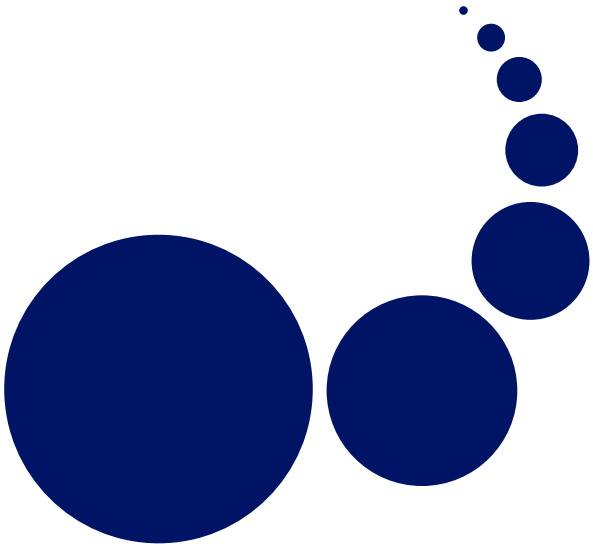
SCALABLE COMPUTING
Practice and Experience

Special Issue: New Directions in Cloud and Grid Computing

Editors: Dana Petcu and Marcin Paprzycki



Volume 12, Number 2, June 2011

ISSN 1895-1767

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

Scalable Computing: Practice and Experience

Volume 12, Number 2, June 2011

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 164–177. http://www.scpe.org
ISSN 1895-1767

USING SERVICE LEVEL AGREEMENTS IN A HIGH-PERFORMANCE COMPUTING ENVIRONMENT

ROLAND KÜBERT *AND STEFAN WESNER †

Abstract. The concept of Service Level Agreements (SLAs) has come to attention particularly in conjunction with Grid computing. SLAs allow for a controlled collaboration between partners, that is service providers and their customers. SLAs have originated in the telecommunication industry but have found broad uptake in the Grid computing community, especially regarding the topics of their automated negotiation, general management and legal implications. High-performance computing (HPC) providers, however, have not been taking up SLAs yet, even though they often provide not only supercomputer resources but at the same time access to cluster or grid resources. This might be due to the fact that SLAs are mostly regarded on a per-job basis, which is not consistent with the contractual model that HPC providers currently use. In this work, we analyze how HPC providers can implement SLAs and what benefits this brings, proposing the usage of SLAs on a long-term basis. We further present a software that has been developed to simulate the scheduling of jobs at an HPC provider site using service levels and analyze how different distributions of service levels between the submitted jobs influence machine usage and average waiting times of jobs. Finally, we present how an HPC provider can practically implement SLA-based scheduling.

1. Introduction. For High Performance Computing resources scheduling of jobs is still realized in most cases using simple batch queues. While batch queues like OpenPBS [2], TORQUE [5] or others offer a quite comprehensive set of functionality for placing jobs in appropriate queues and optimizing the load of the cluster systems, also across sites, there is no mapping from business level requirements down to the low-level specifications. Low-level specifications are typical elements of a job description, for example desired number of CPUs, maximum wall- or run-time. The provision of such low-level properties requires a high level of expertise of the user and can only be specified if the target platform is predetermined as different node and CPU architectures require different values. Additionally the number of queues is limited and therefore requirements have to be mapped to a particular queue. While advanced reservation, allowing a predefined start time, can be specified the drawback of potentially significantly reduced efficiency due to fragmentation of the schedule is not mapped to potential business penalties such as dynamically adapted pricing for such requests depending on the concrete loss in efficiency.

If an HPC provider wants to offer its services as utilities and aims to map different possible flavors of the services on different queue structures, the following problems can occur:

- Typically the use of certain queues is mapped to Unix credentials and groups. So all users of a certain group can or cannot use e.g. the express queue. However, depending on time of day or load situations, the "express queue service" might not be available to the same group of users all the time.
- While queues for specialized nodes (for example with graphics processing units (GPUs) or high memory nodes) are underutilized and normal nodes are oversubscribed there is no way to allow clients and providers to agree on special "discounts" for them. An automatic movement from normal to premium node queues would require interaction with accounting services.
- Customers might want to differentiate quite fine-grained about the treatment of their jobs. In such cases nowadays manual movement of jobs within the queues to "prioritize" them might be agreed beyond existing queue structures and group memberships. Such manual interactions cannot scale.
- Not all elements describing the Quality of Service (QoS) or Quality of Experience (QoE) can be mapped on queue properties and parameters. The overall service covers a wider range of properties such as the availability of a certain compute environment, application versions and licenses, proper treatment of data or specific configurations of the cluster system such as "require logical partition to isolate from other users".

This limitation that only a few functional parameters can be specified when submitting a job (also reflected in standards like the Job Service Description Language (JSDL) [1]) means that there is basically no way for the user to express his requirements on a QoS or QoE level. As an HPC provider is offering a service to users that is potentially replaceable with services from other providers, it is necessary to bridge the gaps between complex demands from the user side and simple services that are currently offered in order to ensure continued service usage.

*High Performance Computing Center Stuttgart, University of Stuttgart, Stuttgart, Germany, E-mail: kuebert@hlrs.de
†High Performance Computing Center Stuttgart, University of Stuttgart, Stuttgart, Germany, E-mail: wesner@hlrs.de

As a result a significant amount of work has been spent on realizing SLA frameworks allowing to mutually agree on the terms of the service between provider and consumer. However, while these frameworks cover well the necessary steps to realize SLAs also as a legally binding agreement, the concrete content of such an SLA and more important how these terms can be guaranteed and provided from the service provider side are not adequately addressed.

So far we have the possibility for the consumer to express the requirements and agree the terms with the provider but

- terms within the SLA are not on the desired business level but mimic the low-level properties of the underlying queuing systems and
- the agreement process is typically detached from the underlying infrastructure such as current load situation of different resources, priority and importance of the consumer in a Customer Relationship Management (CRM) system and the accounting and billing services.

Consequently there is a gap between the demand of defining business level SLAs and their implementation using available methods and tools for the management of them on different type of computing facilities ranging from commodity of the shelf (COTS) clusters over specialized compute systems to cloud computing and storage systems.

Management systems on the provider side between the SLAs agreed with the consumer and the concrete physical resources need to interact with a range of different elements within the providers IT infrastructure and must look beyond individual SLAs to optimize the overall operation of all resources within a HPC computing service provider.

This work is structured as follows: section 2 analyzes related work, section 3 examines business models for the provisioning of HPC services, section 4 elaborates on benefits for HPC providers when implementing SLAs and section 5 describes the advantage of long-term SLAs against typically used short-lived, per-job contrast. Section 6 transfers the theoretical work from the previous section to a more practical setting by analyzing the impact of SLAs on resource usage and job waiting time and section 7 addresses the issue of implementing SLA management in an HPC environment. Finally, section 8 gives and outlook on how the previous findings can be transferred from HPC to cloud services and section 9 concludes the work.

2. Related work. There are various approaches to the usage of service level agreements for job scheduling. While they differ in many respects - detail of the presentation, assumed parameters, implementation level, etc. - they all share the fact that they treat SLAs as agreements on a per-job basis. That means that, for each job to be submitted, a unique SLA is established before the job can be submitted; an exception to this is the work of Kübert and Wesner, who propose to use service level agreements as long-term contracts [17]. In [31], Yarmolenko et al., after having identified the fact that SLA-based scheduling is not researched as intensively as it could be, investigate the influence of different heuristics on the scheduling of parallel jobs. SLAs are identified as a means to provide more flexibility, increase resource utilization and to fulfill a larger number of user requests. Parameters either influence timing (earliest job start time, latest job finish time, job execution time, number of CPU nodes) or pricing. They present a theoretical analysis of scheduling heuristics and how they are influenced by SLA parameters and do not investigate how the heuristics might be integrated into an already existing setup. The same authors identify in [23] the need to provide greater flexibility in service levels offered by high-performance, parallel, supercomputing resources. In this work they present an abstract architecture for job scheduling on the grid and come to the conclusion that new algorithms are necessary for efficient scheduling in order to satisfy SLA terms but that little research has been published in this area. MacLaren et al. come to a similar conclusion, stating that SLAs are necessary in an architecture supporting efficient job scheduling [20].

SLAs that express a job's deadline as central parameter for deadline-constrained job admission control have been investigated by Yeo and Buyya [32]. The main findings were that these SLAs depend strongly on accurate run time estimates, but that it is difficult to obtain good run time estimates from job traces.

Djemame et al. present a way of using SLAs for risk assessment and management, thereby increasing the reliability of grids [7]. The proposed solution is discussed in the scope of three use cases: a single-job scenario, a workflow scenario with a broker that is only active at negotiation-time and a workflow scenario with a broker that is responsible at run-time. It is claimed that risk assessment leads to fewer SLA violations, thus increasing profit, and to increased trust into grid technology.

Dumitrescu et al. have explored a specific type of SLAs, usage SLAs, for scheduling of grid-specific workloads using the bio informatics BLAST tool with the GRUBER scheduling framework [8]. Usage SLAs are

characterized by four parameters: a user's VO and group membership, required processor time and required disk space. The work analyzes how suitable different scheduling algorithm are. Additionally, it comes to the conclusion that there is a need for using good grid resource management tools, which should be easy to maintain and to deploy.

Sandholm describes how a grid, specifically the accounting-driven Swedish national grid, can be made aware of SLAs [24]. It is presented how the architecture can be extended with SLAs and it is stated the greatest benefit would be achieved by insisting on formally signed agreements.

A comprehensive overview of resource management systems and the application of SLAs for resource management and scheduling is given by Seidel et al. [26]. The connection of service level and resource management to local schedulers is clearly shown as a gap in nearly all solutions.

In summary, it can be said that isolated aspects of the usage of SLAs have partly been investigated in detail: scheduling algorithms and heuristics, abstract architectures, parameters which are to be used as service levels, SLA negotiation etc. Gaps, however, can be easily identified: the analysis of the "big picture", that is the composition of individual aspects of SLA usage into a complete system and the integration of SLAs and SLA management with local resource management. This is not only true for the "traditional" field of high performance and grid computing but can also be extended to the field of cloud computing. Furthermore, SLAs are solely treated on a per-job basis, the analysis of SLAs as long-term contracts is not covered.

3. HPC Service Provisioning Business Models. The increased importance of modeling and simulation in many academic disciplines and similarly in industry over the last years is one of the drivers leading to an increased diversity of the user community for High Performance Computing service providers. Another element is the hardware development towards multi- and manycore computing systems or GPGPUs making parallelism a commodity at any desktop. As a result the user community of a typical HPC service provider ranges from entry level users that have just started parallel programming up to high-end users able to use very large computing systems with many hundred teraFLOPS up to the petaFLOP scale.

Another dimension for classifying users is their typical workflow in using the HPC resources. Depending on the research discipline the demand for computing resources can be rather constant or may vary quite significantly over time. A user with a constant demand is interested in a service that delivers a guaranteed level of capacity of the computing system and costs that are lower as for a self-operated computing system. A user with a largely varying computing demand e.g. because he is relying on experimental data demands for an elastic computing system. Additionally all users change their use profile over time e.g. during development cycles of their simulation software more short runs are requested whereas during production phases long running jobs are the major use case.

Beside different job types in terms of scale or duration the expected HPC offer is also determined by the use case. Users performing open research & development typically operates based on grants with a duration of several months and years. The condition associated with this kind of access that is typically without costs is to publish all results obtained at the end of the grant period. The conditions of this kind of offer are typically defined by the provider for a large class of users and not negotiated bi-laterally. For example there could be a defined profile for university users, federal project users and large scale research projects detailing the service level, which resources can be accessed and what kind of applications and tools are available. If two different users receive a grant of the same type from the options above, their access conditions are completely identical.

Another offer type is a production level offer with bi-lateral agreed conditions and obligations defined in a legally binding contract and negotiated on a case by case basis. Such an offer is typically applied for commercial users that are willing to accept higher costs for computing services in order to receive special access conditions, user tailored environments (e.g. a dedicated file system or queue). Obviously a wide spectrum of offers in-between these two different extremes are possible such as special agreed conditions for a group of users.

3.1. Provider defined SLAs. Provider defined SLAs or service offers use the available hardware resources as the starting point for the offer definition. The typical model applied by many HPC service providers is to use a reduced amount of production hours due to maintenance or loss in utilization in the scheduling system (for example due to very large computing jobs) per year e.g. 5000-6000 hours instead of the theoretically possible 8760 hours ($365 * 24$). Additional resources such as application licenses or special hardware are considered similarly.

Based on the available resources and preallocated shares of a system (e.g. 30% for European Research projects) for the different system parts and the intended use model of the system an appropriate set of queues

are defined for the system. The queue design allows to control the major purpose of the system as capacity or capability machine, if large scale debugging or scalability test sessions are possible (queues for jobs of large size for short job duration and reasonable waiting time) or if large jobs should be prioritized. Additionally a set of constraints are defined on how certain external servers supporting the use of the big computing system should be used. For example that compilation of software should only be done on login servers, analysis of result data on dedicated post-processing servers.

Based on the definition of the environment that is offered to the users very different models are still possible controlling fair access to the computing system. One model deployed widely is to operate the resource in a competitive manner allowing all users to place their jobs in the different queues and schedule jobs driven by queue and user priority. Such a model has potentially the problem of overbooking and undetermined waiting times in queues but realize a quite high level of utilization. The quality of the service experienced by the users will vary and depends on the arrival process of competing jobs in the queues by other users.

Alternative approaches are to offer dedicated access to a share of the machine for a certain user group or project for a a given time period (several months or even a year) with more certainty in terms of waiting time or predictability of job launch but with significantly reduced utilization of the overall system.

The major characteristic of provider defined SLAs is the resource driven viewpoint aiming to define a very small set of offers for different user groups. Furthermore it is assumed that user demands are (1) well understood by the HPC provider, (2) are defined by the community or project the user belongs to and (3) the demand is not depending on the project phase or type of work currently done. These kind of SLAs are not negotiated but are policies or use models that need to be signed up by the users during the application procedure for accessing the resources. These conditions apply for all compute jobs submitted during the whole project lifetime.

3.2. User negotiated SLAs. User or user group negotiated SLAs start from the particular demand of the research or commercial project. These demands expressed on the user domain level reflecting the workflow and processes applied by the users need to be mapped on the concretely available resources at the provider side. As such user tailored SLAs are as of now implemented by manually changing the configuration of the HPC provider environment such offers are limited to a small set of customers.

If part of such a user negotiated SLA are specific guarantees or dedicated access to a decent fraction of the system the utilization of the system is reduced and is in contradiction to the major goal of the HPC service provider to achieve the highest possible level of utilization. Consequently such special offers must be associated with an appropriately increased price reflecting the loss in efficiency. As an example consider the demand to start compute jobs spanning across 50% of the overall system with a guaranteed start time of less than one hour. This means that upon submission of such a job under this SLA all jobs running longer than one hour blocking the part of the system to be used must be terminated and re-started after the large job. The already used computing time until the last checkpoint is lost and needs to be added to the costs of the large job.

