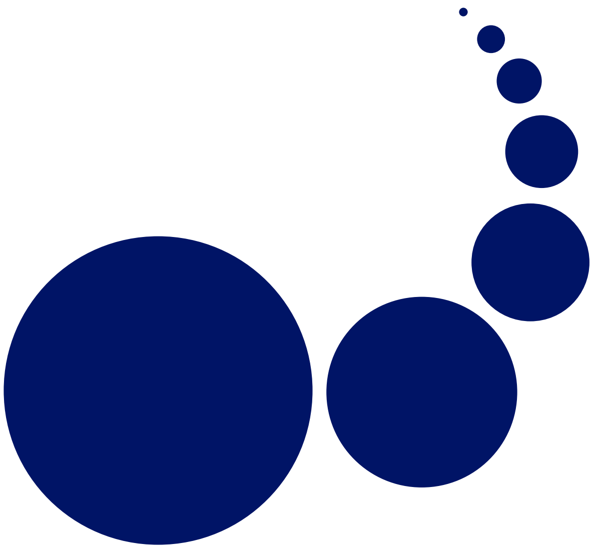
SCALABLE COMPUTING
Practice and Experience

Special Issue: Large Scale Computations on Grids

Editors: Dana Petcu, Marcin Paprzycki



Volume 9, Number 2, June 2008

ISSN 1895-1767

SWPS

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

Scalable Computing: Practice and Experience

Volume 9, Number 2, June 2008

TABLE OF CONTENTS

Special Issue Papers:

Introduction to the Special Issue **i**
Marcin Paprzycki

Object Oriented Condensed Graphs **83**
Sunil John and John P. Morrison

Simulation Models of Natural Disasters and Service-Based Visualization of Their Datasets in the Medigrid Project **95**
Peter Slížik and Ladislav Hluchý

A Fault Tolerance Solution for Sequential and MPI Applications on the Grid **101**
Gabriel Rodríguez, Xoán C. Pardo, María J. Martín, Patricia González and Daniel Díaz

Service-oriented Symbolic Computing with SymGrid **111**
Dana Petcu, Alexandru Cârstea, Georgiana Macariu and Marc Frîncu

Mobile Agent Systems Integration into Parallel Environments **127**
David E. Singh, Alejandro Miguel, Félix García and Jesús Carretero

Web portal for large-scale computations based on Grid and MPI **135**
Assel Zh. Akzhalova, Daniar Y. Aizhulov, Galymzhan Seralin and Gulnar Balakayeva

Performance Measurements of Computing Networks **143**
Igor Rozman, Marjan Šterk, Jaka Močnik, Borut Robičaška and Roman Trobec

Book Review:

The π-Calculus: A theory of mobile processes **151**
Reviewed by Raheel Ahmad

Scalable Computing: Practice and Experience
Volume 9, Number 2, p. i. http://www.scpe.org
ISSN 1895-1767
© 2008 SCPE

INTRODUCTION TO THE SPECIAL ISSUE: LARGE SCALE COMPUTATIONS ON GRIDS

The annual Large Scale Computation on Grids (LaSCoG) Workshop, as well as the Scalable Computing: Practice and Experience (SCPE) Journal, attempt at providing interactive and professional forum for the distributed computing community. Therefore the SCPE was selected to publish the extended versions of the latest (2007) LaSCoG meeting.

The 3rd LaSCoG meeting was held in Gdansk, Poland, on September 11th, 2007, in conjunction with the Seventh International Conference on Parallel Processing and Applied Mathematics (PPAM). Of the 16 submissions received, 6 were chosen for delivery and inclusion in the PPAM post-proceedings. These papers were limited to ten pages, thus their authors were invited to publish extended versions of their contributions in the SCPE. After careful re-refereeing and further improvements, it is my pleasure to provide you with the following collection of papers.

The first paper, by Sunil John and John Morrison, introduces the object oriented condensed graph, a computational paradigm which combines condensed graphs model for expressing parallelism with object oriented model as an effective programming paradigm in order to be able to develop large scale parallel and distributed systems.

In the second paper, Peter Slížik and Ladislav Hluchý presents several components of the simulation models of natural disasters developed in the frame of the European project MediGrid.

The third paper, by Gabriel Rodríguez, Xoán C. Pardo, María J. Martín, Patricia González, and Daniel Díaz, describes the design and implementation of a service-based architecture named CPPC-G (Controller-Precompiler for Portable Checkpointing on Grid) designed to manage execution of fault tolerant applications on Grids.

In the fourth paper, Dana Petcu, Alexandru Cârstea, Georgiana Macariu, and Marc Frîncu describe an approach to integrate legacy computer algebra systems into a service-oriented architecture, namely the SymGrid-Services.

The next paper, by David E. Singh, Alejandro Miguel, Félix García, and Jesús Carretero discusses MASIPE, an agent-based tool for monitoring parallel applications in Grids.

The last paper from the LaSCoG Workshop is authored by Assel Akzhalova, Daniar Aizhulov, Galymzhan Seralin, and Gulnar Balakayeva, and describes a problem solving environment implemented as Web portal. Its application is to allow Grid utilization of numerical methods that use MPI for their parallel implementations.

Finally, the seventh paper, by Igor Rozman, Marjan Šterk, Jaka Močnik, Borut Robič, and Roman Trobec, was submitted directly to this Special Issue. It analyzes communication and computational performance results on an ad-hoc computing network composed of geographically wide spread distributed nodes.

Note that while the above mentioned papers represent rather different topics, all of them are intensively studied in the last years and I hope that the reader of this issue will find the papers interesting for his or her future research activities.

I would also like to thank my co-Chair of LaSCoG—Prof. Dana Petcu—who has withdrawn herself from editing this Special Issue due to the fact that she has co-authored one of the included papers. In 2007 most work in preparation of LaSCoG was done by her and I am really grateful for all that she has done. I would also like to thank all members of the Program Committee, for their diligent work promoting LaSCoG and hard work refereeing papers. Thank you all!

Finally, it is my pleasure to invite all readers to take part in LaSCoG 2008, which will take place in Wisła, Poland, on October 20–22, 2008. There is still time to contribute. More information can be found at: http://www.lascog.imcsit.org

Marcin Paprzycki

Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 83–94. http://www.scpe.org

ISSN 1895-1767
© 2008 SCPE

OBJECT ORIENTED CONDENSED GRAPHS[*]

SUNIL JOHN AND JOHN P. MORRISON[†]

Abstract. Even though Object Orientation has been proven to be an effective programming paradigm for software development, it has not been shown to be an ideal solution for the development of large scale parallel and distributed systems. There are a number of reasons for this: the parallelism and synchronisation in these systems has to be explicitly managed by the programmer; few Object Oriented languages have implicit support for Garbage Collection in parallel applications; and the state of a systems of concurrent objects is difficult to determine. In contrast, the Condensed Graph model provides a way of explicitly expressing parallelism but with implicit synchronisation; its implementation in the WebCom system provides for automatic garbage collection and the dynamic state of the application is embodied in the topology of the Condensed Graph. These characteristics free programmers from the difficult and error prone process of explicitly managing parallelism and thus allows them to concentrate on expressing a solution to the problem rather than on its low level implementation. **Object Oriented Condensed Graphs** is a computational paradigm which combines Condensed Graphs with object orientation and this unified model leverages the advantages of both paradigms. This paper illustrates the basic features of Object Oriented Condensed Graphs as well as its support for large scale software development.

Key words: condensed graphs, object oriented systems, software engineering, distributed and parallel computing

1. Introduction. The support of large scale software systems has long been the focus of research in software engineering. Object Oriented Systems have attracted wide attention due to its many desirable properties which aid software development such as enhanced maintainability and reusability. With the support of characteristics such as inheritance, modularity, polymorphism and encapsulation, this paradigm can help the development of complex software programs [17]. However, the development of parallel and distributed applications is not significantly simplified by Object Orientation. The onus is still on the programmer to explicitly manage each parallel component and to ensure proper synchronisation. The interaction of parallel components in a large system can be complex and this complexity is compounded in sophisticated environments such as heterogeneous clusters and computational grids. Moreover, the encapsulation concept in OO is complex in a parallel environment. OO does not impose constraints upon invocation of an object's attributes or member functions. This complicates the relationship among objects when there are several method invocations. Similarly, memory management is a major concern. Most of the current Garbage Collection methodologies work sequentially; only a few OO languages can support automatic garbage collection in a parallel system. In addition, parallelism poses additional challenges for access control mechanisms and object state determination.

The Condensed Graph (CG) model of computing is based on Directed Acyclic Graphs (DAGs). This model is language independent and it unifies the imperative, lazy and eager computational models [2]. Due to its features including Implicit Synchronisation and Implicit Garbage Collection this computational model can be effectively deployed in a parallel environment. CGs have been employed in a spectrum of application domains, from FPGAs to the Grid [7, 4]. The most advanced reference implementation of the model is the WebCom abstract machine [5]. WebCom is being used as the basis of Grid-Ireland's middleware development and deployment [3].

This paper addresses the concept of **Object Oriented Condensed Graphs** (**OOCG**) as well as its development support in a large scale environment. Object Oriented Condensed Graphs is a unified model that combines the Object Oriented paradigm with the Condensed Graph methodology. This unified model helps to leverage the advantages of both paradigms. By integrating Condensed Graphs with Object Oriented principles, negative aspects of Object Orientation, when deployed in a large scale parallel environment, can be successfully addressed.

Some of the similar research in this area of modelling languages are Object Oriented Petri Nets (OOPN) [8] and Object Petri Nets (OPN) [9, 10, 11] in which the Object Orientation principles has combined with that of Petri Nets [13]. Other notable Petri Net modelling languages embodying OO concepts are CPN [12], HOOPN [14] and CO-OPN [15, 16].

This paper is organised as follows: Section 2 provides an overview of Condensed Graphs, and Section 3 presents the basics of Object Oriented Condensed Graphs. Section 4 presents Development support, particularly pattern based OOCG Development, as well as its performance analysis.

[*]The support of Science Foundation Ireland is gratefully acknowledged.

[†]Centre for Unified Computing, Dept. of Computer Science, National University of Ireland, University College Cork, Cork, Ireland (s.john,j.morrison@cs.ucc.ie). http://www.cuc.ucc.ie

2. Condensed Graphs. Like classical dataflow, the CG model is graph-based and uses the flow of entities on arcs to trigger execution. In contrast, CGs are directed acyclic graphs in which every node contains not only operand ports, but also an operator and a destination port. Arcs incident on these respective ports carry other CGs representing operands, operators and destinations. Condensed Graphs are so called because their nodes may be *condensations*, or abstractions, of other CGs. (Condensation is a concept used by graph theoreticians for exposing meta-level information from a graph by partitioning its vertex set, defining each subset of the partition to be a node in the condensation, and by connecting those nodes according to a well-defined rule.) Condensed Graphs can thus be represented by a single node (called a condensed node) in a graph at a higher level of abstraction. The number of possible abstraction levels derivable from a specific graph depends on the number of nodes in that graph and the partitions chosen for each condensation. Each graph in this sequence of condensations represents the same information but in a different level of abstraction. It is possible to navigate between these abstraction levels, moving from the specific to the abstract through condensation, and from the abstract to the specific through a complementary process called evaporation.

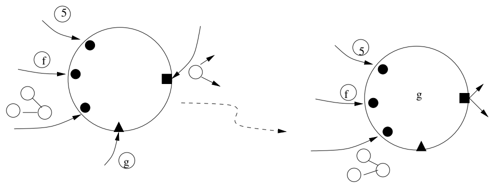


Fig. 2.1. *CGs congregating at a node to form an instruction.*

The basis of the CG firing rule is the presence of a CG in every port of a node. That is, a CG representing an operand is associated with every operand port, an operator CG with the operator port and a destination CG with the destination port. This way, the three essential ingredients of an instruction are brought together (these ingredients are also present in the dataflow model; only there, the operator and destination are statically part of the graph).

Any CG may represent an operator. It may be a condensed node, a node whose operator port is associated with a machine primitive (or a sequence of machine primitives) or it may be a multi-node CG.

The present representation of a destination in the CG model is as a node whose own destination port is associated with one or more port identifications. Figure 2.1 illustrates the congregation of instruction elements at a node and the resultant rewriting that takes place.

Strict operands are consumed in an instruction execution but non-strict operands may be either consumed or propagated. The CG operators can be divided into two categories: those that are value-transforming and those that only move CGs from one node to another in a well-defined manner. Value-transforming operators are intimately connected with the underlying machine and can range from simple arithmetic operations to the invocation of software functions or components that form part of an application. In contrast, CG moving instructions are few in number and are architecture independent.

By statically constructing a CG to contain operators and destinations, the flow of operand CGs sequences the computation in a dataflow manner. Similarly, constructing a CG to statically contain operands and operators, the flow of destination CGs will drive the computation in a demand-driven manner. Finally, by constructing CGs to statically contain operands and destinations, the flow of operators will result in a control-driven evaluation. This latter evaluation order, in conjunction with side-effects, is used to implement imperative semantics. The power of the CG model results from being able to exploit all of these evaluation strategies in the same computation, and dynamically move between them using a single, uniform, formalism.

3. Object Oriented Condensed Graphs. The Object Oriented Condensed Graphs paradigm, OOCG, has been developed to address the limitations of Object Oriented Systems mentioned in Section 1. As a unified model, Object Oriented Condensed Graphs preserves all the current functionalities of Condensed Graphs while featuring many major Object Oriented Concepts. The core features and functionalities are given below [1]:

3.1. Object Annotations. The *definition* of a Condensed Graph may be viewed as a class from which instances can be created. Such instances are analogous to objects. A class can contain nested graphs representing individual methods and attributes.

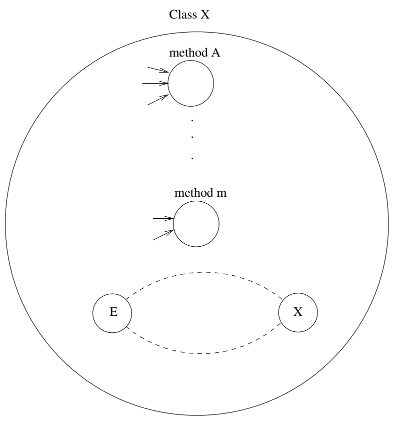


Fig. 3.1. *A Node to represent Class X. The member functions form child nodes within that node.*

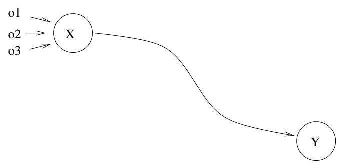


Fig. 3.2. *Invoking Object X with Operands. Sending the result to Y.*

In the CG model, Graph instance creation can occur implicitly when an appropriate CG node is invoked. Alternatively, explicit instance creation will give rise, not only to an appropriate graph instance but also, to a CG node which can be used to represent that instance. In effect, these nodes are coherent names of dynamically created objects.

OOCG specifies two kinds of visibility for its member functions and attributes. As is typical in an Object Oriented language, *private* members belonging to an Object can be accessed only within that Object. In contrast, a *public* member has a broad visibility. It can be accessed from inside as well as from outside of that Object. In other words, it has a *Global Scope*. Some of these scenarios are shown in Figs. 3.1, 3.2 and 3.3.

3.2. Inheritance in Condensed Graphs. Object Oriented Condensed Graphs incorporate *single inheritance*. A Class can inherit from other classes. In this way, a *sub class* can inherit the properties of its *super class*. The sub class can introduce new properties or can override the inherited properties of its super class. As shown in Figs. 3.4 and 3.5, subclass **Class B** extends superclass **Class A**. By default, public members of A are available in Class B and these can be used for operations within B.

The class definitions can be instantiated as Objects. The characteristics of *Polymorphism* also can be observed in the class instances. By overriding the parent class operation in the child class, the behaviour can be changed in the sub class. In the above example, we can redefine the member function *func1* in the sub class so that the whole operation behaviour can be redefined. This also gives rises *dynamic binding.* During compile time the contexts between the operations can be switched.

3.3. Concurrency and Synchronisation. OOCG adheres to the standard OO principle of Encapsulation. The public methods and attributes are visible and accessible from outside the Object. A method within an object can be subject to several invocations. These invocations can be intra Object or can be inter Object.

Condensed Graphs are inherently parallel by design. The definition of the graph dictates the parallel execution. The node dependencies within the graph explicitly specifies the order of execution. *Cross Cutting Condensed Graphs* [6] is a methodology to cater for inter graph synchronisation for concurrently executing graphs. This methodology helps to overcome the model's basic criteria that values exit a graph only through its X Node. In the cross-cutting version, values may exit via a special construct thus helping to coordinate independent graphs.

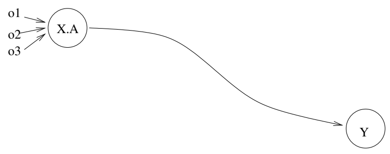


FIG. 3.3. *Invoking Method A of Object X with Operands. Sending the result to Y.*

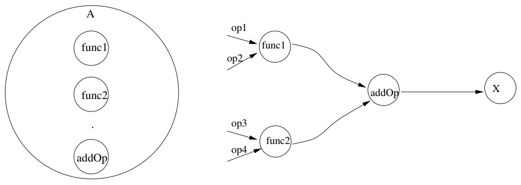


FIG. 3.4. *The parent super class A.*

A new graph model to support access to shared resources is proposed in Figure 3.6. Many operands may converge to the E node of this graph. A priority based queuing mechanism is employed in the E node to handle these accesses and a local semaphore is used to enforce sequential access to the underlying graph. Typically it sets a flag when it allows the invocation of the graph. When the X node of the graph fires this flag reverts back to its previous value.

3.4. Explicit Naming Scheme. OOCG Node names adapt from the code semantics. A Class Node bear the name of the Class in the program. A Class method name is normally preceded by a colon. As an example, the Nodes **C** and **C:getDetails** illustrate Class **C** and the method **getDetails** within class **C** respectively. Inheritance and the nested class scenarios also can be incorporated. The Node annotation **A.B.C:getDetails** illustrates the member function **getDetails** of the class **C** which is derived from base classes **B** and **A**. Some research regarding the naming scheme using the Condensed Graph model is addressed in [5].

3.5. Object Creation and Automatic Garbage Collection. Condensed Graphs support Object Creation and *Implicit Object Destruction* or Automatic Garbage Collection [1]. Object Creation can be performed by invoking the **Create** instruction. This instruction takes the Class Definition as an input Operand and creates an Object as output. Figs. 3.7 and 3.8 illustrates some scenarios of the Object creation. Typically when an Object's X Node fires the Object is collected automatically.

4. OOCG Development Support. OOCG development framework has been developed for the creation of OOCG applications for concurrent and distributed environments. **OOCGLib** is a programming package consisting of a set of Java classes and libraries to address integrated development. OOCGLib is integrated with *WebCom-*G_{IDE} [3] which allows developers to develop OOCG applications in an interactive, distributed environment (Figure 4.1). This development environment is implemented as an application layer on top of the WebCom [5] middleware. As an integrated framework, it benefits from the features of WebCom such as load balancing, fault tolerance and security policies.

4.1. Multiphase Patterns. In modelling a system, patterns often used to efficiently express solution in a standard manner. Patterns are template solutions that have been developed to address common recurring problems in particular application domains.

Gamma et al. [18] suggested recurring solutions to common problems in design. They presented their ideas in the context of Object Oriented design. Le Guennec et al. [21] has incorporated some of this work on design patterns in UML using their customised tool UMLAUT. W.M.P. van der Aalst et al. suggested *Workflow Patterns* [22] as Requirements for workflow languages. Later research in this area has also addressed control flow patterns [23] and provided a formal description of each of them in the form of a Coloured Petri-Net (CPN) model.

Patterns currently available in software engineering only apply to particular engineering phases and do not extent into the multiple phases. A pattern paradigm which addresses problems in all phases of a software

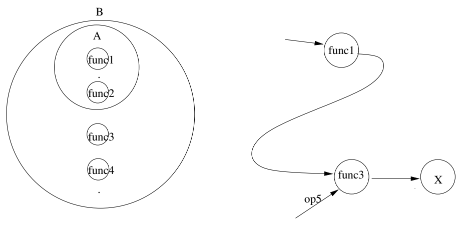


FIG. 3.5. *The child sub class B. The child class inherited the public contents of class A.*

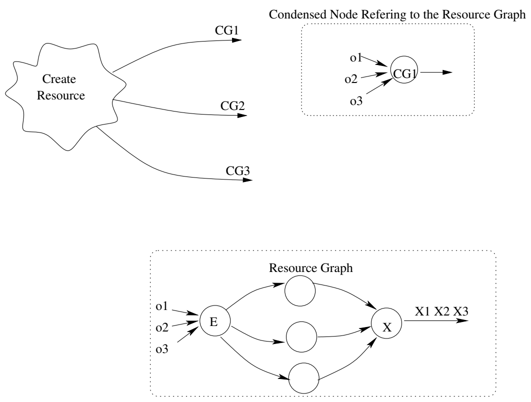


FIG. 3.6. *The Condensed Nodes refer to the Resource Graph. The E node of the Resource Graph contains a local semaphore which restricts the access to resource one at a time.*

lifecycle, from high level modelling to design, implementation and testing is currently missing from the software engineering landscape. The concept of a **Multiphase pattern** is presented here to address solutions to all phases of the software engineering lifecycle ranging from modelling, design to implementation.

Two Multiphase patterns, as part of the OOCG model, are presented in this paper to address common scenarios. These are design patterns which are being adapted and extended to cover multiple phases in the engineering lifecycle. Similar to the strategy adopted by Gamma et.al. [18], the OOCG Multiphase Patterns also name, motivate, and explain solutions for generic problems.

4.1.1. Predecessor-Successor Pattern. This pattern addresses the sequential execution of dependant tasks.

Name of the Pattern. Predecessor-Successor.

Related Patterns. Sequential Routing, Serial Routing [19]. In this Petri net based workflow pattern, a process wait for the other process for serial execution.

Motivation. In a concurrent system of dependent tasks, the order of execution of tasks is a matter of interest. As an example, if there are two dependent tasks and if any of the tasks requires a value that needs to be transferred from the other task, it is important to know their order of execution.

Description of the Problem. Consider two tasks G_1 and G_2. G_2 expects a value X_1 that needs to be transferred from task G_1. In this case, task G_1 has to be executed first to produce the value X_1.

Solution of the Problem. Induce the notion of *Sequencing* in the task modelling. If a task needs a value from another task, execute the first task prior to the second task. The value produced as a result of the execution of the first task is transferred to the second task.

Implementation. Arrange the order of execution of the tasks. The task which executes first will be denoted as *Predecessor* and the task which waits for the value from the *Predecessor* task will be termed as *Successor*. Once the value has been produced by the Predecessor task, the value will be consumed by the Successor. The value transfer will be based on the assumption that the Successor task waits for the value (Figure 4.2).

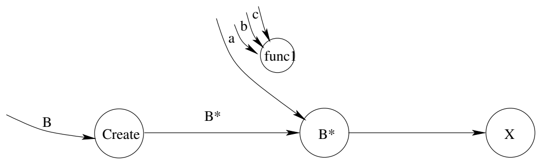


FIG. 3.7. *In this Condensed Graph realisation, object B* has been created and is used to invoke the function* func1 *with operands a,b and c.*

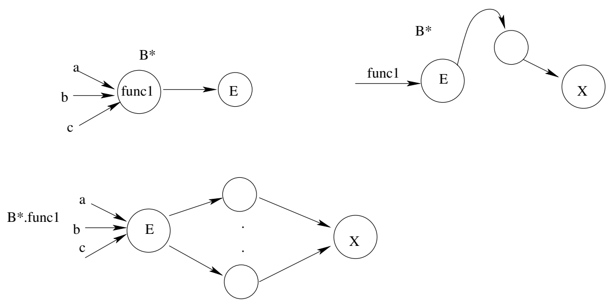


FIG. 3.8. *Different CG scenarios for the Object creation.*

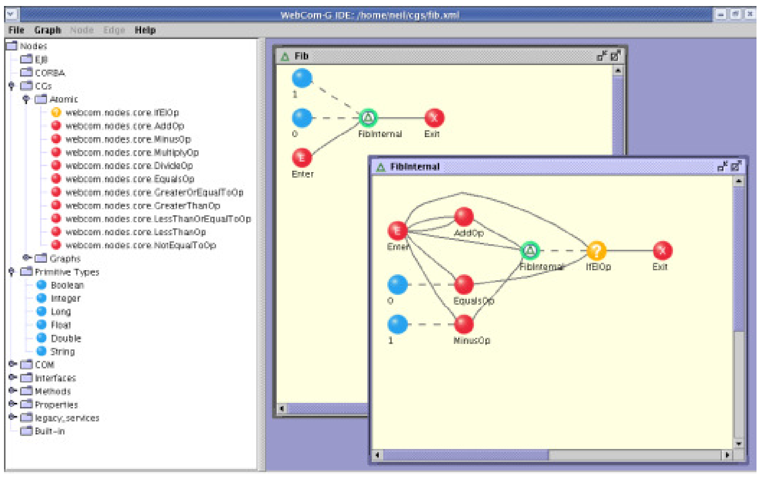

WebCom-G IDE: /home/neil/cgs/fib.xml
File
Graph
Node
Edge
Help
Nodes
EJB
CORBA
CGs
Atomic
webcom.nodes.core.IfElOp
webcom.nodes.core.AddOp
webcom.nodes.core.MinusOp
webcom.nodes.core.MultiplyOp
webcom.nodes.core.DivideOp
webcom.nodes.core.EqualsOp
webcom.nodes.core.GreaterOrEqualToOp
webcom.nodes.core.GreaterThanOp
webcom.nodes.core.LessThanOrEqualToOp
webcom.nodes.core.LessThanOp
webcom.nodes.core.NotEqualToOp
Graphs
Primitive Types
Boolean
Integer
Long
Float
Double
String
COM
Interfaces
Methods
Properties
legacy_services
Built-in
Fib
1
0
FibInternal
Exit
Enter
FibInternal
Enter
AddOp
FibInternal
IfElOp
Exit
0
EqualsOp
1
MinusOp

FIG. 4.1. *Sample Screen shot of* $WebCom\text{-}G_{IDE}$*, showing the Graphs used to generate a Fibonacci sequence.*

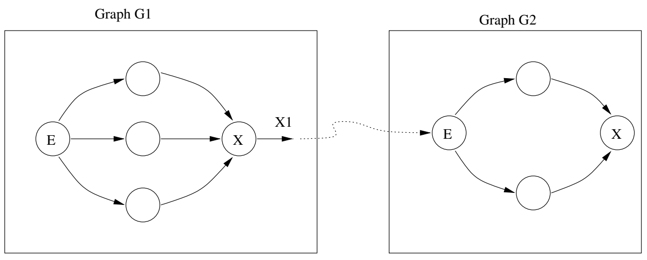


FIG. 4.2. *Predecessor-Successor pattern. Graph* G_2 *needs to wait for graph* G_1 *hence the graph* G_2 *is Successor.*

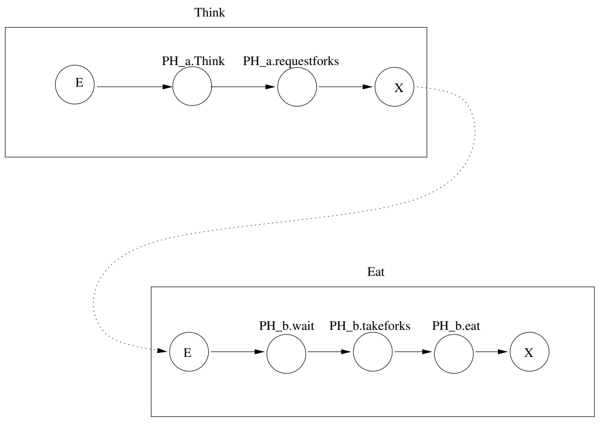


FIG. 4.3. *Graph* Eat *needs to wait for graph* Think.

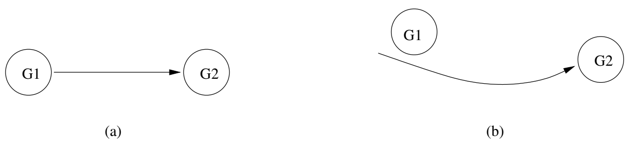


FIG. 4.4. *(a) Connected dataflow graphs and (b) Non-connected Lazy graphs. In scenario (a), G_2 needs to wait for G_1. This scenario is applicable if the Operand port in G_2 is strict. In scenario (b), G_2 can start before G_1 executes. The Operand port in G_2 is non-strict in this scenario. In the first case, G_1 is the Predecessor graph and G_2 is the Successor graph. In the second case (b), G_1 becomes Successor and G_2 becomes Predecessor.*

Example. Consider the classical dining philosophers problem [20]. In this example, a group of philosophers sit around a table sharing one fork between each pair of philosophers. In this problem the forks are a shared resource and in order to eat a philosopher must have the two forks on either side of him/her. In Figure 4.3, graph *Eat* has to wait for the value *fork* from the graph *Think*. As is evident from this scenario, *Think* acts as an *Predecessor* graph while *Eat* as the *Successor* graph.

Predecessor-Successor in OOCG. The sequencing of the execution of nodes in an OOCG may be *lazy* or *eager*. The actual order of evaluation is determined by the strictness of Node Operand Ports and the form of the Operand, i. e., whether the Operand is a normal order form or is in need of further reduction. An Eager evaluation sequence results when the operand ports of a node are *non-strict* regardless of the form of the operands. When the operand ports are *strict*, eager evaluation will result when the Operands is in normal order form and lazy Evaluation will be triggered when they are not in normal order form.

Figure 4.4 illustrates these two scenarios with tasks G_1 and G_2. Figure 4.4 (a) depicts a *strict* scenario in which the output of task G_1 is required for input of task G_2. Task G_2 will not execute without this input. This is *eager* mode of graph sequencing. In the *non-strict* sense, illustrated in Figure 4.4 (b), task G_1 represents the operand of task G_2. G_2 can be Predecessor in this *lazy* scenario. In certain cases, G_1 need not be evaluated at all.

In Figure 4.5, Multi-relationship between the graphs are depicted. In this case as well, *strictness* of the port determines Predecessor-Successor relationship.

In the Lazy way of graph sequencing in OOCG, Predecessor becomes Successor as the Successor becomes lazy. Thus, in lazy evaluation, Predecessor and Successor change their roles. The existence of Predecessor is used as the mechanism to drive the execution of Successor.

Transferring of data between the tasks is often performed using data *pipelining* in this pattern. Apart from *pipelining* of value, OOCG can also support data *streaming* between the tasks in which the values will be transferred in bulk as *streams*. In streaming, when the first value arrives as an operand to a task node, the task fires. In order to stop the task from executing twice, the Operand will be *deconstructed*. As soon as the first execution is finished, it paves way for the second set of value in the stream. The task fires again and makes way for the next stream of values.

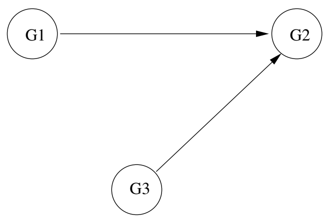


FIG. 4.5. *G_1 is Predecessor to G_3. If the port in G_2 is strict, G_3 and G_1 are Predecessors to G_2. Strictness of the port determines Predecessor-Successor relationship.*

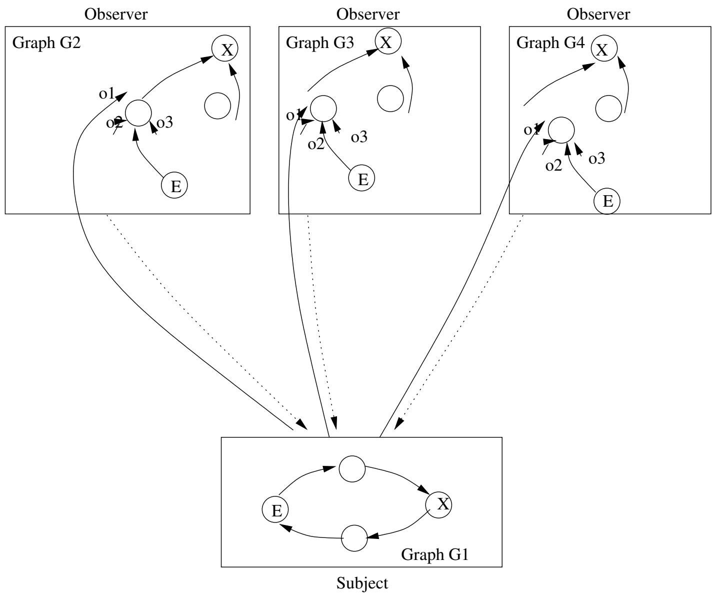


FIG. 4.6. *Observer pattern. Subject Graph* G_1 *notifies Observer Graphs* G_2*,* G_3 *and* G_4*.*

4.1.2. Multiphase Observer Pattern. This pattern is based on the Observer pattern [18] introduced by Gamma et al. **Multiphase Observer Pattern** is an extended version of Observer pattern with Multiphase concepts. This is categorised as a *Behavioural Pattern*.

