least few updates of monitoring performance data (used for the prediction) to occur.

It should be noted that result of performance prediction is a sorted list of storage nodes. The nodes are sorted according to the value obtained for the given replication policy. A normalized value (between 0–100) is assigned to each node indicating its relative storage performance. In order to prevent overloading of one storage node (by always sending the request to the best node) the script selects a storage node with a certain probability which is proportional to its current storage performance (obtained from the monitoring and prediction).

Figure 6.2 provides results of prediction tests for the R_ST policy with the following coefficient values: $\alpha_1 = 0.6$, $\alpha_2 = 0.2$, $\alpha_3 = 0.1$. Each point represents the fraction of requests for which a particular host has been selected within the given range of request numbers. The range has been set to 100 requests. At a given moment additional storage load has been issued to the best storage node (PCSS) to study the adaptability of the system in changing environment (mark A). We can see that shortly after that the requests distribution gets reorganized and the TASK storage node gets serving more requests.

We can see that the requests are distributed between the nodes according to their storage processing power—the most powerful storage nodes (*PCSS* and *TASK*) have served the majority of requests.

6.4. Throughput tests. The throughput tests are intended to compare the overall storage throughput of the system for the R_ST replication policy and the RR policy. The testing procedure is the same as the one for the load balancing tests. The tests have been conducted for idle and loaded storage devices. The results are presented in Table 6.3. We can see that the R_ST policy is much better than the RR policy especially

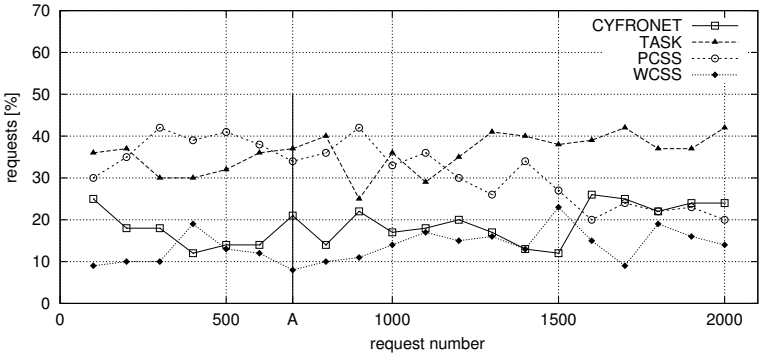


FIG. 6.2. *Replication tests for R_ST policy*

if additional load is put on the system. We can also observe that for the R_ST policy the throughput is not influenced by the additional storage load. It is because the load gets distributed proportionally to the other nodes which are still having unused storage performance.

TABLE 6.3
Storage throughput

Replication policy	Storage throughput [MiB/s]	
	idle	loaded
R_ST	325	327
RR	233	101

7. Summary and future work. In this paper the application of CMSSM in the national distributed storage system, NDS, has been described. The PLBS subsystem being a part of the NDS system and providing advanced monitoring and prediction functionalities has been presented. The system makes use of replication techniques to increase availability and performance of data access. Monitoring parameters, methods for retrieving them and replication policies have been described. The influence tests showed that the monitoring did not cause essential storage system performance degradation. The system response times are within the tens of milliseconds range which is satisfying. Load balancing test shows that requests get distributed between the nodes proportionally according to their storage processing power. Our future plans focus on using CMSSM and its ontological representation in knowledge supported distributed storage system with quality of service.

Acknowledgments. This research is partially supported by the MNiSW grant nr N N516 405535 and PLATON project POIG.02.03.00-00-028/08. AGH-UST grant nr 11.11.120.865 is also acknowledged. Thanks go to: prof. K. Zieliński for giving access to the JIMS software, M. Brzeźniak for support with PLATON infrastructure, M. Jarząb for support with JIMS, PLATON partners for sharing storage resources.

REFERENCES

[1] National Data Storage project, Polish MNiSW grant nr R02170170170172055 03, `https://kmd.pcss.pl`
[2] Pionier—Polish Optical Internet, `http://www.pionier.gov.pl`
[3] D. Nikolow, R. Slota, J. Kitowski, "Knowledge Supported Data Access in Distributed Environment", Proc. of CGW'08, October 13-15 2008, ACC-Cyfronet AGH, 2009, Krakow, pp. 320-325.
[4] J.H. Han, D.H. Lee, H. Kim, H. P. In, H.S. Chae, Y.I. Eom "A situation-aware cross-platform architecture for ubiquitous game", Computing and Informatics, vol. 28, nr 5, 2009, pp. 619-633
[5] Funika, W. - Korcyl, K. - Pieczykolan, J. - Skital, L. - Balos, K. - Slota, R. - Guzy, K. - Dutka, L. - Kitowski, J. - Zieli1717ski, K. "Adapting a HEP Application for Running on the Grid", Computing and Informatics, vol. 28, nr 3, 2009, pp. 353-367
[6] Słota, R., Nikolow, D., Kuta, M., Kapanowski, M., Skałkowski, K., Pogoda, M., Kitowski, J., "Replica Management for National Data Storage", Proceedings PPAM09, LNCS 6068, Springer, in print.
[7] Chai, E., Matsumoto, K., Uehara, M., Mori, H., "Virtual Large-scale Disk Base on PC GRID", Scalable Computing: Practice and Experience, vol. 10, nr 1, SWPS, 2009, pp.87-98.
[8] Srinivas, A., Janakiram, D., "Data Management in Distributed Systems: A Scalability Taxonomy", Scalable Computing: Practice and Experience, vol. 8, nr 1, SWPS, 2007, pp.115-130.

[9] Ranganathan K., Foster I.: "Identifying Dynamic Replication Strategies for a High-Performance Data Grid". in: Proc. Int. Workshop on Grid Computing, Denver, Nov. 2001.
[10] Lamehamedi H., Szymański B., Deelman E., "Data Replication Strategies in Grid Environments", in: IEEE Computer Science Press, Los Alamitos, CA, 2002, pp. 378-383.
[11] Park S., Kim J., Ko Y., Yoon W., "Dynamic Data Grid Replication Strategy Based on Internet Hierarchy", LNCS 3033, Springer, 2004, pp.838-846.
[12] Rahman R.M., Barker K., Alhajj R., "A Predictive Technique for Replica Selection in Grid Environment", in: 7-th IEEE Int. Symp. on Cluster Computing and the Grid, IEEE Computer Society, 2007.
[13] Li, J., "A Replica Selection Approach based on Prediction in Data Grid", in: Proc. Third Int. Conf. on Semantics, Knowledge and Grid - SKG2007, 29-31 Oct. 2007, Xi'an, Shan Xi, China, pp. 274-277.
[14] Rahman R.M., Barker K., Alhajj R., "Replica placement Strategies in Data Grid", in: J Grid Computing (2008) 6:103-123, Springer Science + Business media B.V. 2007.
[15] Rasool, Q., Li, J., Oreku, G.S., Zhang, S., Yang, D., "A load balancing replica placement strategy in Data Grid", Third IEEE Int. Conf. on Digital Information Management (ICDIM), Nov. 13-16, 2008, London, UK, Proc. IEEE 2008, pp. 751-756.
[16] Słota, R., Skitał, L., Nikolow D., Kitowski J., "Algorithms for Automatic Data Replication in Grid Environment", in: LNCS, 3911, Springer, 2006, pp. 707-714.
[17] D. Nikolow, R. Słota, and J. Kitowski, "Grid Services for HSM Systems Monitoring", LNCS 4967, Springer, 2008, pp.321-330.
[18] Nagios, `http://www.nagios.org/`
[19] Ganglia monitoring system, `http://ganglia.sourceforge.net`
[20] B. Baliś, M. Bubak and B. Labno, GEMINI: Generic Monitoring Infrastructure for Grid Resources and Applications, Proc. of Cracow'06 Grid Workshop, Oct 15-18, 2006, Cracow, Poland, ACC Cyfronet AGH, 2007, pp. 60-73.
[21] Distributed Management Task Force, `http://www.dmtf.org`
[22] Common Information Model, `http://www.dmtf.org/standards/cim/`.
[23] Storage Management Initiative Specification (SMI-S), `http://www.snia.org/tech_activities/standards/curr_standards/smi`.
[24] Grid Laboratory Uniform Environment (GLUE), `http://forge.gridforum.org/sf/projects/glue-wg`.
[25] Zieliński, K., Jarząb, M., Balos, K., Wieczorek, D., "Open Interface for Autonomic Management of Virtualized Resources in Complex Systems—Construction Methodology", in: FGCS, vol. 24, Issue 5, May 2008, pp. 390-401.
[26] PLATON—Service Platform for e-Science, `http://www.platon.pionier.net.pl/`.

Edited by: Norbert Meyer
Received: March 30, 2010
Accepted: June 18, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 131–136. http://www.scpe.org
ISSN 1895-1767

DISTRIBUTED DATA INTEGRATION AND MINING USING ADMIRE TECHNOLOGY*

ONDREJ HABALA†, MARTIN SELENG†, VIET TRAN†, LADISLAV HLUCHY†,
MARTIN KREMLER‡, AND MARTIN GERA‡

Abstract. In this paper we present our work on the engine for integration of environmental data. We present a suite of selected environmental scenarios, which are integrated into a novel data mining and integration environment, being developed in the project ADMIRE . The scenarios have been chosen for their suitability for data mining by environmental experts. They deal with meteorological and hydrological problems, and apply the chosen solutions to pilot areas within Slovakia. The main challenge is that the environmental data required by scenarios are maintained and provided by different organizations and are often in different formats. We present our approach to the specification and execution of data integration tasks, which deals with the distributed nature and heterogeneity of required data resources.

Key words: environmental applications, distributed data management, data integration, OGSA DAI

1. Introduction. We present our work in the project ADMIRE, where we use advanced data mining and data integration technologies to run an environmental application, which uses data mining instead of standard physical modeling to perform experiments and obtain environmental predictions. The paper starts with description of the project ADMIRE, its vision and goals. Then we describe the history and current status of the environmental application. The core of the paper then presents our approach to the integration of data from distributed resources. We have developed a prototype of data integration engine that allows users to specify data integration process in form of a workflow of reusable processing elements. This paper has been originally presented in [10].

1.1. The EU ICT Project ADMIRE. The project ADMIRE (Advanced Data Mining and Integration Research for Europe [1]) is a 7th FP EU ICT project aims to deliver a consistent and easy-to-use technology for extracting information and knowledge from distributed data sources. The project is motivated by the difficulty of extracting meaningful information by mining combinations of data from multiple heterogeneous and distributed resources. It will also provide an abstract view of data mining and integration, which will give users and developers the power to cope with complexity and heterogeneity of services, data and processes. One of main goals of the project is to develop a language that serves as a canonical representation of the data integration and mining processes.

1.2. Flood Forecasting Simulation Cascade. The Flood Forecasting Simulation Cascade is a SOA-based environmental application, developed within several past FP5 and FP6 projects [2], [3], [4]. The application's development started in 1999 in the 5th FP project ANFAS [5]. In ANFAS, it was mainly one hydraulic model (the FESWMS [6]). It then continued with a more complex scenario in 5th FP project CrossGrid, turned SOA in 6th FP projects K-Wf Grid and MEDIgRID, and finally extended the domain to environmental risk management in ADMIRE. The application is now comprised of a set of environmental scenarios, with the necessary data and code to deploy and execute them. The scenarios have been chosen and prepared in cooperation with leading hydro-meteorological experts in Slovakia, working mainly for the Slovak Hydrometeorological Institute (SHMI), Slovak Water Enterprise (SWE), and the Institute of Hydrology of the Slovak Academy of Sciences (IH SAS). We have gathered also other scenarios from other sources, but in the end decided to use the ones presented below, because they promise to be the source of new information for both the environmental domain community, as well as for the data mining community in ADMIRE. Together with the scenarios, we have gathered a substantial amount of historical data. SWE has provided 10 years of historical data containing the discharge, water temperature, and other parameters of the Vah Cascade of waterworks (15 waterworks installations and reservoirs in the west of Slovakia). SHMI has provided 9 years of basic meteorological data (precipitation, temperature, wind) computed by a meteorological model and stored in a set of GRIB (Gridded Binary) files, hydrological data for one of the scenarios, and also partial historical record from their nation-wide

*This work is supported by projects ADMIRE FP7-215024, APVV DO7RP-0006-08, DMM VMSP-P-0048-09, SMART ITMS: 26240120005, SMART II ITMS: 26240120029, VEGA No. 2/0211/09.
†Institute of Informatics of the Slovak Academy of Sciences, Dubravska cesta 9, 84507 Bratislava, Slovakia ({Ondrej.Habala, Martin.Selen, Viet.Tran, Ladislav.Hluchý}@savba.sk)
‡Comenius University, Faculty of Mathematics Physics and Informatics, Mlynska dolina, 84248 Bratislava, Slovakia

TABLE 2.1
Depiction of the predictors and variables of the ORAVA scenario

Time	Rainfall	$Temp_{Air}$	Discharge	$Temp_{Reservoir}$	$Height_{St}$	$Temp_{St}$
T-2	R_{T-2}	F_{T-2}	D_{T-2}	E_{T-2}	X_{T-2}	Y_{T-2}
T-1	R_{T-1}	F_{T-1}	D_{T-1}	E_{T-1}	X_{T-1}	Y_{T-1}
T	R_T	F_T	D_T	E_T	X_T	Y_T
T+1	R_{T+1}	F_{T+1}	D_{T+1}	E_{T+1}	X_{T+1}	Y_{T+1}
T+2	R_{T+2}	F_{T+2}	D_{T+2}	E_{T+2}	X_{T+2}	Y_{T+2}

network of meteorological data. They have also provided several years of stored weather radar data, necessary for one of the scenarios. The programs used by the application are in the context of ADMIRE described in Data Mining and Integration Lanuage (DMIL) [7]. The processes described in DMIL perform data extraction, transformation, integration, cleaning and checking. Additionally, in some scenarios we try to predict future values of some hydro-meteorological variables; if necessary, we use a standard meteorological model to predict weather data for these cases.

2. Environmental Scenarios of ADMIRE. In this chapter we present the suite of environmental scenarios, which we use to test the data mining and integration capabilities of the ADMIRE system. The scenarios are part of the Flood Forecasting and Simulation Cascade application, which has been in the meantime expanded beyond the borders of flood prediction into a broader environmental domain. There are four scenarios, which are in the process of being implemented and deployed in the ADMIRE testbed. These scenarios have been selected from more than a dozen of candidates provided by hydro-meteorological, water management, and pedological experts in Slovakia. The main criterion for their selection was their suitability for data mining application. The scenarios are named ORAVA, RADAR, SVP, and O3, and they are in different stages of completion, with ORAVA being the most mature one, and O3 only in the beginning stages of its design.

2.1. ORAVA. The scenario named ORAVA has been defined by the Hydrological Service division of the Slovak Hydrometeorological Institute, Bratislava, Slovakia. Its goal is to predict the water discharge wave and temperature propagation below the Orava reservoir, one of the largest water reservoirs in Slovakia.

The pilot area covered by the scenario (see Figure 2.1) lies in the north of Slovakia, and covers a relatively small area, well suitable for the properties of testing ADMIRE technology in a scientifically interesting, but not too difficult setting.

The data, which has been selected for data mining, and which we expect to influence the scenario's target variables—the discharge wave propagation, and temperature propagation in the outflow from the reservoir to river Orava—is depicted in Table 2.1.

For predictors in this scenario, we have selected rainfall and air temperature, the discharge volume of the Orava reservoir and the temperature of water in the Orava reservoir. Our target variables are the water height and water temperature measured at a hydrological station below the reservoir. As can be seen in Figure 2.1, the station directly below the reservoir is no.5830, followed by 5848 and 5880. If we run the data mining process in time T, we can expect to have at hand all data from sensors up to this time (first three data lines in Table 1). Future rainfall and temperature can be obtained by running a standard meteorological model. Future discharge of the reservoir is given in the manipulation schedule of the reservoir. The actual data mining targets are the X and Y variables for times after time T (T being current time).

2.2. RADAR. This experimental scenario tries to predict the movement of moisture in the air from a series of radar images (see for example). Weather radar measures the reflective properties of air, which are transformed to potential precipitation before being used for data mining. An example of already processed radar sample (with the reflection already re-computed to millimeters of rainfall accumulated in an hour) can be seen in Figure 2.2.

The scenario once again uses both historical precipitation data (measured by sensors maintained by SHMI) and weather predictions computed by a meteorological model. Additionally to these, SHMI has provided several years' worth of weather radar data (already transformed to potential precipitation).

2.3. SVP. This scenario, which is still in the design phase, is the most complex of all scenarios expected to be deployed in the context of ADMIRE. It uses the statistical approach to do what the FFSC application

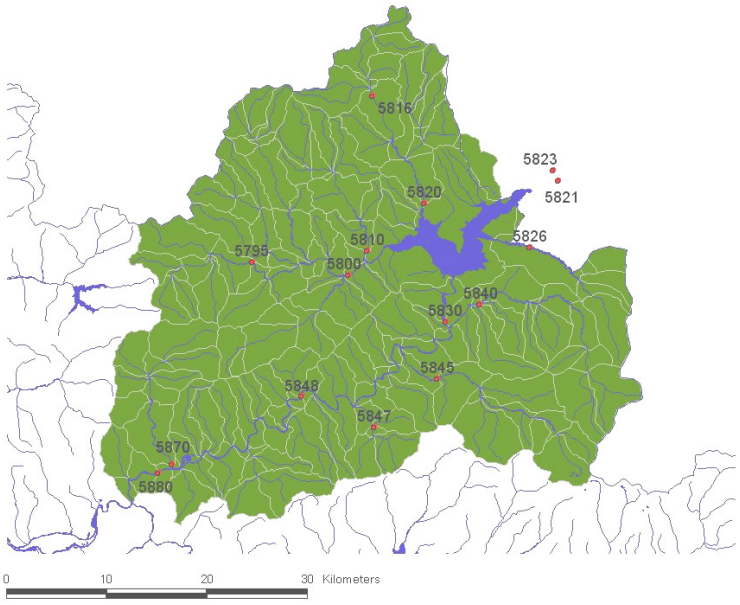


FIG. 2.1. *The geographical area of the pilot scenario ORAVA*

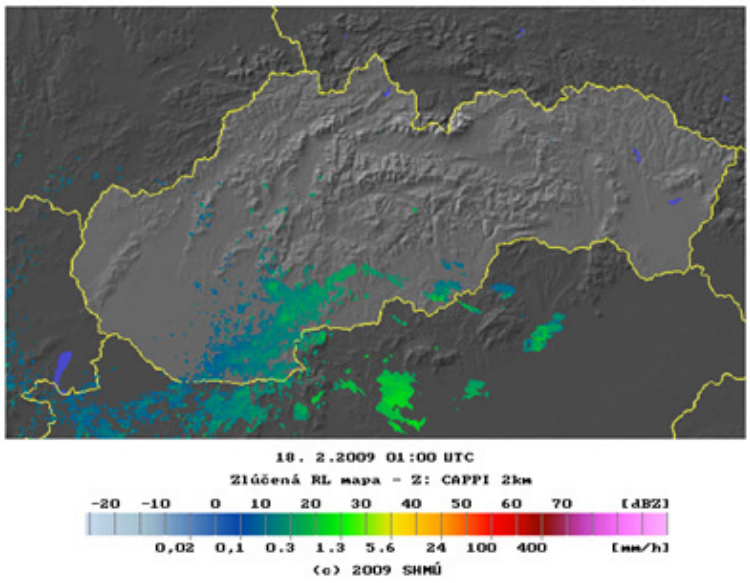


FIG. 2.2. *An example of weather radar image with potential precipitation*

did before ADMIRE—predict floods. The reasons why we decided to perform this experiments are mainly the complexity of simulation of floods by physical models when taking into account more of the relevant variables, and the graceful degradation of results of the data mining approach when facing incomplete data—in contrast to the physical modeling approach, which usually cannot be even tried without having all the necessary data.

For predicting floods, we have been equipped with 10 years of historical data from the Vah cascade of waterworks by the Slovak Water Enterprise, 9 years of meteorological data (precipitation, temperature, wind) computed by the ALADIN model at SHMI, hydrological data from the river Vah, again by SHMI, and additionally with measured soil capacity for water retention, courtesy of our partner Institute of Hydrology of the Slovak Academy of Sciences. We base our efforts on the theory, that the amount of precipitation, which actually

reaches the river basin and contributes to the water level of the river is influenced by actual precipitation and its short-term history, water retention capacity of the soil, and to lesser extent by the evapotranspiration effect.

3. Data integration engine for environmental data. In this section, we discuss the data integration engine designed for the environmental data integration and mining. It is motivated by the scenarios described in previous section. We first describe requirements that we took into account and then we present our approach to environmental data integration. In the discussion, we give examples mainly from Orava scenario; the first scenario implemented using our data integration engine.

In Orava river management scenario, the data from three different sources are used. The data are owned and maintained by different organizations. To allow the data mining operations proposed for this scenario, the data from those different sources must be integrated first. Furthermore, the data are kept in different formats. In the case of Orava scenario, two data sets are stored in relational database (waterworks data, water stations measurements) and one is kept in binary files (precipitation data are stored in GRIB files—binary file format for meteorological data). From technical point of view, we must be able to work with the heterogeneous data stored in distributed, autonomous resources. In our work, we have considered so far the data in the form of lists of tuples.

In the following, we use the term data resource to denote a service providing access to data, with a single point of interaction. We use the term processing resource to denote a service capable of performing operations on the input lists of tuples. Data resource can have capabilities of a processing resource.

Atomic units used for data access and transformations are called processing elements (PE). Following types of processing elements are needed:

- Data retrieval PEs—operations able to retrieve the data from different, heterogeneous data sources. Data retrieval PEs are executed at data resources. This class of PEs is also responsible for transforming raw data sets to the form of tuples.
- Data transfer PE—able to transfer list of tuples between distinct processing resources.
- Data transformation PE—operations that transform input list of tuples. These PEs can perform data transformation on per tuple basis, or can be used to aggregate tuples in the input lists.
- Data integration PEs—given input lists of tuples, data integration operations combine the tuples from input lists into a coherent form.

An operation has one or more inputs and one or more outputs. Inputs can be either literals or list of tuples and a outputs are list of tuples. Operations can be chained to form a data integration workflow—an oriented graph, where nodes are operations and edges are connection of inputs and outputs of the operations.

The term Application Processing Element (APE) will denote a data integration workflow that can be executed at a single resource. APE is a composition of atomic operations that provides functionality required by a data integration task. For example, in Orava scenario we use the precipitation data from GRIB files. The GRIB reader processing element extracts the data from GRIB files; it has two inputs—the first is a list of GRIB files and the second is a list of indexes in GRIB value arrays. The GRIB reader activity outputs all the values at input indexes from all the input files. We use an operation that queries the GRIB metadata database to determinate GRIB files of interest and another operation that transform given geo-coordinates in WGS84 to the indexes consumed by GRIB reader activity. This small workflow of three operations forms a single APE that provides precipitation data for given time period and geo-coordinates. The idea behind APE is to provide data integration blocks that can be executed at a single processing or data resource and can be reused for in multiple data integration tasks. Similarly to atomic PE, the inputs of APE can be literals or list of tuples and outputs are list of tuples.

The goal of our proposed data integration engine is to provide means of executing data integration tasks that are composed of multiple APEs and can integrate the data from distributed, autonomous and possibly heterogeneous data resources. Our data integration engine is designed to run the data integration tasks, given the input parameters and the APE workflow specification.

APE workflow specification is composed of four components: definition of APEs instances, mapping between inputs and outputs of connected APEs, mapping between the definition of integration task parameters and the parameter inputs of APEs in workflow and the definition of the result output.

In alignment with ADMIRE project vision, the APEs are specified in Data Mining and Integration Language (DMIL) [7] that is being developed within the project. The goal of DMIL is to be a canonical representation of data integration process, described in an implementation independent manner. The APE instance is specified by

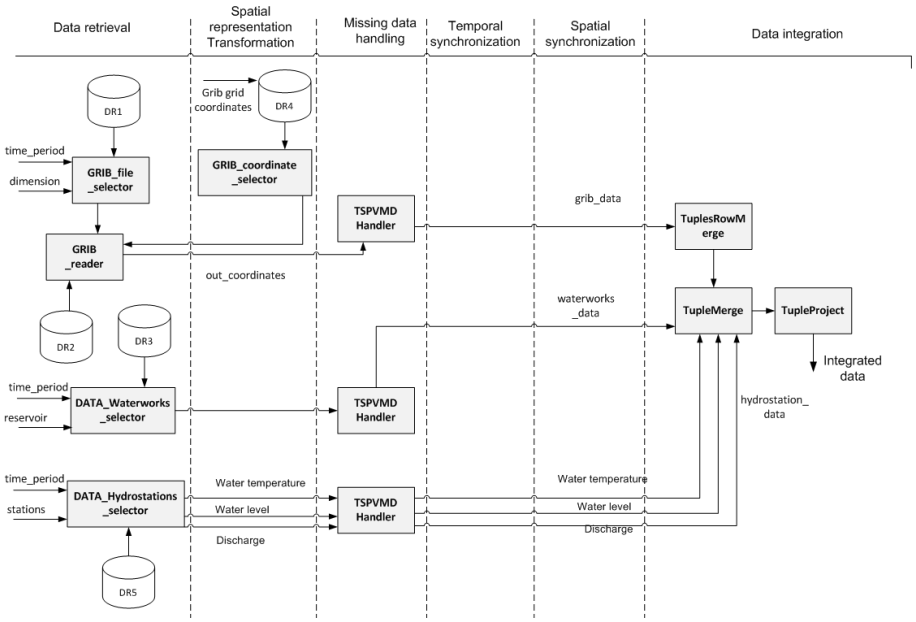


FIG. 3.1. *Orava river management scenario - APEs workflow*

the DMIL description of the process that should be executed, the specification of the data/processing resource it should be executed at and APE instance identifier that is unique within the APE workflow specification. Figure 3.1 depicts the APEs workflow of the Orava river management scenario.

In our view, the main advantage of proposed data integration engine is that user can specify sub-workflows that are executed on a separate data resources and the engine automatically connects the results of APEs executed on distributed resources. This helps to deal with the complexity of the distributed data integration.

3.1. Implementation. The prototype of proposed data integration engine for environmental data (DIEED) is implemented in JAVA programming language. It uses OGSA-DAI ([8], [9]) framework as the platform for exposing data resources in the distributed testbed and for executing the partial workflows of processing elements; it also provide us with the data transfer capabilities and streaming of the list of tuples between remote nodes. The data integration engine takes as inputs the integration task parameters and APE workflow specification. From the APE workflow specification, the engine constructs an oriented graph of APEs (defined by the mapping between inputs and outputs of APEs). For each node of the graph (containing an APE specified in DMIL) the DIEED performs following actions:

1. Compiles DMIL code—the DMIL specification of the node process is compiled to JAVA class that constructs an OGSA-DAI workflow.
2. JAVA class containing OGSA-DAI workflow is compiled by JAVA compiler, it is instantiated and OGSA-DAI workflow object is created
3. workflow object is submitted to OGSA-DAI service for execution
4. workflow execution on remote server is monitored

The whole APEs workflow is monitored during execution (providing information on the state of each of APEs); after execution is finished, the results can be retrieved in form of WebRowSet object.

DIE was integrated with the toolkit being developed in the project; this allows the user to submit APEs workflows, visualize the specified workflow and monitor its execution via graphical user interface based on Eclipse platform. Figure 3.2 depicts the graphical user interface for DIEED.

4. Conclusion. In this paper, we have presented preliminary results of our ongoing work on the data integration engine for environmental data that is being developed in the scope of ADMIRE project. We have first described four scenarios dealing with the integration and mining of environmental data. The main challenge is that the environmental data required by scenarios are maintained and provided by different organizations and are often in different formats. Our work concentrated on providing a platform that would allow integration

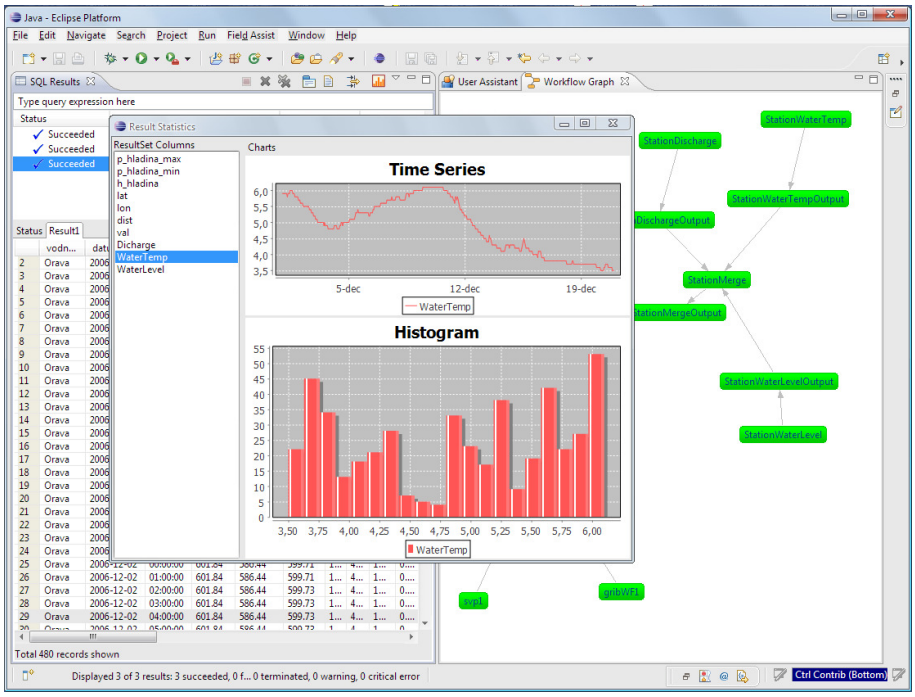


FIG. 3.2. *GUI showing results of DIEED—APEs workflow and its results*

of data from distributed, heterogeneous resources. Our results allow users to construct reusable application processing elements specified in DMIL [7] (language for data mining and integration, which is being designed within the project) and the engine executes them transparently on distributed data resources.

REFERENCES

[1] ADMIRE. EU FP7 ICT project: Advanced Data Mining and Integration Research for Europe (ADMIRE), 2008-2011. Grant agreement no. 215024.
[2] CROSSGRID. EU FP5 IST RTD project: Development of Grid Environment for Interactive Applications (2002-05) IST-2001-32243.
[3] K-Wf Grid. EU FP6 RTD IST project: Knowledge-based Workflow System for Grid Applications (2004-2007) FP6-511385, call IST-2002-2.3.2.8.
[4] MEDIGRID. EU FP6 RTD Sust.Dev. project: Mediterranean Grid of Multi-Risk Data and Models (2004-2006) GOCE-CT-2003-004044, call FP6-2003-Global-2.
[5] ANFAS. EU FP5 IST RTD project: datA fusioN for Flood Analysis and decision Support (2000-03) IST-1999-11676.
[6] Finite Element Surface Water Modeling System (FESWMS).
[7] Atkinson M. et al.: ADMIRE White Paper: Motivation, Strategy, Overview and Impact, v0.9. 2009.
[8] Antonioletti M., Atkinson M. P., Baxter R., Borley A., Chue Hong N. P., Collins B., Hardman N., Hume A., Knox A., Jackson M., Krause A., Laws S., Magowan J., Paton N. W., Pearson D., Sugden T., Watson P., Westhead M.: The Design and Implementation of Grid Database Services in OGSA-DAI. Concurrency and Computation: Practice and Experience, Volume 17, Issue 2-4, Pages 357-376, Feb. 2005.
[9] Karasavvas K., Antonioletti M., Atkinson M. P., Chue Hong N. P., Sugden T., Hume A. C., Jackson M., Krause A., Palansuriya C.: Introduction to OGSA-DAI Services. Lecture Notes in Computer Science, Volume 3458, Pages 1-12, May 2005, ISBN 978-3-540-25810-0.
[10] Ciglan, M., Habala, O., Tran, V., Hluchy, L., Kremler, M., Gera, M.: Application of ADMIRE Data Mining and Integration Technologies in Environmental Scenarios. To appear in Proc. of the 4th Grid Applications and Middleware Workshop (GAMW2009), Wroclaw, Poland, September 13-16, 2009, LNCS 6067-6068, Springer.

Edited by: Ewa Deelman
Received: March 30, 2010
Accepted: May 25, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 137–147. http://www.scpe.org
ISSN 1895-1767

ULTRA-FAST CARRIER TRANSPORT SIMULATION ON THE GRID. QUASI-RANDOM APPROACH.*

EMANOUIL ATANASSOV†, TODOR GUROV†, AND ANETA KARAIVANOVA†

Abstract. The problem for simulation ultra-fast carrier transport in nano-electronics devices is a large scale computational problem and requires HPC and/or Grid computing resources. The most widely used techniques for modeling this carrier transport are Monte Carlo methods.

In this work we consider a set of stochastic algorithms for solving quantum kinetic equations describing quantum effects during the femtosecond relaxation process due to electron-phonon interaction in one-band semiconductors or quantum wires. The algorithms are integrated in a Grid-enabled package named **S**tochas-tic **A**Lgorithms for **U**ltra-fast **T**ransport in s**E**miconductors (SALUTE).

There are two main reasons for running this package on the computational Grid: (i) quantum problems are very computationally intensive; (ii) the inherently parallel nature of Monte Carlo applications makes efficient use of Grid resources. Grid (distributed) Monte Carlo applications require that the underlying random number streams in each subtask are independent in a statistical sense. The efficient application of quasi-Monte Carlo algorithms entails additional difficulties due to the possibility of job failures and the inhomogeneous nature of the Grid resource. In this paper we study the quasi-random approach in SALUTE and the performance of the corresponding algorithms on the grid, using the scrambled Halton, Sobol and Niederreiter sequences. A large number of tests have been performed on the EGEE and SEEGRID grid infrastructures using specially developed grid implementation scheme. Novel results for energy and density distribution, obtained in the inhomogeneous case with applied electric field are presented.

Key words: ultra-fast carrier transport, Monte Carlo methods, quasi-Monte Carlo, scrambled Halton, Sobol and Niederreiter sequences, grid computing

1. Introduction. The Monte Carlo Methods for quantum transport in semiconductors and semiconductor devices have been actively developed during the last two decades [3, 10, 16, 20, 24]. These Monte Carlo calculations need large amount of computational power and the reason is as follows: If temporal or spatial scales become short, the evolution of the semiconductor carriers cannot be described in terms of the Boltzmann transport and therefore a quantum description is needed. Let us note that in contrast to the semiclassical transport when the kernel is positive, the kernel in quantum transport can have negative values. The arising problem, sometimes referred to as the "negative sign problem," leads to additional computational efforts for obtaining the desired solution. That is why the quantum problems are very computationally intensive and require parallel and Grid implementations.

Quasi-Monte Carlo (QMC) simulation, using deterministic sequences that are more uniform than random ones, holds out the promise of much greater accuracy, close to $O(N^{-1})$ [23] in optimal cases. While these sequences (called low discrepancy sequences or quasirandom sequences) do improve the convergence of applications like numerical integration, it is difficult to apply them for Markov chain based problems due to correlations. These difficulties can be overcame by reordering, hybrid algorithms or randomization techniques. Successful examples can be found in [13, 14, 15, 17, 19] and many other papers.

In this paper we present quasirandom approach for ultrafast carrier transport simulation. Due to correlation, the direct quasirandom variants of Monte Carlo methods do not give adequate results. Instead of them, we propose hybrid algorithms with pseudorandom numbers and scrambled quasirandom sequences. We use scrambled modified Halton [2], scrambled Sobol [1] and scrambled Niederreiter (the scrambling algorithm is similar to described in [1]). In this paper we present also the developed grid implementation scheme which uses not only the computational capacity of the grid but also the available grid services in a very efficient way. With this scheme we were able to obtain new estimates about important physical quantities.

This paper is an extended version of [5], presented at the the international conference *Parallel Processing and Applied Mathematics* **(PPAM09)**, held in September 2009 in Poland. Here we have added more details about the quasirandom approach for problems of considered kind (including our algorithm for generating Sobol and Niederreiter sequences). We also have updated the Grid implementation description; the numerical tests are obtained using the most recent Grid middleware and services.

*Supported by the Ministry of Education and Science of Bulgaria under Grant No. DO02-146/2008.

†IPP—Bulgarian Academy of Sciences, Acad. G. Bonchev St., Bl.25A, 1113 Sofia, **Bulgaria**, E-mails:{emanouil, gurov, anet}@parallel.bas.bg

The paper is organized as follows: Section 2 describes very briefly the problem and the Monte Carlo algorithms, section 3 presents the quasirandom sequences and hybrid algorithms, section 4 describes the grid implementation scheme, section 4 contains the new estimates and performance analysis.

2. Background (Brief description of SALUTE). The first version of SALUTE was designed at IPP-BAS in 2005 as a set of Monte Carlo algorithms for simulation of ultra-fast carrier transport in semiconductors together with simple Grid implementation, [3, 4]. Later on, we extended the the area of application (quantum wires), the algorithms (more complicated equations to be solved) and the implementation scheme. In this paper we present the quasirandom approach in SALUTE and discuss the new results. The physical model describes a femtosecond relaxation process of optically excited electrons which interact with phonons in one-band semiconductor, [21]. The interaction with phonons is switched on after a laser pulse creates an initial electron distribution. Experimentally, such processes can be investigated by using ultra-fast spectroscopy, where the relaxation of electrons is explored during the first hundred femtoseconds after the optical excitation. In our model we consider a low-density regime, where the interaction with phonons dominates the carrier-carrier interaction. Two cases are studied using SALUTE: electron evolution in presence and in absence of electric field.

As a mathematical model we consider Wigner equation for the nanometer and femtosecond transport regime. In the homogeneous case we solve a version of the Wigner equation called Levinson (with finite lifetime evolution), [18], or Barker-Ferry equation (with infinite lifetime evolution), [6]. Another formulation of the Wigner equation considers inhomogeneous case when the electron evolution depends on the energy and space coordinates. The problem is relevant e.g. for description of the ultra-fast dynamics of confined carriers. Particularly we consider a quantum wire, where the carriers are confined in the plane normal to the wire by infinite potentials. The initial condition is assumed both in energy and space coordinates.

The numerical results that we present in this paper are for the inhomogeneous case with applied electric field (see figures in the Numerical tests section). We recall the integral form of the quantum-kinetic equation, [22]:

$$
\begin{aligned}
f_w(z,k_z,t) =& f_w(z-\frac{\hbar k_z}{m}t+\frac{\hbar \mathbf{F}}{2m}t^2,k_z,0)+\int_0^t \partial t'' \int_{t''}^t \partial t' \int d\mathbf{q}'_\perp \int dk'_z \\
&\times \left[S(k'_z,k_z,t',t'',\mathbf{q}'_\perp) f_w\left(z-\frac{\hbar k_z}{m}(t-t'')+\frac{\hbar \mathbf{F}}{2m}(t^2-t''^2)+\frac{\hbar q'_z}{2m}(t'-t''),k'_z,t''\right)\right. \\
&\left. -S(k_z,k'_z,t',t''\mathbf{q}'_\perp) f_w\left(z-\frac{\hbar k_z}{m}(t-t'')+\frac{\hbar \mathbf{F}}{2m}(t^2-t''^2)-\frac{\hbar q'_z}{2m}(t'-t''),k_z,t''\right)\right] \\
S(k'_z,k_z,t',t'',\mathbf{q}'_\perp) =& \frac{2V}{(2\pi)^3}|G(\mathbf{q}'_\perp)\mathcal{F}(\mathbf{q}'_\perp,k_z-k'_z)|^2 \\
&\times \left[(n(\mathbf{q}')+1)\cos\left(\frac{\epsilon(k_z)-\epsilon(k'_z)+\hbar\omega_{\mathbf{q}'}}{\hbar}(t'-t'')+\frac{\hbar}{2m}\mathbf{F}.q'_z(t'^2-t''^2)\right)\right. \\
&\left. +n(\mathbf{q}')\cos\left(\frac{\epsilon(k_z)-\epsilon(k'_z)-\hbar\omega_{\mathbf{q}'}}{\hbar}(t'-t'')+\frac{\hbar}{2m}\mathbf{F}.q'_z(t'^2-t''^2)\right)\right]
\end{aligned}
\tag{2.1}
$$

Here, $f_w(z,k_z,t)$ is the Wigner function described in the $2D$ phase space of the carrier wave vector k_z and the position z, and t is the evolution time.

$\mathbf{F}=e\mathbf{E}/\hbar$, where $\mathbf{E}$ is a homogeneous electric field along the direction of the wire z, e being the electron charge and $\hbar$—the Plank's constant.

$n_{\mathbf{q}'}=1/(\exp(\hbar\omega_{\mathbf{q}'}/\mathcal{K}T)-1)$ is the Bose function, where $\mathcal{K}$ is the Boltzmann constant and T is the temperature of the crystal, corresponds to an equilibrium distributed phonon bath.

$\hbar\omega_{\mathbf{q}'}$ is the phonon energy which generally depends on $\mathbf{q}'=\mathbf{q}'_\perp+q'_z=\mathbf{q}'_\perp+(k_z-k'_z)$, and $\varepsilon(k_z)=(\hbar^2k_z^2)/2m$ is the electron energy.

$\mathcal{F}$ is obtained from the Fröhlich electron-phonon coupling by recalling the factor $i\hbar$ in the interaction Hamiltonian, Part I:

$$
\mathcal{F}(\mathbf{q}'_\perp,k_z-k'_z)=-\left[\frac{2\pi e^2\omega_{\mathbf{q}'}}{\hbar V}\left(\frac{1}{\varepsilon_\infty}-\frac{1}{\varepsilon_s}\right)\frac{1}{(\mathbf{q}')^2}\right]^{\frac{1}{2}},
$$

where (ε_∞) and (ε_s) are the optical and static dielectric constants. The shape of the wire affects the electron-phonon coupling through the factor

$$G(\mathbf{q}'_\perp) = \int d\mathbf{r}_\perp e^{i\mathbf{q}'_\perp \mathbf{r}_\perp} |\Psi(\mathbf{r}_\perp)|^2 \,,$$

where Ψ is the ground state of the electron system in the plane normal to the wire.

In terms of **numerical solution** the problem consists of determining different quantities of interest by evaluating linear functional of the following type:

$$J(f_w) = h(x) f_w(x) dx = (h, f_w), \tag{2.2}$$

The given function $h(x)$ depends on the choice of the physical quantities. In our quantum model, we are interested from the following physical quantities:

1. Wigner function for fixed evolution times;
2. The wave vector;
3. Electron density distributions;
4. The energy density.

In the inhomogeneous case the wave vector (and respectively the energy) and the density distributions are given by the integrals

$$f(k_z, t) = \int \frac{dz}{2\pi} f_w(z, k_z, t); \qquad n(z, t) = \int \frac{dk_z}{2\pi} f_w(z, k_z, t). \tag{2.3}$$

Our aim is to estimate these quantities (2.3), as well as the Wigner function (2.1) by quasi-MC approach.

At present, SALUTE numerical experiments use GaAs material parameters. In one of the next version of the SALUTE application, we will provide results for other types of semiconductors like Si or for composite materials.

Detailed description of MCMs for this problem can be found in [3, 12, 16]. Let us mention that MCMs have the advantage that MCMs estimate directly the necessary quantities, i. e. without calculating the solution of the Wigner function in the whole domain. The serious problem with MCMs is the large variance of the random variable which is proportional to the $\exp(T^2)$ where T is the evolution time. As the physicists are interested in the quantum effects for large evolution time, the problem becomes computationally very intensive—we have to perform billions of trajectories in order to obtain reasonable results. This was our motivation for applying quasirandom approach and using computational grid.