As the loss of such a model can be quite significant and if too many user negotiated access offers have been agreed the implementation might be impossible (or the risk of violating the SLA would become too high) such offers are typically limited to a small fraction of the system in the range of 10-20%.

4. Benefits of SLAs for HPC service provisioning. The current operation model for high-end computing resources is conceptually still the same as fifty years ago where users placed a set of punching cards at the registry desk. The only difference is that users now can submit their compute jobs to a set of different queues and instead of the human operator the scheduling system is picking the jobs from the different queues depending on defined policies aiming for an optimized load of the system partially reaching 99% utilization. The major shortcoming of this approach is that the optimization strategy defined by the queues and the scheduling system policies is oriented towards a global optimization rather than an individual service offer.

If a user needs a special service (e.g. guaranteed start time of a job during a demonstration, interactive visualization or exhibition) beyond regular job submission the negotiation is typically done directly with the system operator and the performed steps are mostly done manually.

The availability of multi-core CPUs will lead to compute nodes with 32 cores and more in the near future, the rise of GPU-based computing with several hundred “cores” per card allows a reasonable number of applications to run on a single node. This is particularly true if the application is not targeting for a high-end simulation e.g. in the area of Computational Fluid Dynamics (CFD) domain with a very fine-grained mesh but more on exploring the problem space. Other examples are cases where the full simulation has been done before and now only small changes in geometry are done interactively demanding much less intensive computing to reach

a stable state again as it is based on the previously achieved results.

Driven by the availability of cloud service providers and emerging products such as the Amazon Cluster Compute Instances also high-end computing service providers change their offers to be more *elastic* and realize a more *dynamically changing* infrastructure having certain queues available only during specific time periods or realizing a dynamic allocation of resources to logical partitions depending on the load situations or specific time bound agreements.

The predominant use of high-end computing services will continue to be highly scalable technical simulations demanding a large number of compute nodes for exclusive use. However additional use cases have emerged driven by changes on the hardware level and competition with cloud service providers in particular for small scale simulations. The exclusive access for a user to one single node might even for compute intensive applications become a relic of the past. This substantially more complex management model for HPC service providers that cannot rely anymore on a quite homogeneous user behavior and long running jobs demands for a more complex management solution for operating their resources. The challenge is to integrate the demands of policies from different levels such as business policies (e.g. users with highly scalable and long running jobs should experience a preferred treatment) with more short term policies reacting on the current load situation (e.g. reducing prices or accepting more small jobs to fill gaps in the current schedule) and the demand of the users on a per-job basis.

The following sections aim to cover in examples the three major use cases driving the need for an SLA-guaranteed HPC service provision. Abstracting from concrete cases three different cases can be identified:

4.1. Interactive Validation. In many areas simulations have already replaced real experiments or physical prototypes during the development process. However at certain control points in the process simulation results have to be verified using physical prototypes. Within the IRMOS project augmented reality techniques are used to overlay real experimental data like a smoke train in the wind channel with a visualization of trace lines from the corresponding simulation. This "hybrid" prototype allows experts to directly compare the behavior of the real prototype with the results of the simulation. Such a design review session typically involving several people of a development team spread around the globe demands a fixed availability of the wind-tunnel, the computing resources, the visualization resources, the corresponding network resources and all involved experts, for example via video-conferencing.

In such a scenario simulation data will be generated continuously by a simulation running on a compute resource that is directly connected to the visualization resources. The current configuration of the wind channel like the air speed will be communicated as boundary condition for the simulation, thus the same parameters for both will be used while the experiment is running. This requires a coordinated and automated provision of the resources involved in the overall setting.

Such a scenario cannot rely on batch queue-based access as the computing and simulation part is just one piece in the overall setting. The demand for a co-ordinated availability also opens questions on how penalties are applied if one of the pieces in the overall setting is failing. For example if the compute resources are not provided as promised in time and the wind tunnel cannot be used the costs for it still accumulate. This applies also the other way around if the wind tunnel is not available or fails to communicate the boundary conditions for the simulations or the network connection is not delivering sufficient bandwidth.

As the resources needed for the full scenario are provided by different organizational entities the different quality levels needed by each individual contributor need to be put in a formal SLA, covering the terms of service as well as the agreed penalties in case of failures.

4.2. Guaranteed Environment. As outlined in [30] beside quality constraints there is also a demand to ensure a certain environment or other procedural constraints such as data handling, security policies or environmental properties (version of the operating system, available Independent Software Vendor (ISV) applications, etc.).

This is especially necessary for simulations performed as part of an overall design cycle for a complex product such as a car or airplane. A software environment is frozen for a full development cycle in order to ensure reproducible simulation results. This fixed environment is typically ranging from operating system over certain versions of numerical libraries up to application codes. A typical approach to address this requirement is to have beside a paper-based SLA agreed for a design cycle period a dedicated computing resource with the requested environment.

Advances in virtualization technologies as well as the possibility to apply different boot images in diskless cluster environments allow a more flexible treatment. Using such technologies a potentially unlimited number

of predefined images, or even user-defined images, might be provided. As not all environments can be provided on all compute resources there must be a negotiation process between the user and the provider where a certain environment is demanded (e.g. expressed in a certain SLA bundle such as "Silver") and a corresponding reply about the conditions for the different options from the provider side is delivered.

The increased flexibility would allow to offer customized environments not only to large customers asking for resources for a long time period but also for users looking to meet their peak demands with outsourcing avoiding tedious customization activities of the environment reducing the entry gap.

4.3. Real-time Constraint Simulations. With the increasing role of simulations in design processes for complex products the demand to have a time-boxed simulation where results need to be delivered in time have emerged. This might be a set of simulations exploring a parameter space as input for a meeting of engineers the other day deciding on the focus for the future (long running simulation jobs). Another possibility is if the results of one single simulation (or a set of simultaneously running simulations) is the input to support an expert in taking a decision.

One important application area demanding for such an operation model is individualized patient treatment. For example in [25] a scenario for using simulations to validate different options to perform a bone implant for a specific patients is presented. In such cases the expert that needs to make a treatment decision has to ensure in advance of starting the simulation at a specific compute service provider that the results will be available in time before the treatment must be executed.

In such a scenario a negotiation with several providers would be started in parallel in order to make a case-by-case decision to which provider the job will be finally submitted. Such a loose binding to a specific provider would also require similarly to the scenario in the previous section a guaranteed or user-provided environment making the different providers interchangeable.

4.4. SLA Service Provision Benefits. From all the scenarios above it becomes clear that a much higher diversity of the offered services must be expected in the future. The requirements of the different scenarios on the provider's infrastructure are quite diverging. Additionally the consumer requirements are contradictory to the goal of the providers reaching a very high level of utilization of the provided resources.

As a result service providers will need to

- offer a mix of different services in order to combine the benefit of best-effort services (high utilization) with the benefit of special services (high value and price),
- offer a framework allowing consumers and providers to agree on the specific conditions for the service and
- actively manage their resources in a way that agreed SLAs are met, resources are most effectively used and any failures and incidents on the resource level are managed to avoid any impact on the agreed service levels.

The underpinning assumption presented in this section is that the provision of SLA controlled services is beneficial for consumers *and* providers. Consumer can negotiate guarantees and specific properties of the provided services as needed enabling new use models for high-end computing resources as outlined above. The provider perspective is clearly driven by business benefits to deliver as a part of the differentiation strategy specific products rather than aiming for a cost leadership approach. Consequently there is a clear need from the consumer side as well as a clear motivation from the provider side to deliver also in the HPC domain SLA based services. In other words the current model where the user needs to fully adapt to the provided environment and access model is changed to a model where the provider is offering certain possibilities or a kind of toolbox where the consumer can arrange the service offer according to their needs. Realistically this space of options needs to be discrete and limited allowing a management of the service offer from the provider side.

5. Using long-term service level agreements for job control. Service level agreements, when they are used for the scheduling of compute jobs, are normally assumed to be on a per-job basis. That means that an individual SLA only contains terms for one specific job and a new SLA needs to be established for each job (see for example [7], [31] and [32]). This may be ideal to investigate the influence of SLAs and parameters specified therein on the scheduling of jobs in an isolated environment but does not correspond with the reality of how contracts are handled at HPC providers. At HPC providers, users usually agree to a contract that specifies charges for computational times and storage for available machines [22]. Jobs are then submitted in accordance with the acknowledged charges which therefore can be thought of as a long-term contract. This contract is, however, missing a specification of service levels. There may be some service level-related parameters specified

- for example the availability of different machines to users and their characteristics, for example CPUs per compute node and memory size per node, but these are only specified in order to compute the amount finally billed to the user. By adding service levels to this contract, a long-term service level agreement is formed.

Long-term SLAs add the missing specification of service levels but keep the familiar contract behavior used by HPC providers intact. A simple specification of priorities, for example, might be realized through the following service levels:

Bronze Computational time is cheap, but there is no assurance on the scheduling of a job. This corresponds to the best-effort services provided today at HPC centers.

Silver Moderate prizing for computing time due to prioritized scheduling. Silver jobs can have timing guarantees and might preempt best-effort jobs. The increased prize is justified since guarantees on the job's scheduling are given.

Gold High-prized jobs that are only rarely used, for example for urgent computing when computations need to be started immediately.

In contrast to the current situation the possibility of providing different service levels allows users to potentially have multiple contracts in place in parallel. On job submission time, a user decides which contract to reference in the submission depending on the current requirements and conditions such as urgency of the simulation result, load situation of the provider(s) etc. This can be seen as a using the middle way between using SLAs on dynamic, per-job basis and solely having singular long-term contracts. As opposed to dynamic, per-job SLAs, this approach reduces the amount of negotiation as only few contracts are in place. Additionally, it avoids the problem that an urgent job cannot be submitted due to a failed negotiation. This approach is more flexible than having only singular contracts and allows users to choose necessary priorities depending on the prize they like to pay.

6. Simulating the scheduling of service level agreements. In the previous sections, we have elaborated on the benefits of the usage of SLAs for both the service provider, who can offer a much more diverse portfolio, and the customer, who can make use of this portfolio and will have certain guarantees assured from the service provider. As contracts that are established for typical HPC providers at the moment are of a long-term nature, it is suitable for the SLA-related contractual information to be of long-term nature as well. This immediately poses a straightforward questions: how can service providers know how the offering of service levels will work out? To put it another way, service providers, who are obliging themselves to various guarantees and may even be penalized if these are not met can not just switch from a best-effort scheduling approach to an implementation using service levels and hope that everything will "just work". Even though the provider is free to formulate service levels and to enter these with customers as it sees fit, the results are not foreseeable.

The provision of service levels in high performance computing is a novel approach, therefore the provider cannot rely on already existing data. Even though data exists that contains real-world workload traces and even models of these exist[1], these traces and models do not include service level data, so another way has to be found. This way needs to provide answers to different questions, for example:

- What distributions of service levels lead to a good quota of fulfillment, or, what service levels can be supported without the provider being liable to paying huge penalties?
- What different target functions can be applied and what does each one entail?
- How do different scheduling policies influence the fulfillment of contracts?
- What is the result on the provider's infrastructure, for example how much is a cluster used?

The right way to answer these questions in a way that does not put the provider in an experimental situation without its outcomes - possibly huge penalties - being foreseeable is to simulate the different service levels the provider envisions. The arrival, scheduling and computation of jobs at an HPC provider's site can be easily simulated with a discrete-event simulation, which represents a system as a chronological series of events.

Jobs arrive at a certain, discrete point in time and are scheduled. Time is advanced and the jobs are finally run on certain resources until they are finished, at which point in time they leave the system. All this can be modeled easily with different events, for example job arrival, computation start and computation end. Many tools for generic discrete-event simulations exist, for example SimJava [11], Tortuga [13] or SimPy [27], as well as tools for specialized applications, for example the GridSim Toolkit for the simulation of scheduling for parallel and distributed computing [28].

[1] The Parallel Workloads Archive at `http://www.cs.huji.ac.il/labs/parallel/workload/` is a good source for both of these.

The addition of service levels into these simulations doesn't change the inherent nature of the problem - a discrete-event simulation can still be used to simulate scheduling taking service levels into account. There is, however no readily extensible tool that can be used for the simulation of different, possibly new scheduling algorithms, especially not in conjunction with service levels. In order to not duplicate work that has already been solved in satisfactory way - however, a very common feature [14] - a simulation program or toolkit that is close to the required functionality should be used and extended. Alea [15] is a job scheduling simulator that is based on the popular GridSim toolkit; its basic functionalities are fulfilling the requirements, but Alea is not extensible, can generally not be readily used and does not provide any support for service levels. As an extension of the GridSim toolkit [28] , Alea's own code base is, however, more easily manageable. This led the authors to the decision to analyze Alea and extend it with the ability to simulate service levels. This was supported by the fact that Alea already supports different scheduling algorithms - as previously mentioned, unfortunately not related to service levels - can read different workload traces (for example from the above mentioned Parallel Workloads Archive) and even provides a simple visualization of results.

Alea works by reading different data files for workload traces, hardware resource description and machine failures. While the specification of failure data is optional, workload traces and resource descriptions are of course mandatory. Alea is implemented in an object-oriented approach and each entity read from a data file is a corresponding class instance. The communication between these classes is performed by a central scheduler component which is divided into a communication and a scheduling part. In order to enrich jobs specified in workload traces, an additional data file that specifies the service level for each job has been developed. This data file is read by a custom loader class, similar to the ones for workload traces and resource descriptions and the service level information is added to the job object. Figure 6.1 shows Alea's architecture with added components and communications represented with dashed lines.