Name of the Patter. Multiphase Observer

Also Known As. Publish-Subscribe

Motivation. If there are multiple dependent tasks and if a task's state change affects the state of other tasks it is important that there should be notification mechanism between the tasks so as to ensure a proper synchronisation.

Description of the Problem. Figure 4.6 illustrates the problem addressed by this pattern. Task G_1 is dependent on tasks G_2, G_3 and G_4. The execution of task G_1 affects the decision making within the tasks G_2, G_3 and G_4. A synchronisation mechanism is needed in this state change event.

Solution of the Problem. The task upon which other tasks are dependent is known as the **Subject** and the dependent tasks which observes the state change event from the Subject are known as **Observers**. A Subject can have any number of Observers. Whenever a Subject's state changes, it notifies its Observers. In response, each Observer synchronise its state with the Subject's state.

Implementation. Establish a hash table to maintain Subject-to-Observer task mapping. For each change the Subject notifies the Observer tasks. This notification mechanism can be modelled as **push** or **pull** [18]. In the **push model**, the subject sends detailed information to the Observers about the state change, whether they require it or not. In the **pull model**, however, the Subject sends minimal notification and Observers enquire explicitly thereafter.

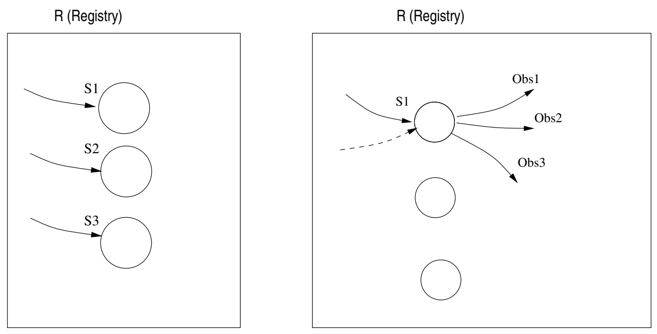


FIG. 4.7. *OOCG implementation of Observer pattern. The node in the registry are created by the Subject to reflect the state elements. Observers register as destination nodes of the Subject. By registering, they become part of the destination of the state node.*

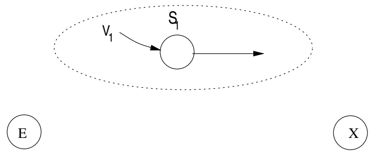


FIG. 4.8. *Subject graph. Value V_1 is being sent to the State Node S_1. V_1 is deconstructed after State Node S_1 is fired.*

Examples. In the dining philosophers problem mentioned before, the task which holds the fork acts as the *Subject* and the tasks which request forks act as *Observers.* When the Subject releases the fork, it notifies the Observers.

Observer Pattern in OOCG. In the OOCG adaptation of Observer pattern, the Subject-Observer relationship is maintained by means of registry mappings. The registry entry is created by the Subject. The nodes in the registry are created by the Subject to reflect the state elements.

Observers register destination nodes with the registry to be advised of change in state elements. By registering they become part of the destination of the appropriate *State Node* (Figure 4.7). In the Figure 4.7, Operand value V_1 must be passed onto the Observers. As Operator, Subject generates this value and Observer, as destination, consumes this value. The Operator copies the Operand value to the destination.

In the Subject graph, there is a link from the state element to the Registry (Figure 4.8). The notification operands are deconstructed when the state node is fired. When the E node of the Observer graph fires, it registers destination nodes with Registry. Similarly, when its X node fires, it removes the destination link from the Registry.

The Subject-Observer state changes can be by means of **push** or **pull** model. In the push model, Operand value converge to the Subject and passes to the Observer destination. In the pull model, Operator *pulls* the value from registry and send to the destination. After sending the value, the destination will be deconstructed. When a new Observer creates, it pulls the value from the registry. In this *Polling* mechanism, new State value arrives and overrides the old value.

4.1.3. Performance Analysis. A sample application based on OOCG is presented here to analyse the performances of the above mentioned patterns. This is an image processing application which applies compression algorithms on a series of images. The execution platform used for this experiment is WebCom [4] middleware. The main OOCG program consists of three sub-programs (*services*). When the main program is executed on the Client, the services are invoked on the hosts with service components interact each other. The initial graphs modelled from the system requirements are shown in Figure 4.9. It should be noted that these OOCG models are depicted as *Eager* graphs since the *Lazy* version is not suitable for this application context.

The Services A, B, and C are situated in different hosts. The Predecessor Successor pattern is applied between the Service CGs in order for proper execution of Service Components. Similarly, Observer pattern is implemented for interaction between the Client and the Service Hosts. In this implementation, the Ob-

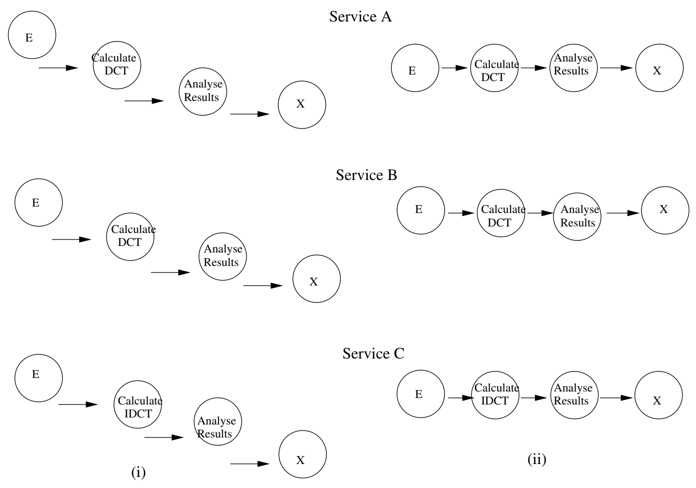


Fig. 4.9. *Initial OOCG models. (i) Lazy version, and (ii) Eager version. Since the tasks are dependent with each other, Lazy scenario is not applicable here.*

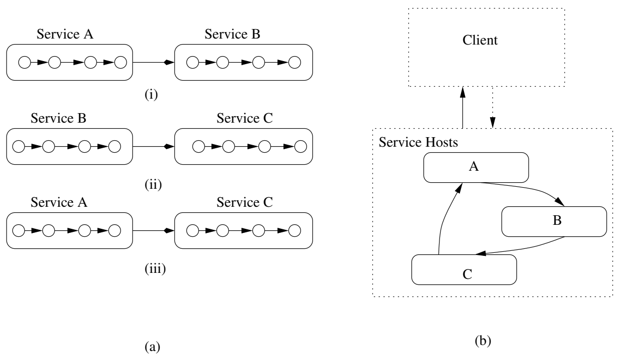


Fig. 4.10. *Some patterns implemented within the OOCG application. (a) Predecessor-Successor: (i) A-before-B, (ii) B-before-C, and (iii) A-before-C. (b) Observer Pattern. Client waits for the results from the Service Hosts.*

server (Client) waits for results from the Subject (Service Hosts). These pattern implementations are shown in Figure 4.10.

The performance of this setup has been observed by measuring the request-response interaction between Services. Performance is thus quantified as the average Request-Response time.

The experimental system is composed of three services and within each service, request to access other services are implemented as request threads. Similarly, responses also are implemented as response threads which will respond to the requests. The system can be fine-tuned by adjusting the number of concurrent requests. It can be observed from the experiments that the performance *decreases* with the increasing number of concurrent requests. Figure 4.11 illustrates this experimental results.

It has been observed from the initial experiments that a performance bottleneck occurs when the number of concurrent requests is high. The reason behind this performance bottleneck is due to the deadlock between the concurrent requests when their number is very large. The experiments have been re-run with a change of design; in which the Predecessor-Successor pattern connecting the Services is replaced by an Observer pattern. In this new design, a *Consumer* Service requests for data from the *Provider* and the Provider Service provides the data as soon as it is available. If the data is not readily available, the Consumer Service *waits* till that is available. With this enhanced design of the notification mechanisms, the deadlock between the Services is avoided. The updated result is shown in Figure 4.12.

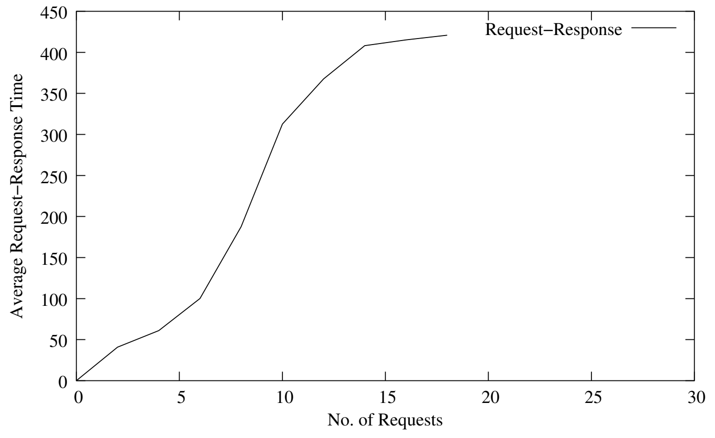


FIG. 4.11. *Average Request-Response time with respect to the concurrent Requests.*

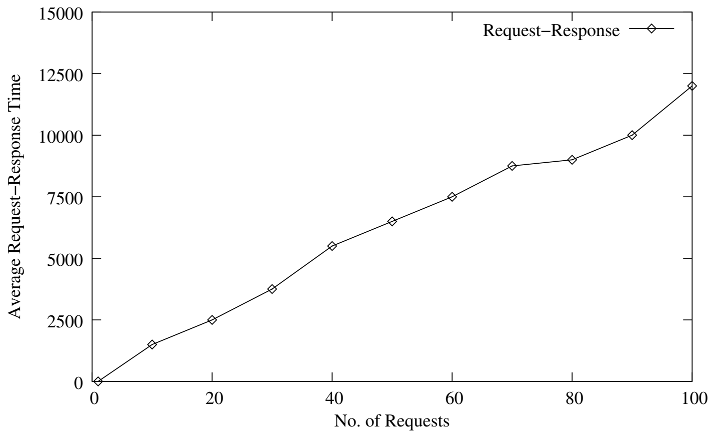


FIG. 4.12. *Average Request-Response time by removing performance bottleneck.*

5. Conclusions and Future Work. The contributions of this paper are the introduction of enhanced version of Condensed Graphs, Object Oriented Condensed Graphs, and its development support in a large scale environment. OOCG is a unified model that combines the Condensed Graphs methodology with object orientation and this leverages the advantages of both paradigms. This unified computational model is beneficial for the development of Large Scale Parallel Systems since the CG aspects allows the developer to think about parallelism and the Object Oriented aspects allow the developer to address the large scale concepts. OOCG provide implicit support of Synchronisation and Garbage Collection capabilities, which help the development of concurrent software. The Object concept and features such as Encapsulation and Inheritance enhance the Reusability and Maintainability of OOCG.

The existing implementation of Condensed Graphs has been extended to implement the OOCG features. Engineering practises have been proposed and implemented for its development in a large scale environment. OOCG model has been integrated into the Condensed Graph Integrated Development Environment, *WebCom-G*$_{IDE}$ [3] which is a development tool that enables visual application development and optimisation. Using the Integrated Development Environment, OOCG applications are created and deployed for concurrent and distributed environments.

Eventhough Object Oriented Condensed graphs have the potential for large scale software development, some limitations in the current model need to be taken into account. These limitations include the model's inability to create reusable datastructures, deficiency to interoperate with other paradigms and the lack of data management capabilities. These limitations would be addressed as future enhancements of the model.

REFERENCES

[1] Sunil John and John P. Morrison: Garbage Collection in Object Oriented Condensed Graphs, *Springer-Verlag LNCS, 3rd Workshop on Large Scale Computation on Grids,* LaSCoG 2007, held jointly with the 7th International Conference on Parallel Processing and Applied Mathematics, Gdańsk, Poland, September 9–12, 2007.

[2] John P. Morrison, Condensed Graphs: Unifying Availability-Driven, *Coercion-Driven and Control-Driven Computing,* PhD Thesis, Eindhoven: 1996.

[3] John P. Morrison, Sunil John, David A. Power, Neil Cafferkey and Adarsh Patil: A Grid Application Development Platform for WebCom-G, *IEEE Proceedings of the International Symposium of the Cluster and Grid Computing,* CCGrid 2005, Cardiff, United Kingdom.

[4] John P. Morrison, Brian Clayton, David A. Power and Adarsh Patil: WebCom-G: Grid Enabled Metacomputing, *The Journal of Neural, Parallel and Scientific Computation.* Special issue on Grid Computing. Vol 2004(12), pp 419–438. Guest Editors: H. R. Arabnia, G. A. Gravvanis and M. P. Bekakos. September 2004.

[5] John P. Morrison, David A. Power and James J. Kennedy: An Evolution of the WebCom Metacomputer, *The Journal of Mathematical Modelling and Algorithms: Special issue on Computational Science and Applications,* 2003(2), pp. 263–276, Editor: G. A. Gravvanis.

[6] Barry P. Mulcahy, Simon. N. Foley and John P. Morrison: Cross Cutting Condensed Graphs, *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA),* Vol. 3, pp. 965–973, Las Vegas, Nevada, USA, June 27–30, 2005.

[7] John P. Morrison, Philip D. Healy and Padraig J. O'Dowd: Architecture and Implementation of a Distributed Reconfigurable Metacomputer, *International Symposium on Parallel and Distributed Computing,* Ljubljana, Slovenia, October 13–17, 2003.

[8] J. Niu, J. Zou, and A. Ren: OOPN: Object-oriented Petri Nets and Its Integrated Development Environment, Proceedings of the Software Engineering and Applications, SEA 2003, Marina del Rey, USA.

[9] C. A. Lakos: From Coloured Petri Nets to Object Petri Nets, *Proceedings of the Application and Theory of Petri Nets,* volume 935, Springer-Verlag, Berlin, Germany, 1995.

[10] C. A. Lakos: Object Oriented Modelling with Object Petri Nets, *Advances in Petri Nets,* LNCS, Springer, Berlin 1997.

[11] C. D. Keen and C. A. Lakos" A Methodology for the Construction of Simulation Models Using Object Oriented Petri Nets, *Proc. of the European Simulation Multi-conference,* 1993, 267–271.

[12] Sarah L Englist: Colored Petri Nets for Object Oriented Modeling, Ph. D. Dissertation of University of Brighton, June 1993.

[13] T. Murata: Petri Nets: Properties, Analysis and Applications, *Proc. of the IEEE,* 77(4), 1989, 541–580.

[14] J. E. Hong and D. H. Bae: HOONets: Hierarchical Object-Oriented Petri Nets for System Modeling and Analysis, *KAIST Technical Report CS/TR-98-132,* November 1998.

[15] D. Buchs and N. Guelfi: CO-OPN: A Concurrent Object Oriented Petri Net approach, *12th Int. Conf. on Application and Theory of Petri Nets,* pp. 432–454, Aahrus, 1991.

[16] Gul A. Agha: Fiorella De Cindio and Grzegorz Rozenberg, *Concurrent object-oriented programming and petri nets: advances in petri nets,* Springer-Verlag New York, Inc., Secaucus, NJ, 2001.

[17] Bertrand Meyer: Object-Oriented Software Construction, second edition, Prentice Hall, ISBN 0-13-629155-4, 1997.

[18] Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides: Design Patterns: Elements of Reusable Object Oriented Software, Addison-Wesley Professional Computing Series.

[19] Li Xi-Zuo, Han Gui-Ying and Kim Sun-Ho: Applying Petri-net-based reduction approach for verifying the correctness of workflow models, *Wuhan University Journal of Natural Sciences,* Wuhan University Journals Press, Volume 11, Number 1, January, 2006.

[20] Edsger Wybe Dijkstra: The Structure of the the Multiprogramming System, *Communications of the ACM,* 11(5): 341–346, May 1968.

[21] Alain Le Guennec, Gerson Sunyé and Jean-Marc Jézéquel: Precise Modeling of Design Patterns, *UML 2000—The Unified Modeling Language. Advancing the Standard,* Third International Conference, York, UK, October 2000, Proceedings.

[22] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski and A. P. Barros: Workflow Patterns, Springer-Verlag, *Distributed and Parallel Databases,* 14(3), pages 5–51, July 2003.

[23] N. Russell, A. H. M. ter Hofstede, W. M. P. van der Aalst and N. Mulyar: Workflow Control-Flow Patterns: A Revised View, *BPM Center Report BPM-06-22,* BPMcenter.org, 2006.

Edited by: Dana Petcu, Marcin Paprzycki
Received: April 30, 2008
Accepted: May 18, 2008

Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 95–100. http://www.scpe.org
ISSN 1895-1767
© 2008 SCPE

SIMULATION MODELS OF NATURAL DISASTERS AND SERVICE-BASED VISUALIZATION OF THEIR DATASETS IN THE MEDIGRID PROJECT[*]

PETER SLÍŽIK[†] AND LADISLAV HLUCHÝ[†]

Abstract. Computational models of natural disasters are invaluable means of disaster reconstruction, crisis management, and disaster prevention. The Medigrid project addressed these issues by developing a framework of simulation models developed in previous projects. The paper describes the models incorporated into the project, together with the visualization service developed for presentation of models' outputs.

Key words: scientific visualization, grid computing, Web services, map services

1. Introduction. This paper describes the Medigrid project, a European RTD project, objective of which was to develop a framework for multi-risk assessment of natural disasters and integrate models developed in the previous projects [1]. The paper is an extension of the paper [2] presented at PPAM 2007[1].

The text puts special emphasis on the models of disasters and on the visualization tools used in the project. The visualization service, which was the core of paper [2], was later extended with its three-dimensional version, which is also discussed here.

2. Simulation of Natural Disasters. Natural disasters have been addressed by many research projects. The ability to model a disaster on a computational system is invaluable. There are three main reasons which motivate the research and development of disaster models:

Reconstruction of past events. A comparison of simulated disasters with the actual event records helps reconstruct the most probable cause of the disaster (like the location of ignition points in case of fire) and evaluate the efficiency of disaster fighting activities.

Simulation of ongoing disasters. Simulation of ongoing events is a strong decision support tool for risk management teams. The models allow to forecast different scenarios of disaster propagation and help find best places for placement of barriers or identify locations that must be evacuated urgently.

Simulation of potential disasters. Fictitious disasters are simulated for two reasons: first, to assess the degree of damage that even the most unprobable events can cause and be prepared for them; and second, for educational and training purposes for mitigation teams.

3. The Medigrid Project. As stated before, the goal of the Medigrid project was to integrate simulation models of natural disasters developed in the previous projects and develop a unified framework for their management. Unfortunately, some of the models were interactive-only (i. e., incapable of being run in batch environment); some other were just not grid-aware. In order to be able to run in the Grid infrastructure, they had to be gridified. The task was made more difficult by different requirements of applications; as they had been developed unaware of one another, they were both sequential and parallel, and running both on Unix and Windows operating systems. The state-of-the-art Grid infrastructure (particularly the GridFTP service) supported data exchange between Unix applications only; data communication with Windows-based applications had to be implemented from scratch.

3.1. Technology. The models provided by the members of the project consortium can be looked upon as a set of loosely coupled services [3]. In order to make them accessible from the standard Grid environment, each of the system components is exposed as a web service. The WSRF technology [4] was chosen as the basic implementation framework. This technology also helps glue the individual components together in a workflow-like manner.

Understandably, the big challenge of the Medigrid project was the dual Linux and Windows nature of the models; the reason being that the models were already developed by project partners or third parties

[*]This work was supported by projects MEDIGRID EU 6FP RTD SustDev Project: Mediterranean Grid of Multi-Risk Data and Models (2004–2006), GOCE-CT-2003-004044 (Call FP6-2003-Global-2, STREP) and VEGA No. 2/6103/6.

[†]Institute of Informatics, Slovak Academy of Sciences, Dúbravská cesta 9, 845 07 Bratislava, Slovakia, {peter.slizik, hluchy.ui}@savba.sk

[1]PPAM 2007—Seventh International Conference on Parallel Processing and Applied Mathematics, September 9–12, 2007, Gdańsk, Poland.

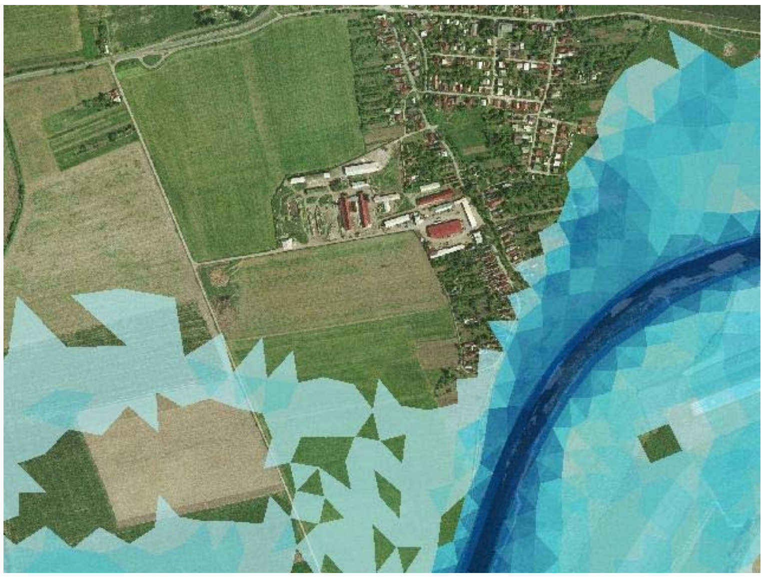


Fig. 3.1. *Visualization of a flood simulation on the Váh river, Slovakia. The water poured out of the river bed and endangers the village. The intensity of colour shows the depth of the water. The colour turns into black in the river bed.*

respecting the local conventions. For the need to support both platforms, the Java implementation of the WSRF specification by the Globus Alliance had been chosen.

The Medigrid architecture consists of six core services: *Data Management Service* for platform-independent management of data sets, *Data Transfer Service* for the copying of data between cooperating services, *Metadata Catalogue Service* for publishing, discovery, and access to metadata for large-scale datasets, *Job Management Service* for automated platform-independent management of applications, and the *Geovisualization service* for drawing the simulation results to maps. An essential part of the system is the *Distributed Data Repository*, which is a decentralised storage for both, the input digital maps, and the outputs produced by the simulations.

The whole system is accessible via a web portal. Application specific portlets allow users to invoke all services in application-specific manner. There are portlets for browsing input data, simulation results, their respective metadata, and also portlets for monitoring and house-keeping functions. The GridSpere portal [5] was chosen as the implementation platform for its support of portlet specification.

3.2. Models. The disaster simulation models incorporated into the Medigrid project include simulation of floods, landslides, forest fires, and soil erosion. All models were developed by the respective project members, except of the models used in the flood simulation application, which were developed by third-party institutions.

3.2.1. Flood Forecasting. The flood forecasting application consists of several simulation models (meteorological, hydrological, and hydraulics). The models are connected into a cascade; outputs from one model are used as inputs for the next one. Meteorological models are used to forecast precipitation, which is used by the hydrological model to compute river discharge. That is used in turn in the final step for the actual computation of a possible flood by the hydraulics model. The output data generated by the models are then used to generate maps visualizing the simulation.

The flood prediction application supports two meteorological models, they can be used interchangeably. The first one, *Aladin* [6], is a mesoscale meteorological simulation model developed by Meteo France. Meteorological models solve the so-called "atmosphere equations" on a regular terrain grid. The global model (operated by Meteo France) computes forecasts for the whole Earth on a small-resolution grid, the global weather situation is then used to set the boundary condition for a locally-operated high-resolution model.

The other model, *MM5* [7], is a limited-area, terrain-following model designed to simulate and predict mesoscale and regional-scale atmospheric circulation. Since MM5 is a regional-area model too, it requires the initial conditions supplied by the global model.

The computed precipitation is an input to HSPF, the hydrological model. *HSPF (Hydrological Simulation Program—Fortran)* [8] simulates the hydrologic water quality, processes on pervious and impervious land sur-

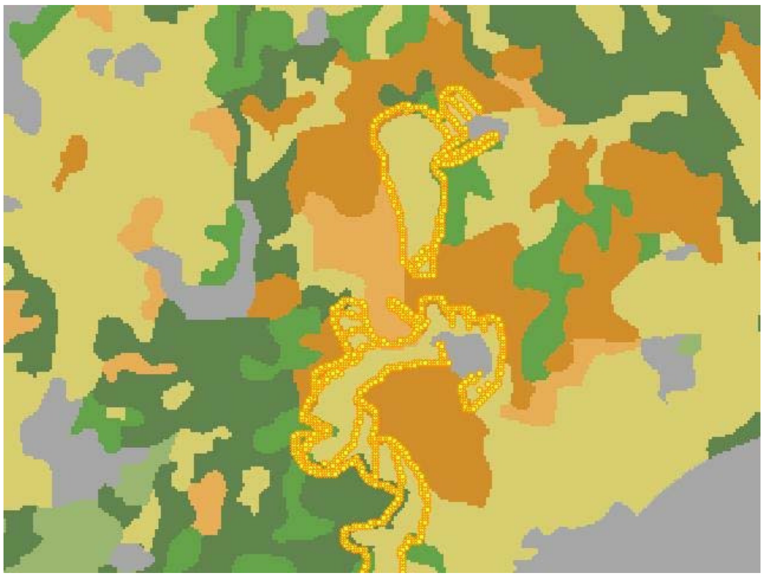


Fig. 3.2. *Visualization of a forest fire simulation. The different background colours represent different types of vegetation (grass, shrubs, trees, etc.) The small circles show the progress of the fire. The spread of the fire corresponds with the areas of grass cover.*

faces and in streams for extended periods of time. HSPF uses continuous rainfall and other meteorologic records to compute hydrographs. HSPF simulates many quantities, such as interception soil moisture, surface runoff, snowmelt, and evapotranspiratrion. Only a few values, most importantly river discharge, are interesting for the consecutive hydraulic model.

The river discharge computed by HSPF is fetched to *DaveF*, a hydraulic finite-element simulation model. The model works with limited irregular terrain mesh; a few values such as water depth, direction, and speed are computed for each mesh element in a given time step. The output of this model is treated as the input for the visualization module, which in turn draws DaveF's simulated data into the map. The map shows the depth of water in the river bed and the adjacent areas, giving the crisis teams hints as to which regions need special attention (see Fig. 3.1).

3.2.2. Water Flow and Sediment Transport. Water flow and sediment transport in river basins were modelled by the *SHETRAN* model. The model provides the hydrological and sediment transport framework for the landslide model. It can be applied to a single basin, to parts of a basin and to groups of contiguous basins. The model depends on meteorological data such as precipitation and evapotranspiration and catchment property data such as topography, soil types, vegetation types, and sediment characteristics. Output includes flow rates, rates of ground surface erosion, sediment discharge rates, and debris flow rates. The model consists of three components: water flow component, sediment transport component, and landslide component.

The water flow component deals with surface water flow on the ground surface and in stream channels. The following processes are simulated: canopy interception of rainfall, evaporation and transpiration, infiltration to surface, and surface runoff.

The sediment transport component deals with soil erosion and multifraction transport on the ground surface and in stream channels. The simulated processes include erosion by raindrop, deposition and storage of sediments, erosion of river beds and banks, and deposition on river bed.

The landslide component simulates the erosion and sediment yield associated with shallow landslides. The simulated processes include landslide incidence, debris flow transport, direct sediment delivery to the channel system, and transport of sediment along the river system.

The simulation model identifies regions of the basin that are at risk from landslides and calculates the soil saturation conditions critical for triggering a landslide at a given location.

3.2.3. Fire Danger and Propagation. The Medigrid project employed two fire danger and propagation models, the FireStation simulator and Algosystems fire simulator.

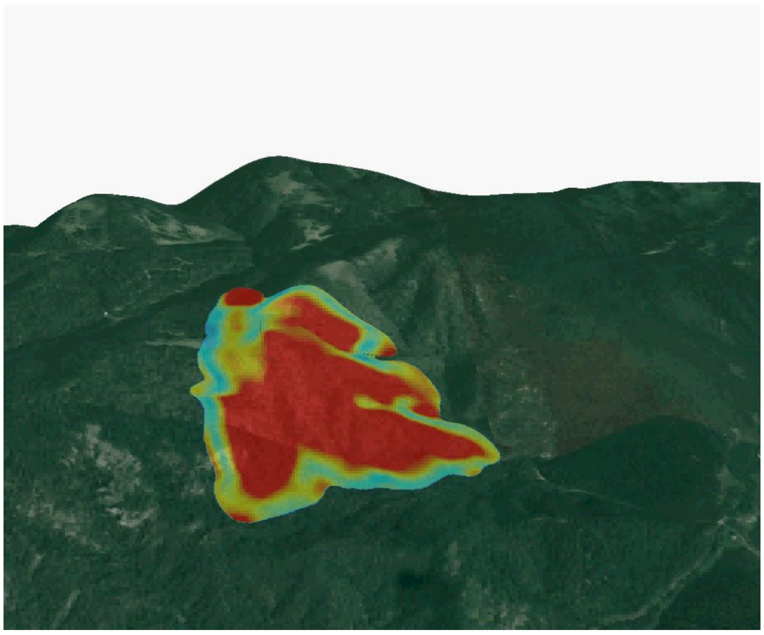


Fig. 4.1. *Visualization of a forest fire in the Krompľa region, Slovenský Raj National Park, Slovakia.*

FireStation simulator implements a semi-empirical model for the rate of fire spread. The simulator works with the elliptical model. It is based on Huygens' principle, which states that each point of the fireline becomes a new ignition point and starts a new local fire in the shape of an ellipse. The front line of the fire is represented by the envelope of all ellipses generated at each point.

Algosystem simulator calculates the danger of occurence of a forest fire in a known geographical target area and simulates the propagation of such a fire, which is deemed to have started within this area.

Both systems work, in their particular way, with geographical data (terrain, slope, and fuel coverage data), meteorological data (wind speed and direction, temperature and relative humidity of air, rainfall), and the location of the point(s) of ignition as inputs.

Based upon the data, the FireStation software computes the evolution of the fire shape in time, the rate of the spread of fire, and fire intensity. The Algosystems software computes a map indicating the danger of fire occurence, and an abstract representation of a series of contours that indicate the projected state of a fire at the various points in the simulation period (see Fig. 3.2).

In addition to local-level prediction, the models also incorporate a fire danger rating system applicable at regional and national level. The rating system incorporates the Canadian Fire Weather Index, which is a set of indicators for the easiness of ignition and spread, the moisture content of loosely compacted forest floor organic layers, and the moisture of deep and compacted floor organic layers. The Fire Weather Index is the final component of the system and gives a numerical rating of the potential frontal fire intensity.

The algorithms for calculating the danger indices work cumulatively. That means the value calculated for an index for a given day takes into account the value computed for the previous day. Over time, the indices become more accurate and more linked to the meteorological history of the target area.