3. Quasirandom approach in SALUTE. Quasi-Monte Carlo methods and algorithms proved to be efficient in many areas ranging from physics to economy. We have applied quasirandom approach for studying quantum effects during ultra-fast carrier transport in semiconductors and quantum wires in order to reduce the error and to speedup the computations. Next, we have used scrambled sequences for two main reasons: (i) the problem is very complicated (the use of scrambling corrects the correlation problem found when we have used a purely quasi-Monte Carlo algorithm), and, (ii) the Grid implementation which needs parallel streams.

The computational Grid (or, shortly, the Grid) proved to be very efficient computing model. The Grid goes well beyond simple communication between computers and aims ultimately to turn the global network of computers into one vast computational resource. Using the Grid is especially useful for Monte Carlo applications as there the amount of similar calculations that has to be done is huge. Technically Grid coordinates resources which are not a subject to central administrative control and utilizes general-purpose protocols. Another distinction is that a Grid could in principle have access to parallel computers, clusters, farms, local Grids, even Internet computing solutions, and would choose the appropriate tool for a given calculation. In this sense, the Grid is the most generalized form of distributed computing. One major advantage of Monte Carlo methods is that they are usually very easy to be parallelized. This is, in principal, also true of quasi-Monte Carlo methods. However, the successful parallel implementation of a quasi-Monte Carlo application depends crucially on various quality aspects of the parallel quasirandom sequences used [8, 9]. Much of the recent work on parallelizing quasi-Monte Carlo methods has been aimed at splitting a quasirandom sequence into many subsequences which are then used independently on the various parallel processes, for example in [1, 2, 7]. This method works well for the parallelization of pseudorandom numbers, but due to the nature of quality in quasirandom numbers, this technique has some difficulties. Our algorithms are based on scrambling (suitable for heterogeneous computing environments).

3.1. Quasirandom sequences. We use scrambled Halton, Sobol and Niederreiter sequences.

Halton sequence. We use the modified Halton sequences introduced in [2] for which the discrepancy has a very small leading term. His construction is based on the existence of some numbers, called "admissible". Here we recall the definitions of admissible numbers and modified Halton sequence.

Definition 1. *Let $p_1, \ldots, p_s$ be distinct primes. The integers $k_1, \ldots, k_s$ are called admissible for them, if $p_i \nmid ki$ and for each set of integers $m_1, \ldots, m_s$, $p_i \nmid m_i$, there exists a set of integers $\alpha_1, \ldots, \alpha_s$, satisfying the congruences*

$$k_i{}^{\alpha_i} \prod_{1 \le j \le n,\ j \ne i} p_j{}^{\alpha_j} \equiv m_i (mod\ p_i), \quad i = 1, \ldots, s.$$

Definition 2. *Let $p_1, \ldots, p_s$ be distinct primes, and the integers $k_1, \ldots k_s$ are admissible for them. The modified Halton sequence $\sigma(p_1, \ldots, p_s; k_1, \ldots, k_s) = \{(x_n{}^{(1)}, \ldots, x_n{}^{(s)})\} n = \{0, \infty\}$ is constructed by setting each sequence $\{x_n{}^{(i)}\}_{n=0}{}^{\infty}$ to be a generalized Van der Corput - Halton sequence in base p_i, with the sequence of permutations $\tau_j{}^{(i)}(t)$ to be the reminder of $tk_i{}^j$ modulo p_i, $\tau_j{}^{(i)}(t) \in \{0, \ldots, p_i - 1\}$.*

Determining "admissible" generation of modified Halton sequence can be found in [2]. In the experiments described in this paper we use the following scrambling: We change the formulas for the permutations as

$$\tau_j{}^{(i)}(t) \equiv tk_i{}^{(j+1)} + b_j{}^{(i)} (mod\ p_i),$$

where the integers $b_j{}^{(i)}$ are chosen independently in the interval $[0, p_i - 1]$. The scrambled sequence has the same estimate for its discrepancy as if for any integers $m_1, \ldots, m_s$ the congruences

$$k_i{}^{\alpha_i} \prod_{1 \le j \le n,\ j \ne i} p_j{}^{\alpha_j} \equiv m_i (mod\ p_i), \quad i = 1, \ldots, s \tag{3.1}$$

have a solution, then the same is true for the congruences

$$k_i{}^{\alpha_i + 1} \prod_{1 \le j \le n,\ j \ne i} p_j{}^{\alpha_j} + b_j \equiv m_i (mod\ p_i), \quad i = 1, \ldots, s. \tag{3.2}$$

The chosen algorithm is very fast, requires a small amount of memory and generates the terms of sequences with maximal error less that 10^{-14} when 10^6 terms are generated. It shows superior results compared to other Halton generators.

Niederreiter and Sobol sequences. We use the Definition 3 (below), which covers most digital (t, m, s)-nets in base 2. The Sobol sequence is a (t, s)-sequence in base 2 and is a particular case of this definition.

Definition 3. *Let $A_1, \ldots, A_s$ be infinite matrices $A_k = \{a_{ij}^{(k)}\}$, $i, j = 0, 1, \ldots$, with $a_{ij}^{(k)} \in \{0, 1\}$, such that $a_{ii}^{(k)} = 1$ for all i and k, $a_{ij}^{(k)} = 0$ if $i < j$. The $\tau^{(1)}, \ldots, \tau^{(s)}$ are sequences of permutations of the set $\{0, 1\}$. Each non-negative integer n may be represented in the binary number system as*

$$n = \sum_{j-0}^{r} b_i 2^i.$$

Then the nth term of the low-discrepancy sequence σ is defined by

$$x_n^{(k)} = \sum_{j=0}^{r} 2^{-j-1} \tau_j^{(k)} (\oplus_{i=0}^{j} b_i a_{ij}^{(k)}),$$

where by $\oplus$ we denote the operation of bit-wise addition modulo 2.

Next lemma explains how we generate consecutive terms of the sequence.

Lemma. *Let σ be a sequence or net satisfying Definition 3, and let the non-negative integers n, p, m be given. Suppose that we desire the first p binary digits of elements in σ with indices of the form $2^m j + n < 2^p$; this implicitly defines a set of compatible $j's$. Thus the only numbers we need to compute are*

$$y_j^{(k)} = \lfloor 2^p x_{2^m j + n}^{(k)} \rfloor.$$

The integers $\{v_r^{(k)}\}_{r=0}^{\infty}$*, which we call "twisted direction numbers", are defined by*

$$v_r^{(k)} = \sum_{t=0}^{p-1} 2^{p-1-t} \oplus_{j=m}^{p-1} a_{tj}^{(k)}.$$

Suppose that the largest power-of-two that divides $2^m(j+1)+n$ *is* l*, i. e.* $2^m(j+1)+n = 2^i(2K+1)$*. Then the following equality holds*

$$y_{j+1}^{(k)} = y_j^{(k)} \oplus v_i^{(k)}.$$

To obtain the results presented in this paper we have used the scrambled Sobol and Niederreiter sequences. The algorithm for generating scrambled Sobol sequence is described in [1]. We have modified this algorithm for generating scrambled Niederreiter sequence. We have to note that not all optimizations for Sobol sequence can be applied for Niederreiter (due to the structure of metrices $A's$. This algorithm allows consecutive terms of the scrambled sequence to be obtained with essentially only two operations per coordinate: one floating point addition and one bit-wise xor operation (this omits operations that are needed only once per tuple). This scrambling is achieved at no additional computational cost over that of unscrambled generation as it is accomplished totally in the initialization. In addition, the terms of the sequence are obtained in their normal order, without the usual permutation introduced by Gray code ordering used to minimize the cost of computing the next Sobol element. This algorithm is relatively simple and very suitable for parallel and grid implementation.

The mathematical explanations can be found in [1], here we present the algorithm in pseudo code.

- Input initial data:
  - if the precision is single, set the number of bits b to 32, and the maximal power of two p to 23, otherwise set b to 64 and p to 52;
  - dimension s;
  - direction vectors $\{a_{ij}\}, i = 0, p, j = 1, \ldots, s$ representing the matrices $A_1, \ldots, A_d$ (always $a_{ij} < 2^{i+1}$);
  - scrambling terms $d_1, \ldots, d_s$ - arbitrary integers less than 2^p, if all of them are equal to zero, then no scrambling is used;
  - index of the first term to be generated - n;
  - scaling factor m, so the program should generate elements with indices $2^m j + n$, $j = 0, 1, \ldots$.
- Allocate memory for $s \star l$ b-bit integers (or floating point numbers in the respective precision) $y_1, \ldots, y_s$..
- Preprocessing: calculate the twisted direction numbers v_{ij}, $i = 0, \ldots, p-1,\ j = 0, \ldots, s$:
  - for all j from 1 to s do
  - for $i = 0$ to $p-1$ do
  - if $i = 0$, then $v_{ij} = a_{ij} 2^{p-m}$, else $v_{ij} = v_{i-1j} \mathbf{xor} (a_{i+m,j} \star (2^{p-i-m}))$;
- Calculate the coordinates of the n-th term of the Sobol sequence (with the scrambling applied) using any known algorithm (this operation is performed only once). Add +1 to all of them and store the results as floating point numbers in the respective precision in the array y.
- Set the counter N to the integer part of $\frac{n}{2^m}$.
- Generate the next point of the sequence:
  - When a new point is required, the user supplies a buffer x with enough space to hold the result.
  - The array y is considered as holding floating point numbers in the respective precision, and the result of subtracting 1. from all of them is placed in the array x.
  - Add 1 to the counter N;
  - Determine the first nonzero binary digit k of N so that $N = (2M+1)2^k$ (on the average this is achieved in 2 iterations);
  - Consider the array y as an array of b-bit integers and updated it by using the k^{th} row of twisted direction numbers:
    for $i = 1$ to d do
    $y_i = y_i \mathbf{xor} v_{ki}$.
  - Return the control to the user. When a new point is needed, go to beginning of this paragraph (Generate the next point of the sequence).

Parallelization. There are three basic ways to parallelize quasirandom number sequences:

- Leap-frog - The sequence is partitioned in turn among the processors like a deck of cards dealt to card players.
- Sequence splitting or blocking- The sequence is partitioned by splitting it into non-overlapping contiguous subsections.
- Independent sequences - Each processor has its own independent sequence.

The first and second schemes produce numbers from a single quasirandom sequence. The third scheme needs a family of quasirandom sequences. Scrambling techniques can generate such a stochastic family of quasirandom sequences from one original quasirandom sequence. Numerical calculations presented in this paper are done using blocking. In this way we can exactly repeat the results for comparison.

3.2. Hybrid Algorithms in SALUTE. We have constructed hybrid Monte Carlo algorithms that use pseudo-random numbers for some dimensions and scrambled quasi-random numbers for the other dimensions.

A schematic description of the algorithm is given below, assuming that we only need to compute the Wigner function at one point $(k1, z1)$. In the algorithm, ϵ_1 is the truncation parameter.

- Input number of trajectories to be used N, relaxation time T, other parameters, describing the initial condition.
- For i from 1 to N sample a trajectory as follows:
  - set time $t := T$, weight $W := 1, k = k_1, z := z_1$
  - prepare the next point of the quasirandom sequence to be used (Niederreiter, Halton or Sobol) $(x_1, x_2, \ldots, x_n)$, with n sufficiently big ($n = 100$ in our case), and set $j = 1$
  - repeat until $t > \epsilon_1$:
    - k is simulated using pseudorandom numbers
    - t', t are simulated using consecutive dimensions of the quasirandom sequence, i. e. the points x_{2j-1}, x_{2j}, by the formula

$$t_2 := tx2_{j-1}, t_1 := t_2 + x_2 j(t - t_2), t' := t_1, t = t_2$$

    - multiply the weight: $W := W \star t(t - t_2)$
    - compute the two kernels K_1 and K_2
    - select which one to use with probability proportional to their absolute values.
    - multiply the weight: $W := W \star (|K1| + |K2|)sgn(K_m)$ if K_m is the kernel selected
    - sample q using a spline approximation of the inverse function
    - multiply the weight by the appropriate integral: $W := W \star I$
    - modify k, depending on the kernel and the electric field applied: $k_{new} = k - c_3 \star (t - t_2)$ if K_1 was chosen or $k_{new} = k - c_3 \star (t - t_2)$ if K_2 was chosen
    - modify $z : z_{new} = z - c_1 \star k \star (t - t_2) - c_2 \star (t - t_2) \star (t + t_2)$
    - compute the contribution of this iteration to the Wigner function: add $W \star \psi(z, k)$ to the estimator, where $\psi(z, k)$ is the value of the initial condition
    - increment $j := j + 1$

The constructive dimensionality of the algorithm is $4n$, where n is the maximal length of the trajectory. We use $2n$ pseudorandom numbers for each trajectory, and the dimensionality of the Halton sequence is $2n$.

4. Grid implementation. A computational grid is a computing environment which enables the unification of geographically widely distributed computing resources into one big (super)computer [11]. The individual computing resources commonly consist mostly of computer clusters or several individual computers, which are interconnected by a high-speed wide area network. At present, the grid is intended for supporting e-Science, however the technology itself is very adaptable for a very wide area of future computer use. The grid accumulates and coordinates as much computing power as possible and make it available for use by applications, which have a particularly high demand for computing resources.

In order to compute the physical quantities (2.1),(2.3) SALUTE application requires considerable computational power and time to obtain sufficiently accurate results. Hundreds (thousands) of jobs with different input data have been implemented on grid clusters included in the SEEGRID infrastructure.

The SEEGRID infrastructure integrates computational and storage resources in South Eastern Europe. Currently there are more than 40 clusters with a total of more than 3000 CPUs and more than 400 TB of storage and this infrastructure is a part of European Grid infrastructure named EGEE [25]. The peculiarities of the region are that the network connectivity of many of these clusters is insufficient, which implies the necessity to avoid network-hungry applications and emphasize computationally intensive applications, that make efficient use of the available resources. It also imposes the need of fault-tolerant implementations.

The SEEGRID infrastructure was built using the gLite middleware [26]. Each of the SEEGRID clusters has the mandatory Grid services:

- Computing Element
- Worker Nodes
- Storage Element
- MON box

The Worker Nodes provide the computational resource of the site, and the Storage Element provides the storage resources. The set of services, that are not tied to the specific site are called core services. They include

- VOMS (Virtual organisation management system)
- MyProxy
- R-GMA registry/schema server (distributed data-base)
- BDII (provides comprehensive information about the resources)
- WMS (distributes and manages the jobs among the different grid sites)
- FTS (file transfer service)
- AMGA (metadata catalog)

4.1. Grid implementation scheme. The need of performing a large number of tests on the EGEE and SEEGRID infrastructures allows us to develop a grid implementation scheme in order to facilitate the execution and monitoring of the submited tasks. In our grid implementation scheme we incorporated the use of the FTS and AMGA services, available in the gLite, and we were able to include the estimation of several new physical quantities, which increased the total amount of data to be generated, stored, processed and visualized.

On the User Interface (UI) computer the scientist launches the Graphical User Interface (GUI) of the application (see Fig. 1). The job submission, monitoring and analysis of the results is controlled from there.

The jobs are monitored from a monitoring thread, started from the GUI, and information about their progress is displayed to the user. Another thread run from the GUI is responsible for collecting the output results from the various Storage Elements to the local one. For each output file a request for transfer is sent to the File Transfer Service (FTS) computer.

The computational tasks are submitted using a messaging broker, that follows the AMQP protocol. The grid jobs contact the AMQP broker and obtain the tasks from a message queue.

The *AMGA* (ARDA Metadata Catalog) is used to hold information about the results obtained so far by the user—for example input parameters, number of jobs executed, execution date etc.

The WMS sends the job to the Grid sites. When the job starts on the WN (Worker Node), it downloads the executable from the Storage element. The executable obtains the input parameters from the AMQP broker, performs the computations and stores the results in the local Storage Element. It registers the output. One of the Worker Nodes is responsible for gradual accumulation of the output of the jobs. At regular intervals the accumulated results are registered and made available to the user.

The FTS is used in order to limit the number of files that are transferred simultaneously, because of the limited bandwidth available. In this way we also avoid some scalability limitations of the middleware and we try not to overload the Storage Elements. This approach is efficient, because in most cases it will not lead to increase of the total time necessary for completing all transfers, since they compete for the same network resource. Additional benefit of the FTS is that it provides reliable transfer of the files, by retrying the transfers if necessary.

5. Numerical Tests and Grid performance analysis. The problems arising when we solve the Wigner equation using Monte Carlo approach are due to the large statistical error. This error is a product of two factors: standard deviation and sample size on the power one half. The standard deviation increases exponentially with time, so in order to achieve reasonable accuracy, we must increase considerably the sample size. This implies the need of computational resources. Using the grid and above described grid implementation scheme, we were able to obtain new results about important physical quantities: Wigner function, wave vector, electron density

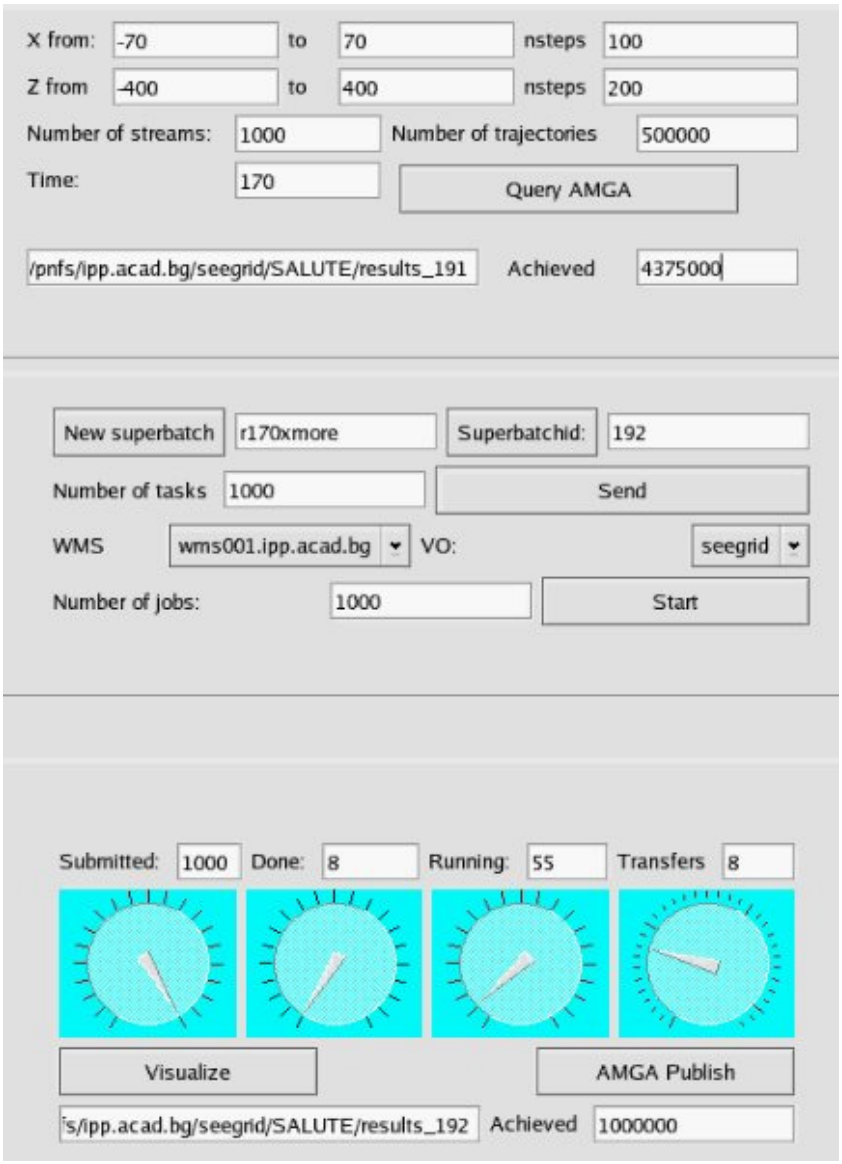


FIG. 4.1. *Graphical User Interface (GUI) for submission and monitoring of SALUTE jobs, and accumulation and visualization of their results.*

and energy density. The results presented here are for inhomogeneous case with applied electric field for 140 femtoseconds evolution time. Normally, the execution times of the jobs at the different sites are similar, and the delay in starting is caused by lack of free Worker Nodes. Thus our new scheme allows the user to achieve the maximum possible throughput.

We have performed experiments with pseudorandom, and scrambled Niederreiter, Sobol and Halton sequences. In order to achieve the sufficient accuracy we have implemented 400 jobs (each with 4000000 trajectories) with our Monte Carlo algorithm, and 64 jobs (each with 2^{22} trajectories) with the hybrid algorithm (correspondingly, with scrambled Halton, Niederreiter and Sobol sequences). The mean square errors of $rez^{(i)}_{MCM} - rez^{(i)}_{Hybrid}$, where rez^i means wave vector, electron density energy density and Wigner function has order ranging from $O(10^{-4})$ to $O(10^{-5})$. But to achieve the **same results with the hybrid method we performed 6 times less trajectories**.

On the Figures 2, 3 and 4 one can see the quantum effects—there is no symmetry when electric field is applied. The results obtained with MCM and the three hybrid algorithms are plotted on the same picture for

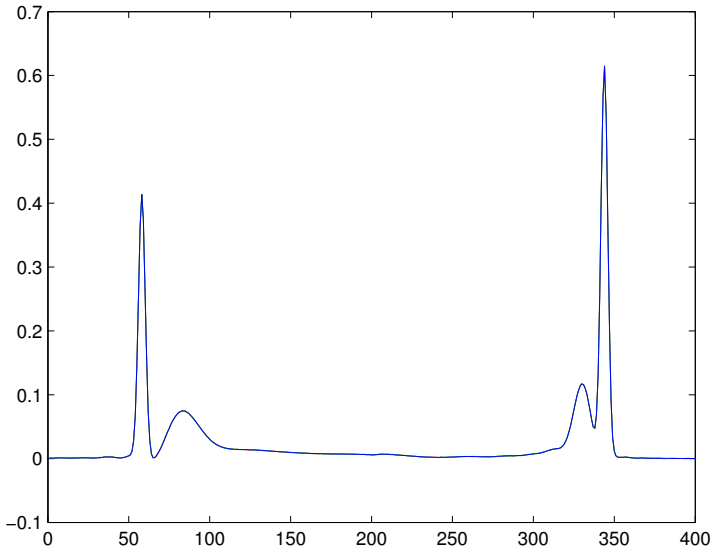


FIG. 5.1. *Wave vector obtained with MCM and Hybrid1 (with Niederreiter) and Hybrid2 (with Halton) algorithm. The electric field is 15[kW/cm] along to the nanowire.*

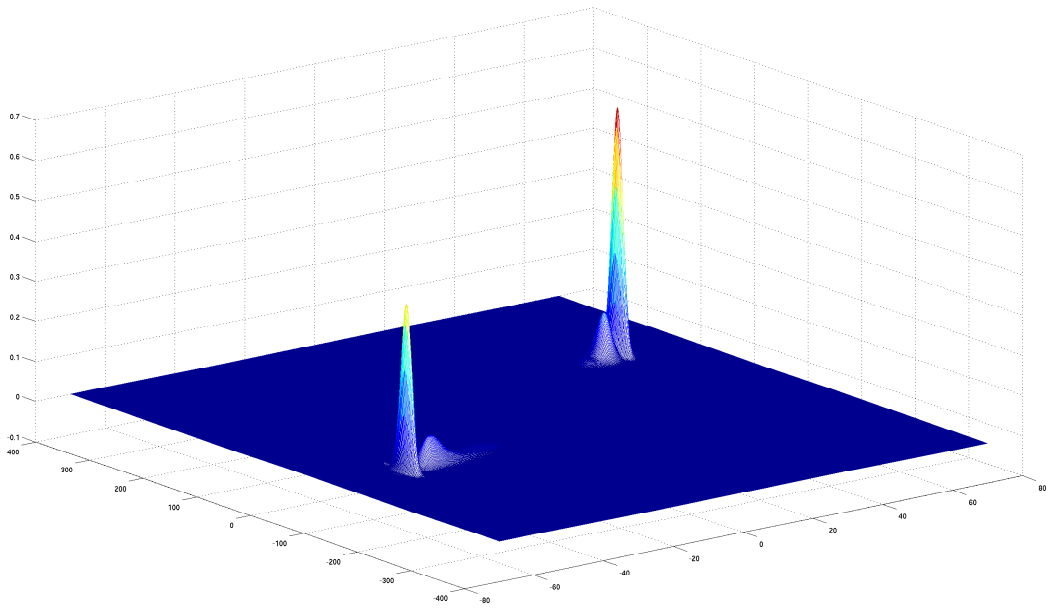


FIG. 5.2. *The Wigner function at* $140fs$ *presented in the plane* $z \times k_z$*. The electric field is 15[kV/cm] along to the nanowire.*

each of the estimated quantities. They are not visible on Fig. 2 because the error is very small. The graph on Fig. 4 is in logarithmic scale and the differences can be seen. The best results are obtained using the Niederreiter sequence (in our version with the described scrambling algorithm).

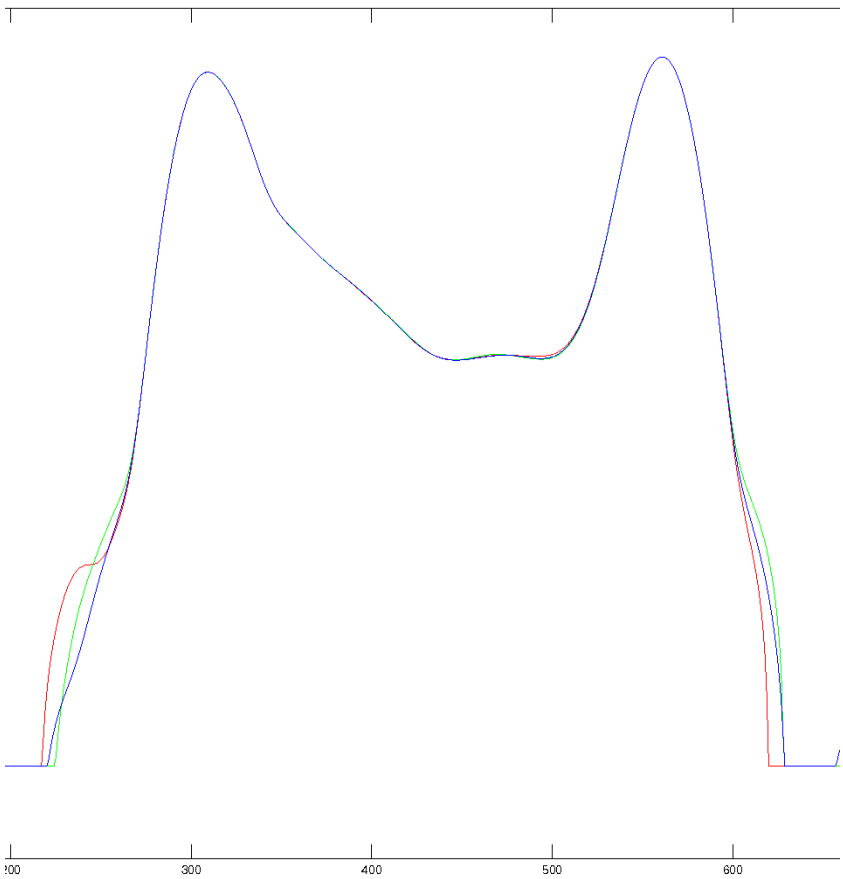


FIG. 5.3. *Electron density obtained with MCM and Hybrid1 (Niederreiter) and Hybrid2 (Halton). The electric field is 15[kV/cm] along to the nanowire.*

6. Conclusion. The use of scrambled quasi-random sequences allows for faster convergence of the algorithms compared to pure Monte Carlo, providing also aposteriory error estimation and avoiding singularities. The chosen computational infrastructure - Grid, enabled the achievement of interesting new results through extensive computations, which would otherwise require considerable amount of time.

Further improvement of the convergence of the algorithms may be obtained by employing improved scrambling algorithms of modified quasi-random sequences and this will be an important direction for our future work.

REFERENCES

[1] E. Atanassov, "A New Efficient Algorithm for Generating the Scrambled Sobol' Sequence", *Numerical Methods and Applications*, Springer-Verlag, LNCS **2542** (2003), 83-90.

[2] E. I. Atanassov, M. K. Durchova, "Generating and Testing the Modified Halton Sequences", *Numerical Methods and Applications*, Springer-Verlag, LNCS **2542** (2003), 91-98.

[3] E. Atanassov, T. Gurov, A. Karaivanova, M. Nedjalkov, "Monte Carlo Grid Application for Electron Transport", *LNCS* **3993**, Springer, 2006: 616-623.

[4] E. Atanassov, T. Gurov, A. Karaivanova, SALUTE Application for Quantum Transport New Grid Implementation Scheme, *Proceedings of Spanish e-Science Grid Conference*, ISBN: 987-84-7834-544-1, NIPO: 654-07-015-9, pp.23-32.

[5] E. Atanassov, A. Karaivanova, and T. Gurov, Quasi-random approach in the Grid application SALUTE, to appear in: Springer, LNCS (Proceedings of PPAM 2009).

[6] J. Barker, D. Ferry, Self-scattering path-variable formulation of high field time-dependent quantum kinetic equations for semiconductor transport in the finite collision-duration regime. *Phys. Rev. L.*, **42**, 1979, pp. 1779-1781.

[7] B. C. Bromley, Quasirandom Number Generation for Parallel Monte Carlo Algorithms, *Journal of Parallel Distributed Computing*, **38(1)**, 1996, 101-104.

[8] H. Chi and E. Jones, "Generating Parallel Quasirandom Sequences by using Randomization", *Journal of distributed and parallel computing*, **67(7)**: 876-881, 2007.
[9] H. Chi and M. Mascagni, "Efficient Generation of Parallel Quasirandom Sequences via Scrambling," *Lecture Notes in Computer Science*, **4487**, Springer, 2007, 723-730.
[10] M. V. Fischetti, S.E. Laux, "Monte Carlo Analysis of Electron Transport in Small Semiconductor Devices Including Band-Structure and Space-Charge Effects", *Phys. Rev. B*, **38**, 1988, pp. 9721-9745.
[11] J. Foster, C. Kesselmann, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1998.
[12] T. Gurov et al., Femtosecond Evolution of Spatially Inhomogeneous Carrier Excitations: Part II: Stochastic Approach and GRID Implementation, *LNCS* **3743**, Springer, 2006, pp. 157-163.
[13] Karaivanova, A., Chi, H. and Gurov, T., Quasi-random walks on balls using c.u.d, In: *Lecture Notes in Computer Science*, Vol. 4310, Springer-Verlag, Berlin Heidelberg New York (2007) 165–172.
[14] A. Karaivanova, H.Chi, T. Gurov, Error Analysis of Quasirandom Walks on Balls, MIPRO/GVS 2009, ISBN 978-953-233-044-1, pp. 285-289.
[15] A. Karaivanova and N. Simonov, A Quasi-Monte Carlo Methods for Investigating Electrostatic Properties of Organic Pollutant Molecules in Solvent, Springer Lecture Notes in Computer Science **3743**:172-180, 2006.
[16] H. Kosina, M. Nedjalkov and S. Selberherr, "An event bias technique for Monte Carlo device simulation", *Math. and Computers in Simulation*, **62**, 2003, pp. 367-375.
[17] C. Lecot and B. Tuffin, Quasi-Monte Carlo Methods for Estimating Transient Measures of Discrete Time Markov Chains,. In *Fifth International Conference on Monte Carlo and Quasi- Monte Carlo Methods in Scientific Computing*, Springer, pages 329-344, 2002.
[18] I. Levinson, Translational Invariance in Uniform Fields and the Equation for the Density Matrix in Wigner Representation, *Sov. Phys. JETP*, **30**, 1970, pp. 362367.
[19] W. Morokoff and R. E. Caflisch, A quasi-Monte Carlo approach to particle simulation of the heat equation, *SIAM J. Numer. Anal.*, **30**: 1558–1573, 1993.
[20] M. Nedjalkov, I. Dimov, F. Rossi, C. Jacoboni, "Convergence of the Monte Carlo Algorithm for the Solution of the Wigner Quantum-Transport Equation", **J. Math. Comput. Model**, **23**, 8/9, 1996, pp. 159-166.
[21] Nedjalkov M., Kosina H. Selberherr S., Ringhofer C., Ferry D.K., Unified particle approach to Wigner-Boltzmann transport in small semiconductor devices, Physical Review B, **70**, pp. 115319-115335,2004.
[22] M. Nedjalkov et al., Wigner transport models of the electron-phonon kinetics in quantum wires, *Physical Review B*, vol. **74**, 2006, pp. 035311-1035311-18.
[23] H. Niederreiter, *Random Number Generations and Quasi-Monte Carlo Methods*. SIAM, Philadelphia, 1992.
[24] T. C. Schmidt and K. Moehring, Stochastic Path-Integral Simulation of Quantum Scattering, *Physical Review A*, vol. **48**, no. 5, 1993, pp. R3418.R3420.
[25] `http://goc.grid.sinica.edu.tw/gstat`
[26] `http://glite.web.cern.ch/glite/documentation/default.asp`

Edited by: Marcin Paprzycki
Received: April 10, 2010
Accepted: June 21, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 149–159. http://www.scpe.org
ISSN 1895-1767

MANAGEMENT OF HIGH PERFORMANCE SCIENTIFIC APPLICATIONS USING MOBILE AGENTS BASED SERVICES

SALVATORE VENTICINQUE*, ROCCO AVERSA*, BENIAMINO DI MARTINO*, RENATO DONINI*, SERGIO BRIGUGLIO†, AND GREGORIO VLAD†

Abstract. High performance scientific applications are currently implemented using native languages for optimizing performance and utilization of resources. It often deals with thousands of code lines in FORTRAN or in C built of legacy applications. On the other hand new technologies can be exploited to improve flexibility and portability of services on heterogeneous and distributed platforms. We propose here an approach that allows programmers to extend their applications to exploit this kind of services for management purposes. It can be done simply by adding some methods to the original code, which specialize application management on occurrence of particular events. We mean that applications do not need to be rewritten into different languages or adopting specific programming models. We implemented a native console that is used by Mobile Agents to control the application life-cycle. Agents implement a mobile service that supports check-pointing, suspension, resume, cloning and migration of managed applications. A WSRF interface has been provided to Grid users who do not need to be aware about agents technology. We used a FORTRAN code for simulation of plasma turbolence as a real case study.

1. Introduction. We aim here at investigating how Mobile Agents technology can be used to develop advanced services for management of resources in distributed systems. Mobile Agents mechanisms such as autonomy, reactivity, clone-ability and mobility can be exploited for resource management and load balancing when system conditions change dynamically. Most of all mobile agents platforms are executed by Virtual Machines which make transparent the hardware/software architecture of the hosting node. It allows to distribute and execute mobile agents code on heterogeneous environments in a flexible way. On the other hand, most of legacy applications have been implemented by languages such as FORTRAN and C, and they are compiled for a target machine in order to optimize their performances. Here we present a mobile agent based service that allows for management of native applications on heterogeneous distributed systems. Agent technology has been used to provide management facility on any node where the execution of applications will be started. Programmers, in order to exploit the management service, can extend their application without modify the original code, but by overriding some methods which specialize the application life-cycle. We aim at supporting checkpoint, resume, migration and monitoring. Furthermore service is targeted to each Grid user, who is unaware about Agent technology and can exploit services facilities by a compliant WSRF interface. A console that allows the application control by an agent has been designed and implemented. An agent based service designed and implemented to automatically perform management strategies. In the second section related works on management services and load balancing mechanism are described. In section 3 we describe the facilities we have implemented in order to control the application life-cycle. In section 4 the software architecture of our service is presented. Section 5 provides an example of simple application that has been extended in order to exploit the service.

2. Related work. Application management and migration are mechanisms developed in many environments for common purposes. There are many platforms which exploit migration to implement load balancing in distributed and parallel systems. When we deal with homogeneous clusters, process migration is supported to share resources dynamically and adaptively. MOSIX [1] provides a set of algorithms to react in real time to the changes of resources utilization in a cluster of workstations. Migration of processes is preemptive and transparent. The objective is to provide high performances to parallel and sequential applications. OpenMosix [2] is a decentralized version of MOSIX. Each node behaves as an autonomous system. Kerrighed [3] supports thread migration. When a node is less loaded a process is moved there from a more busy node. OpenSSI [3] does not need to save part of the process on the original hosting node. System calls are called locally. Not all the processes are candidates for migration.

On the other hand in heterogeneous systems process migration is not generally supported. Different environments are virtualized by a middleware that hides architectural details to the applications and supports portability. In this case is possible to migrate code and status of applications, but not to resume the process status. An hybrid approach that manage a Grid of homogeneous clusters is presented in [4] as an advance

*Second University of Naples, Aversa (CE), Italy, emails: {rocco.aversa, salvatore.venticinque}@unina2.it, renato.donini@gmail.com, beniamino.dimartino@unina.it
†Associazione EURATOM-ENEA sulla Fusione, Frascati, Rome, Italy,emails: {brigulgio, vlad}@enea.it

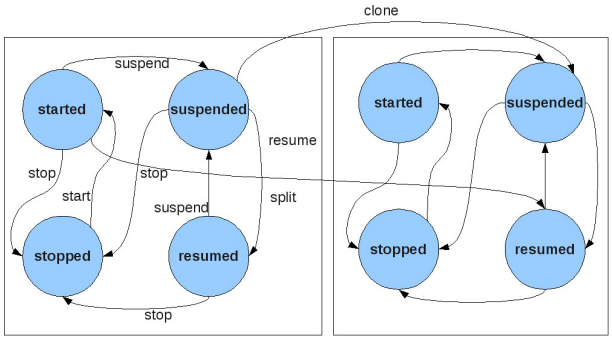


FIG. 3.1. *Application life-cycle*

of research cited above. Some relevant contributions, which exploit Mobile Agents technology are [5, 6]. We aim at exploiting flexibility of Mobile Agent programming to manage legacy applications without changing the original code and without rewriting the application into another language or adopting a new programming paradigm. Some approaches which should be compared with the one presented in this paper are cited below. [7] presents an approach to support mobility of applications in a GRID environment. A mobile agent is described by an UML-like language called blueprint. The blueprint description is transferred together its status. The receiving platform translates the agent blueprint description in a Java or a Python application that implement the original functionalities Blueprint is a language for a high level specification of agent functionalities and of its operational semantics. A specification describes the behavior of an agent in terms of its basic building blocks: components, control flow and data flow. Agent factories interpret the agent blueprint description and generate executable code composed of, for example, Java, Python, or C components. To support migration of common user applications, in [8], authors provide the possibility to insert some statements, such as go( ) or hop( ), between blocks of computation, such as for/while loops. A pre-compiler, such as ANTLR for C/C++ and JavaCC for Java source code, is used to substitute these statements with code that implements mobility. A user program may be entirely coded in C/C++ and executed in native mode. As a result Java agent starts the native code using a wrapper implemented by the JNI technology. In our approach programmers who want to support migration do not need to deal with any models, but they have just to handle such events which ask to save or resume the application status.

3. Management facilities and application life-cycle. We exploited Mobile Agents technology to allow services execution on heterogeneous platforms. We mean that we can execute monitoring and control facilities on heterogeneous nodes in a distributed environment. Furthermore we can move dynamically a service instance from a node to another resuming the execution at destination. Mechanisms of Mobile Agents technology have been adopted to design and implement advanced management facilities. Besides we provide the possibility to extend the same facilities to the managed application. Our model of application life-cycle is shown in Fig: 3.1. Regardless of the location, the process state could assume the following values: started, suspended, stopped and resumed. In the suspended mode, the application status has been saved in order to allow process restoring when a resume action will be invoked. Let us clClrify that the application status is saved not the process one. That's because we aim at supporting mobility across heterogeneous architectures. Life-cycle can be monitored and controlled by a software console that generates and handles the following events:

1. *start*: starts the execution of a native application on a cluster node
2. *suspend*: suspends the native application saving its execution status.
3. *resume*: resumes the native application restoring the status saved on the last suspension.
4. *stop*: stops the native application.
5. *checkpoint*: saves the status of the native application without stopping its execution.
6. *migrate*: migrates the native application to a target node restoring its status at destination.
7. *split*: clones the native application and splits the job between the clones.

We have developed both an interactive GUI and a batch interpreter for submitting commands to the application console.

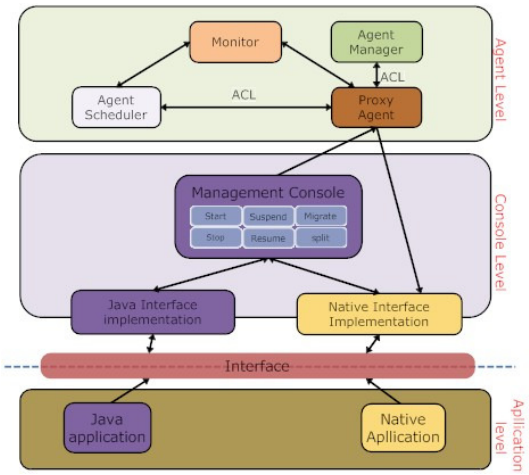


FIG. 4.1. *Software Architecture*

4. Service architecture. As shown in Fig:4.1 software architecture of the management service is composed of different elements at different levels. A first set of components implements a portable service executed by Mobile Agents. Distributed Agents perform different roles. They implement user interface, application monitoring and control, code management. A monitor detects relevant changes of system conditions and notifies these events to the Agent Scheduler. The Agent Scheduler reacts in order to avoid performance degradation. It communicate and coordinates other agents. The Proxy Agent is responsible of the application execution. It receives requests from other agents and manages the application by a Java console. An abstract console defines the interface between the proxy and the application. It is obviously independent by the kind of application that will be controlled. Native applications will be linked to the console by mean of a native implementation of abstract methods. In order to support the execution on heterogeneous architectures, the programmer has to make available a new shared library that overrides native methods and that has been compiled for a target architecture. Libraries will be stored on a remote repository and the proxy agent is able to automatically download them, when the execution is moved to a new heterogeneous node.

4.1. Agent level.

4.1.1. Agent Proxy. An Agent Proxy is delegated to manage an instance of users' application. It can:

- save and resume the status of the native application (save, resume);
- migrate, clone or split the application;
- handle special conditions such as termination.

For instance whenever the agent manager requires the migration of a native application to a selected node, the Agent Proxy has to transparently:

- suspend the native application as soon as possible;
- save the status of the native application;
- migrate to the target node;
- detect the target node hardware and software equipment;
- whenever it is necessary download and install the right library;
- restore the status of the native application;
- resume the native application.

4.1.2. Agent Manager. The Agent Manager provides a graphic interface for creating, migrating, cloning and destroying both Agent Proxies and the native applications. It sends standard ACL messages (ACL stands for Agent Communication Language) to ask for the actions ProxyAgents should take. The Manager allows to load references to libraries which have been built and made available for different hardware and software architecture. It can handle independently multiple execution instances of the same application. A snapshot of the Management Console GUI is shown in Figure 4.2.