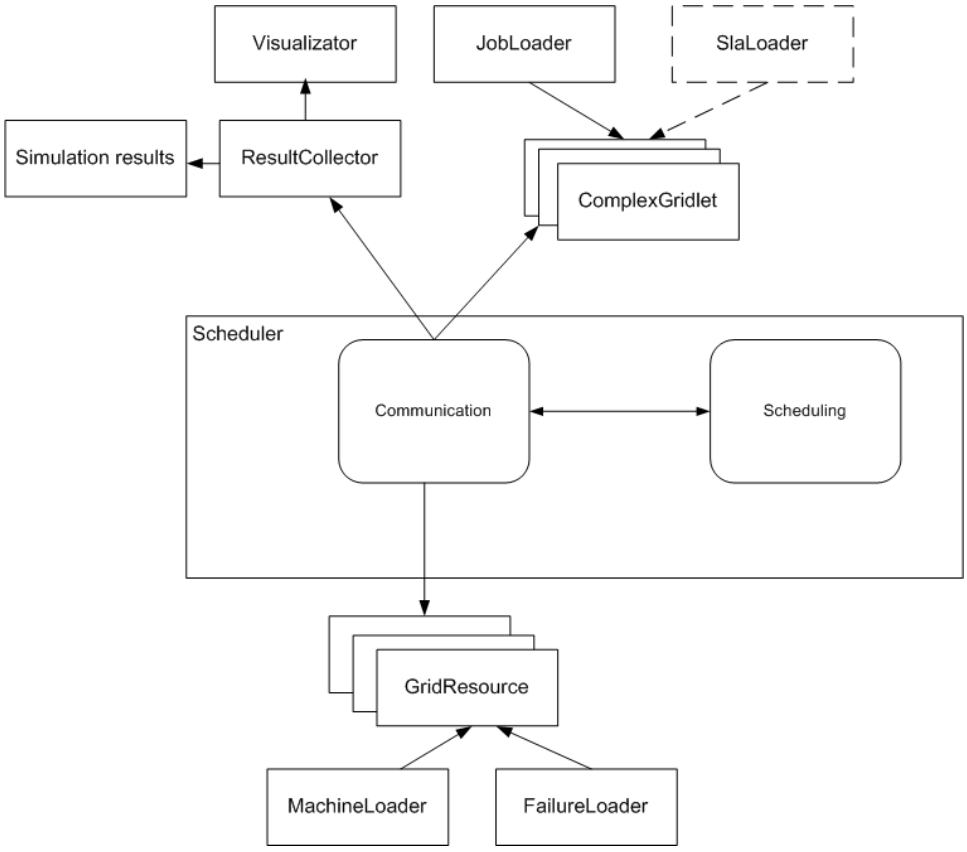


Fig. 6.1: The Alea architecture with the added SLA loader [15]

The workload formats that Alea uses are more or less standardized, so the decision was taken to not

change the format. Instead, a special file mapping jobs to service levels has been created. This file is only read if service level-scheduling is used, other algorithms which Alea supports are not touched. The scheduling supporting service levels is currently using a queue-based approach (Alea supports schedule-based approaches as well). Each newly arriving job is sorted into the queue according to its priority using a special *Comparator*[2].

An initial implementation of the work has already brought tangible results that can guide service providers to rough estimations on what distribution of service levels can be offered [16]. The initial simulation was performed using three different service levels - gold, silver and bronze which corresponded to urgent computing, prioritized scheduling and best-effort and it was investigated in how far different distributions of these service levels influence the average waiting time of jobs - in each service level and in total - and the machine usage.

Due to the workload distribution itself, the machine usage did not change. This is, however, not universal, as it is simple to construct example cases where the machine usage is changed. This is especially true if additional service parameters apart from pure prioritization are used, for example exclusive access to resources. Scheduling a job for exclusive use of a resource will then reduce the total machine usage if the job running exclusively will not use as many resources as would be consumed at the same time using non-exclusive access. The machine usage is most likely not the foremost factor to be considered when offering service levels, as it is of more importance to satisfy the service levels and only then can other parameters be taken into consideration.

An interesting question is: given a possible distribution of service levels, could the provider give any absolute guarantees ("X% of jobs submitted in the silver service level will wait less than y minutes") or is this impossible and the only guarantees possible are relative ("gold service level jobs are scheduled before silver service level jobs, which are scheduled before bronze service level jobs")? In order to answer this question, we have investigated different distributions of the three service levels and have looked at how each distribution changes the average waiting time of jobs and, as well, the average waiting times of jobs in each service level. The base case, best-effort scheduling, can be seen as 100% of jobs in the bronze service level.

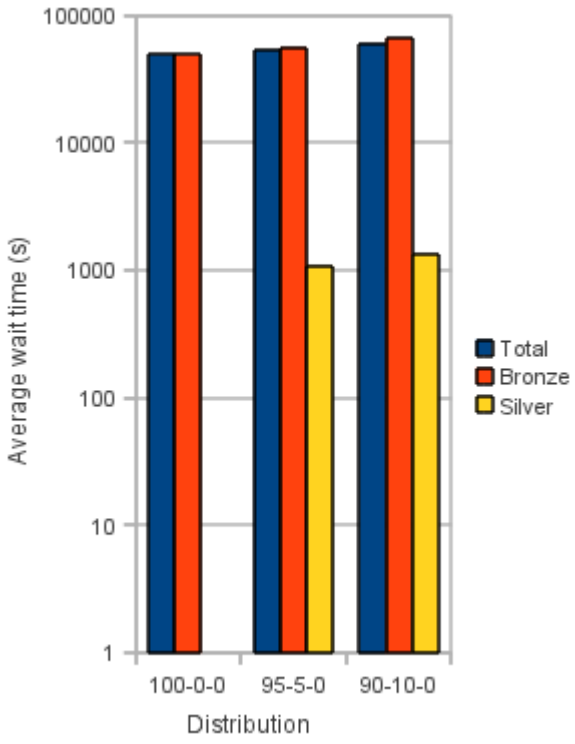


Fig. 6.2: Distribution of waiting times with only silver and bronze service levels

[2]See `http://download.oracle.com/javase/6/docs/api/java/util/Comparator.html`.

Figure 6.2 shows the base case on the left and two additional distributions of only the silver service level, first with 95% of jobs in the bronze service level and 5% in the silver service level (middle), then with 90% of jobs in the bronze service level and 10% in the silver service level (right). The average waiting time increases with rising number of silver jobs, of course, as bronze jobs are potentially queued longer; the same is true for the bronze level itself, but with a higher increase in the average waiting time – silver jobs have a short waiting time and therefore reduce the average case. Between 5% and 10% there is only very small increase in waiting time. The provider can therefore give guarantees that with a high probability will not be breached; he would, however, have to limit the amount of silver jobs a single user can have in the system as otherwise it is trivial to force the provider into a situation where it breaches SLAs, for example by submitting a high number of silver jobs. Even though the increase in waiting time is very small for the silver service level, the provider will not want to increase the number of silver level jobs too much, as the increase in waiting time for the best-effort bronze level will then reach a point which makes the provider unattractive to customers only using the bronze service level.

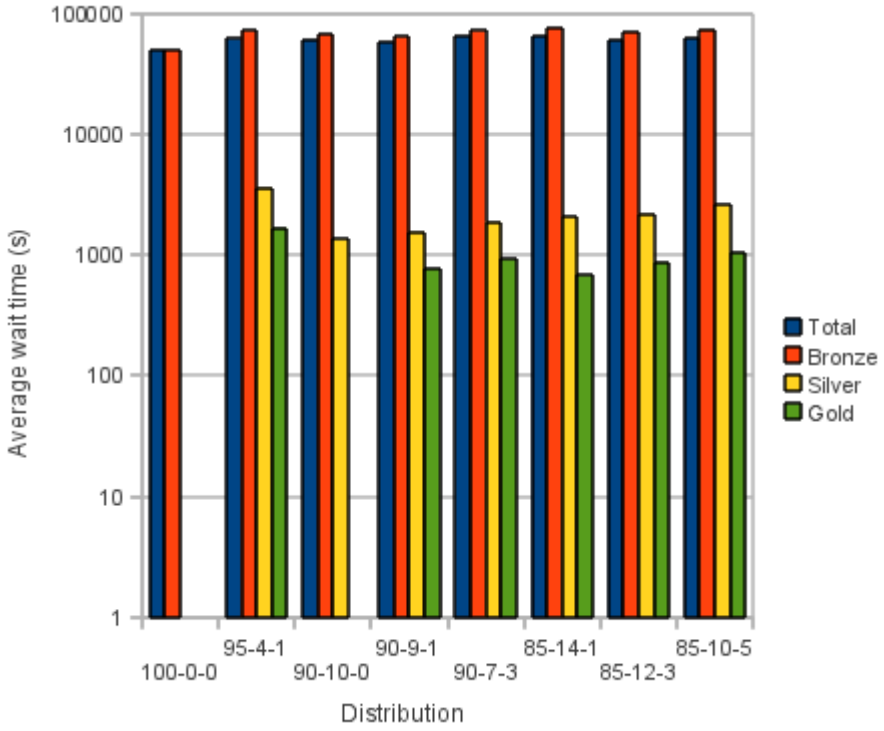


Fig. 6.3: Distribution of waiting times with gold, silver and bronze service levels

Figure 6.3 shows different distributions of service levels, taking all service levels (gold, silver and bronze) into account. The trend that can be seen here is similar to the one shown before: the average waiting increases moderately while the waiting time for the bronze service levels has a much bigger increase. Additionally, for a constant percentage of bronze jobs, the waiting time for both silver and gold jobs increases with increasing number of gold jobs. Once again the results hint at guarantees the provider might give: the average waiting time for gold jobs rises only twice above 1,000s, the average waiting time for silver jobs only twice about 2,500s.

As these results are simulated with fixed workload traces, a provider cannot be 100% sure these findings are transferable directly to an implementation of service-level based scheduling. Taking more complex considerations into account can, however, align the simulation closer to reality. Findings after an implementation - for example the number of jobs that users who have the possibility to submit to different service levels submit to each service level - can help refine the simulation with user behavior. Nonetheless, the simulation shows that a provider can at least simulate rough impacts of different distributions of service levels.

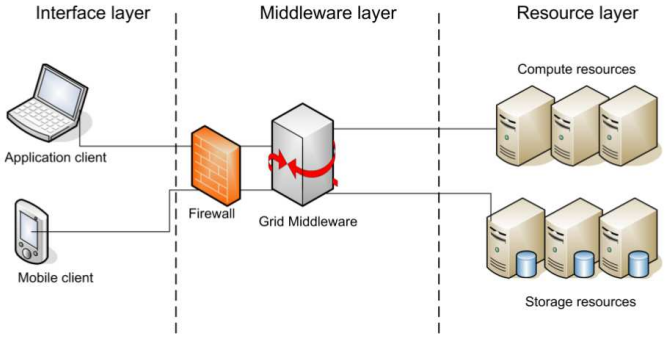


Fig. 7.1: Layered architecture

7. An integrated approach to service level management. The basic service level approach given above can be realized with current techniques, for example the usage of specialized priority queues; however, providing more complex service levels cannot be realized that easily but require the integration of service level management techniques across interface, middleware and resource layer.

Figure 7.1 shows the typical three-layered setup that is used by HPC providers. The left-hand sides shows clients of the HPC provider, either static or mobile[3]. The middleware layer is positioned between the client and the low-level resources and serves as a central entry point to the HPC provider's system and provides access through grid middlewares, for example the Globus Toolkit. The Grid middleware takes jobs submitted by the client and passes them on to low-level resources by means of a resource manager which employs a job scheduler in order to determine which jobs are placed on which resources.

7.1. Service level selection by the client. Enhancing the client with the ability to select service levels is very straightforward. This can be either done by changing the job submission client, adding SLA information to the message sent to the middleware or by integrating SLA functionalities into the application, if job submission is performed directly out of it.

7.2. Enhancing the middleware. SLA management on a middleware level has been investigated by various research projects and therefore different components and solutions already exist; the NextGRID project [21] aimed at providing a generic solution in the broader realm of a full architecture for next-generation grids while other projects focused on solutions for specific problems - the FinGrid project [9] on the financial industry, the IRMOS project [12] on the applicability of SLAs to the execution of real-time aware applications on Service-Oriented Architectures (SOAs) and the BEinGRID project [4] on a solution that can be adapted to different business cases. As these solutions often have the drawback of being very complex, a simpler solution is preferable, as it eases the amount of work necessary for installation, integration and maintenance.

Figure 7.2 is a diagram depicting how easily SLA management can be implemented on a middleware level and the underlying resource layer. The client thereby can either communicate with the SLA Manager, a central component on the HPC provider side responsible for SLA management, or submit jobs to the cluster front-end in the manner already explained.

The SLA Manager provides data regarding the long-term SLA contracts, for example contract information, accounting pertaining to contracts etc. It uses an internal SLA Repository for storing the contracts and other relevant information and is the central point that is queried by other components regarding SLAs. The cluster front-end, for example, on submission of a job, can query the SLA Manager for the validity of SLAs and can, after job completion, send accounting data to the SLA Manager.

The SLA functionality for the cluster front-end in the grid middleware can be realized in a non-intrusive way, for example through a policy decision point (PDP) that checks incoming requests and their SLA specification for validity. Incoming requests that do not contain SLA specifications can be mapped internally to a default SLA

[3]Mobile in this context should not be mixed with cellular phones and is understood as a nomadic user that is connecting from different locations without predefined IP addresses.

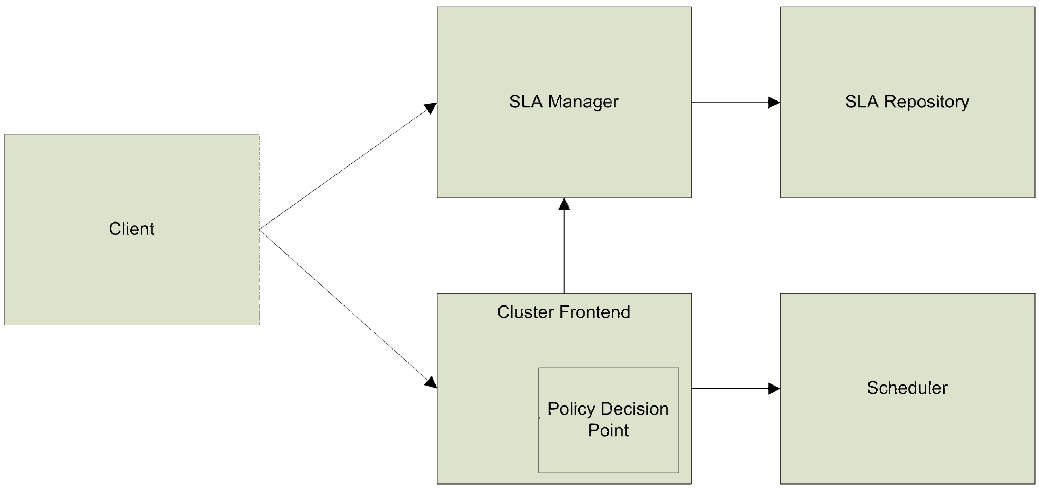


Fig. 7.2: High-level SLA management components

specifying a best-effort style service level, thereby realizing complete SLA-functionality and being backwards-compatible to clients.

7.3. Acknowledgement of SLAs. Honoring service levels of submitted jobs depends on the software used on this low level. Very simple schedulers, like the default scheduler supplied with the TORQUE resource manager, cannot honor service levels and need to be replaced by an SLA-enabled scheduler. This can be implemented by the provider itself, but this is a time-consuming and error-prone job. Rather, an SLA-enabled scheduler, for example the Moab Cluster Suite, should be used. It allows the formulation of quality of service levels for resource access, priority and accounting.

The providers main task is then to express the high-level SLAs offered to customers in such a way that the scheduler can implement them on the resource layer. Additionally, the incoming job requests have to be mapped to the corresponding service levels.