4. Visualization. Historically, each of the supported models had its own means of data visualization. For instance, the results of the third-party DaveF flood simulation model were displayed by a system based on the GRASS GIS software [9]. Other models implemented their own visualization engines. For compatibility reasons, a decision was made to create a unified way of data visualization. The Geovisualization service accomplished this for 2D maps, the 3D visualization tool developed later has accomplished it for 3D virtual worlds.

4.1. Geovisualization Service. The Geovisualization Service, treated thoroughly in [2], is built upon the client-server architecture. The service consists of two parts, relatively independent, tied only with data dependencies. This split is necessary in order to ensure quick response of the interface to the user actions.

The first, non-interactive part is responsible for preparing the simulation outputs for the rendering part. This part parses the input files, does the necessary conversions, prepares colour palettes, generates templates, etc.

The second, interactive part, provides the user interface. It consists of two components, a server-side component responsible for data rendering, and a client-side part.

The server-side component is based on Minnesota MapServer [10], an open-source, multi-platform framework for the development of spatially enabled web applications. Unlike full-featured GISes, which are capable of complex data analyses, map servers focus strongly on data rendering. Their outputs are designed for embedding into web pages. MapServer supports many vector and raster data formats through third-party OGR [11] and GDAL [12] libraries. MapServer is valued for the easiness of configuration, simple programming model, and excellent documentation.

The client is invoked by the user from the internet browser. The Medigrid portal provides a suitable map client for easy access to geographical data and simulation results.

4.2. The 3D Visualization Tools. The 3D visualization tool is a successor to the Geovisualization Service. Its task is to create VRML-based 3D worlds from the same input data that the Geovisualization Service uses.

The 3D tool works in three phases. Firstly, the input data are collected and fetched from the simulation application. In the next step, they are converted into a displayable form (virtual scenes). The last step is the display of virtual scenes on user's device. The first phase runs in the Grid, the last one on the user's computer. The second phase can be run in both environments, depending on the decision of the designer. Placing this phase into the Grid environment enables it to use vast Grid computational resources.

Using an appropriate plugin, the VRML virtual scenes can be displayed in a standard web browser (see Fig. 4.1). The scenes have already been successfully ported to a full-featured Virtual Reality device (see Fig. 5.1).

4.3. Input and Output Data. The data with which the service works differ according to the simulation being computed; however, they share some common core. Typically, each visualization uses a background image that serves as a reference to which other map elements are related. The outputs of simulations are rendered on this background; sometimes assissted with another reference mesh. The following four types of data are seen in virtually all visualization systems for geographical simulations:

Terrain texture. It is an image (usually an orthophotomap) used as a background for referencing other map elements.

Terrain model. Often referred to with acronyms DEM (Digital Elevation Model) and DTM (Digital Terrain Map). Used by the 3D visualization tool.

Reference structure. A mesh of polygons or a set of points that the simulation algorithm actually works with. Drawing this structure on the map may prove helpful for the end user.

Simulation results. The actual (numerical) results of simulations. They almost always represent time steps showing the development of the disaster in time.

The terrain model and terrain texture files use raster data formats. The Medigrid system directly supports ARC/INFO ASCIIGRID [13] [14] and GRASS ASCII [15] formats. Other formats are supported by the GDAL library [12] (used by MapServer). The reference structure and other auxiliary data are stored in vector formats. ESRI Shapefile [16], MapServer's native format, is supported directly. Other vector formats are supported through the OGR library [11]. Imagery data are stored in general image formats such as JPEG, GIF, PNG, or TIFF.

5. Conclusions. The Medigrid project has succeeded in creating a framework for multi-risk assessment of natural disasters. The provided models were gridified (accomodated for the use in Grid infrastructure) and the interactive models were adapted for the use with the WSRF protocol. Together with the supporting infrastructure and the web portal, the project has created an easy-to-use virtual laboratory for running natural disaster simulations.

The Geovisualization Service has provided an unified means of visualization for the simulation models. Both, spatial and temporal progress of an event can be visualized. The service is scalable, e.i., it is able to display simulations consisting either of a few or a few hundreds of time steps.

Acknowledgements. The authors would like to thank the members of the Medigrid Consortium for providing their advice and expert knowlegde during the development of the project.

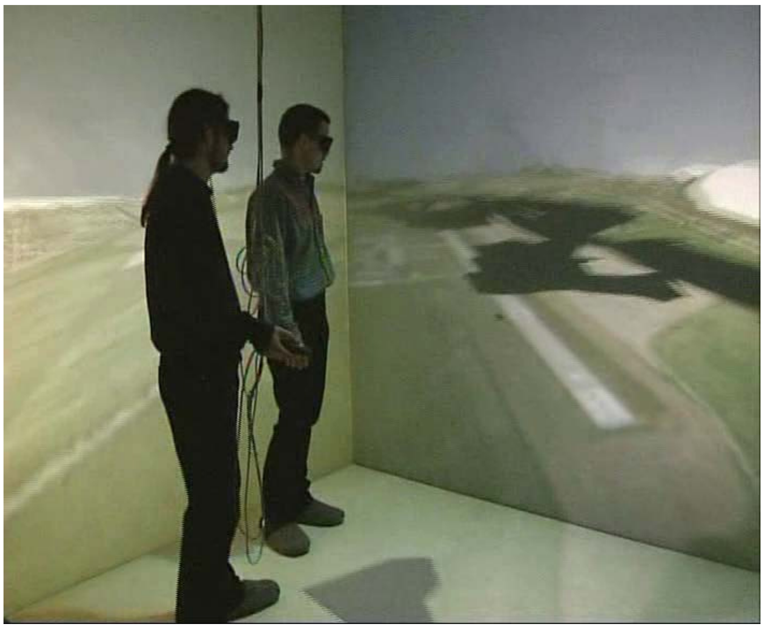


FIG. 5.1. *Visualization of a simulated flood on the Váh river in the CAVE virtual reality device. Virtual Reality Center, Institute of Graphics and Parallel Processing, Johannes Kepler University Linz, Austria.*

REFERENCES

[1] L. HLUCHÝ, O. HABALA, G. NGUYEN, B. ŠIMO, V. TRAN, M. BABÍK, *Grid computing and knowledge management in EU RTD projects of IISAS*, in Proceedings of 1st International Workshop on Grid Computing for Complex Problems—GCCP 2005, VEDA, Bratislava, Slovakia, 2006, pp. 7–19, ISBN 80-969202-1-9.

[2] P. SLÍŽIK, L. HLUCHÝ, *Geovisualisation Service for Grid-based Assessment of Natural Disasters*, LNCS 4967, 2008, presented at PPAM 2007—Seventh International Conference on Parallel Processing and Applied Mathematics, September 9–12, 2007, Gdańsk, Poland. —*in print*—

[3] B. ŠIMO, M. CIGLAN, P. SLÍŽIK, M. MALIŠKA, M. DOBRUCKÝ, *Mediterranean Grid of Multi-Risk Data and Models*, in Proceedings of 1st International Workshop on Grid Computing for Complex Problems—GCCP 2005, VEDA, Bratislava, Slovakia, 2006, pp. 129–134, ISBN 80-969202-1-9.

[4] *Web Service Resource Framework*, The Globus Alliance, `http://www.globus.org/wsrf/`

[5] *GridSphere Portal Framework*, `http://www.gridsphere.org`

[6] *ALADIN Numerical Weather Prediction Project*, Meteo France, `http://www.cnrm.meteo.fr/aladin/`

[7] *MM5 Community Model*, National Center for Atmosperic Research, Pennsylvania State University, `http://www.mmm.ucar.edu/mm5/`

[8] *Hydrological Simulation Program—Fortran (HSPF)*, Water Resources of United States, U.S. Geological Survey, `http://water.usgs.gov/software/HSPF/`

[9] *Geographic Resources Analysis Support System GRASS*, `http://grass.itc.it`

[10] *Minnesota MapServer*, `http://mapserver.gis.umn.edu/`

[11] *OGR Simple Feature Library*, `http://www.gdal.org/ogr/`

[12] *GDAL—Geospatial Data Abstraction Library*, `http://www.gdal.org/`

[13] *ARC/INFO ASCIIGRID file format description*, `http://www.climatesource.com/format/arc_asciigrid.html`

[14] *ESRI ARC/INFO ASCII raster file format description (GRASS manual)*, `http://grass.itc.it/grass63/manuals/html63_user/r.in.arc.html`

[15] *GRASS ASCII raster file format description*, `http://grass.itc.it/grass63/manuals/html63_user/r.in.ascii.html`

[16] *ESRI Shapefile Technical Description, an ESRI white paper*, Environmental Systems Research Institute, Inc., 1998, `http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf`

Edited by: Dana Petcu, Marcin Paprzycki
Received: May 4, 2008
Accepted: May 27, 2008

Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 9, Number 2, pp. 101–109. http://www.scpe.org © 2008 SCPE

A FAULT TOLERANCE SOLUTION FOR SEQUENTIAL AND MPI APPLICATIONS ON THE GRID[*]

GABRIEL RODRÍGUEZ, XOÁN C. PARDO, MARÍA J. MARTÍN, PATRICIA GONZÁLEZ, AND DANIEL DÍAZ[†]

Abstract. The Grid community has made an important effort in developing middleware to provide different functionalities, such as resource discovery, resource management, job submission or execution monitoring. As part of this effort this paper addresses the design and implementation of an architecture (CPPC-G) based on services to manage the execution of fault tolerant applications on Grids. The CPPC (Controller/Precompiler for Portable Checkpointing) framework is used to insert checkpoint instrumentation into the code of sequential and MPI applications. Designed services will be in charge of submission and monitoring of the execution of CPPC-instrumented applications, management of checkpoint files generated by the fault-tolerant applications, and detection and automatic restart of failed executions.

Key words: fault tolerance, grid computing, Globus, MPI, checkpointing

1. Introduction. Parallel computing evolution towards cluster and Grid infrastructures has created new fault tolerance needs. As parallel platforms increase their number of resources, so does the failure rate of the global system. This is not a problem while the mean time to complete an application execution remains well under the mean time to failure (MTTF) of the underlying hardware, but that is not always true on long running applications, where users and programmers need a way to ensure that not all of the computation done is lost on machine failures.

Checkpointing has become a widely used technique to obtain fault tolerance on such environments. It periodically saves the computation state to stable storage, so that the application execution can be resumed by restoring such state. A number of solutions and techniques have been proposed [4], each having its own pros and cons. The advent of the Grid requires the evolution of checkpointers towards portable tools focusing on providing the following fundamental features:

(i) *OS-independence:* checkpointing strategies must be compatible with any given operating system. This means having at least a basic modular structure to allow for substitution of certain critical sections of code (e.g. filesystem access) depending on the underlying OS.

(ii) *Support for parallel applications with communication protocol independence:* the checkpointing framework should not make any assumption about the communication interface or implementation being used. Computational Grids include machines belonging to independent entities which cannot be forced to provide a certain version of the MPI interface. Even recognizing the role of MPI as the message-passing de-facto standard, the checkpointing technique cannot be theoretically tied to MPI in order to provide a truly portable, reusable approach.

(iii) *Reduced checkpoint file sizes:* the tool should optimize the amount of data being saved, avoiding dumping state which will not be necessary upon application restart. This improves performance, which depends heavily on state file sizes. It also enhances performance in case of process migration in computational Grids.

(iv) *Portable data recovery:* the state of an application can be seen as a structure containing different types of data. The checkpointing tool must be able to recover all these data in a portable way. This includes recovery of opaque state, such as MPI communicators, as well as of OS-dependent state, such as the file table or the execution stack.

CPPC (Controller/Precompiler for Portable Checkpointing) [3, 19, 20] provides all these features which are key issues for fault tolerance support on heterogeneous systems. CPPC[1] appears to the user as a runtime library containing checkpoint-supporting routines, together with a compiler tool which automates the use of the library. The CPPC Compiler automatically inserts checkpoint instrumentation, composed of CPPC Library calls and flow control code. The analyses and transformations performed by the compiler completely automate the instrumentation process.

[*]This work has been supported by the Spanish Ministry of Education and Science (TIN-2004-07797-C02, TIN2007-67537-C03-02 and FPU grant AP-2004-2695) and CYTED (Iberoamerican Programme of Science and Technology for Development) Project (ref. 506PI0293).

[†]Computer Architecture Group, University of A Coruña, SPAIN. (grodriguez@udc.es).

[1]CPPC is an open source project, and its current release can be downloaded at http://cppc.des.udc.es.

This paper introduces CPPC-G, a set of new Grid services[2] implemented on top of Globus 4 [5], which is able to manage the execution of CPPC-instrumented fault tolerant applications (CPPC applications from now on). The designed set of services will be in charge of submitting and monitoring the execution, as well as of managing and replicating the state files generated during the execution. It is widely accepted that the performance of an MPI application on the Grid remains a problem, caused by the communication bottleneck on wide area links. To overcome such performance problem, in this work it is assumed that all processes of an MPI application are executed on the same computing resource (e.g. a cluster or an MPP machine with MPI installed). Upon a failure, CPPC-G services will restart the application from the most recent consistent state, in a completely transparent way. At the present stage of CPPC-G development, only failures in the computing resource where the CPPC application is executed are being considered.

The structure of the paper is as follows. Section 2 gives an overview of the CPPC framework. Section 3 describes CPPC-G, its architecture, implementation details and deployment. The operation of the CPPC-G tool is shown in Section 4. Section 5 describes related work in the field. Finally, Section 6 concludes the paper.

2. The CPPC framework. Grid computing presents new constraints for checkpointing techniques. Its inherently heterogeneous nature makes it impossible to apply traditional state saving techniques, which use non portable strategies for recovering structures such as application stack, heap, or communication state. Therefore, modern checkpointing techniques need to provide strategies for *portable* state recovery, where the computation can be resumed on a wide range of machines, from binary incompatible architectures to computers using incompatible versions of software facilities, such as different implementations for communication interfaces.

CPPC (Controller/Precompiler for Portable Checkpointing) is a checkpointing tool focused on the insertion of fault tolerance into long-running message-passing applications. It is designed to allow for execution restart on different architectures and/or operating systems, also supporting checkpointing over heterogeneous systems, such as the Grid. It uses portable code and protocols, and generates portable checkpoint files while avoiding traditional solutions which add an unscalable overhead, such as process coordination or message-logging. This section details various aspects of CPPC design associated with these major issues.

2.1. Portability. A state file is said to be portable if it can be used to restart the computation on an architecture (or OS) different from that where the file was generated on. This means that state files should not contain hard machine-dependent state, which should be recovered at restart time using special protocols. The vast majority of checkpointing research has focused on systems implemented either inside the OS kernel or immediately above the OS interface. This kind of solutions generally become locked into the platform for which they were originally developed. For instance, when checkpointing parallel communication APIs, such as MPI, the typical approach has been to modify the existing implementations of such APIs. The problem arises when, for this approach to be practical, it becomes necessary to adopt the modified libraries in real systems, that use already highly tuned and optimized communication libraries. Other solutions store in the checkpoint file the outcome of the APIs functions, becoming dependent of its implementation.

The solution used in CPPC is to recover non-portable state by means of the re-execution of the code responsible for creating such opaque state in the original execution. Moreover, in CPPC the effective data writing will be performed by a user-selected writing plugin, each using its own format. This enables the restart on different architectures, as long as a portable dumping format is used. Currently, a writing plugin based on HDF5 [17] is provided.

2.2. Memory requirements. The solution of large real scientific problems may need the use of large computational resources, both in terms of CPU effort and memory requirements. Thus, many scientific applications are developed to be run on a large number of processors. The full checkpointing of this kind of applications will lead to a great amount of stored state, the cost being so high as to become impractical.

CPPC reduces the amount of data to be saved by including, in its compiler, a live variable analysis in order to identify those variable values that are only needed upon restart. Besides, a compressed format based on the ZLib library [6] is included. This does not only help saving disk space and network transfers (if needed), but also can improve performance when working with large datasets with high compression rates. A multithreaded dumping option is also provided to improve performance when working with large datasets. If a failure occurred

[2]With the term Grid service we denote a Web service that complies with the Web Services Resource Framework (WSRF) specifications.

in the checkpointing thread, inconsistent checkpoint files would be created. CPPC generates a CRC-32 for the checkpoint file. This CRC-32 is checked upon restart to ensure file correctness.

2.3. Global consistency. When checkpointing parallel applications, special considerations regarding message-passing have to be taken to ensure that the coordination implicitly established by the communication flow between processes is not lost when restarting the application. If a checkpoint is placed in the code between two matching communication statements, an inconsistency will occur when restarting the application, since the first one will not be executed. If it is a send statement, the message will not be resent and becomes an in-transit message. If it is a receive statement, the message will not be received, becoming a ghost message. Checkpoint consistency has been well-studied in the last decade [4]. Approaches to consistent recovery can be categorized into different protocols: uncoordinated, coordinated and message-logging. In uncoordinated checkpoint protocols the checkpoint of each process is executed independently of the other processes, leading to the so called domino effect (processes may be forced to rollback up to the beginning of the execution). Thus, these protocols are not used in practice. An important drawback, both of coordinated protocols and message-logging solutions, is their scalability. Increasing the number of processors will multiply the number of flying messages, thus enlarging the computation per process needed to be protocol-compliant.

CPPC avoids the overhead caused by coordination and message-logging by focusing on SPMD parallel applications and using a spatially coordinated approach. Checkpoints are taken at the same relative code locations by all processes, without performing interprocess communications or runtime synchronization. To avoid problems caused by messages between processes, checkpoints must be inserted at points where it is guaranteed that there are no in-transit, nor ghost messages. These points will be called safe points. Checkpoints generated in safe points are transitless and consistent, both being conditions for a checkpoint to be called strongly consistent [10]. Safe point identification and checkpoint insertion is automatically performed by the CPPC compiler.

3. CPPC-G design. In this section the design and implementation of a set of Grid services for the remote execution of CPPC applications is described. They have been implemented on top of Globus 4 using the Java API. The new Grid services must provide different functionalities such as resource discovery, remote execution and monitoring of applications, detection and restarting of failed executions, etc. This section discusses the most relevant design and implementation issues to achieve all these features.

3.1. System architecture. Figure 3.1 shows the proposed CPPC-G architecture that comprises a set of five new services that interact with Globus RFT [8] and WS-GRAM services [9]. A `FaultTolerantJob` service is invoked to start the fault-tolerant execution of a CPPC application. `CkptJob` services provide remote execution functionalities. Available computing resources hosting a `CkptJob` service are obtained from a `SimpleScheduler` service. `StateExport` services are responsible for tracking local checkpoint files periodically stored by the CPPC application. And finally, `CkptWarehouse` services maintain metadata and consistency information about stored checkpoint files. In the following, the functionality of each service is described in depth.

CPPC applications running in a computing resource can store checkpoint files in locations not accessible to services hosted in different computing resources (e.g. the local filesystem of a cluster node). The `StateExport` service is responsible for tracking these local checkpoint files and move them to a remote site that could be accessed by other services. There must be a `StateExport` service for every computing resource where a CPPC application could be executed. To help finding checkpoint files, the CPPC library has been slightly modified. Now processes of CPPC applications write, besides checkpoint files, metadata files in a previously agreed location in the computing resource filesystem. `StateExport` resources periodically check (by using GridFTP) for the existence of the next metadata file in the series. When a new one is found, the resource parses it, locates the newest checkpoint file using the extracted information, and replicates it via RFT in a previously agreed backup site. When the replication is finished a notification is sent to the `CkptJob` service.

The `CkptJob` service provides remote execution of CPPC applications. This service coordinates with `StateExport` and `CkptWarehouse` to extend WS-GRAM adding needed checkpointing functionality. Job descriptions used by the `CkptJob` service are those of WS-GRAM augmented with the End-Point Reference (EPR) of a `CkptWarehouse` service and a `StateExport` resource description element, that will be used as a template to create `StateExport` resources. The `CkptJob` service can be configured to register useful information with an MDS index service in the form of a sequence of tags. These tags are specified by the administrator in a configuration file and used to indicate particular properties (e.g. that MPI is installed in the computing resource).

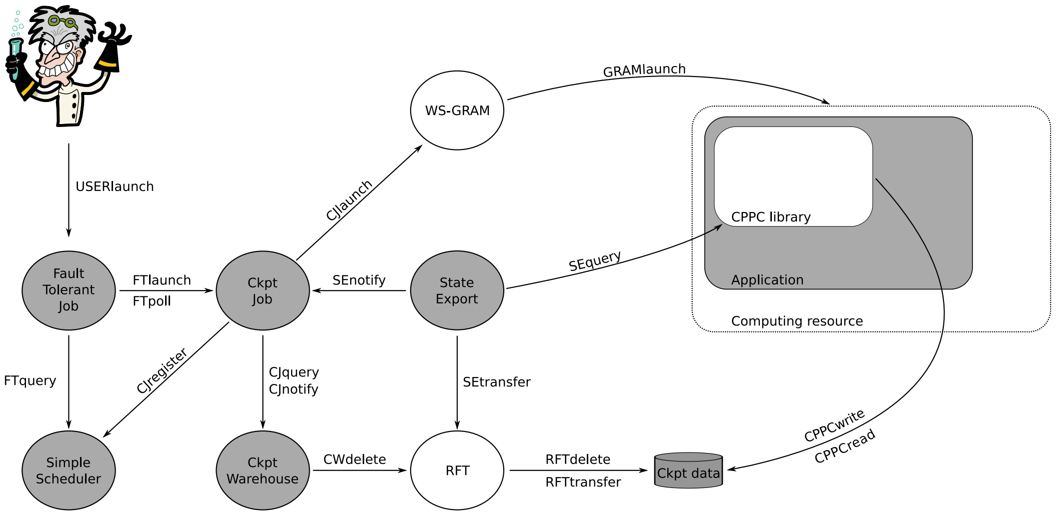


Fig. 3.1. *CPPC-G system architecture*

The `CkptWarehouse` service maintains metadata about checkpoint files generated by running CPPC applications. Each time a checkpoint file is successfully exported the proper resource in the `CkptWarehouse` service is notified, being responsible for composing sets of globally consistent checkpoint files. When a new globally consistent state is composed, checkpoint files belonging to the previous state (if they exist) become obsolete and can be discarded (they are deleted by using RFT).

The `SimpleScheduler` service keeps track of available computing resources hosting a `CkptJob` service. The service subscribes to an MDS index service that aggregates the information published by registered `CkptJob` services. In particular, the sequences of tags published by `CkptJob` services are used to select the proper computing resource that satisfies some given scheduling needs. As of now, the only supported scheduling need is the required presence of a given tag, but this mechanism could be used in future versions to support more complicated selections.

The `FaultTolerantJob` service is the one which the user invokes to start the execution of a CPPC application. One resource is created for each application, being responsible for starting and monitoring it. The monitoring of the execution is done by periodically polling the state of the computing resource. In case of failure, the execution is restarted automatically. In case the execution fails, `FaultTolerantJob` requests another node from the scheduler and tries to restart the application there from the last saved state. Computing resources needed for executing the application are obtained by querying a `SimpleScheduler` service, so it is not possible to know beforehand where the application will be executed. As a consequence, credential delegation has to be deferred until the precise `CkptJob` service to be invoked to execute the application is known.

3.2. Component deployment. Figure 3.2 shows the expected deployment of CPPC-G services in a Grid. As it was already mentioned, it is assumed that all processes of a CPPC application will be executed in the same computing resource for performance reasons. In a typical configuration of CPPC-G, a `CkptJob` service and a `StateExport` service will be present in that resource (as will be Globus WS-GRAM and RFT services). The `CkptWarehouse`, `SimpleScheduler` and `FaultTolerantJob` services will reside in other computing resources. It is enough that one instance of each of these last three services exists in a Grid for the system to work. It must be noted that the use of `SimpleScheduler` and `FaultTolerantJob` is optional. They must be present only if automatic restart of failed executions is wanted. The rest of services can be used on their own if only remote execution of CPPC applications is necessary. In this case it will be responsibility of the user to manually restart a failed execution.

Other configurations besides the one shown in the figure are possible. Although it is usual for a `CkptJob` service to invoke the `StateExport` service hosted in the same Globus container, it is not mandatory. The `StateExport` and `CkptJob` services could reside in different computing resources provided that GridFTP servers are present in both of them. In any case, the `StateExport` service must be configured to inspect the same computing resource where the `CkptJob` service submits the CPPC application, otherwise no checkpoint files

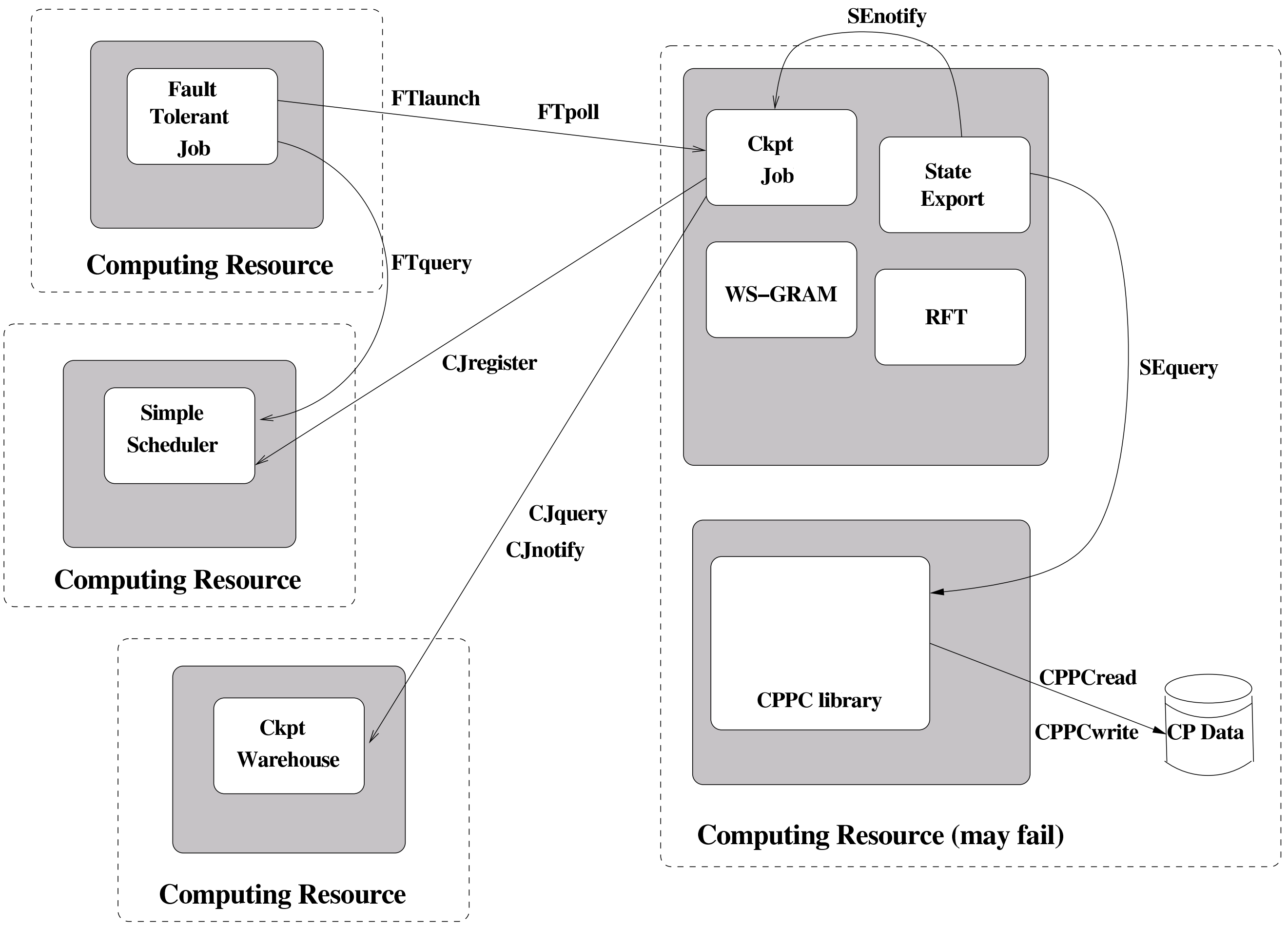


FIG. 3.2. *CPPC-G deployment*

will ever be found. It is usual for a `CkptJob` service to invoke WS-GRAM in the same Globus container, but it is also possible to host them in different computing resources. In a similar way, it is not mandatory for the `CkptWarehouse` service to reside in the same resource in which checkpoint files are stored. They can be remotely accessed in any resource containing RFT or GridFTP (or both). All these alternatives provide more flexibility to administrators when deploying the system.

3.3. Some implementation issues. In this section some general questions, that are common to the implementation of all the services, are discussed. They are related to management of long operations, security issues, chained invocations among services, resource creation requests and provided clients.

3.3.1. Managing long operations. Using a simple invocation of a Grid service to implement an operation that takes a long time to complete lacks flexibility, since there is no way to cancel or interrupt it. Instead, an approach based on the factory pattern is preferred. In this approach, widely used in the implementation of Globus, an operation is started by invoking a factory service that creates an instance resource in an instance service using a given factory resource as template. In this paper, the terms resource and service are used to refer to resource and service instances. The newly created resource is responsible for tracking the progress of the operation and it will be possible to query its state or subscribe in order to receive notifications when it changes. Furthermore, resources created in this way can be explicitly destroyed at any time or be automatically destroyed when a termination time specified by the user expires. It is responsibility of the user to extend the resource lifetime if it was not long enough to complete the operation. Resource lifetimes are a means to ensure that resources will be eventually released if communication with the user is lost.

3.3.2. Security issues. The developed services depend on the underlying GSI security infrastructure of Globus for authentication, authorization, and credential delegation. Credential delegation can be performed directly by the user, or automatically by a service that itself has access to a delegated credential. The standard Globus services follow the principle of always making the user delegate the credentials himself beforehand, never resorting to automatic delegation. This is more cumbersome for the user, but allows a greater control over where the credentials will end up. The developed CPPC-G services also try to follow that same principle whenever possible. However an exception is made in situations in which the service to be invoked is not known beforehand because it is dynamically selected. Automatic delegation has been used in these cases.

3.3.3. Chained service invocations. Most operations are implemented by invoking a service that itself invokes other services. It is usual to end up with several levels of chained invocations. With more than two levels, some difficulties arise with the delegation of credentials. In Globus, services that require user credentials publish the delegation factory service EPR as a resource property of their corresponding factory service. The user must use that EPR to delegate his credentials before being allowed to create resources in the service. Services that invoke other services that also require the delegation of credentials must publish one delegation factory service EPR for each invoked service. In the following, the term delegation set will be used to refer to the set of delegation factory service EPRs where the user must delegate his credentials before using a service. Once the user has delegated his credentials to the proper delegation services, delegated credential EPRs are passed to invoked services as part of resource creation requests, that will be explained later in this section. In the following, the term credential set will be used to refer to the set of delegated credential EPRs.

When there are a large number of services involved in a chained invocation, the use of delegation sets becomes complicated for users and administrators. From the user's point of view, the delegation set is a confusing array of delegation EPRs to which he must delegate his credentials before invoking a service. To help users, XML entities have been defined to be used in the service WSDL file to describe the delegation sets in a hierarchical fashion. Once the WSDL is processed and stubs are generated, helper classes can be defined to handle delegation automatically. From the administrator's point of view, all the EPRs in the delegation set of a service must be specified in its configuration files, which is an error-prone task. To avoid this problem, a technique based on queries to build delegation sets dynamically has been implemented.

3.3.4. Resource creation requests. In order to create a resource, factory services take as parameters the following creation request datatypes:

(i) *The initial termination time of the resource.*

(ii) *The resource description.* This is the main component. It may include additional resource descriptions associated with chained invocations (e.g. WS-GRAM job descriptions include RFT transfer descriptions for file staging).

(iii) *A credential set.* That is, the delegated credential EPR to be used by the resource, plus the delegated credential EPRs to be used in invocations to other services. This is separated from the resource description to potentially allow different requests to reuse the same credentials (e.g. repeated invocations to RFT with different source/destination URL pairs but with the same user credentials).