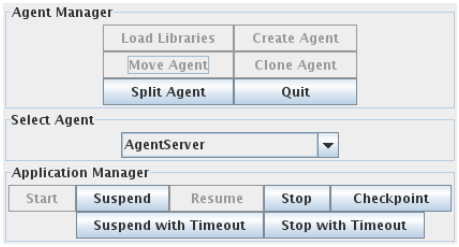


FIG. 4.2. *Management Console Gui*

4.1.3. Agent Monitor. Our architecture includes a monitor module to detect relevant changes of system conditions and notifies these events to the Autonomous Agent. As shown in the following the system is able to react to the events which affect the performance of the both service and application. The way to generate the events to send to the Batch Agent, could be less o more complex according to the particular requirements. Currently a simple configuration of management strategy based on threshold mechanisms is supported. The agent monitor checks the application's performance and compares it with a target input, such as the throughput of a web server; when a performance degrade was detected the right actions could be performed.

4.1.4. Agent Scheduler. The Agent Scheduler uses management facilities provided by Agent Proxies in order to carefully distribute the cluster workload or to optimize performance of applications. For instance, it can migrates native applications from a platform to another one that is less loaded. It can redistribute groups of applications in order to reduce network traffic by minimizing inter-communications, or reducing the overhead due to data transfer. Actually the user can configure the Scheduler behavior by associating a set of actions to a notification event. When a specific ACL messages has been received from the agent monitor the related batch file is interpreted and executed. A batch file can be written as described in Figure 4.1.4. In the example we show a sequence of commands which are executed on the occurrence of two events: *idle_node* and *busy_node*. Parameters of commands are extracted from content of related events notified by ACL messages.

```
<?xml version="1.0" encoding="UTF-8" ?>
<batch>
 <activation>
   <and>
     <event name="idle_node">
     <event name="busy_node">
   </and>
 </activation>
 <sequence>
  <operation>
   <command type="suspend" agent="$busy_node.agent_name[0]"/>
  </operation>
  <operation>
   <command type="move" agent="$busy_node.agent_name[0]">
    <parameters>
     <parameter>$idle_node.container[0] </parameter>
    </parameters>
   </command>
  </operation>
  <operation>
   <command type="resume" agent="$busy_node.agent_name[0]"/>
  </operation>
 <sequence>
</batch>
```

FIG. 4.3. *An XML batch file define the list of action that must be taken on an event occurence*

We are planning to adopt a more complete language such as the ones which support choreography or orchestration [9].

4.2. Console level. In order to make transparent the management of different kinds of applications we defined the ApplicationManager abstract class. An implementation needs to support management of each kind of application. We provided a NativeManager class with java and native methods. Other classes implements special utilities. In particular:

- ApplicationManager: is an abstract class that represents an application manager.
- NativeManager: is a class that extends ApplicationManger It overrides abstract methods in order to interface with native applications.
- ApplicationThread: is a thread that starts native application and waits for events.
- Library: is a class that contains a set of parameters, which are used to identify and retrieve the compliant version of application library for the target machine architecture.
- ManagementException: is a class that implements an exception that could be thrown by the ApplicationManager.

JNI (Java Native Interface) is the technology that has been used to link the native implementation of the ApplicationManager to the Java implementation. JNI defines a standard naming and calling convention so the Java virtual machine can locate and invoke native methods. A native implementation of these methods have been developed in POSIX C and compiled for different architectures (AMD64, IA32, ...). As it is shown in Figure 4.4, in order to allow its application to be managed a programmer has to add those methods which override the ones defined in the native console. Linking the original application code with the new methods and the native console, a dynamic library for a target machine can be built. The native console is implemented by two POSIX processes: a father and its son. The son is spawn when the application starts. The father waits for requests from the Java part of the service, forwards them to the son that executes the application code by POSIX signals. The son before to start registers signal handlers and communicate by two POSIX pipes with the father. Pipes are used by the son to communicate to the father the file name where the application status has been stored and to communicate events such as a termination. Handlers can be specialized by the programmer by overriding the following methods (using C, FORTRAN, or any other native languages): *start_()*, *resume_(char status[])*, *stop_(char status[])*, *suspend_(char status[])*, *checkpoint_(char status[])*, *split_(char status[])*, whose meanings has been described in the previous sections.

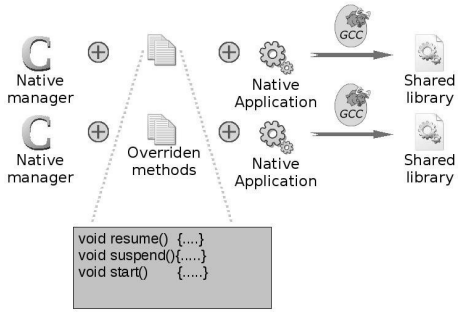


FIG. 4.4. *Building a new library*

5. Plasma turbulence simulation. As case study, we chose a Fortran application for Particle-in-cell simulation. Particle-in-cell simulation consists [15] in evolving the coordinates of a set of N_{part} particles in certain fluctuating fields computed (in terms of particle contributions) only at the points of a discrete spatial grid and then interpolated at each particle (continuous) position. Two main strategies have been developed for the workload decomposition related to porting PIC codes on parallel systems: the *particle decomposition* strategy [11] and the *domain decomposition* one [12, 13]. Domain decomposition consists in assigning different portions of the physical domain and the corresponding portions of the spatial grid to different processes, along with the particles that reside on them. Particle decomposition, instead, statically distributes the particle population among the processes, while assigning the whole domain (and the spatial grid) to each process. As a general fact, the particle decomposition is very efficient and yields a perfect load balancing, at the expenses of memory overheads. Conversely, the domain decomposition does not require a memory waste, while presenting

particle migration between different portions of the domain, which causes communication overheads and the need for dynamic load balancing [14, 13]. The typical structure of a PIC code for plasma particle simulation can be represented as follows. At each time step, the code

1. computes the electromagnetic fields only at the N_{cell} points of a discrete spatial grid (*field solver* phase);
2. interpolates the fields at the (continuous) particle positions in order to evolve particle phase-space coordinates (*particle pushing* phase);
3. collects particle contribution to the pressure field at the spatial-grid points to close the field equations (*pressure computation* phase).

We can schematically represent the structure of this time iteration by the following code excerpt:

```
call field_solver(pressure,field)
call pushing(field,x_part)
call compute_pressure(x_part,pressure)
```

Here, `pressure(1:n_cell)`, `field(1:n_cell)` and `x_part(1:n_part)` (with `n_cell`= N_{cell} and `n_part`= N_{part}) represent pressure, electromagnetic-field and particle-position arrays, respectively. In order to simplify the notation, we will refer, in the pseudo-code excerpts, to a one-dimensional case, while the real code refers to a three-dimensional (3-D) application. In implementing a parallel version of the code, according to the distributed-memory domain-decomposition strategy, different portions of the physical domain and of the corresponding spatial grid are assigned to the n_{node} different nodes, along with the particles that reside on them. This approach yields benefits and problems that are complementary to those yielded by the particle-decomposition one [11]: on the one hand, the memory resources required to each node are approximately reduced by the number of nodes (`n_part`$\sim N_{part}/n_{node}$, `n_cell`$\sim N_{cell}/n_{node}$); an almost linear scaling of the attainable physical-space resolution (i. e., the maximum size of the spatial grid) with the number of nodes is then obtained. On the other hand, inter-node communication is required to update the fields at the boundary between two different portions of the domain, as well as to transfer those particles that migrate from one domain portion to another. Such a particle migration possibly determines a severe load unbalancing of the different processes, then requiring a dynamic balancing, at the expenses of further computations and communications. Let us report here the schematic representation of the time iteration performed by each process, before giving some detail on the implementation of such procedures:

```
 call field_solver(pressure,field)
 call check_loads(i_check,n_part,n_part_left_v,
&    n_part_right_v)
 if(i_check.eq.1)then
  call load_balancing(n_part_left_v,n_part_right_v,
&     n_cell_left,n_cell_right,n_part_left,n_part_right)
  n_cell_new=n_cell+n_cell_left+n_cell_right
  if(n_cell_new.gt.n_cell)then
   allocate(field_aux(n_cell))
   field_aux=field
   deallocate(field)
   allocate(field(n_cell_new))
   field(1:n_cell)=field_aux(1:n_cell)
   deallocate(field_aux)
  endif
  n_cell=max(n_cell,n_cell_new)
  n_cell_old=n_cell
  call send_receive_cells(field,x_part,
&     n_cell_left,n_cell_right,n_part_left,n_part_right)
  if(n_cell_new.lt.n_cell_old)then
   allocate(field_aux(n_cell_old))
   field_aux=field
   deallocate(field)
   allocate(field(n_cell_new))
   field(1:n_cell_new)=field_aux(1:n_cell_new)
   deallocate(field_aux)
```

```
 endif
 n_cell=n_cell_new
 n_part=n_part+n_part_left+n_part_right
endif
call pushing(field,x_part)
call transfer_particles(x_part,n_part)
allocate(pressure(n_cell))
call compute_pressure(x_part,pressure)
call correct_pressure(pressure)
```

In order to avoid continuous reallocation of particle arrays (here represented by `x_part`) because of the particle migration from one subdomain to another, we overdimension (e.g., +20%) such arrays with respect to the initial optimal-balance size, N_{part}/n_{node}. Fluctuations of `n_part` around this optimal size are allowed within a certain band of oscillation (e.g., ±10%). This band is defined in such a way to prevent, under normal conditions, index overflows and, at the same time, to avoid excessive load unbalancing.

5.1. Restructuring the original code. Preliminary experiments deal with restructuring and management of the sequential program that solves the problem presented above. The program performs, after an inizialization phase, a number of iterations that at each run update values of fields, pressures, position and velocity of simulation particles. The original application (about 5-thousands lines of fortran code) has been provided by the ENEA[1]. It saves at the end of each iteration the intermediate results. We indentified the set of additional relevant information that need to be saved, at the end of each iteration in order to checkpoint and resume the application after a suspension. It means that before a new iteration starts, the application status has been already saved. A suspension will cause the lost of work that has performed since the beginning of the current iteration.

We restructured the original code by adding a number of functions to handle the *start*, *stop*, *suspend* and *resume* events. *start:* it calls the *init* function that reads the input data. The *init* subroutine starts also the main cycle that use many iterations to update particles data step by step.

```
integer function start()
     start=1
     call init
     call main_cycle
     end function
```

stop: it stops the execution, that will restart from the beginning of the application when the start-command will be invoked. When the application restarts input data will be read again and intermediate results will be lost.

```
integer function stop()
     stop=1
     end function
```

checkpoint: it writes in a file the intermediate information which describe the particles (pressure, position, volume ...), algorithm data such as the current iteration, the max number of iterations, and others. These data are stored in two modules, which are *global_data* and *particles_data*. The *write_data* subroutine uses the *dataout* parameter as filename.

```
integer function checkpoint(dataout,n)
          integer n
      character dataout(n)
          use global_data
          use particles_data
          checkpoint=1
          call write_data(dataout)
         end function
```

suspend: it suspends the execution that can be resumed at any time, restoring the application status. The write status saves the application status in a file as it has been explained above. If a suspension has been requested,

[1]ENEA is the Italian Agency for New Technologies, Energy and Sustainable Economic Development - *http* : *//www.fusione.enea.it*

Iteration	StartTime (sec)	Duration (sec)
1	5,864	86,414
2	92,278	87,236
3	179,514	87,574
4	267,087	87,836
5	354.923	87,657
6	442,580	87,835
7	530,414	87,792
8	618,206	87,695
9	705,900	87,658
10	793,558	87,620

FIG. 6.1. *Mean time for each iteration of plasma simulation*

the function is executed at the end of the current iteration. In this case it does not need to perform any actions.

```
integer function suspend()
        suspend=1
        end
```

*resume:*it resumes the application execution after a suspension. The *read_data* subroutine restores the application status. The *read_data* subroutine initializes the gobal varaiables of the program modules with the values stored in the *indata* file. The *start_main_cycle* starts from the beginning of the same iteration that was interrupted by the last suspension.

```
integer function resume(indata,n)
         integer n
         character indata(n)
             use global_data
             use particles_data
             resume=1
         call read_data(indata)
         call start_main_cycle
         end function
```

Migration is transparent to the programmer. It transfer the Java agent code and the *data_file*, nedded for resumption.

6. Experimental results. In order to build the native library we used the Ubuntu Linux Operative System, GNU FORTRAN 4.4.1, GCC 4.4.1 and SUN JDK 1.6. The Jade platform v. 1.3.5 has been used to develop and run agents. The *C* implementation of the native console and the FORTRAN restructured application have been compiled and linked together. Agents can download different builds, for 64 bit and 32 bit Linux platforms by a FTP server. Computing nodes are connected by a 100MB Ethernet network. Successful experiments demonstrated that the prototype works properly, in fact the platform successfully supports checkpointing, resume, migration and monitoring of native applications. Results allowed us to evaluate the overhead due to cloning and migration when it is necessary to migrate the native code which was previously compiled for the target machine, and the status of execution.

We experienced the native management of the application for the plasma simulation. The original code has been executed on a dual-cpu Intel Xeon 3.06 GHz, 512K cache size and 2GB RAM. In a first experiment an *Agent Proxy* is started from an *Agent Manager* on its same node. The *Agent Proxy* get the cpu architecture and the operative system (i386,linux) and asks for the correct version of native library. A shared library is downloaded via a public ftp server connected to the Internet by a 8MB connection in a different geographic site . The application executes 10 iterations without suspension and resume, but saving at each iteration the intermediate results of its computation in case of asynchronous requests for migration. In Table 6 we show the measures we have collected after 100 runs. Wen can see that time needed to start the application, and to read the input data is 5,86 sec. Before that the time needed to ask for the shared libraries, to get their location and download it was 1,794 (sec). In a second experiment the *Agent Manager* sends randomly a request for

migration from a node to another one of the cluster. Even if we know in advance that the nodes have the same cpu an operative system we chose to ask for and download again the shared library. Furthermore, even if the nodes share the same file system via NFS, however the agent servers and the application execute in two different directory. In the first one we have the agents code and the input data, in the second one only the intermediate results will be saved. The agent code will be transfered during migration and saved in a cache. The status of application contains intermediate values of the program variables and of the particles. The total size of information is about 95MB. To optimize the data transfer, the application compresses the data before the migration, and decompresses the file at destination. The time spent to suspend, migrate and resume the application will be:

$$T_{total} = T_{write} + T_{compress} + T_{decompress} + T_{transfer} + T_{resume} + T_{lost} \tag{6.1}$$

When migration is asked the array of particles has been already written into a file at the end of the previous iteration. T_{write} represents the time needed to write only the current values of variables necessary for resumption. $T_{compress}$ is the compression time that takes 6.2 sec, while the decompression takes 1.4 sec. The compression reduce the data size to 30 MB. In the *resume* method of the native console, overridden by the application, we call the tar utility, with the czf options to compress the application status. It could be done in the java code if such utility is not available. The *Proxy Agent*, before to move, store the compressed file in a byte array and uses the Jade APIs to migrate with its own code and attributes. $T_{transfer}$ is the time needed to transfer the agents and the application status. On the target node we only need a running agent platform. T_{resume} represents the time to restart the application that will read the intermediate values of its variable from the file system. T_{lost} is the time elapsed since the beginning of the current iteration before the migration request. It depends on when the signal interrupts the main thread. It will be less than the duration of a complete iteration. In our implementation all the work that has been done since the beginning of the current iteration is lost. In the formula we do not consider the time needed to download the native code because we could send an agent at the new destination to identify the target architecture and to download the right version of native library before and meanwhile the application is suspended. However in the following measures this strategy as not been applied and T_{total} will include also that contribution.

After a test-set of 100 runs, we observed a mean turnaround of 16m 22,611s without migration, and 17m 18,993s with a random migration. That means a mean time of T_{total} equals to 56,382 secs to be spent for migrating the execution from a node to another one of the local network. We can observe that the migration, in this case, takes about 64% of a single iteration, that is not relevant if compared to a real execution of hundreds iterations. We mean that the service can be effectively used to move the execution to a faster node, or to continue when the hosting machine should be upgraded or restarted. Furthermore many performance improvements can be designed. For example the choice to save the application status at each iteration is not the most effective one. We could plan different strategies taking into account the duration of each iteration, the time needed to suspend, migrate and resume the application, and the probability that it could be happening. In Fig. 6 it is shown the time trace for one execution of the test-set. We can see step by step what happened. In the first column we have *who takes the time*, in the second one the *item* is described and in the last one the time duration. On the left side we describe what happened on the first node (Container 1) and on the right side the second Container nodes works after the migration. The character @ represents the time elapsed since the start of the native application, otherwise the time value represents the duration of that event. In this case the suspension is received at 15,073 sec from the init of the application and T_{lost} is about 9,17 secs. As the mean iteration lasts for 88,728 secs, the other contributions of T_{total} are all relevant. We can observe that the library download takes more time at Container-2 because the Agent Proxy is running on a different node than the Agent Manager, so the communication use the cluster network. Nevertheless the application needs to resume the intermediate results before to start the first iteration, however all the needing initialization are already done. That is because the first iteration starts already at 0,35 (sec).

7. Conclusions. We presented an approach for agents based management of high performance scientific applications, providing a real case study as a proof of concept. We designed and implemented a Mobile Agents based service and its interface to POSIX applications. Programmers can extend their native applications in order to support checkpoint, migration, suspension, resuming, etc. Service implementation is in Java. It is portable and mobile. Application portability on heterogeneous node is supported through the dynamic linking of native libraries compiled for the target machine. The application life cycle can be controlled interactively from

Container-1			Container-2		
Code	Event	Time (sec)	Code	Event	Time
Proxy Agent	Library down.	1,629	- - - - -	- - - - -	- - - - -
Application	Iteration 1	@5,893 ; 0	- - - - -	- - - - -	- - - - -
Application	Suspend	@15,073	- - - - -	- - - - -	- - - - -
Agent manager	T_{total}	17,349	- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -	Proxy Agent	Library down.	2,015
- - - - -	- - - - -	- - - - -	Application	Iteration 1	@0,329 ; 89,168
- - - - -	- - - - -	- - - - -	Application	Iteration 2	@89,497 ; 89,194
- - - - -	- - - - -	- - - - -	Application	Iteration 3	@178,691 ; 89,186
- - - - -	- - - - -	- - - - -	Application	Iteration 4	@267,878 ; 89,047
- - - - -	- - - - -	- - - - -	Application	Iteration 5	@356,926 ; 88,348
- - - - -	- - - - -	- - - - -	Application	Iteration 6	@445,274 ; 88,615
- - - - -	- - - - -	- - - - -	Application	Iteration 7	@533,889 ; 88,537
- - - - -	- - - - -	- - - - -	Application	Iteration 8	@622,426 ; 88,364
- - - - -	- - - - -	- - - - -	Application	Iteration 9	@622,426 ; 88,88
- - - - -	- - - - -	- - - - -	Application	Iteration 10	@710,790 ; 87,932

FIG. 6.2. *Trace of a migrated execution*

a GUI or can be programmed by a scheduler that reacts to changes in the environment. An abstract console defines the methods which are used by agents in order to interface with the application. An implementation for POSIX application has been implemented and tested. We presented preliminary results which will be extended and discussed in the camera ready version. We Future work will deal with the management of parallel and distributed applications by cooperation of mobile agents which implement strategies for distributing the workload in order to optimize system utilization or application performance.

REFERENCES

[1] Barak A. and La'adan O., The MOSIX Multicomputer Operating System for High Performance Cluster Computing. Journal of Future Generation Computer Systems (13) 4-5, pp. 361-372, March 1998.
[2] J. Bilbao, G. Garate, A. Olozaga, A. del Portillo: Easy clustering with openMosix, World Scientific and Engineering Academy and Society (2005)
[3] R. Lottiaux, B. Boissinot, P. Gallard, G. Valle, C. Morin: openMosix, OpenSSI and Kerrighed: a comparative study, INRIA (2004)
[4] Maoz T., Barak A. and Amar L., Combining Virtual Machine Migration with Process Migration for HPC on Multi-Clusters and Grids, IEEE Cluster 2008, Tsukuba, Sept. 2008.
[5] I. Foster, N. Jennings, Kesselman, Brain Meets Brawn: Why Grid and Agents Need Each Other, in Proceeding of the 3rd Joint Conference Autonomous Agents and Multi-Agent Systems, pp. 8-15, 2004.
[6] B. Di Martino, O. F. Rana, Grid Performance and Resource Management using Mobile Agents, in: V. Getov et al. (Eds.), Performance Analysis and Grid Computing, pp. 251-264, Kluwer Academic Publishers, 2004 (ISBN 1-4020-7693-2).
[7] F. M. T. Brazier, B. J. Overeinder, M. van Steen, and N. J. E. Wijngaards, "Agent Factory: Generative Migration of Mobile Agents in Heterogeneous Environments", Proceedings of the 2002 ACM symposium on Applied computing, ISBN:1-58113-445-2, pp 101 - 106.
[8] M. Fukuda, Y. Tanaka, N. Suzuki, L. F. Bic, S. Kobayashi, "A mobile-agent-based PC grid", Autonomic Computing Workshop, ISBN: 0-7695-1983-0, pp 142- 150, June 2005.
[9] C. Peltz. Web Services Orchestration and Choreography. IEEE Computer Magazines, 2003.
[10] R. Donini, R. Aversa, B. Di Martino e S. Venticinque. Load balancing of mobile agents based applications in grid systems. In Proceedings of 8 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Rome, 23-25 June 2008. IEEE.
[11] Di Martino, B., Briguglio, S., Vlad, G., Sguazzero, P.: Parallel PIC Plasma Simulation through Particle Decomposition Techniques. Parallel Computing **27**, n. 3, (2001) 295–314.
[12] Fox, G. C., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., Walker, D.: Solving Problems on Concurrent Processors (Prentice Hall, Englewood Cliffs, New Jersey, 1988).

[13] Ferraro, R. D., Liewer, P., Decyk, V. K.: Dynamic Load Balancing for a 2D Concurrent Plasma PIC Code, J. Comput. Phys. **109**, (1993) 329–341.
[14] Cybenko, G.: Dynamic Load Balancing for Distributed Memory Multiprocessors. J. Parallel and Distributed Comput., **7**, (1989) 279–391.
[15] Birdsall, C. K., Langdon, A. B.: Plasma Physics via Computer Simulation. (McGraw-Hill, New York, 1985).

Edited by: June 21, 2010
Received: March 30, 2010
Accepted: June 21, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 161–172. http://www.scpe.org
ISSN 1895-1767

VINE TOOLKIT—TOWARDS PORTAL BASED PRODUCTION SOLUTIONS FOR SCIENTIFIC AND ENGINEERING COMMUNITIES WITH GRID-ENABLED RESOURCES SUPPORT

DAWID SZEJNFELD, PIOTR DZIUBECKI, PIOTR KOPTA, MICHAL KRYSINSKI, TOMASZ KUCZYNSKI, KRZYSZTOF KUROWSKI, BOGDAN LUDWICZAK, TOMASZ PIONTEK, DOMINIK TARNAWCZYK, MAŁGORZATA WOLNIEWICZ*, PIOTR DOMAGALSKI, JAROSLAW NABRZYSKI, AND KRZYSZTOF WITKOWSKI†

Abstract. In large scale production and scientific, academic environments, the information sets to perform computations on come from various sources. In particular, some computations may require the information obtained as a result of previous computations. Workflow description offers an attractive approach to formally deal with such complex processes. Vine Toolkit [1] solution addresses some major challenges here such as the synchronization of distributed workflows, establishing a community driven grid environment for the seamless results sharing and collaboration. In order to accomplish these goals Vine Toolkit offers integration on different layers starting from rich user interface web components embedded in portal frameworks like GridSphere [28] or Liferay [26], integration with workflow engine and grid security and ending up with a built-in meta-scheduling mechanisms, that allow IT administrators to perform load balancing automatically among computing clusters and data centers to meet peak demands. As a result of the wow2green [5] project a complete solution has been developed and delivered and still is being adopted in the pending projects like PL-Grid [4].

Key words: grid portals, web portal, workflows, grid computing, grid technology, grid, Vine toolkit, Vine, GRMS, flowify, wow2green, PL-Grid, GRIA, middleware, GridSphere, liferay

1. Introduction. Vine Toolkit (in short Vine) is a modular, extensible Java library that offers developers an easy-to-use, high-level Application Programming Interface (API) for Grid-enabling applications and more as it will be described in this article. It was developed within EU-funded projects like OMII_Europe [2] or BEinGRID [3] and polish infrastructural project PL-Grid [4]. It was used to build advanced portal solution integrating many different grid technologies within BEinGRID business experiment called wow2green (Workflows On Web2.0 for Grid Enabled infrastructures in complex Enterprises) [5]. The solution will be described to illustrate the great scope of Vine usage and its advanced features. Integration with web frameworks like Liferay [26] allows developers to build production quality web collaboration solutions. The article is an extended version of the PPAM 2009 article [27]—individual paragraphs were extended and some new added.

1.1. State of the art. Currently there are several grid portals on the market. While some of them allow only to submit and manage jobs in the dedicated system, other are application development platforms designed for building and running simulations in multiple middleware solutions. There are also environments which put focus on the workflow or collaboration capabilities. Possibility of working with multiple middleware at the same time combined with workflow support, collaboration capabilities as well as possibility of integration with multiple portal frameworks are features which distinguish Vine Toolkit amongst competitive open-source grid portal solutions.

Some of the grid portals which are related to the Vine Toolkit are described in the following subsections.

1.1.1. P-GRADE. [6]—first highly integrated parallel application development system for Grid and clusters. It uses Globus, Condor-G and MPICH-G2 as grid-aware middleware to conduct computations. To provide Grid programming interface with support for workflow and orchestration P-GRADE uses GRAPNEL language which allows a programmer to set a workflow of objects/library calls. Special tool may be used to generate MPI code from a GRAPNEL program. Supports legacy applications written in: C, C++, Fortran, MPI, PVM.

P-GRADE Grid Portal—is a web based, service rich environment for the development, execution and monitoring of workflows and workflow based parameter studies on various grid platforms. P-GRADE Portal hides low-level grid access mechanisms by high-level graphical interfaces, making even non grid expert users capable of defining and executing distributed applications on multi-institutional computing infrastructures. Workflows and workflow based parameter studies defined in the P-GRADE Portal are portable between grid platforms without learning new systems or re-engineering program code. Technology neutral interfaces and concepts of the P-GRADE Portal help users cope with the large variety of grid solutions. More than that, any

*Poznan Supercomputer and Networking Center, ({dejw, piotr.dziubecki, pkopta, mich, tomasz.kuczynski, krzysztof.kurowski, bogdanl, piontek, dominikt, gosiaw}@man.poznan.pl).
†FedStage Systems, ({piotr.domagalski, krzysztof.witkowski, jarek.nabrzyski}@fedstage.com).

P-GRADE Portal installation can access multiple grids simultaneously, which enables the easy distribution of complex applications on several platforms. Interoperability with Globus Toolkit 2, Globus Toolkit 4, LCG and gLite grid middlewares. Secured grid access mechanisms using X.509 certificates and proxy credentials. Built-in graphical editor to design and define grid workflows and grid parameter studies. Integrated workflow manager that orchestrates the fault tolerant execution of grid applications. On-line workflow and job monitoring and visualization facilities. MPI execution in Globus and gLite grid environments. Graphical MDS and BDII grid information system browser. Support to include local and remote storage files into grid applications. User level storage quota management to protect against server overloading. Legacy application publish and reuse capabilities by the GEMLCA mechanism. Workflow import-export-archive service.

License: GPL

1.1.2. myExperiment.org. [8]—is a collaborative environment where scientists can safely publish their workflows and experiment plans, share them with groups and find those of others. Workflows, other digital objects and bundles (called Packs) can now be swapped, sorted and searched like photos and videos on the Web. Unlike Facebook or MySpace, myExperiment fully understands the needs of the researcher and makes it really easy for the next generation of scientists to contribute to a pool of scientific methods, build communities and form relationships—reducing time-to-experiment, sharing expertise and avoiding reinvention. Launched in November 2007, contains the largest public collection of workflows across multiple workflow systems including Taverna (over 800 workflows) and Trident and is used by thousands of users ranging from life sciences and chemistry to social statistics and music information retrieval. Project source code is accessible and written in Ruby on Rails.

License: BSD

1.1.3. G-POD. [7]—European Space Agency Earth Observation G-POD (Grid Processing on Demand), it is a generic GRID-based operational environment where specific data handling applications can be seamlessly plugged into system. Coupled with high-performance and sizeable computing resources managed by GRID technologies, it provides the necessary flexibility for building an application virtual environment with quick accessibility to data, computing resources and results. The G-POD web portal is a flexible, secure, generic and distributed platform where the user can easily manage all its tasks. From the creation of a new task to the result publication, passing by the data selection and the job monitoring, the user goes trough a friendly and intuitive interface accessible from everywhere. The portal offers access to, and support for, science-oriented Earth Observation GRID services and applications, including access to a number of global geophysical ENVISAT products. Globus and gLite is used as main target middleware.

1.1.4. EnginFrame. [9] (EF) is a web-based front-end for job submission, tracking and integrated data management for HPC applications and other services. EnginFrame can be easily plugged on several different scheduler or grid middleware like: Platform LSF, Sun Grid Engine, PBS, gLite. Every service defined in the EF portal respects the JSR168 standard and can be exposed in any portlet container. EF is certified to be compliant with IBM WebSphere Enterprise Portal. Every service defined in the portal is automatically exposed trough a WSDL interface so it can be accessed like a web-service application and comes with a J2EE API so you can plug your java application to it. Can be linked with any authentication system (NIS, LDAP, Active Directory, MyProxy). Authorization system decides which services the user is able to see and/or use. Portal administrators can restrict the access to some applications to a defined class of users. Supports remote desktop tools like: Citrix metaFrame, IBM DCV, NoMachine's NX, VNC.

License: EnginFrame uses a proprietary license.

1.2. Problem Description. The general problem solved within wow2green business experiment is presented in this paragraph. This use case is just an illustration of how and where Vine Toolkit can be used. This article is focused mainly on Vine itself and only on the business logic layer without going into end user interface and functionality description.

Big companies consist of several units generating a variety of information sets and performing advanced computations. Workflow description offers an attractive approach to formally deal with complex processes. Unfortunately, existing workflow solutions are mostly legacy systems which are difficult to integrate in dynamically changing business and IT environments. Grid computing offers flexible mechanisms to provide resources in the on-demand fashion. Wow2green system—called officially Flowify [10] which is based on the Vine Toolkit—is a

Grid solution managing computationally intensive workflows used in advanced simulations. Each department within big enterprise or a small company in a supply chain:

- performs unsynchronized computations among departments,
- exchanges computing results independently with other project participants, so input data is re-entered in different formats, output data is lost or overwritten,
- overloads computing resources during deadlines.

Flowify helps users to synchronize the data processing for execution and collection of results in an automated fashion. Easy-to-use Flowify workflow interfaces are provided as intuitive Web applications or integrated with commonly used tools. Thus many users can simultaneously form dynamic workspaces, share not only data for their computing experiments but also manage the whole process of advanced computations. Built-in GRMS meta-scheduling mechanisms provided by Flowify allows IT administrators to perform load balancing automatically among computing clusters and data centers to meet peak demands.

Similar scenarios are present also in scientific, academic environments where people use scientific applications (often open-source) to conduct computations and analyze big sets of data. Often the relations between persons involved are less formal and there is no such restrictive bi-directional relations between members of the team working on the given problem. Vine with a set of web applications is an excellent solution to prepare web portal environment for advanced scientific and engineering applications with grid-enabled resources in the backend. Moreover, the heterogeneity of resources brings about an opportunity to integrate different sets of geographically-scattered grid resources. Integration scenario could be tightened by applying meta-scheduler for different sets of grid resources and middleware systems and make it the main entry to the grid. Component-based architecture of the GRMS meta-scheduler allows administrators to apply scheduling policies and brokering functionality across organization boundaries.

1.3. Main Requirements and Demands. The following main requirements have been identified within wow2green experiment:

- *enable workflow executions via the wow2green portal* (portal based solution should provide a suitable, user-friendly interface for workflow executions using Web 2.0 style easy-to-use web interfaces. Users should have a fully transparent access to underlying computing resources so they can take advantage of high level workflow management)
- *provide detailed information and feedback for users about their computing workflow simulations* (every workflow is a kind of template for creating a computing simulation which could be later executed on different computing resources. External workflow engine was selected—Kepler [11] which supports provenance mechanisms required to search and discover templates in created workflows. These features are also available via the wow2green portal)
- *enable more efficient platform usage through applying brokering of jobs defined in workflow nodes* (GRMS [12] broker is used as a meta-scheduling and brokering component for load balancing among computing resources. Thus, end users will not have to select computing resources manually and the overall utilization of computing resources will be improved), end user is freed from resource selection decision every time something is computed what is troublesome in large infrastructures
- *easy-to-use stage in and stage out simulation data* (users are able to upload and load simulation data and workflows to and from our portal in an intuitive way), this requirement is quite generic and appears in many other scenarios—users wants easier data management what could be especially hard to master in a large and heterogeneous infrastructures—portal solution makes it easier, users may overcome many troubles like firewall issues for example
- *easy-to-use, Kepler-like graphical user interfaces for workflow management and control* (to provide user friendly web interfaces we selected a new web technology called Adobe Flex. It will enable us to implement various web applications for easy management and control of workflows via portal)
- *support for new workflow definitions* (additional web tools for workflows will be developed to enable users to create, share and modify workflows. Users (workflow creators) will be able to share created workflows with other users interested only in performing simulations of existing workflows).
- *provide a repository of workflow definitions*

2. Flowify Architecture. The overall Flowify architecture is presented below, all components are described to show its function and reveal the complexity of the prepared environment (Fig.2.1).

Identified components:

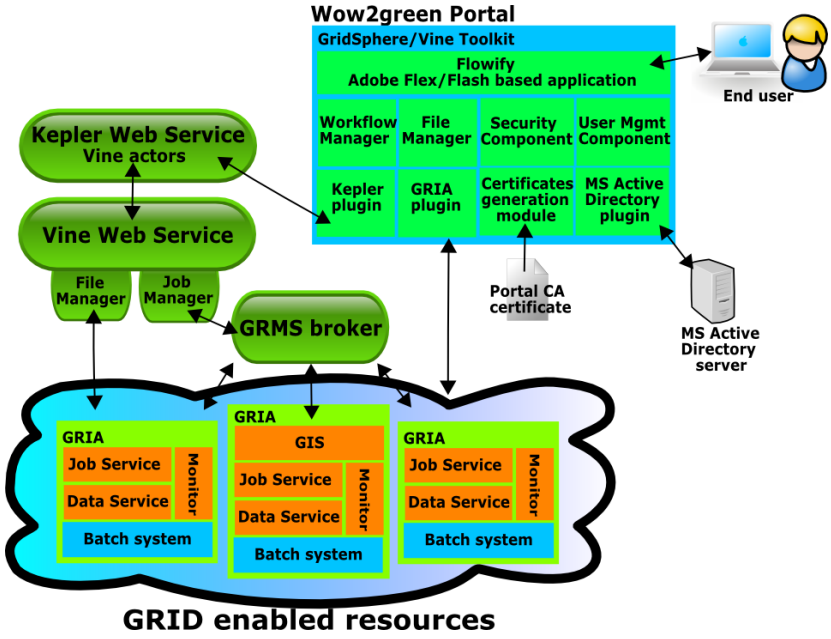


FIG. 2.1. *Wow2greem system (Flowify) overall architecture.*

- *Gridsphere portal framework*—portal framework that provides an open-source portlet based Web portal. GridSphere enables developers to quickly develop and package third-party portlet web applications that can be run and administered within the GridSphere portlet container. The wow2green portal is based on this framework, production-ready portal for a larger community could be deployed on Liferay
- *Vine Toolkit*—used as business model layer within the portal, applications created for the scenario uses the Vine mechanisms to interact with grid services and workflow engine, integrated Adobe Flex with BlazeDS technology allows creating advanced web applications. Vine Toolkit components used in portal:
  - *Workflow Manager*—general component used to interact with different workflow management solutions through common API; in case of the wow2green solution kepler plugin was introduced
  - *Role Manager*—role management component which allows administrators to manage role-based access to GUI elements and accessible actions and also to define role-based access policies regarding data managed through logical file manager component
  - *File Manager*—File Manager component which allows for interacting with different data management systems by means of common API
  - *Portal Security Component*—used for portal user certificates and proxies management, together with the Certificates generation module allows for automatic certificate generation for registered portal users
  - *Portal User Management*—used for portal users management, user registration in the portal; together with the MS Active Directory plugin it enables using existing users in the given organisation to make the integration easier with the existing IT environment (it is also possible to use some existing LDAP service it desired)
- *Vine web service*—web service which exposes a subset of functionality of the Vine Toolkit component to external systems/services—regarding Job and File Manager, the web service is used by Kepler actors to interact with grid enabled resources
- *Kepler workflow engine*—core part of the Kepler/Ptolemy standalone application/framework for workflow management, it is used in the wow2green solution as primary workflow management engine, kepler workflow description called MOML is used within the system
- *Kepler engine web service*—the web service exposes the workflow engine functionality to external systems/services, portal workflow manager component uses it as the workflow management service
- *Set of kepler's vine actors*—base components of the workflows' definitions to submit jobs and manage data through the Vine Toolkit service layer

- *GRMS broker with web service interface*—job broker component with advanced scheduling technology, scheduling policies could be easily exchanged dependently on the requirements, for the wow2green experiment version 2.0 was deployed
- *GRIA middleware [13]*—grid middleware—service-oriented infrastructure (SOI) designed to support B2B collaborations through service provision across organizational boundaries. Following services are used in the wow2green system: Job Service job management, Data Service data management, Membership Service to manage groups of users relevant to the roles defined in the portal
- *Griddle Information Service*—web service which exposes the information about available GRIA resources and batch queues statistics

In case of new scenarios for scientific communities, GRMS 3.0 is used, bringing about new possibilities like support for cross-cluster MPI application support. Such functionality was successfully applied during the QosCosGrid [22] project and could be useful with some scientific applications applied on many clusters. The new version of broker also introduces built-in support for workflows, which makes the whole architecture simpler and more robust rendering the workflow management much consistent and powerful. More advanced scheduling policies are possible and job-related data management is much more consistent and better operated. The workflow definition is also simplified which makes it more suitable for a subset of scenarios and enables faster implementation for the new scientific applications.

3. Vine Toolkit portal based solution. Vine Toolkit (Fig. 3.1) is a modular, extensible Java library that offers developers an easy-to-use, high-level Application Programming Interface (API) for Grid-enabling applications. Vine can be deployed for use in desktop, Java Web Start, Java Servlet 2.3 and Java Portlet 1.0 environments with ease. Additionally, Vine Toolkit supports a wide array of middleware and third-party services. Using Vine Toolkit, one composes applications as collections of resources and services for utilizing those resources (basic idea in Vine—generic resource model—any service and data source can me modeled as an abstract entity called resource and can be integrated with web applications using high-level APIs).

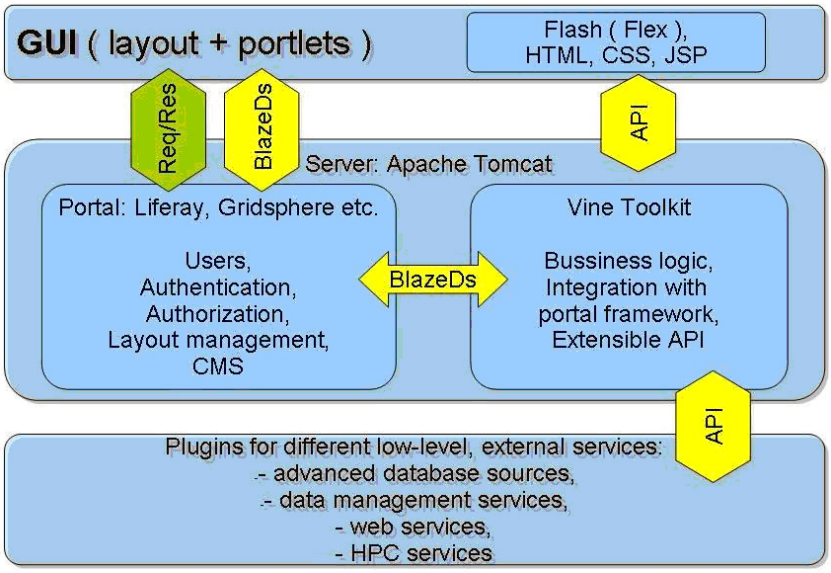


FIG. 3.1. *Vine Toolkit high level architecture in portal deploy scenario.*

The Vine Toolkit makes it possible to organize resources into a hierarchy of domains to represent one or more virtual organizations (VOs). Vine offers security mechanisms for authenticating end-users and authorizing their use of resources within a given domain. Other core features include an extensible model for executing tasks (every action is persisted as Task) and transparent support for persisting information about resources and tasks with in-memory or external relational databases. Adobe Flex [14] and BlazeDS [15] technology allows for creating advanced and sophisticated web applications similar to many stand-alone GUIs. Other GUI technologies like GWT, HTML, CSS and Javascript with Ajax could be used instead where Vine business logic layer could be used below to interact with different resources. Vine comes with many co-bundled components.

There is a set of bundled components. User / Role / Application managers—administrative tools for user management, their roles, application related right management for accessible methods of web applications, user login and user registration—it is module-based so the process could be extended and span any external service. like ldap or ms active directory and portal incorporates authenticated users automatically. Resource manager where the configuration of resources is accessible, could be displayed and changed by advanced users if desired. File browser component allows users to manage data on different data management servers and also portal file system (PFS)—Vine provides such components to allow storing of users' data in the portal. Job Manager component based on JSDL specification allows users to prepare JSDL based job description and submits jobs. Jobs are monitored and outputs could be retrieved later. Basic information about the job is also displayed. Different job managers are used here according to the installed and configured plugins. Credential manager allows for getting proxy credentials from MyProxy server configured for the portal. User is able also to add some mapping to his accounts for the accessible DN—so later it is possible to log in using MyProxy user account—single sign-on is possible then, proxy certificates are loaded automatically and user can use grid services directly. Resource browser is able to display information about grid resources—globus MDS services are used here—it is also possible to show dynamic values such as free memory.

3.1. Workflow Engine Support. Vine Toolkit API has been enriched with the generic Workflow Engine API. It was created as a separate subproject which could be added as a feature during the portal installation. API reflects identified requirements regarding functionality used within end user web applications to manage users' workflows through the independent API. The following main functionality was included in the workflow submission interface: *startWorkflow*, *stopWorkflow*, *pauseWorkflow*, *resumeWorkflow*, *terminateWorkflow*, *getWorkflowOutputs*, *getWorkflowStatusById*, *createWorkflowSpec* and beside this a set of support methods like *getWorkflowByTaskId* to retrieve workflow instance from Vine using its task id. It allows quite simply add support for different workflow engines if desired or required by some applications constraints. For the wow2green experiment Kepler engine was used as the primary workflow engine. The base API allows define so called *WorkflowDefinition* which represents the workflow definition prepared as a XML string. In the case of Kepler engine MoML [16] based workflow description is used. Through the use of generic *WorkflowDefinitionParameter* a set of input parameters could be passed along with the workflow definition to the target workflow engine plugin. *WorkflowManager* component plays role of the generic workflow manager while the concrete instance is created during workflow submission to the service deployed on the target host. Resources defined in the Vine domain description (it is a resource registry in the given domain). Vine contains Kepler plugin project which encapsulates the Kepler engine client to implement the generic Vine Workflow API. The figure 2.1 clearly shows the connection from the portal to the *Kepler Web Service* it is another component which exposes the functionality of the Kepler workflow engine to the external clients acting as a workflow management service. The key point here are so called the *Vine actors*—small java classes which are in fact *Vine Web Service* clients. These "bricks" are used by the workflow designer to prepare workflow description to execute applications using grid resources and to acquire results of the computations. User is not aware where the computations take place—*Vine Web Service* by using grid services like GRMS which meta-scheduler and job broker—dispatches the jobs on the accessible grid environment. Vine is used here in different separate role in servlet mode to be a backed of the *Vine Web Service* and provide ease and unified access to different grid services like in this case GRMS and GRIA. Such construction allows easily switch to different grid technology and reconfiguration of the whole environment to different middleware stack like Unicore 6 [17]. In case of new application scenarios and new GRMS 3.0 it is possible to use the broker directly regarding application workflows. New GRMS broker supports DAG-based workflows, which are enough for most of the potential applications and possible dependencies between them. New project added to the Vine allows send XML-based GRMS workflows to the broker service. It gives a great opportunity for many scientists to connect their computing clusters on-demand to share computing resources and run large-scale simulations reducing the execution time or dealing with much bigger problem instances. Two parallel programming environments like OpenMPI and ProActive cover a wide range of programming languages, including Fortran, C/C++, Python (Open MPI) or Java (ProActive), thus can be easily integrated with legacy code or naturally used as a communication layer among parallel processes and threads. So by using GRMS workflow description with massive parallel applications could bring new possible scenarios and even reduce potential costs of extending of the existing infrastructure. Resources could be shared to run big parallel applications between different organizations.