8. From high-performance to cloud computing. In the previous sections we have elaborated a concept to enhance "classical" high-performance computing with service levels through the use of long-term service level agreements. Cloud computing, in many terms similar to the previous scenarios, seems like a logical step for the provisioning of services and can be a sensible offer to provide for HPC providers besides their usual role. Even though the term cloud computing is not clearly defined, it can be seen as distributed computing with virtual machines. Virtualization allows for more flexibility, scalability and abstraction of the underlying resources. Accounting and billing are usage-dependent [3].

Cloud computing brings benefits both for consumers and providers. Virtualization allows the provider to use free resources for the execution for virtual machines as the underlying hardware is mostly irrelevant, although requirements specified by the user of course still need to be met. The usage of virtual machines means that the provider can offer a multitude of different environments tailored to customers, which was previously infeasible. Users might even be allowed to provide their own virtual machines, therefore giving them control over the complete environment.

Cloud computing began on a best-effort basis and many solutions provided today don't offer any more service [18] [29]. Service level agreements for cloud computing are, however, provided by some service providers, but they provide only minimal service levels [6] [19].

It has been shown that both Infrastructure as a Service and Platform as a Service - two types of cloud computing where the first one offers the concept described above and the second one offers a scalable, flexible but predefined environment to users - can benefit from service level agreements as well [10].

9. Conclusions. The preceding work has described an integrated approach to using service level agreements for the control of compute jobs which allows HPC providers to offer support for various quality of service levels. The approach was motivated through a discussion of business models for service provisioning in an HPC

environment and it has been presented how the usage of SLAs can be simulated before implementing it in a production infrastructure. Due to the proposed solution including both high-level SLA management and low-level resource management and job scheduling, service providers can take advantage of service level agreements through their complete infrastructure. We shortly concluded in how far this can be transferred to the recently introduced cloud computing paradigm.

A proof of concept implementation of the above mentioned HPC scenario using three distinct service levels has been implemented, providing overarching service level support from the resource layer to the middleware layer. Following the trend to provide Infrastructure as a Service (IaaS) solutions, we are currently investigating how the general concept can be realized with the Gridway meta-scheduler which is compatible with the OpenNebula cloud toolkit and therefore would allow for the realization of an SLA-based cloud offering. This enables HPC providers to offer IaaS with distinct service levels, which is not possible at the moment.

In general, the offering of service levels can be a distinctive advantage for HPC providers as current contracts normally do not foresee the provision of service levels. Customers gain flexibility by having the possibility to choose between different service levels when submitting jobs. This also allows providers the option of offering previously unsupported service models, for example for urgent computing, which can generate a new revenue stream.

REFERENCES

[1] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva, *Job submission description language (jsdl) specification, version 1.0*. `http://forge.gridforum.org/sf/go/doc12582?nav=1`. [Online, accessed 8-March-2010].

[2] Argonne National Laboratories, *OpenPBS Public Home*. http://www.mcs.anl.gov/research/projects/openpbs/.

[3] C. Baun, M. Kunze, J. Nimis, and S. Tai, *Web-basierte dynamische it-services*, (2009).

[4] BEinGRID Consortium, *BEinGRID project home page*. `http://beingrid/`, 2008.

[5] Cluster Resources Inc., *TORQUE Resource Manager*. http://www.clusterresources.com/products/torque-resource-manager.php.

[6] M. Corporation, *Download details: Windows Azure Compute SLA document*. `http://go.microsoft.com/fwlink/?LinkId=159704`, 2010. [Online, accessed 2-March-2010].

[7] K. Djemame, I. Gourlay, J. Padgett, G. Birkenheuer, M. Hovestadt, O. Kao, and K. Voss, *Introducing risk management into the grid*, in e-Science, IEEE Computer Society, 2006, p. 28.

[8] C. L. Dumitrescu, I. Raicu, and I. Foster, *Usage sla-based scheduling in grids: Research articles*, Concurr. Comput. : Pract. Exper., 19 (2007), pp. 945–963.

[9] FinGrid Consortium, *FinGrid project home page*. `http://141.2.67.69/`, 2008.

[10] G. Gallizo, R. Kuebert, K. Oberle, A. Menychtas, and K. Konstanteli, *Service level agreements in virtualised service platforms*, in eChallenges 2009, Istanbul, Turkey, 2009.

[11] F. Howell and R. Mcnab, *simjava: A discrete event simulation library for java*, in In International Conference on Web-Based Modeling and Simulation, 1998, pp. 51–56.

[12] IRMOS Consortium, *IRMOS project home page*. `http://irmos-project.eu/`, 2008.

[13] James Barkley et al., *tortugades – Tortuga is a software framework for discrete-event simulation in Java*. `http://code.google.com/p/tortugades/`.

[14] R. Katz and T. J. Allen, *Investigating the not invented here (nih) syndrome: A look at the performance, tenure, and communication patterns of 50 r & d project groups*, R&D Management, 12 (1982), pp. 7–20.

[15] D. Klusáček and H. Rudová, *Alea 2 – job scheduling simulator*, in Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), ICST, 2010.

[16] R. Kübert, *Providing quality of service through service level agreements in a high-performance computing environment*, in PARENG 2011, 2-nd Int. Conf. on Parallel, Distributed, Grid and Cloud Computing for Engineering, Ajaccio, France, Apr. 2011, Civil-Comp Press, pp. ***–***. To appear.

[17] R. Kübert and S. Wesner, *Service level agreements for job control in high-performance computing*, in IMCSIT, 2010, pp. 655–661.

[18] O. P. Leads, *Opennebula: The open source toolkit for cloud computing*. [Online; accessed 22-June-2010].

[19] A. W. S. LLC, *Amazon EC2 SLA*. `http://aws.amazon.com/ec2-sla/`. [Online; accessed 2-March-2010].

[20] J. MacLaren, R. Sakellario, K. T. Krishnakumar, J. Garibaldi, and D. Ouelhadj, *Towards service level agreement based scheduling on the grid*, in Proceedings of the 2 nd European Across Grids Conference, 2004, pp. 100–102.

[21] NextGRID Consortium, *NextGRID project home page*. `http://nextgrid.org/`, 2008.

[22] M. Resch, *Entgeltordnung fr die Nutzung der Rechenanlagen und peripheren Gerte des Hchstleistungsrechenzentrums Stuttgart (HLRS) an der Universitt Stuttgart*, 2008.

[23] R. Sakellariou and V. Yarmolenko, *Job Scheduling on the Grid: Towards SLA-Based Scheduling*, IOS Press, 2008.

[24] T. Sandholm, *Service level agreement requirements of an accounting-driven computational grid*, Tech. Report TRITA-NA-0533, Royal Institute of Technology, Stockholm, Sweden, September 2005.

[25] R. Schneider, G. Faust, U. Hindenlang, and P. Helwig, *Inhomogeneous, orthotropic material model for the cortical structure of long bones modelled on the basis of clinical ct or density data*, Computer Methods in Applied Mechanics and Engineering, 198 (2009), pp. 2167 – 2174.

[26] J. Seidel, O. Wldrich, P. Wieder, R. Yahyapour, and W. Ziegler, *Using sla for resource management and scheduling - a survey*, in Grid Middleware and Services - Challenges and Solutions, D. Talia, R. Yahyapour, and W. Ziegler, eds., CoreGRID Series, Springer, 2008. Also published as CoreGRID Technical Report TR-0096.
[27] SimPy Developer Team, *SimPy Simulation Package Homepage*. `http://simpy.sourceforge.net/`.
[28] A. Sulistio, U. Cibej, S. Venugopal, B. Robic, and R. Buyya, *A toolkit for modelling and simulating data grids: an extension to gridsim*, Concurr. Comput. : Pract. Exper., 20 (2008), pp. 1591–1609.
[29] E. Systems, *Eucalyputs - your environment. our industry leading cloud computing software.* [Online; accessed 22-June-2010].
[30] S. Wesner, *Integrated Management Framework for Dynamic Virtual Organisations*, dissertation, Universität Stuttgart, Stuttgart, Germany, 2008.
[31] V. Yarmolenko and R. Sakellariou, *An evaluation of heuristics for sla based parallel job scheduling*, in Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International, April 2006, pp. 8 pp.–.
[32] C. S. Yeo and R. Buyya, *Managing risk of inaccurate runtime estimates for deadline constrained job admission control in clusters*, in ICPP '06: Proceedings of the 2006 International Conference on Parallel Processing, Washington, DC, USA, 2006, IEEE Computer Society, pp. 451–458.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 179–191. http://www.scpe.org
ISSN 1895-1767

BEYOND CLOUDS – TOWARDS REAL UTILITY COMPUTING*

MATTHIAS ASSEL, LUTZ SCHUBERT, DANIEL RUBIO BONILLA AND STEFAN WESNER†

Abstract. With the growing amount of computational resources available, not only locally (e.g. multicore processors), but also across the Internet, utility computing (aka Clouds and Grids) becomes more and more interesting as a means to outsource applications and services, respectively. So far, these systems still act like external resources / devices that have to be explicitly selected, integrated, accessed and so forth. In this paper, we present our conceptual approaches of dealing with increased capacity, scale and heterogeneity of future systems by integrating and using remote resources and services through a kind of web-based "fabric".

Key words: distributed systems, resource fabric, utility computing, service-oriented operating system, distributed application execution, management of scale, multicore architectures

1. Introduction. Over the last few years distributed computing has gained in relevance, not only as a concept, but – more importantly – as a commercial reality: through "Clouds" and multicore processors which make concurrent compute units available for the average user [1, 2]. Notably, usage of these two environments differs significantly. Cloud systems, on the one hand, are essentially server-like machines available over the Internet and accessed accordingly (via e.g. remote desktop or SSH). On the other hand, the capabilities of multicore machines are basically locally available – implicitly there are no specific means needed to access and use such resources.

What is more, cloud systems typically deal with scaling a specific service or application multiple times according to access and availability needs, i.e. perform scale by replication. As opposed to this, multicore systems, just like high performance computers deal with scaling a single application "vertically" across the resources in the form of instantiating multiple processes that together form the application logic, as opposed to individually [3]. It should be noted in this context though that desktop systems and high performance computing (HPC) systems differ in this respect: whilst the latter typically only deal with execution of one single process or thread per core at a time, desktop usage typically implies concurrent execution of multiple services and applications in a time-sharing manner.

In both the cloud and the multicore case we talk of "distributed systems" in the sense that the system on which services, applications or even single processes are being executed consists of multiple resources (i.e., compute node vs. compute cores) connected via a communication and / or messaging link (i.e., interconnect vs. buses). From this perspective, we can specifically note that clouds and multicore systems provide (very) similar capabilities on different environments. Section 2 will elaborate how a common "denominator" of such a distributed system could look like.

By exploiting the commonalities, rather than focusing on the differences, it would in particular be possible to exploit remote resources as if local, thus truly realising the Grid's original concept [4, 5]. For example, an application could be replicated beyond the restriction of the local system, and new applications executed remotely without interfering with any local executions, thus competing over resources. In addition, even simple laptop systems would be enabled to execute demanding applications in the same fashion as on a home or office PC [6]. Whilst this is not a new idea as such (see [4]), its realisation has many impacts not only on a middleware, but more importantly on operating system and programming model. Sections 3 and 4 of this paper will highlight how such a "resource fabric" [20, 21] could be realised and to which specific technological issues it could be applied.

Merging the different paradigms of "distributed systems" would enable new types of systems and implicitly new types of applications that allow following the worldwide trend of internet integration, dynamic outsourcing etc. in short, the Future Internet. Section 5 will describe two types of application areas in more detail. As will be shown, it is not yet possible to overcome all technical issues easily, in particular since the "natural" technological development (i.e., the industrial / commercial provisioning) follows the laws of stepwise development, rather than disruptive (r)evolution. We will conclude this paper with an analysis of the main outstanding issues in section 6.

*This work is partially supported by the S(o)OS project under FP7 grant agreement no. 248465 (http://www.soos-project.eu).
†HLRS – University of Stuttgart, Dpt. of Intelligent Service Infrastructures and Dpt. of Applications & Visualisation, Nobelstr. 19, D-70569 Stuttgart, Germany({assel,schubert,rubio,wesner}@hlrs.de).

2. A "New" Von Neumann Architecture. "Distributed systems" in their widest sense , i.e. including multicore processors, clusters, clouds and grids all have in common that they integrate compute units over a communication link. The main difference thereby being the specifics of the linkage: whilst communication between cores in a multicore system has a very low latency, cloud and grid systems generally use the intra- / internet for communication, meaning high latency but considerably large bandwidth. Finally, HPC systems integrate different levels of linkage, ranging from multicore interconnects to fast, broadband networks (100GB Ethernet, Infiniband).

In principal, latency can be compensated by bandwidth, i.e. if the delay is l and the bandwidth b, the effective data d_e communicated in a set of messages over a time-frame t is

$$d_e(t) = t * \frac{b}{l}. \tag{2.1}$$

In other words, latency is anti proportional to bandwidth: if latency is high, bandwidth should be large too, and vice versa. Low latency can compensate a small bandwidth, in order to reach the same effective data throughput. However, an important factor has been ignored in this calculation, namely the numbers of communications within that time-frame. Obviously latency impacts only per individual invocation, respectively message exchange (actually leading to half the throughput per communication side), meaning that the full throughput depends on the number of invocations, which is accordingly high if the latency is small, and low with a large latency.

$$Invocations = \frac{t}{l}. \tag{2.2}$$

For non interactive systems this does not play a major role, the delays in communication are not noticeable as such – however, in applications that directly interact with the user (such as a word processor), any delay occurring between user input and system reaction beyond 0.1 seconds leads to the impression of a "non-reactive" system, and beyond 1 second, it will even disrupt the user's flow of thought [7]. Browsers take a position somewhere between interactive and non-interactive and users show a slightly higher tolerance towards waiting time than in local applications (4 seconds for maintaining the flow of thoughts according to Young and Smith [8]). Implicitly, offering applications via the web typically leaves an impression of them being slow and unresponsive.

Latency is always a source for problems, when task execution is synchronously dependent on communication, i.e. when the querying task is blocked as long as it is waiting for a response. This affects in particular parallelised applications where the individual processes need to synchronise data. The average developer however is not aware of the connection details – not only are they difficult to acquire, but even more difficult to represent and cater for in the program in some form. The general tendency is therefore to make use of asynchronous messaging in the web domain (clouds, grids) where latency and bandwidth is high, and synchronous messaging inside compute clusters, trying to minimise latency. However, applications with little communication needs may nonetheless occupy multiple resource instances for their execution - in particular embarrassingly parallel applications fit and scale well in either environment. By removing the interactive part from the processing tasks, web-based systems can even be exploited for demanding user applications.