(iv) *A refresh specification.* For each of the services the resource is expected to invoke, a lifetime refresh period and a ping period are specified. The lifetime refresh period is the amount of seconds to extend the lifetime of resources in invoked services. The ping period is the frequency with which the resources in invoked services are checked to be up and reachable. If no service invocations are expected, no refresh specification is needed.

3.3.5. Provided clients. For each CPPC-G service two client programs are provided: a command-line client and a resource inspector. Command-line clients are used for the creation of resources from their descriptions. They prepare creation requests, contact the proper factory services and return the EPRs of the created resources. Resource inspectors are used for interacting with the created resources. They have a graphical interface to monitor resource notifications, query resource properties and invoke service operations.

4. CPPC-G operation. To initiate the execution of an application the following steps are taken in sequence. It is assumed that, before the user submits an application, all available `CkptJob` services are already registered with a `SimpleScheduler` service (`CJregister` in Figure 3.1):

1. In order to prepare the application submission, the user must create in advance an instance of the `CkptWarehouse` service and a credential set. This information will be included as part of the resource creation request used to submit the application.

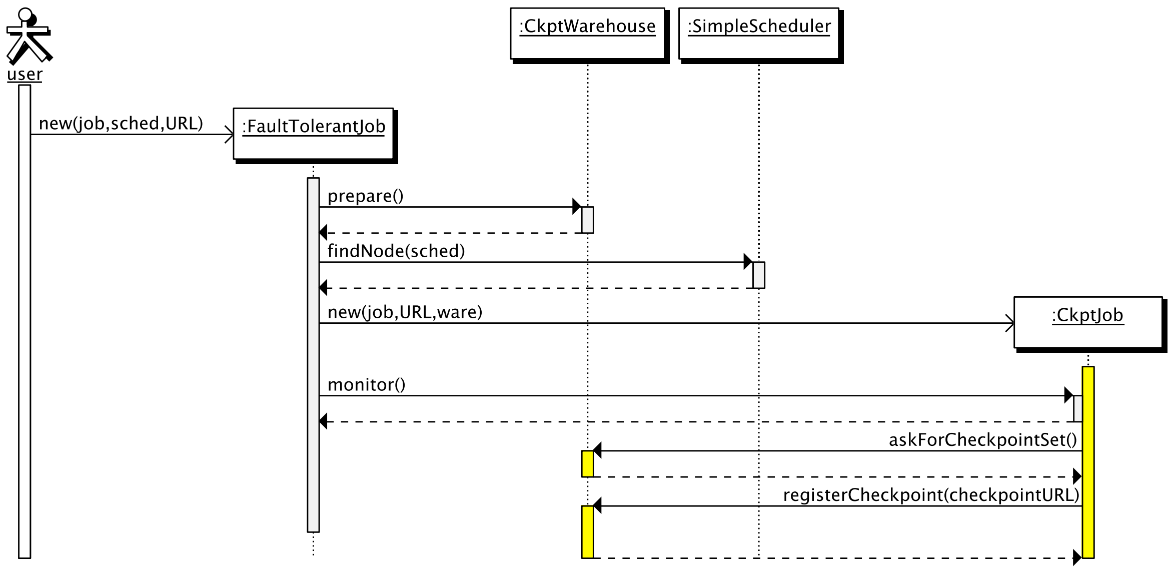


Fig. 4.1. *Sequence diagram for a fault-free execution*

2. The user submits the application to a `FaultTolerantJob` service (`USERlaunch` in Figure 3.1).
3. The `FaultTolerantJob` service invokes a `SimpleScheduler` service (`FTquery` in Figure 3.1), asking for an available computing resource.
4. The `FaultTolerantJob` service queries the `CkptJob` service on the selected computing resource to get its delegation set. The `CkptJob` service builds the delegation set dynamically by querying the services hosted in the computing resource (i. e. the `StateExport` service, WS-GRAM and RFT). With this delegation set and one of the delegated credentials of the user, the `FaultTolerantJob` service creates a credential set that will be included as part of the resource creation request used to start the CPPC execution.
5. The `FaultTolerantJob` service invokes the `CkptJob` service on the selected computing resource to start a CPPC execution (`FTlaunch` in Figure 3.1).
6. The `CkptJob` service queries the `CkptWarehouse` service to obtain the last consistent set of checkpoint files (`CJquery` in Figure 3.1). Checkpoint files will be moved, as part of the staging of input files, by WS-GRAM on the selected computing resource when the application was started.
7. The `CkptJob` service invokes WS-GRAM to initiate the application execution (`CJlaunch` in Figure 3.1).
8. The `CkptJob` service invokes also the `StateExport` service to initiate the exporting of checkpoints.

When a process of the CPPC application generates a checkpoint file (`CPPCwrite` in Figure 3.1) the following steps are taken in sequence:

9. The corresponding `StateExport` resource detects the presence of the newly created checkpoint file (`SEquery` in Figure 3.1) by using the technique based on monitoring metadata files already explained in Section 3.2.
10. The `StateExport` service uses RFT to export the checkpoint file to a previously agreed backup site (`SEtransfer` in Figure 3.1).
11. Once the transfer is finished, the `StateExport` service notifies the `CkptJob` service (`SEnotify` in Figure 3.1) about the existence of a new checkpoint file.
12. After receiving the notification, the `CkptJob` service notifies the `CkptWarehouse` service in its turn (`CJnotify` in Figure 3.1). The notification includes information about the process that generated the checkpoint file.
13. When, upon arrival of more recent checkpoint files, a new consistent state is composed, the `CkptWare-house` service deletes obsolete files by using RFT (`CWdelete` in Figure 3.1).

A simplified sequence diagram showing a typical fault-free execution is shown in Figure 4.1.

Currently two general types of execution failures are being considered: failures in the CPPC application execution, or failures in the `CkptJob` service. In both cases the `FaultTolerantJob` service is finally aware of the failed execution and a restart is initiated going back to step 3. The process ends when the execution terminates successfully. All resources are released when the user acknowledges the finished execution.

5. Related work. Important effort has been made in developing middleware to provide Grids with functionalities related to application execution. However, support for fault tolerant execution is either lacking or limited. WS-GRAM [9], the execution manager of the Globus Toolkit, handles file transfer before and after an application execution, but offers no handling of the checkpoint files generated while the execution is underway. Other fault tolerance-related functionalities are absent too.

GridWay [11, 12] is a Grid-level scheduler provided with the Globus Toolkit (interfacing with other Grid systems is also possible). It handles the search for nodes that are suitable to the needs of the user. It offers checkpointing support for jobs, and allows job migration. The checkpointing system of Gridway does not cover message-passing applications that perform distributed checkpointing, which requires global checkpoint consistency determination and garbage collection of obsolete checkpoint files.

Regarding checkpoint file storage, the usual solution in computational Grids consists in storing several replicas of files in dedicated servers managed by replica management systems [1, 18]. For opportunistic Grids, a middleware that provides reliable distributed data storage using the free disk space from shared Grid machines is presented in [2].

Several approaches for the implementation of fault tolerance in message-passing applications exist. MPICH-GF [21] is a checkpointing system based on MPICH-G2 [14], a Grid-enabled version of MPICH. It handles checkpointing, error detection, and process restart in a manner transparent to the user. But, since it is a particular implementation of MPICH, it can not be used with other message-passing frameworks. The checkpointing is performed at a data segment level, that is, it stores the entire application state, thus generating non-portable files. To achieve global consistency MPICH-GF uses process coordination, which is a non-scalable approach.

There have been a number of initiatives towards achieving fault tolerance on Grids. The Grid Checkpoint and Recovery (GridCPR) Working Group [7] of the Global Grid Forum was concerned with defining a user-level API and associated layer of services that will permit checkpointed jobs to be recovered and continued on the same or on remote Grid resources. A key feature of Grid Checkpoint Recovery service was recoverability of jobs among heterogeneous Grid resources.

The CoreGrid checkpointing work group of the CoreGrid Network of Excellence has proposed an alternative Grid checkpointing architecture [13, 15]. The difference with the GridCPR proposal is that the latter assumes that the checkpointing tool should be a part of the application, and would be tightly connected to various Grid services such as communication, storage, etc. In the CoreGrid proposal, checkpointers are system-level and external to the applications.

MIGOL [16] is a fault-tolerant and self-healing grid middleware for MPI applications built on top of the Globus Toolkit. MIGOL supports the migration of MPI applications by checkpointing and restarting the application on another site. However, as for now the current version of the middleware depends of locally stored checkpoints, which have to be accessible after an execution failure to enable auto-recovery. No checkpoint replication is performed. This means that if the machine goes down or becomes otherwise inaccessible, application execution must start from the beginning.

6. Conclusions and future work. Services for fault tolerance are essential in computational Grids. The aim of this work is to provide a set of new Grid services for remote execution of fault-tolerant parallel applications. CPPC is used to save the state of sequential and MPI processes in a portable manner. The new Grid services ask for the necessary resources; start and monitor the execution; make backup copies of the checkpoint files; detect failed executions; and restart the application. All this is done in a completely transparent fashion.

The proposed Grid services are loosely coupled, up to the point that it is not necessary for them to reside in the same Globus container. Distributing the functionality into a number of separate services improves both modularity and reusability. Also, it allows to easily replace current services by new ones with desirable features. For instance, other scheduler service can be used instead of `SimpleScheduler`. Also, the CPPC framework could be replaced by any other checkpoint framework provided that it generates the necessary metadata files.

The functionality of already existing Globus services is harnessed whenever possible: CPPC-G uses WS-GRAM as job manager and to monitor the applications; RFT to transfer the state files; GridFTP to detect new state files; MDS to discover available computing resources; and GIS for authentication, authorization and credential delegation. Additionally, the modifications made to the existing CPPC library have been kept to a minimum.

At the moment, the CPPC-G architecture is not itself fault-tolerant. In the future it is planned to use replication techniques for the `FaultTolerantJob`, `SimplerScheduler` and `CkptWarehouse` services. Other future direction will be to automate the finding of potential checkpoint backup repositories over the Grid by querying a MDS index service.

REFERENCES

[1] A. L. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman, and R. Schwartzkopf, *Performance and Scalability of a Replica Location Service*, in HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA, 2004, IEEE Computer Society, pp. 182–191.

[2] R. Y. de Camargo, F. Kon, and R. Cerqueira, *Strategies for Checkpoint Storage on Opportunistic Grids*, IEEE Distributed Systems Online, 7 (2006), p. 1.

[3] D. Díaz, X. Pardo, M. J. Martín, P. González, and G. Rodríguez, *CPPC-G: Fault-Tolerant Parallel Applications on the Grid*, in 3rd Workshop on Large Scale Computations on Grids (LaSCoG'07), vol. 4967 of Lecture Notes in Computer Science, Springer, 2008.

[4] E. N. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson, *A Survey of Rollback-Recovery Protocols in Message-Passing Systems*, ACM Computing Surveys, 34 (2002), pp. 375–408.

[5] I. Foster, *Globus Toolkit Version 4: Software for Service-Oriented Systems*, Journal of Computer Science and Technology, 21 (2006), pp. 513–520.

[6] J. Gailly and M. Adler, *ZLib Home Page*. `http://www.gzip.org/zlib/`.

[7] Global Grid Forum, *Grid Checkpoint Recovery Working Group*. `http://forge.ogf.org/sf/projects/gridcpr-wg`

[8] Globus Alliance, *RFT: Reliable File Transfer Service*. `http://globus.org/toolkit/docs/4.0/data/rft/`

[9] ———, *WS-GRAM Execution Management Service*. `http://globus.org/toolkit/docs/4.0/execution/wsgram/`

[10] J. Hélary, R. Netzer, and M. Raynal, *Consistency Issues in Distributed Checkpoints*, IEEE Transactions on Software Engineering, 25 (1999), pp. 274–281.

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

[12] ———, *A Modular Meta-Scheduling Architecture for Interfacing with pre-WS and WS Grid Resource Management Services*, Future Generation Computing Systems, 23 (2007), pp. 252–261.

[13] G. Jankowski, R. Januszewski, R. Mikolajczak, and J. Kovacs, *Grid Checkpointing Architecture—a Revised Proposal*, Tech. Report TR-0036, Institute on Grid Information, Resource and Workflow Monitoring Systems, CoreGRID—Network of Excellence, May 2006.

[14] N. T. Karonis, B. Toonen, and I. Foster, *MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface*, Journal of Parallel and Distributed Computing, 63 (2003), pp. 551–563.

[15] J. Kovacs, R. Mikolajczak, R. Januszewski, and G. Jankowski, *Application and Middleware Transparent Checkpointing with TCKPT on Clustergrid*, in Distributed and Parallel Systems—Cluster and Grid Computing, Proceedings of 6th Austrian-Hungarian Workshop on Distributed And Parallel Systems (DAPSYS), P. Kacsuk, T. Fahringer, and Z. Németh, eds., Springer Verlag, 2007, pp. 179–189.

[16] A. Luckow and B. Schnor, *Migol: A Fault-Tolerant Service Framework for MPI Applications in the Grid*, Future Generation Computer Systems—The International Journal of Grid Computing: Theory, Methods and Applications, 24 (2008), pp. 142–152.

[17] National Center for Supercomputing Applications, *HDF-5: File Format Specification*. `http://hdf.ncsa.uiuc.edu/HDF5/doc/`.

[18] M. Ripeanu and I. Foster, *A Decentralized, Adaptive Replica Location Mechanism*, in HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA, 2002, IEEE Computer Society, p. 24.

[19] G. Rodríguez, P. González, M. J. Martín, and J. Touriño, *Enhancing Fault-Tolerance of Large-Scale MPI Scientific Applications*, in PaCT, V. E. Malyshkin, ed., vol. 4671 of Lecture Notes in Computer Science, Springer, 2007, pp. 153–161.

[20] G. Rodríguez, M. J. Martín, P. González, and J. Tourio, *Controller/Precompiler for Portable Checkpointing*, IEICE Transactions on Information and Systems, E89-D (2006), pp. 408–417.

[21] N. Woo, H. Jung, H. Y. Yeom, T. Park, and H. Park, *MPICH-GF : Transparent Checkpointing and Rollback-Recovery for Grid-enabled MPI Processes*, IEICE transactions on information and systems, 87 (2004), pp. 1820–1828.

Edited by: Dana Petcu, Marcin Paprzycki
Received: April 30, 2008
Accepted: May 16, 2008

Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 111–125. http://www.scpe.org

ISSN 1895-1767
© 2008 SCPE

SERVICE-ORIENTED SYMBOLIC COMPUTING WITH SYMGRID

DANA PETCU, ALEXANDRU CÂRSTEA, GEORGIANA MACARIU, AND MARC FRÎNCU*

Abstract. Recent software engineering concepts, like software as a service, allow the extension of the legacy code lifetime and the reduction of software maintenance costs. In particular, exposing computer algebra systems as services allows not only the integration in complex service-oriented architectures but also their further development. While existing standards may be used for service deployment, discovery and interaction, the particularities of services to be built require specialized solutions. A recent technical approach aimed at integrating legacy computer algebra systems into modern service-oriented architectures is presented and discussed in detail in this paper. A special emphasis is put on the ability to compose symbolic services in complex computing scenarios. A short description of how such systems were extended to allow the access of external services is provided as well. The proposed approach was implemented into a specific framework, namely, SymGrid-Services. Simple examples are provided to demonstrate usefulness of the framework.

Key words: service-oriented architecture, symbolic computing, wrappers for legacy software, service composition, computer algebra systems.

1. Introduction.

Symbolic computation is one of the most high demanding fields in terms of computer power as well as memory requirements. The current tools for symbolic computations are the Computer Algebra Systems (CAS). Standard CASs are designed to be used in the isolated context of the local machine on which they are installed. Their ability to interact with the external world is mostly restricted to a command line user interface, to the file I/O based operations and only occasionally the ability to interact using the TCP/IP sockets. The issues related to the data exchange between CASs were partially solved in the last decade by the introduction of OpenMath [24] and MathML [36] standards (XML-based data description format, designed specifically to represent computational mathematical objects). However, even in this case the inter-operability between the CASs is still an open problem.

Recent solutions to the CAS inter-operability problem through service-oriented approaches where proposed by an on-going international collaborative project, SCIEnce. This paper presents an overview of the achievements of the SCIEnce project relative to a particular middleware component that was developed in the last one and a half year, the SymGrid-Services. While this paper is an extended version of the [5] and focuses mostly on the wrapper services for CASs, it includes also a short descriptions of the other components that allow the composition of the wrapper services, as well as the seamless access of services within CASs, (these components are detailed in [3, 4, 6, 18]).

Overall, the paper is organized as follows. Section 2 discusses shortly the related work. Section 3 presents the SymGrid and its main components. Section 4 points to the solution proposed for service access from inside a CAS. Section 5 goes deep inside the solution adopted to present CASs as services. Section 6 gives some hints about the solution adopted for service compositions. Finally, Section 7 draws the conclusions and highlights the future steps.

2. Related work on mathematical Web and Grid services.

The problem of integrating legacy software into modern distributed systems has two obvious solutions: reengineering of the legacy software (an invasive procedure) and creation of wrapper components (an non-invasive procedure).

Due to the complexity of the computer algebra systems, a non-invasive procedure is more appropriate, as stated in [8, 34]. Several efforts have revealed the paths to be followed in wrapping CAS. We present in what follows the most relevant ones. A more detailed overview of the bellow described initiatives can be found in [26].

2.1. General wrapping solutions.

Exposing functionality of legacy components can be done using general wrapper tools. In this category we can include Soaplab [30] and JACAW [13].

More specifically geared toward mathematical problems is JavaMath [33], a Java API for connecting to mathematical software systems (a request is made to a broker for a particular software system and the broker establishes a session to such system). OpenMath encoding of objects can be used. An specific abstract interface for service access is given, but there is little abstraction from service implementation.

*Institute e-Austria Timişoara, România, B-dul Vasile Pârvan 4, 300223 Timişoara, Romania (science@ieat.ro).

2.2. Web services. Result of an academic activity, the MathWeb-SB [41] is a software bus that allows to combine mathematical services like computer algebra systems: a broker provides access object for service by name, ensuring the abstraction from service locations and from object encodings, but there is no way to interact with unknown services.

Internet-Accessible Mathematical Computation (IAMC) site [14] maintains a list of projects and systems related to mathematics that are accessible on the Internet; in the frame of the IAMC project a HTTP-like protocol was developed for server-client communication—a non-standard informal description of service and an abstract protocol for service access (machine-readable) were provided, but it requires insight to be used (not machine-understandable).

A general framework for the description and provision of Web-based mathematical services was designed within the MONET [22], aiming at demonstrating the applicability of the semantic Web to the world of mathematical software. It allows dynamic discovery of services based on published descriptions which express both their mathematical and non-mathematical attributes. A symbolic solver wrapper was designed to provide an environment that encapsulates CASs and exposes their functionalities through symbolic services. Maple was chosen as computational engine in the initial implementation and it is loaded from the XML configuration file [31]. Simple examples of mathematical Web services were provided: integration and differentiation services, limits and series services, root-finding and solving systems of polynomials. Axiom was used to demonstrate the ability to incorporate different computational engines without changes.

Mathematical Services Description Language (MSDL [2]) was introduced to describe mathematical Web services so that these services can be discovered by clients. It implements a service description model that uses a decomposition of descriptors into multiple inter-linked entities: problem, algorithm, implementation, and realization. More recently, a MathBroker [28] implementation was based on a Web registry to publish and discover mathematical Web services. A usage example of the MathBroker was provided in [2].

MapleNET and WebMathematica are commercial counterparts to these initiatives. In the MapleNET [20] a server manages concurrent Maple instances launched to serve client requests for mathematical computations and display services, and facilitates additional services such as user authentication, logging information, and database access. Similarly, WebMathematica [38] offers access to Mathematica applications through a web browser.

2.3. Grid services. GridSolve [42], a component of one of the earliest Grid systems developed, the NetSolve, is a middleware between desktop systems equipped with simple APIs and the existing services supported by the Grid architecture—this API is available for the Mathematica.

The GENSS project [10] followed the ideas formulated in the MONET project. Its aim was to combine Grid computing and mathematical Web services using a common open agent-based framework. The research was focused on matchmaking techniques for advertisement and discovery of mathematical services, and design and implementation of an ontology for symbolic problems.

Another academic initiative, MathGridLink [35] proposed both the development and deployment of Mathematica computational services on the Grid and the usage of existing Grid services from within Mathematica; this initiative is continued by the SCORUM project [23].

Separately, Maple2g (Maple-to-Grid) described in [25] allows the connection between Maple and computational Grids based on the Globus Toolkit. The prototype consists of two parts: a Maple-dependent one, a library of new functions allowing to interact with the Grid, and a Globus-dependent part, a package of Java CoG classes. Maple2g allows the access to Grid services, the exposure of Maple facilities as Grid services, and the cooperation between Maple kernels over the Grid. SymGrid-Services generalizes the Maple2g development experiences to the level of CAS.

GridMathematica [37] was constructed as a commercial solution for dedicated clusters facilitating parallel computing within Mathematica. Another example of exposing CAS to the Grid is HPC-Grid for Maple [12]. It is a distributed computing package using Maple that allows users to distribute computations across the nodes of a network of workstations; it offers a message passing API as well as a set of high-level parallelization commands. Based on MapleNet and HPC-Grid, the recent Grid Computing Toolbox for Maple [21] allows to distribute computations across nodes of a network of workstations, a supercomputer or across the CPUs of a multiprocessor machine (in the same administrative domain), and offers an MPI-like message passing API as well as a set of high-level parallelization commands.

The recent GEMLCA [7] is a solution to deploy a legacy code application (including a computer algebra system) as a Grid service without modifying the code. The front-end, described in the WSDL, offers Grid services to deploy, query, submit, check the status of, and get the results back from computational jobs. In order to access a legacy code program, the user executes the Grid service client that creates a code instance with the help of a code factory, and the system submits the job.

2.4. Overview. Overall, it can be observed that using the above mentioned technical solutions, several CASs can be remotely accessible, but, almost all do not use a standard data model for interactions with CASs. Moreover, none of the above shortly described systems conforms to all three of the following basic requirements (as the SymGrid-services does):

(a) deploy symbolic services;
(b) access available services from within the symbolic computing system;
(c) couple different symbolic services into a coherent whole.

Furthermore, the pre-WRSF versions of Web and Grid middleware were used in the previous described projects, making the service discovery a difficult task.

3. SymGrid. The aim of the SCIEnce project (Symbolic Computation Infrastructure for Europe, `http://www.symbolic-computation.org`), funded in the frame of the European Commission Programme FP6, is to improve integration between CAS developers and application experts. The project includes developers from four major CASs: GAP [9], Maple [19], MuPAD [29] and KANT [15]. Its main objectives are to:

– develop versions of the CASs that can inter-communicate via a common standard service interface, based on domain-specific results produced by the OpenMath [24] and the MONET [22] projects as well as generic standards for Web and Grid services, such as the WSRF;
– develop common standards and middleware to allow production of Web or Grid-enabled symbolic computation systems;
– promote and ensure uptake of recent developments in programming languages, including automatic memory management, into a symbolic computation systems.

The research is primarily concerned with parallel, distributed, Web and Grid-based symbolic computations. The five year workplan includes the followings stages:

1. produce a portable framework that will allow symbolic computations to access Grid services, and allow symbolic components to be exploited as part of larger Grid service applications on a computational Grid (finalized stage);
2. develop resource brokers that will support the irregular workload and computation structures that are frequently found in symbolic computations (on-going stage);
3. implement a series of applications that will demonstrate the capabilities and limitations of Grid computing for symbolic computations (future stage).

In what follows we describe the portable framework, namely the SymGrid. It was designed and presented first in [11]. SymGrid allows multiple invocations of symbolic computing applications to interact via the Web or Grid and it is designed to support the specific needs of symbolic computations.

SymGrid comprises of two components: the SymGrid-Par to support the construction of high-performance applications on computational Grids, and the SymGrid-Services to manage Web and Grid services.

The SymGrid-Par middleware is used to orchestrate computational algebra components into a parallel application and allows symbolic computations to be executed as high-performance parallel computations on a computational Grid. SymGrid-Par components communicate using the Symbolic Computation Software Composability Protocol (developed in the frame of SCIEnce project), SCSCP [17], which in turn builds on OpenMath. SymGrid-Par provides an API for parallel heterogeneous symbolic components, which extends the Grid-GUM [39], and comprises in two generic interfaces:

CAG interface: Computational Algebra system to Grid middleware, that links CASs to the Grid-GUM;
GCA interface: Grid middleware to Computational Algebra system, that links the Grid-GUM to these systems.

The purpose of the CAG/GCA interfaces is to enable computational algebra systems to execute on computational Grids, e.g. on a loosely-coupled collection of Grid-enabled clusters. Details about SymGrid-Par are provided in [40]. Here, a GAP library has been build as demonstrator of usage of the SymGrid-Par.

The SymGrid-Services middleware is used to access, from computational algebra systems, Grid and Web services, and to access and compose the CASs deployed as Grid and Web services. It is based on the WSRF standard that ensures uniform access to Grid and Web services. A GAP library is available also for this SymGrid component. As in the case of the SymGrid-Par, the SymGrid-Services has two interfaces that will be detailed in the next sections:

CAGS interface: Computational Algebra system to the Grid and Web services, links CASs to external services;

GCAS interface: Web and Grid services for the Computational Algebra system, integrating these systems into a service-oriented architecture and allowing service composition.

While there are several parallel computer algebra systems suitable for either shared-memory or distributed memory parallel systems, work on Grid-based symbolic systems is still nascent. A review of current Grid-based systems for symbolic computation can be found in [26] and, in a short version, in the next section.

The main novelty of the SymGrid-Services consists of the fact that it is the only current middleware package that allows generic access to both Web and Grid symbolic and non-symbolic computing services, as well as their composition. The specific mathematical Web services like the ones defined by the MONET project [22], or standard Web services, are easily accessible, and the Grid services wrapping Kant, MuPAD, GAP and other computational algebra systems provided by the SymGrid service container can be called from inside a CAS.

A number of major new obstacles need to be overcome by the SymGrid in the near future. Amongst the most important future developments are mechanisms for adapting to dynamic changes in either computations or systems. This is especially important for symbolic computations, which may be highly irregular in terms of data structures and general computational demands, and which therefore present an interesting challenge to current and projected technologies for computational Grids in terms of their requirements for autonomic control.

SymGrid intends to go beyond current systems by developing a generic framework supporting heterogeneous Grid components derived from a critical set of complementary symbolic computing systems, by incorporating new and vital tools such as dynamic resource brokers and schedulers that can work both at a task and system level, and by creating a number of large new demonstrator applications.

4. CAGS interface of SymGrid-Services.

CAGS allows Computer Algebra Systems to leverage the computing capabilities offered by external Grid or Web services. Details about its implementation are provided in [3]. In this paper we present shortly its main functionality.

4.1. Description of the interface's implementation.

A CAS user must be able to: discover services, connect to remote services, call remote operations, run jobs, and transfer files in a seamless fashion. CAGS provides this functionality. The SymGrid-Services's CAGS interface consists of three Java classes (Figure 4.1):

SGServices provides three kinds of operations—retrieval of a list of services registered in a certain Web or Grid services registry; retrieval of signatures for the exposed operations of a service; and calling remote operations.

SGProxyCert handles issues arising from the need to support 'single sign-on' for users of the Grid and delegation of credentials: namely the creation and destruction of proxy certificates, retrieval of information about the owner of a certificate and about the lifetime of a proxy certificate.

SGUtils provides additional functionality for explicit file transfer, file deletion and remote job execution.

To access the functionality provided by these three classes it is necessary to create new class instances. Generally, however, CASs do not offer such functionality by default and therefore it is necessary to run the supplied Java classes in a child process created by the CAS. This process then communicates with the CAS using standard input/output streams to pass parameters and return values (Figure 4.2). The CAS will create a child processes by launching a script that starts a Java class called RunManager. The main method of this class should be called with an array of string type arguments:

$$\texttt{java RunManager}\ arg0\ arg1\ arg2\ \ldots\ argN$$

where the *arg0* represents the Java class to load, *arg1* is the name of the method to invoke, and remaining arguments are passed to the method. RunManager is a generic command that exploits Java reflection capabilities to allow the execution of any class.

4.2. Usage scenario.

The primary functionality of CAGS lies in obtaining a list of Grid or Web services registered at a certain URL; obtaining the signatures of those operations that are exposed by a certain Grid or Web service; calling an operation and retrieving the result of an operation call. Secondary functionality includes file transfer and job submission, and managing utilities for proxy certificates.

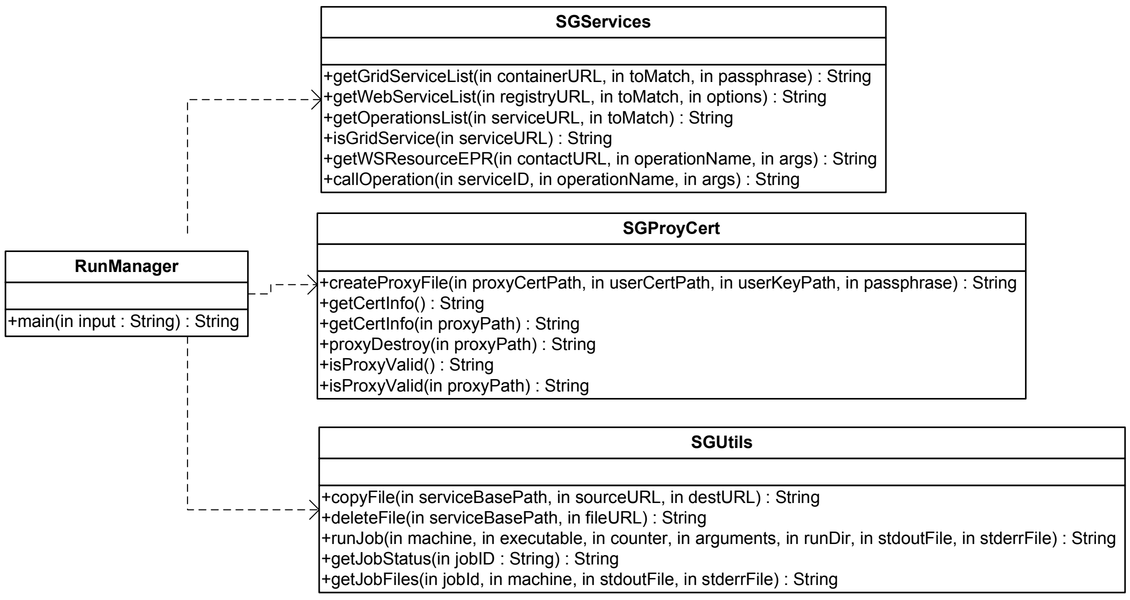

SGServices
+getGridServiceList(in containerURL, in toMatch, in passphrase) : String
+getWebServiceList(in registryURL, in toMatch, in options) : String
+getOperationsList(in serviceURL, in toMatch) : String
+isGridService(in serviceURL) : String
+getWSResourceEPR(in contactURL, in operationName, in args) : String
+callOperation(in serviceID, in operationName, in args) : String
SGProyCert
+createProxyFile(in proxyCertPath, in userCertPath, in userKeyPath, in passphrase) : String
+getCertInfo() : String
+getCertInfo(in proxyPath) : String
+proxyDestroy(in proxyPath) : String
+isProxyValid() : String
+isProxyValid(in proxyPath) : String
RunManager
+main(in input : String) : String
SGUtils
+copyFile(in serviceBasePath, in sourceURL, in destURL) : String
+deleteFile(in serviceBasePath, in fileURL) : String
+runJob(in machine, in executable, in counter, in arguments, in runDir, in stdoutFile, in stderrFile) : String
+getJobStatus(in jobID : String) : String
+getJobFiles(in jobId, in machine, in stdoutFile, in stderrFile) : String

Fig. 4.1. *CAGS structure*

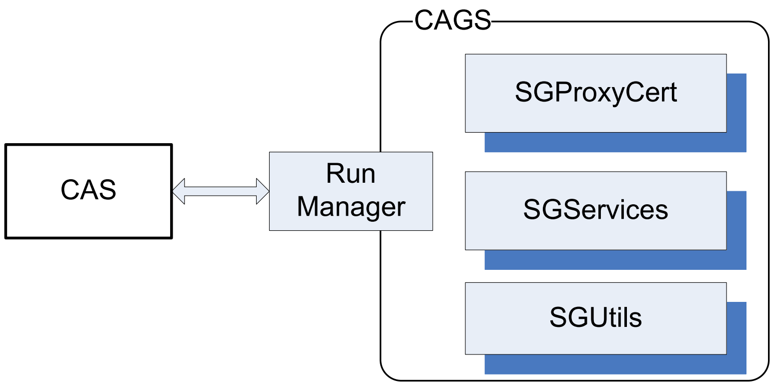


Fig. 4.2. *Interactions CAS—CAGS*

A typical scenario begins with the discovery of a service by consulting a service registry URL:

```
start scenario(registry_URL)
 if(is_Web_service_registry(registry_URL))
  service_list:=get_Web_service_list(registry_URL,toMatch,options)
 else
  service_list:=get_Grid_service_list(registry_URL,toMatch)
 endif
 service:=select_service(service_list)
 operation_list:=get_operation_list(service,toMatch)
 operation:=select_operation(operation_list)
 [create_proxy_certificate();]
 result:=call_operation(service,operation,parameters)
end scenario
```

Here, the *registry_URL* parameter is a valid URL of a UDDI registry or a Globus container. The *toMatch* parameter is a selection string that must be a substring of the service name in the *get_Web_service_list*/*get_Grid_service_list* combined with a substring of the operation name in the *get_operation_list*. The selection functions *select_service*/*select_operation* are user-defined functions that can be used to select the desired service/operation.