3.2. Uniform interface to HPC infrastructures. In order to provide end-users with transparent access to resources, we developed a mechanism responsible for the management of uniform interfaces to diverse Grid middleware. Advanced classloader management mechanism enables dynamic loading of components that provide access to the functionality of specific Grid middleware through a uniform interface. These components are called plug-ins in this context (and are realized as Vine subprojects with separate set of client jars). These uniform interfaces are provided for such functionality like job submission, control, monitoring, data management, role management, user registration and authentication. They are based on standards where possible (e.g. JSDL for job submission) and take into account both the gathered requirements and the functionality provided by Grid middleware deployed in HPC centers. The provided interface could be extended if needed of course. Such flexibility is required as we use different workflow or single job xml descriptions regarding different workflow management services like Kepler or GRMS. By extending *WorkflowDefinition* or *JobSpec* interfaces, developer is able to add support for other workflow or job descriptions. Thus many workflow or job management services could work at the same time hiding complexity of the grid infrastructure by web application unified GUI interfaces.

3.2.1. Job Submission, Control and Monitoring (JSMC) Interface. One of the interfaces which makes use of the mechanism described above is the uniform interface for job submission functionality. It was based, firstly, on an analysis of the functionality of middleware systems and, secondly, on the Job Specification Description Language (JSDL) [18] created by the JSDL working group at the Open Grid Forum [19]. JSDL specifies different information required for successful application execution. One of the sub elements is application element which allows the user to specify the application to be executed on the Grid and the various application parameters and arguments, including input and output files. Requirements element allows the user to specify different requirements needed for the execution of the job, such as amount of memory, number of CPUs, etc. Data element allows the specification of input and output staging, in order for the user to optionally define which files should be copied to and / or from the working directory on the host where the job will be executed before job submission and / or after job completion respectively. The following main functionality was included in the job submission interface: *startJob*, *stopJob*, *getJobOutputs*, *getJobStatusById*, *createJobSpec* and beside this a bunch of support methods like *getJobByTaskId* to retrieve job instance from Vine using its task id. Currently Vine supports such middleware like gLite 3.x [20] WmProxy and CREAM, Unicore 6 UnicoreX job submission, Globus GT4 [21] WsGram, GRIA Job Service and recently QosCosGrid SMOA Computing and GRMS. Vine's plugin management allows for simultaneous usage of different middleware technologies what enables uniform access to the different HPC resources. In case of wow2grenn use case GRIA and GRMS Plugins were used to access grid services. Another motivation beside seamless usage of different grid technologies was ease of configuration and possibility to switch on to different mieddleware stack without modifications to the developed web applications (Vine plays role of the abstract access layer with well defined interface). JSMC service persists the information about all executed jobs along with all required details and the submitted specification. Monitoring data consists of job state history and basic information like start, finish time for example.

3.2.2. Data Management Interface. Another interface is the uniform interface for data management functionality. It was based, first, on an analysis of the functionality of storage services of middleware systems trying to find common API. The following main functionality was included in the data management interface: *getHomeDirectory*, *createInputStream*, *createOutputStream*, *info*, *exists*, *list*, *goUpDirectory*, *changeDirectory*, *makeDirectory*, *copy*, *rename*, *move*, *delte*, *upload*, *download*, *createNativeURLString* and beside this a bunch of support methods like *getDefaultFileManager* to retrieve default data manager instance from Vine. Currently Vine supports such middleware like gLite 3.x SRM and LFC, Globus GridFTP, Unicore 6 Storage Management System (SMS), WebDAV, Storage Resource Broker (SRB), GRIA Data Service, HTTP. Vine's plugin management allows to simultaneous usage of different storage technologies and copying data between them. In case of wow2green use case GRIA Data Service and HTTP Data Service plugins were used to access grid services. By using Vine abstract layer the web applications could be written without taking into account different technological nuances. Vine also allows use different storage servers as uniform services making data management quite easy for the application developer. Vine is also shipped with the built-in Portal File System (PFS) which allows store portal end users files directly in the disk storage managed by the portal container. It is very convenient for developers to expose data by using disposable links and store users configuration files and others close to the portal instance. PFS is also used while downloading files from remote data storage systems. In conjunction with security related mechanisms and components Vine simplifies usage of different data management systems

not only by unifying the API but also by covering the whole complexity related to the security issues. Vine also defines logical data management API which is similar to the physical one described earlier but has also support for meta data attributes, permissions related to files, multi-criteria file search related to meta data attributes, tags support. The logical data management API is used to organize users data and describe them with meta data attributes and tags. During the wow2green experiment all experiments results were stored and tagged in the logical file system to facilitate searching for finished experiments and concrete results.

3.3. Security Issues. Vine can solve different security issues while accessing different grid services. A significant interoperability problem results from the differences in the security models of systems such as UNICORE or GRIA (where authentication is based on standard X.509 certificates) and Globus-based Grids (authentication based on GSI [23]). The main difference between these systems is that basic versions of UNICORE and GRIA do not support dynamic delegation while GSI-compliant systems do. Currently, to overcome this problem, we allow a portal to authenticate on behalf of end-users. To do this, the portal signs a job with its own certificate and adds the distinguished name of the end-user before submitting the job to, for example, a GRIA site. In case of Unicore Vine supports GSI-enabled extension for Unicore gateway what enables to authenticate users by using proxy certificates. In case of gLite3 middleware voms proxy certificate are created and used—Vine can be configured to support different voms servers and allow users interactions from different VO without any problem. Vine support MyProxy [24] server to provide true single sign-on—during the user logging in to the portal proxy certificate is retrieved and used while accessing different middleware services. In case of the wow2gren solution the GRIA middleware was used. So as mentioned before portal common certificate was used to submit jobs (GSI is not supported). To distinguish users so called GRIA policy rules are used to set job and data file owners by using users' public certificates and their distinguished names. One of the requirements of the experiment was to simplify the security management issues. So user certificates are maintained by the Vine-based portal integrated with Certificate Authority—when user is registered for the first time, their certificate is generated and hold in the safe place on the server. Later proxy certificates are used to interact with the grid middleware. Common portal certificate is used to submit jobs to enable GRMS broker to manage them on behalf of users without the need to pass the private user's key certificate. Vine supports all aspects of security-related issues and processes. Account creation allows automated, user-friendly user registration at the portal and to underlying Grid middleware and third-party services used through the portal (registration modules are used to register new user accounts in external services). User authentication allows the user to login into the portal and be authenticated, login portlet is shipped together with Vine and could be embedded in different portal frameworks. Single-Sign-On (SSO) management allows automated SSO user authentication in various Grid middleware and services used through the portal. The user must have been previously registered in these external services. Authorization allows the administrator to access third-party authorization systems to change user permissions.

3.4. User Management. Common problem to solve while using different middleware infrastructures is user management and account provision. Vine is able to register users in different middleware infrastructures like gLite3, Unicore 6 or Globus GT4. Vine contains *registration modules* which allow register users on different resources. Currently the following are provided: *GridsphereRegistrationResource*—allows adding new users to Gridsphere portal, *LiferayRegistrationResource*—allows adding new users to Liferay portal, *GssCertificateRegistrationResource*—creates x509 certificate and key pairs for new users, *Gt4RegistrationResource*—creates user accounts on Globus Toolkit 4 host machine, *GriaRegistrationResource*—creates user accounts in Gria 5.3 Trade Account Service, *Unicore6RegistrationResource*—creates user accounts on Unicore 6 XUUDB, *DmsRegistrationResource*—registers users on Data Management Service, *VomsRegistrationResource*—registers users to Virtual Organization Membership Service. Modular architecture and plugin management system allows easily plugin other registration modules if desired and extend Vine with new functionality. In case of wow2green use case *GssCertificateRegistrationResource*, *GridsphereRegistrationResource* and *DmsRegistrationResource* were used to generate users' certificates, create portal accounts and logical file system DMS [25]. Vine also comes with bundled administrative web applications regarding user managamnet. One of them is *UserRegistrationApp*, which allows create requests for new accounts in Vine. Another is *UserManagementApp* where administrator has got option for deactivate user account and edit user's profile information.

3.5. Role Management. Vine also introduces role management mechanism to authorize users while taking different actions in the web applications. Roles are used in two scenarios—as roles at the portal level

and middleware stack level. Roles at the portal level allow administrators to set permissions to access different functionality in the web applications based on Vine. Administrator is able to set permissions for roles to different part of application and distinguish users regarding their possible scope of actions. Roles at the middleware stack are used to set access permissions regarding jobs and data files at the storage services. For the wow2green use case *GriaRoleResource*, *DmsRoleResource* were implemented. Thus the roles created in the portal are propagated to the middleware stack—in this case of GRIA. The end user interface allows share data for the selected roles. Such action is propagated to the middleware resources and could be used later while accessing remote storage by using given role name. Such approach makes permission management much simpler and does not multiply remote policies for every user of the system—it is enough to assign policy for some role and assign an user to the given role at the target system. Of course it is applicable only on middleware system which supports roles management like GRIA or DMS logical file system.

4. Towards production quality portals. Vine integrates with different portal frameworks and proper one should be used to provide production quality of the final solution.

FIG. 4.1. *Vine Toolkit integrated with Liferay portal framework.*

4.1. Liferay as a primary solution for the production quality portal. Vine was initially designed to work with Gridsphere portlet container. The reason for that was Gridsphere's open source nature and its target community—mainly scientists dealing with High Performance Computing problems. It was good starting point but after a few years it became obvious that concurrent solutions need to be taken into account as well. After analysis and evaluation of possible options a decision has been made to create a Liferay integration thread. Liferay is a modern approach to create complex web portals. It supports many standards for content presentation and management. It is integrated with various application containers, database systems and has modular structure with well designed API allowing creation of custom extensions and integration with third party software.

4.2. Vine Toolkit integration with Liferay. Vine is designed with its modular structure in mind. It has several places—hooks providing easy adaptation to the external environment. Integration with the portal acting as a portlet container is conducted on several levels. The most important are:

4.2.1. Build system.

- Flex sources compilation for the Liferay context path. Every flex component needs to establish communication with BlazeDS backend server. As for that a context path must be compiled into the specific component. Vine's build system takes care of that task and simply prepares a version of component specific to the destination portal environment.
- New deployment routines including modification of portal configuration. Vine Toolkit deployed in the application container acts as complex web application. Mainly due to its advanced class loaders preventing from the conflict of jars it has to be deployed in a specific way. Vine's build system is responsible for the proper jar distribution, replacing specific tokens in the deployment descriptors, altering portal layouts etc.
- New installers designed for Liferay portal. To shorten the learning curve for new users, new installers have been introduced. They are general examples of proper configuration both Vine Toolkit along with its basic services and Liferay environment. Thanks to that user is able to startup and test new portal without any advanced configuration actions.

4.2.2. Integration with Liferay services.

- Authentication/password policies. Integration with the parent authentication and password policies is a crucial task to provide synchronization of applications' state between portal and Vine. Appropriate plugins have been developed specifically for Liferay which provide seamless authentication of user against Liferay and Vine.
- User management. Vine has its own user registry to enable Vine's deployment to various destination environments. In the portal integration scenario Vine has to integrate and synchronize with the existing portal users. To accomplish that *RegistrationModule* plugins are used. *LiferayRegistrationModule* plugin is provided for adding/editing/deleting portal users.
- Applications management. Vine components are accessible as portlet applications from the Liferay perspective. They are fully manageable with no difference to the ordinary portlets. On top of that it is possible to put more control over Vine applications via Vine Toolkit administration interfaces. They provide advanced management actions for all Vine components, i. e. enabling/disabling certain features in specific applications, granting/revoking special permissions to them etc.
- JSR 168 compliant renderer for Vine components within Liferay portlets. Vine uses portlet component to render Flex components. In that way it preserves internal components integrity and make Vine applications independent from the external portal. As an output a html wrapper code is generated with a reference to the Vine Flex object inside.

4.3. Future plans for enhancing user experience using portal.

- Introducing web messaging based on BlazeDs/Flex technologies to greatly improve applications functionality.

Currently the primary way of communication in Vine GUI is remoting based on BlazeDs/Flex implementation. It provides efficient and responsive link between the client and server side. In Vine Toolkit the Model View Controller design pattern has been implemented enabling reuse of business logic with different presentation layers and leaving a place for future GUI mechanisms evolution. The next enhancement will be adding web messaging as an alternate way of communication between the server and client side. It enables server to trigger notification of state's change directly to the registered client interfaces. It will enable developers to design new components, for instance, a status monitoring applications with great ease. Until now they had to implement a querying mechanism on the client side to ask for the updated model constantly. With the web messaging a business logic on the server side will trigger notifications only if its state has changed. That will result in more efficient and sophisticated applications delivered to end users.

- Integration with Liferay Collaboration Suite package.

Liferay Collaboration Suite is a set of web applications, collaboration tools that come boundled with Liferay portal framework. The set of applications consists of blogs, message boards, instant messenger, shared calendar, address book, mail client, RSS client, Wikis, meta-tagging support. The main interest here is to idea of sharing experiments results form executed jobs and workflows by the Vine management engine and files stored in a portal file system with other portal users. Integration will be achieved by development of dedicated plugins which will communicate with Liferay API. This approach will enable all applications using File Manager with

seamless integration with Liferay Collaboration Suite with respect to security policies expressed by Liferay. First stage of integration, includes sharing data with wikis and message boards.

5. Next step—nanotechnology support portal. Currently our main focus is on the web application level scenarios and domain specific scientific and engineering applications. The idea is to prepare portal-based solution for the nanotechnology community to take advantage from nanotechnology related scientific application ran in a large Polish grid infrastructure under the Pl-Grid project umbrella. Initial plan is to start with the support for open source nanotechnology applications like Abinit and Fireball. The basic functionality foreseen is to simplify application every day usage by applying graphical interface, identify possible scientific use cases and propose different GUI layout. Another important thing is to prepare workflow definitions where the raw results will be transformed and prepared for further analysis—automation of many steps usually done manually will speed up work and allow to focus on the given problem. Advanced problems also require parameter sweep parameters what is especially useful in case of preparing big set of potential experiments varying only by some parameter value. The intermediate results from the given experiment and also from parameter swep steps will be used to prepare and show graphical plot of different physical values almost in a real time. End user will be able to control current execution and make possible decision to break the process for example. GMRS v3.0 meta-scheduler and broker plays here significant role as it supports workflows, parameter sweep and cross cluster job execution. Advanced data management features make possible to extract intermediate data during job execution. Extracted data will be used not only to prepare 2D plot display but also to prepare 3D visualization of the molecules structures. 3D visualization thread was started to prepare web based with latest web standards like HTML5 and WebGL. Portal solution will bring up the desktop application and strong collaboration environment for the whole domain specific community allowing to interact between geographically separated scientists.

6. Conclusions. In this paper we presented Vine Toolkit as universal framework to build complete web solutions to interact with different services and technologies (among others different grid middleware stacks) by means of uniform, easy API. Integration with production quality portal frameworks like Liferay opens new opportunities for creating production-ready portal environments directed for large communities. The generic architecture of the Vine Toolkit was presented. We also included some requirements and initial ideas for the wow2green business experiment as illustration of Vine application in a real, practical use case. Vine allows submit jobs to different middleware stacks through a uniform user interface. In similar way uniform API for workflow management, data management, user authentication, role and user management are provided. Resource based model allows straightforward service support and configuration by uniform way regardless of the target technology. Security issues which always occur while accessing different middleware stacks are solved by using Vine registration, authorization modules and proxy certificates support. Role management mechanism allows administrators and end users to easily manage permissions in the applications and easy grant permissions to data files on the storage server if applicable. Beside simple jobs, whole workflows could be managed by the uniform workflow interface. Currently only the Kepler engine is supported but it could be easily extended by adding another workflow engine plugin. Vine Toolkit could be used as framework for building advanced web applications for example with use of Adobe Flex technology as well as business logic provider for advanced web service solutions. Prospective application scenario related with nanotechnology science will be beneficial for the scientists who want to focus on the concrete problems and large grid infrastructure exploitation. The on-going project will enrich our experience regarding generic scientific application support.

Acknowledgments. We are pleased to acknowledge support from the EU funded OMII-Europe and BEinGRID projects and polish infrastructural project PL-Grid.

REFERENCES

[1] Vine Toolkit web site `http://vinetoolkit.org/`
[2] OMII-Europe project `http://www.omii-europe.com/`
[3] BEinGRID project `http://www.beingrid.eu/`
[4] PL-Grid project `http://www.plgrid.pl/en`
[5] Wow2green business experiment `http://www.beingrid.eu/wow2green.html`, `http://www.it-tude.com/wow2green_sol.html`
[6] P-GRADE `http://www.p-grade.hu/`
[7] G-POD `http://gpod.eo.esa.int/index.asp`
[8] myExperiment.org `http://www.myexperiment.org/`, `http://wiki.myexperiment.org/index.php/Main_Page`

[9] EnginFrame http://www.enginframe.com/
[10] Flowify http://www.flowify.org/
[11] Kepler https://kepler-project.org/
[12] GRMS http://www.gridge.org/content/view/18/99/
[13] GRIA http://www.gria.org/about-gria/overview
[14] Adobe Flex http://www.adobe.com/products/flex/
[15] Adobe BlazeDS http://opensource.adobe.com/wiki/display/blazeds/BlazeDS/
[16] MoML language specification http://ptolemy.eecs.berkeley.edu/publications/papers/00/moml/moml_erl_memo.pdf
[17] Unicore 6 http://www.unicore.eu/
[18] JSDL http://www.ogf.org/documents/GFD.56.pdf
[19] Open Grid Forum (OGF) http://www.gridforum.org/
[20] gLite 3.x http://glite.web.cern.ch/glite/default.asp
[21] Globus GT4 http://www.globus.org/toolkit/
[22] QosCosGrid http://node2.qoscosgrid.man.poznan.pl/gridsphere/gridsphere
[23] Grid Security Infrastructure (GSI) http://www.globus.org/security/overview.html
[24] MyProxy http://grid.ncsa.illinois.edu/myproxy/
[25] Data Management System (DMS) http://dms.progress.psnc.pl/
[26] Liferay http://www.liferay.com/
[27] Dawid Szejnfeld, Piotr Domagalski, Piotr Dziubecki, Piotr Kopta, Michal Krysinski, Tomasz Kuczynski, Krzysztof Kurowski, Bogdan Ludwiczak, Jaroslaw Nabrzyski, Tomasz Piontek, Dominik Tarnawczyk, Krzysztof Witkowski, Malgorzata Wolniewicz, *Vine Toolkit—Grid-enabled Portal Solution for Community Driven Computing Workflows with Meta-scheduling Capabilities*, PPAM 2009.
[28] GridSphere http://www.gridsphere.org/

Edited by: Marcin Paprzycki
Received: April 10, 2010
Accepted: June 21, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 173–188. http://www.scpe.org
ISSN 1895-1767

FAST MULTI-OBJECTIVE RESCHEDULING OF WORKFLOWS TO CONSTRAINED RESOURCES USING HEURISTICS AND MEMETIC EVOLUTION

WILFRIED JAKOB, FLORIAN MÖSER, ALEXANDER QUINTE, KARL-UWE STUCKY AND WOLFGANG SÜß*

Abstract. Scheduling of jobs organized in workflows to a computational grid is a permanent process due to the dynamic nature of the grid and the frequent arrival of new jobs. Thus, a permanent rescheduling of already planned and new jobs must be performed. This paper will continue and extend previous work, which focused on the tuning of our **G**lobal **O**ptimising **R**esource **B**roker and **A**llocator GORBA in a static planning environment. A formal definition of the scheduling problem and a classification will be given. New heuristics for rescheduling based on the "old plan" will be introduced and it will be investigated how they contribute to the overall planning process. As an extension to the work published in [22] a simple local search is added to the basic Evolutionary Algorithm (EA) of GORBA and it is examined, whether and how the resulting Memetic Algorithm improves the results within the limited time frame of three minutes available for planning. Furthermore, the maximal possible load, which can be handled within the given planning time, will be examined for a grid of growing size of up to 7000 grid jobs and 700 resources.

Key words: scheduling, multi-objective optimisation, evolutionary algorithms, memetic algorithms, benchmarks, heuristics, computational grid, restricted resources

1. Introduction. A computational grid can be regarded as a virtualised and distributed computing centre [11]. Users describe their *application jobs*, consisting of one or more elementary *grid jobs*, by workflows, each of which may be regarded a directed acyclic graph defining precedence rules between the grid jobs. The users state which resources like software, data storage, or computing power are needed to fulfil their grid jobs. Resources may need other ones. A software tool, for instance, may require a certain operating system and appropriate computer hardware to run on. This leads to the concept of co-allocation of resources. Furthermore, users will give due dates, cost budgets and may express a preference for cheap or fast execution [20]. For planning, execution times of the grid jobs are needed. In case of entirely new jobs, this can be done by estimations or by the use of prediction systems only. Otherwise, values coming from experience can be used. The grid middleware is expected to support this by providing runtimes and adding them to the workflow for further usage. According to the policy of their owners, resources are offered at different costs depending on e.g. time of day or day of the week and their usage may be restricted to certain times. In addition, heterogeneous resources usually differ in performance as well as cost-performance ratios.

To fulfil the different needs of resource users and providers, the following four objectives are considered: *completion time* and *costs* of each application job measured as fulfilment of user-given limits and averaged, and to meet the demands of resource providers, the *total makespan* of all application jobs and the ratio of *resource utilisation*. Some of these criteria like costs and time are obviously conflicting.

As grid jobs are assumed to require computing time in the magnitude of several minutes at the minimum, a certain but limited time frame for planning is available. A time limit of three minutes was regarded reasonable for planning. All grid jobs, which will be started within this time slot according to the old schedule, are regarded *fixed jobs* and will not become subject of rescheduling.

This paper extends work on rescheduling published at the PPAM 2009 conference [22] by adding two left out benchmark scenarios for the investigation of increasing workload. Furthermore, a simple heuristic search mechanism is introduced and incorporated in the basic EA of GORBA in such a way that it improves generated offspring. Thus, the EA is turned into a Memetic Algorithm (MA), an algorithm class, which has already proven its superiority to pure EA in various domains [25, 23, 15, 16, 29, 2]. The investigation presented in [22] is enhanced in this paper by the assessment of the new Memetic Algorithm and its comparison to the EA results. The main problem of creating a Memetic Algorithm is the definition of suited local searchers for the task on hand. We will come back to this in §3.

In §2 a formal definition of the problem, a classification, and a comparison with other scheduling tasks will be given. The section ends with a discussion of related work. Section 3 will describe the used algorithms, especially the new heuristics and the new local searcher, and give a summary of the work carried out so far. The results of the experiments for assessing the effect of the new rescheduling heuristics will be presented in §4, which will also report about first investigations regarding the maximum possible load for a grid, the size of

*Karlsruhe Institute of Technology (KIT), Institute for Applied Computer Science, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany, {wilfried.jakob, florian.moeser, alexander.quinte, uwe.stucky, wolfgang.suess}@kit.edu

which is growing proportionally to the amount of grid jobs. In §5 a summary and outlook to future research will be provided.

2. Problem Definition and Classification. A notation common to scheduling literature [6, 7] is used to facilitate comparisons with other scheduling problems. Given are a set $M = \{M_1, \ldots, M_m\}$ of resources, a set $J = \{J_1, \ldots, J_l\}$ of application jobs, and a set O of grid jobs. The n_i grid jobs of application job J_i are denoted $O_{i1}, \ldots, O_{in_i}$. The following functions are given:

(i) a precedence function $p : O \times O \to \{TRUE, FALSE\}$ for the grid jobs

(ii) an assignment function $\mu : O \to \mathcal{P}(\mathcal{P}(M))$ from grid jobs to resource sets. $\mathcal{P}(M)$ is the power set of M. μ_{ij} is the set of all possible combinations of resources from M, which together are able to perform the grid job O_{ij}

(iii) a function $t : O \times \mathcal{P}(M) \to \mathbb{R}$, which gives for every grid job O_{ij} the time needed for the processing on a resource set $R_{ij} \in \mu_{ij}$

(iv) a cost function, $c : \mathbb{R} \times \mathcal{P}(M) \to \mathbb{R}$, which gives for every time $z \in \mathbb{R}$ the costs per time unit of the given resource set

Optimisation is done by choosing suitable start times $s(O_{ij}) \in \mathbb{R}$ and resource allocations $R_{ij} \in \mu_{ij}$. A solution is valid, if the following two restrictions are met:

1. All grid jobs are planned and resources are allocated exclusively:

$$\begin{gathered}\forall O_{ij} : \exists s(O_{ij}) \in \mathbb{R}, R_{ij} \in \mu_{ij} \colon \forall M_j \in R_{ij} : \\ M_j \text{ is in } [s(O_{ij}); s(O_{ij}) + t(O_{ij}, R_{ij})] \text{ exclusively allocated by } O_{ij}.\end{gathered} \tag{2.1}$$

2. Precedence relations are adhered to:

$$\forall i, j \neq k : p(O_{ij}, O_{ik}) \Rightarrow s(O_{ik}) \geq s(O_{ij}) + t(O_{ij}, R_{ij}) \tag{2.2}$$

A violation of the two following soft constraints is treated by penalty functions in such a way that the amount of time and cost overruns is considered as well as the number of application jobs affected.

1. All application jobs J_i have a cost limit c_i, which must be observed:

$$\forall J_i : c_i \geq \sum_{j=1}^{n_i} \int_{s(O_{ij})}^{s(O_{ij})+t(O_{ij},R_{ij})} c(z, R_{ij}) \mathrm{d}z \tag{2.3}$$

2. All application jobs J_i have due dates d_i, which must be adhered to:

$$\forall J_i : d_i \geq s(O_{in_i}) + t(O_{in_i}, R_{in_i}) \text{ where } O_{in_i} \text{is the last grid job of } J_i \tag{2.4}$$

The fitness calculation is based on the above-mentioned four objectives and a auxiliary objective. It measures the average delay of each non-terminal grid job (i. e. a grid job with no successors) relative to the earliest starting time of its application job and it is aimed at rewarding the earlier completion of non-terminal grid jobs. The idea is to support the process of starting grid jobs earlier, such that the final grid job can be completed earlier in the end, which is recognised by the main objective *completion time*. Lower and upper estimations for costs and processing times are calculated in the first planning stage of GORBA, which will be described in the next section. Except for the utilisation rate, the value $crit_{i,val}$ of every $criterion_i$ is calculated relative to these limits based on the actual value $criterion_{i,act}$ as follows:

$$crit_{i,val} = \frac{criterion_{i,act} - criterion_{i,min}}{criterion_{i,max} - criterion_{i,min}} \tag{2.5}$$

This makes the single values $crit_{i,val}$ independent of the task on hand and results in a percentage-like range. These values are weighted and summed up, which yields the *raw fitness*. To avoid unwanted compensation effects, the criteria are sorted singly or in groups according to priorities. The criteria of the highest priority always contribute to the sum, while the others are added, if all criteria of the next higher priority fulfil a given threshold value. Weights and priorities are based on experience and aimed at reaching a fair compromise between users

and resource providers. The tuning of the suggested adjustment is left to the system administrator. If the two soft constraints are violated, the raw fitness is lowered to the *end fitness* by a multiplication by the corresponding penalty function, each of which delivers a factor between 0 and 1. Otherwise, end fitness and raw fitness are identical.

Generalising, this task contains the *job shop scheduling* problem as a special case. The extensions are co-allocation of heterogeneous and alternative resources of different performances and time-dependent availability and costs, earliest start times and due dates, parallel execution of grid jobs, and more than one objective. As our task includes the job shop problem, it is NP-complete. For this reason and because of the three minutes runtime limit, approximated solutions can be expected only.

A comparable problem could not be found in literature, see e.g. [6] and [7] for a comprehensive presentation of scheduling problems. This corresponds to the results of the literature review found in [32]. There, it is concluded that only few publications deal with multiple objectives in scheduling and, if so, they mostly deal with single machine problems. Within the grid domain some papers dealing with multi-criteria optimisation were published recently. In [35] it is reported that most of them deal with two criteria, like e.g. [10], and that in most cases only one criterion is really optimised while the other serves as a constraint, see e.g. [31, 37]. The approach from [37] uses matrix-like chromosomes, which is probably the reason why they can handle about 30 jobs within one hour only. Kurowski et al. [24] use a modified version of the weighted sum for a real multi-criteria optimisation, but do not handle workflows. The same holds for Xhafa et al. [36], who use two criteria, *makespan* and *average flow time*. They can be regarded as less conflicting as time and costs, which are obviously contradicting goals. Their approach and ours have the following points in common: The usage of a Memetic Algorithm, a structured population, the concept of fast and permanent replanning, and experiments based on workloads and ressource pools of comparable sizes. One difference regarding both, EA and MA, is that we use much more heuristics for seeding the initial population and for resource allocation. In [20] we have shown that heuristic resource selection outperforms its evolutionary counterpart if the runtime for planning is strictly limited. GLEAM uses a structured population based on the diffusion model since the early 90's, see [13, 14, 19, 18]. Regarding the balance between exploration and exploitation it has properties comparable to the cellular approach used by Xhafa et al. But cellular neighbourhoods are harder to control and to adjust (cf. [36]) than the ring structure we use. The latter needs only one parameter to steer the ratio between exploration and exploitation and this is the *neighbourhood size*. The cellular approach allows the genetic information to spread in two dimensions, which requires smaller neighbourhood sizes, as also Xhafa et al. observed in their experiments [36]. Assuming the same populaton size and a comparable slow loss of genotypic diversity during the course of evolution the ring topology allows larger neighbourhoods and thereby the establishment of better elaborated niches of individuals. This and the simpler control ability can be regarded as advantages and are arguments in favour for our continuing usage of the ring based diffusion model introduced by Gorges-Schleuter 20 years ago [13, 14].

Summarising, beside the work of Xhafa et al. [36] we did not find any report about a comparable amount of resources and grid jobs organised in workflows and subject to a global multi-criteria optimisation. Of course, a lot of publications focus on partial aspects of this problem. For instance, the well-known Giffler-Thompson algorithm [12, 28] was extended to the given problem, but surprisingly produced inferior results than our heuristics [20] described below.

3. Algorithms of GORBA and Results from the Tuning Phase. GORBA [21, 20, 22] uses advanced reservations and is based on Globus toolkit 4 at present. It executes a two-stage planning process. In the first stage the data of new application jobs are checked for plausibility and a set of heuristics is applied that immediately delivers first estimations of costs and completion times. These results are also used to seed the start population of the subsequent run of the Evolutionary Algorithm GLEAM (**G**lobal **L**earning **E**volutionary **A**lgorithm and **M**ethod) [4, 5].

Firstly, the old heuristics used for the tuning of GORBA reported in [33, 20] are described, followed by the new ones for rescheduling. This is followed by a short introduction of GLEAM and a description of the local search procedure and the adaptive mechanism of its application as a meme of GLEAM.

3.1. Heuristics for Planning and Replanning. In a first step a sequence of grid jobs is produced by the following three heuristic rules taking the precedence rules into account:

1. *Shortest due time*: grid jobs of the application job with the shortest due time first
2. *Shortest working time of grid job*: grid jobs with the shortest working time first

3. *Shortest working time of application job*: grid jobs of the application job with the shortest working time first

In the next step resources are allocated to the grid jobs using one of the following three resource allocation strategies (**RAS**):

RAS-1: Use the fastest of the earliest available resources for all grid jobs
RAS-2: Use the cheapest of the earliest available resources for all grid jobs
RAS-3: Use RAS-1 or RAS-2 for all grid jobs of an application job according to its time/cost preference

As every RAS is applied to each grid job sequence, nine schedules are generated, from which the best one is selected as result. These heuristics are rather simple and, therefore, more sophisticated ones have been investigated. Among them, such well-known ones like the Giffler Thompson Algorithm [12, 28], Tabu Search [3], Shifting Bottleneck [1, 8], and GRASP (Greedy Randomised Adaptive Search Procedure) [30]. To our surprise, they all performed inferior to our best heuristic, which is based on *shortest working time of application job* [20, 27]. Thus, the first planning phase is still based on the three first heuristics, which can be used for both creating a new schedule and replanning an already started one. In the latter case the information of the old plan is ignored, of course.

3.2. New Replanning Heuristics. The new heuristics tailored to the replanning situation use the grid job sequence of the old plan for grid jobs, which are subject to rescheduling, i. e. all grid jobs which have not already been started or will be started within the three minutes time frame. The new grid jobs are sorted according to one of the three heuristic rules already described and added to the sequence of old jobs, yielding three different sequences. Resources are allocated using the three RAS and again, nine more schedules are generated, but this time based on the old plan. The best of these eighteen schedules is the final result of the first planning stage, while all are used to seed the subsequent EA run of the second stage.

3.3. Evolutionary Search. The EA GLEAM already contains some evolutionary operators designed for combinatorial problems. They are summarised here only, due to the lack of space, and the interested reader is referred to [4, 5]. A chromosome consists of a sequence of segments, containing a sequence of genes, whose structure is defined by their associated problem-configurable *gene types*. The definition of a gene type constitutes its set of real, integer, or Boolean parameters together with their ranges of values. The latter are observed by the standard genetic operators of GLEAM so that explicit constraints of parameters are always met. For the problem in hand there are two basic gene types, the *grid job genes* and the RAS-gene described later. Each grid job gene consists of a grid job identifier (*id* in figure 3.1), represents a grid job, and has no parameters. The gene sequence is important, as it determines the scheduling sequence described later. Apart from the standard mutation, which changes the sequence of genes by simply shifting one of them, GLEAM contains genetic operators for arbitrary movements of gene segments and for the inversion of their internal order. As segment boundaries can be changed by some mutations, the segments form an evolvable meta structure over the chromosomes. Segment boundaries are also used for the standard 1- and n-point crossover operators, which include a genetic repair that ensures that no offspring lacks genes in the end. The evolvable segmentation and the problem-configurable gene types as well as their associated operators among others distinguish GLEAM from most standard EAs.

Figure 3.1 illustrates the coding by an example consisting of two simple application jobs or workflows, one containing five grid jobs and the other two. The grid jobs of all workflows are labeled by distinct identifiers *id*:

$$1 \leq id \leq n = \sum_{l=1}^{l} n_i \tag{3.1}$$

Besides the grid job genes, each chromosome contains a special additional gene for the selection of the RAS, the so called *RAS-gene*. It has one integer parameter, the *ras-id*: $1 \leq \textit{ras-id} \leq 3$. This is the only parameterised gene in this coding and thus, the only one which can undergo parameter mutation (in this case a uniform mutation). The position of the RAS-gene has no significance for the interpretation of the chromosome.

A schedule is constructed from the grid job genes in the sequence of their position within the chromosome as follows:

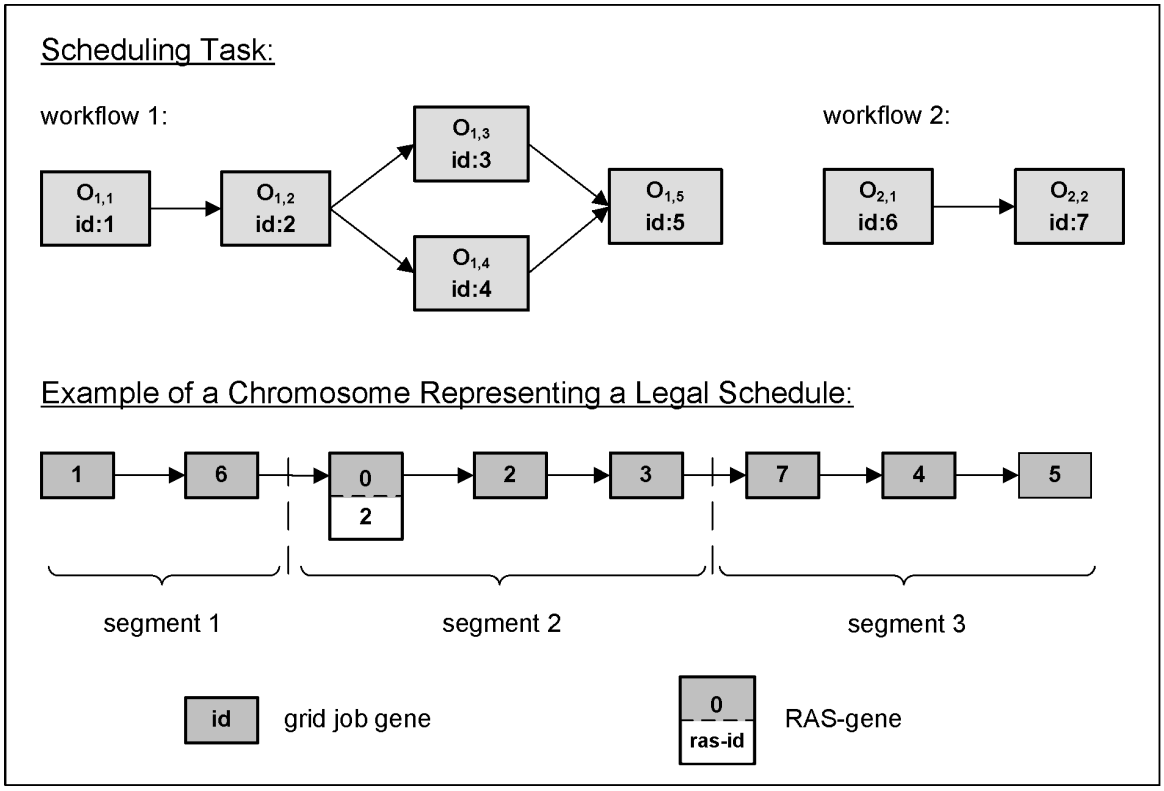


FIG. 3.1. *Example of a scheduling task consisting of two simple workflows and a chromosome representing a legal schedule. It is legal because there is no gene representing a direct predecessor of a grid job, which is located after the gene of that grid job. Thus, all precedence rules will be observed. The example chromosome consists of three segments, the role of which during evolution is described in the text. The parameter part of the RAS-gene is marked by a white background.*

Step 1: The earliest start time of a grid job is either the earliest start time of its application job or the latest end time of its predecessors, if any.
Step 2: According to the RAS selected by the RAS-gene, a list of alternatives is produced for every primary resource.
Step 3: Beginning with the first resources of the lists, the duration of the job is calculated and it is searched for a free time slot for the primary resource and its depending ones, beginning at the earliest start time of step 1. If no suitable slot is found, the resources at the next position of the lists are used.
Step 4: The resources found are allocated to the grid job with the calculated time frame.

The example chromosome shown in figure 3.1 is a legal solution, as all grid jobs are scheduled in such a way that no job is scheduled before its predecessor. In this case the described scheduling mechanism ensures that the precedence rules are adhered to. Alternatively, a phenotypic repair is performed, which is described and assessed in detail in [33, 20]. It retains all grid jobs with missing predecessors from beeing scheduled until their predecessors are scheduled.

3.4. Memetic Search. The basic idea of a Memetic Algorithm is to mimic the learning capability of individuals in nature by an improvement procedure, usually in the form of some local or heuristic search applied to the offspring. This type of enhanced EA is called Memetic Algorithm, a term coined by Moscato [26]. For the problem on hand, the multi-objective nature of the fitness function makes it hard, if not impossible to define a local searcher (LS), for which local assessment is sufficient. This is, because it is hard to decide whether a small local change of e.g. a job sequence is an improvement or not without setting up the complete allocation matrix. For the travelling salesman problem, for instance, which is a common benchmark task for combinatorial optimisation, it is easy to judge whether the swapping of two cities, for example, yields a better result or not. Consequently, we have dropped this desirable property of local assessment and now accept the necessity of elaborating the entire schedule to evaluate a change. The first result is the local searcher described at the end of this section.

```
initialise and evaluate start population
REPEAT                                                    // generational loop
  FOR all individual of the population                    // loop of matings
    choose partner
    FOR all predefined sets of genetic operators          // offspring generation
      generate offspring and evaluate them
      IF best_improvement                                 // best-improvement
        improve best offspring by local search
      ELSE                                                // adaptive all-improvement
        FOR all offspring
          IF offspring is best from evolution OR with probability p_all
            improve offspring by local search
            record effort for adaptation of p_all         // preparation
        calculate and record relative fitness gain for p_all // of adaptation
        IF enough matings                                 // adaptation
          perform adjustments of p_all                    // of p_all
      IF fitness of best offspring is sufficient
        best offspring substitutes parent (individual which chose a partner)
      IF best offspring was locally improved AND
          is accepted for the next generation
        update chromosome according to results of local search
      delete all not accepted offspring
UNTIL termination criterion is satisfied
deliver best individual at minimum as result
```

FIG. 3.2. *Pseudocode of the basic EA GLEAM and its memetic extension (marked by a dark grey background)*

Firstly, the LS is regarded a black box and it will be shown how it is integrated in the EA and what is adjusted adaptively. Figure 3.2 shows both the basic EA and its memetic extension indicated by a dark grey background. Within the generational loop, every individual of the population chooses a partner and produces some offspring as determined by the sets of genetic operators and their predefined probabilities. According to the strategy parameter *best_improvement*, only the best offspring is locally enhanced or its siblings have the chance to undergo local search, too (indicated as *adaptive all-improvement* in figure 3.2). The general adaptation scheme, which is described in detail in [16, 17, 18], controls strategy parameters like termination thresholds or iteration limits of the applied local searchers (LS) as well as their application intensity in case of *adaptive all-improvement*. If more than one LS are used, their selection also is adaptively controlled. The adaptation is based on the costs caused by the local search measured in additional fitness calculations and by the benefit obtained measured in relative fitness gain. The relative measurement of the fitness improvement is motivated by the fact that the same absolute amount may be achieved easily in the beginning of an optimisation run when the overall quality is low and hardly in the end of an optimisation process. For more details about the adaptation see [16, 17, 18].

In this particular case, only one local searcher is applied and it has no strategy parameters to control its search intensity, as will be described later. Thus, figure 3.2 only contains the adaptation of the probability *p_all*, by which the siblings of the best offspring are selected for improvement. When a predefined amount of matings has been processed, the adaptive adjustment of *p_all* is performed based on the observed relative fitness gain and the required evaluations recorded for the matings since the last adjustment. If the quality of the best offspring of a mating is sufficient to substitute its parent and if it was improved by local search, the chromosome is updated according to the LS results. For the details of offspring acceptance and the rationale of chromosome update (Lamarckian evolution), see [4, 5, 18]. The outer loop is performed until a given termination criterion, which is here the time limit of three minutes, is fulfilled.