To achieve this, we do no longer need to distinguish between different resource types, but between their connectivity. From this perspective, the different notions of distributed systems fall together as a complex system consisting of computational resources connected over a communication link. Modern systems are therefore no longer building up a strict von Neumann architecture, but a modular platform that connects any amount of von Neumann like units. In particular from a programming perspective, we can therefore derive a modified von Neumann architecture as depicted in Figure 2.1, in particular the I/O moves closer to the processing unit (PU) thus allowing for connectivity between multiple PUs without an explicit single central instance. We also distinguish between processing and memory units (MU), which are connected through a dedicated I/O. It must be noted that current multicore systems do not make use of an explicit I/O unit for core-2-core communication – whilst this is expected to change, the according I/O will still most likely differ from the one connecting with the outside of the processor. It must also be mentioned that cache access is not linked to an I/O as such, but to a memory controller – again, long-term developments (e.g. ring over caches) will impact here.

All this however, has no impact on the programming layer. In this view, clusters, multicore processors and cloud or grid based systems are essentially identical. So far, this model is realised through according means

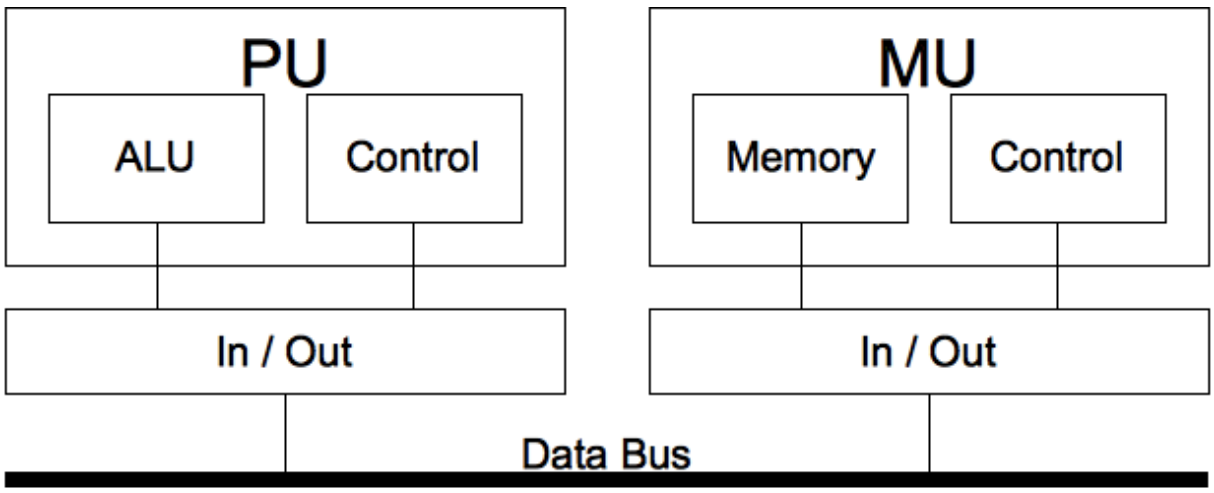


Fig. 2.1: A modified von Neumann architecture where I/O is directly coupled to multiple compute units and memory units, respectively.

on the middleware level (with the highest level of abstraction being represented through distributed workflows, e.g. [22]).

2.1. Data Management in the "new Model". The massive distribution of data over multiple machines (i.e., storage points) as provided by this new architecture fundamentally changes current data management concepts, too, in particular data generation, exchange and storage. As data usage grows faster than bandwidth (and even faster than storage), any form of data movement implicitly consumes a large amount of time and resources, respectively. To compensate for this restriction, new mechanisms, strategies and tools are required that manage the movement of particular data sets (similar to [9]) in segments across a large storage hierarchy. Ideally, data movement is restricted to the minimum amount needed at any specific time and maintained in a way so as to reduce the distance between sender and recipient. Similar to the code (see below), data must therefore be managed in a more intelligent fashion, according to the points of access and the degree of synchronisation, respectively coherency across usage points (see also [10]). As such, data that is frequently used should be moved to highly parallel dynamic storage (e.g. parallel file systems like Oracle's Lustre[1] or virtual distributed file systems like GFS [11] or a combination of both), while archived data should reside in "passive" storage devices (e.g. low-cost storage devices or robotic tape libraries).

To allow for this separation, a fundamental requirement is the ad-hoc allocation, use and release of particular storage space. Hence, algorithms should automatically request necessary space but also track and delete unused data from these dynamic storages, so as to minimise storage costs and increase throughput. This is not only relevant for any storage device but also main memory and caches are affected. Both are extremely useful to achieve fast processing of data sets and thus higher throughput, too. It should be noted here that the latter units are most effective and relevant for multicore processor rather than distributed systems. However, the entire data management hierarchy ranging from caches and main memory to any (external and / or remote) storage device has to be considered and optimised in order to achieve better performance and thus reduce latency. Of particular relevance is also that data source and applications are not necessarily directly coupled. Collections of data sets should be organised as hierarchical directories. Such abstraction will essentially change the way the I/O is expressed by applications and will involve data exchange and storage management in a form that maps data sets into physical devices without affecting the application's behaviour.

Similarly, replication of data represents a key issue to increase data availability through locality. Management of replicas has to carefully deal with lifetime issues to remove outdated pieces immediately, so as to not unnecessarily block storage space. New replicas should be dynamically created, distributed and / or destroyed

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

based on users' and applications' needs but also according to technical requirements [12]. In shared environments, i.e. where multiple access points require the same data, serious consistency issues across nodes have to be addressed [13].

3. Weaving Resource Fabrics. The classical approach towards dealing with distributed systems consists in providing a form of middleware that translates function invocation, instantiation etc. into a set of remote procedure calls or web service calls. The actual details depend not only on the realisation but also on the domain applied to.

In multicore systems, the actual transaction logic is encapsulated and realised via the hardware; supercomputer clusters employ some form of dedicated communication programing model (with either explicit (e.g. MPI[2]) or implicit (e.g. PGAS[3]) messaging) that is translated into ports and sockets at compile time. The grid / cloud support typically builds on HTTP protocols that are typically translated into ports and sockets at runtime. In other words, the actual type of communication bridge is transparent to the user, though he will still have to use it in different ways, depending on use case and environment. Though there are ways of controlling and configuring the communication link, there is little possibility to exploit this dynamically for maintaining distributed code – in other words, parallel programming models that use synchronisation for controlling the execution behaviour base on the assumption that the underlying infrastructure is effectively homogeneous, or at least give this impression to the user. In reality however neither the resource infrastructure, nor the program is effectively homogeneous: even in the strong scaling based HPC domain, the actual algorithm consists of both parallel and sequential segments that interchange during execution.

What is more, e.g. in typical (unstructured) numerical grid algorithms (such as blood-flow simulation etc.), the communication relationship between the parallel processes is not symmetrical, meaning that the code would benefit from a distribution on the infrastructure aligning the connectivity between compute units with the neighbourhood model of the code. The classical means to developing parallel applications consists in either segmenting the work or the data by identifying natural partitions of either of those [14]. Whilst this is the most common approach, it suffers from the drawback that it a) requires good knowledge about the program and the concurrency in work and data, necessitating additional development work; b) the partitioning may be too small or to big to be efficient, in particular if the communication exchange between segments is not considered properly; c) the specific capabilities of a heterogeneous system are mostly unused – what is more, if the according effort is undertaken to adapt code to the specific system, it will become less portable; and finally d) not all kind of segments can be identified this way. Heterogeneity and architecture of the system are typically regarded as obstacles, but program execution can actually benefit from this structure by reflecting the "natural" behaviour of an application.

We can identify the following key aspects of any algorithm:

1. Concurrency – some functions are executed without explicitly sharing data or resources. In this case these segments can be executed in parallel.
2. Parallelism – similar to concurrency, some functions operate on a common data set, but the actions they perform are not directly dependent on one another. This is typical for loops over large grids ("loop unrollment").
3. Interactivity – in particular in desktop applications, the interface towards the user can be easily defined by the connectivity to external input resources.
4. Background Tasks – are tasks waiting for specific events to execute and with little relationship to the main execution (regarding data dependency).

Similarly, the communication needs are not the same between all these parts, though there is no general statement about the relationships possible: concurrency and parallelism do not necessarily imply high connectivity (low latency), as e.g. embarrassingly parallel applications show. On the other hand, events processed in the background may require immediate response, and interactivity does not mean that all processing on the input data has to be executed immediately (cf. word count and spell-checking in modern word editors: even though they react "immediately" to any input, the delay until the actual results are available is of no concern to the user and hence typically not even noticeable). Again, keep in mind that if the result is in some form relevant (in the sense of often checked by the user), the time frame for maintaining a flow of work is 1 second [7].

[2] http://www.mcs.anl.gov/research/projects/mpi/
[3] http://pgas.org/

3.1. The Structure of Applications. The structure of an application can therefore be used as an indicator for its distributability (and, to a degree, parallelisability). The (runtime) behaviour provides additional information about the actual connectivity between the individual segments and thus its requirements towards the communication model, i.e. the relationship of latency versus bandwidth. Implicitly, runtime behaviour effectively provides more information about the potential code and data distribution than the programmer can currently encode in the source code. This is simply due to the fact that this is not in-line with our current way of writing programs and is implicitly not directly supported by programming models. The foundation is however laid out by integration of remote processes (web services) and dedicated synchronisation points in parallel processes – this does not always reflect the best distribution though, as the according invocations are mainly functionality- rather than communication-driven. A way of identifying and exploiting the "behavioural" structure of the application for distribution purposes consists in runtime analysis and annotation of the code and data memory to produce a form of dependency graph (cf. Figure 3.1) which depicts the invocations of memory locations (code) and read / write operations on data.

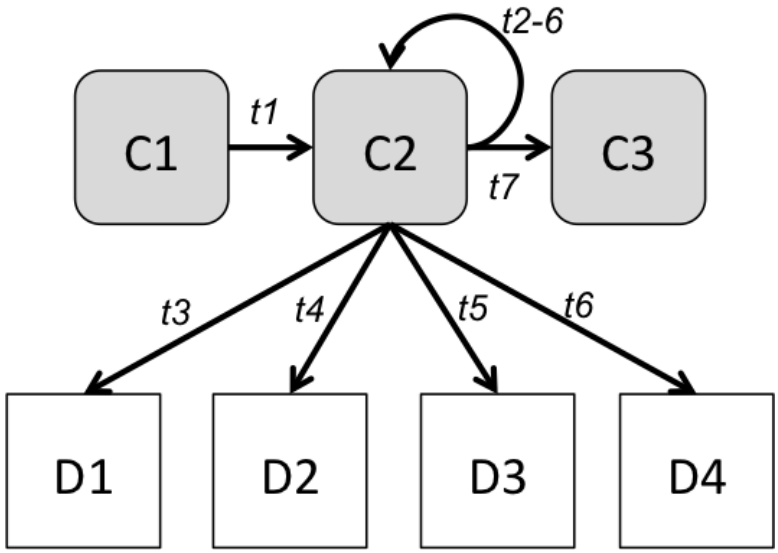


Fig. 3.1: A simple code (C) and data (D) dependency graph of a sequential application with a loop over an array (C2). The graph denotes a sequence of actions with t(x) representing the xth transition, respectively access action.

In order to acquire the according code and data blocks, the basic starting point consists in identifying jumps and non-consecutive data accesses. Figure 3.1 exemplifies how the graph information can be used to identify an unrollable loop (C2) that consecutively accesses unrelated memory blocks to produce a result. In this simplified case there is no data dependency between C1 and C2, or even between C2 and C3, which means that the unrolled C2 blocks do not even have to be synchronised. An example of such a loop would be

C2: for (int i = 0; i < 4; i++) a[i] = 0;

As opposed to this, Figure 3.2 depicts the example of a loop that can not be unrolled due to dependency issues in C2. This loop could look like

*C2: for (int i = 1; i < 3; i++) a[i] = a[i-1] * 2;*

Notably, the examples given are not sensible for parallelisation, as the actual work load per iteration is too small to compensate the overhead for spreading out, communication etc. All analysis so far is based on the simple fact of data relationship and dependency – not unlike the approach pursued in StarSS [15], which takes a directive based approach to task parallelisation on top of C, C++ and Fortran. The runtime takes care of dependency

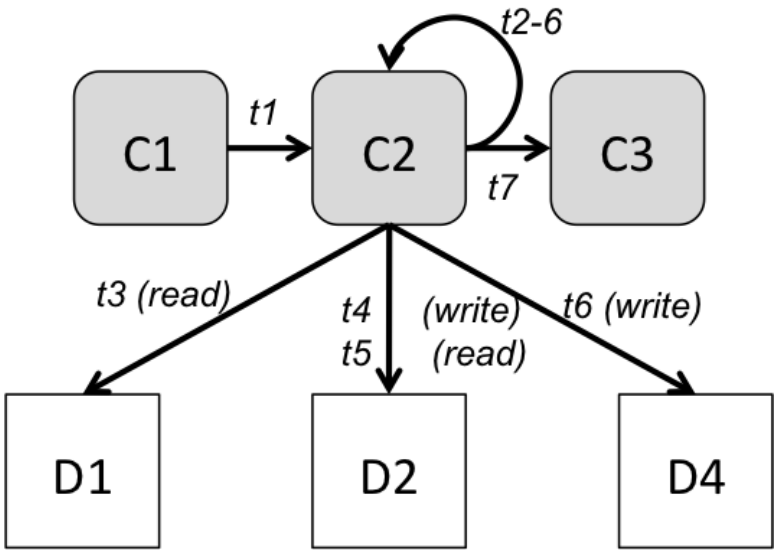


Fig. 3.2: Example of a loop that is not unrollable and its representation in a dependency graph.

tracking, synchronisation, and scheduling. Source code annotations allow the programmer to designate tasks for concurrent execution along with their data dependencies. During runtime, such tasks are placed into a dependency tree and scheduled for execution by a number of working threads a soon as the data dependencies are met.In a nutshell, the StarSs directives 1) designate specific code functions / subroutines to be executed concurrently as so call tasks by the runtime; 2) specify the direction of a function subroutines parameters, i.e. input, output, or inout, which is later used to infer the inter-dependencies of task in a task-dependency graph.

Accordingly, even though a pattern based approach like the one presented above does principally provide information about the distribution and parallelisabilty, it does not per se increase the execution performance. In order to not only identify principle points of distribution and parallelisation, but to also make code execution more efficient, so as to exploit the specific benefits of parallel architectures and infrastructures, further information about the code behaviour is needed – in particular the "strength" of code and data relationships, and the size of the segment. Note that timing can be derived through more detailed sequential information (i.e., in Figure 3.1 or Figure 3.2 by storing actual timestamps in t_n). Strength of relationship is thereby proportional to amount of invocations divided by the full execution time.

With this information, we can derive a graph (Figure 3.3) where the strength of the relationship (e.g. C2 calls C3 less often than C1 calling C2) and the size of the underlying code and data is encoded as weights of vertices and edges. For simplicity reasons we left out timing information in the figure and concentrated on a very simple code structure (without loops or similar).

The dependency information in this graph, in combination with the size information can be used to extract different segments in the form of subgraphs according to nearness (connection strength) and combined size. Or to put it in computational terms again: according to the number of memory accesses, whereby fewer accesses imply a potentially good cutting point.