Note that this scenario assumes that the user only knows the *registry_URL*. If the user already knows, for instance, the service URL and the signature of the operation, the unnecessary steps can be omitted.

One of the initial SymGrid-Services targets is the GAP [9] computational algebra system. A demonstrator library of GAP functions was built to allow a GAP user to access the CAGS functionality without needing to know the details of the implementation. Here we ahev developed a single function in the GAP library for each method in the CAGS interface. The general pattern used to wrap the CAGS functionality is:

```
directoryName := DirectoryCurrent();
fileName:=Filename(directoryName,"script.sh");
ListMethods := function(arg2\dots argN)
  local jm, response;
  jm:= InputOutputLocalProcess(DirectoryCurrent(),fileName,[]);
  #send handler method: class, method, no. args, args, end signal
  WriteLine(jm, "java_class_to_call"); WriteLine(jm, "method_name");
  WriteLine(jm, "nr_of_parameters");
  WriteLine(jm, arg2);  \dots   WriteLine(jm, argN);
  WriteLine(jm, "quit");
  #retrieve response from the process
  repeat
    response := ReadLine(jm);
    if (response <> fail) Print(response); fi;
  until (response = fail);
end;
```

Since Java cannot be invoked directly from a GAP program, the solution is to invoke instead a shell script that starts a Java program using the *InputOutputLocalProcess* function of the GAP. The needed parameters are passed to the script *script.sh*. The variable *jm* is the handle to the running process. It can be used as an argument to the *WriteLine* function to pass the arguments. Arguments provided to the shell script will be mapped to the parameters required to call the RunManager.

4.3. Usage examples. We illustrate below how the CAGS can be used from the command line by invoking several methods in the public interface of the tool with the help of the RunManager utility.

Three categories of services have been used for the initial tests:

1. general Web services (see the paper [3]);
2. domain-specific symbolic Web services such as these provided by MONET [22] and GENSS [10];
3. simple test Grid services, that we have deployed on a single cluster and wraped publicly available CASs: CoCoA, Fermat, GAP, Kant, Macaulay, MuPAD, PARI, Singular and Yacas. These test services were deployed in a Globus container (available, for example at http://matrix.grid.info.uvt.ro:8082/).

A registry can be interrogated to obtain the list of services registered to that registry. For example, the command:

```
java science.run.RunManager science.clients.wrappers.SGServices
getWebServiceList "http://matrix.grid.info.uvt.ro:8082/wsrf/services/"
"Service" "caseSensitiveMatch"
```

produces the result:

```
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/CoCoAService
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/FermatService
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/GAPService
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/KANTService
...
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/YACASService
```

This is a list of service URLs representing all services in the registry whose names include the substring "Service".

Once the address of a service is known, CAGS can supply the signatures of the operations exposed by the service. Based on the list of the methods exposed, the user can then discover all details that are needed to call a remote operation. In the case of the service wrapping Fermat that is deployed on the SymGrid testbed, as the result of the command:

```
java science.run.RunManager science.clients.wrappers.SGServices
getOperationsList "http://matrix.grid.info.uvt.ro:8082/wsrf/services/
science/FermatService" ""
```

the following list of operations can be obtained:

```
    string Bin (string)
    string Prime (string)
    ...
```

Remote operation invocation is one of the main reasons for using CAGS. To call the Fermat service operation that calculates the greatest common divisor of 96 and 80, one should use the following:

```
java science.run.RunManager
science.clients.wrappers.SGServices callOperation
http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/
FermatService GCD "96,80"
```

obtaining as result *string:16*. Note that the GCAS version of the services described in the next section accepts only two OpenMath objects instead of the two integer values currently allowed by the service used in this example.

To show how GAP can use CAGS to interact with external services, we have built an example in which GAP calls an external service. The external service is a YACAS instance, that easily interacts with OpenMath objects. The first step in the example is to list all the services from a Globus container that can be matched using the string "YACAS". From the list of services that are obtained, we choose the wrapping service, and ask for the list of operations supported by that service that relate to OpenMath (OM). The final step of the example launches a call from GAP to the YACAS service. The result of the call is displayed by GAP on the console:

```
gap> SG_CreateProxy("path_proxy","","","pswd");
gap> gridServList := SG_GridServiceList(
"http://matrix.grid.info.uvt.ro:8082/wsrf/services/","YACAS");
 http://matrix.grid.info.uvt.ro:8082/wsrf/services/science/
 YACASService
gap> operationList := SG_OperationList(gridServiceList[1], "OM");
 string YACAS_OMDef (string)
 string YACAS_OMForm (string)
 string YACAS_OMRead (string)
gap> SG_CallOperation(gridServiceList[1],operationList[2], "25");
 string:<OMOBJ> <OMI>25</OMI> </OMOBJ>
```

5. GCAS interface of SymGrid-Services.

This section introduces the CAS Server architecture as the main component of the GCAS interface of the SymGrid-Services. The CAS Server was presented first in the paper [5], and the description is extended by this paper.

The main functionality of the CAS Server is to enable virtually any CAS to have its own functions remotely accessible. Several CASs can be exposed through the same CAS Server at the same time. Additionally to the services that where exposed in the testing phase of the CAGS, the CAS Server allows the limitation of the number of functions exposed for a CAS and the interrogation of the CAS Server about the functions exposed. The following subsection is an overview of the CAS Server architecture.

Let us stress that integration of legacy software in service oriented architectures must consider three major issues: data encoding, exposing technology, and wrapping the legacy system. These issues are treated in different subsections that follow. Implementation issues are provided at the end of the section.

5.1. Architecture of CAS Server.

GCAS aims to expose CASs functionality as services. The interaction between these services can lead to computing efficiency gains when solving complex symbolic problems. Unfortunately, the different data encoding and algorithms used within distinct CASs to solve the same problem hinders the integration process.

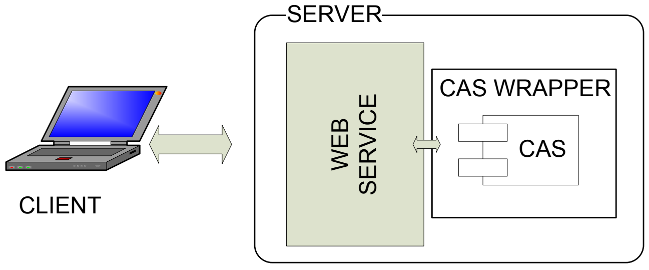


Fig. 5.1. *CAS-wrapper service architecture.*

We assume that it is the client's responsibility to choose the right functions needed for a computation, just as she would have had the CAS installed on the local machine. Alternatives like predefined workflows or patterns for service composition are subject of later developments of the SymGrid-Services' GCAS component. Patterns description were described first in [11] and were recently implemented in the composer component of the GCAS described in [6] and shortly in Section 6.

The first and most important issue to deal with in the component design is to find means to expose CAS functions in order to make them available through service invocation. Other issues that must be taken care of are related to user assistance tools and security.

A significant number of existing CASs is not conceived to be remotely accessible. Most of them allow interaction with the external world through line commands. Only few of them have special facilities like socket connections or multi-threading. In general, the redesign of these systems to allow modern communication with outside world cannot be achieved easily. Due to these reasons, we have considered the wrapper approach for integrating CASs.

Wrapping legacy software and exposing their functionality using service technologies involves the creation of a three level architecture at the server side, as shown in Figure 5.1. The server has the role of receiving the calls from the client, resolving them using underlying legacy software and returning the result(s) of the computation to the client.

The proposed architecture intends to expose functions implemented by several CASs in a secure manner. To achieve this goal the simple wrapper architecture is augmented with several features. Setting up the architecture on a server machine requires that all necessary software tools and components are available on the machine, namely the service tools and the CASs. A simple tool should allow the administrator of the machine to register into a CAS registry the CAS functions that he wants to expose. Every function exposed by a CAS will have an entry in the registry. Thus, a method that does not appear in this registry is considered inaccessible to remote invocations (for example the system function available in different CASs).

The remote clients of the system may interrogate the system in order to find out the available functions and their signatures. The signatures of the functions should not differ in any way from the original signature of the function as provided by a CAS. The remote invocation of a function should be reduced to the invocation of a remote

execute(CAS_ID,call_object)

where the *CAS_ID* is the CAS unique identifier and the *call_object* represents an OpenMath object, as described in the following subsection.

Additionally, several other related operations should be available: find the list of the CASs that are installed on the CAS Server machine or find the list of the available functions that were exposed.

5.2. Parameter encoding level. One important aspect of interactions between the client and the legacy system is the model of data exchange. The data model used by the client must be mapped to the internal data model of the legacy software. Currently, the available CASs use a variety of encoding standards, from plain text to XML structured documents. Due to the benefits involved in representation and manipulation of the data in a structured manner, the XML standard was adopted as the natural choice for machine to machine interaction. Representation of mathematical objects was achieved by MathML and OpenMath standards. However, while the former is well suited for representing mathematical objects in Web browsers, the latter is more appropriate for describing mathematical objects with semantic context.

Efforts to enable parsing of OpenMath objects are under way for several CASs. A standard encoding for CAS invocations is described in [17].

Since most currently existing CASs do not support OpenMath, for the immediate implementation purposes, we considered an intermediate approach: a function *procedure(Arg1,Arg2)* is translated into a corresponding OpenMath object as the one shown in the following:

```
<OMOBJ>
    <OMA>
        <OMS cd="casall1" name="procedure_call"/>
        <OMSTR>procedure</OMSTR>
        <OMOBJ>Arg1</OMOBJ>
        <OMOBJ>Arg2</OMOBJ>
    </OMA>
<OMOBJ>
```

The parser will use the information encapsulated in the OpenMath object to create the appropriate CAS command.

The internal OMOBJ objects must be OpenMath encoding of the corresponding mathematical objects, either atoms or compound. For the case of a CAS that does not parse OpenMath objects it is possible to encapsulate the generic representation of *Arg1, Arg2* using OMSTR atoms; the CAS is then responsible to convert the encapsulated string to the CAS object.

5.3. Exposing technology. In what follows we argue that exposing the full functionality of the CASs is difficult due to the high number of functions that CASs implement. Another issue is security since certain functions exposed could represent a security gap.

The first approach to expose functions of a CAS that one might consider is a one-to-one approach. This means that for every function of a CAS a corresponding operation of a service should be implemented. The experience gained by constructing the CAGS tool (see [3] for more details) leads us to the conclusion that this approach is not feasible for a large number of operations. A service with many operations exposed makes impossible dynamic creation and invocation of the service. Additionally, the WSDL 2.0 standard explicitly forbids that operations with the same name exist within the same service definition, while in a CAS functions from different libraries can have the same name.

The second approach (considered also in the GENSS platform) is to implement a generic operation: *execute(function_name, parameters)*. In this call the *function_name* represents the name of the CAS function and the *parameters* represent encoding for the parameters of the function. In this case, a Java method with the name *function_name* can be dynamically invoked using the reflection mechanisms. Afterwards, this method has to invoke the function exposed by the CAS. Hoever, also this solution, has some drawbacks. Deploying such services into a service container is not efficient and obtaining the list of the exposed functions and assuring access to them on a per user basis is not trivial.

The solution that we have considered for the GCAS uses the second approach as a starting point. We have created a registry mechanism that allows the administrator of the server to register CAS functions into a database. The general execution schema associated with this approach is composed from several steps:

1. The client invokes an *execute(CAS_ID,call_object)* operation on the service.
2. The server verifies that the CAS identified by *CAS_ID* is available on the server and that the function encoded in the *call_object* is available.
3. The server returns a unique job identifier that identifies the job and starts the execution.
4. At a later moment the client will use the job identifier to retrieve information about the status of the job and the results of the computation.

As mentioned above, the interaction between the client and the server is carried out in an asynchronous manner. Additional functionality is available using this approach, such as: the list of the CASs exposed, the list of functions of a certain CAS that are exposed, the signature of functions, and so on.

5.4. Wrapper interaction with the legacy system. CASs were meant to run on a single machine, or sometimes on clusters, in order to solve very specific computational tasks. Usually the interaction with these systems involves a command line interface. According to [32] software can be encapsulated at several levels: job level, transaction level, program level, module level and procedure level. The way that the wrapper interacts with the legacy software component depends on the native technology of the legacy component. The wrapper may use TCP/IP sockets, redirecting of the I/O streams for a process, input and output files or it may have to use JNI encapsulation.

The communication model we had to use depends on the CAS that we want to communicate with. As a regular basis, we communicate with the CASs by redirecting I/O streams in a program level encapsulation style. For the GAP and the KANT we have used the RunManager component shortly described in the previous section and detailed in [3]. The interaction with Maple was implemented using the Java API that it provides.

The WS-GRAM service offered by Globus Toolkit enables to run command line batch jobs. As an alternative to the RunManager component we implemented the interaction to the CAS by making appropriate calls to the WS-GRAM service. An efficiency comparison of the two approaches is presented in [5] proving that RunManager approach is much faster.

5.5. Implementation details. The CAS server that we implemented exposes two main categories of service operations. The first category includes operations that offer information about the CASs exposed on that server and the functions that the client is able to call. The second category refers to the generic operation that allows remote invocation of CAS functions.

The first operation category includes:

getCASList(),
getFunctionList(CAS_ID),
getNrFunction(CAS_ID),
getFunctionsByIndex(CAS_ID, startIndex, endIndex),
getFunctionsMatch(CAS_ID, stringToMatch).

These functions offer functionality for the client to discover the CASs installed on the server machine, and, for the exposed CASs, the functions being exposed. The parameter *CAS_ID* uniquely identifies a CAS system. The identifier of CASs can be obtained by calling the *getCASList()* function. Since the number of functions exposed on the CAS Server can be large, we created convenient operations to filter the list of function signatures when issuing a *getFunctionList(CAS_ID)* call. For example, we can display functions with a certain name or we can display a limited number of functions that match a certain criteria.

The actual execution of a command can be achieved by calling a generic operation *execute()*. The operation *String execute(CAS_ID, call_object)* returns a unique identifier for the submitted job. This identifier will be used to retrieve information about the status and the result of the job.

The registry of the exposed functions can be populated by the system administrator using a registry editing tool that we implemented. CAS related information stored into registry includes, but is not restricted to, the identifier of the CAS, the path where the CAS is installed and a brief description of the CAS. Function related information includes the function name, the signature and a brief description. If the CAS allows socket connections, it can reside on another machine and the specific path can include the shell command that allows the connection to that machine that runs the CAS.

GCAS's main component, CAS Server, is implemented using Java 5 SDK with supporting tools and software from Globus Toolkit 4, Apache Axis, Apache Tomcat 5 and RIACA's OpenMath Java API. The generic services implemented in the CAS Server architecture invoke CAS functions that are applied to OpenMath objects.

5.6. Grid services benefits. Several benefits already stated bellow and several others mentioned in [27] motivate the migration to the WSRF standards and exposing CAS functionality using Grid service technology.

SymGrid-Services complies with the WSRF standard for Grid services. While Grid services have different goals from pure web services (sharing computing power and resources like disk storage databases and software applications, versus sharing information), a Grid service is basically a Web service with some additions: stateful services, service instantiation, named service instances, two-level naming scheme, a base set of service capabilities, and lifetime management. These supplementary capabilities allow improved user interaction with remote symbolic computing services: previous computations performed by a service can be stored as service state, personalized services are possible due to the instantiation facilities, services can be easily modified due to the naming schemes, standard search facilities can be implemented due to the standard service data elements, and resource management can be easily done due to the transient character of the services. Complying the WSRF standard imposes not only that the interface is offered to the user, but that it preserves the original specified behavior.

With non standard Web or Grid services, the architect of the service is the one that designs the access interface to data. With the WSRF services, most of those access interfaces are already presumed and the

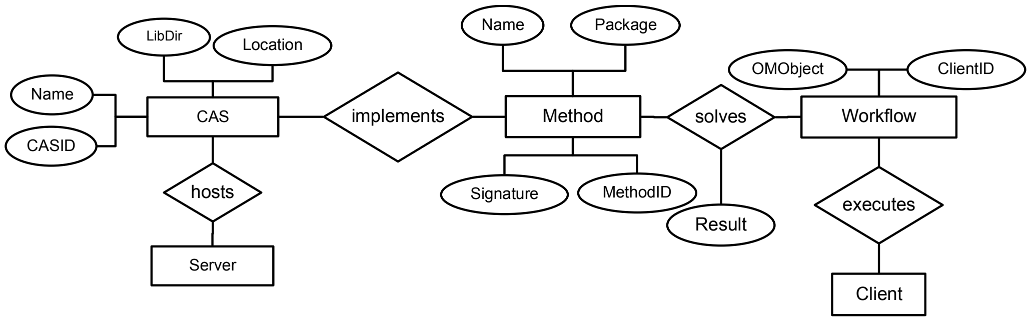


FIG. 5.2. *Data model for creating CAS services.*

developer must only provide the functionality behind the scenes. Migrating from non standard approach to a standard WSRF solution is not covered so far, to the best of our knowledge, by any methodology.

Grid services are the solution for sharing resources in a uniform way. Interfaces are described using WSDL language and XSD technologies and the resources are described using XSD data types. The structure of a Grid Resource is described, in XML terms, as a root document element that maps to the resource and several top level complex elements representing the Resource Properties (RP). Implementations of the WSRF standards, e.g. by Globus Toolkit 4, map XML documents that describe resources to in-memory representations as JavaBeans classes. The access to attributes stored in resources, as specified by the WSRF, is achieved mainly using standardized Web service operations. The Grid resource mechanism is not intended to state how data is stored at server level; the most common mechanism to store data remains backend databases.

Databases normalization usually has as a basis the conceptual Entity-Relation (ER) diagram. A methodology that transforms the ER into a blueprint of a Grid service that supports the access operations to data is an important step in the widely adoption of Grid services. Several papers [1, 16] cover the process of transforming ER diagrams to XML normalized documents. While the algorithms presented in these papers seem to offer a satisfactory solution for converting ER to XML documents, these solutions cannot be used as they are because they use attributed elements and reference attributes that are not allowed by the WSRF standard and they cannot be used in the context of Grid Services to create JavaBeans.

Several changes may be applied on existing implementation of non WSRF Web services to create corresponding Grid services. However, this approach would not represent the WSRF implementation in the semantic sense because standard intended operations functionality is replaced by non standard ones. To leverage the full power of Grid Services a simple translation would not be enough. Thus, having a set of guidelines in the Grid Service process would improve the quality of the design. The experience gained in designing and implementing Grid symbolic services led to identifying several design rules that were highlighted in [18].

Starting with the ER that was the conceptual basis for the Web service model we have presented in [5], we expose in Figure 5.2 the process that led to the current Grid services model. The structure of the Grid services we created uses the basic principles for designing Grid Resources defined in [18] and a variation of the algorithm presented in [16].

For the simplicity reasons we assume that the user only wants to gather information about the CASs exposed by the system and the functions he/she is allowed to use, to submit a request for a service, and to get the result of a computation. Several attributes of several entities must be available for the user. Moreover, entities such as the *CAS* and the *Methods* hold information that changes more often and the user should not have the right to modify them. For this reason, the Factory service that we have implemented exposes these attributes as RPs of a single Resource, not as multiple Resources, as the ER states. The Factory service is responsible for creating an additional Resource that holds the details regarding the result of the computation.

As a result of this analysis we came to the conclusion that the GCAS interface must be composed by two Grid services, i. e. a Factory stateful service and a Resource. The Factory stateful service has two RPs, the *CAS* and the *Method* that hold the *(casID, name)*, and the *(methodID, casID, name, signature, package)* respectively. The Resource has as the RP the result of the computation.

The CAS Servers are implemented as Grid services deployed in GT4 containers. A CAS Server provides access to CASs (e.g. GAP, Kant) installed on the same machine. The Resource associated with the service

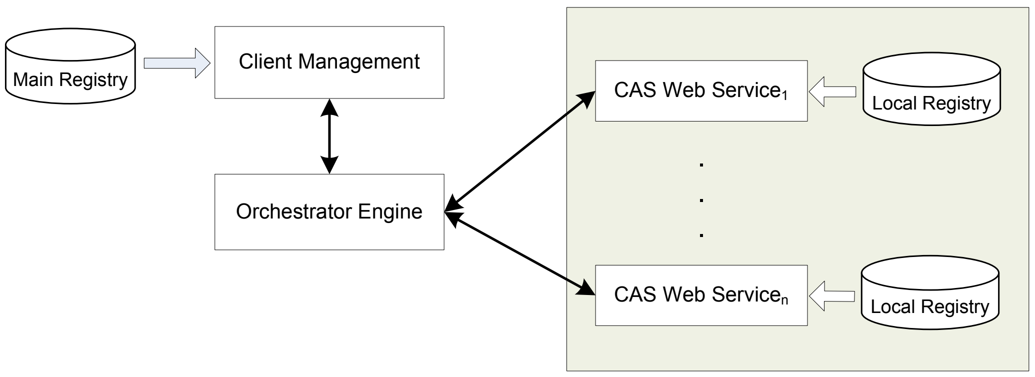


FIG. 6.1. *Server side architecture*

keeps track of the exposed CASs and functions. The Grid service's Resource is stored persistently in a backbone database rising problems of mapping these resources to database tables. For each received computation request the server creates a new WS-Resources to handle the request. The results of the computations can also be retrieved using operations specified by the WSRF standard. This is rather important advantage of using Grid services for implementing CAS servers as they offer a standard interface for accessing information about the exposed CASs.

6. Composing the SymGrid's services. A service oriented architecture relays on loosely coupled software components exposed as services. The service composition is usually done based on workflows descriptions and engines.

The static composition in workflows is achieved at the design time by specifying all the details of the composition elements, i. e. services and binding details for these services. A special type of static composition, namely the workflows with dynamic bindings, offers the ability to specify the actual addresses of the services at runtime.

The SymGrid' set of publicly available Web and Grid services intends to be highly dynamic: some of the services are occasionally available, others are vanishing, new and potentially better services become available, while for some of them the interface changes. In this situation the static composition approach fails to offer a robust solution. The alternative is to use dynamic workflows, generated at the runtime using latest information available on the services to be composed. In this case special mechanisms and techniques to identify and invoke the right Web services are required.

An in-between approach, namely semi-dynamic composition, that can be used when several characteristics of the services involved in the composition are known, was considered for SymGrid-Services and reported in [4]. The system does not have to have any a'priori knowledge about addresses of Web services needed by the composition workflow thus the binding is dynamic. Known is the structure of the operations exposed by the services.

In what follows we outline the proposed solutions. Details can be found in the following recent papers [4, 6, 18].

6.1. GCAS' composer architecture. The composer obtains the execution of mathematical based workflows with the aid of several software components. At the server side level, the main components needed to carry out the workflow execution and to manage related issues include a client manager component, an engine needed to execute the workflow, and several CAS servers that expose CAS functionality as services (Figure 6.1).

A typical scenario implies that the users specify within a CAS the interdependent tasks that compose the workflow. To execute it, the workflow is submitted to a service located at a previously known URL address. An assumption is that the user is indicating the CAS system for every task.

Several steps are needed to transform the workflow described within the CAS to a format that complies with a standard orchestration language. The workflow that results at the client side is not complete because the client is not, and should not be aware of the URL addresses of the services that will compute the subtasks. As a result, the client component sends an incomplete workflow to the client manager component.

One of the most important responsibilities of the client manager component is to obtain addresses of the CAS servers that will execute the task by consulting the main registry and to supply this information to the execution engine. For every CAS server there is a corresponding local registry that contains information about the CAS systems exposed and the functionalities supported by them. The main registry duplicates information contained into local registries that are spread on different sites of the distributed infrastructure and it is used to find appropriate CAS servers to solve the atomic tasks of the workflow.

Another responsibility of the client manager component is to send back to the client a workflow identifier that will be used later by the client to retrieve the result of the whole computation.

The management of the workflow execution must be carried out by a workflow engine located at the server side. The client management component is the one that invokes the operations provided by server machines that expose CAS functionality as Web services. It must be emphasized that all client-server interactions as well as servers to server component interactions are encoded in XML format. The description of mathematical expressions uses the standard OpenMath.

Details about the composer implementation are provided in [4].

6.2. Using standard workflow patterns. The suitability of the BPEL workflow description language for the dynamic composition of SymGrid services was investigated recently and results are reported in [6]. General workflow patterns are helping the CAS user to describe the relationships and sequence of service calls; the resulted description is deployed and executed by SymGrid-Services components implemented using Java 5 SDK relaying on the ActiveBPEL workflow engine and the PostgreSQL database servers.

Application specialists need not be aware of all the details required for the complete specification of the whole workflow using a specialized language. Instead, they only need to be able to combine several workflow patterns in order to describe a high level solution to their problem. The user-specified workflow can be automatically translated into a specialized workflow language, deployed and executed by a workflow management server. The blueprint of the client component that we have implemented can be used to enable this functionality within every CAS with a minimal effort. GAP system is used, again as representative CAS in the demonstration, to combine workflow patterns and execute workflows that use the functionality of several other CASs installed on remote machines.

The description of the problem specified at the client level is submitted to a server that will manage the rest of the process. At the client side, the workflow specified within the CAS is encoded using the XML language similar to BPEL that was described in [18]. The main reason for using an XML intermediate language instead of a complete BPEL description is the significantly larger size of the completely specified BPEL workflow. The drawback of this approach is the additional server load needed to convert the XML format to the BPEL format. The client manager component is responsible not only for receiving new workflows and for providing clients access to the result of their computation, but also for translating the XML workflow representation received from the client to the corresponding BPEL workflow format, to deploy the workflow into the ActiveBPEL engine and to launch the execution of the process.

6.3. A simple example. SymGrid-Services' client is currently able to specify workflows by composing standard workflow patterns. A very simple example is provided in [4, 6] to demonstrate the system functionality: compute the value of the Gcd(Bernoulli(1000), Bernoulli(1200)) using remote machines and two different CASs, GAP and KANT. The Gcd() is computed using the KANT system by combining the Bernoulli values obtained from two separate instances of GAP. The system allows the execution of workflows that are not bound to a two level invocation scheme. The corresponding GAP code that would allow obtaining the same result as the previous system is:

```
startWorkflow();
 startSequence();
   startParallel();
     v1:=invoke("KANT",Bernoulli(1000));
     v2:=invoke("KANT",Bernoulli(2000));
   endParallel();
   invoke("GAP",gcd(v1,v2));
 endSequence();
endWorkflow();
readOutput(processHandler);
```

The above code is translated at the client level into the simplified BPEL like format and it is submitted to a server. The simplified BPEL format is the following:

```
<workflow xmlns = "http://ieat.ro" >
   <sequence>
      <parallel>
         <invoke invokeID = "invoke_0">
            <casid>KANT</casid>
            <call>Bernoulli(1000)</call>
         </invoke>
         <invoke invokeID = "invoke_1">
            <casid>KANT</casid>
            <call>Bernoulli(2000)</call>
         </invoke>
      </parallel>
      <invoke invokeID = "invoke_2">
         <casid>GAP</casid>
         <call>gcd($invoke_0,$invoke_1)</call>
      </invoke>
   </sequence>
</workflow>
```

The server will translate this code into a regular BPEL workflow format (over 300 lines for the above described example) and will manage the execution.

At a later time, the user may access the computed result based on the identifier that it is received when submitting the workflow. More complex examples are provided in [6, 18]. The GAP library is presented in [18].

7. Conclusions. A service-oriented framework, SymGrid-Services, was introduced to manage mathematical services. One of its main components consists of several CAS servers representing symbolic computing services. The services are wrapping legacy codes. The novelty of the wrapping approach, compared with similar ones that were identified, is the fact that is based on current standards for Web and Grid services and the only non-standard technique is given by the usage of a software and function register that is proved to be more efficient than using the standard approach of a special service call. The novelty of the service access component consists in the fact that it allows the seamless access from inside of computer algebra systems to both Web and Grid services. Moreover, the symbolic computing services can be combined into complex application using standard workflow patterns. SymGrid-Services will be further developed in the near future to include specific dynamic resource brokers and schedulers. Performance tests are only at an infancy stage and a number of large new demonstrator applications need to be provided soon.

Acknowledgments. This research is partially supported by European Framework 6 grant RII3-CT-2005-026133 SCIEnce: Symbolic Computation Infrastructure for Europe.