As mentioned before, the RAS-gene determines the RAS used in the second step of the schedule construction. For crossover operators the RAS-gene of the better parent is inherited to the offspring assuming that it will

perform best for the new child. But what, if not? This question motivates a check-out of all reasonable alternatives in the form of a local search. Reasonable means in this context that alternative RAS are determined for testing based on the actual RAS and the properties of the associated schedule. If, for example, a schedule is too costly and RAS-2 preferring cheap resources was used for its construction, only RAS-3 may perform better. If fast resources were preferred (RAS-1) both other RAS may yield better results. All other situations are treated accordingly.

Due to the fact that this simple local search procedure does not possess any parameters to control its search intensity, the adaptation of the Simple Adaptive Memetic Algorithm (ASMA) is limited to the adjustment of its adaptive all-improvement part. As pointed out in [16, 18], this type of Memetic Algorithm is called simple, because it works with one meme or local searcher only.

3.5. Benchmarks and Results from Earlier Investigations. In a first development phase of GORBA the incorporated algorithms were tested and tuned using our standard benchmark set without co-allocation described in [34]. We use synthetic benchmarks, because it is easier to ensure and steer dissimilarities as described below. The set consists of four classes of application jobs, each class representing different values of the following two characteristics: the degree of dependencies between grid jobs D and the degree of freedom of resource selection R:

$$D = \frac{\sum_{i=1}^{l} \sum_{j=1}^{n_i} p_{ij}}{n(n-1)/2} \tag{3.2}$$

$$R = \frac{\sum_{i=1}^{l} \sum_{j=1}^{n_i} card(\mu_{ij})}{nm} \tag{3.3}$$

where p_{ij} is the number of predecessors of grid job O_{ij}, n is the number of all grid job as defined in (3.1).

The resulting values are in the range between 0 and 1 with small values for few dependencies or a low number of resource alternatives respectively. The benchmarks are constructed in such a way that random values for p_{ij} and $card(\mu_{ij})$ are chosen using a uniform distribution within given ranges. These bounds are adjusted so that either a small degree of about 0.2 or a large one of about 0.8 is achieved. The four combinations of low and high degrees result in four basic benchmark classes, which are abbreviated by *sRsD*, *sRlD*, *lRsD*, and *lRlD*, where *s* stands for small and *l* for large values of R and D.

The duration of grid jobs ranges uniformly distributed between three and seven time units. The same holds for the amount of grid jobs per application job. As the total number of grid jobs is another measure of complexity, benchmarks containing 50, 100, and 200 grid jobs are defined using the same set of resources for every amount. A fourth benchmark set again consists of 200 grid jobs, but with a doubled set of resources available. Time and cost budgets of the application jobs were manually adjusted so that the described heuristics just could not solve them and the exciting question was whether the EA can produce schedules that observe the budgets. The four benchmark classes, together with four different loads, yielded a total of 16 benchmarks [34]. They were used for former investigations, the results of which are summarised at the end of this section.

The benchmarks used for the experiments presented here are based on the described ones with the following peculiarities. In the experiment section four questions are investigated, see §4, which deal firstly with the usability of the old plan for rescheduling and the effectiveness of the new rescheduling heuristics. For these questions the old benchmarks with 100 and 200 grid jobs are used. And secondly, the processible workload and the effect of the Memetic Algorithm compared to the pure evolutionary search is investigated. For that the amount of grid jobs and resources is increased proportionally while preserving the remaining properties of the basic benchmarks based on 200 grid jobs. But in this case, the budgets are no longer adjusted manually to reduce the effort. The new constraints are based on the results of a first heuristic planning with a given percentage of average reduction to make them harder.

The investigations presented here are based on the results reported in [20, 33]. This means that the described coding of the chromosomes is used together with the already mentioned phenotypic repair of possible violations of precedence rules of the grid jobs. The advantage of this approach compared to repair mechanisms on the gene level is that intermediate steps of shifting genes, which themselves may be faulty, are allowed to occur. The idea

is that the resulting schedule is improved after some generations. Furthermore, the well-known OX crossover reported by Davis [9] is added to the set of genetic operators. It preserves the relative order of the parent genes and is aimed at combining of useful gene sequences rather than disturbing them, as (larger) disturbances are provided by the standard crossover operators.

4. Experimental Results for Fast Rescheduling. There are various reasons for rescheduling, of which the introduction of new application jobs is the most likely one. Others are job cancellations or terminations, new resources, resource breakdowns, or changes in the availability or prices of resources. The experiments are based on the most likely scenario of new application jobs and shall answer the following four questions:

1. Does rescheduling benefit from the old plan? If so, to which fraction of finished and new grid jobs?
2. How effective are the old and new heuristics and the subsequent GLEAM run?
3. Up to which amount of grid jobs and resources does the EA or ASMA improve the best heuristically generated schedule?
4. Does the ASMA perform better than the EA and if so, which improvement type is better, *best-improvement* or *adaptive all-improvement*?

As the two benchmark classes based on large degrees of dependencies turned out to be harder than those using small degrees [20, 33], they were used here to answer the questions one, two, and four to limit the effort. In contrast to the work presented at the already mentioned PPAM conference [22], the examination of the third question was now based on all four classes. As pointed out in [20] and [33], the time and cost limits of the application jobs were set so tightly that the heuristics could not solve them without violating these soft constraints. One criterion of the usefulness of the EA run was to find fitting schedules, which was achieved in most, but not all cases. In addition to this criterion, the end fitness values obtained were also compared for the new investigations.

For the experiments reported here, the only EA parameter tuned was the population size varying from 90 to 800 for the runs investigating the first two questions. For the last two ones, smaller population sizes also had to be used, as will be described later on. For every benchmark setting and population size, 50 runs were done, with the results being based on the averages. Confidence intervals and t-tests were used to check the significance of differences at a confidence range of 99%.

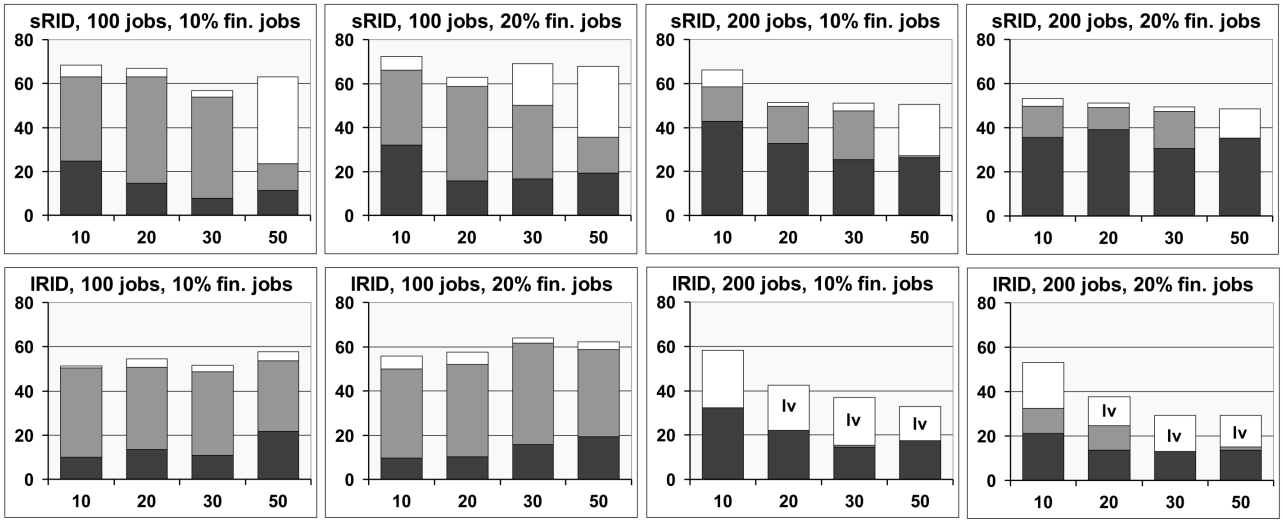


FIG. 4.1. *Comparison of the fitness shares obtained from the basic heuristics (dark grey), rescheduling heuristics (light grey) and GLEAM (white) for all 32 rescheduling settings. X-axis: fraction of new grid jobs in percent relative to the original schedule size, y-axis: normalised end fitness. All GLEAM runs improve the fitness significantly. Even for the smallest improvement of benchmark lRlD, 100 grid jobs, 10% fin. jobs, the best heuristic fitness is clearly outside of the confidence interval of the GLEAM result (three times more than necessary). Abbreviations: fin. jobs: finished grid jobs, lv: limit violations (mostly 1 to 3 application jobs violating the due date), for sRlD and lRlD see previous page.*

For the first two questions, the two basic benchmark scenarios were used for the first planning, with 10 resources and application jobs consisting of 100 and 200 grid jobs, respectively. Eight rescheduling events were compared, which take place when 10 or 20% of the grid jobs are finished and new application jobs with 10, 20,

30, or 50% grid jobs (relating to the original schedule size) are added. This results in 32 benchmark settings. We concentrated on small amounts of already processed grid jobs, because this is more likely in practice and gives a chance for the old schedule to be useful. Otherwise, the situation is coming closer to the already investigated "new planning" situation.

Figure 4.1 compares the results for all 32 benchmark settings. It must be mentioned that a certain variation in the resulting normalised fitness values is not relevant, as the benchmarks are generated with some stochastic variations. Values between 50 and 70 may be considered good results, as the lower and upper bounds for the evaluations are theoretical minima and maxima. Results close to 100% would indicate either a trivial case or a software error. The most important outcome is that for 10% new grid jobs, all eight scenarios perform well. The contribution of the heuristics is clearly situation-dependent and if they tend to yield poor results, GLEAM compensates this in most cases. In other words, if the heuristics can solve the problem well, there is smaller room left for an improvement at all. Another nice result is that this compensation is also done to a certain extent for more new grid jobs, even if the schedules cannot be made free of limit violations (cf. the six columns indicated by *lv*, which stands for limit violations). It can be expected that more new grid jobs will lower the contribution of the replanning heuristics and, in fact, Figure 4.1 confirmes this for the instance of 50% new grid jobs. The case of lRlD, 200, and 10% finished grid jobs is somewhat exceptional, as the replanning heuristics do not work well even in the case of few new jobs.

TABLE 4.1

Comparison of the contributions of all rescheduling heuristics for the different fractions of finished and new grid jobs. The best values of each column are marked dark grey, while values which reach 90% of the best at the minimum are marked light grey. Abbreviations: SDT: shortest due time, SWT: shortest work time, AJ: application job, GJ: grid job, RAS: see §3.

Finished grid jobs:	10%				20%				Ave-
New grid jobs:	10%	20%	30%	50%	10%	20%	30%	50%	rage
RH SDT & RAS-3	0.90	0.96	0.88	0.92	0.86	0.83	0.89	0.92	0.90
RH SDT & RAS-2	0.70	0.44	0.73	0.80	0.64	0.59	0.75	0.61	0.66
RH SDT & RAS-1	0.48	0.44	0.54	0.45	0.59	0.22	0.53	0.45	0.46
RH SWT (GJ) & RAS-3	0.97	0.86	0.81	0.69	0.94	0.78	0.81	0.62	0.81
RH SWT (GJ) & RAS-2	0.74	0.42	0.63	0.53	0.66	0.50	0.68	0.39	0.57
RH SWT (GJ) & RAS-1	0.47	0.41	0.46	0.28	0.57	0.24	0.54	0.26	0.40
RH SWT (AJ) & RAS-3	0.90	0.88	0.82	0.70	0.86	0.83	0.77	0.70	0.81
RH SWT (AJ) & RAS-2	0.70	0.41	0.70	0.56	0.64	0.51	0.57	0.46	0.57
RH SWT (AJ) & RAS-1	0.48	0.44	0.57	0.31	0.59	0.24	0.49	0.43	0.44
SDT & RAS-3	0.45	0.42	0.35	0.56	0.45	0.41	0.47	0.51	0.45
SDT & RAS-2	0.58	0.52	0.38	0.72	0.51	0.43	0.56	0.69	0.55
SDT & RAS-1	0.42	0.42	0.39	0.54	0.39	0.37	0.37	0.45	0.42
SWT (GJ) & RAS-3	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
SWT (GJ) & RAS-2	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
SWT (GJ) & RAS-1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
SWT (AJ) & RAS-3	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
SWT (AJ) & RAS-2	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
SWT (AJ) & RAS-1	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Table 4.1 illustrates the contribution of each heuristic. For each of the 32 benchmark settings, the fitness of each heuristic is calculated relative to the best heuristic result for this setting. The four values for both grid job amounts for *sRlD* and *lRlD* are averaged and shown in the table. The right column again averages the values for the different finished and new grid job fractions. The old heuristics based on short working times in the lower half of the table show the same poor behaviour as for planning an empty grid [20], but when taken as a basis for the new rescheduling heuristics, they contribute quite well. According to the table, RAS-3 performs best, but the raw material not shown here has thirteen cases, in which the two other RAS are the best ones. Thus, it is meaningful to use them all for replanning, but it seems to be wise to drop the six old heuristics sorting according to short working times.

To investigate the last two questions concerning the processible load, the rescheduling scenario with 10% finished and 10% new grid jobs is used with proportionally growing numbers for grid jobs and resources. The studies on the behaviour of the EA GLEAM are based on all benchmark classes, while the ASMA is checked with sRlD and lRlD only to limit the effort. The penalty functions showed a weakness when applied to an increasing load, with the penalty factor converging to one for small numbers of penalty causing application jobs. Thus, the two functions penalising the number of too costly or late application jobs were modified to yield a value 0.8 instead of near 1.0 in case of just one job. 0.8 was chosen to have a distinct separation from penalty-free schedules with weaker overall quality. It must be borne in mind that a suitable definition of the fitness function is crucial to the success of every evolutionary search. To achieve comparability, all runs were done with the modified fitness function, so that the results reported in [22] differ in the details of the figures from those reported here. But the general tendencies remain the same as expected.

The comparisons are based on the fitness improvement obtained by the EA compared to the best heuristic result and on the success rate, which is the ratio between violation-free and total runs per benchmark setting. Figure 4.2 shows the results. As expected, success rate and EA improvement decrease with growing load. For large resource alternatives, good success rates can be preserved much longer. This is not surprising, as more resources mean more planning freedom. The relatively large variations of the EA improvement can be explained by the varying ability of the heuristics to produce schedules with more or less limit violations. As limit violations are penalised severely, a small difference already can produce a relatively large fitness alteration.

No violations at all leave little room for improvements like in the two cases of sRlD and lRsD and 200 grid jobs. There is a clear correspondence between the success and the magnitude of the confidence intervals. This is because the penalty function reduces the fitness significantly. For loads of up to 400 grid jobs and 40 resources, GLEAM can always produce violation-free schedules. For up to 1000 grid jobs and 100 resources, this can be achieved for the cases of large resource alternatives. From that on to 1600 grid jobs and 160 resources, the chance of producing successful schedules is slowly diminishing. But improvements of the overall quality and below the level of violation-free schedules are still possible for loads of up to 7000 grid jobs and 700 resources, as shown in the lower part of Figure 4.2. In other words, the amount of application jobs keeping the budgets is still increased compared to the heuristic results.

The fourth question concerns the effect of adding local search to the evolutionary process. The results for sRlD and lRlD exhibit better success rates for an increasing load, as shown in Figures 4.3 and 4.4. For sRlD both improvement types of the ASMA yield a success for 600 grid jobs and 60 resources, which could not be achieved by GLEAM alone. In case of lRlD, the ASMA runs with adaptive all-improvement produce the better results for the three loads between 1200 and 1600 grid jobs. Looking at the improvement rate, it is striking that there are only minor differences, which in most cases are not significant, for all completely or nearly successful runs. Again, the ASMA with adaptive all-improvement works better for the lRlD benchmarks in all cases, but one (5000 grid jobs) and in many cases of the other benchmark series. As there are four sRlD cases and one lRlD case, where GLEAM performs best (900, 1000, 1200, and 2400 grid jobs), more investigations are necessary. For results like sRlD and 3200 or 5000 grid jobs, the t-test shows that the differences are not significant. In summary, there is a clear tendency in favour of the ASMA variant using adaptive all improvement, which is based mainly on the achieved improvements of the success rates.

Finally, the questions for how long improvements can be expected and which population sizes are good shall be studied. The more grid jobs must be planned, the less evaluations can be processed within the three minutes time frame. Figure 4.5 shows that this amount decreases continuously with growing load. The more resource alternatives are available, the more must be checked by the RAS, which lasts longer and explains the lower numbers for the two lR cases. In the long run, the evaluations possible decrease to such an extent that the population size must be reduced to 20 or 30 to obtain two dozens of generations at least in the end. It is obvious that with such small numbers, only poor results can be expected and it is encouraging that even with the largest load there still is a significant improvement. Hence, a faster implementation of the evaluation software or better hardware will enlarge the possible load or deliver better results for the loads investigated.

As mentioned above, the only tuned strategy parameter of the EA and the ASMA is the population size. For GLEAM, sizes between 20 and 400 were observed, if the exceptional cases of 10 and 800 are left out. The values are a little bit lower for the ASMA using best-improvement and about half (20 to 200) for adaptive all-improvement. This means that the range of potential good population sizes is significantly smaller for the ASMA based on adaptive all-improvement. This is due to the more intensive search done when more siblings of a mating are locally improved.

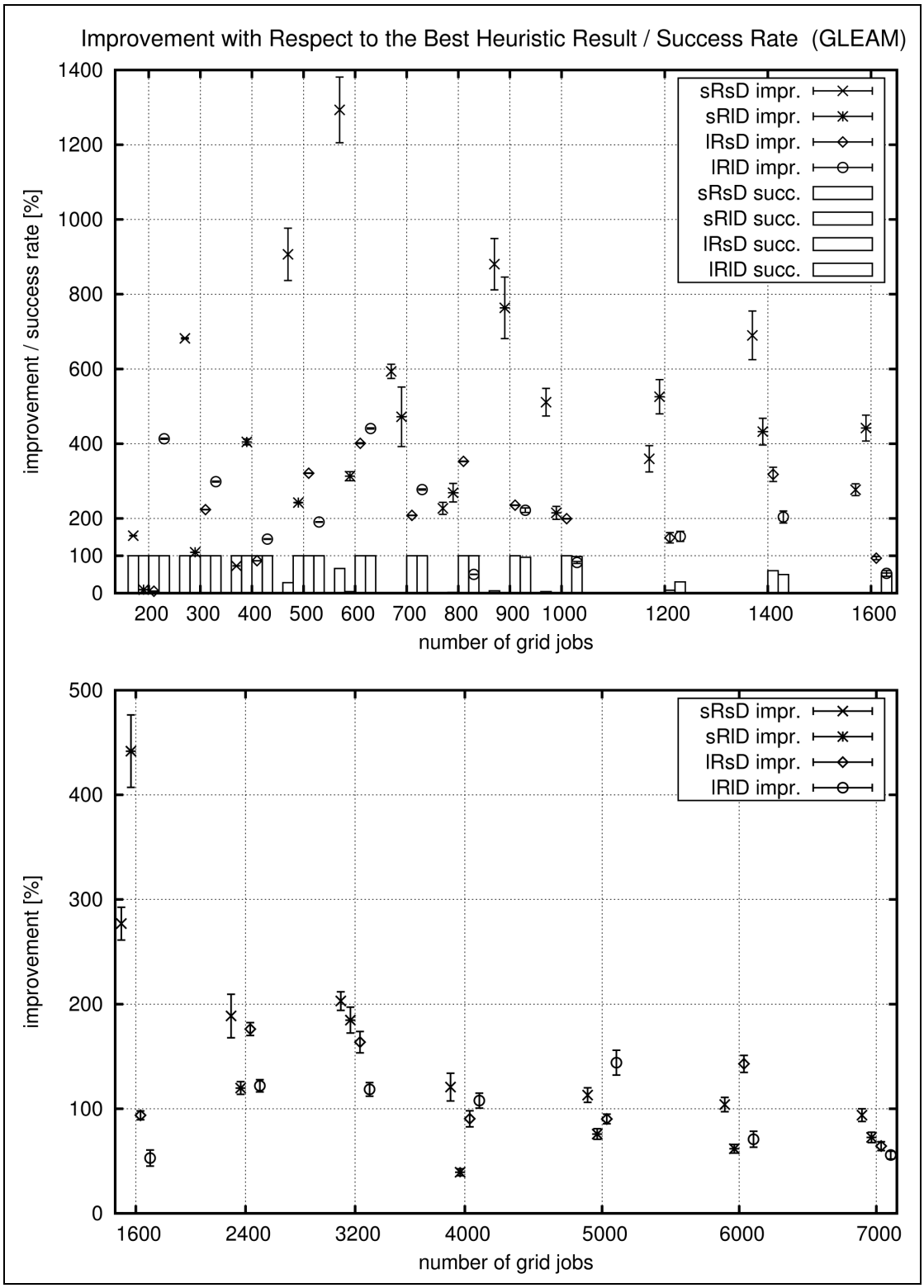


FIG. 4.2. *Success rate and EA improvement compared to the best heuristic at increasing load for all four benchmark classes and 10% finished and new grid jobs. The improvement factors are given with their confidence intervals. The bars of the success rates are not shaded, so that the marks of low success rates remain visible. They are always located in the same column as the corresponding success rate. The number of resources is in each case 10% of the number of grid jobs. For a better visibility the results for 2400 grid jobs and more are plotted in a separate diagram, where success rates already shown in the upper part are left out.*

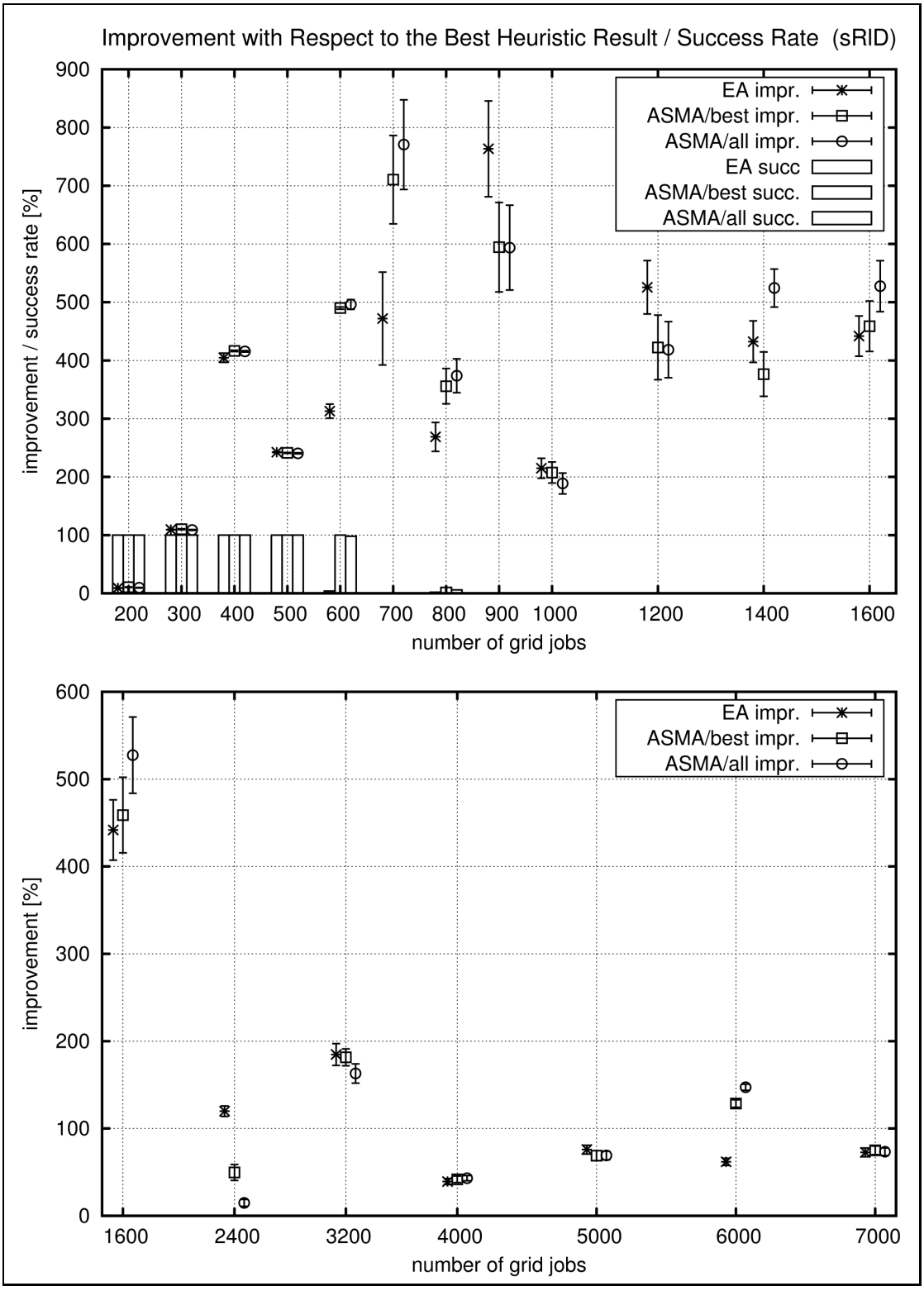


FIG. 4.3. *Success rate and EA improvement as described in Figure 4.2 for the sRlD benchmark series comparing GLEAM with the two ASMA variants described in §3.4.*

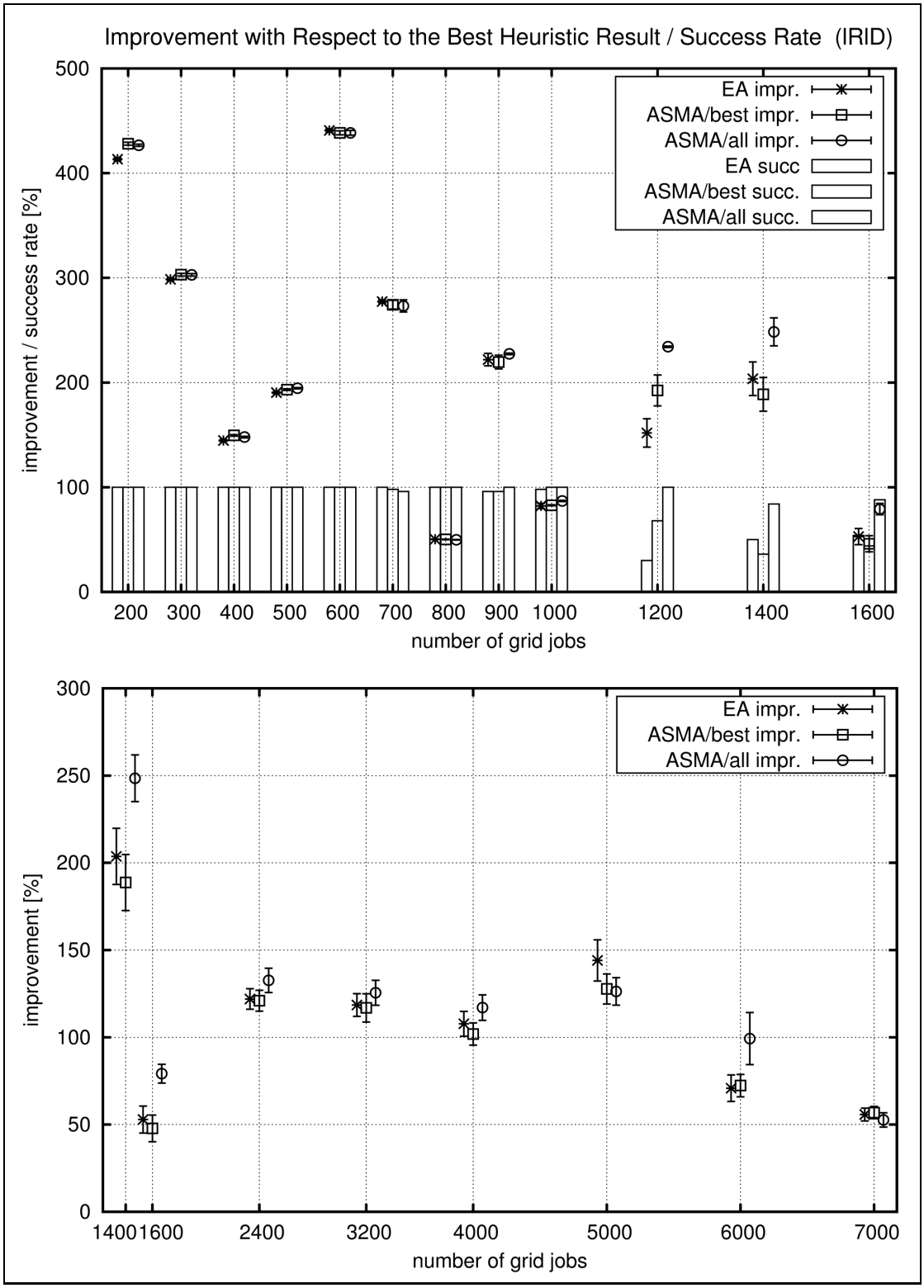


FIG. 4.4. *Success rate and EA improvement as described in Figure 4.2 for the lRlD benchmark series comparing GLEAM with the two ASMA variants described in §3.4.*

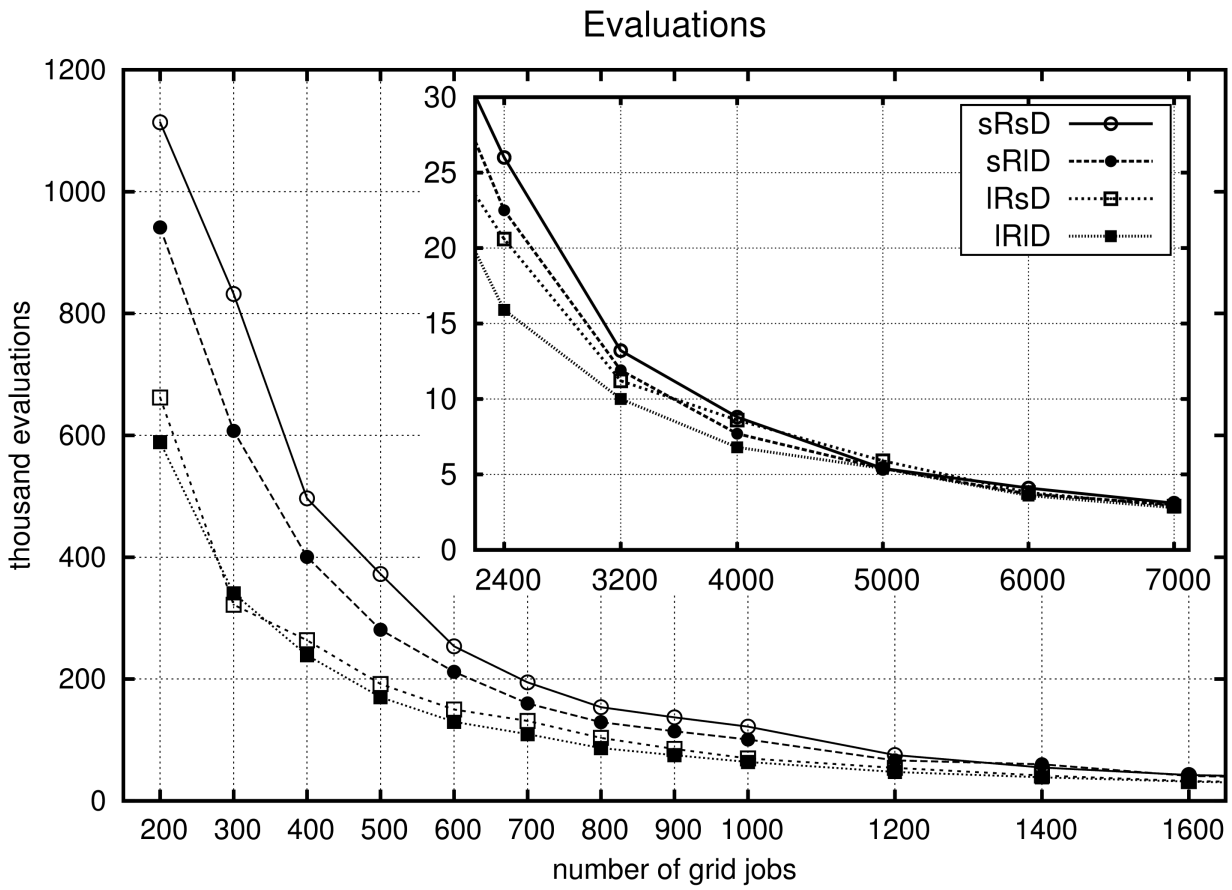


FIG. 4.5. *Evaluations possible within the three minutes time frame at increasing load. The number of resources is in each case 10% of the number of grid jobs. For a better visibility, the diagram for 2400 and more grid jobs is shown separately.*

5. Conclusion and Future Work. It was shown that the problem of scheduling grid jobs to resources based on realistic assumptions and taking the demands of resource users and providers into account is a much more complex task than just *job shop scheduling*, which, in fact, is complex enough, as it is NP-complete. The task on hand enlarges the classical problem by alternative and heterogeneous resources, co-allocation, and last, but not least by multi-objective optimisation. A local searcher was presented, which checks reasonable alternatives of the used resource allocation strategy (RAS) and it was shown how it was integrated in the Evolutionary Algorithm GLEAM. The resulting adaptive simple Memetic Algorithm (ASMA) was assessed using two of the four benchmark classes introduced. The promising results encourage us to implement and examine three more local searchers, which shift genes based on information obtained from the corresponding schedule.

The investigated problems are rescheduling problems, which are the common case in grid resource management. Rescheduling is necessary, if new jobs arrive, planned ones are cancelled, resources break down or new ones are introduced, to mention only the more likely events. For this purpose, new heuristics that exploit the information contained in the "old plan" were introduced. It was shown that the solution for the common case of smaller changes, i. e. in the range of up to 20% finished and new grid jobs, could be improved significantly.

The processible work load was also investigated for 10% finished and new grid jobs at an increasing number of jobs and resources. It was found that for loads of up to 7000 grid jobs and 700 resources, it was still possible to gain an improvement by the EA run, even if it is comparably small. As with this load only 27 generations and population sizes in the range of 20 or 30 are possible within the three minutes time frame for rescheduling, not many greater loads can be expected to be processible. A faster implementation of the evaluation software or better hardware could help. On the other hand, single resources only have been considered up to now. In

the future we want to extend the system to resources with capacities like clusters of homogeneous hardware or data storage resources.

REFERENCES

[1] J. Adams, E. Balas, and D. Zawack, *The shifting bottleneck procedure for job shop scheduling*, MANAGEMENT SCIENCE, 34 (1988).

[2] M. B. Bader-El-Den, R. Poli, and S. Fatima, *Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework*, Memetic Computing, 1 (2009), pp. 205–219.

[3] A. Baykasoğlu, L. Özbakir, and T. Dereli, *Multiple dispatching rule based heuristic for multi-objective scheduling of job shops using tabu search*, in Proceedings of MIM 2002: 5th Int. Conf. on Managing Innovations in Manufacturing (MIM), Milwaukee, Wisconsin, USA, 2002, pp. 396–402.

[4] C. Blume and W. Jakob, *Gleam - an evolutionary algorithm for planning and control based on evolution strategy*, in GECCO 2002, E. Cantú-Paz, ed., vol. LBP, 2002, pp. 31–38.

[5] ———, *GLEAM - General Learning Evolutionary Algorithm and Method : ein Evolutionärer Algorithmus und seine Anwendungen*, vol. 32 of Schriftenreihe des Instituts für Angewandte Informatik - Automatisierungstechnik, (in German), KIT Scientific Publishing, Karlsruhe, 2009.

[6] P. Brucker, *Scheduling Algorithms*, Springer, Berlin Heidelberg, 2004.

[7] ———, *Complex Scheduling*, Springer, Berlin Heidelberg, 2006.

[8] K.-P. Chen, M. S. Lee, P. S. Pulat, and S. A. Moses, *The shifting bottleneck procedure for job-shops with parallel machines*, Int. Journal of Industrial and Systems Engineering 2006, 1 (2006), pp. 244–262.

[9] L. D. Davis and M. Mitchell, *Handbook of genetic algorithms*, Van Nostrand Reinhold, (1991).

[10] P.-F. Dutot, L. Eyraud, G. Mounié, and D. Trystram, *Bi-criteria algorithm for scheduling jobs on cluster platforms*, in SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, New York, NY, USA, 2004, ACM, pp. 125–132.

[11] I. Foster, C. Kesselman, and S. Tuecke, *The anatomy of the grid-enabling scalable virtual organizations*, International Journal of Supercomputer Applications, 15 (2001), pp. 200–222.

[12] B. Giffler and G. L. Thompson, *Algorithms for solving production scheduling problems*, Operations Research, 8 (1960), pp. 487–503.

[13] M. Gorges-Schleuter, *Genetic Algorithms and Population Structures—A Massively Parallel Algorithm*, PhD thesis, University of Dortmund, FRG, 1990.

[14] ———, *Explicit parallelism of genetic algorithms through population structures*, in PPSN I (1990), H.-P. Schwefel and R. Männer, eds., LNCS 496, Springer, 1991, pp. 150–159.

[15] W. E. Hart, N. Krasnogor, and J. Smith, eds., *Recent Advances in Memetic Algorithms. Studies in Fuzziness and Soft Computing*, vol. 166, Springer, 2004.

[16] W. Jakob, *Towards an adaptive multimeme algorithm for parameter optimisation suiting the engineers' need*, in PPSN IX, T. P. Runarsson, H.-G. Beyer, and J. J. Merelo-Guervos, eds., LNCS 4193, Berlin, 2006, Springer, pp. 132–141.

[17] ———, *A cost-benefit-based adaptation scheme for multimeme algorithms*, in Conf. Proc. PPAM 2007, LNCS 4967, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., Springer, Berlin, 2008, pp. 509–519.

[18] ———, *A general cost-benefit based adaptation framework for multimeme algorithms*, Memetic Computing, (accepted, to be published).

[19] W. Jakob, M. Gorges-Schleuter, and C. Blume, *Application of genetic algorithms to task planning and learning*, in PPSN II, R. Männer and B. Manderick, eds., North-Holland, Amsterdam, 1992, pp. 291–300.

[20] W. Jakob, A. Quinte, K.-U. Stucky, and W. Süss, *Fast multi-objective scheduling of jobs to constrained resources using a hybrid evolutionary algorithm*, in PPSN X, G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, eds., LNCS 5199, Springer, 2008, pp. 1031–1040.

[21] W. Jakob, A. Quinte, W. Süss, and K.-U. Stucky, *Optimised scheduling of grid resources using hybrid evolutionary algorithms*, in Conf. Proc. PPAM 2005, LNCS 3911, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Waśniewski, eds., Springer, Berlin, 2006, pp. 406–413.

[22] ———, *Fast multi-objective rescheduling of grid jobs by heuristics and evolution*, in Conf. Proc. PPAM 2009, LNCS 6067 or 6068 (to be published in July, 2010), R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., Springer, Berlin, 2010.

[23] N. Krasnogor, B. P. Blackburne, E. K. Burke, and J. D. Hirst, *Multimeme algorithms for protein structure prediction*, in Conf. Proc. PPSN VII, LNCS 2439, 2002, pp. 769–778.

[24] K. Kurowski, J. Nabrzuski, A. Oleksiak, and J. Weglarz, *Scheduling jobs on the grid - multicriteria approach*, Computational Methods in Science and Technology, 12 (2006), pp. 123–138.

[25] M. W. S. Land, *Evolutionary Algorithms with Local Search for Combinatorial Optimization*, PhD thesis, University of California, USA, 1998.

[26] P. Moscato, *On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms*, 1989. Tech. Rep. Caltech Concurrent Computation Program, Rep. 826, California Institute of Technology, Pasadena, CA.

[27] F. Möser, W. Jakob, A. Quinte, K.-U. Stucky, and W. Süss, *An assessment of heuristics for fast scheduling of grid jobs*. Submitted to the ICSOFT 2010 conference.

[28] K. Neumann and M. Morlock, *Operations Research*, Carl Hanser, München, 2002.

[29] Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong, *Classification of adaptive memetic algorithms: A comparative study*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 36 (2006), pp. 141–152.

[30] L. S. Pitsoulis and M. G. C. Resende, *Greedy randomized adaptive search procedures*, in Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende, eds., Oxford University Press, 2001, pp. 168–181.

[31] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, *Scheduling workflows with budget constraints*, in in Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds.: CoreGrid series, Springer-Verlag, 2007.

[32] A. Setämaa-Kärkkäinen, K. Miettinen, and J. Vuori, *Best compromise solution for a new multiobjective scheduling problem*, Computers & Computers and Operations Research archive, 33 (2006), pp. 2353–2368.

[33] K.-U. Stucky, W. Jakob, A. Quinte, and W. Süss, *Tackling the grid job planning and resource allocation problem using a hybrid evolutionary algorithm*, in Conf. Proc. PPAM 2007, LNCS 4967, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., Springer, Berlin, 2008, pp. 589–599.

[34] W. Süss, A. Quinte, W. Jakob, and K.-U. Stucky, *Construction of benchmarks for comparison of grid resource planning algorithms*, in ICSOFT 2007, Proc. of the Second ICSOFT, Volume PL/DPS/KE/WsMUSE, Barcelona, Spain, July 22-25, 2007, J. Filipe, B. Shishkov, and M. Helfert, eds., Inst. f. Systems and Techn. of Inf., Control and Com., INSTICC Press, 2007, pp. 80–87.

[35] M. Wieczorek, A. Hoheisel, and R. Prodan, *Taxonomy of the multi-criteria grid workflow scheduling problem*, in Grid Middleware and Services - Challenges and Solutions, D. Talia, R. Yahyapour, and W. Ziegler, eds., Springer US, New York, 2008, pp. 237–264.

[36] F. Xhafa, E. Alba, B. Dorronsoro, B. Duran, and A. Abraham, *Efficient batch job scheduling in grids using cellular memetic algorithms*, in Metaheuristics for Scheduling in Distributed Computing Environments, F. Xhafa and A. Abraham, eds., Springer, Berlin, 2008, pp. 273–299.

[37] J. Yu and R. Buyya, *A budget constrained scheduling of workflow applications on utility grids using genetic algorithms*, in Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC, IEEE, IEEE CS Press, 2006.

Edited by: Marcin Paprzycki
Received: March 30, 2010
Accepted: June 21, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 189–204. http://www.scpe.org
ISSN 1895-1767

VIESLAF FRAMEWORK: FACILITATING NEGOTIATIONS IN CLOUDS BY APPLYING SERVICE MEDIATION AND NEGOTIATION BOOTSTRAPPING

IVONA BRANDIC*, DEJAN MUSIC*, AND SCHAHRAM DUSTDAR*

Abstract. Cloud computing represents a novel and promising computing paradigm where computing resources have to be allocated to software for their execution. Self-manageable Cloud infrastructures are required in order to achieve that level of flexibility on one hand, and to comply to users' requirements specified by means of Service Level Agreements (SLAs) on the other. However, many assumptions in Cloud markets are old fashioned assuming same market conditions as for example in computational Grids. One such assumptions is that service provider and consumer have matching SLA templates and common understanding of the negotiated terms or that they provide public templates, which can be downloaded and utilized by the end users. Moreover, current Cloud negotiation systems have based themselves on common protocols and languages that are known to the participants beforehand. Matching SLA templates and a-priori knowledge about the negotiation terms and protocols between partners are unrealistic assumption in Cloud markets where participants meet on demand and on a case by case basis. In this paper we present VieSLAF, a novel framework for the specification and management of SLA mappings and meta-negotiations facilitating service mediation and negotiation bootstrapping in Clouds. Using VieSLAF users may specify, manage, and apply SLA mappings bridging the gap between non-matching SLA templates without a-priori knowledge about negotiation protocols, required security standards or negotiated terms. We exemplify two case studies where VieSLAF represents an important contribution towards the development of open and liquid Cloud markets.