We can thereby distinguish between different types of data dependencies that relate to the "strength" of coupling, in the sense of the speed required, or rather assumed for executing the according transfer. In other words, the strength represents the implicit cost for execution performance if the according link should be delayed. Obviously, the strongest link is therefore the exchange of values via the register set of the processor, this is followed by memory access and finally any external I/O. It may come as a surprise that the implicit linkage through sequential execution of operations is similar to the normal workflow. This is simply due to the fact that effectively any code can be split and, more importantly, even be executed in parallel, given that this does not affect any values that are processed in the overall algorithms. In other words, if the two segments or operations are concurrent, each partition reflects the code to be executed on one compute unit (core, node etc.) and connections across segments need to be realised through a cross-process communication link. Information

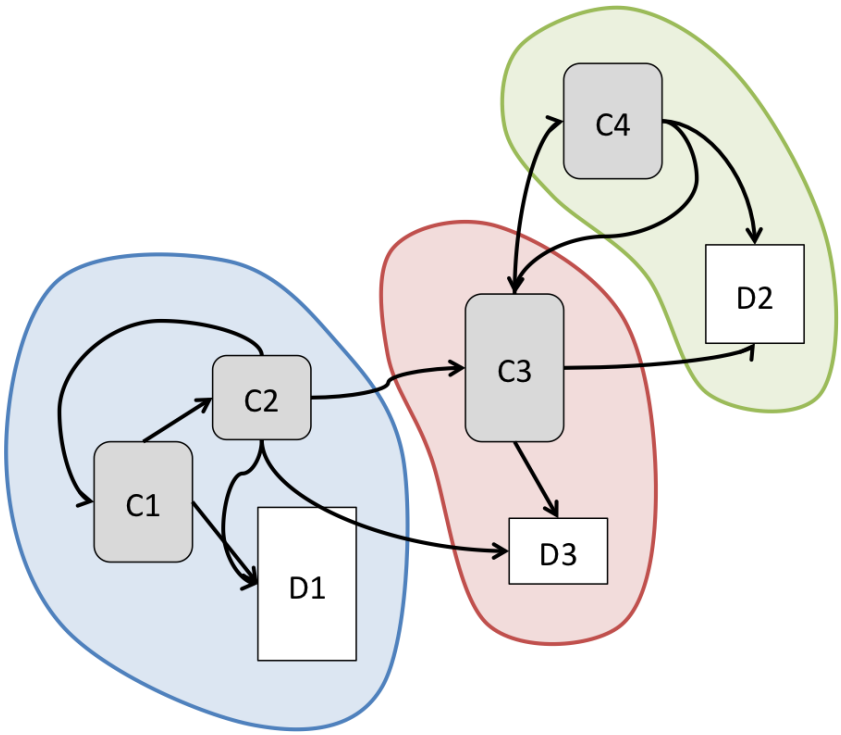


Fig. 3.3: A dependency graph with implicit relationship strength (length of the vector) and size information (size of the block). The three areas depict potential areas of segmentation.

about the bandwidth and latency of the system's specific communication links can thereby serve as an indicator for cutting point identification, if it is respected that access across segments is effectively identical to data passing – accordingly, the temporal sequence and dependency of access constrains the segments' independency and hence concurrency.

We can regard this as a specific case of the maximum flow theorem, where the maximum communication between any two given nodes in a flow network (represented by a graph) with limited capacities (bandwidth) is searched for. In our specific case, the maximum flow of a network denotes the code instances / parts that should least be cut: If we interpret "flow" as the amount of data shared between two instances, than "capacity" is the restriction imposed by the type of data exchange. By segmenting the network we imply a further reduction of capacity at the cutting point – we therefore try to select the points where this impact is minimised, i.e. the minimum cut. The min cut algorithm applied therefore looks for the edges with the minimal capacity and flow [23].

As the goal of this process consists in exploiting concurrency between different code logic segments, distance between nodes (and therefore capacity and flow) are also particularly influenced by the potential degree of execution overlap. Execution overlap between two segments is effectively represented by the distance between the connecting nodes, i.e. the number of operations executed between the two connected access commands. For example, assume that a program writes a variable at the beginning of its execution, but will actually only use this variable after multiple seconds of execution. If no further relationships in between these two commands exists, the program can be split at any place in between and these two segments can be executed in parallel at the same time, as long as the first write command is executed before any further data manipulations take place. We thereby want to maximise the degree of execution overlap to make best use of the resources and thus minimise the total execution time.

The degree of potential overlap (i.e., the maximum degree of overlap between two potential code segments) thus obviously impacts on the weight between two nodes in the dependency graph. This means that the higher the degree of overlap, the less relevant the factor of capacity and flow. In other words, if there is enough time to communicate the according value by any means, the fact that the program assigns a specific strength to

the access plays no role anymore. The segmentation process must therefore respect this when calculating the minimum cut. It must be noted in this context that with the explanation above, we ignore the time for actually communicating the according data, which impacts on the degree of partial overlap, too.

The full algorithm would exceed the scope of this paper and is therefore not elaborated further here. It is obvious that one of the major problems for an efficient segmentation consists in the right data gathering granularity: whilst too fine data blocks will cause memory and algorithm to go over bounds, too coarse information will lead to too strong relationships and too large segments, so that the effective gain through distribution is counterweighted by the overhead for communication and memory swaps.

3.2. Lifecycle of Applications in a Resource Fabric. As noted, the major part of the information about the code is acquired at actual runtime – implicitly, the distribution information may change during execution, leading to potential instabilities and dependencies, respectively. In this section we will examine the full lifecycle of an application executed in such an environment and implicitly the steps involved to better exploit a scalable (and potentially dynamic) environment can be captured:

1. Analysis of the application behaviour ("Analyse"): In an initial step (first time the application is executed), there is little to no dependency information available, unless an according programming extension (such as StarSS) has been applied. This means that initially the application is executed locally in a virtual memory environment that logs the runtime behaviour and hence the dependencies between code partitions and memory segments. Implicitly first time invocation is effectively identical to sequential execution – even though user and or compiler (e.g. using the OpenMP[4] model) provided parallelisation (if any) can still be exploited.
2. Identification of appropriate resources ("Match"): The dependencies in the application graph reflect the communication needs between segments and thus implicitly indicate the required infrastructure architecture, including type and layout of interconnects. But also specific core types can be exploited to a degree – e.g. Figure 3.1 shows clear vectorisable behaviour and other microarchitecture specific patterns can be identified.
3. Distribution and adaptation of code and data ("Distribute"): When appropriate resources could be identified, the code segments can be distributed across the infrastructure accordingly. In the simplest case, all partitions will be uploaded prior to actual execution, in which case no additional data has to be distributed at runtime (besides for the data transported during communication). However, in principle, it is possible to distribute the segments in their order of invocation, though this runs the risk of potential delays.
4. Execution and runtime analysis ("Execute"): Actual execution is principally identical to any distributed program execution with explicit communication between process instances. However, as opposed to an explicitly developed parallel program, the source code in this model has not been altered and the communication points and tasks are unknown to the algorithm itself. Accordingly, the infrastructure has to take care of communicating the right data at the appropriate time. From the development perspective, this is principally identical to the PGAS (Partitioned Global Address Space) approach, which provides a virtual shared (global) memory and deals with the communication necessary to enable remote data access. Effectively the system for enabling distributed execution in the model proposed here must enact the same tasks directly on the (virtual) memory environment of the operating system (OS), rather than on the programming level. During execution, the system may continue analysing behaviour (cf. step 1 "Analyse") in order to further improve the segmentation information and granularity. As program behaviour changes according to code dependency, the main issue in this phase consists in ensuring and maintaining an efficient stability of the distribution. Monitoring and segmentation must therefore not only consider the dependencies, but also higher-level parameters that implicitly define the stability of a given segmentation. Though there is some relationship to SLA based monitoring, these parameters should not be confused with common quality metrics.
5. Information storing ("Store"): Behaviour and distribution information should be stored after execution in order to be retrieved in the next iteration, thus allowing to skip step 1 and thus improving the process.

The main issues in the lifecycle consist obviously data exchange and synchronisation of the segments which are treated as distributed processes. If data and execution are not carefully aligned, inconsistency may lead

[4]http://openmp.org/wp/

to serious crashes, deadlocks, or serious efficiency issues due to overhead. It is therefore not only relevant, from where data is accessed, but also when and in which order. Ideally, data is being transported in the "background" whilst the segment is mostly inactive. We will not elaborate these issues here – suffice to say that if the segments can be treated completely isolated from each other and all memory is swapped during execution passover, coherency and consistency is ensured.

4. A Middleware for Resource Fabrics. In order to realise an environment that enables such distributed execution, it must accordingly provide some capabilities to capture memory access and intervene with code progression so as to pass the execution point at the boundary of the individual segments. Effectively this means that the system provides a virtual environment in which to host and execute the code - a full virtual system however would impact on execution performance. Whilst this may be acceptable for some type of applications where simplicity of development ranks higher than performance, in particular in scalable environments, efficiency typically ranks higher.

However, the system does not necessarily need to provide a full virtual machine, but in particular a virtual memory environment and a set of interfaces to access (shared) resources – in other words, an operating system. Since effectively in all execution parts memory access needs to be controlled, a centralised operating system approach would create a serious non-scalable bottleneck and additional delays due to message creation, synchronisation and communication would turn out a major performance stopper. This relates to the major reasons why monolithic, centralised operating systems show bad horizontal scaling capabilities [16]. In order to achieve better scalability the individual compute units hence need some local support to reduce overhead and enable virtual memory management in a form that can also identify and handle segment passover.

The S(o)OS project[5] funded by the European Commission investigates into new operating system architectures that can deal with exactly this type of scenarios. The approach thereby essentially bases on a concept of distributed microkernel instances that fit into the local memory of a compute unit (processor core) without obstructing it, i.e. leaving enough space for code and data stack. We can essentially distinguish between two types of OS instances in a resource fabric: firstly, the main instance that deals with initiating execution and distribution, as well as scheduling the application as a whole. Secondly, the local instances that effectively only deal with communication and virtual memory management. In effect, all instances could have the same capabilities [16], which would make them larger and more complex to adapt to specific environments though. In S(o)OS the principle is extended with on-the-fly adaptation of local OS instances according to application requirements and resource capabilities – this way, the kernel can even support adaptation to resource specifics without the central instance having to cater for that [17].

In Figure 4.1 we depict the principle of such an operating system with respect to the relationship between cores (or compute units, comes to that): typically one selected unit will take over the initial responsibility for code and data, i.e. loading it from storage, analysing it (respectively retrieving the annotations) and distributing it accordingly. The segmentation information (i.e., the memory structure derived from the dependency graph) will be passed with the code segments, allowing the local instance to build up the local virtual memory and implant system invocations at the appropriate time (during segment passover).

4.1. Distributed Execution. Pure distributed execution without any execution overlap is effectively similar to context switching during time-sharing, only that the "new" state is not uploaded from local memory, but provided over the communication link between the initiating unit and the one taking over. However, parts of the state (code and base data) can principally already be available at the destination site due to predistribution according to the dependency graph (cf. above).

Similar to context switching, the application itself does not have a dedicated point at which to perform the switch (or even enable it), though with additional programming effort context switching can be avoided (thus leading to single-task execution, up to a point). As opposed to multi-tasking operating systems, however, a S(o)OS like system must initiate the context switch (execution passover) at a dedicated point, according to the segmentation analysis (cf. Figure 3.3). This point is effectively the end of the local segment and can thus be initiated by a simple jmp (jump) command into the virtual memory address space of the new segment and can thus simply be appended to the partition. Just like with any access to remote data (be that due to branching or data access), the operating system has to intercept the request prior to its execution, similar to classical virtual memory management. The MMU (Memory Management Unit) of modern processors supports this task

[5] http://www.soos-project.eu/

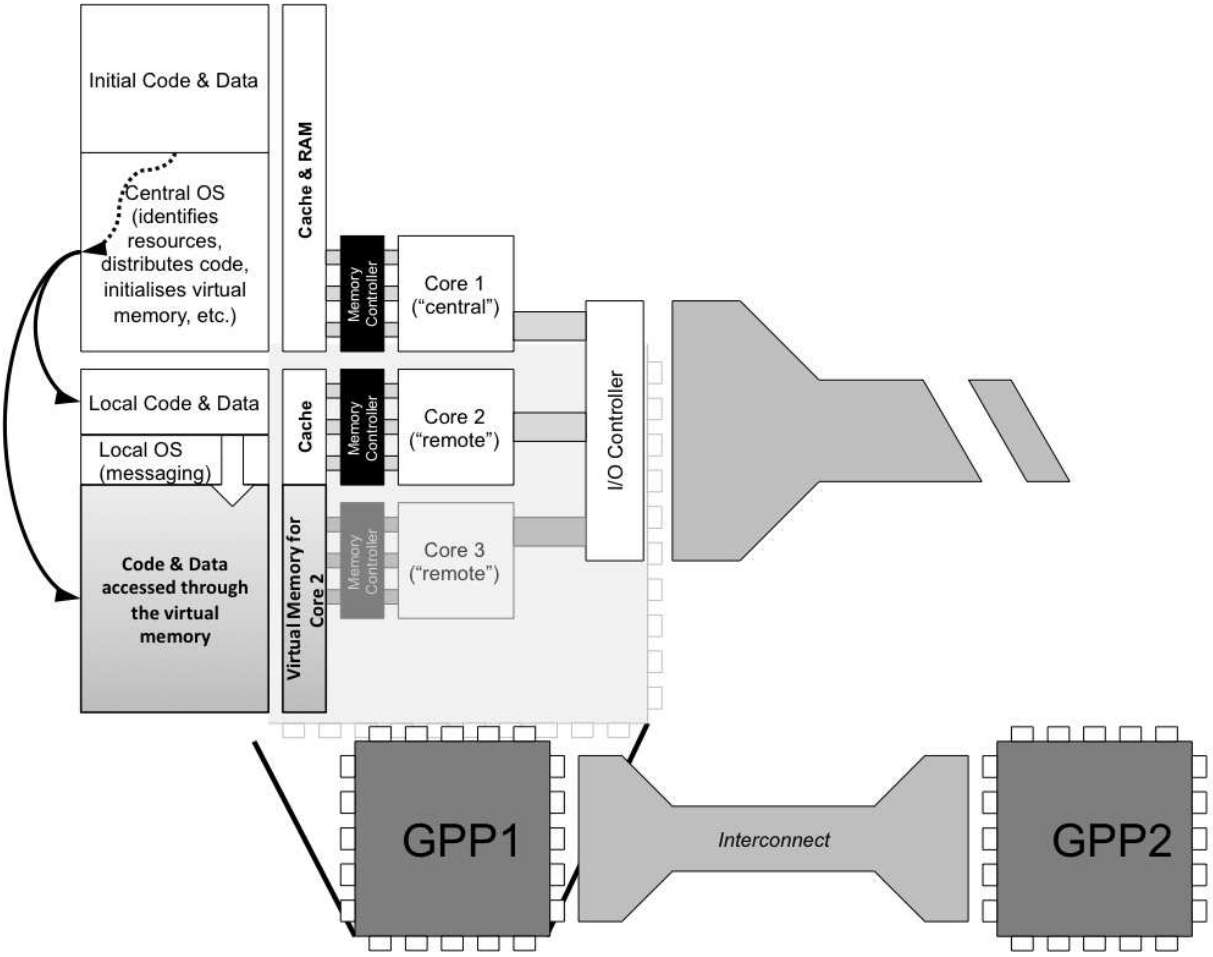


Fig. 4.1: A distributed micro operating system and its relation to the individual compute units in the system. "Remote" cores can be hosted within the same processor or principally any remote machine.

already, but would need to be extended to also cater for the specific needs of a distributed (cross-unit) memory virtualisation.