REFERENCES

[1] M. Arenas and L. Libkin, *A normal form for XML documents*, ACM Transactions on Database Systems, Vol. 29 (1), ACM New York (2004), pp. 195–232.
[2] R. Baraka, O. Caprotti and W. Schreiner, *A Web registry for publishing and discovering mathematical services*, in Procs. EEE-05 (2005), 190-193.
[3] A. Cârstea, M. Frîncu, G. Macariu, D. Petcu and K. Hammond, *Generic access to Web and Grid-based symbolic computing services: the SymGrid-services framework*, in Procs. ISPDC 2007, IEEE CS Press (2007), pp. 143–150.
[4] A. Cârstea, G. Macariu, M. Frîncu and D. Petcu, *Composing Web-based mathematical services*, in Procs. SYNASC 2007, IEEE Computer Press (2007), pp. 327–334.
[5] A. Cârstea, M. Frîncu, A. Konovalov, G. Macariu and D. Petcu, *On service-oriented symbolic computing*, in Procs. PPAM 2007, LNCS 4967, Springer (2007), in print.
[6] A. Cârstea, G. Macariu, D. Petcu and A. Konovalov, *Pattern based composition of Web services for symbolic computations*, in Procs. ICCS 2008, LNCS, Springer (2008), in print.
[7] T. Delaittre, T. Kiss, A. Goyeneche, G. Terstyanszky, S.Winter and P. Kacsuk, *GEMLCA: running legacy code applications as Grid services*, Journal of Grid Computing Vol. 3. Springer Science (2005), pp. 75–90.
[8] J. Denemark, A. Kulshrestha and G. Allen, *Deploying legacy applications on Grids*, in Procs. 13th Annual Mardi Gras Conference, Frontiers of Grid Applications and Technologies (2005), pp. 29–34.
[9] GAP Group, *Groups, Algorithms& Programming*, http://www.gap-system.org
[10] GENSS Consortium, *Grid-enabled numerical and symbolic services*, (2005), http://genss.cs.bath.ac.uk/

[11] K. Hammond, A. Al Zain, G. Cooperman, D. Petcu and P. Trinder, *SymGrid: a framework for symbolic computation on the Grid*, LNCS 4641 (2007), pp. 447–456.
[12] MapleConnect, *HPC-Grid for Maple*, 2006, `http://www.hpcgrid.com`
[13] Y. Huang, I. Taylor, D.W. Walker and R. Davies, *Wrapping legacy codes for Grid-based applications*, in Procs. IPDPS'03, IEEE Computer Society (2003), pp. 139–147.
[14] Institute for Computational Mathematics, *IAMC: Internet accessible mathematical computation*, `http://icm.mcs.kent.edu/research/iamc.html`
[15] KANT Group, *KANT/KASH—Computational algebraic number theory*, `http://www.math.tu-berlin.de/~kant/kash.html`
[16] C. Kleiner and U.W. Lipeck, *Automatic generation of XML DTDs from conceptual database schemas*, in Tagungsband der GI/OCG-Jahrestagung, Kurt Bauknecht et. al. (Eds.) (2001), pp. 396–405.
[17] A. Konovalov and S. Linton, *Symbolic Computation Software Composability Protocol Specification*, CIRCA preprint 2007/5, University of St Andrews, `http://www-circa.mcs.st-and.ac.uk/preprints.html`
[18] G. Macariu, A. Cârstea, M. Frîncu and D. Petcu, *Towards a Grid oriented architecture for symbolic computing*, submitted to ISPDC'08 (2008).
[19] MapleSoft, *Maple*, `http://www.maplesoft.com`
[20] MapleSoft, *MapleNet*, `http://www.maplesoft.com/maplenet/`
[21] MapleSoft, *Grid Computing Toolbox* (2008), `http://www.maplesoft.com/products/toolboxes/GridComputing/`
[22] Monet Consortium, *MONET: Mathematics on the net* (2004), `http://monet.nag.co.uk`
[23] M. Naifer, A. Kasem and T. Ida, *A system of Web services for symbolic computation*, in Procs. Asian Workshop on Foundation of Software, Xiamen, (2006), `http://www2.score.cs.tsukuba.ac.jp/publications/2006-1/publications/`
[24] OpenMath Society, *OpenMath*, `http://www.openmath.org/`
[25] D. Petcu, D. Dubu and M. Paprzycki, *Extending Maple to the Grid: design and implementation*, in Procs. ISPDC'04, J.Morrison et al (eds.), IEEE Computer Press (2004), pp. 209–216.
[26] D. Petcu, D. Tepeneu, M. Paprzycki and T. Ida, *Symbolic computations on Grids*, in Engineering the Grid, B. Di Martino, J. Dongarra, A. Hoisie, L. T. Yang, H. Zima (eds.) American Scientific Publishers (2006), pp. 91–107.
[27] D. Petcu, *Between Web and Grid-based mathematical services*, in Procs. ICCGI'06, P. Dini, C. Popoviciu, C. Dini, Gunter Van de Velde, E. Borcoci (eds.), IEEE Computer Society Press (2006), pp. 41–47.
[28] W. Schreiner, *Brokering mathematical services in the global network*, in Procs. IIWAS 2003, G. Kotsis, S. Bressan and I. Ibrahim (eds.), Austrian Computer Society, vol. 170 (2003).
[29] SciFace, *MuPAD*, `http://www.mupad.de`
[30] M. Senger, P. Rice and T. Oinn, *Soaplab—a unified sesame door to analysis tools*, in Procs. UK e-Science, All Hands Meeting 2003, Cox, S. J. (ed.) (2003), pp. 509–513.
[31] E. Smirnova, C.M. So and S.M. Watt, *Providing mathematical Web services using Maple in the MONET architecture*, in Procs. MONET Workshop (2004), `http://monet.nag.co.uk/cocoon/monet/proceedings/`
[32] H. M. Sneed, *Encapsulation of legacy software: a technique for reusing legacy software components*, Annals of Software Engineering, Springer (2000), pp. 293–313.
[33] A. Solomon and C. A. Struble, *JavaMath—an API for Internet accessible mathematical services*, in Procs.Asian Symposium on Computer Mathematics (2001).
[34] A. Solomon, *Distributed computing for conglomerate mathematical systems*, in Integration of Algebra& Geometry Software Systems, Joswig M. et al (eds.) (2002).
[35] D. Tepeneu and T. Ida, *MathGridLink—A bridge between Mathematica and the Grid*, in Proc. JSSST'03 (2003), pp. 74–77.
[36] W3C, *MathML 2.0* (2001), `http://www.w3.org/Math/`
[37] Wolfram Research, *gridMathematica*, `http://www.wolfram.com/products/gridmathematica/`
[38] Wolfram Research, *webMathematica*, `http://www.wolfram.com/products/webmathematica/`
[39] A. Al Zain, P. W. Trinder, H. W. Loidl, G. J. Michaelson, *Supporting high-level Grid parallel programming: the design and implementation of Grid-GUM2.* UK e-Science Programme All Hands Meeting (AHM) (2007).
[40] A. Al Zain, K. Hammond, P. Trinder, S. Linton, H. W. Loidl, and M. Costantini, *SymGrid-Par: design a framework for executing computational algebra systems on computational Grids*, in Procs. ICCS'07, Intl. Conference on Computer Science Beijing (2007), LNCS, Springer, in print.
[41] J. Zimmer, A. Franke and M. Kohlhase, *MATHWEB-SB: A Software Bus for MathWeb* (2006), `http://www.mathweb.org/mathweb/`
[42] A. YarKhan, J. Dongarra and K. Seymour, *GridSolve: The Evolution of Network Enabled Solver*, in Procs. WoCo9, A. Prescott (ed), Series IFIP Intern. Federation for Information Processing vol. 239, Springer (2007), pp. 215–224.

Edited by: Marcin Paprzycki
Received: May 4, 2008
Accepted: May 20, 2008

Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 127–134. http://www.scpe.org
ISSN 1895-1767
© 2008 SCPE

MOBILE AGENT SYSTEMS INTEGRATION INTO PARALLEL ENVIRONMENTS[*]

DAVID E. SINGH[†] ALEJANDRO MIGUEL, FÉLIX GARCÍA, AND JESÚS CARRETERO

Abstract. In this work MASIPE, a tool for monitoring parallel applications, is presented. MASIPE is a distributed tool that gives support to user-defined mobile agents, including functionalities for creating and transferring these agents through different compute nodes. In each node, the mobile agent can access the node information as well as the memory space of the parallel program that is being monitored. In addition, MASIPE includes functionalities for managing and graphically displaying the agent data. In this work, its internal structure is detailed and an example of a monitored scientific application is shown. We also perform a study of the MASIPE requirements (in terms of CPU and memory) and we evaluate its overhead during the program execution. Experimental results show that MASIPE can be efficiently used with minimum impact on the program performance.

Key words: program monitoring, mobile agents, distributed system.

1. Introduction. Nowadays, the increasing computational power of commodity computers and interconnection networks allows efficiently executing parallel applications in low-cost platforms. Commodity computers can be organized conforming clusters which are usually specifically designed for CPU-intensive computing. For this configurations, there are several tools [1, 2, 3, 4] that provide solutions for system management. However, commodity computers can also be organized in labs or computer rooms, usually as single-user, stand-alone computers. In this scenario, CPU-intensive computing is a secondary task. Under these circumstances, the use of cluster management tools is complicated, given that they can interfere with the front-end user applications. For example, it would be necessary to explicit start the monitor tools in each compute node or performing a system reboot for starting a different operative system. Consequently, in this kind of environments it is more interesting to use non-intrusive tools for monitoring parallel programs. These tools should be automatically executed, using the same execution environment as the one employed with parallel application.

The work presented in this paper addresses this problem by means of a new monitoring tool called *Mobile Agent Systems Integration into Parallel Environments* (MASIPE). This tool was presented in [5] and this paper is an extension of the work presented there. MASIPE is designed for monitoring parallel applications by means of mobile agents. It provides a platform for executing agent-based programs in each computing node where the parallel program is being executed. The agent is executed sharing the same memory space than the parallel program thus it is able to read program variables and modify its value. In addition, the mobile agent includes code for executing user-defined operations on each compute node. All these operations are performed asynchronously without interrupting/modifying the normal program execution. The agent is able to obtain information about the compute node (memory and CPU consumption, network use, etc.). All these information (related to each parallel program process and compute node) is collected in the agent private memory space, transferred along the agent itinerary compute nodes and finally stored and visualized in the front-end GUI. MASIPE allows the integration with any kind of C, C++ or Fortran parallel programs for distributed memory (MPI [6]) and shared memory (OpenMP [7]) architectures. Our tool is freely distributed. More information about MASIPE can be found in [8].

The structure of this paper is as follows. In Section 2 a description of the internal structure of each element of MASIPE is shown. Section 3 presents an example of its use for monitoring a parallel application that simulates the emission, transport and deposition of pollutants produced by a power plant. Next, Section 4 evaluates the impact of MASIPE on the program performance taking into account both memory and CPU requirements. Section 5 analyzes the related work and finally, Section 6 presents the main conclusions of this work.

2. MASIPE internal structure. Due to the massive expansion of Internet usage, distributed computation has become one of the most common kinds of programming paradigms. New alternatives have appeared, some examples of these are grid architectures, peer to peer systems, remote method invocation, web services, network services or mobile agent systems. MASIPE is a distributed mobile agent system that provides support for transferring and executing mobile agents in different compute nodes in an autonomous way. The communications between the compute nodes are implemented using the CORBA middleware. More specifically, we

[*]This work has been partially funded by project TIN2007-63092 of Spanish Ministry of Education and project CCG07-UC3M/TIC-3277 of Madrid State Government.

[†]Universidad Carlos III de Madrid, Computer Science Department, 28911 Leganés, Madrid, Spain. (dexposit@inf.uc3m.es, alejandro.miguel@gmail.com, fgcarbal@inf.uc3m.es, jcarrete@inf.uc3m.es).

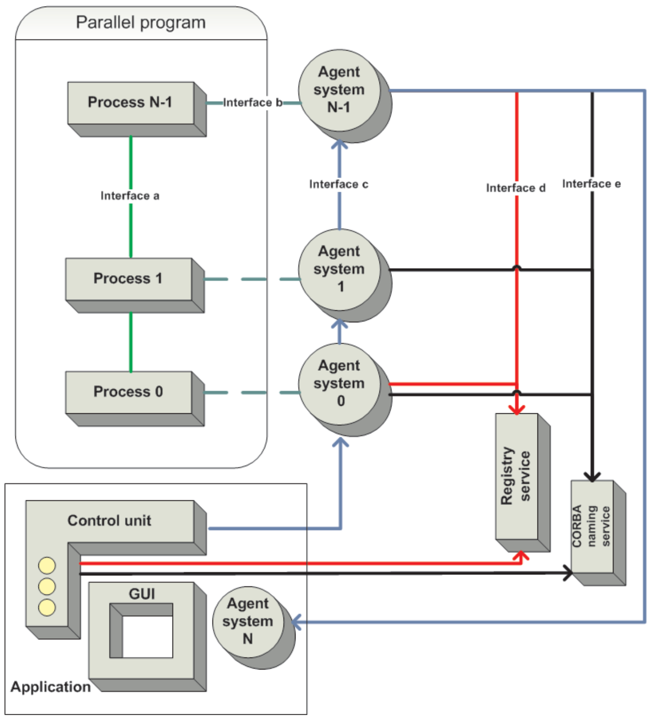


FIG. 2.1. *Diagram of MASIPE distributed and internal structure.*

have adopted the MASIF [9] specification proposed by OMG. This specification contains a description of the methods and interfaces of the agent-based architecture.

Figure 2.1 shows a diagram of MASIPE architecture as well as the relationships between its elements. MASIPE consists of three main components:

- **Agent System (AS).** This module receives the mobile agent, deserializes it, extracts its data and executes its associated code. When the agent execution concludes, AS collects and packs the agent code plus processed data, serializes it and sends it to the following compute node (where another AS is running). AS is attached to each parallel program process, sharing its memory space. This property allows reading or modifying (by means of the mobile agent) the data structure contents (variables or arrays) of the parallel program that is being monitored.
- **Register Service (RS).** This component has the task of registering AS and agents as well as providing its localization, that is, the IP address and port number of each element. Using this information, when a mobile agent arrives to an AS, the agent itinerary (stored with its data) is read in order to obtain the following AS where the given agent has to be transferred.
- **User Application (UA).** This module consists of three different elements: control unit, ending AS and GUI. The **control unit** creates the agent which is constituted by a header and a payload. The header contains the control data (including the agent itinerary). The payload contains the user-defined code (which performs user-defined operations) and its associated data. The control unit packs all this information (both code and data) and sends it to the first AS. Then, the agent is transferred along all the AS included in its itinerary. Finally, the **ending AS** receives the agent and unpacks all the collected data. These data are shown to the user by means of the **GUI** which includes components for graphically displaying the collected data. The User Application includes the following additional functionalities: it allows defining the system topology (agent itinerary); it performs agent tracking; it gives support for sending periodically (under a defined frequency) different types of agents; it receives, manages and displays the agent data and it includes policies for log generation.

All the communication operations described above (with the AS, RS and UA) are performed using pre-defined interfaces that are declared according the MASIF specification. Despite having this complex structure, the use of MASIPE is quite simple. Both RS and UA modules are independently executed as stand-alone

```
        First stage: add parameters
L1      call AS (0, 0, myrank)
L2      call AS (0, 1, it)
L3      call AS (0, 2, acum_time)
        Second Stage: start agent system
L4      call AS (1, 0, 0)
```

FIG. 2.2. *Example of a Agent System call in a Fortran program.*

programs. AS module is provided as a library that is linked with the parallel program subjected to be monitored. In addition, an *entry point* has to be defined in the parallel program. This point represents a call to the AS routine that starts executing the monitoring service.

Figure 2 shows an example of its use in a Fortran program. In the first stage, the user provides the program data structures that will be accessed by the agent. This task is performed in lines `L1`, `L2` and `L3` where calls to AS routine receive a pointer of three different program variables[1]. In this example, variables *rank*, *it* and *acum_time* are monitored by the agent. A zero value in the first argument indicates to the AS routine that has to store the given pointer value (third argument) in an internal data structure using the integer given by the second argument as an index. In the second stage, the Agent System is started calling the AS routine with a value in the first argument equal to one (line `L4`). Internally, a new thread is started and the code of the AS routine is executed concurrently with original process. Note that this process is automatically performed in each compute node where the parallel program is being executed.

3. Case of study: Monitoring STEM-II parallel application. In this section we show an overview of MASIPE functionalities when used in a real environment of parallel computing. More specifically, our proposal was used for monitoring STEM-II application.

STEM-II is a 3D grid-based model that simulates $SO_x/NO_x/RHC$ multiphase chemistry, long-range transport and dry plus wet acid deposition. This application is used for calculating the distribution of pollutants in the atmosphere from specified emission sources such as cities, power plans or forest fires under particular meteorological scenarios. The prediction of the atmospheric pollutants behaviors includes the simulation of a large set of phenomena, including diffusion, chemical transformations, advection, emission and deposition processes. This model was successfully used for the control of the emissions of pollutants produced by the Endesa power plant of As Pontes (Spain). In addition, STEM-II was chosen as case of study in the European CrossGrid project, proving its relevance for the scientific community from an industrial point of view as well as its suitability for the high performance computing.

STEM-II code structure is complex, consisting of more than 150 functions and 13.000 lines of Fortran code. STEM-II performs iterative simulation and has a multiple nested structure. The outmost loop is the temporal loop that controls the simulation time. Inner loops traverse the mesh dimensions computing for each cell (associated with 3D volume elements) the chemical processes and transport of the pollutants. This task is performed in parallel using the MPI standard library for implementing communications. A detailed description of the parallel program structure can be seen in [10].

For this scenario, two different agents were designed. The first type of agent (called `A-Type`) is used for tracking different STEM-II variables. More specifically, we have considered the following ones:

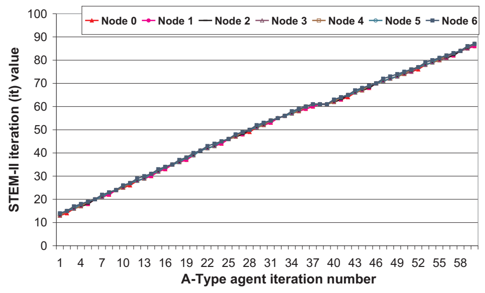
1. Rank identification (*myrank*). This variable is the MPI rank associated to each process. It is used for identifying each MPI processes during the communication operations.
2. Iteration (*it*). It corresponds to the iteration number of the outmost loop. Each loop iteration corresponds to a minute of the simulation time step.
3. Accumulated time (*acum_time*). It accumulates the execution time of the `rxn` routine. This routine performs the chemical simulation in each mesh node. `Rxn` is the most time-consuming routine of the program (70% of the complete program execution). Given that STEM-II is iterative, this routine is executed in each time step (outmost loop). In *acum_time* variable, the execution time of `rxn` is accumulated along different iterations.

[1]Note that in Fortran an argument variable of a routine is internally considered as a pointer. In case of programs written in C, standard pointers should be used instead.

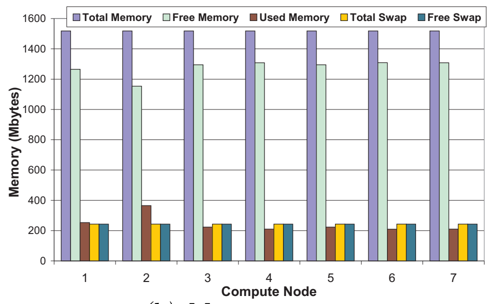
An `A-Type` agent includes logic for reading, checking and modifying these parameters. In the case of *acum_time*, the mobile agent includes a threshold (with a value specified and initialized by the agent). When agent is executed it tests whether *acum_time* reaches this threshold. In case of being affirmative, the agent resets its value. Otherwise, its value is kept. In addition, the values of all these parameters are stored with agent data and further transferred to the UA for its visualization.

A second agent, called `B-Type`, was designed for collecting information about the compute nodes. Several features were collected; examples of them are the memory, CPU, disk-swap use and number of processes in execution.

In our experiments, we have executed STEM-II in a computer room of AMD Athlon(tm) XP 1700+ computers with 1.5 GB RAM memory, interconnected by a Fast Ethernet 100 Mbps network. Operative system is Linux Sarge 2.6.13. STEM-II was compiled with MPI-2 library LAM 7.1.1 and was executed using seven computing nodes. Regarding MASIPE, we used the Mico 2.3.12 CORBA distribution. We used an extra node for storing the CORBA name service and Register Service. Another extra node was used for executing the User Application component.

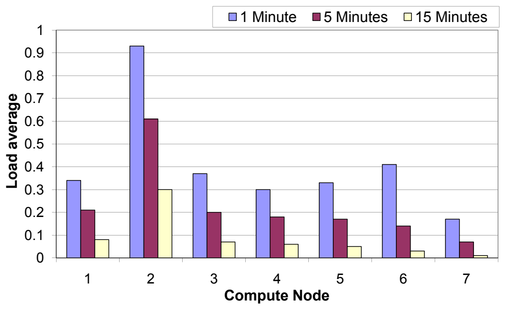


(a) Stem-II iteration (it) value.

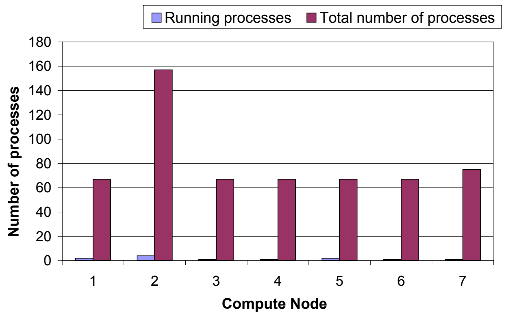


(b) Memory usage.

FIG. 3.1. *Examples of captured data using `A-Type` and `B-Type` agents.*



(a) System load average.



(b) System active processes.

FIG. 3.2. *Examples of captured data using the `B-Type` agents.*

Taking advantage of MASIPE infrastructure, `A-Type` and `B-Type` agents were periodically sent for monitoring both program and computer parameters. Figure 3.1(a) shows the *it* value captured for the `A-Type` agent. X axis represents the agent execution number (agent iteration). That is, each `A-Type` agent has associated a new iteration number when completes the itinerary that has assigned. In each agent iteration, seven STEM-II *it* values are captured (one for each compute node). Note that these values are asynchronously read by the agent which monitors the parallel application without interfering with the program execution. Figure 3.1(b) shows the memory usage values captured by `B-Type` agent related to the memory use. In this case, the agent captures this information in each compute node.

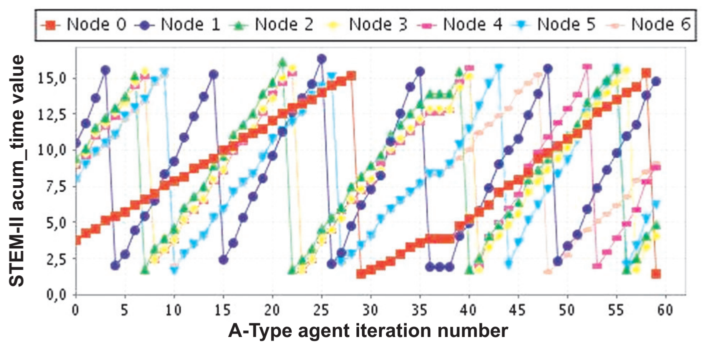


FIG. 3.3. *Example of data capture and visualization using MASIPE GUI.*

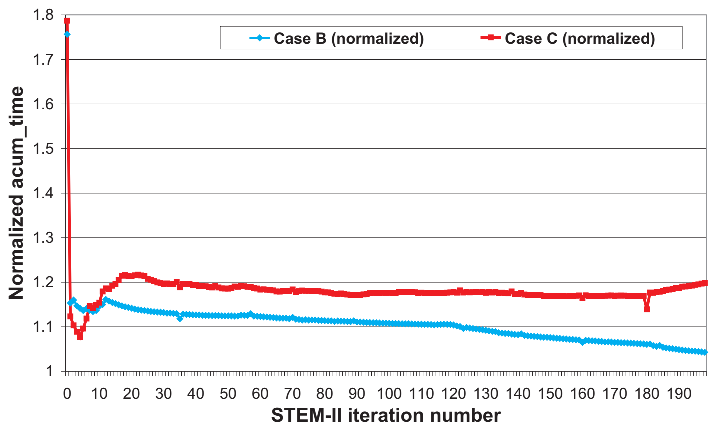


FIG. 4.1. *Evaluation of MASIPE overhead on STEM-II execution time.*

Figure 3.2 illustrates another example of captured values. Figure 3.2(a) shows the *load average* for last minute, 5 minutes and 15 minutes. This magnitude represents the average of processes in Linux's run queue marked running or uninterruptible during last minute. The load averages differs from CPU percentage in two significant ways:

1. Load average measures the trend in CPU utilization not only an instantaneous snapshot, as does percentage.
2. Load average includes all demands for the CPU not only how much was active at the time of measurement.

Note that in this test the compute node 2 has more computational load than the rest of the nodes. Figure 3.2(b) shows number of running processes and the total number of processes for each computational node. We can see that, again, compute node 2 has a higher number of processes than the rest of the nodes.

All values shown in Figures 3.1 and 3.2 were collected by the agent from each compute node and subsequently transferred to the User Application at the end of the agent itinerary. The User Application includes functionalities for storing in disk all these values, thus, they can subsequently be read, processed and displayed. In addition, the User Application includes a GUI for visualizing these data (as shown in these figures). Figure 3.3 shows an example of this visualization for the *acum_value* variable. Again, a `A-Type` agent was periodically sent, thus X axis represents the agent iteration number. But now, the *acum_value* variable is modified according a given threshold (with a value of 15). In the figure we can observe that the variable is reset when reaches the threshold. Note that each processor takes different time executing `rxn` routine thus *acum_value* increases at different rate for each node.

4. Performance evaluation. We have taken advantage of the computing environment described in the previous section for evaluating the impact of MASIPE in the parallel program performance. In order to evaluate all possible situations, three different scenarios where considered:

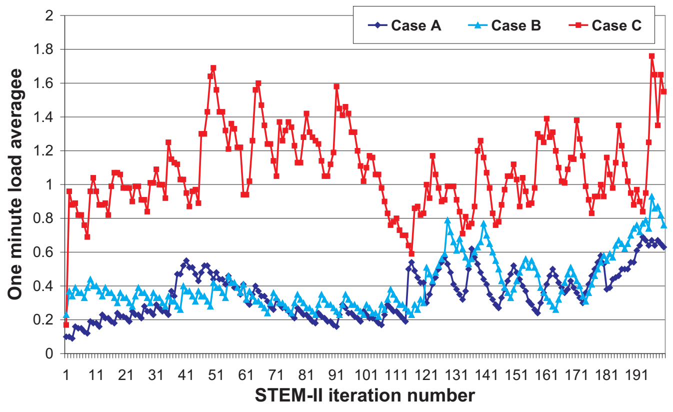


Fig. 4.2. *Evaluation of MASIPE overhead on load average.*

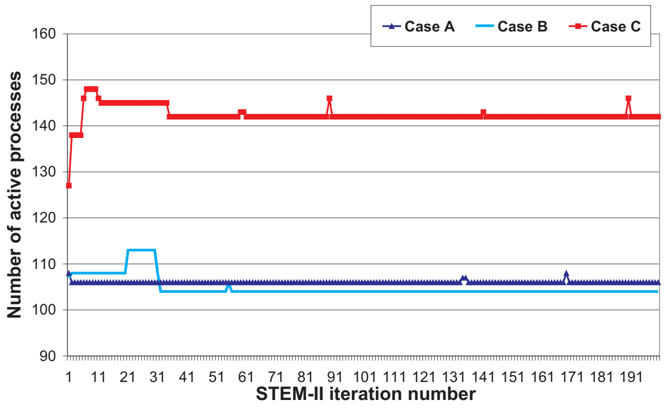


Fig. 4.3. *Evaluation of MASIPE overhead on number of active processes.*

1. **Case A:** STEM-II is executed as an stand-alone program. MASIPE tool is not used. This represents a reference scenario.
2. **Case B:** STEM-II is executed in combination with MASIPE but no agents are executed. In this scenario we evaluate the impact of the Agent System on STEM-II performance.
3. **Case C:** STEM-II is executed in combination with MASIPE. In addition, an agent is periodically sent with the maximum frequency that the system can achieve. In this scenario evaluates a full-operational MASIPE execution environment.

In all these cases, STEM-II is executed on seven compute nodes. In Case C we use a `A-Type` agent which performs an infinite round-circuit across the entire ASs. We modified STEM-II allowing collecting the performance measurements directly from this program in all the scenarios.

Figure 4.1 shows the impact of MASIPE on `rxn` subroutine. Results are normalized by Case A. We can see that the use of MASIPE without agents implies and increment around 10% on the `rxn` execution time. When agents are being periodically executed, this overhead reached up the 20% of `rxn` time. Note that we are considering only `rxn` routine because: Firstly, it is a compute-intensive routine with no E/S nor communications operations (we want to evaluate the CPU overhead) and secondly, it is the most consuming part of the program.

Figure 4.2 represents the one-minute load average for the three cases. In this figure we can see that the MASIPE load average overhead is minimal when no mobile agents are employed. Otherwise, the load average is incremented in one unit that means that are two CPU-consuming processes in execution[2].

Figure 4.3 shows the number of active process (including non CPU-demanding processes) for each one of the considered cases. Again, the use of MASIPE infrastructure increases this number in two processes. In contrast, when mobile agents are continuously used, the number of active process increases from 114 up to 142.

[2]Note that although the number of process in execution increases, the whole application execution time is not doubled as we can see in Figure 4.1.

This increment is not related to the MASIPE nor the mobile agent implementation but to the CORBA internal structure where multiple threads are automatically created.

The memory requirements of STEM-II (Case A) are 350 MB in each compute node. Starting the ASs (Case B) consumes 8 MB extra memory (an increment about 2%). When a mobile agent is fully operational (Case C) this extra memory rises up to 57 MB (16% increment respect to Case A). Taking into account the amount of installed memory in commodity computers, the memory requirements of MASIPE for this case of study are reduced.

5. Related work. There are many cluster administration tools. One well known example is Rocks [1]. Rocks is a management tool that makes complete Operating System installation and its further administration and monitoring. Other relevant schemes for system monitoring are Ka-admin project [2], Vampir [3] and Paraver [4]. However, all of them do not include functionalities for monitoring parallel applications accessing to their memory space.

Example of agent systems are Jade [11] and Mobile-C [12]. The agent mobility in Jade is achieved through Java object serialization, but Java implementation limits the integration with C/C++/Fortran codes. Mobile-C uses ACL messages for mobile agent migration which is an alternative to our implementation based on CORBA. In [13] an instrumentation library for message passing parallel applications, called GRM, is presented. GRM collects trace data from parallel applications using a similar way than our scheme. However, there are important differences in the tool architecture: in GRM all the ASs send the data to the User Application introducing contention risks in this point. The integration with Mercury Monitor presented in the paper does not solve this drawback. In contrast, we use a mobile agent program paradigm that reduces these risks given that the agent traverses different ASs. The agent itinerary in user-defined and can be dynamically changed (adding decision logic to the agent). In addition, with MASIPE it is possible to introduce new designed agents without changing (even restarting) the tool. This strength provides MASIPE a broad range of uses. Finally, CORBA allows using our tool on heterogeneous platforms.

6. Conclusions. MASIPE is a distributed application that gives support to user-defined mobile agents, including functionalities for creating and transferring the mobile agent along the compute nodes of its itinerary. In each node, the mobile agent can access to the node information as well as the parallel program memory space. In addition, MASIPE includes functionalities for managing the agent data.

MASIPE allows the integration of user-defined code (included in the mobile agent) with a generic parallel/sequential program. This process only requires of a single functional call to ASs (represented as a dynamic library). The mobile agent can not only read the program data but also modify it. This introduces a vast number of possibilities of dynamic interaction between the mobile agent and the program: check pointing (the mobile agent collects and stores the critical data of the program), convergence checking (the mobile agent evaluates the numerical convergence of iterative me-thods, even introduces corrections), evaluation of the program and system performance, etc. These interactions are performed asynchronously, without interfering with the monitored program.

The tool design was performed based on MICO Corba and Java IDL following the MASIF specification, which strengths its capacity of use on heterogeneous platforms (including compute rooms, clusters and grid environments). The installation requirements are minimal, being necessary just a single dynamic library (AS component) in each compute node. No administrator grants are needed for its use.

MASIPE was applied with success to monitor STEM-II parallel application. Several tests were made including reads and writes in the application memory space as well as capturing information of each compute node. Experimental results show that MASIPE overhead is reduced both in terms of CPU and memory consumption.

As future work we plan to develop more MASIPE functionalities like new database functionalities for managing the data collected by the mobile agents (specially in checkpoint operation), improving the GUI for dynamically visualizing the distributed data (obtaining an snapshot of the simulation), introducing new commands for controlling the program behavior (I/O operations, start and resuming the program) and its application to grid platforms.

REFERENCES

[1] F. D. Sacerdoti, S. Chandra and K. Bhatia, *Grid systems deployment & management using Rocks,* in CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing. (2004) pp. 337–345.

[2] P. Augerat, C. Martin and B. Stein, *Scalable Monitoring and Configuration Tools for Grids and Clusters,* in 10th Euromicro Workshop on Parallel, Distributed and Network-Based Processing. (2002) pp. 147–153.

[3] *Vampir*, in `http://www.vampir-ng.de`

[4] *Paraver: the flexible analysis tool*, in `http://www.cepba.upc.edu/paraver`

[5] David E. Singh, Alejandro Miguel, Félix García, Jesús Carretero, *MASIPE: A tool based on mobile agents for monitoring parallel environments,* Third Workshop on Large Scale Computations on Grids in conjunction with the Seventh International Conference on Parallel Processing and Applied Mathematics 2007. Lecture Notes in Computer Science. **4967**. (2008).