Key words: grid services, cloud computing, autonomic computing, service negotiation

1. Introduction. Service-oriented Architectures (SOA) represent a promising approach for implementing ICT systems [9, 24, 30] by packaging the software to services that can be accessed independently of the used programming languages, protocols, and platforms. Despite remarkable adoption of SOA as the key concept for the implementation of ICT systems, the full potential of SOA (e.g., dynamism, adaptivity) is still not exploited [28]. SOA approach and Web service technologies represent large scale abstractions and a candidate concept for the implementation Cloud Computing systems, where massively scalable computing is made available to end users as a service [9, 24]. The key benefits of providing computing power as a service are (a) avoidance of expensive computer systems configured to cope with peak performance, (b) pay-per-use solutions for computing cycles requested on-demand, and (c) avoidance of idle computing resources [16].

Non-functional requirements of a service execution are termed as *Quality of Service (QoS)*, and are expressed and negotiated by means of *Service Level Agreements (SLAs)*. *SLA templates* represent empty SLA documents with all required elements like parties, SLA parameters, metrics and objectives, but without QoS values [12]. However, most existing Cloud frameworks assume that the communication partner know about the negotiation protocols, required security standards and negotiated terms before entering the negotiation and that they have matching SLA templates. These assumptions rely on related technologies (like computational Grids) and cannot be transferred to computational Cloud markets. In case of computational Clouds a priori knowledge about negotiation protocols and strategies as well as matching SLA templates represent unrealistic assumption since services are discovered dynamically and on demand.

In this paper we approach the gap between existing QoS methods and Cloud services by proposing a *Vienna Service Level Agreement Framework (VieSLAF)* architecture for Cloud service management with components for *service mediation* and *negotiation bootstrapping* [7, 8]. Thereby, we introduce so-called *meta-negotiations* to allow two parties to reach an agreement on what specific negotiation protocols, security standards, and documents to use before starting the actual negotiation. Moreover, we discuss the concept of SLA-mappings to bridge the gap between inconsistent SLA templates. The concept of SLA mappings can be exemplified in differences in terminology for a common attribute such as *price*, which may be defined as *usage price* on one side and *service price* on the other, leading to inconsistencies during the negotiation process. VieSLAF framework has been successfully applied to (i) develop Cloud infrastructures for the SLA-based resource virtualization [20] by utilizing our meta negotiation approach and (ii) to facilitate liquidity management in Clouds by using our SLA mapping approach [29]. Besides performance evaluation of the VieSLAF we discuss successful case studies for the application of VieSLAF to establish open and liquid Cloud markets.

*Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, Vienna, Austria, Emails: {ivona, dejan, dustdar}@infosys.tuwien.ac.at

The main contributions of this paper are (1) description of the scenarios for the definition of *SLA mapping* documents; (ii) development of the architecture for the *meta-negotiations* in Cloud systems; (iii) description of the *meta-negotiation document*; (iv) definition of the *VieSLAF* architecture used for the semi-automatic management of *SLA mappings* and *meta-negotiations* and (iv) demonstration of the usability of the VieSLAF framework for real-world Cloud negotiations.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 gives an overview about the goals of the adaptable, versatile, and dynamic services, in particular goals considering negotiation bootstrapping and service mediation. In Section 4 we discuss the meta-negotiation approach, whereas in Section 5 we present the SLA mapping approach. Section 6 presents the *VieSLAF* architecture. In Section 7 we evaluate SLA mapping and meta negotiation approach and report successful VieSLAF case studies. Section 8 concludes this paper and describes the future work.

2. Related Work. Currently, a large body of work exists in the area of service negotiation and SLA-based QoS. Most of the related work can be classified into the following four categories: (1) adaptive SLA mechanisms based on OWL, DAML-S and other semantic technologies [13, 26, 38]; (2) SLA based QoS systems, which consider varying service requirements but do not consider non matching SLA templates [1, 34]; (3) systems relying on the principles of autonomic computing [3, 21, 22]; and systems addressing versatile negotiation in Grids/SOAs [25, 27, 18]. Since there is very little work on service mediation and negotiation bootstrapping in Clouds we look in particular into related systems like Grids and SOAs [4].

Work presented in [27] discusses incorporation of SLA-based resource brokering into existing Grid systems. Oldham et al. describe a framework for semantic matching of SLAs based on WSDL-S and OWL [26]. Dobson at al. present a unified quality of service (QoS) ontology applicable to the main scenarios identified such as QoS-based Web services selection, QoS monitoring and QoS adaptation [13]. Zhou et al. survey the current research on QoS and service discovery, including ontologies such as OWL-S and DAML-S. Thereafter, an ontology is proposed, DAML-QoS, which provides detailed QoS information in a DAML format [38]. Hung et al. propose an independent declarative XML language called WS-Negotiation for Web services providers and requestors. WS-Negotiation contains three parts: negotiation message, which describes the format for messages exchanged among negotiation parties, negotiation protocol, which describes the mechanism and rules that negotiation parties should follow, and negotiation decision making, which is an internal and private decision process based on a cost-benefit model or other strategies [17]. Work presented in [1] extends the service abstraction in the Open Grid Services Architecture (OGSA) for QoS properties focusing on the application layer. Thereby, a given service may indicate the QoS properties it can offer or it may search for other services based on specified QoS properties.

Quan et al. discuss the process of mapping a light communication workflow within an SLA context with different kinds of sub-jobs and resources [25]. Dan et al. present a framework for providing customers of Web services differentiated levels of service through the use of automated management and SLAs [12]. Ardagana et al. present an autonomic grid architectures with mechanisms to dynamically re-configure service center infrastructures, which is basically exploited to fulfill varying QoS requirements [3]. Koller et al. discuss autonomous QoS management using a proxy-like approach. The implementation is based on WS-Agreement [36]. Thereby, SLAs can be exploited to define certain QoS parameters that a service has to maintain during its interaction with a specific customer [21]. König at al. investigate the trust issue in electronic negotiations, dealing with trust to a potential transaction partner and selection of such partners based on their past behavior [22].

Quan et al. and Ouelhadj et al. discuss incorporation of SLA-based resource brokering into existing Grid systems [25, 27]. Li et al. discusses Rudder framework, which facilitates automatic Grid service composition based on semantic service discovery and space based computing [23]. Hill et al. discusses an architecture that allows changes to the Grid configuration to be automated in response to operator input or sensors placed throughout the Grid based on principles of autonomic computing [18]. Similarly to Hill et al. work discussed in Vambenepe et al. addresses global service management based on principles of autonomic computing [33]. Vu et al. present an extensible and customizable framework for the autonomous discovery of semantic Web services based on their QoS properties [35]. Condor's ClassAds mechanism is used to represent jobs, resources, submitters and other Condor daemons [31].

However, to the best of our knowledge none of the discussed approaches deals with user-driven and semi-automatic definition of SLA mappings enabling negotiations between inconsistent SLA templates. Also, none of the presented approaches address *meta-negotiations (MN)* where participating parties may agree on a specific negotiation protocol, security standards or other negotiation pre-requisites.

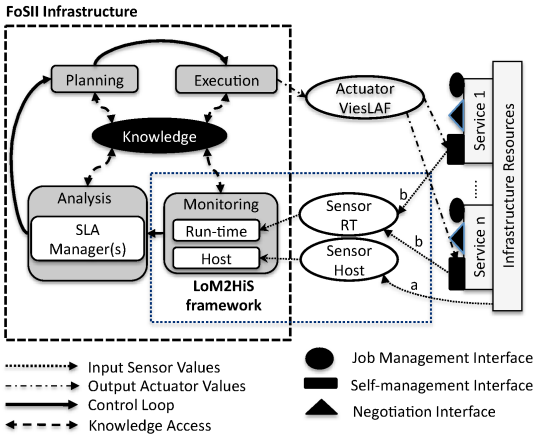


FIG. 3.1. *FoSII infrastructure*

3. Adaptable, Versatile, and Dynamic services. In this section we discuss how service mediation and negotiation bootstrapping can be realized using the concepts of autonomic computing [19]. First, we introduce the *Foundations of Self-governing ICT Infrastructures (FoSII)* project (Section 3.1). Thereafter we discuss how the *VieSLAF* architecture contributes to the implementation of *FoSII* goals (Section 3.2).

3.1. FoSII infrastructure. To facilitate dynamic, versatile, and adaptive IT infrastructures, SOA systems should react to environmental changes, software failures, and other events which may influence the systems' behavior. Therefore, adaptive systems exploiting self-* properties (self-healing, self-controlling, self-managing, etc.) are needed, where human intervention with the system is minimized. In *Foundations of Self-governing ICT Infrastructures (FoSII)* project we propose models and concepts for adaptive services, utilizing autonomic computing concepts [3, 19]. As shown in Figure 3.1 the *FoSII* infrastructure is used to manage the whole Monitoring, Analysis, Planning and Execution (MAPE) lifecycle of self-adaptable Cloud services [5]. Each FoSII service implements three interfaces: (i) negotiation interface necessary for the establishment of SLA agreements, (ii) job-management interface necessary to start the job, upload data, and similar job management actions, and (iii) the self-management interface necessary to devise actions in order to prevent SLA violations.

The self-management interface shown in Figure 3.1 is implemented by each Cloud service and specifies operations for sensing changes of the desired state and for reacting to those changes. The host monitor sensors continuously monitor the infrastructure resource metrics (input sensor values arrow a in Figure 3.1) and provide the autonomic manager with the current resource status. The run-time monitor sensors sense future SLA violation threats (input sensor values arrow b in Figure 3.1) based on resource usage experiences and predefined threat thresholds. The treat thresholds should be retrieved by the knowledge management systems as described later on in the paper. The mapping between the sensed host values and the values of the SLA parameters is described next.

As shown in Figure 3.1 we distinguish between *host monitor* and *runtime monitor*. Resources are monitored by *host monitor* using arbitrary monitoring tools (e.g. Ganglia). Thus, resources metrics include e.g., down-time, up-time, available in and out bandwidth. Based on the predefined mapping rules stored in a database monitored metrics are periodically mapped to the SLA parameters. An example SLA parameter is service availability Av, which is calculated using the resource metrics *downtime* and *uptime* and the according mapping rule looks like the following one:

$$Av = 1 - downtime/uptime \tag{3.1}$$

The mapping rules are defined by the provider using appropriate Domain Specific Languages (DSL). These rules are used to compose, aggregate, or convert the low-level metrics to form the high-level SLA parameter including mappings at different complexity levels e.g., $1 : n$ or $n : m$. Thus, calculated SLA values are compared with the predefined threat threshold in order to react before SLA violations happen. The concept of detecting future SLA violation threats is designed by defining a more restrictive threshold than the SLA violation threshold known as threat threshold. Generation of the *threat threshold* is far from trivial and should be defined and mananaged by the FoSII's knowledge management system.

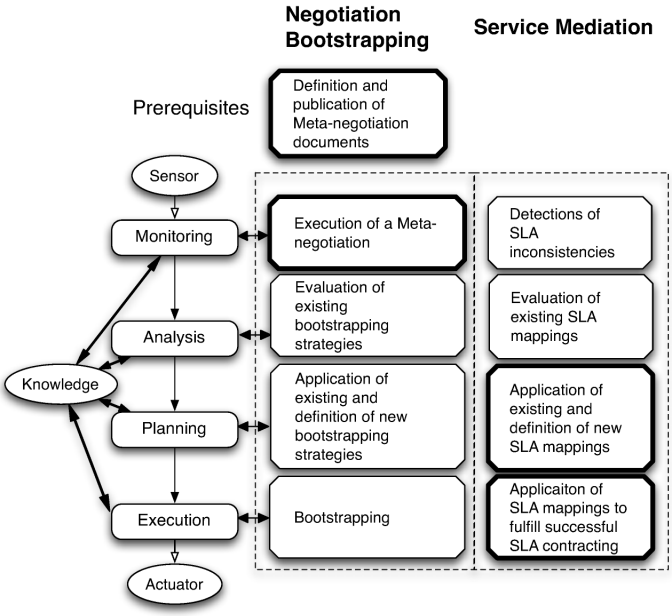


FIG. 3.2. *Negotiation Bootstrapping and Service Mediation as Part of the Autonomic Process*

As described in [11] we implemented a highly scalable framework for mapping of Low-level Resource Metrics to High Level SLA parameters *LoM2HiS framework* facilitating exchange of large numbers of messages. We designed and implemented a communication model based on the Java Messaging Service (JMS) API, which is a Java Message Oriented Middleware (MOM) API for sending messages between two or more clients. We use Apache ActiveMQ as a JMS provider that manages the sessions and queues.

As shown in Figure 3.1 *VieSLAF* framework represents an actuator mediating between inconsistent SLA templates and bootstrapping between different protocols. In the following we discuss how the *VieSLAF* contributes to the implementation of the MAPE cycle in self-adaptable Cloud services considering service negotiation phase.

3.2. Negotiation Bootstrapping and Service Mediation. Figure 3.2 depicts how the principles of autonomic computing can be applied to negotiation bootstrapping and service mediation. As a prerequisite of the negotiation bootstrapping users have to specify meta-negotiation document describing the requirements of a negotiation, as for example required negotiation protocols, required security infrastructure, provided document specification languages, etc. During the *monitorig phase* all candidate services are detected which need negotiation bootstrapping, e.g. which do not have matching negotiation protocol with the potential consumer. During the *analysis phase* existing knowledge base is queried and potential bootstrapping strategies are found. In case of missing bootstrapping strategies users can define in a semi-automatic way new strategies (*planning phase*). Finally, during the *execution phase* the negotiation is started by utilizing appropriate bootstrapping strategies.

The same procedure can be applied to service mediation. During the service negotiation inconsistencies in SLA templates may be discovered (*monitoring phase*). During the *analysis phase* existing SLA mappings are analyzed. During the *planning phase* new SLA mappings can be defined, if existing mappings cannot be applied. Finally, during the *execution phase* the newly defined SLA mappings can be applied.

As indicated with bold borders in Figure 3.2, in this paper we present solutions for the definition and accomplishment of meta-negotiations (Section 5) and for the specification and applications of SLA mappings (Section 4) as described next.

4. Service Mediation with SLA Mappings. In the presented approach each SLA template has to be published into a registry where negotiation partners i. e., provider and consumer, can find each other. The management of SLA mappings and published services is presented in Section 4.1. The transformations between remote and local SLA templates are discussed in Section 4.2. Finally, an example SLA mapping document is presented in Section 4.3.

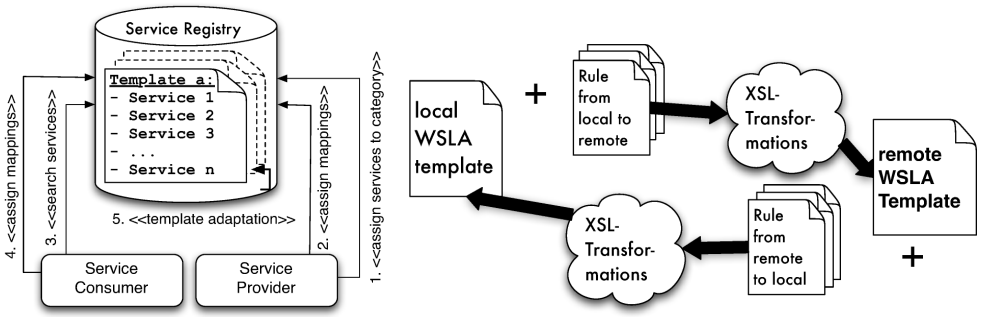


FIG. 4.1. *(a) Management of SLA-Mappings (b) QoS basic scenario*

4.1. Management of SLA Mappings. Figure 4.1(a) depicts the architecture for the management of SLA mappings and participating parties. The registry comprises different *SLA templates* whereby each of them represents a specific application domain e.g., SLA templates for the medical, telco or life science domain. Thus, each service provider may assign his/her service to a particular template (see step 1 in Figure 4.1(a)) and afterwards assign SLA mappings, if necessary (see step 2). Each template a may have n services assigned. Available templates can be browsed using an appropriate GUI.

Service consumers may search for the services using meta-data and search terms (step 3). After finding appropriate services each service consumer may define mappings to the associated template (step 4). Thereafter, the negotiation between service consumer and service provider may start as described in the next section. SLA mappings should be defined in a dynamic way. Thus, SLA templates can be updated frequently to reflect the actual SLAs used by service provides and consumers based on predefined adaptation rules (step 5). The adaptability functionality facilitates the generation of user driven public SLA templates.

4.2. SLA-Mappings Transformations. Figure 4.1(b) depicts a scenario for defining XSL transformations. As the SLA specification language we use Web Service Level Agreements (WSLAs) [37]. We also developed first bootstrapping strategies for communication across different SLA specification languages [6].

Templates are publicly available and published in a searchable registry. Each participant may download already published templates and compare it in a semi-automated or automated way with the local template. If there are any inconsistencies discovered, the service consumer may write rules (XSL transformation) from his/her local SLA template to the remote template. The rules can also be written by using appropriate visualization tools, for example using a GUI as depicted in Figure 6.1. Thereafter, the rules are stored in the database and can be applied during the runtime to the remote template. Since during the negotiation process transformations are done in two directions, the transformations from the remote SLA template to the local template are necessary as well.

As depicted in Figure 4.1(b), a service consumer is generating an SLA. The locally generated SLA plus the rules defining transformations from local SLA to remote SLA deliver an SLA which is complaint to the remote SLA. In the second case the remote template has to be translated into the local one. In that case the remote SLA plus the rules defining transformations from the remote to local SLA deliver an SLA which is compliant to the local SLA. Thus, the negotiation may be done between non-matching SLAs in both directions: from service consumer to service provider and vice versa.

The service provider can define rules for XSL transformations in the same way as depicted in Figure 4.1(b) from the publicly published SLA templates to the local templates. Thus, both parties, provider and consumer, may match on a publicly available SLA template.

4.3. SLA-Mappings Document (SMD). Figure 4.2 shows a sample rule for XSL transformations where price defined in Euros is transformed to an equivalent price in US Dollars. Please note that for the case of simplicity we use a relatively simple example. Using XSLT more complicated mappings can also be defined. Explanation of this is out of scope of this paper.

As shown in Figure 4.2, the Euro metric is mapped to the Dollar metric. In this example we define the mapping rule returning Dollars by using the *Times* function of *WSLA Specification* (see line 4). The *Times*

```
1.~\dots
2. <xsl:template \dots >
3.  <xsl:element name="Function" \dots >
4.   <xsl:attribute name="type"> <xsl:text>Times</xsl:text> </xsl:attribute>
5.   <xsl:attribute name="resultType"> <xsl:text>double</xsl:text> </xsl:attribute>
6.  <xsl:element name="Operand" \dots >
7.   <xsl:copy> <xsl:copy-of select="@*|node()"/> </xsl:copy>
8.  </xsl:element>
9.  <xsl:element name="Operand" \dots >
10.  <xsl:element name="FloatScalar" \dots > <xsl:text>1.27559</xsl:text> </xsl:element>
11. </xsl:element>
12. </xsl:element>
13.</xsl:template>
14\dots .
```

FIG. 4.2. *Example XSL Transformation*

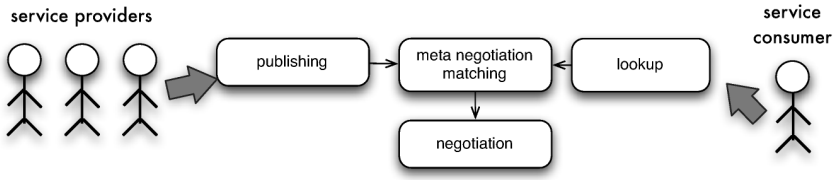


FIG. 5.1. *Meta-negotiation phases*

function multiplies two operands: the first operand is the Dollar amount as selected in line 7, the second operand is the Dollar/Euro quote (1.27559) as specified in line 10. The dollar/euro quote can be retrieved by a Web service and is usually not hard coded.

With similar mapping rules users can map simple syntax values (values of some attributes etc.), but they can even define complex semantic mappings with considerable logic. Thus, even syntactically and semantically different SLA templates can be translated into each other.

5. Negotiation Bootstrapping with Meta-negotiations. In this section, we present an example scenario for the meta-negotiation architecture and describe the document structure for publishing negotiation details into the meta-negotiation registry.

5.1. Scenario. As depicted in Figure 5.1, the meta-negotiation infrastructure can be employed in the following manner:

Publish. A service provider publishes descriptions and conditions of supported negotiation protocols into the registry (see Section 6).

Lookup. Service consumers perform lookup on the registry database by submitting their own documents describing the negotiations that they are looking for.

Match. The registry discovers service providers who support the negotiation processes that a consumer is interested in and returns the documents published by the service providers.

Negotiate. Finally, after an appropriate service provider and a negotiation protocol is selected by a consumer using his/her private selection strategy, negotiations between them may start according to the conditions specified in the provider's document.

Note that in this scenario, the consumer is looking for an appropriate service provider. The reverse may happen as well, wherein a consumer advertises a job or a task to be carried out and many providers bid to complete it. In such cases, the providers would perform the lookup.

5.2. Registry Document. The participants publishing into the registry follow a common document structure that makes it easy to discover matching documents. This document structure is presented in Figure 5.2 and consists of the following main sections. Each document is enclosed within the

```
<meta-negotiation>...</meta-negotiation>
```

tags. The document contains an `<entity>` elements defining contact information, organization and ID of the participant. The `<ID>` element defines the unique identifier given to the meta-negotiation document by the registry. The publisher can update or delete the document using the identifier. Each meta-negotiation

```
1.  <meta-negotiation
2.    xmlns:xsi="\dots ‘‘xsi:noNamespaceSchemaLocation="\dots ‘‘>
3.    <entity>
4.     <contact name="\dots ‘‘ phoneNumber="\dots ‘‘ />
5.     <organization name= ‘‘University of \dots ‘‘
6.~\dots
7.     <ID name="1234"/>
8.    </entity>
9.    <pre-requisite>
10.    <role name="consumer"/>
11.    <security> <authentication value="GSI location="uri"/> </security>
12.    <negotiation-terms>
13.     <negotiation-term name="beginTime"/>
14.     <negotiation-term name="endTime"/>
15.     <negotiation-term name="price"/>
16.    </negotiation-terms>
17.   </pre-requisite>
18.   <negotiation>
19.    <document name="WSLA" value="uri" version="1.0" />
20.     <document name="WS-Agreements" value="uri" version="1.0" />
21.     <protocol name="alternateOffers" schema="uri" version="1.0" location="uri"/>
22.   </negotiation>
23.   <agreement>
24.    <confirmation name="arbitrationService" value="uri"/>
25.   </agreement>
26. </meta-negotiation>
```

FIG. 5.2. *Example document for meta-negotiation registry*

comprises three distinguishing parts, namely *pre-requisites*, *negotiation* and *agreement* as described in the following paragraphs.

Pre-requisites. The conditions to be satisfied before negotiations are defined within the `<pre-requisite>` element (see Figure 5.2, lines 9–17). Pre-requisites define the *role* a participating party takes in a negotiation, the *security credentials* and the *negotiation terms*. The `<role>` element defines whether the specific party wants to engage in the negotiation as a provider or as a consumer of resources. The `<security>` element specifies the authentication and authorization mechanisms that the party wants to apply before starting the negotiation process. For example, in Figure 5.2, the consumer requires that the other party should be authenticated through the *Grid Security Infrastructure (GSI)* [15] (line 11). The negotiation terms specify QoS attributes that a party is willing to negotiate and are specified in the `<negotiation-term>` element. For example, in Figure 5.2, the negotiation terms of the consumer are *beginTime*, *endTime*, and *price* (lines 13–15).

Negotiation. Details about the negotiation process are defined within the `<negotiation>` element. In Figure 5.2, the consumer supports two document languages and one negotiation protocol. Each document language is specified within `<document>` element. In Figure 5.2, *WSLA* and *WS-Agreements* are specified as supported document languages. Additional attributes specify the *URI* (Uniform Resource Indicator) to the API or WSDL for the documents and their versions supported by the consumer (lines 18–22). In Figure 5.2, *AlternateOffers* is specified as the supported negotiation protocol. In addition to the *name*, *version*, and *schema* attributes, the URI to the WSDL or API of the negotiation protocols is specified by the *location* attribute (line 21).

Agreement. Once the negotiation has concluded and if both parties agree to the terms, then they have to sign an agreement. This agreement may be verified by a third party organization or may be lodged with another institution who will also arbitrate in case of a dispute. These modalities are specified within the `<agreement>` clause of the meta-negotiation document. For example, in Figure 5.2, a third party service, called *arbitrationService*, is specified for confirming the agreement between the two parties.

6. VieSLAF framework. In this section we present the architecture used for the semi-automatic management of *meta-negotiations* and *SLA mappings*. We discuss a sample architectural case study exemplifying the usage of VieSLAF. Thereafter, we describe each *VieSLAF*'s core component in detail.

6.1. VieSLAF architecture. As discussed in Section 3 *VieSLAF* framework represents the first prototype for the management of self-governing ICT Infrastructures. The *VieSLAF* framework enables application developers to efficiently develop adaptable service-oriented applications simplifying the handling with numerous Web service specifications. The framework facilitates management of QoS models as for example management of meta-negotiations [8] and SLA mappings [7]. Based on *VieSLAF* framework service provider may easily manage

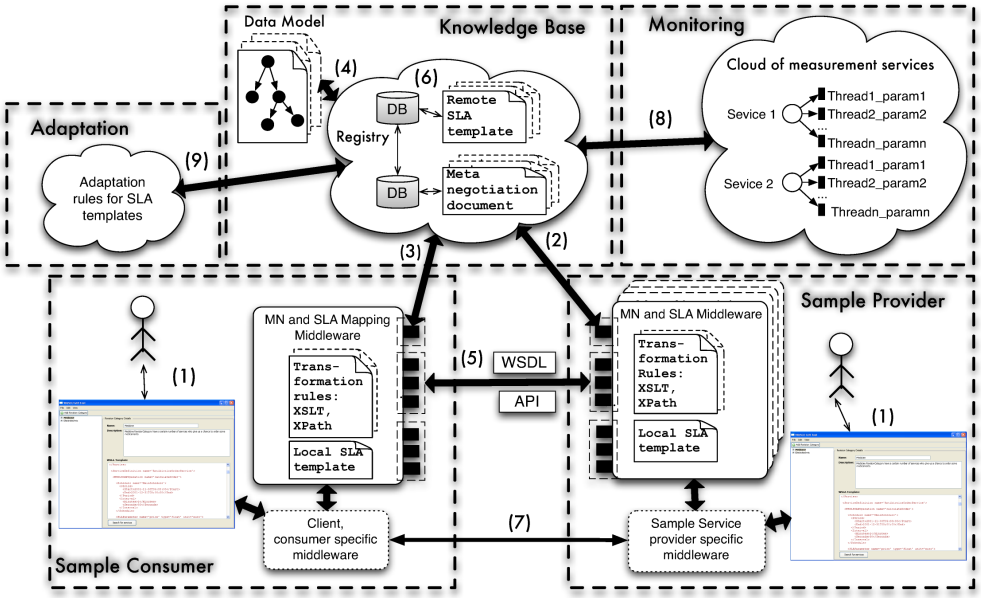


FIG. 6.1. *VieSLAF Architecture*

QoS models and SLA templates and frequently check whether selected services satisfy developer's needs e.g., specified QoS-parameters in SLAs. Furthermore, we discuss basic ideas about the adaptation of SLA templates.

We describe the *VieSLAF* components based on Figure 6.1. As shown in step (1) in Figure 6.1 users may access the registry using a GUI, browse through existing templates and meta-negotiation documents using the MN and SLA mapping middleware. In the next step (2), service provider specify MN documents and SLA mappings using the MN and SLA mapping middleware and submit it to the registry. Thereafter, in step (3), service consumer may query existing meta-negotiation documents, define own SLA mappings to remote templates. MN and SLA mapping middleware on both sides (provider's and consumer's) facilitate management of MNs and SLA mappings. Submitted MN documents and SLA mappings are parsed and mapped to a predefined data model (step 4). After meta-negotiation and preselection of services, service negotiation may start using the negotiation protocols, document languages, and security standards as specified in the MN document (step 5). During the negotiation SLA mappings and XSLT transformations are applied (step 6). After the negotiation, invocation of the service methods may start (step 7). SLA parameters are monitored using the monitoring service (step 8). Based on the submitted SLA mapping publicly available SLA templates are adapted reflecting the majority of local SLA templates (step 9).

6.1.1. Knowledge Base. As shown in Figure 6.1 *knowledge base* is responsible for storing SLA templates, SLA mappings and meta-negotiation documents. For storing of SLA templates and MN documents we implemented registries, representing searchable repositories. Currently, we implemented a MS-SQL 2008 database with a Web service front end that provides the interface for the management of SLA mappings and a PostgreSQL for the management of meta-negotiations. Thus, for scalability issues we rather intent to host the registries using a cloud of databases hosted on a service provider such as Google App Engine [14] or Amazon S3 [2]. The database is manipulated based on the role-model. The registry methods are implemented as Windows Communication Foundation (WCF) services and can be accessed only with the appropriate access rights. We define three roles: *service consumer*, *service provider* and *registry administrator*. *Service consumers* are able to search suitable services for the selected service categories e.g., by using the method *findServices*. Service consumer may also create SLA mappings using the method *createAttributeMapping*. *Service providers* may publish their services and bind it to a specific template category using the method *createService*. Furthermore, both service consumer and provider may submit and query MN documents.

6.2. Meta-negotiation Middleware. The *meta-negotiation middleware* facilitates publishing of the meta-negotiation documents into the registry and the integration of the meta-negotiation framework into the existing client and/or service infrastructure, including, for example, negotiation or security clients. Besides

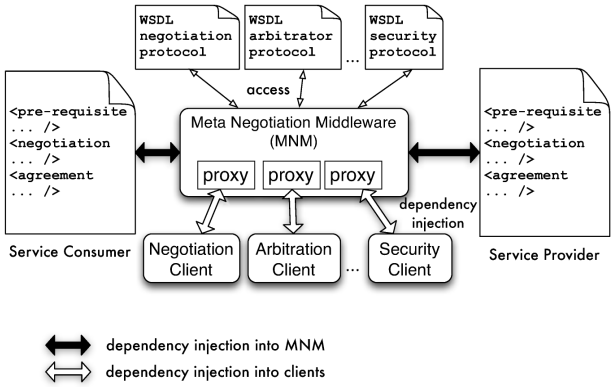


FIG. 6.2. *Meta-negotiation middleware*

acting as a client for publishing and querying meta-negotiation documents (steps 1 and 2 in Figure 6.1), the middleware delivers necessary information for the existing negotiation clients, i. e. information for the establishment of the negotiation sessions (step 4, Figure 6.1) and information necessary to start a negotiation (step 5 in Figure 6.1). As shown in Figure 6.1 each service consumer may negotiate with multiple service providers concurrently. As mentioned in Section 5 even the reverse may happen as well, wherein a consumer advertises a job. In such cases, the providers would negotiate with multiple consumers.

After querying the registry and applying a client-based strategy for the selection of the appropriate service, the information from the service's meta-negotiation document is parsed. Thereafter, meta-negotiation information is incorporated into the existing client software using a dependency injection framework such as Spring[1]. This dependency injection follows an Inversion of Control approach wherein the software is configured at runtime to invoke services that are discovered dynamically rather than known and referenced beforehand. This is suitable for meta-negotiation wherein a participant discovers others at runtime through the registry and has to dynamically adapt based on the interfaces provided by his counterpart (usually through a WSDL document).

Figure 6.2 shows an example of how this would work in practice. On the consumer side, the middleware queries the registry and obtains matching meta-negotiation documents. The middleware parses the meta-negotiation document of the selected provider and dynamically injects the interfaces discovered from the WSDLs in the document for security, negotiation and arbitration services into the existing abstract clients. Currently, we support semi-automatic integration of existing clients into meta-negotiation middleware wherein the existing clients are extended with the XML-based configuration files which are then automatically populated with the discovered interfaces.

6.2.1. SLA Mapping Middleware. As already mentioned in Section 6.1.1 SLA mapping middleware is based on different WCF services. For the sake of brevity, in the following we discuss just a few of them. The *RegistryAdministrationService* provides methods for the manipulation of the database where administrator rights are required e.g., creation of template categories. Another example represents *WSLAMappingService*, which is used for the management of SLA mappings by service consumer and service provider. *WSLAQueryingService* is used to query the SLA mapping database. The database can be queried based on template categories, SLA attributes and similar attributes. Other implemented WCF service are for example services for SLA parsing, XSL transformations, and SLA validation.

Service consumers may search for appropriate services through *WSLAQueryingService* and define appropriate SLA mappings by using the method *createAttributeMapping*. Each query request is checked during the runtime, if the service consumer has also specified any SLA mappings for *SLAElements* and *SLAAttributes* specified in the category's SLA template. Before the requests of service consumers can be completely checked, SLA transformations are applied. The rules necessary for the transformations of attributes and elements can be found in the database and can be applied using the consumer's SLA template. Thereafter, we have the consumer's template completely translated into category's SLA template. Transformations are done by *WSLATransforma-*

[1]http://www.springframework.org/

tor implemented with the .NET 3.5 technology and using LINQ[2]. In the following we explain monitoring and adaptation service in more detail.

Monitoring Service. As depicted in Figure 6.1, we implemented a lightweight concept for monitoring of SLA parameters for all services published in a specific template category. The aim of the monitoring service is to frequently check the status of the SLA parameters of an SLA agreement and deliver information to the service consumer and/or provider. Monitoring starts after publishing a service in a category and is provided through the whole lifetime of the service. Monitoring service is implemented as an internal registry service, similar to other services for parsing, transformation, and validation, that we have already explained in previous sections. Resources are monitored by *host monitor* using arbitrary monitoring tools (e.g. Ganglia). Resources metrics include e.g., down-time, up-time, available storage. Based on the predefined mappings stored in a database, monitored metrics are periodically mapped to the SLA parameters as described in [11].

After publishing service and SLA mappings, SLAs are parsed and it is identified which SLA parameters have to be monitored and how. We distinguish between periodically measured SLA parameters and the parameters which are measured on request. The values of the periodically measured parameters are stored in the so-called *parameter-pool*. The monitoring service provides two methods: a knock-in method for starting the monitoring and a method for receiving the measured SLA parameters from the measurement pool. Whenever a user requests monitoring information of the particular SLA (i) SLAs parameters are requested from the *parameter-pool* in case of periodically measured parameters or (ii) SLA parameters are immediately measured as defined in the parsed and validated SLAs in case of on-request parameters.

Adaptation Service. Remote SLA templates should not be defined in a static way, they should reflect provider's and consumer's needs. We implemented a first prototype of an internal registry's adaptation service. Thereby, mappings supplied by the consumers or the providers are evaluated. Based on the evaluation outcome a new version of the particular SLA template can be automatically defined.

Each SLA mapping can be defined as a *ParameterWish* (add/delete) and stored as an XML chunk. *Registry administrators* have to configure a learning capability property for each template category. Regression models represent one of the promising learning functions. Whenever a new *ParameterWish* is accepted a new revision category of an SLA template is generated. All services and consumers who voted for that specific *wish* are automatically re-published to the new revision. Also all SLA mappings are automatically assigned to the new template revision. Old SLA mappings of the consumers and services are deleted and also all old background threads used for calculation for old SLA template are aborted. The newly generated SLA template is thereafter parsed and new background monitoring threads are created and started for each service. Thus, based on the presented adaptation approach public templates can be derived in a user driven way reflecting the majority of local templates. Application of the learning functions is discussed in more detail in Section 7.3.2.

7. VieSLAF Evaluation and Case Studies. In this section we evaluate the *VieSLAF* framework. In Section 7.1 we evaluate SLA mappings. In Section we 7.2 we evaluate meta-negotiations. In Section 7.3 we report some successful VieSLAF case studies.

7.1. Evaluation of SLA mappings. In Section 7.1.1 we measure the overhead produced by SLA mappings compared to Web service invocation without mappings. We describe the experimental testbed and the setup used. Thereafter, we discuss the experimental results. In Section 7.1.2 we discuss stress tests with the varying number of concurrently invoked SLA mappings. In Section 7.1.3 we present results with the varying number of SLA mappings per single Web service invocation.

7.1.1. Overhead Test. In order to test the *VieSLAF* framework we developed a testbed as shown in Figure 7.1(a). As a client machine we used an Acer Aspire Laptop, Intel Core 2 Duo T5600 1.83 GHz, 2 MB L2 Cache, 1GB RAM. For hosting of 10 sample services, calculator services with 5 methods, we used a single core Xenon 3.2Ghz, L1 cache, 4GB RAM Sun blade machine. We use the same machine to host *VieSLAF*s WCF services. The aim of our evaluation is to measure the overhead produced using *VieSLAF*'s *WSLAQueryingService* for search and mappings of the appropriate services.

We created 10 services (S1,. . . , S10) and 10 accounts for service providers. We also created the registry administrator's role, which manages the creation of template categories with the corresponding SLA templates. The SLA template represents a remote calculator service with five methods: *Add, Subtract, Multiply, Divide and Max*. Both, the provider and the consumers define five *SLAMappings*, which have to be used during the

[2]Language Integrated Query

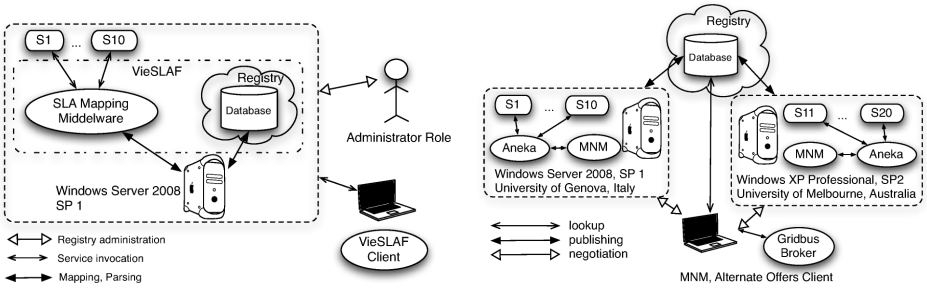


FIG. 7.1. *VieSLAF Testbed (a) for the evaluation of SLA mappings and (b) meta-negotiations*

TABLE 7.1
SLA Mappings Overhead Compared to Simple Web Service Invocation (Without SLA Mappings)

	Service Search Time				**Total**
	SLA-Mapping			**Remaining Time**	
	Validation	Consumer Map.	Provider Map.		
Time in sec	0.046	0.183	0.151	1.009	1.389
Time [%]	3.32	13.17	10.87	72.64	100.00

runtime. We specify three simple, syntactic mappings where we only change the name of an element or attribute. The other two mappings consider also semantic mappings, where we map between structurally different SLA templates.

Table 7.1 shows the experimental results. The measured values represent the arithmetic mean of 20 service invocations. The overhead measured during the experimental results includes the time needed for validation of SLA documents (column *Validation* in Table 7.1), the time necessary to perform mappings from the local consumers to the remote SLA templates (column *Consumer Mapping*) and the time necessary to transform the remote SLA templates to the local providers (column *Provider Mapping*). Furthermore, we measured the *remaining time* necessary to perform a search. The remaining time includes the round trip time for a search including data transfer between the client and the service and vise versa. As shown in Table 7.1 the time necessary to handle SLA mappings ($Validation + ConsumerMapping + ProviderMapping$) represents 0.38 seconds or $27,36\%$ of the overall search time.

Please note that the intention of the presented experimental results is the proof of concept of the SLA mapping approach. We did not test the scalability issues, since we intend to employ computing Clouds like Google App Engine [14] or Amazon S3 [2] in order to cope with the scalability issues.

7.1.2. Stress Tests. In this section we describe tests on how the *VieSLAF* middleware copes with the multiple SLA mappings executed concurrently with differing complexity. Evaluation is done on an Acer Aspire Laptop, Intel Core 2 Duo T5600 1.83 GHz, 2 MB L2 Cache, 1GB RAM. For the evaluation we have used two different SLA mappings:

- Simple: Invocation of the simple SLA mappings, an example is translation of one attribute to another attribute e.g., *usage price* to *price*.
- Complex: Represents the invocation of the complex SLA mappings, as for example semantic mappings considering two structurally different SLA templates.

We tested *VieSLAF* with different versions of XSLT transformers, namely with *XSLTCompiledTransform*, .Net version 3.0 and with the obsolete *XSLTTransform Class* from .Net 1.1. Figure 7.2(a) shows the measurements with the *XSLTCompiledTransform* Transformer and with the *XSLTTransform* Class. The x axis depicts the number of SLA mappings performed concurrently i. e., number of runs. The y axis depicts the measured time for the execution of SLA mappings in seconds.

Considering the measurement results we can observe that the *XSLTTransform* Class is faster than the *XSLTCompiledTransform* Transformer from the newer .Net version. Complex mappings executed with the *XSLTTransform* Class almost overlap with the simple mappings executed with the *XSLTCompiledTransform*.

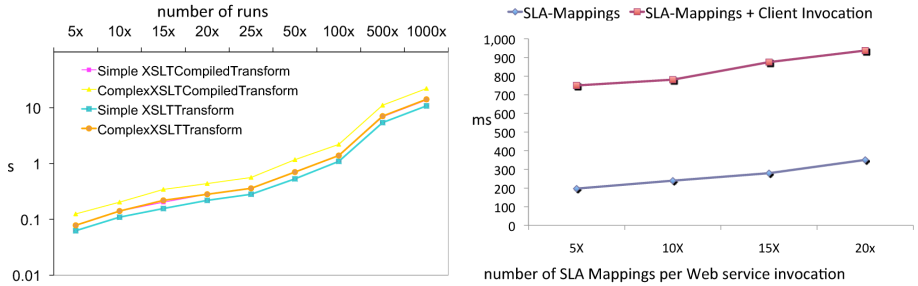


FIG. 7.2. *(a)Stress Tests with* XSLTCompiledTransform *Transformer and* XSLTTransform Class *(b) Measurements with varying number of SLA mappings per Web Service Invocation*

We can observe that in both cases, simple and complex mapping, the performance starts to significantly decrease with the number of SLA mappings > 100. If the number of mappings < 100, the execution time is about or less than 1 second.

7.1.3. Multiple SLA Mapping Tests. In this section we discuss performance results measured during a Web service call with varying numbers of SLA mappings per service. We measured 5, 10, 15 and 20 SLA mappings per Web service call. In order to create a realistic testbed we used SLA mappings which depend on each other: e.g., attribute A is transformed to attribute B, B is transformed to C, C to D, and so on. Thus, we simulate the worst case, where SLA mappings can not be performed concurrently, they have to be performed sequentially.

Evaluation is done on an Acer Aspire Laptop, Intel Core 2 Duo T5600 1.83 GHz, 2 MB L2 Cache, 1GB RAM. Figure 7.2(b) shows measured results. The x axis depicts the number of SLA mappings performed concurrently or sequentially considering attribute dependencies. The y axis depicts the measured time for the execution of SLA mappings in milliseconds. We executed SLA mappings between the remote template and the provider's template (i. e., provider mappings as described in Table 7.1) before the runtime, because these mappings are known before consumer requests. Thus, only mappings between the consumer's template and the remote template are done during the runtime as indicated with the *SLA Mapping* line. The line *SLA Mapping + Client invocation* comprises the time for the invocation of a Web service method including SLA mapping time. The *SLA Mapping + Client invocation* line does not comprise round-trip time, it comprises only the request time.