In other words, as soon as the instruction pointer leaves the area of available, and more importantly, designated code operations, the system needs to be able to judge how to handle the according situation. In the traditional, unhandled fashion, the system would interpret this similar to a cache miss for data access and initiate fetching further instructions from memory. In the distributed case, this should however trigger an execution passover to the remote processor / memory unit. As the point of passover is defined by the segmentation process as detailed above, it seems appropriate to extend the machine code operations with a set of instructions for handling the passover at the cutting point.

Notably, code segments in such a resource mesh are essentially treated as independent, separate processes, that can principally be executed in parallel. The execution restrictions are given by the data dependencies that can implicitly be regarded as execution triggers. From this perspective, execution passover is hence not so much a question of identifying the next appropriate segment, but of validating that all relevant preconditions for execution of the respective segment are met.

In the straight-forward approach we could therefore maintain the data dependency graph as a workflow description of execution, whereas dedicated flags indicate whether an edge is "satisfied". In this case, this means that the according data access operation that forms the precondition for execution of the second segment has been met and that hence the execution of the dependent process can start, once all these preconditions have been satisfied. Programmatically this means that flags per datum / edge are maintained which are constantly

checked by a scheduler which selects the appropriate process to execute on basis of these flags.

Whilst this approach is nice due to its simplicity, it leads to insufficient resource usage – in particular since the partial overlap between execution segments are not exploited. This can be easily demonstrated in the case already introduced above, where an algorithm assigns a variable early in its execution, but only uses the variable comparatively late. With the partial overlap condition in the segmentation process as described above, we can assume without restriction of generality that the write operation will be executed in the middle of the first segment, whilst the read operation will occur in the last third of the second segment, so as to increase resource utilisation. If however, execution of the second segment is delayed until the first segment has reached the point of the according write operation, the core dedicated to the second segment will idle for at least half of the execution time. This would lead to an execution performance that is only minimally better than the sequential case, even though the situation would allow for twice the execution speed.

The obvious alternative consists in executing the processes till the point of data access and stall further processing until the data becomes available. As discussed below, this could be achieved by having the process actively query / access the data source process according to its data dependency specification in the segmentation graph. Whilst this approach obviously performs well for the case presented above, it may nonetheless lead to a similarly weak resource utilisation when the first process writes data quite late and the second tries to read early, i.e. if the execution overlap is small (respectively non-existent). This is obviously the case for segments that are intended to be executed at a later point in the process in parallel with other segments than this first one.

Even though this latter case creates weak resource utilisation, it must nonetheless be noted that the access delay does not create the same execution delay as in the first case. This is due to the fact that the maximum overlap is still exploited in the case of delayed access. However, to exploit this approach implicitly required that a large amount of otherwise unused resource are available. Since we cannot assume that (in particular from an energy-saving perspective), we must instead try to improve the utilisation.

Accordingly, the segments in the dependency graph must be sorted so as to form a workflow that respects two crucial aspects: (1) availability of resources in principle, and (2) maximising the total degree of overlap and thereby reducing the total execution time. The second aspect implicitly prevents, respectively reduces the delay time for data access.

These concepts are implicitly closely related to classical virtualisation concepts, even though in this case, it has to act on a much lower level than typical virtualisation approaches – namely in between hardware and machine code. The main task of virtualisation thereby also does not consist in exposing different hardware capabilities (even though it may be exploited to this end), but more importantly in maintaining an integrated view on a dispersed environment.

4.2. Data Maintenance. Nonetheless, it must still be taken into consideration that the actual transfer of the according data consumes time, too. This specific delay was part of our initial considerations of where and how to perform the code segmentation. In addition, the execution time of a process can only be roughly estimated and it may therefore happen often enough that a dependent process overtakes the data source process and therefore attempts to access data prior to its availability.

In order to reduce the impact from communication and delay, data must be handled intelligently by the system. We can note thereby that the OS instance has principally three options to deal with data access:

1. preemptive distribution: provide the data to all requestors, before they actually try to access it;
2. context switch: pass all status data when passing over the execution point to a remote instance (does not apply to parallel processes);
3. on demand: make data available and accessible at the moment the remote process requests it.

The actual decision will not only depend on time of access (in particular for parallel processes), but also on size of the data and on outstanding tasks of the processing segment. For example, if the data will only be ready by the end of the segment's processing, there is no point in distributing it earlier. However, if the data size is too large for single provisioning without introducing unnecessary delays, partial data updates according to the data segments (cf. above) may be distributed ahead of time (preemptive). Note that current processor architecture does not allow for easy background data transmission and in most cases the communication will stall execution of the main process. As noted, a particular issue in this context consists in ensuring data coherency across segments and avoiding deadlocks. Intel's MESIF protocol over the Quick Path Interconnect [18] is one means to ensure cache coherency in a distributed environment at the cost of access delays. The basic principle of

this approach consists in checking the consistency of a datum at access time by verifying it against all other replicated instances. Obviously this protocol does not scale very well and requires a specific type of architecture that will most likely not be supported in future large scale environments any more [19].

5. Exploiting Resource Fabrics. Since the system primarily caters for distribution of an application across a (potentially large-scale) environment for effectively sequential execution, how would this approach help solve the problem of dealing with future infrastructures? The system offers two major contributions that will be discussed in more detail in this section, namely support for high performance computing but also common web applications.

5.1. Supporting High Performance Computing. Though the primary concern is distribution and not parallelisation, the features and principles provided by a S(o)OS like environment deal with essential issues in large scale high performance computing, namely:

1. exploiting cache to its maximum;
2. providing a scalable operating system;
3. matching the code structure with the infrastructure architecture;
4. managing communication and synchronisation in a virtual distributed shared memory environment.

Whilst the system does not explicitly provide a programming model that deals with scalability over heterogeneous, hierarchical infrastructures, it does support according models by providing additional and enhanced features to deal with such infrastructures. In particular, it implements essential features as pursued e.g. by StarSS and PGAS: the concurrency information extracted from the memory analysis is consistent with the dependency graph that StarSS tries to derive [15] and could be used as an extension along that line. The main principle behind PGAS on the other hand consists in providing a virtual shared memory to the programmer, with the compiler converting the according read / write actions into remote procedure calls, access requests etc.

5.2. Office@World. A slightly different use case supported by the S(o)OS environment consists in the current ongoing trend of resource outsourcing into the web (cf Sec. 1). As has been noted before, an essential part for web exploitation consists in maintaining interactivity whilst offloading demanding tasks. As we have shown, this does not only affect code, but equally data used in an application – accordingly, the features are of particular relevance in a future environment where most code and personal data will be stored in the web. As discussed in detail in [6], the system thus not only allows that code and data becomes accessible and usable from anywhere at any time (given internet connectivity), but also virtually increases local performance of the system. This enables in particular frequent travelers to exploit a local system environment for their applications whilst using data and code from the web – future meeting places could thus offer compute resources that can be exploited by a laptop user, but also the other way round, i.e. a home desktop machine could replicate the full environment of the laptop. A low-power, portable device could thus turn into a highly efficient system by seamlessly integrating into the "Resource Fabric" without requiring specific configuration or development overhead [6]. This is similar to carrying an extended secure full work, respectively private profile with you.

6. Conclusions. We have presented in this paper an approach to dealing with future distributed environments that span across the Internet, multiple resources and multiple cores (i.e., computational units), but also across different types of usage and applications (such as High Performance Computing and Clouds). "Resource Fabrics" are thereby an effective means to integrate compute units non-regarding their location and specifics.

We have shown how a general view on computing resources can serve as basis for such a development, enabling higher scalability. To this end, a more scalable operating system is required that enables the explicit usage of resource fabrics as discussed here. The according OS is still in a highly experimental stage, and more experiments are needed for concrete performance estimations. However, the basic principle has already been implicitly proven by the Cloud Platform as a Service movement, where the dichotomy of interactive and "executive" parts of the application is used to exploit remote resources and services.

Many issues still remain speculative, though, for example regarding the execution of parallel applications where time-critical alignment is crucial. Along this line in particular the concurrent exploitation of remote resources, leading to potential time sharing of individual units (and their resources, such as I/O) still needs to be assessed more critical and deeply. With the general movement towards multicore systems, it can however generally expected that time-sharing during execution time is no longer a valid execution model. It will be noticed however, that we did not even touch upon the issues of security and privacy, which will be a major

concern in scenarios such as the "Office@World" one. So far, we did not touch upon this as other more technological issues are more relevant for the time being – according concepts are expected in the near future.

REFERENCES

[1] M.A. Rappa, *The utility business model and the future of computing services*, IBM Systems Journal, 43,1 (2004), pp. 32–42.

[2] W. Fellows, *The State of Play: Grid, Utility, Cloud* Presentation at CloudScape 2009, OGF Europe. Available at http://old.ogfeurope. eu/uploads/Industry%20Expert%20Group/FELLOWS_Cloudscape Jan09-WF.pdf, 2009.

[3] L. Schubert, K. Jeffery, B. Neidecker-Lutz, and others, *Cloud Computing Expert Working Group Report: The Future of Cloud Computing*, European Commission. Available at: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf, 2010.

[4] I. Foster and C. Kesselman, *The Grid. Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1998.

[5] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, *A Note on Distributed Computing*, Sun Microsystems Technical Report, Available at: http://labs.oracle.com/techrep/1994/smli_tr-94-29.pdf, 1994.

[6] L. Schubert, A. Kipp, B. Koller, and S. Wesner, *Service Oriented Operating Systems: Future Workspaces*, IEEE Wireless Communications, 16 (2009), pp. 42–50.

[7] R.B. Miller, *Response time in man-computer conversational transactions*, In: Proceedings AFIPS Fall Joint Computer Conference, 33, ACM, New York (1968), pp. 267–277.

[8] J. Young and S. Smith, *Akamai and JupiterResearch Identify '4 Seconds' as the New Threshold of Acceptability for Retail Web Page Response Times*, Akamai Press Release. Available at: http:// www.akamai.com/html/about/press/releases/2006/press_110606.html, 2006.

[9] D. Yuan, Y. Yang, X. Liu, and J. Chen, *A data placement strategy in scientific cloud workflows*, Future Generation Computer Systems, 26,8 (2010) pp. 1200–1214.

[10] D. Nikolow, R. Slota, and J. Kitowski, *Knowledge Supported Data Access in Distributed Environment*, In: Proceedings of Cracow Grid Workshop - CGW'08, October 13-15 2008, ACC-Cyfronet AGH, Krakow, (2009), pp. 320–325.

[11] S. Ghemawat, H. Gobioff, and S. Leung, *The Google file system*, ACM SIGOPS Operating Systems Review, 37 (2003), pp. 29–43.

[12] G. Aloisio and S. Fiore, *Towards Exascale Distributed Data Management*, International Journal Of High Performance Computing Applications, 23 (2009), pp. 398–400.

[13] S. Grottke, A. Koepke, J. Sablatnig, J. Chen, R. Seiler, and A. Wolisz, *Consistency in Distributed Systems*, TKN Technical Report TKN-08-005, Technische Universitaet Berlin. Available at http://www.tkn.tu-berlin.de/publications/papers/consistency_tr.pdf, 2007.

[14] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison Wesley, 1995.

[15] J. Planas, R.M. Badia, E. Ayguade, and J. Labarta, *Hierarchical Task-Based Programming With StarSS*, International Journal of High Performance Computing Applications, 23,3 (2009), pp. 284–299.

[16] A. Baumann, P. Barham, P.E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schuepbach, and A. Singhania, *The multikernel: a new OS architecture for scalable multicore systems*, In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, ACM (2009), pp. 29–44.

[17] L. Schubert, A. Kipp, and S. Wesner, *Above the Clouds: From Grids to Service-oriented Operating Systems*, Towards the Future Internet - A European Research Perspective, G. Tselentis, J. Domingue, A. Amsterdam: IOS Press (2009), pp. 238–249.

[18] Intel Corporation, *An Introduction to the Intel QuickPath Interconnect*, Intel Whitepaper. Available at: http://www.intel.com/technology/quickpath/introduction.pdf, 2009.

[19] F. Petrot, A. Greiner, and P. Gomez, *On Cache Coherency and Memory Consistency Issues in NoC Based Shared Memory Multiprocessor SoC Architectures*, In: Proceedings of the 9th EUROMICRO Conference on Digital System Design. IEEE Computer Society (2006), pp. 53–60.

[20] L. Schubert, M. Assel, and S. Wesner, *Resource Fabrics: The Next Level of Grids and Clouds*, In: M. Ghanza and M. Paprzycki (Eds.), Proceedings of the International Multiconference on Computer Science and Information Technology (2010), pp. 677–684.

[21] L. Schubert, S. Wesner, A. Kipp, and A. Arenas, *Self-Managed Microkernels: From Clouds Towards Resource Fabrics*, In: Proceedings of the First International Conference on Cloud Computing, Springer (2009), pp. 167–185.

[22] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor, and I. Wang, *Programming scientific and distributed workflow with Triana services*, Concurrency and Computation: Practice and Experience, 18 (2005), pp. 1021–1037.

[23] M. Stoer and F. Wagner, *A simple min-cut algorithm*, J. ACM 44,4 (1997), pp. 585–591.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 193–208. http://www.scpe.org
ISSN 1895-1767

OPTIMIZING DATA DISTRIBUTION IN VOLUNTEER COMPUTING SYSTEMS USING RESOURCES OF PARTICIPANTS

ABDELHAMID ELWAER*, IAN J. TAYLOR, AND OMER RANA

Abstract. Many scientific projects use BOINC middleware to build a volunteer computing project. BOINC uses centralized data servers to distribute data to its users and some projects can attract thousands of participants. Such large numbers of users coupled with large datasets can cause a bottleneck for the centralized organization of the BOINC data servers, which has a knock-on effect on the performance of the project as a whole by limiting the throughput of jobs. Alternative methods have been proposed, such as the Attic file system, which decentralize data distribution to BOINC participants. This has been shown to scale but does not attempt to optimize the use of the various distributed data centres being used. We describe performance techniques based on trust algorithms that when layered on the Attic file system can significantly improve data availability and access time through intelligent selection of the data center for each user, based on the optimization of three parameters: trust, current connection speed and availability.