[6] William Gropp and Ewing Lusk and Anthony Skjellum, *USING MPI Portable Parallel Programming with the essage-Passing Interface,* in The MIT Press. (2004).

[7] R. Chandra, *Parallel programming in OpenMP,* in Morgan Kaufmann Publishers. (2001).

[8] *Mobile Agent Systems Integration into Parallel Environments (MASIPE)*, in `http://www.arcos.inf.uc3m.es/~masipe`

[9] *OMG Mobile Agent Facility Specification* in `http://www.omg.org/technology/documents/formal/mobile_agent_facility.htm` (2007)

[10] M. J. Martn, D. E. Singh, J. Carlos Mourio, F. F. Rivera, R. Doallo and J. D. Bruguera, *High performance air pollution modeling for a power plant environment,* in Parallel Computing. **29**, (2003) pp. 11–12.

[11] F. Bellifemine, A. Poggi and G. Rimassa, *Developing Multi-agent Systems with JADE,* in Lecture Notes in Computer Science. **1986**, (2001) pp. 42–47.

[12] B. Chen, H. H. Cheng and J. Palen, Mobile-C *A mobile agent platform for mobile C/C++ agents,* in Software. Practice and Experience **36**, (2006) pp. 1711–1733.

[13] N. Podhorszki, Z. Balaton and G. Gombás, *Monitoring Message-Passing Parallel Applications in the Grid with GRM and Mercury Monitor,* in European Across Grids Conference. (2004) pp. 179–81.

Edited by: Dana Petcu, Marcin Paprzycki
Received: May 5, 2008
Accepted: May 20, 2008

Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 135–142. http://www.scpe.org

ISSN 1895-1767
© 2008 SCPE

WEB PORTAL FOR LARGE-SCALE COMPUTATIONS BASED ON GRID AND MPI

ASSEL ZH. AKZHALOVA[*], DANIAR Y. AIZHULOV[*], GALYMZHAN SERALIN[*], AND GULNAR BALAKAYEVA[†]

Abstract. The real and specific problem that underlies the Grid concept is coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. This concept is realized in the most popular problem solving environments (PSEs) such as NetSolve and WebPDELab. These systems use some approaches to computational Grids and Web browser interfaces to back-end computing resources. The aim of our work is to build PSE implemented as a Web portal that allows clients to choose the most appropriate services to solve some problems using of matrix-algebra and numerical methods based on MPI techniques. In addition, it is available to extend the library of the Web-portal by loading computational algorithms. The proposed system allows to users a rapid prototyping of ideas, detailed analysis, and higher productivity.

Key words: grid, web portal, online problem solving, MPI.

1. Introduction.

The development in IT industry and manufacturing produce the most complex problems and the majority of them require finding exact results. Today the most actual problems are connected with such commercial applications as financial services, life sciences and manufacturing. These applications include seismic analysis, statistical analysis, risk analysis, and mechanical engineering, weather analysis, drug discovery, and digital rendering. To solve rising problems in these areas we need a large infrastructure consisting of supercomputers, advanced algorithms, and programming tools. The most appropriate tool to realize this infrastructure is the Grid computing.

The third generation of Grid computing introduced a service-oriented approach leading to commercial projects in addition to the scientific projects. The real and specific problems that underlie the Grid concept are coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations. This concept is realized in the most popular problem solving environments (PSEs) such as NetSolve and WebPDELab. These systems use some approaches to computational Grids and Web browser interfaces to back-end computing resources.

As known, Grid systems can be classified depending on their usage as: computational Grid, data Grid, service Grid. For example, TeraGrid—NSF funded linking 5 major research sites at 40 Gbs (`www.teragrid.org`), European Union Data Grid—grid for applications in high energy physics, environmental science, bioinformatics (`http://www.eu-egee.org`), Access Grid—collaboration systems using commodity technologies (`www.accessgrid.org`), Network for Earthquake Engineering Simulations Grid—Grid for earthquake engineering (`www.nees.org`). Network Enabled Solvers (NetSolve) is an example of a Grid based hardware/software server. Its original goal was to free domain scientists from having to perform these tedious tasks when they needed to use numerical software, particularly on multiple platforms. The principle of NetSolve is client/agent/server model [2]. All requests from client are accepted by the agents, the agents manage the requests and allocate servers to service, and the servers receive inputs for the problem, do the computation, and return the output parameters to the client. NetSolve can be invoked via C, FORTRAN, MATLAB, or Mathematica interfaces. Fault tolerance and load balancing are supported within the system.

Another example of PSE is WebPDELab. WebPDELab is a Web server that provides access to PELLPACK [3], a sophisticated problem-solving environment for partial differential equation (PDE) problems. WebPDELab interface supports users by a collection of case studies such as flow, heat transfer, electromagnetism, conduction. Any registered user can upload PELLPACK problem-definition files, such as .e files, mesh files, and solution files from the previous sessions. WebPDELab is implemented with the help of virtual networking computing (VNC). It runs the application and generates the display, a viewer that draws the display on the client screen and a TCP/IP connection between a server and a viewer. Within WebPDELab it is realized security for the user, server, and payment for computing services. Despite of existence of the described systems more and more scientists face such problem as a result of computational resources' lack. Certainly, there are several powerful high performance computers that are allocated in developed countries. However most of supercomputers are local and joining to them is not trivial. Sometimes a scientist has to overcome a number of bureaucratic obstacles before getting an access to supercomputer and solve the problem. Moreover, only huge problems are usually

[*]Kazakh National University, Mechanics and Mathematics Faculty, Computer Science Department, Masanchi street 39/47,050012 Almaty, Kazakhstan ({`ak_asel, aizhol`}`@yahoo.com, gseralin@mail.ru`).
[†]Kazakh British Technical University, Tole be street 59, 050091 Almaty, Kazakhstan (`g.balakaeva@kbtu.kz`).

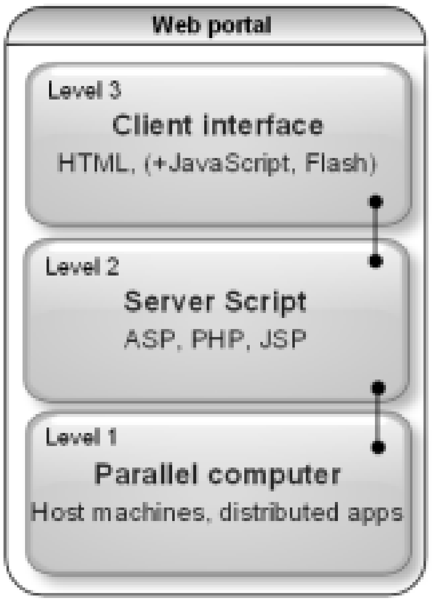


Fig. 2.1. *Three levels model of proposed system.*

introduced for the execution on such resources. In the same time, there are a lot of problems that should be solved by scientists over the world with the help of open computational resources. The emergence of networked computers has led to the possibility of using networks to launch parallel tasks remotely.

One of the problems of existing parallel systems is scalability. The rapid growth of computer networks and continuous changing of the network technologies such a huge number of different types of Web-services has a straight impact for reliable, flexible, and high-performing network software development.

The previous work [1] presented in LaSCoG'07 was concentrated on the architecture design of the proposed Web-portal. It was describing the principles of making large-scale computations through the Web-portal. This article is an extension of the part of Web-portal's functionality. Here, we propose the Service Manager component that supports the load balancing into the system and, improves the performance of the system. We consider the system performance problem as an optimal resource allocation problem. We formulate the problem of optimal resource allocation as finding an optimal sequence of servers at each time to keep the system on desirable response time and the total cost of the using servers would be minimal. This kind of problem belongs to optimal control problem. In this article, we propose to use a dynamic programming technique in order to solve the optimal control problem.

We propose a system that allows solving different scientific problems at any time through the Internet technologies. In this article we present a library that includes a parallel algorithm for two-dimensional non-stationary heat conductivity equation. The system offers an attractive alternative to expensive supercomputers and parallel computer systems for high-performance computing.

2. Design.

2.1. Architecture. The main idea is to merge server technologies and MPI programming tools in one system. The basic scheme of the system can be presented by three levels infrastructure. Figure 2.1 shows this model.

Data flows between levels from the third level to the first one. The second level is a middleware. The third level is a client interface. Client interface gives to the client Web form where input data can be uploaded to the server. There are two ways to input the data: completing the Web form or uploading the text file to the server. The second way is better when working with a large amount of data. Predefined templates allow to structure data before uploading. In addition, third level offers flash and Java scripts to check data and also display graphics. The second level is responsible for several tasks. Firstly, the information is processed

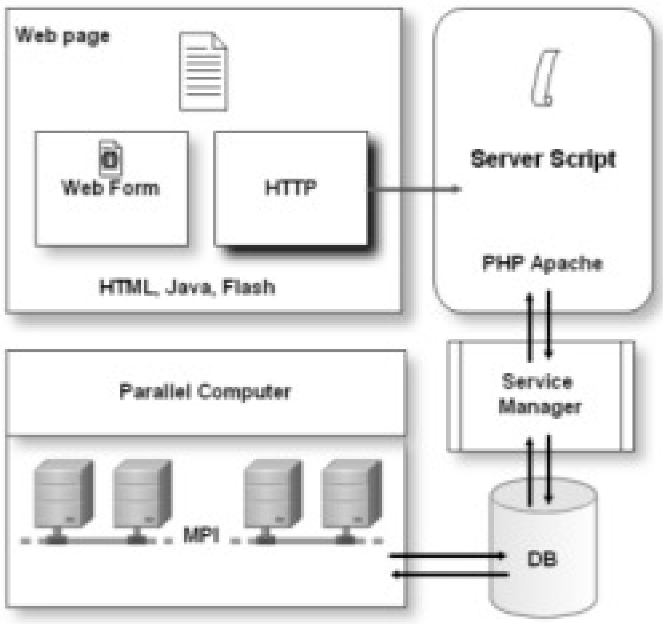


Fig. 2.2. *The architecture of proposed PSE.*

and converted to data according to the template that depends on the problem being used. The second level also provides security. The server scripts validate the data on the conformity and check whether user is going to use parallel computer or not with relevant purposes. The server is connected to the database that stores all information about previous executed tasks. The user can control and analyze the results with the help of diagrams and tables. Special agents find resources, control time execution and store it in the database. In fact, the database supports all needs starting from storage of authorization data and ending on control load balancing of parallel computer. In our case, the virtual parallel computer is the set of workstations connected by bus in P2P topology. The parallel computations are performing using MPI techniques. The main program, which is precompiled executable, reads the input data and calls for the appropriate task. Each task makes references to mpi.h library to implement communication and synchronization between processes. To implement parallel numerical calculations there is an additional library with prepared and ready for execution numerical methods. Synchronization is done by non-blocking sending and receiving routines [4]. If the same task is called by many users at once then the main program find free workstations and send them equal portions of tasks. There are different techniques to manage shared task set: cooperative lists manipulation and parallel access queue using fetch&add [5]. We offer a centralized load balancing technique. Figure 2.2 shows the architecture of the proposed PSE.

2.2. Functionality. The connection between third and second level is based on HTTP protocol. There is no need to make persistent connection between client and server as client has to make only two actions: to send the input data and receive the result. It is clear that the browser should be forced to check whether parallel computer has finished the work or not.

After the data had been posted to server, server script starts validating all received information. It authenticates the user, creates session, and validates the data, checks workload of parallel computer. After all these actions have been completed the server is ready to communicate with parallel computer. The communication is carried out by reading and writing files. This process does not waste much time, because reading and writing are made only once for each computation. The server starts an execution of the chosen problem. The agent starts an appropriate application ("calculator") on host-machines which were chosen according to strategy described in previous subsection. "Calculator" is one program from the set of given samples which are the set of programs allocated on hosts-machines. The agent transfers two arguments to the "calculator": amount of processes and input data. It calls MPIRun with required arguments and a parallel program [6].

The code of the program contains compiling directives with call functions from MPI library to spawn required quantity of processes and start implementation requested by the client. The output information after termination of calculations is saved in a special log-file that subsequently will be presented on a browser of the client. The client can also download the output data to a file. All input and output data are saved on the database. All data will be written down in a text file. In the failure case, the information with mistakes report will be written down in log-file. Server script opens the log-file with results, reads, and writes down in a database. Figure 2.3 shows the user interface of the Web-portal.

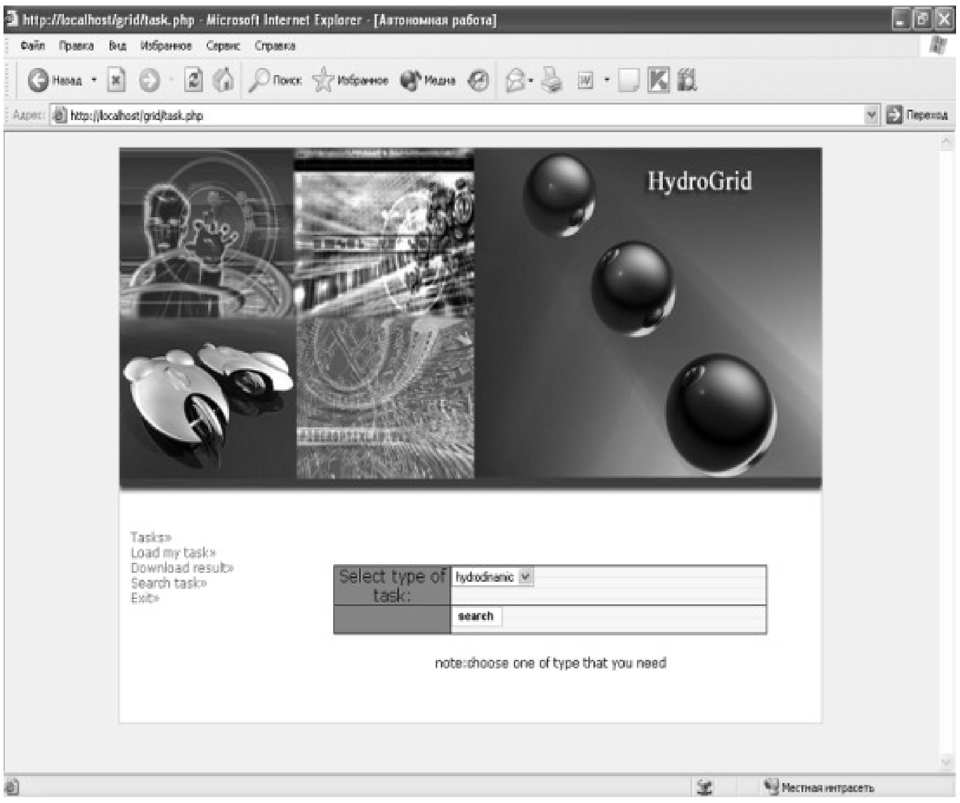

http://localhost/grid/task.php - Microsoft Internet Explorer - [Автономная работа]
Файл Правка Вид Избранное Сервис Справка
Назад
Поиск
Избранное
Медиа
Адрес: http://localhost/grid/task.php
Переход
HydroGrid
Tasks»
Load my task»
Download result»
Search task»
Exit»
Select type of task:
hydrodinamic
search
note:choose one of type that you need
Местная интрасеть

Fig. 2.3. *GUI of the Web portal.*

We developed the parallel algorithm for two-dimensional non-stationary heat conductivity equation in rectangular area. To solve the problem we create two-dimensional grid $M \times N$ that was divided on six parts (Figure 2.4).

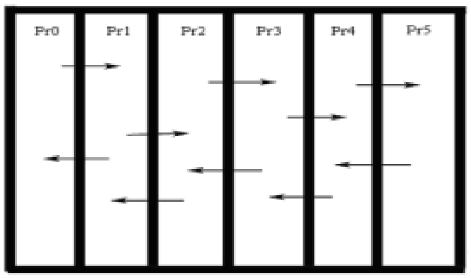


Fig. 2.4. *The divided area for the problem solution.*

We use the explicit scheme for solving of the equation in private derivatives. The parallelization technique involves not blocked data exchange between processes. MPI has a number of procedures for realization of asynchronous data transmission. In order to cancel an asynchronous exchange we call additional procedure that checks whether operation is completed or it is waiting for completion. We use send-buffer after all processes have terminated. Figure 2.5 shows us the computation time of the algorithm and it also confirms the benefit of parallel execution. The prototype of the algorithm was included to the library.

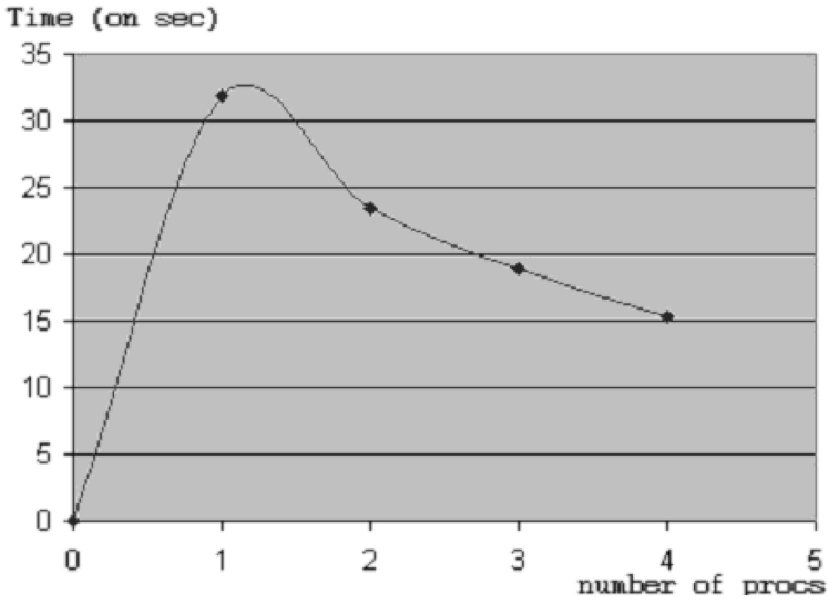


FIG. 2.5. *Execution time of the parallel algorithm.*

In addition, the server script generates a graphical representation of obtained results. When server script reads data from log-file it uses a system of flags. The flag will be equal to zero if "calculator" still works else it equals to 1 if "calculator" has finished work. The keeper of a flag is an output file, which is also provided in the termination detection service. It is available to extend the library of system by loading actual problem-solving algorithms. The system gives opportunity to view obtained results, save it in different text and graphics formats, and to analyze them.

2.3. Performance Control. The proposed PSE has some features of dependability and reliability. These features are realized on the second level by using of Service Manager component.

For instance, we face a situation when there is more than one request done simultaneously. It leads to the problem of sharing resources between the clients. To resolve this problem the system proposes a combination of the two decisions:

1) To block the server until the problem of one client is executed.

2) In regular intervals to distribute resources between clients, and supply them with full statistics about a quantity of clients that are on-line at the moment.

The task is analyzed by the following rules: how much time we need to spend to execute it and how many processes are required. Those functions are carried out by Service Manager's component. Here, we describe the principle of component functionality and the embedded algorithm.

Thus, we take a queued network model as the basis of Service Manager's component of the proposed system. We now consider queued networks that can evolve over discrete time. We treat the architecture of a queued network as abstract, in the sense that each service in the architecture defines a required functionality, but is yet to be instantiated with an actual service.

As we are only concerned with performance properties of evolving networks, we model such directories of alternative instantiating services by sets of service rates. From our performance oriented perspective, an evolving architecture consists of a sequence of service allocation choices to an abstract queue over time.

We now define our models of service communication and reconfiguration. An open queued network consists of a set of services connected to each other. We assume each service can receive messages, process the message and then send a new message. Each service takes a finite amount of time to process a message. Processing time may vary over the life of the network. Messages are sent between services via connections. Each service has its own queue and, if more messages arrive than can be immediately processed, they are stored in the queue and will be processed subsequently by the service according to a FIFO strategy.

The connections between services may take different forms. We use the same model as was proposed by [7]. Now we formulate the optimal control problem as reconfiguration of the system to be as near as possible to given desirable response time RT_{des} with minimum total processing cost of the system. We control the number of servers in each component at each moment of time.

The problem was solved by dynamic programming technique [8]. We have tested a simple example. We simulated our solution applied to two abstract services in sequence, with a Poisson distribution arrival rate. Both abstract services have (separate) sets of 6 equivalent services, each with different service rates and costs.

Table 2.1
Possible services for abstract service A.

Service chosen for A	**Service rate**	**Cost**
1	35	10
2	40	20
3	45	30
4	50	40
5	55	50
6	60	60

Table 2.2
Possible services for abstract service B.

Service chosen for B	**Service rate**	**Cost**
1	37	15
2	43	23
3	47	35
4	51	51
5	52	52
6	55	55

The arrival rate to the first component had Poisson distribution. The desirable response time was given as 0.4 sec. In general, the results showed us that it is optimal to use the first, second, and the fourth servers for the first component and the first server for the second component (Figure 2.6).

From Figure 2.7 we see that the total cost decreases. It can be explained by the fact that while a number of requests arrived to the first component was approaching to its maximum arrival rate, the Service Manager has used cheap servers. The servers were switched on the expensive ones only when a number requests became small, however, the response time was stayed on desirable level. We avoid using of heavy loaded servers which has a high cost when arrival rate of requests is extremely high and we allocate a small number of jobs to them if their quantity is low.

3. Conclusion. Using a Web portal has a number of well-enumerated advantages over specially designed multiprocessor systems such as: high performance workstations and PCs are readily available at low cost, the latest processors can easily be incorporated into the system as they become available, existing software can be used or modified. Grid infrastructure gives to end users interactive control, access to remote storage system at any time, and tool to solve a computational problem. The main idea of the given work was to build PSE implemented as a Web portal that allows clients to choose the most appropriate services to solve some problems with using of matrix-algebra and numerical methods based on MPI techniques. To provide desirable performance the Service Manager's component approach was developed. The main idea of Service Manager is to control the load of the system and satisfy to given requirements. To reach this goal dynamic programming technique was used. In addition, a parallel algorithm for two-dimensional non-stationary heat conductivity equation in rectangular area was included to the library of the Web-portal. In general, it is available to extend

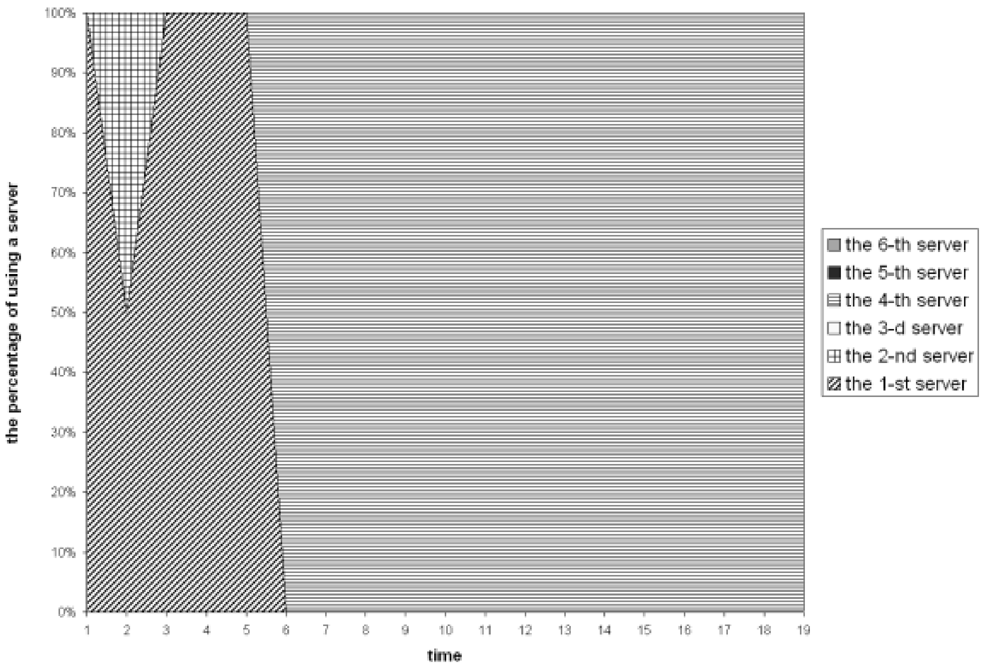


FIG. 2.6. *The percentage of using servers for the 1-st component.*

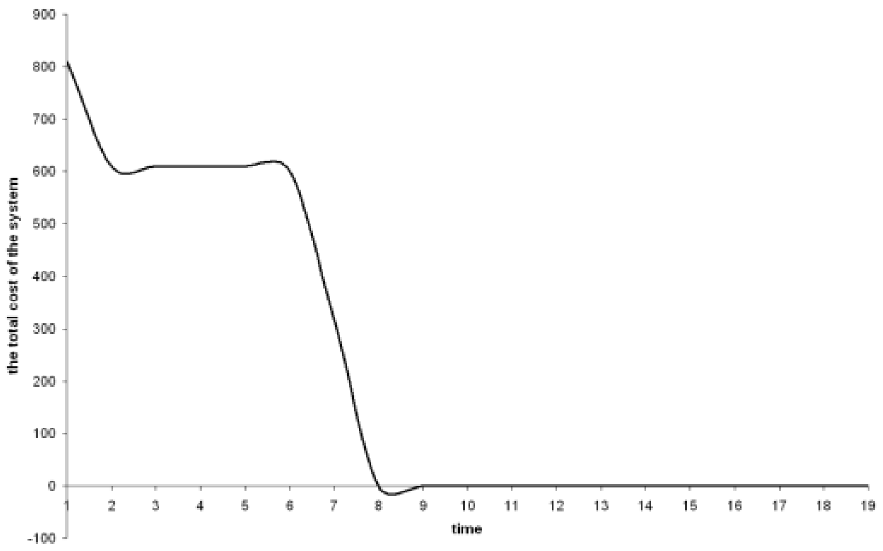


FIG. 2.7. *The total cost of the system.*

the library of system by uploading user's solutions. Thus, the proposed system allows the rapid prototyping of ideas, detailed analysis, and higher productivity.

In future, it is expected to make the proposed system self-adaptive by using of the Service Manager's component. In conclusion, the Web portal was created to solve different applied problems with use of MPI on the basis of the above described three-level platform. Any registered user can access the portal on-line with any Web-browser. Using of the proposed system will allow constructing the Internet services which would be able to solve vital applied problems.

REFERENCES

[1] Assel Zh. Akzhalova, Daniar Y. Aizhulov: Web portal to make large-scale scientific computations based on Grid computing and MPI, *Proceedings of PPAM 2007 Seventh International Conference on Parallel Processing and Applied Mathematics,* (LaSCoG-07) pp. 57–62.

[2] Jack Dongarra: Sourcebook of parallel computing. 2nd edn. Morgan Kaufmann Publishers, San Francisco (2003).

[3] A. C. Catlin, S. Weeravarana, E. N. Houstis, and M. Gaitatzes: The PELLPACK User Guide. Department of Computer Sciences, Purdue University, Lafayette, IN, 2000.

[4] Harry Jordan, Gita Alaghband: Fundamentals of parallel processing Pearson Education, Inc. New Jersey (2003).

[5] V. Wilkinson and M. Allen: Parallel programming: techniques and applications using networked workstations and parallel computers. Published by Prentice-Hall, Inc. (1999).

[6] S. A. Nemnjugin, O. L. Stesik: Parallel programming for high-efficiency multiprocessing systems. St.-Petersburg, (2002).

[7] Mahadevan Iyer, Wei Kang Tsai: Time-Optimal Network Queue Control: The Case of Multiple Congested Nodes, *Proceedings of 10th International Conference on Parallel and Distributed Systems (ICPADS'04),* pp. 709–718.

[8] P. Dmitri: Bertsecas Dynamic programming and optimal control. Athena Scientific, 2000.

Edited by: Dana Petcu, Marcin Paprzycki
Received: May 4, 2008
Accepted: May 24, 2008

Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 143–150. http://www.scpe.org
ISSN 1895-1767
© 2008 SCPE

PERFORMANCE MEASUREMENTS OF COMPUTING NETWORKS

IGOR ROZMAN[*], MARJAN ŠTERK[*], JAKA MOČNIK[†], BORUT ROBIČ[‡] AND ROMAN TROBEC[§]

Abstract. This paper presents measured communication and computational performance results on an ad-hoc computing network composed of several geographically distributed nodes. Communication speed was measured for three basic communication patterns commonly used in parallel and distributed applications: point-to-point, bidirectional ring, and all-to-all communication. For that purpose simple test programs were made, based on the MPI library. The parallel execution time of two real-life computationally intensive parallel applications is also presented. Results show that only the applications with low communication requirements could be beneficially executed on computing networks.

Key words: computing network, ad-hoc networks, grid, Internet, communication speed, MPI, performance evaluation

1. Introduction. Scientific and industrial applications require ever more computing capacity, and for the majority of institutions and enterprises it is hard to follow this trend by buying more and more powerful computers [1]. On the other hand, there is an increasing number of networked computers, which are only partially utilized. Grid technology has been developed and proposed in recent years [2], which provides the infrastructure required to harness the combined computing power of these machines for computationally intensive tasks at a considerably lower price than special-purpose supercomputers. Grid technology is a standardized solution for distributed computing on a larger number of geographically separated computers.

A Grid could represents a group of computers, which are originally installed for some other general tasks, like for office workstations, but with unexploited resources made available for executing Grid computing tasks. These computers are generally not located in the same place and can have very heterogeneous resources in terms of processor type, storage capacity, available network bandwidth, operating system type, etc. Also, their availability can change with time, depending on their primary use. A Grid differs from a computer cluster in the way of resource management. While centralized in a cluster, it is distributed in a Grid, and far more complicated with every node having its own resource manager. The user should not care about details of resource allocation. This should be done by the Grid middleware transparently to the user. Resources must first be discovered and their access and use granted, which includes user and computer authentication, according to assigned rights and priorities. Computational nodes must be assigned according to their processing and communication capacities. Communication between nodes is usually performed over public networks and must be made secure. Broad public usage of computational and storage resources would also require charging according to use in the way electric power, phone service, Internet access or other public services are charged nowadays [3, 4].

Users of the Grid computing power should not bother about where the computing is executed. Instead, a suitable Grid computing tool should provide for fast, secure and repeatable execution of computing tasks, enabling the use of computer resources available in the network. Grids, which are specialized for the execution of computationally intensive tasks, are called computational Grids. The most important resources here are processors, and such a network would consist of a large number of computers with large computational power or even of supercomputers and computer clusters. Tasks can run on a single computer, on a number of them, or they can even move between networked computers during their execution. However, it was experienced that the efficiency of computational Grids is not always satisfactory, particularly if the communication requirements are high.

It is known that only scalable algorithms can be implemented efficiently on massively parallel machines [5]. Typically, such algorithms need only a small amount of communication between processors [6]. By increasing the number of processors, the computational load on each processor decreases. There is a limit beyond which further increase in the number of processors is not profitable, because the computational load becomes too small when compared to the communication load. The latter also depends on the characteristic communication patterns of the algorithm, in particular, the amount of data to be transferred and the frequency of communication needed during calculation.

[*]Jozef Stefan Institute, Jamova 39, Ljubljana, Slovenia.
[†]XLAB d.o.o., Teslova 30, Ljubljana, Slovenia.
[‡]Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, Ljubljana, Slovenia.
[§]Jozef Stefan Institute, Jamova 39, Ljubljana, Slovenia (roman.trobec@ijs.si).

The goal of our work was to build an ad-hoc computing network, inspired by Grid technologies, and to investigate the impact of communication performance on computation performance. Nodes of our computing network are workstations and home computers at three different locations: Ljubljana, Salzburg and Zagreb. The communication speeds in different communication patterns were measured with simple test programs based on the MPI library [7, 8] in order to prevent the overhead of a Grid middleware [9]. Additionally, the computational performance is checked by running two real-life computation-intensive parallel applications.

The rest of the paper is organized as follows. In the next section, the testing methods and the tested ad-hoc computing network are described. In Section 3, the measured communication results for asynchronous point-to-point communication, bidirectional communication in a ring topology, and all-to-all communication are presented. Section 4 is devoted to the measured execution times of two real-life applications. The paper concludes with the summary of obtained results and with some directions for further work.