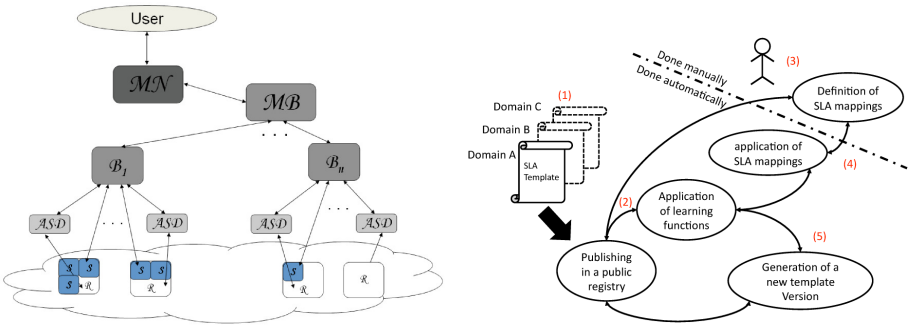
We can conclude that even with the increasing number of SLA mappings and considering the worst case scenario with sequentially performed mappings the SLA mapping time represents about 20% of the overall execution time.

7.2. Evaluation of the Meta-Negotiation Approach. In this section we evaluate the meta-negotiation approach as shown in Figure 7.1(b). We have used the Gridbus broker [32] as an example service consumer and an enterprise Grid constructed using Aneka [10] as a service provider. The aim of this evaluation was to test the overhead of the meta-negotiation infrastructure on the overall negotiation process.

7.2.1. Testbed. As shown in Figure 7.1, we deployed the registry in a machine running Windows Server 2003. The registry was accessible through a Web service interface and used a PostgreSQL database on its backend. Since the aim of these experiments was only to test the meta-negotiation framework, we isolated the Negotiation Service from the resource management system. Hence, it would reject any proposal for node reservation as it would not be able to determine node availability. We deployed 20 such services—*(S1,...S10)* on machines in a student lab in the Department of Computer Science and Software Engineering, University of Melbourne, Australia and *(S11,...S20)* on machines in the Department of Communication Computer and System Sciences, University of Genova, Italy. Negotiations with services located in Melbourne would terminate in single rounds (a proposal followed by a rejection). Services located in Italy would terminate after 2 retries. We published a meta-negotiation document for each service into the registry with different negotiation terms and document languages. The Gridbus broker was started on a machine in the Department of Computer Science, University of Melbourne and queried the registry in order to select an appropriate service provider. It would then open a negotiation process with the selected Aneka Negotiation Service.

TABLE 7.2
Experimental results of the meta negotiation approach

	Overall Negotiation			Total
	Meta-Negotiation		Negotiation	
	Querying	Parsing		
Time in sec	2.91	0.02	15.10	18.03
Time [%]	16.16	0.01	83.73	100.00



(a) (b)

FIG. 7.3. *(a) Architecture for the SLA-based Resource Virtualization [20] (b) Lifecycle of the SLA Template as used in [29]*

7.2.2. Experimental Results. The results of our evaluation are shown in Table 7.2. As shown in Table 7.2 the time necessary to query the registry represents 2.91 seconds or 16.16% of the overall negotiation time. Query time is calculated as the time necessary to get the list of the IDs, i. e. invocation of the method *query(XMLdocument)*, plus the time necessary to fetch each document, i. e. multiple invocations of the method *getDocument(ID)*. The time necessary to fetch each document represents about 0.2 sec. Thus, in our experiments we fetched about 15 XML documents in average, since $2.91/0.2 \approx 15$. Please note, that all times used in Table 7.1 are average times measured over 10 rounds. Time necessary to parse the selected meta-negotiation document and to inject the WSDL information into the client is 0.02 seconds or 0.01% of the overall negotiation time. Thus, time for the completion of the meta-negotiation is 2.93 seconds or 16.17% of the overall negotiation time. The time for the meta-negotiation is calculated as the the sum of the time necessary to query the registry (2.91 seconds) and the time necessary to parse the selected meta document (0.02 seconds).

The time necessary to negotiate with an *Aneka* service represents 15.10 seconds or 83.73% of the overall negotiation time. We observed that the negotiation time with services located in Italy represents about 15 seconds (due to 2 retries), since the time necessary to negotiate with services located in Melbourne represents about 5 seconds. Thus, in our experiments we have obviously negotiated only with services located in Italy. We started an alternate offers negotiation with only one round. Thus, the overall negotiation time is 18.03 seconds. Overall negotiation time is calculated as the sum of the time necessary to complete the meta-negotiation and time necessary to complete the negotiation.

Considering the fact that the time necessary to complete a meta-negotiation represents only 16.17% of the overall negotiation time, and considering the fact that we have used negotiations with only one round, we can show that the overhead of the meta-negotiations do not significantly influence the overall negotiation time.

With the presented experiments we demonstrated the applicability of our approach to the proposed architecture. Since we plan to use computational clouds in the future, the intention of the presented experiments was not to test the scalability of our approach.

7.3. Case Studies. Besides FoSII project, which was the primary reason for the development of the VieSLAF framework, VieSLAF has been successfully applied to additional case studies. In Section 7.3.1 we discuss SLA-based resource virtualization approach for on-demand service provision. In Section 7.3.2 we discuss the application of VieSLAF framework for the liquidity management in Cloud markets.

7.3.1. An SLA-based Resource Virtualization Approach For On-demand Service Provision. As discussed in [20] VieSLAF's meta-negotiation concept has been successfully utilized for the realization of the

SLA based resource virtualization environment. Thereby, an integrative infrastructure has been provided for on demand service provision based on SLAs. As depicted in Figure 7.3(a) users describe the requirements for an SLA negotiation on a high level using the concept of meta-negotiations (MN). During the meta-negotiation only those services are selected, which understand specific SLA document language and negotiation strategy or provide a specific security infrastructure. After the meta-negotiation process, a meta-broker (MB) selects a broker that is capable of deploying a service with the specified user requirements. Thereafter, the selected broker negotiates with virtual or physical resources (R) using the requested SLA document language and using the specified negotiation strategy. Once the SLA negotiation is concluded, service (S) can be deployed on the selected resource using the Automatic Service Dployer (ASD).

7.3.2. Liquidity Management in Cloud Markets. As discussed in [29] we demonstrated the problems caused in computational Cloud markets by a large number of resource definitions, namely low liquidity for each available resource type. To counteract this problem, we applied SLA mappings, which ensures sufficient liquidity in the market. SLA mapping techniques not only simplify the search for similar offers but also allow us to derive public SLA templates from all existing offerings (i. e. consumer-defined service level contracts or unsigned service level agreements). Figure 7.3(b) depicts the lifecycle of a public template. As indicated through step (1), we assume that for specific domains, specific SLA templates are generated, e.g. medicine, telecommunication. These generated SLA templates are then published in the public registry (step (2)). At the same time, learning functions for the adaptation of these public SLA templates are defined. Thereafter, SLA mappings are defined manually by users (step (3)). During the lifetime of an SLA template adaptation of SLA mappings are done automatically as described in Section 6 (step (4)). Based on the learning function and based on the submitted SLA mappings, a new version of the SLA template can be defined and published in the registry (step (5)).

8. Conclusion and Future Work. In this paper we presented the goals of the Foundations of Self-Governing ICT Infrastructures (FoSII) project and how these goals can be achieved using the principles of autonomic computing. We discussed novel meta-negotiation and SLA mapping solutions for Cloud services bridging the gap between current QoS models and Cloud middleware and representing important prerequisites for the establishment of autonomic Cloud services. We discussed the approaches for meta-negotiation and SLA mapping representing implementation of negotiation bootstrapping and service mediation approaches. Furthermore, we presented the *VieSLAF* framework used for the management of meta-negotiations and SLA mappings. We discussed how SLA templates can be adapted based on the submitted SLA mappings. We presented performance evaluation of the *VieSLAF* representing first proof of concepts. Moreover, we briefly introduced two case studies, namely the SLA-based resource virtualization approach for on-demand service provision and the approach for the liquidity management in Cloud markets. Both case studies showed the impact of the *VieSLAF* framework beyond the aforementioned FoSII project.

In the future we will improve learning function and facilitate different knowledge management methods as for example case based reasoning.

Acknowledgments. The work described in this paper was partially supported by the Vienna Science and Technology Fund (WWTF) under grant agreement ICT08-018 Foundations of Self-governing ICT Infrastructures (FoSII). The authors want to thank Vincent C. Emeakaroha for carefully proofreading the paper.

REFERENCES

[1] R. J. Al-Ali, O. F. Rana, D. W. Walker, S. Jha, and S. Sohail. *G-qosm: Grid service discovery using qos properties.* Computing and Informatics, 21:363–382, 2002.

[2] Amazon Elastic Compute Cloud (Amazon EC2), `http://aws.amazon.com/ec2/`

[3] D. Ardagna, G. Giunta, N. Ingraffia, R. Mirandola, and B. Pernici. *QoS-Driven Web Services Selection in Autonomic Grid Environments.* Grid Computing, High Performance and Distributed Applications (GADA) 2006 Internat. Conference, Montpellier, France, Oct 29 - Nov 3, 2006.

[4] D. Bein, A. K. Datta, S. Yellenki.A Link-Cluster Route Discovery Protocol For ad hoc Networks. Scalable Computing: Practice and Experience Volume 9, Number 1, pp. 21–28, 2008.

[5] I. Brandic. *Towards Self-manageable Cloud Services.* RTSOAA 2009, in conjunction with the 33rd Annual IEEE International Computer Software and Applications Conference, Seattle, USA, July 2009.

[6] I. Brandic, D. Music, S. Dustdar. *Service Mediation and Negotiation Bootstrapping as First Achievements Towards Self-adaptable Grid and Cloud Services.* Grids meet Autonomic Computing Workshop 2009 - GMAC09. In conjunction with the 6th International Conference on Autonomic Computing and Communications Barcelona, Spain, June 15–19, 2009.

[7] I. Brandic, D. Music, Ph. Leitner, S. Dustdar. VieSLAF Framework: Enabling Adaptive and Versatile SLA-Management. The 6th International Workshop on Grid Economics and Business Models 2009 (Gecon09). In conjunction with Euro-Par 2009, 25–28 August 2009, Delft, The Netherlands

[8] I. Brandic, S. Venugopal, Michael Mattess, and Raykumar Buyya. *Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services*. Workshop on Service-Oriented Engineering and Optimizations 2008. In conjunction with International Conference on High Performance Computing 2008 (HiPC 2008), Bangalore, India, December 17–20, 2008.

[9] R. Buyya, Ch. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. *Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility*. Future Generation Computer Systems, 25(6):599-616, June 2009.

[10] X. Chu, K. Nadiminti, Ch. Jin, S. Venugopal, and R. Buyya *Aneka: Next-Generation Enterprise Grid Platform for e-Science and e-Business Applications*. Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing (e-Science 2007), Dec. 10-13, 2007, Bangalore, India.

[11] V. C. Emeakaroha, I.Brandic, M. Maurer, S. Dustdar. *Low Level Metrics to High Level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in Cloud environments*. The 2010 High Performance Computing and Simulation Conference (HPCS 2010) June 28—July 2, 2010, Caen, France. *to appear*.

[12] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. *Web services on demand: WSLA-driven automated management*. IBM Systems Journal, 43(1), 2004.

[13] G. Dobson, A. Sanchez-Macian. *Towards Unified QoS/SLA Ontologies*. Proceedings of the 2006 IEEE Services Computing Workshops (SCW 2006), Chicago, Illinois, USA, 18-22 September 2006.

[14] Google App Engine, `http://code.google.com/appengine`

[15] I. Foster, and C. Kesselman, and G. Tsudik, and S. Tuecke. *A Security Architecture for Computational Grids*, Proc. 5th ACM Conference on Computer and Communications Security Conference, San Francisco, CA, USA, ACM Press, New York, USA, 1998.

[16] Foundations of Self-Governing ICT Infrastructures (FoSII) Project, `http://www.wwtf.at/projects/research_projects/details/index.php?PKEY=972_DE_0`

[17] P. C. K. Hung, L. Haifei, and J. Jun-Jang. *WS-Negotiation: an overview of research issues*. Proceedings of the 37th Annual Hawaii International Conference on System Sciences, Big Island, Hawaii, 5-8 January 2004.

[18] Z. Hill, J. C. Rowanhill, A. Nguyen-Tuong, G. S. Wasson, J. C. Knight, J. Basney, M. Humphrey. *Meeting virtual organization performance goals through adaptive grid reconfiguration*. 8th IEEE/ACM International Conference on Grid Computing (Grid 2007), Austin, Texas, USA, September 19-21, 2007.

[19] J. O. Kephart, D.M. Chess, *The vision of autonomic computing*. Computer, 36:(1) pp. 41-50, Jan 2003.

[20] A. Kertesz, G. Kecskemeti, I. Brandic. *An SLA-based Resource Virtualization Approach for On-demand Service Provision*. VTDC 2009, In conjunction with the 6th International Conference on Autonomic Computing and Communications Barcelona, Spain, June 15–19, 2009.

[21] B. Koller, L. Schubert. *Towards autonomous SLA management using a proxy-like approach*. Multiagent Grid Syst. 3(3), 2007, IOS Press, Amsterdam, The Netherlands, The Netherlands.

[22] S. König, S. Hudert, T. Eymann, and M. Paolucci. *Towards Reputation Enhanced Electronic Negotiations for Service Oriented Computing*. In Proceedings of the 11th International Workshop on Trust in Agent Societies (TRUST 2008), Estoril, Portugal, May 12-13, 2008.

[23] Z. Li, M. Parashar: An Infrastructure for Dynamic Composition of Grid Services. 7th IEEE/ACM International Conference on Grid Computing (Grid 2006), Barcelona, Spain, September 28-29, 2006.

[24] D. Nurmi, R. Wolski, Ch. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, D. Zagorodnov. *The Eucalyptus Open-source Cloud-computing System*. Proceedings of Cloud Computing and Its Applications 2008, Chicago, Illinois, October 2008.

[25] D. M. Quan, J. Altmann. *Resource allocation algorithm for light communication grid-based workflows within an SLA context*. International Journal of Parallel, Emergent and Distributed Systems (IJPEDS), 24(1):31-48, 2009.

[26] N. Oldham, K. Verma, A. P. Sheth, and F. Hakimpour. *Semantic WS-agreement partner selection*. Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, UK, May 23-26, 2006.

[27] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. *A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing*. in Proceedings of the 2005 European Grid Computing Conference (EGC 2005), Amsterdam, The Netherlands, February, 2005.

[28] M.P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges, IEEE Computer, 40(11): 64-71, November 2007

[29] Marcel Risch, Ivona Brandic, Jörn Altmann. Using SLA Mapping to Increase Market Liquidity. Workshop on Non Functional Properties and Service Level Agreements Management in Service Oriented Computing Workshop (NFPSLAM-SOC'09).The 7th International Joint Conference on Service Oriented Computing, November 23-27 2009, Stockholm, Sweden.

[30] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. M. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. B.-Y., W. Emmerich, F. Galan. *The RESERVOIR Model and Architecture for Open Federated Cloud Computing.*, IBM Journal of Research and Development, 53(4) (2009)

[31] D. Thain, T. Tannenbaum, and M. Livny. *Distributed Computing in Practice: The Condor Experience*. Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[32] S. Venugopal, R. Buyya, and L. Winton, *A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids*, Concurrency and Computation: Practice and Experience, 18(6): 685-699, Wiley Press, New York, USA, May 2006.

[33] W. Vambenepe, C. Thompson, V. Talwar, S. Rafaeli, B. Murray, D. S. Milojicic, S. Iyer, K. I. Farkas, M. F. Arlitt. *Dealing with Scale and Adaptation of Global Web Services Management*. International Journal of Web Services Research, 4(3): 65-84, 2007.

[34] D. W. Walker, L. Huang, O. F. Rana, and Y. Huang. *Dynamic service selection in workflows using performance data*. Scientific Programming 15(4):235-247, 2007.

[35] L.-H. Vu, F. Porto, K. Aberer, M. Hauswirth. *An Extensible and Personalized Approach to QoS-enabled Service Discovery.*,

Eleventh International Database Engineering and Applications Symposium (IDEAS 2007), Banff, Alberta, Canada, September 6-8, 2007.
[36] Web Services Agreement Specification (WS-Agreement), http://www.ogf.org/documents/GFD.107.pdf
[37] Web Service Level Agreement (WSLA), http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf
[38] Ch. Zhou, L. T. Chia, and B. S. Lee. *Semantics in service discovery and QoS measurement.* IT Professional, 7(2): 29–34, Mar-Apr 2005.

Edited by: Marcin Paprzycki
Received: March 30, 2010
Accepted: June 09, 2010

Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 205–220. http://www.scpe.org
ISSN 1895-1767

LARGE SCALE PROBLEM SOLVING USING AUTOMATIC CODE GENERATION AND DISTRIBUTED VISUALIZATION

ANDREI HUTANU[†], ERIK SCHNETTER[†‡], WERNER BENGER[†], ELOISA BENTIVEGNA[†], ALEX CLARY[†§], PETER DIENER[†‡], JINGHUA GE[†], ROBERT KOOIMA[†¶], OLEG KOROBKIN[†‡], KEXI LIU[†¶], FRANK LÖFFLER[†], RAVI PARUCHURI[†], JIAN TAO[†], CORNELIUS TOOLE[†¶], ADAM YATES[†], AND GABRIELLE ALLEN[†¶]

Abstract. Scientific computation faces multiple scalability challenges in trying to take advantage of the latest generation compute, network and graphics hardware. We present a comprehensive approach to solving four important scalability challenges: programming productivity, scalability to large numbers of processors, I/O bandwidth, and interactive visualization of large data. We describe a scenario where our integrated system is applied in the field of numerical relativity. A solver for the governing Einstein equations is generated and executed on a large computational cluster; the simulation output is distributed onto a distributed data server, and finally visualized using distributed visualization methods and high-speed networks. A demonstration of this system was awarded first place in the IEEE SCALE 2009 Challenge.

Key words: computer algebra systems, high performance computing, distributed systems, high speed networks, problem solving environment, scientific visualization, data intensive applications

1. Introduction. We describe the motivation, design, and experimental experiences of an end-to-end system for large scale, interactive and collaborative numerical simulation and visualization that addresses a set of fundamental scalability challenges for real world applications. This system was awarded first place in the IEEE SCALE 2009 Challenge in Shanghai, China in May 2009.

Our system shows a single end-to-end application capable of scaling to a large number of processors and whose output can be visualized remotely by taking advantage of high speed networking capabilities and of GPU-based parallel graphics processing resources. The four scalability challenges that are addressed are described below (see Figure 1.1).

1.1. Programming productivity. Programming productivity has long been a concern in the computational science community: the ever-growing complexity of many scientific codes make the development and maintenance of many large scale scientific applications an intimidating task. Things get even worse when one is dealing with extremely complicated systems such as the Einstein equations which, when discretized, typically result in over 20 evolved variables and thousands of source terms. In addressing these issues, we present our latest work on generic methods for generating code that solves a set of coupled nonlinear partial differential equations using the *Kranc* code generation package [1]. Our work greatly benefits from the modular design of the *Cactus* framework [2, 3], which frees domain experts from lower level programming issues, i. e., parallelism, I/O, memory management, et cetera. In this collaborative problem-solving environment based on *Cactus* and *Kranc*, application developers, either software engineers or domain experts, can contribute to a code with their expertise, thus enhancing the overall programming productivity.

1.2. Scalability to large number of processors. With the advent of Roadrunner, the first supercomputer that broke the petaflop/s mark in year 2008, the petaflop era was officially entered. There are a great number of challenges to overcome in order to fully leverage this enormous computational power to be able to solve previously unattainable scientific problems. The most urgent of all is the design and development of highly scalable and efficient scientific applications. However, the ever-growing complexity in developing such efficient parallel software always leaves a gap for many application developers to cross. We need a bridge, a computational infrastructure, which does not only hide the hardware complexity, but also provides a user friendly interface for scientific application developers to speed up scientific discoveries. In our project, we used a highly efficient computational infrastructure that is based on the *Cactus* framework and the *Carpet* AMR library [4, 5, 6].

1.3. I/O bandwidth. We are faced with difficult challenges in moving data when dealing with large datasets, challenges that arise from I/O architecture, network protocols and hardware resources: I/O archi-

[†]Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803, USA
[‡]Department of Physics & Astronomy, Louisiana State University, Baton Rouge LA 70803, USA
[§]Department of Electrical & Computer Engineering, Louisiana State University, Baton Rouge LA 70803, USA
[¶]Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA

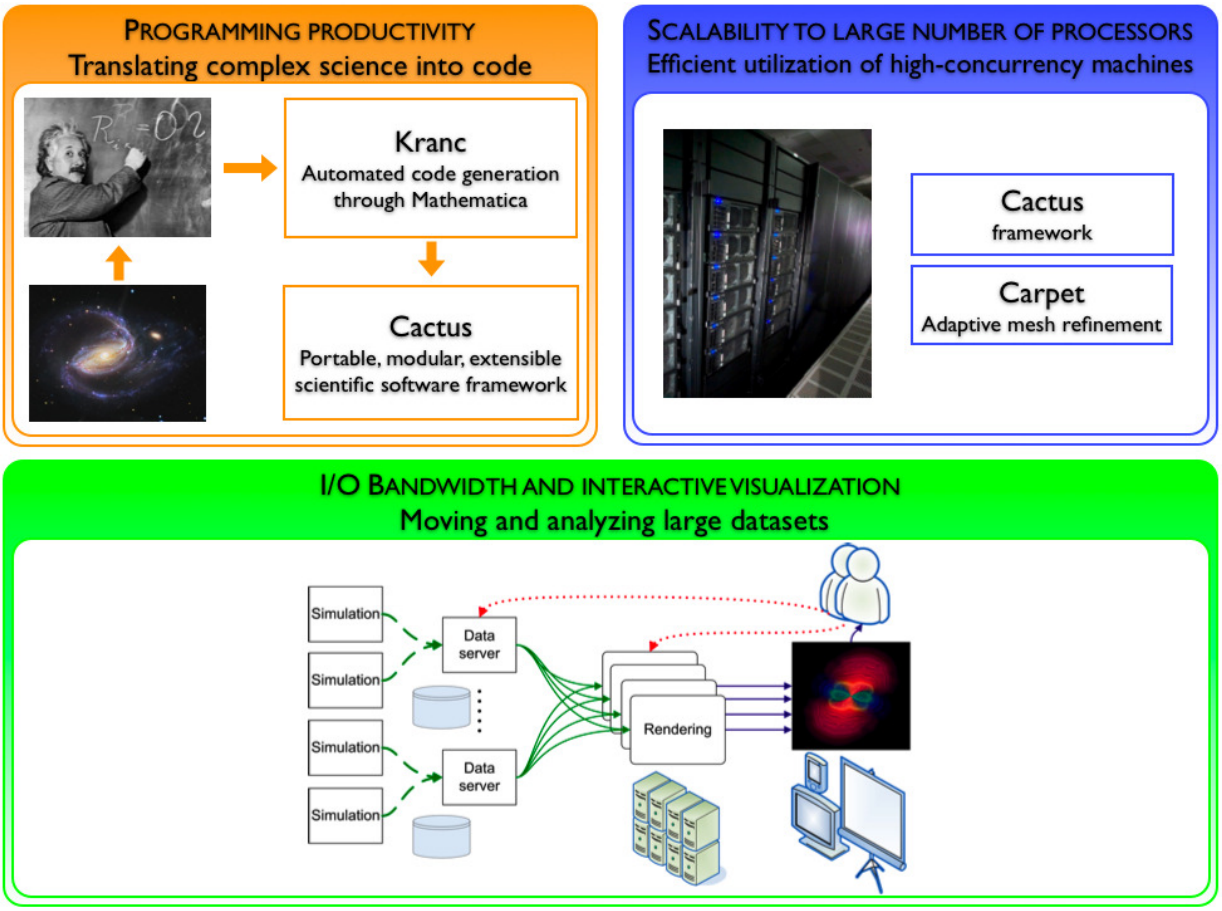


FIG. 1.1. *Scalability challenges involved in an end-to-end system for large scale, interactive numerical simulation and visualization for black hole modeling.*

tectures that do not use a non-blocking approach are fundamentally limiting the I/O performance; standard network protocols such as TCP cannot utilize the bandwidth available in emerging optical networks and cannot be used efficiently on wide-area networks; single disks or workstations are not able to saturate high-capacity network links. We propose a system that combines an efficient pipeline-based architecture, takes advantage of non-standard high-speed data transport protocols such as UDT, and uses distributed grid resources to increase the I/O throughput.

1.4. Interactive visualization of large data. Bringing efficient visualization and data analysis power to the end users' desktop while visualizing large data and maintain interactiveness, by giving the user the ability to control and steer the visualization, is a major challenge for visualization applications today. We are looking at the case where sufficiently powerful visualization resources are not available at either the location where the data was generated or at the location where the user is visualizing it, and propose using visualization clusters in the network to interactively visualize large amounts of data.

2. Scientific Motivation: Black Holes and Gamma-Ray Bursts. Over ninety years after Einstein first proposed his theory of General Relativity, astrophysicists are increasingly interested in studying the regions of the universe where gravity is very strong and the curvature of spacetime is large.

This realm of strong curvature is notoriously difficult to investigate with conventional observational astronomy. Some phenomena might not be observable in the electromagnetic spectrum at all, and may only be visible in the gravitational spectrum, i. e., via the gravitational waves that they emit, as predicted by General Relativity. Gravitational waves have today not yet been observed directly, but have attracted great attention

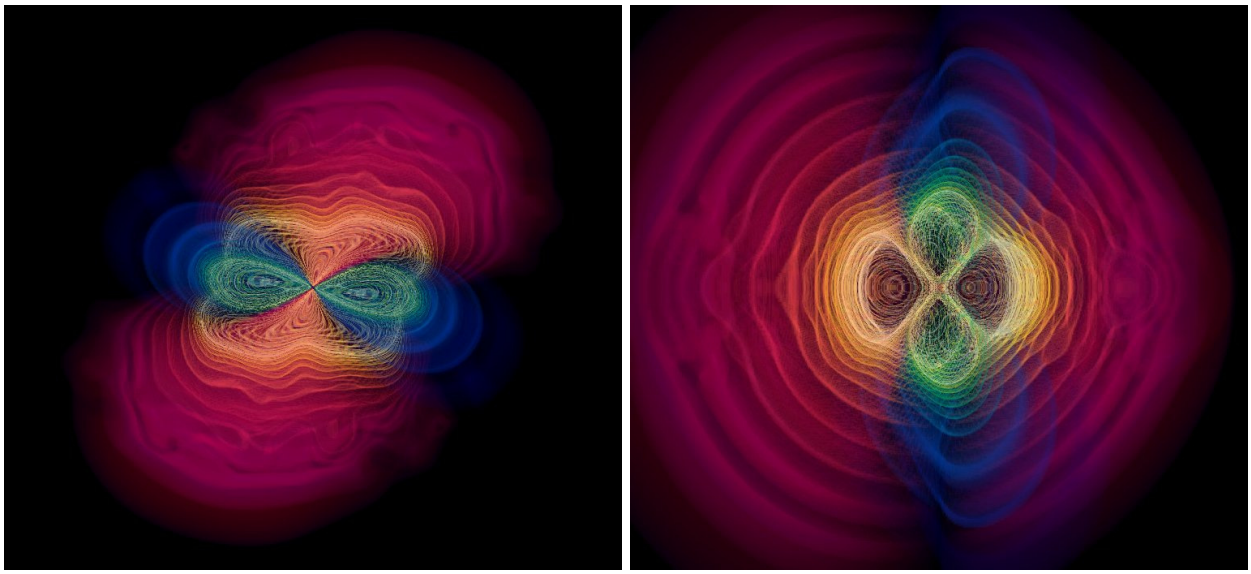


FIG. 2.1. *Volume rendering of the gravitational radiation during a binary black hole merger*

thanks to a wealth of indirect evidence [7, 8]; furthermore, gravitational wave detectors (LIGO [9], GEO [10], VIRGO [11]) will soon reach sufficient sensitivities to observe interesting astrophysical phenomena.

In order to correctly interpret the gravitational-wave astronomy data, astrophysicists must rely on computationally challenging large-scale numerical simulations to study the details of the energetic processes occurring in regions of strong curvature. Such astrophysical systems and phenomena include the birth of neutron stars or black holes in collapsing evolved massive stars, the coalescence of compact binary systems, Gamma-Ray Bursts (GRBs), active galactic nuclei harboring supermassive black holes, pulsars, and oscillating neutron stars.

Of these, Gamma-Ray Bursts (GRBs) [12] are among the most scientifically interesting. GRBs are intense, narrowly-beamed flashes of γ-rays originating at cosmological distances, and the riddle concerning their central engines and emission mechanisms is one of the most complex and challenging problems of astrophysics today. The physics necessary in such a model includes General Relativity, relativistic magneto-hydrodynamics, nuclear physics (describing nuclear reactions and the equation of state of dense matter), neutrino physics (weak interactions), and neutrino and photon radiation transport. The complexity of the GRB central engine requires a multi-physics, multi-length-scale approach that cannot be fully realized on present-day computers and will require petascale computing [13, 14].

At LSU we are performing simulations of general relativistic systems in the context of a decade-long research program in numerical relativity. One pillar of this work is focused particularly on 3D black hole physics and binary black hole inspiral and merger simulations. This includes the development of the necessary tools and techniques to carry these out, such as mesh refinement and multi-block methods, higher order numerical schemes, and formulations of the Einstein equations. A second pillar of the group's research is focused on general relativistic hydrodynamics simulations, building upon results and progress achieved with black hole system. Such simulations are crucial for detecting and interpreting signals soon expected to be recorded from ground-based laser interferometric detectors.

The specific application scenario for the work presented at the SCALE 2009 competition is the numerical modeling of the gravitational waves produced by the inspiral and merger of binary black hole systems (see Figure 2.1).

2.1. Use-case Scenario. The motivating and futuristic scenario for this work is based on enabling scientific investigation using complex application codes on very large scale compute resources:

- Scientists working together in a distributed collaboration are investigating the use of different algorithms for accurately simulating radiation transport as part of a computational model of gamma-ray bursts which uses the Cactus Framework. The resulting simulation codes use adaptive mesh refinement to dynamically add additional resolution where needed, involve hundreds of independent modules coordi-

nated by the Cactus Framework, require the use of tens of thousands of cores of modern supercomputers and take several days to complete.

- The scientists use the Kranc code generation package to automatically generate a suite of codes using the different algorithms that they wish to compare. Kranc writes these codes taking advantage of appropriate optimization strategies for the architectures on which they will be deployed, for example using GPU accelerators where available, or matching grid loops to the available cache size.
- The simulations are deployed on multiple supercomputers available to the collaboration, using co-scheduling services across different institutions to coordinate the simultaneous reservation of resources, networks, and displays. Web services are used to enable the real-time, highly configurable, collaboration of the scientists, with the simulations autonomously publishing appropriate information to services such as Twitter and Flickr.
- As the simulations run, output data is directly streamed across high speed networks to powerful GPU rendering clusters which produce the visualizations, and in turn stream their video outputs to large high resolution displays located at the collaborating sites. The displays aggregate the video outputs from each of the different simulations, allowing the scientists to visualize and compare the same output, while simultaneously interacting with and steering the visualization using tangible devices.

The scientists are thus able to use the most powerful computational resources to run the simulation and the most powerful visualization resources available to interactively visualize the data and are not limited by either their local visualization resources, or the visualization resources available at the location where the simulation is being run.

3. Automatic Parallel Code Generation.

3.1. Cactus–Carpet Computational Infrastructure. *Cactus* [2, 3] is an open source software framework consisting of a central core, the *flesh*, which connects many software components (*thorns*) through an extensible interface. *Carpet* [4, 5, 6] serves as a driver layer of the *Cactus* framework providing adaptive mesh refinement, multi-patch capability, as well as memory management, parallelism, and efficient I/O. In the *Cactus–Carpet* computational infrastructure, the simulation domain is discretized using high order finite differences on block-structured grids, employing a Berger-Oliger-style adaptive mesh refinement method [15] with sub-cycling in time, which provides both efficiency and flexibility. We use explicit Runge-Kutta methods for time integration.

Cactus is highly portable and runs on all current HPC platforms as well as on workstations and laptops on all major operating systems. Codes written using *Cactus* have been run on various brands of the fastest computers in the world, such as various Intel and AMD based systems, SGI Altix, the Japanese Earth Simulator, IBM Blue Gene, Cray XT, and the (now defunct) SiCortex architecture, among others. Recently, the *Cactus* team successfully carried out benchmark runs on 131,072 cores on the IBM Blue Gene/P at the Argonne National Laboratory [16].

3.2. Kranc Code Generation Package. *Kranc* [17, 1, 18] is a Mathematica-based computer algebra package designed to facilitate analytical manipulations of systems of tensorial equations, and to automatically generate C or Fortran code for solving initial boundary value problems. *Kranc* generates complete *Cactus* thorns, starting from a high-level description including the system of equations formulated in high-level Mathematica notation, and discretizing the equations with higher-order finite differencing. *Kranc* generated thorns make use of the Cactus Computational Toolkit, declaring to Cactus the grid functions which the simulation will use, and computing the right hand sides of the evolution equations so that the time integrator can advance the solution in time.

3.3. McLachlan Code. The *McLachlan* code [19, 20] was developed in the context of the XiRel project [21, 22], a collaboration between several numerical relativity groups worldwide to develop a highly scalable, efficient, and accurate adaptive mesh refinement layer for the *Cactus* framework, based on the *Carpet* driver, aimed at enabling numerical relativists to study the physics of black holes, neutron stars and gravitational waves. The *McLachlan* code is automatically generated using the *Kranc* code generation package (see above) from a high-level description of the underlying set of partial differential equations. The automation is of particular importance for experimenting with new formulations of the equations, new numerical methods, or particular machine specific-code optimizations. *McLachlan* employs a hybrid MPI/OpenMP parallelism.

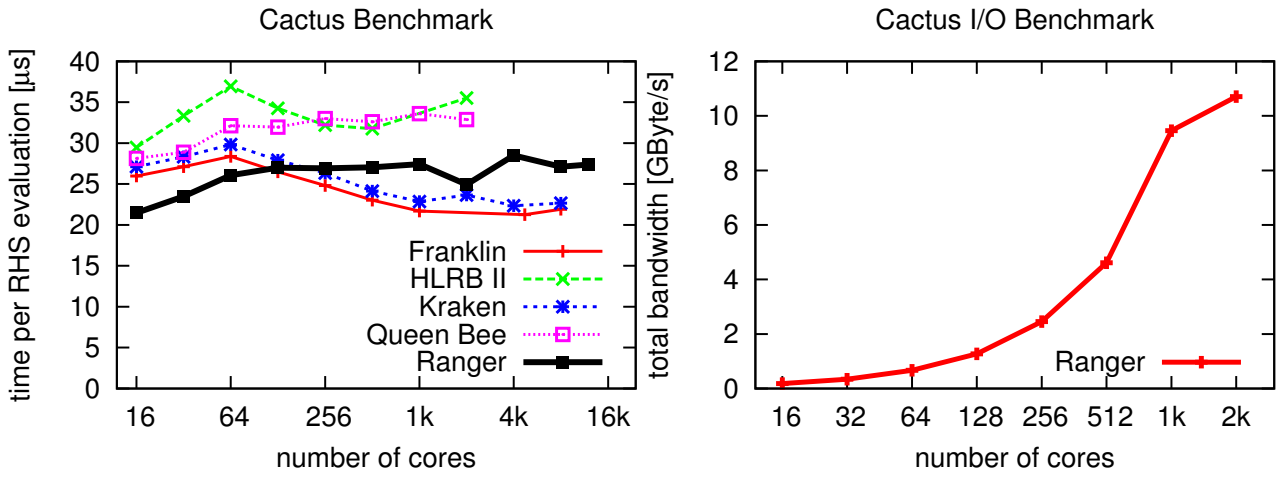


FIG. 3.1. ***Left:*** *Weak scaling benchmark results of the* McLachlan *code on several current leadership HPC systems. This benchmark simulates a single black hole with nine levels of mesh refinement. The code scales well up to more than 12,000 cores of Ranger at TACC.* ***Right:*** *I/O benchmark on Ranger, showing the total I/O bandwidth vs. the number of cores. Cactus achieves a significant fraction of the maximum bandwidth already on 1,000 cores.*

As can been seen from Figure 3.1, *on TACC's Ranger,* McLachlan *and the supporting infrastructure scale well up to more than 12,000 cores. Cactus–Carpet* is also able to use a significant fraction of the theoretical peak I/O bandwidth already on 1,000 cores.

4. Interactive and Collaborative Simulations.

4.1. Monitoring, Profiling and Debugging. Supporting performance and enforcing correctness of the complex, large-scale codes that Cactus generates is a non-trivial task, targeted by the NSF-funded Application-Level Performance and Correctness Analysis (Alpaca) project [23, 24].

In order to reap the benefits of the high-concurrency machines available today, it is not sufficient that a code's parallel efficiency remain constant as the size of the problem is scaled up, but also that its ease of control remains close to that of simulations carried out on a few number of computing cores; if this is not the case, the process of debugging and optimizing the code may be so time-consuming as to offset the speed-up obtained through parallelism. The Alpaca project addresses this issue through the development of application-level tools, i. e., high-level tools that are aware of the Cactus data structures and execution model.

In particular, Alpaca's objectives concentrate on three areas: (i) high-level debugging, devising debugging strategies that leverage high-level knowledge about the execution actors and the data processing, and develop tools that extract such information from a simulation and provide it in an abstract format to the user; (ii) high-level profiling, devising algorithms for extracting high-level information from timing data; and (iii) remote visualization, using visual control over the simulation data as a high-level correctness check. In particular, work within the Alpaca project includes the development of HTTPS, a Cactus module that spawns an SSL web server, with X.509 certificate authorization, at the beginning of a simulation and uses it to receive incoming connections, expose the simulation's details and provide fine-grained control over its execution.

4.2. Leveraging Web 2.0 for Collaborations. In response to the fact that computer simulations are becoming more complex and requiring more powerful resources, the way science itself is carried out is similarly changing: the growing use of computers and world-wide networks has radically modified the old custom of individual (or small-collaboration) work and hand-written data collected in notebooks. When Stephen Hawking worked out the basic theoretical framework for two colliding black holes [25] in the early seventies and Larry Smarr carried out early numerical simulations [26] a few years later, both involved only very small teams and generated perhaps a megabyte of data. The same problem has been studied in full 3D [27], now with a team size of perhaps 15 researchers, a growing number of involved institutes and an increase in generated data by a factor of about a million. Currently unsolved problems like the Gamma-Ray Burst riddle [14] will require still larger collaborations, even from different communities, and generate even more data.

In order for scientific collaborations to work at this scale, for the large amounts of data to be handled properly, and for the results to be reproducible, new methods of collaboration must be developed or already existing tools from other fields must be leveraged. Cactus can now use two tools from the latter class to announce information about simulations to existing Web 2.0 services, as described in the following.

Twitter's [28] main service is a message routing system that enables its users to send and read each others' updates, known as *tweets*. Tweets have to be very short (at most 140 characters in length) and can be sent and read via a wide range of devices, e.g. mobile texting, instant message, the web, or external applications.

Twitter provides an API [29] which allows the integration of Twitter with other web services and applications. One of the most important functions is the "statuses/update" API call, which is used to post a new Twitter message from the specified user. This Twitter API is used in a Cactus thorn to announce live information from a simulation (Figure 7.2).

Flickr [30] was launched as an image hosting website targeted at digital photographs, but short videos can be uploaded today as well. Flickr can be used at no charge with limits on the total size of images that can be uploaded (currently 100 MByte) and on the number of images which can be viewed (currently 200), along with other potential services available.

One important functionality, besides the image upload, is to be able to group images. Flickr offers a capability to group images into "Sets", and also can group different "Sets" into a "Collection". This provides a hierarchical structure for organizing simulation images.

Flickr has many features that can be taken advantage of for providing a collaborative repository for Cactus-produced images and information. All of them are accessed through a comprehensive web service API for uploading and manipulating images [31].

A new Cactus thorn uses the Flickr API to upload live images from the running simulation. Images generated by one simulation are grouped into one "Set". It is also possible to change the rendered variables, or to change the upload frequency on-line through an Cactus-internal web server (see section 4.1).

5. Distributed Visualization.

5.1. Visualization Scenario. Our scenario is the following: the visualization user is connected over a network link to a grid system of various types of resources (visualization, network, compute, data). The data to be visualized is located on a data server near the location where the scientific simulation was executed and this data server is also connected to the grid system.

There are various ways in which a visualization application can be created to solve the problem, such as running the visualization on the data server and transferring a video stream to the client, or running the visualization on the local client and transferring a data stream between the data server and the client.

These two solutions are limited by the visualization power available near the data server or near the local machine, respectively. Since powerful visualization resources are not available at the client and may not be available near the data server, we have built a three-way distributed system that uses a visualization cluster in the network, data streaming from the data server to the visualization cluster, and video streaming from the visualization cluster to the local client.

We have taken the problem one step further and considered the case where the network connection of the data server is a relatively slow one—much slower than the network capacity of the rendering machine. In this situation we are dealing with I/O scalability issues, and the solution we propose is to create a temporary distributed data server in the grid. The distributed data server uses compute and data resources that are not dedicated for this application but are allocated on-demand to support it when it needs to execute. The distributed data server can sustain much higher data transfer rates than a single data source. Data is loaded in advance from the source on the distributed data server. The architecture of this approach is illustrated in Figure 5.1. Because the visualization resources are not local to the end client, a remote interaction system is necessary in order for the user to be able to connect to and steer the visualization.

5.2. I/O. High-performance data transmission over wide-area networks is difficult to achieve. One of the main factors that can influence performance is the network transport protocol. Using unsuitable protocols on wide area network can result in bad performance—for example a few Mbit/s throughput on a 10 Gbit/s dedicated network connection using TCP. The application needs to use protocols that are suitable for the network that is utilized; our system uses the UDT [32] library for wide-area transfers in order to achieve high data transmission rates. Another issue is blocking on I/O operations: blocking I/O reduces the performance

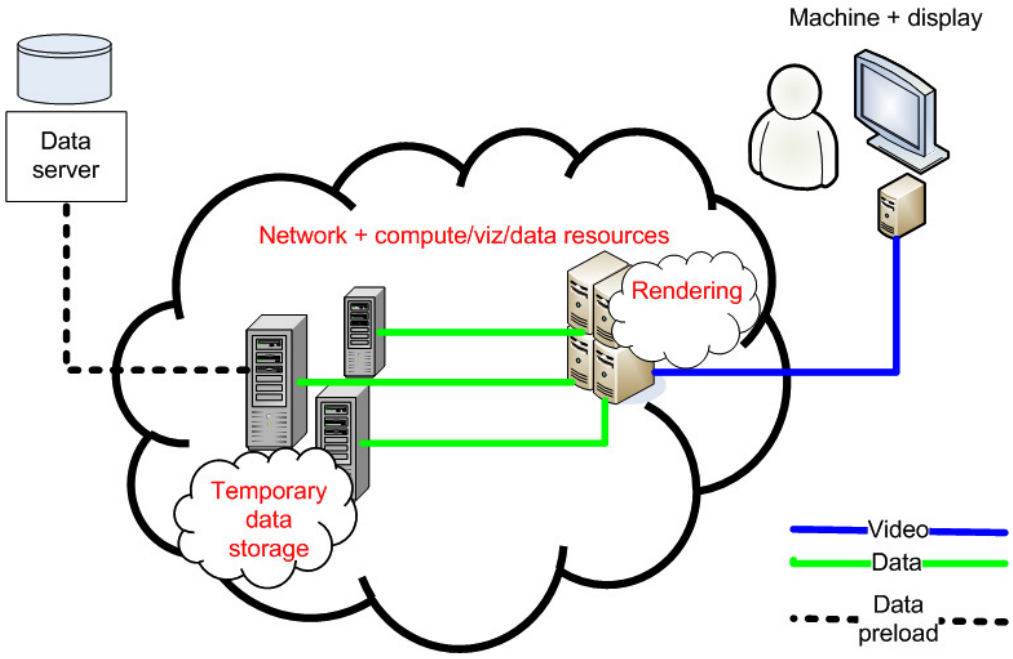


FIG. 5.1. *Architecture of demonstration system which involves a temporary distributed data server allocated on-demand to improve sustained data transfer rates.*

that is seen by the application, and the solution we use is based on a completely non-blocking architecture using a large number of parallel threads to keep the data flow moving.