2. Testing environment.

2.1. Terminology. The *execution time* of a parallel application is the sum of *calculation time*, *idle time* and *communication time*. The calculation time is spent for data transfer between internal memory and CPU and their manipulation in the processing unit. The idle time is a period during which processing unit is available, but is not in use, for example due to the lack of data from other resources. The communication time consists of the time needed to transfer the required application messages, and is for each message the sum of the *set-up time* and the *transfer time*. The set-up time dominates for short messages, while the transfer time is more important by longer messages. The *bandwidth* is the message size divided by its communication time.

The following standard abbreviations are used: bit (b), byte (B), second (s), minutes (min), giga (G), mega (M) and kilo (k).

2.2. Measuring Methodology. Communication bandwidth was measured as *(message length/round-trip time)* for the following lengths of messages: 80 B, 320 B, 640 B, 16 kB and 64 kB. Each measurement was performed eight times, every 30 minutes, in a period of 24 hours. Two measurements with the lowest communication bandwidth were eliminated in order to exclude various system activities, which could have a significant impact on measured results. Therefore the shown communication bandwidth is given as an average of six measurements.

2.3. Computing Network. We set up a heterogeneous computing network composed of distributed nodes on the following distant locations:

- computing node at Jozef Stefan Institute (PC_IJS), located in Ljubljana,
- workstation at Jozef Stefan Institute (Zelda), located in Ljubljana,
- home computer (PC_Lj), located in Ljubljana,
- computing node at University of Salzburg (PC_Slz), located in Salzburg,
- workstation at Rudjer Boskovic Institute (PC_Zg), located in Zagreb.

The computer PC_IJS is locally connected to Zelda, via Fast Ethernet (LAN) and further to the wide area network (WAN). All remote nodes PC_Lj, PC_Slz and PC_Zg are also connected to the WAN (see Figure 2.1). Some technical details of our network nodes are given in Table 2.1.

The Grid infrastructural services (authentication, authorization, data transfer, job management etc.) on our network are provided by the Globus Toolkit [10], a widely used open source middleware for building Grid systems. However, all these services contribute to a substantial run-time overhead, therefore we chose to run our parallel test applications that use solely the MPI communication library MPICH v1.2.7 [11] with the ch_p4 device, which supports communication between workstations in heterogeneous TCP/IP networks. It enables practical, portable, efficient and simple message communication among processes. MPI is usually applied in parallel computers and clusters, if efficient resource management is important, while the safety issues are not imperative.

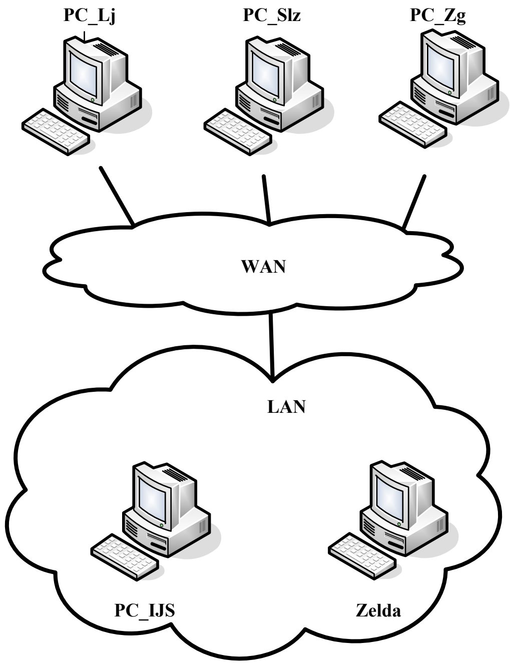


Fig. 2.1. *The topology of the tested ad-hoc computing network.*

Table 2.1
Hardware configuration of tested computing network nodes.

Computer	OS	CPU	RAM
PC_IJS	Fedora 6-2.6.19	Athlon XP 2000+	512 MB
PC_Lj	Fedora 6-2.6.19	Celeron 1700	512 MB
PC_Slz	Debian 3.3.5-2.6.9	Pentium 4 3.0	512 MB
Zelda	Fedora 2-2.6.10	Opteron 1.8	2 GB
PC_Zg	Debian 3.3.5-2.6.8	Xeon 2.8	2 GB

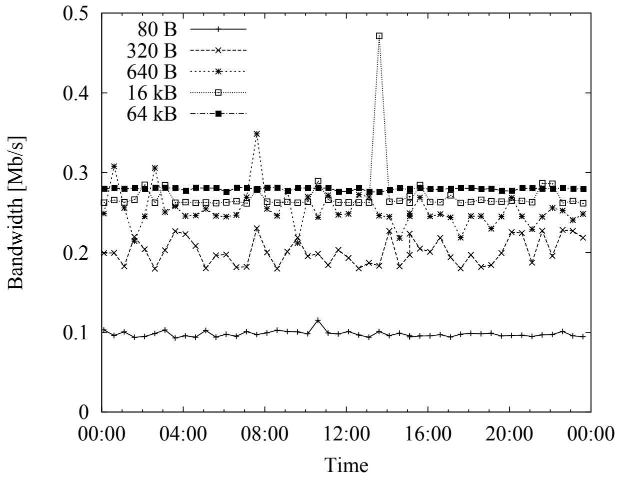
3. Results of Communication Performance.

3.1. Point-to-point Communication. The simplest communication action between a pair of processes, one side sending and the other receiving, is called point-to-point communication. Its bandwidth was measured with a test program, which uses asynchronous MPI functions `MPI_Isend` and `MPI_Irecv`. The program implements "ping-pong" by sending/receiving messages of various lengths between two nodes and measures the round-trip time.

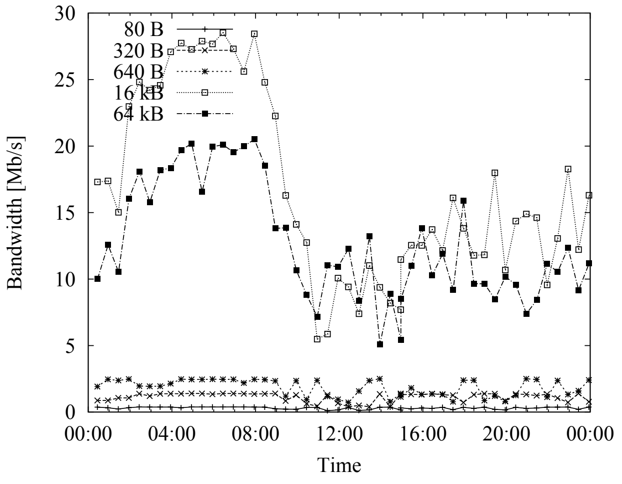
Measurements of asynchronous communication bandwidth between nodes of our test network were conducted in the period from March 13 to 21, 2007, using the methodology described in Subsection 2.2. The results are shown in Figure 3.1. Note that different scales are used on y-axes. On all graphs the bandwidth is low for short messages, which is due to the set-up time, and rises with the message length.

The slowest connection is between PC_IJS and PC_Lj (Figure 3.1(a)), because the node PC_Lj has limited upload speed of 0.25 Mb/s by Internet service provider. In average the highest bandwidth of 0.29 Mb/s is reached at 64 kB messages. In contrast, the highest bandwidth on most of other connections is reached at 16 kB messages. According to our previous experiments [12] this could be explained with the limited sizes of system and network buffers. Longer messages are fragmented into more packets, further, some of them can be lost and therefore resubmit, which could result in significant lowering of the communication bandwidth.

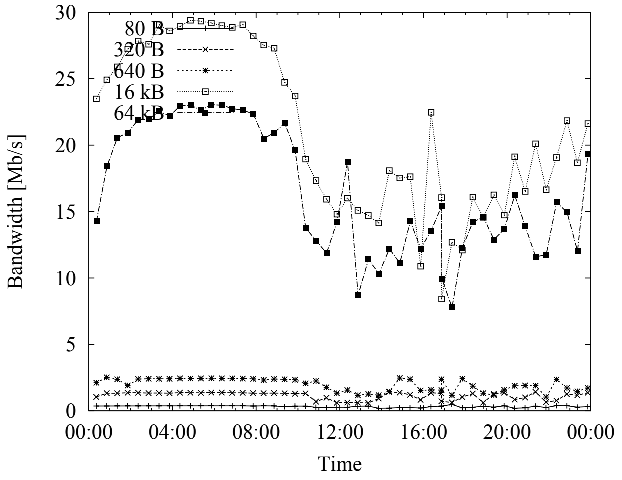
The communication bandwidth of the connection between Ljubljana and Zagreb (Figures 3.1(b), 3.1(c)) is the fastest among all Internet connections, however, it is very variable. It reaches the minimum at 5 Mb/s and the maximum at 28 Mb/s for 16 kB messages. Again, longest messages reached lower bandwidth as expected. Two measurements conducted in two different days show the same pattern of the bandwidth, which starts to



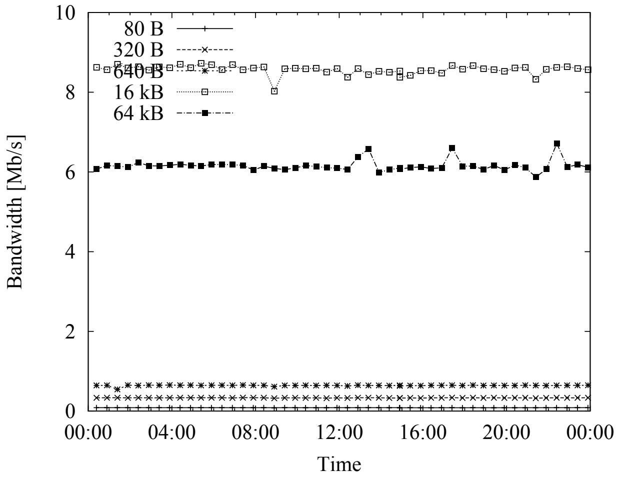
(a) PC_IJS and PC_Lj on March 13, 2007.



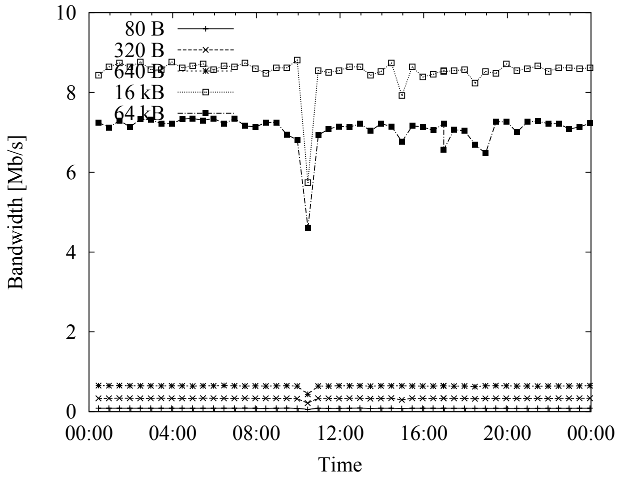
(b) PC_IJS and PC_Zg on March 13, 2007.



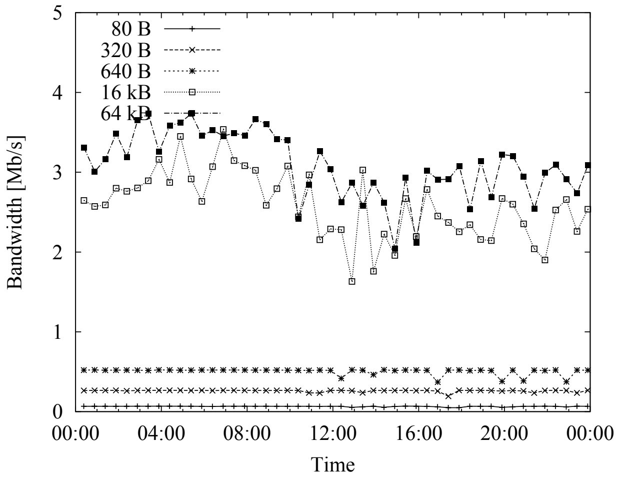
(c) Zelda and PC_Zg on March 15, 2007.



(d) PC_IJS and PC_Slz on March 13, 2007.



(e) Zelda and PC_Slz on March 15, 2007.



(f) PC_Zg and PC_Slz on March 21, 2007.

FIG. 3.1. *The bandwidth of asynchronous point-to-point communication between nodes of test network.*

increase after 1. a.m. and decrease after 8 a.m. The reason is probably the daily usage of Internet which also increases the load on the connection between Ljubljana and Zagreb.

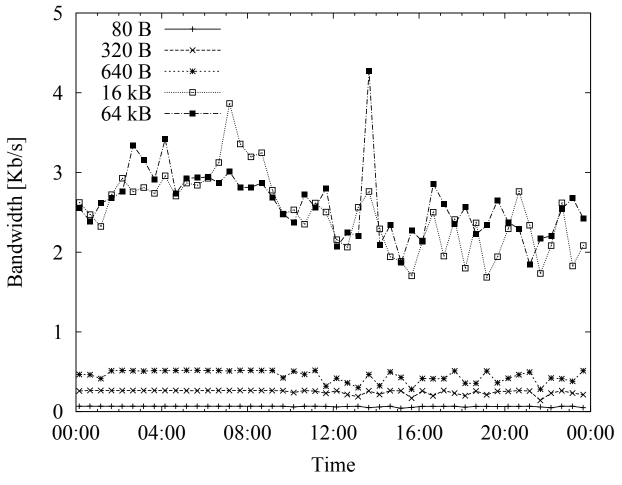
The connection between Ljubljana and PC_Slz has quite a constant bandwidth of 9 Mb/s in case of 16 kB messages (Figures 3.1(d), 3.1(e)) with no significant difference during the day and night period. The bandwidth of the connection between PC_Slz and PC_Zg (Figure 3.1(f)) is not as constant, with highest values in the interval between 2 Mb/s and 3.5 MB/s for 64 kB messages. Also, the bandwidth is a little higher between 1 a.m. to 8 a.m., similar to the previously described connections from Figures 3.1(b), 3.1(c).

We may conclude that there is no uniform pattern for communication bandwidth because it is limited by physical connection, daily usage, or by limitations posed from connection providers.

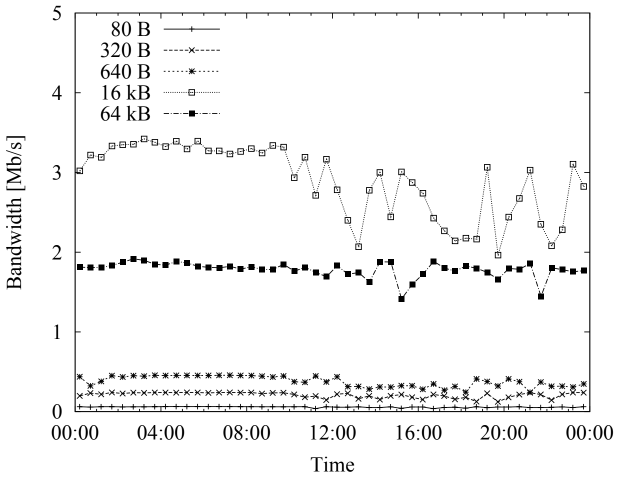
3.2. Bidirectional Communication in Ring. In parallel numerical simulations, the problem domain is often decomposed in subdomains, e.g., slices in 1-D decomposition, which are distributed among processors [13]. In every simulation step, an exchange of boundary values between neighboring processors is needed. In the case of a linear array or ring topology, each node thus sends the same amount of data to its left and right neighbor, and also receives the same amount from both, all at the same time. Ring topologies are vulnerable to node and link failures. A single crashed process or communication channel makes it impossible for some other processes to communicate data further along the ring. Thus, the ring topology and algorithms based on it are not suitable for the Internet environment.

The test program shifts data with messages of the same length applying non-blocking MPI functions `MPI_Isend` and `MPI_Irecv`. The bandwidth is now the length of a single message divided by the time needed for message exchange with left and right neighbors, i. e. two sent and two received messages in each node of the ring.

The performance of bidirectional communication among PC_IJS, PC_Slz, PC_Zg and Zelda in the ring topology is shown in Figure 3.2(a). From the point-to-point measurements we know that the connection between PC_Lj and WAN is more than ten times slower than other connections, therefore we eliminated it from the further measurements of bidirectional and all-to-all communications. The communication bandwidth remains limited, again with by the slowest link, which is in this case the link between Zagreb and Salzburg.



(a) Bidirectional communication in ring.



(b) All-to-all communication.

FIG. 3.2. *Collective communication among PC_IJS, PC_Slz, PC_Zg and Zelda on March 21, 2007.*

3.3. All-to-all Communication. In all-to-all communication, each node sends a distinct message of size m to every other node. Many parallel applications, for example, molecular dynamics [14] are based on such a collective communication. All-to-all communication test was implemented by the MPI function `MPI_Allgather`, which, on all processes, gathers data from all other processes and distributes local data to all other processes. The bandwidth represents the length of a single message divided by the time needed for execution of the function `MPI_Allgather`.

The performance of all-to-all communication for PC_IJS, PC_Slz, PC_Zg and Zelda is shown in Figure 3.2(b). The communication is limited by the slowest link (Zagreb-Salzburg). Because of relatively low bandwidth we did not detect any problems that could be attributed to switch or router congestion, which would slow down the collective communication speed in clusters.

4. Results of Computational Performance. The execution times of two applications were measured to evaluate computational performance: simulation of a human knee cooled after surgery by gel packs [15], and searching for the optimal solution to the Rubik cube [16].

4.1. Description of Applications. In the *knee cooling simulation*, heat transfer is modeled by a linear diffusion equation and solved by the finite difference method and explicit time stepping. This implies a relatively small amount of calculation and an exchange of boundary values for each time-step. The problem domain was 3-D and composed of several million points with substantial amount of communication.

The parallel version of the simulation was implemented using 1-D domain decomposition—domain slicing [17]. Load balancing is easily achieved in a cluster architecture with equal nodes and communication speed. The calculation time is inversely proportional to the number of processors, while the communication time remains constant. It can be expected that the speedup will behave as a monotonously increasing function of number of processors, soon approaching some constant value limited by the communication time. Performance results in a cluster environment confirmed these expectations and were already presented in [12]. However, the parallel execution time on a computing network will be saturated much earlier because of slow communication links.

The *Rubik cube solver* attempts to solve (order) one initial setup of the Rubik cube in an optimal (minimal) number of moves. To this end, it uses a highly-optimized IDA* (Iterative Deepening A*) depth-first search using a memory-based heuristics based on partial solutions of the Rubik cube [18, 16]. The algorithm searches the problem space which is represented by a search tree with the initial cube setup at the root. Each parent-child link in the tree represents one (valid) move during solving the Rubik cube. Each move is a rotation of one layer of the cube. At the first level (link stemming from the root), all 18 different moves are possible, giving the root a branching factor of 18. At the lower levels, some links are pruned from the tree *a priori* as they represent redundant moves that do not lead to an optimal solution: as an example, consider the move that reverses the previous one, which is clearly redundant. Therefore, an average number of children (branching factor) for nodes at all the lower levels is 13.3 [16].

Note that simple depth-first search does not find the optimal solution, while full A* search is not appropriate because of its exponential memory requirements. Iterative deepening, on the other hand, first executes a depth-first search with maximum depth limited to 1, followed by 2 etc, ensuring the optimal solution is found. A typical Rubik cube can be solved, with the described algorithm, in 16-20 moves and thus finding the optimal solution requires a search over about 10^9–10^{14} nodes. In our test we use a single initial setup that is known to have an optimal solution consisting of 16 moves and the times to discover that it can not be solved in 15 moves is measured.

The parallel implementation of the solver is a simple master-slave type of computation. The master node first searches the top two levels of the search tree and then distributes the next 243 (as the average branching factor on the second level is 13.5, see above) subtrees of the problem space to the slave machines. Each slave executes the job, i. e. searches the subtree to the current maximum depth, and either returns the solution or replies that no solution can be found up to the current maximum depth. The communication in each IDA* iteration consists solely of distributing the jobs to the slaves and collecting the results.

4.2. Measured Execution Time. Both applications were run first on each single node. Except on the node PC_Slz, there was enough internal memory to prevent swapping with the hard disk. Then the applications were run in parallel on various sets of computing nodes. Difficulties appeared when we tried to use the node Zelda with other nodes. We found out that the application programs do not strictly use complex MPI data types which results in incompatibilities between nodes. Zelda runs on 64-bit operating system while all other nodes run on 32-bit operating systems. Consequently, the node Zelda was not a part of measurements in our computing network. The execution time of both test applications is presented in Table 4.1. Speed-up is not calculated because of heterogeneity in computation and communication performances.

From the separate runs of the *simulation of knee cooling* on each node we see that the execution time was quite similar, with the shortest on the node Zelda and the longest on the node PC_Zg. The node PC_Slz reported “Out of memory”.

Table 4.1
Execution time of the knee cooling simulation for 1 s of the real time, and of the Rubik cube solution.

Computing nodes	Execution time Knee cooling	Execution time Rubik cube
Zelda	11.6 s	4.63 min
PC_IJS	12.2 s	9.08 min
PC_Zg	13.8 s	6.55 min
PC_Slz	Out of memory	4.48 min
PC_Lj	17.0 s	13.5 min
Zelda (use of both CPUs)	6.4 s	2.55 min
PC_IJS, PC_Zg	11.0 s	3.92 min
PC_IJS, PC_Zg and PC_Slz	22.9 s	2.27 min
PC_IJS, PC_Zg, PC_Slz, PC_Lj	223.4 s	1.97 min

Next the simulation was run on Zelda with the use of both CPUs. The execution time was almost halved compared to a single CPU, as expected. This is a consequence of efficient usage of two processors concurrently, and fast communication speed of internal bus between them, which did not slow down the execution time. Then we ran the simulation on nodes PC_Zg and PC_IJS, because they are connected with the fastest connection as we know from our previous measurements of communication performance. The execution time was about 20% shorter compared to the average execution time of both nodes, if they run as single nodes. Further, the simulation was run on the nodes: PC_IJS, PC_Zg and PC_Slz. The execution time was longer than on the slowest single node. This happened because the communication bandwidth between PC_Slz and other nodes is too low too support the required data transfer. The same phenomena is visible even more if the node PCLj is joined to the computing pool. The execution time becomes excessively long as a consequence of increased communication time of PC_Lj.

Data decomposition was always even for all nodes, which works well if the nodes have similar computation and communication ability as, e.g. on computing clusters. This was evidently not the case in our last experiment, therefore some method for load balancing has to be implemented to achieve more efficient execution on computing networks with substantial heterogeneity in node performance [19, 20, 21].

The results show that the *Rubik cube solver* scales well with the number of nodes, regardless of either their communication bandwidth or their computational capabilities for two reasons.

The first reason is that the execution time to communication time ratio is very high. This is first due to the fact that both input and output of each job are in the order of magnitude of 100 B with the tested implementation. Then, due to the master-slave paradigm and a small number of jobs, only 243 master to slave round-trips are required. This behavior results in only minimal impact of the communication bandwidth.

The second reason for good scaling is an adaptive load distribution strategy that first distributes one job to each slave node, and assigns a new job to any slave only after receiving the results of its previous job. In this manner, the faster slaves process more jobs, and the impact of heterogeneous nodes with different computational capabilities is much lesser when compared to the knee cooling simulation where a fixed load distribution proves inappropriate. Once again, due to very small size of the input, the fact that we are not prestaging input data to slaves does not lessen the performance.

We can conclude that such applications, characterized with high execution to communication ratio and with the capability of adaptive load distribution, prove more suited to network computing.

5. Conclusions. The obtained results of communication bandwidth in computing network show that it is impossible to predict in advance, and for a longer period, expected communication bandwidth of a particular connection. Some slow connections can appear, which represent communication bottlenecks. Therefore, for efficient distributed computationally intensive applications some additional mechanism is needed for on-line monitoring of the available communication resources.

Grids and other computing networks are generally composed of heterogeneous computers not tailored for intensive communication and prone to frequent faults. Any communication or computer failure can deteriorate the whole application if implemented without fault-tolerance measures. Computing networks are limited to simple topologies because each node has typically just a single network interface.

In parallel applications, which require many short messages, very distant nodes can represent a serious drawback, because of extra time needed for messages to reach these distant nodes, even if travelling with the speed of light. Travelling time can be much larger than the node set-up time, however, it must be added to the node set-up time.

The current results obtained and the mentioned drawbacks of computing networks indicate that there will always be a need to distinguish between distributed and parallel systems. For most scientific applications, except those with almost no communication, parallel systems will be preferred as they have balanced computation and communication speed. Computational networks or Grids need on-line monitoring of computation and communication resources and some mechanism for load balancing in order to be useful for parallel applications. In further work, we will focus on required approaches needed to increase the competitiveness of computing networks.

Acknowledgments. The authors acknowledge financial support from the state budget by the Slovenian Research Agency under grant P2-0095, and would also like to thank University of Salzburg and Rudjer Boskovic Institute for their support in setting-up our testing environment.

REFERENCES

[1] World Community Grid on WEB, `http://www.worldcommunitygrid.org/index.jsp`

[2] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, 1999.

[3] C. S. Yeo, M. D. de Assuncao, J. Yu, A. Sulistio, S. Venugopal, M. Placek and R. Buyya, *Utility Computing and Global Grids*, Technical Report, University of Melbourne, GRIDS-TR-2006-7, 2006.

[4] M. A. Rappa, *The utility business model and the future of computing services,* IBM Systems Journal 43 (2004), pp. 32–42.

[5] G. Golub and J. M. Ortega, *Scientific Computing, an Introduction with Parallel Computing*, Academic Press Inc., San Diego, 1993.

[6] V. Avbelj, M. Šterk and R. Trobec, *Lead Selection in Body Surface Electrocardiography by Exhaustive Search Optimisation*, in *Parallel Numerics '02 theory and applications*, Jozef Stefan Institute and Salzburg University, Ljubljana, 2002, pp. 201–207.

[7] W. Gropp, E. Lusk and A. Skjellum *Using MPI: Portable Parallel Programming with the Message-passing Interface*, MITPress, 1999.

[8] MPI on WEB, `http://www-unix.mcs.anl.gov/mpi/`

[9] R. Prodan and T. Fahringer, *Overhead analysis of scientific workflows in grid environments*, IEEE Transactions on Parallel and Distributed Systems 19 (2008), pp. 378-393.

[10] Globus Alliance on WEB: `http://www.globus.org/`

[11] MPICH on WEB, `http://www-unix.mcs.anl.gov/mpi/mpich/`

[12] I. Rozman, M. Šterk, R. Trobec, *Communication performance of LAM/MPI and MPICH on a Linux cluster*, Parallel Process. Lett., 16 (2006), pp. 323–334.

[13] M. Šterk, R. Trobec, *Biomedical simulation of heat transfer in a human heart*, J. chem. inf. mod., 45 (2005), pp. 1558–1563.

[14] U. Borštnik, M. Hodošček, D. Janežič, *Improving the performance of molecular dynamics simulations on parallel clusters*, J. chem. inf. comput. sci., 44 (2004), pp. 359–364.

[15] R. Trobec, M. Šterk, S. AlMawed, M. Veselko, *Computer Simulation Of Topical Knee Cooling*, Computes in Biology and Medicine, accepted and under revision.

[16] R. Korf, *Finding Optimal Solutions to Rubik's Cube Using Pattern Databases* in *Proceedings of the Workshop on Computer Games (W31), IJCAI'97*, Nagoya Japan, 1997, pp. 21–26.

[17] S. G. Akl, *Parallel Computation: Models and Methods*, Prentice Hall, 1995.

[18] R. Korf, *Artificial Intelligence Search Algorithms* in *Algorithms and Theory of Computation Handbook*, CRC Press, 1999.

[19] C. H. Hsu, T. L. Chen, K. C. Li, *Performance effective pre-scheduling strategy for heterogeneous grid systems in the master slave paradigm*, Future Generation Computer Systems, 23 (2007), pp. 569–579.

[20] J. Močnik, R. Trobec, B. Robič, *Integration of load balancing in a CORBA environment*, Parallel Algorithms and Applications, 18 (2003), pp. 99–105.

[21] K. Power and J. P. Morrison, *Ad Hoc Metacomputing with Compeer*, Scalable Computing: Practice and Experience 6, no. 1, (2005), pp. 17–32.

Edited by: Dana Petcu, Marcin Paprzycki
Received: April 6, 2008
Accepted: May 20, 2008

Scalable Computing: Practice and Experience
Volume 9, Number 2, pp. 151–152. http://www.scpe.org
ISSN 1895-1767
© 2008 SCPE

BOOK REWIES

EDITED BY SHAHRAM RAHIMI

The π-Calculus: A theory of mobile processes
by Davide Sangiorgi and David Walker

Formal methods have formed the foundation of Computer Science since its inception. Although, initially these formal methods dealt with processes and systems on an individual basis, the paradigm has shifted with the dawn of the age of computer networks. When dealing with systems with interconnected, communicating, dependent, cooperative, and competitive components, the older outlook of analyzing and developing singular systems—and the tools that went with it—were hardly suitable. This led to the development of theories and tools that would support the new paradigm. On the tools end, the development has been widespread and satisfactory: programming languages, development frameworks, databases, and even end-user software products such as word processors, have gained network-awareness. However on the theoretical end, the development has been far less satisfactory. The major work was done by Robin Milner, Joachim Parrow, and David Walker who developed the formalism known as π-calculus in 1989. π-calculus is a process calculus that treats communication between its components as the basic form of computation. It has been quite successful as a foundation of several other calculi in the field and as Milner puts it, "it has become common to express ideas about interactions and mobility in variants of the calculus."

1. Introduction. The current book serves as a comprehensive reference to π-calculus. Besides Milner's own book on the subject, this is the only other book-length publication on the topic. In many ways, it is much more comprehensive than Milner's: a much wider area of topics are dealt with and in more detail as well.

2. Contents. The book is split into seven part. The first part presents the basic theory of π-calculus. However, "basic" does not mean concise: every topic is discussed in great detail. The section on bisimulation is particularly intensive and provides several insights about the motivation for the theory. Part two discusses several variants of the original calculus. By varying several characteristics of the calculus, such as whether a process can communicate with more than processes at a time, we can obtain these variants. A number of interesting properties of the language are studied by the other when discussing these variants. As can be understood from the title, the calculus' contribution to analyzing mobile processes is a major topic, and it is dealt with extensively starting from part three. The basics are introduced by the way of a sophisticated *typing* system whose application in speciying complex processes is presented in part four. Part five looks at higher-order π-calculus in which composed systems are considered as first-class citizens. Part six is one of the best in the book and discusses the relation between π-calculus and lambda-calculus, which is an older and more basic calculus. Finally part seven shows how π-calculus can be employed in studying practical, modern software engineering concepts such as object-oriented programming.

3. Impressions. One of my disappointments with this book is in how often the reader is left perplexed with some of the theoretical developments, specially in part three. π-calculus is a complicated topic, even for the graduate student audience to which this book is directed, and the author would have done much better by reducing the number of topics and instead focusing on more lucid and detailed explanations. There are several experimental digressions throughout the book, which although interesting, take away from some of the momentum of sequential study. For example, topics such as comparison and encoding of one language to another could be easily moved to a separate section in order to make the content more suitable for self-study. Another issue is the little effort towards making the connection from the theoretical to the practical. The main reason why formal methods have not been adopted in mainstream software development pracitces is that often it is unclear to developers how formalisms can contribute towards the software engineering process. The book would have served its purpose much better if it had spent part of eah chapter discussing the practical application of that chapter's content. For example, congruence checking and bisimulation can be incredbily exciting topics for programmers to learn if they can see practical applications of such powerful techniques.

Beyond the above criticism, the book is absolutely indispensible to students and researchers in the field of formal methods. Currently it serves as the primary reference for anyone who wishes to learn the various aspects of π-calculus in detail.

Raheel Ahmad

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.