5.3. Parallel Rendering. Parallel rendering on HPC or visualization clusters is utilized to visualize large datasets. For the SCALE 2009 demonstration we have used a self-developed parallel GPU ray-casting volume rendering implementation to interactively visualize the time-dependent numerical relativity dataset, where each timestep has a size of about 1 GByte. GPU ray-casting does floating point compositing in a single pass using a fragment shader. The trade-off between rendering time and visual quality can be steered directly by the user(s). Our rendering system overlaps communication with (visual) computation, in order to achieve maximum performance. Parallelism is achieved by data domain decomposition (each node renders a distinct subsection of the data), and compositing of the resulted partial view images in order to create a single view of the entire dataset.

5.4. Video Streaming and Interaction. Interaction with the remote parallel renderer is necessary to modify navigation parameters such as the direction of viewing or the level of zoom, and to control the trade-off of visual quality and image deliver time. Since the visualization application is not local, an interaction system consisting of three components was developed. The components are a local interaction client running on the local machine, an interaction server running on the rendering cluster, and an application plug-in that connects the interaction server to the application and inserts interaction commands into the application workflow (see Fig. 5.2).

For our demonstration we used specialized interaction devices developed by the Tangible Visualization Laboratory at CCT [33] that are very useful in long-latency remote interaction systems, and can support collaboration (collaborative visualization) from multiple sites.

The final component of the system is the video streaming. Images that are generated from the remote visualization cluster need to be transported to the local client for the user to see. In the past, we have successfully utilized hardware-assisted systems running videoconferencing software (*Ultragrid* [34]) and software-based video streaming using *SAGE* [35]. Our system supports various video streaming methods including SAGE, a self developed video streaming subsystem, or VNC [36].

6. Related Work. OptIPuter [37] is a large project that has built an advanced infrastructure connecting computational infrastructure with high-speed "lambda" networks to create virtual distributed metacomputers. OptIPuter technologies are used in scientific applications such as microbiology [38] and climate analysis [39].

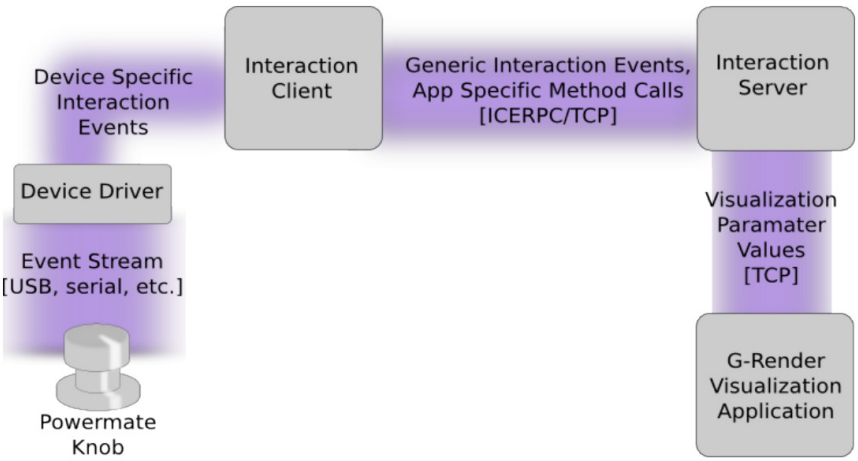


FIG. 5.2. *Architecture of interaction system*

One method for tightening the integration of applications and networks is to use reserved, exclusive access to network resources controlled by the user. Several projects, including DOE UltraScienceNet [40, 41], NSF CHEETAH [42], Phosphorus [43], G-lambda [44] and Internet2 ION [45] have explored mechanisms for providing such network services to applications.

The limitations of desktop-based visualization led to the development of parallel visualization systems and frameworks such as ImageVis3D [46], Chromium [47] and Equalizer [48] that can take advantage of computational clusters to visualize large datasets. Equalizer and Chromium are parallel rendering frameworks that can be used to build parallel rendering applications. ImageVis3D is a parallel rendering tool for interactive volume rendering of large datasets. These and other tools and techniques are being developed (for example as part of the Institute for Ultra-Scale Visualization [49, 50]) to be able to take advantage of parallel resources for visualization.

Other visualization systems such as Data Vault (DV) [51], ParaView [52], and VisIt [53] were designed to facilitate the visualization of remote data and, while they have the capability to transmit data and images over the network, they are not able to take advantage of the full capacity of high-speed networks and thus have low data transport performance, can suffer from a lack of interactivity and image quality, and do not support collaborative visualization.

Data Vault (DV) is a visualization and data analysis package for numerical codes that solve partial differential equations via grid-based methods, in particular those utilizing adaptive mesh refinement (AMR) and/or running in a parallel environment. DV provides a of built-in functions to analyze 1D, 2D, and 3D time-dependent datasets.

ParaView is a parallel visualization application designed to handle large datasets. It supports two distribution modes: client–server and client–rendering server–data server.

VisIt is a visualization software designed to handle large datasets using client–server distribution of the visualization process. Similar to ParaView's client–server distribution, VisIt uses a parallel rendering server and a local viewer and interaction client. Most commonly, the server is as a stand-alone process that reads data from files. An alternative exists where a simulation code delivers data directly to VisIt, separating the server into two components. This allows for visualization and analysis of a live running simulation.

Several visualization systems such as RAVE [54] or the visualization system by Zhu et al. [55] are focused on the theoretical aspects of distributed visualization and do not provide the level of performance and scalability needed for current scientific applications. The distributed visualization architecture proposed by Shalf and Bethel [56] could support a variety of distributed visualization applications and inspired the development of our system.

The German TIKSL [57] and GriKSL [58] projects developed technologies for remote data access in Grid environments to provide visualization tools for numerical relativity applications. Based on GridFTP [59] and the HDF5 library [60], these projects prototyped a number of remote visualization scenarios on which this work is based.

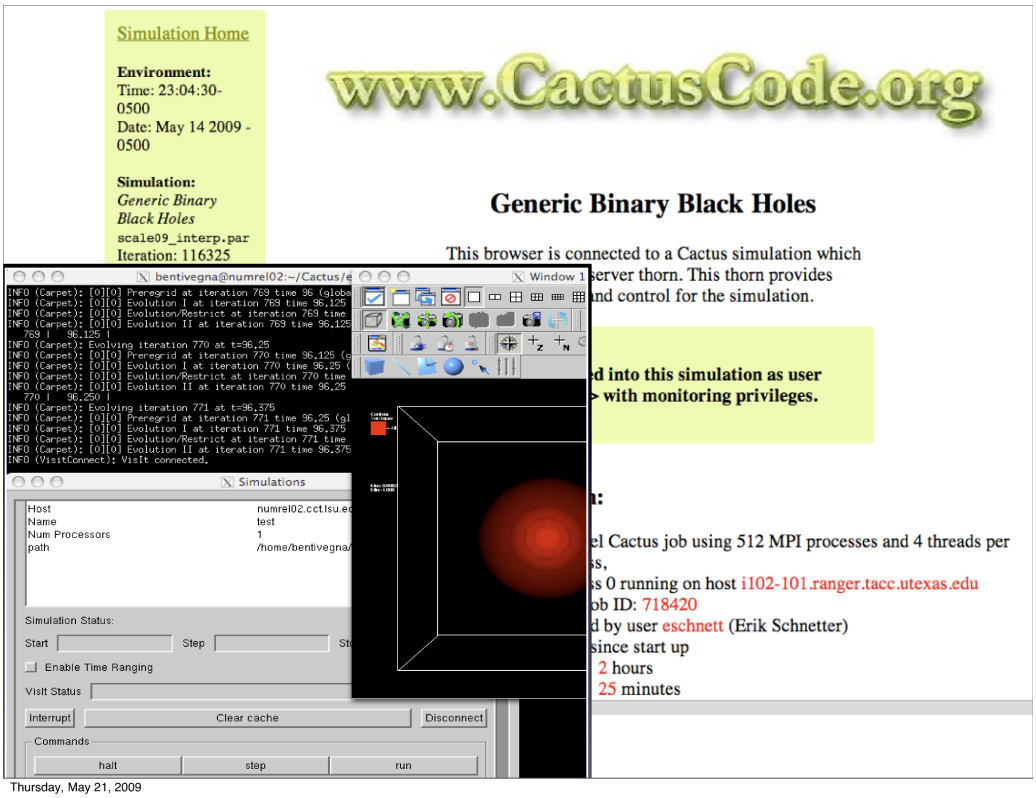


FIG. 7.1. *The Alpaca tools provide real-time access to simulations running on remote machines, allowing monitoring, interactive visualization, steering, and high-level debugging of large-scale simulations.*

7. SCALE 2009 Demonstration and Results. The resulting Cactus application *McLachlan* used the Carpet Adaptive Mesh Refinement infrastructure to provide scalable, high order finite differencing, in this case running on 2048 cores of the Texas Advanced Computing Center (TACC) Ranger machine. The simulation ran for altogether 160 hours on Ranger, generating 42 TByte of data. Live interaction with the simulation was shown, via the application-level web interface HTTPS (Fig. 7.1, Section 4.1). The simulation also used new thorns co-developed by an undergraduate student at LSU to announce runtime information to Twitter and real-time images of the gravitational field to Flickr (Fig. 7.2).

Interactive visualization of the data produced was shown using a visualization system distributed across the Louisiana Optical Network Initiative (LONI), see Fig. 7.3. A data server deployed on the Eric and Louie LONI clusters cached 20 GByte of data at any time in RAM using a total of 8 compute nodes. This data was then transferred using TCP and UDT protocols over the 10 Gbit/s LONI network to rendering nodes at a visualization cluster at LSU.

The average aggregate I/O rate achieved by the SCALE 2009 system was 4 Gbit/s. Loading time from the remote distributed resources was six times faster than local load from disk (2 s remote vs. 12.8 s local).

Here a new parallel renderer used GPU acceleration to render images, which were then streamed using the SAGE software to the final display. VNC (Fig. 7.4) was used instead of SAGE in Shanghai due to local network limitations. Tangible interaction devices (also located in Shanghai) provided interaction with the renderer.

The size of a rendered timestep was 1024^3 bytes, for a total data size of 1 GB/timestep. The rendering performance of the SCALE 2009 system for this data, executed on an 8 node rendering cluster each node equipped with NVidia Geforce 9500 GT 256 MB DDR3 was 5 frames per second.

The visualization system demonstrated its capability for interactive, collaborative and scalable visualization, achieving the team's goal of showing how distributed systems can provide enhanced capabilities over local systems. This system was awarded first place in the IEEE SCALE 2009 Challenge.

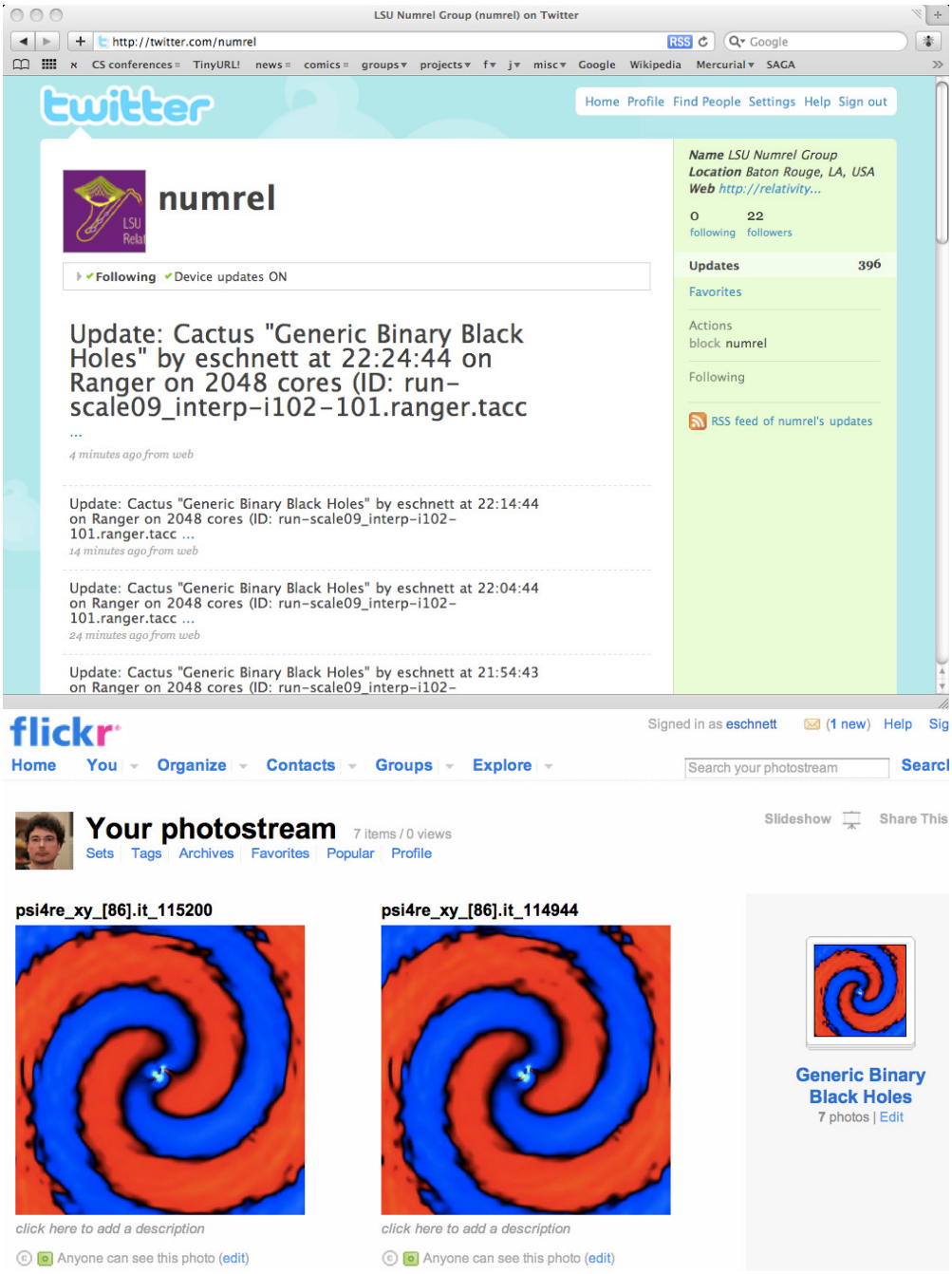


FIG. 7.2. *New Cactus thorns allow simulations to announce live information and images to (top) Twitter, (bottom) Flickr enabling a new mode of scientific collaboration using Web 2.0 technologies.*

8. Results After Demonstration. After SCALE 2009 we continued to improve the system, in particular the rendering and I/O subsystems. In our code generation and simulation infrastructure, we have been concentrating on adding new physics (in particular radiation transport), a new formulation of the Einstein equations, and on improving single-node (single-core and multi-core) performance.

We evaluated the visualization system performance and compared it with the performance of Paraview and VisIt using a sample dataset with a resolution of 4096^3 bytes with a total size of 64 GB.

Benchmarks were performed on the 8-node visualization cluster at LSU, each node having two Quad-core Intel Xeon E5430 processors (2.66 GHz), 16 GB RAM. The performance of the rendering cluster was improved

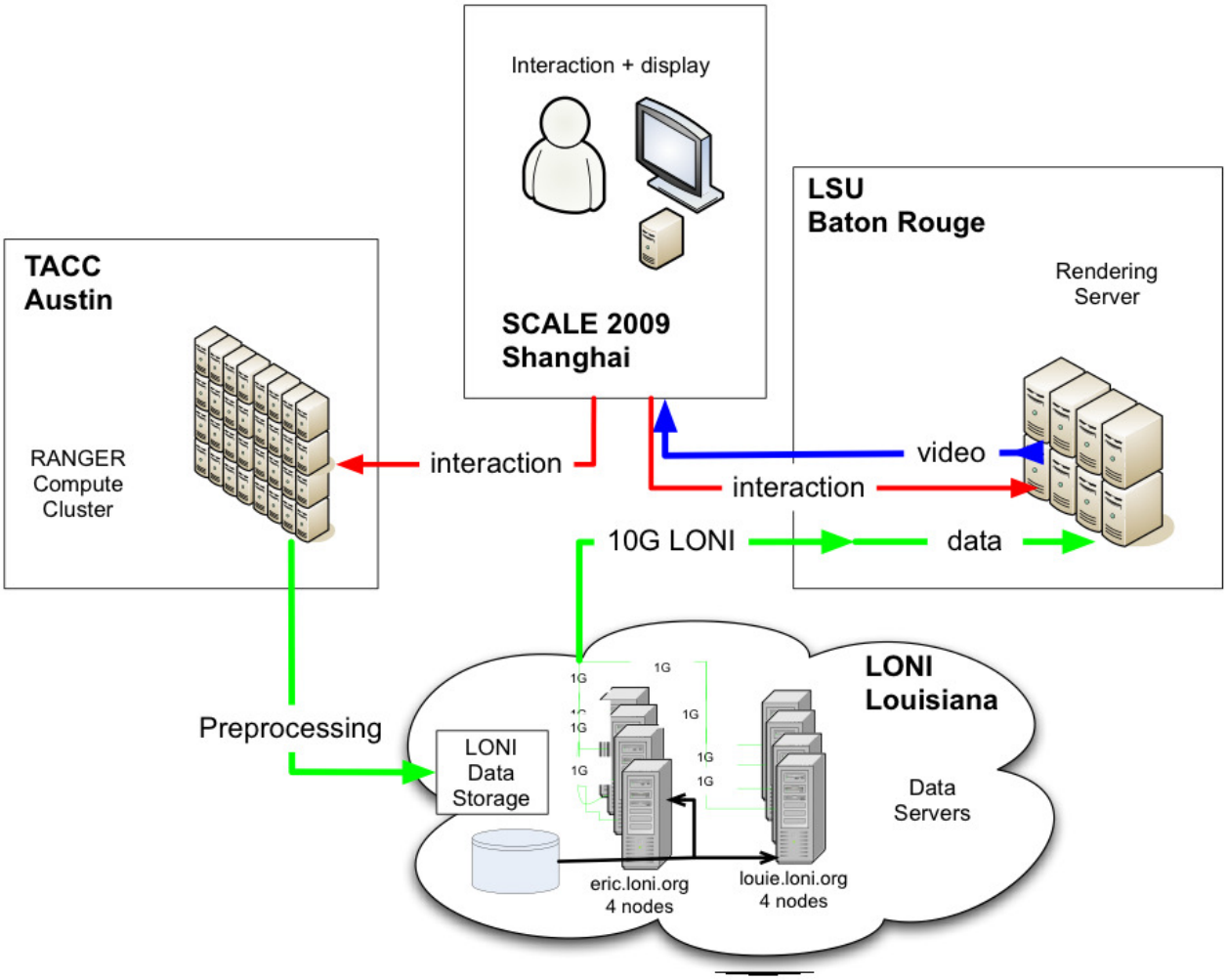


FIG. 7.3. *Set up of the SCALE 2009 demonstration that involved the resources of the NSF TeraGrid, the Louisiana Optical Network Initiative (LONI) and the Center for Computation & Technology.*

by upgrading the graphics hardware to four NVidia Tesla S1070 units. Each Tesla contains 4 GPUs, has 16 GB video memory and services two rendering nodes, each node thus having access to two GPU units and 8 GB video memory. The cluster interconnect is 4x Infiniband and the software was compiled and executed using MPICH2, version 1.1.1p1 using IP emulation over Infiniband.

The rendering frame rate was measured and local I/O performance was compared with remote (network) performance for three scenarios: rendering 15 GB data using 8 processes, rendering 30 GB data using 16 processes (two processes per node), and rendering 60 GB data using 32 processes (four processes per node, two processes per GPU).

The network data servers were deployed on two LONI clusters, using up to 32 distributed compute nodes to store data in the main memory. The network protocol used for data transfer was UDT.

For reference, the performance of the system when running on a single workstation was measured (workstation specifications: Intel Core2 CPU X6800, 2.93 GHz, 4 GB RAM, graphics: GeForce 8800 GTX, 1 GB video memory, 1 Gbps network interface). The rendering resolution for the benchmark is 1024x800 pixels.

The results are shown in Table 8.1. We can see that as we increase the number of rendering processes we can render more data, however the frame rate is decreasing. This reduction in speed is expected because the communication overhead increases with the number of processes. The effect is a reduction in frame rate, showing a fundamental issue with parallel rendering: at some point as the data size (and thus number of

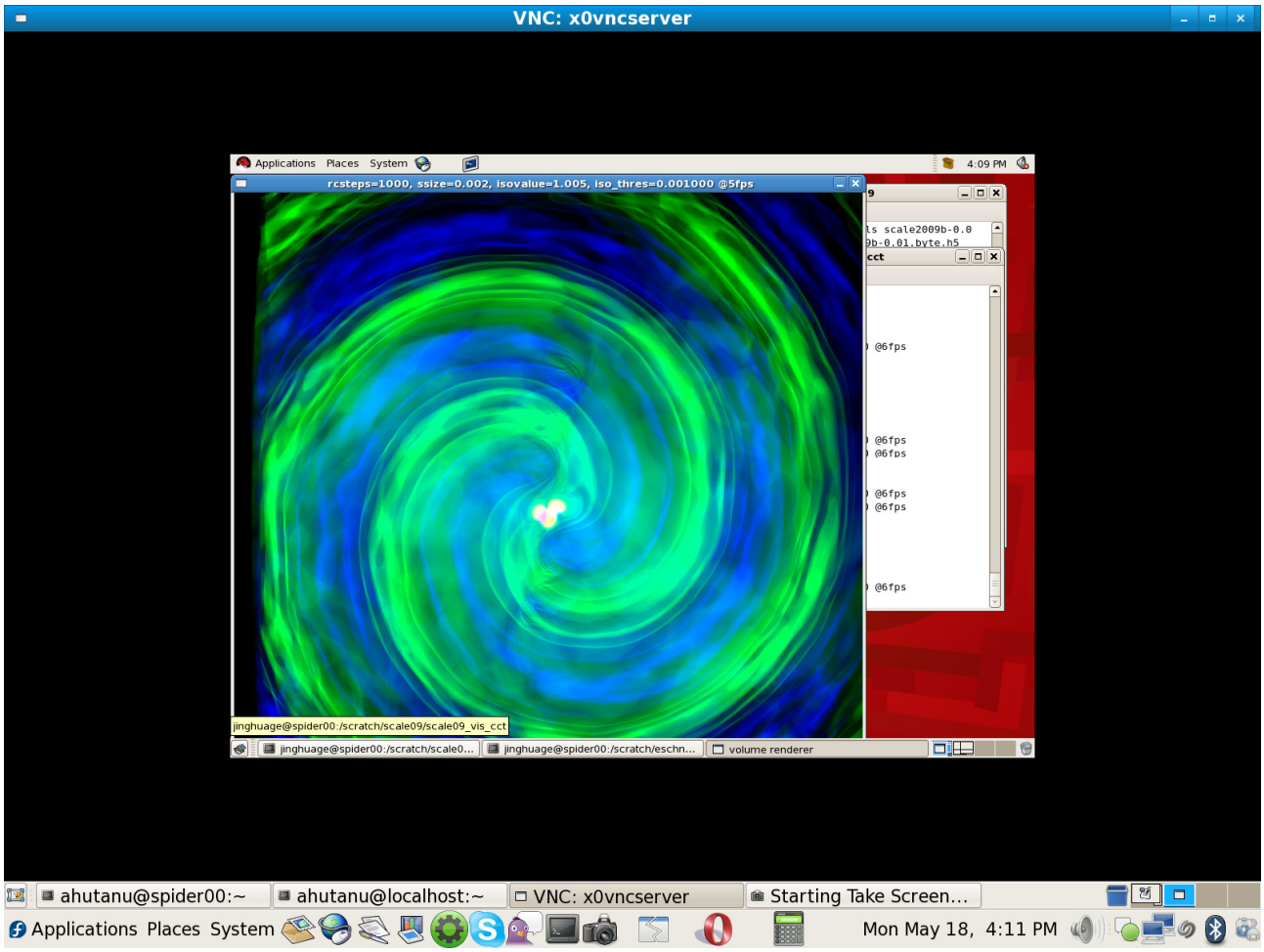


FIG. 7.4. *Visualization client on the end display showing rendering of gravitational waves emitted from the inspiral collision of two black holes.*

processes required to render it) increases, the frame rate drops to a level below the point of interactivity. The results show that the system is able to utilize the rendering cluster to interactively render 35 times more data than a typical workstation, and maintain an acceptable level of interactivity while rendering more than 70 times more data than on the workstation. The current system is able to interactively visualize data 60 times larger than that supported by the SCALE 2009 system.

TABLE 8.1
Data throughput and rendering scalability results.

# processes	Data size	Frame rate (fps)	Local speed	Network speed
1 (workstation)	0.8 GB	30	0.68 Gbps	0.8 Gbps
8 (cluster)	15 GB	15-21 (18 avg)	0.11 Gbps	6.6 Gbps
16 (cluster)	30 GB	11-13 (12 avg)	0.12 Gbps	5.3 Gbps
32 (cluster)	60 GB	4-5 (4.5 avg)	0.2 Gbps	4.3 Gbps

Regarding data speed, we see a big advantage when using network I/O, proving the value of the proposed approach of designing the system to be able to take advantage of high-speed networks. The system achieves 6.6 Gbps throughput over the LONI wide-area network (the limit being the network interfaces on the cluster) when using 8 processes. As we increase the number of processes the network speed decreases slightly because of the increased contention on the network interface on the same node. The remote I/O performance is 20-60

times better than local I/O performance, and the current system is able to sustain up to 6.6 Gbps transfer rates, both showing significant improvements over the SCALE 2009 system.

To better understand the features and the trade-offs of our system (named eaviv) a comparison with alternative visualization systems was made. Two appropriate comparison systems were identified, ParaView (version 3.6.1) and VisIt (version 1.12.0).

TABLE 8.2
Comparison of visualization systems features and performance: I/O methods, rendering and other items

Feature	*eaviv*	ParaView	VisIt
Data protocols	UDT, TCP, fully configurable	TCP only	TCP only
High-speed data limit	*Yes: Main memory*	*No: Disk size*	*No: Disk size*
Frame rate	*11-12 fps (30 GB)*	*0.5-1 fps (32 GB)*	*0.28-0.35 fps (32 GB)*
Render size limit	*60 GB (GPU memory)*	*120 GB (CPU memory)*	*120 GB (CPU memory)*
Collaborative support	Yes: SAGE video distribution, tangible devices	No	No
Video streaming	Parallel (SAGE)	Serial	Serial
Direct simulation connectivity	No	No	Yes
Fully-featured visualization application	No (Prototype)	Yes	Yes

The comparison was made in three different areas: data input; parallel rendering; and miscellaneous items. Both qualitative and quantitative items were analyzed. Slightly different data sizes were used due to the different modes of selecting the section of interest in each system.

Starting with data input, our system supports multiple data protocols allowing it to take advantage of high-speed networks. The benchmark results shown in Table 8.2 executed on the rendering cluster shows how our system can take advantage of the high-speed network to achieve a high data throughput. This throughput can however only be sustained for an amount of data equal to the main memory size available in the network. Both ParaView and VisIt throughput is limited by disk speed.

The second area of interest is the parallel rendering component. Our system uses a GPU-based parallel renderer, allowing it to take advantage of graphics acceleration for volume rendering and enabling high frame rate. ParaView and VisIt do not currently support parallel GPU acceleration, and in consequence the frame rate that they can achieve is below 1 frame per second. For VisIt the ray-casting parallel rendering method was used for comparison. GPU-based rendering is however limited in the data size that it can render by the amount of video memory of the graphics cards. CPU-based rendering can usually render more data, as the amount of main memory in a system is generally higher than that of video memory.

Parallel video streaming is a feature supported by our use of the SAGE system. Each rendering node, after generating a section of the final image can transmit it directly to the viewer client. In contrast, VisIt and ParaView rendering processes transmit their results first to the master node which combines them and transmits the complete image to the client. Serial video streaming introduces additional overhead and latency into the system.

Our prototype has integrated support for tangible interaction devices while allowing mouse and keyboard interaction through the use of third-party software, such as VNC. The use of SAGE and tangible interaction devices enables direct support of collaborative visualization, where multiple users, potentially at different locations around the world can simultaneously interact and collaborate using the visualization system. SAGE bridges can be used to multicast the video stream from the application to multiple users, and interaction devices deployed at each user location can be used to interact with the visualization.

One of the important missing features of our system is the lack of a direct connection between the visualization and the simulation, needed in order to visualize live data, as it is being generated (this feature is already supported by VisIt). Our system is designed as a prototype to explore the possibilities of distributed visualization, it only supports volume rendering of uniform scalar data, and has only a small fraction of the features of complete visualization systems such as VisIt and ParaView.

9. Conclusion and Future Work. Our system shows the viability of the proposed approach of using the *Cactus* framework, automated code generation, and modern numerical methods to scale to a large number of processors. It also shows how distributing the visualization system into separate components increases the amount of data that can be handled, increases the speed at which the data can be visualized compared to local techniques, and improves data transfer rates and interactivity.

The driving principle behind our approach is that, following a careful analysis and based on a detailed description of a particular technical problem, a scalable solution is to build a fully optimized integrated software system. Our proposed system is still modular, however the interfaces between the various components are flexible, and were carefully designed to enable optimizations across multiple components. The scalability of this system would suffer if each component would be designed and implemented in isolation of the other components and the coupling between components would be limited to legacy or rigid interfaces.

Our integrated approach enables us to take advantage of the latest improvements in GPU architectures, networks, innovative interaction systems and high-performance data and video transmission systems and provides a solution and a model for future scientific computing, and we believe many other applications will be able to benefit from adopting a similar approach.

In the future, we are planning to tighten the connection between the simulation and visualization components, to enable our system to visualize data on the fly as it is generated by the simulation. Our future plans include integrating automated provisioning of network resources in our application. Towards this we are currently in the process of building a testbed that connects compute, storage, graphics and display resources (provided by TeraGrid sites Louisiana State University and National Center for Supercomputing Applications and international partners such as Masaryk University) together with dynamic network circuit services provided by Internet2 and the Global Lambda Integrated Facility international community.

A fundamental service required by our system is co-allocation and advance reservation of resources. We are working together with resource providers and actively encourage them to enable these services that are crucial to enable the routine execution such complex distributed applications.

10. Acknowledgements. We thank the current and former members of the AEI/LSU numerical relativity group who developed the physics code that made this work possible, and the extended Cactus team for their computational infrastructure. We also thank the original authors of the Kranc code generation package for making their work publicly available. We acknowledge Jorge Ucan and Kristen Sunde for preparing the high-definition video for the SCALE 2009 Challenge, and Debra Waters for help with organization. The demonstration would not have been possible without the help of our colleagues at TACC and LONI.

This material is based upon work supported by the National Science Foundation (Alpaca #0721915, Blue Waters #0725070, Viz Tangibles #0521559, XiRel #0701566, Louisiana RII *CyberTools* #0701491, OCI EAGER *eaviv* #0947825) and by the Center for Computation & Technology at LSU. The simulations and benchmarks were performed on Queen Bee at LONI under allocations `loni_cactus03` and `loni_numrel03`, and on Ranger at TACC under the NSF TeraGrid allocation TG-MCA02N014. The distributed visualization development was supported by the NSF TeraGrid allocation TG-CCR080027T.

REFERENCES

[1] Sascha Husa, Ian Hinder, and Christiane Lechner. Kranc: a Mathematica application to generate numerical codes for tensorial evolution equations. *Comput. Phys. Comm.*, 174:983–1004, 2006.
[2] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Massó, Thomas Radke, Edward Seidel, and John Shalf. The Cactus framework and toolkit: Design and applications. In *High Performance Computing for Computational Science - VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002*, pages 197–227, Berlin, 2003. Springer.
[3] Cactus Computational Toolkit home page.
[4] E. Schnetter, S. H. Hawley, and I. Hawke. Evolutions in 3D numerical relativity using fixed mesh refinement. *Class. Quantum Grav.*, 21(6):1465–1488, 21 March 2004. gr-qc/0310042.
[5] Erik Schnetter, Peter Diener, Nils Dorband, and Manuel Tiglio. A multi-block infrastructure for three-dimensional time-dependent numerical relativity. *Class. Quantum Grav.*, 23:S553–S578, 2006.
[6] Mesh refinement with Carpet.
[7] R.N. Hulse and J.H. Taylor. Discovery of a pulsar in a binary system. *Astrophys. J.*, 195:L51–L53, 1975.
[8] Ramesh Narayan and Jeffrey E. McClintock. Advection-dominated accretion and the black hole event horizon. *New Astron. Rev.*, 51:733–751, 2008.
[9] LIGO: Laser Interferometer Gravitational Wave Observatory.
[10] GEO 600.
[11] VIRGO.

[12] Peter Mészáros. Gamma-ray bursts. *Rep. Prog. Phys.*, 69:2259–2321, 2006.
[13] Christian D. Ott, Erik Schnetter, Gabrielle Allen, Edward Seidel, Jian Tao, and Burkhard Zink. A case study for petascale applications in astrophysics: Simulating Gamma-Ray Bursts. In *Proceedings of the 15th ACM Mardi Gras conference: From lightweight mash-ups to lambda grids: Understanding the spectrum of distributed computing requirements, applications, tools, infrastructures, interoperability, and the incremental adoption of key capabilities*, number 18 in ACM International Conference Proceeding Series, Baton Rouge, Louisiana, 2008. ACM.
[14] Erik Schnetter, Christian D. Ott, Gabrielle Allen, Peter Diener, Tom Goodale, Thomas Radke, Edward Seidel, and John Shalf. Cactus Framework: Black holes to gamma ray bursts. In David A. Bader, editor, *Petascale Computing: Algorithms and Applications*, chapter 24. Chapman & Hall/CRC Computational Science Series, 2008.
[15] Marsha J. Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Comput. Phys.*, 53:484–512, 1984.
[16] Cactus Benchmark Results and Papers.
[17] Christiane Lechner, Dana Alic, and Sascha Husa. From tensor equations to numerical code — computer algebra tools for numerical relativity. In *SYNASC 2004 — 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania*, 2004.
[18] Kranc: Automated Code Generation.
[19] David Brown, Peter Diener, Olivier Sarbach, Erik Schnetter, and Manuel Tiglio. Turduckening black holes: an analytical and computational study. *Phys. Rev. D*, 79:044023, 2009.
[20] McLachlan, a Public BSSN Code.
[21] Jian Tao, Gabrielle Allen, Ian Hinder, Erik Schnetter, and Yosef Zlochower. XiRel: Standard benchmarks for numerical relativity codes using Cactus and Carpet. Technical Report CCT-TR-2008-5, Louisiana State University, 2008.
[22] XiRel: Next Generation Infrastructure for Numerical Relativity.
[23] Erik Schnetter, Gabrielle Allen, Tom Goodale, and Mayank Tyagi. Alpaca: Cactus tools for application level performance and correctness analysis. Technical Report CCT-TR-2008-2, Louisiana State University, 2008.
[24] Alpaca: Cactus tools for Application-Level Profiling and Correctness Analysis.
[25] S. W. Hawking. Black holes in general relativity. *Comm. Math. Phys.*, 25:152, 1972.
[26] L. Smarr. Spacetimes generated by computers: Black holes with gravitational radiation. *Ann. N. Y. Acad. Sci.*, 302:569–604, 1977.
[27] Miguel Alcubierre, W. Benger, B. Brügmann, G. Lanfermann, L. Nerger, E. Seidel, and R. Takahashi. 3D Grazing Collision of Two Black Holes. *Phys. Rev. Lett.*, 87:271103, 2001.
[28] Twitter.
[29] Twitter Application Programming Interface.
[30] Flickr.
[31] Flickr API Web page.
[32] Yunhong Gu and Robert L. Grossman. Udt: Udp-based data transfer for high-speed wide area networks. *Comput. Networks*, 51(7):1777–1799, 2007.
[33] Brygg Ullmer, Rajesh Sankaran, Srikanth Jandhyala, Blake Tregre, Cornelius Toole, Karun Kallakuri, Christopher Laan, Matthew Hess, Farid Harhad, Urban Wiggins, and Shining Sun. Tangible menus and interaction trays: core tangibles for common physical/digital activities. In *TEI '08: Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 209–212, New York, NY, USA, 2008. ACM.
[34] Petr Holub, Ludek Matyska, Milos Liska, Lukás Hejtmánek, Jirí Denemark, Tomás Rebok, Andrei Hutanu, Ravi Paruchuri, Jan Radil, and Eva Hladká. High-definition multimedia for multiparty low-latency interactive communication. *Future Generation Comp. Syst.*, 22(8):856–861, 2006.
[35] Luc Renambot, Byungil Jeong, Hyejung Hur, Andrew Johnson, and Jason Leigh. Enabling high resolution collaborative visualization in display rich virtual organizations. *Future Gener. Comput. Syst.*, 25(2):161–168, 2009.
[36] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, 1998.
[37] Larry L. Smarr, Andrew A. Chien, Tom DeFanti, Jason Leigh, and Philip M. Papadopoulos. The optiputer. *Communications of the ACM*, 46(11):58–67, 2003.
[38] Larry Smarr, Paul Gilna, Phil Papadopoulos, Thomas A. DeFanti, Greg Hidley, John Wooley, E. Virginia Armbrust, Forest Rohwer, and Eric Frost. Building an optiplanet collaboratory to support microbial metagenomics. *Future Gener. Comput. Syst.*, 25(2):124–131, 2009.
[39] Venkatram Vishwanath, Robert Burns, Jason Leigh, and Michael Seablom. Accelerating tropical cyclone analysis using lambdaram, a distributed data cache over wide-area ultra-fast networks. *Future Gener. Comput. Syst.*, 25(2):184–191, 2009.
[40] N.S.V. Rao, W.R. Wing, S.M. Carter, and Q. Wu. Ultrascience net: network testbed for large-scale science applications. *Communications Magazine, IEEE*, 43(11):S12–S17, Nov. 2005.
[41] N. S. V. Rao, W. R. Wing, S. Hicks, S. Poole, F. Denap, S. M. Carter, and Q. Wu. Ultrascience net: research testbed for high-performance networking. Proceedings of International Symposium on Computer and Sensor Network Systems, April 2008.
[42] Xuan Zheng, M. Veeraraghavan, N.S.V. Rao, Qishi Wu, and Mengxia Zhu. CHEETAH: circuit-switched high-speed end-to-end transport architecture testbed. *Communications Magazine, IEEE*, 43(8):s11–s17, Aug. 2005.
[43] S. Figuerola, N. Ciulli, M. De Leenheer, Y. Demchenko, W. Ziegler, and A. Binczewski on behalf of the PHOSPHORUS consortium. PHOSPHORUS: Single-step on-demand services across multi-domain networks for e-science. In *Proceedings of the European Conference and Exhibition on Optical Communication '07*, 2007.
[44] Atsuko Takefusa, Michiaki Hayashi, Naohide Nagatsu, Hidemoto Nakada, Tomohiro Kudoh, Takahiro Miyamoto, Tomohiro Otani, Hideaki Tanaka, Masatoshi Suzuki, Yasunori Sameshima, Wataru Imajuku, Masahiko Jinno, Yoshihiro Takigawa, Shuichi Okamoto, Yoshio Tanaka, and Satoshi Sekiguchi. G-lambda: coordination of a grid scheduler and lambda path service over GMPLS. *Future Gener. Comput. Syst.*, 22(8):868–875, 2006.

[45] Internet2 ION. Web page, 2009. http://www.internet2.edu/ion/.
[46] 2009. ImageVis3D: A Real-time Volume Rendering Tool for Large Data. Scientific Computing and Imaging Institute (SCI).
[47] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002.
[48] Stefan Eilemann, Maxim Makhinya, and Renato Pajarola. Equalizer: A scalable parallel rendering framework. In *IEEE Transactions on Visualization and Computer Graphics*, 2008.
[49] Kwan-Liu Ma, Chaoli Wang, Hongfeng Yu, Kenneth Moreland, Jian Huang, and Rob Ross. Next-Generation Visualization Technologies: Enabling Discoveries at Extreme Scale. *SciDAC Review*, 12:12–21, February 2009.
[50] Kwan-Liu Ma, Robert Ross, Jian Huang, Greg Humphreys, Nelson Max, Kenneth Moreland, John D. Owens, and Han-Wei Shen. Ultra-Scale Visualization: Research and Education. *Journal of Physics*, 78, June 2007. (Proceedings of SciDAC 2007 Conference).
[51] Data Vault Tutorial.
[52] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre. Remote Large Data Visualization in the ParaView Framework. *Proceedings of the Eurographics Parallel Graphics and Visualization*, pages 162–170, 2006.
[53] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A Contract Based System For Large Data Visualization. *Visualization, IEEE 2005*, pages 25–25, 2005.
[54] I. J. Grimstead, N. J. Avis, and D. W. Walker. RAVE: the resource-aware visualization environment. *Concurrency and Computation: Practice and Experience*, 21(4):415–448, 2009.
[55] Mengxia Zhu, Qishi Wu, Nageswara S. V. Rao, and Sitharama Iyengar. Optimal pipeline decomposition and adaptive network mapping to support distributed remote visualization. *J. Parallel Distrib. Comput.*, 67(8):947–956, 2007.
[56] J. Shalf and EW Bethel. The grid and future visualization system architectures. *Computer Graphics and Applications, IEEE*, 23(2):6–9, 2003.
[57] Werner Benger, Hans-Christian Hege, André Merzky, Thomas Radke, and Edward Seidel. Efficient Distributed File I/O for Visualization in Grid Environments. In B. Engquist, L. Johnsson, M. Hammill, and F. Short, editors, *Simulation and Visualization on the Grid*, volume 13 of *Lecture Notes in Computational Science and Engineering*, pages 1–6. Springer Verlag, 2000.
[58] Hans-Christian Hege, Andrei Hutanu, Ralf Kähler, André Merzky, Thomas Radke, Edward Seidel, and Brygg Ullmer. Progressive Retrieval and Hierarchical Visualization of Large Remote Data. *Scalable Computing: Practice and Experience*, 6(3):57–66, September 2005. http://www.scpe.org/?a=volume&v=23.
[59] William Allcock, John Bresnahan, Rajkumar Kettimuthu, and Michael Link. The Globus Striped GridFTP Framework and Server. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2005. IEEE Computer Society.
[60] HDF5: Hierarchical Data Format Version 5.

Edited by: Marcin Paprzycki
Received: March 30, 2010
Accepted: June 09, 2010

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2$_\varepsilon$ using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.