Key words: volunteer computing , P2P, data distribution, trust

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction. Volunteer computing systems provide a useful infrastructure for supporting scientific discovery, examplified through systems such as SETI@Home [16], Einstein@Home [18] , Entropia [17], PlanetLab and recently the Berkeley Open Infrastructure for Network Computing (BOINC) [1]. Each of these systems provide middleware to enable inclusion of donated resources into a particular scientific project. Such distributed computing middleware makes use of idle CPU time of donated volunteered resources within the system. BOINC, for instance, is based on a client/server architecture, where the server generated jobs are pulled onto the clients for local execution and the results are uploaded back to the server upon completion. Although job execution may be supported on remote clients in BOINC (with each job being replicated multiple times to improve reliability), only limited data management support is provided within such systems. The BOINC middleware uses a fixed set of centralized servers to provide data to each client. As some volunteer computing projects may attract thousands of participants (table 1.1), access to a single data server becomes a bottleneck. Additionally, the BOINC data server cannot manage data distribution to all of these participants concurrently, especially when their bandwidth is limited. The BOINC middleware is therefore primarily designed for CPU-intensive jobs, by using the idle CPU time of its participants, but it does not take account of the bandwidth and storages capabilities of the contributing participants.

Projects	Users
SETI@Home	1,174,317
World Community Grid	337,456
Roseta@Home	314,822
Einstein@Home	292,175
Climate Prediction	245,355
MilyWay@home	95,492
ABC@Home	45,734
QMC@Home	45,433
PrimeGrid	40,241
SIMAP	36,693

Table 1.1: The top BOINC projects (from [14])

To overcome this bottleneck and improve data distribution, alternative approaches using Peer-to-Peer (P2P) networks have been proposed. A P2P approach, being far more decentralized in nature, employs a distributed data layer for the participants of the network and can provide far better data-throughput for the project. This

*School of Computer Sciences & Informatics, Cardiff University, Cardiff ,UK, Corresponding Author (a.elwaer@cs.cardiff.ac.uk).

data layer can be extended dynamically, as each volunteer can also now play the role of a data storage server (referred to as a data centre), and thereby also limit the static nature of BOINC servers. Data centres are therefore interim distributed storage facilities that provide a buffer between the data serving application and the client applications. The availability of multiple such data centres reduces access time and improves resilience of the system (at the cost of ensuring that all copies of the data are consistent).

The participants of volunteer computing projects have limited bandwidth and access speed. Furthermore, they are unlikely to overload their internet connection by serving a large number of data requests, due to the availability of limited bandwidth and also due to other applications that they may wish to run concurrently. We propose a framework built on top of the Attic File System (AtticFS) [13] that can offer file-sharing facilities that can control bandwidth and limit the number of connections a participant can serve, as well as the *keep alive* time of these connections. In addition, the framework supports data provisioning and deals with various distributed network conditions that can occur during the distributed allocation of multiple data centres. For example, it is possible for data to become corrupted (with or without the intent of the volunteered resource owner); data centres can also have different upload speeds and their availability can change over time; servers may also become unresponsive or unavailable during the operation. Our framework addresses these issues and allows a client to decide which data centre to download data from (based on preferences identified by the client) using a trust model.

The rest of this paper is organized as follows. Section 2 provides a background to the problem. Section 3 covers related work in P2P data management and trust and Section 4 contains a discussion about the requirements for a system that meets the objectives outlined in the introduction above. Section 5 describes the system architecture and Section 6 describes the integration of the AtticFS system with BOINC. Section 7 contains a description of the associated trust framework and Section 8 contains a number of experiments used to evaluate the framework, by comparing it with a system that does not support trust using the core AtticFS [13] system. We demonstrate that the use of the trust framework makes the resulting volunteer computing environment more predictable and reliable.

2. Background. The context for the approach discussed in subsequent sections is presented. A brief overview of volunteer computing and AtticFS is provided in subsections 2.1 and 2.2 respectively.

2.1. Volunteer Computing. Volunteer computing is a relatively new distributed computing paradigm; it uses computers which are connected to the Internet and donated by their owners for distributed computing application, using the computing and storage capability of each computer. Volunteer computing is currently used in many scientific projects. SETI@home [16] is a scientific project for analyzing radio-telescopic signals to attempt to detect extra-terrestrial activity. Einstein@home [18] is a project to search for spinning neutron stars using data from the LIGO and GEO gravitational wave detectors. Climatprediction.net [19] is a scientific project which aims to produce forecasts of the climate in the 21st century. It makes use of a functional programming approach to analyse large volumes of data. In April 2011, these and other scientific projects using BOINC middleware have recorded 5,620 Teraflops through 2,167,503 users, and the SETI@home project has more than 1,174,317 participants, with a sustained performance of about 533 TeraFlops.

2.2. Attic File System. AtticFS is a decentralized, P2P data sharing software architecture for accessing distributed storage resources available over a network in a similar way to BitTorrent. The AtticFS consists of four main elements: (i) a data serving application that replicates data on the network; (ii) data centres that cache data, providing the distributed data source overlay; (iii) a look up service that keeps track of which data centres have individual data items, and (iv) client applications that download data from data centres on the network. The primary differences between AtticFS and BitTorrent are the concept of data centres and the use of HTTP. Since AtticFS is a distributed P2P file serving system, some peers can play the role of data centres to distribute data to other peers in the network.

Data centres are interim storage facilities that provide a buffer between the data serving application and client applications. This buffer is particularly important for volunteer computing environments because it ensures that the data sources can be trusted by clients. Trust plays a crucial role in volunteer computing environments (and the number of volunteers involved in a project can determine its success or failure), as a client may often need to interact with or download data from a peer that may exist for a short period of time and may be operated by an unknown provider. If trust is broken (i.e. the providing peer does not behave as expected) then the project will fail to attract volunteers and become unsuccessful. Verifying the identity of a peer is also an important concern in this context, hence in *controlled/closed* environments, data centres are

typically issued with certificates signed by a trusted certificate authority (e.g. the project itself) allowing client applications to verify the identity of the data centre when downloading. Alternatively, clients can be configured with known data centre hosts. However, the AtticFS architecture allows any client to also serve data (i.e., become a data centre) and thus the use of trusted data centres is primarily a deployment choice.

Attic is particularly suitable because it has been designed with volunteer computing in mind, and does not require all participants to share data, or share equally, therefore differing it from other types of P2P file-sharing systems e.g. BitTorrent, etc. Attic provides not only a level of security for the data sharing hosts and a managed lookup facility to coordinate data mirrors and results while minimizing stale information, but also, unlike most P2P systems, Attic allows client nodes to forgo "tit-for-tat" algorithms to receive data. This is useful in volunteer computing systems, because many data receiving hosts will not be actively sharing input data with the network (as is required in many P2P systems in order to continue receiving data) for various security reasons. Instead, these clients are acting solely as data recipients and are using the P2P system only to receive data from (potentially) multiple locations. This scenario is of course perfectly valid for BOINC-like projects because these clients are contributing to the overall project by providing CPU cycles, and therefore are not required to also share data to be useful. Further, the use of HTTP for all AtticFS transactions allows easy integration with existing software and firewall configurations and client applications that choose not to server data require only an out-going HTTP connection.

3. Related Work. Two aspects are discussed in this section: data management in P2P systems and the general notion of trust in distributed systems. In section 3.1 we outline related work with reference to the BOINC system also covering related commercial deployments based on BitTorrent and Amazon S3. In section 3.2 we identify how trust has been used to support service provisioning and how the various views on trust can be incorporated into a data management system.

3.1. Desktop Grids, Data Management and P2P Systems. BOINC contains a scheduling server and a client program that is installed on user machines. The client software periodically contacts the scheduling server to report its hardware and availability, and in response receives a *work unit* for downloading and executing a job. Once the client completes the work, it uploads resulting output files to the scheduling server and requests more work. For data distribution, BOINC projects generally use a single centralized server or, in the case of more popular projects, a set of server mirrors, which all have to be configured with the BOINC stack. The centralized nature of this system incurs additional costs on BOINC projects, in both hardware cost and administration and can quickly become inefficient, especially as replication factors increase and many tasks begin to share the same input files.

XtremWeb [2], like BOINC, is a software system that gathers unused resources on donated computers and makes them available to scientific projects. Unlike BOINC, which has centralized servers and users subscribe to a relatively static set of projects for which they will donate their CPU time, XtremWeb allows for multiple-users and multiple (changing) applications to run concurrently on the system. XtremWeb provides a platform on which scientists and volunteers can share their respective resources with one another, and it is often the case that providers of resources are also the users of the system. If one were to think of BOINC as a large, stable, managed, and centralized volunteer computing system for distributing workloads, XtremWeb would be its smaller P2P counterpart that organizes resources in an ad-hoc manner and allows the running of arbitrary code provided it has been signed and verified by a trusted party. Similar to BOINC, work units in XtremWeb are provided with the URIs of input files, and these are downloaded as a preprocessing step when a client job is launched.

There are many popular and proven P2P technologies such as BitTorrent [5], or commercial solutions like Amazon S3 or Google GFS [3], that could be promising alternatives to the current data handling in BOINC and XtremWeb. However, in the case of commercial products, there is a direct monetary cost involved that is often difficult to justify and fund for many scientific projects. Likewise, in P2P systems like BitTorrent, KaZaa, or FastReplica [4], the facility to easily secure or limit who is able to receive, cache, or propagate different pieces of data within the same network is generally limited or non-existent. Like most other P2P systems, these systems discussed above have focused on ensuring conventional file-sharing features that appeal to the broader user-base, such as efficient transfers, equal sharing to promote network stability and availability, and file integrity.

Desktop Grids, and in particular, volunteer computing environments, have significantly different requirements to more conventional P2P file-sharing because security is more complex. In Desktop Grids, it can be a prerequisite that only certain amounts of data are shared with an individual peer, or that strict bandwidth

caps are enforced to guarantee that data-sharing volunteers' Internet connections do not become saturated. It is important to support both data integrity and reliability, whilst also providing safeguards that can limit a peer nodes' exposure to malicious attacks. Furthermore, certain project also require peers that share data to be restricted to certain trusted entities to ensure the distribution can be relied upon. In this paper, we attempt to automate this process and provide safeguards and trust mechanisms that can reliably assess each data server and provide strong cues to the client for choosing a consistent data server at each period of time.

3.2. Trust. The general notion of trust is excessively complex and appears to have many different meanings depending on how it is used in electronic service provisioning. There is also no consensus in the computer and information sciences literature on a common definition of trust, although its importance has been widely recognized and the literature available on trust is substantial. Generally, trust may be used as a metric to guide an agent in deciding how, when and who to interact with. An agent in this context refers to either a service user or a provider. Such a metric takes into account the subjective probability with which an agent views its interaction partners, taking into account local state and external recommendations made by other agents. To establish a trust mode, agents must gather data about their counterparts. This has been achieved in three ways in the literature: (i) using prior interaction experience: in this context, trust is computed as a rating of the level of performance of the trustee using historical data. The trustee's performance is assessed over multiple interactions to check how good and consistent it is at doing what it says it does. Interactions that have taken place recently are treated preferentially to those that have taken place in the distant past. Witkowski et al. [6] propose a model whereby the trust in an agent is calculated based on its performance in past interactions.

Similar to Witkowski et al., Sabater et al. [7] (using the REGRET system) propose a similar model but do not just limit the overall performance to the agent's direct perception, but they also evaluate its behavior with other agents in the system; (ii) information gathered from other agents: trust in this approach is drawn indirectly from recommendations provided by others. As the recommendations could be unreliable, the agent must be able to reason about the recommendations gathered from other agents. The latter is achieved in different ways: (1) deploying rules to enable an agent to decide which other agents' recommendation they give greater preference, as introduced by Abdul-Rahman et al. [8]; (2) weighting the recommendation by the trust the agent has in the recommender, EigenTrust [9] and PageRank [10] are examples of this approach. In both of these approaches, the connectivity graph between recommenders is used to infer trust. Generally, an agent that has successfully delivered its advertised capability and recommends another agent will cause some of its trust to be transferred to its recommended agent.

Both PageRank and EigenTrust are therefore based on the assumption that a general user, searching over a set of possible service providing peers, will eventually end up finding a more trustworthy peer if they follow the recommendation chain from any point in the network. The PowerTrust [11] model works in a similar way to EigenTrust, focusing on creating overlay hashing functions to assign score managers (i.e. peers who calculate trust values for other peers, thereby preventing peers from maliciously changing their own trust values) for peers in the system and for combining trust values to create a global reputation score. Both of these approaches have limited benefit when considering multiple criteria (as considered in this work and as outlined in section 3) when calculating trust, i.e., both EigenTrust and PowerTrust are focused on searching for objects using a single keyword, such as a file name; (iii) socio-cognitive trust: in this context, trust is drawn by characterizing the known motivations of the other agents. This involves forming coherent beliefs about different characteristics of these agents and reasoning about these beliefs in order to decide how much trust should be put in them. An example of this is work by Castelfranchi [12]. Our focus in this work is primarily on characteristics (i) and (ii) defined above. We used historical data to establish a trust model for a given data centre, and use three metrics (honesty, availability and speed) to evaluate the level of trust that one can place in a data centre. We consider reputation to be an aggregated community view about a data provider, i.e. the greater the number of participants who trust a data centre, the greater the reputation the data centre holds.

4. System Requirements. We identify a set of requirements for a framework that makes use of resources provided by participants within a volunteer computing project. The core aim is to subsequently provide a tool based on this framework that is usable by BOINC project participants, allowing them to contribute as a data centre (data distributor) in the decentralized data centre layer, a data worker, or both. Various subsections below identify the components of the framework.

4.1. Data Caching. In BOINC projects, clients contact the scheduler server to get jobs to execute on their local resources, then request input data files from a data server to process the downloaded job, and finally

upload the results. The input data files on a BOINC client can be obtained by using dynamic caching. This increases data availability and improves fault tolerance. Moreover, dynamic caching avoids the bottleneck in the BOINC data server, because data can now be downloade from different places rather than just the data server. Furthermore, it reduces data download time as client can concurrently download data from these different sources. To form such a decentralized data centre layer, AtticFS has been used to add caching functionality to the BOINC client, enabling a BOINC client previously capable of only processing jobs to also be able to cache data and provide it to other clients.

4.2. Trust. The participants of BOINC projects are ordinary internet users, who have different behaviours and connection capability. Therefore, to utilize their resources effectively for data distribution, some optimization or trust mechanisms are needed. In our work we use a trust mechanism that makes use of particular properties of a data centre, such as it's bandwidth, connection speed, availability and other preferences that may be chosen by a client (provided that data associated with these preferences is recorded). The trust mechanism is subsequently used to select one or more data centres to download data from, based on preferences identified by a client.

4.3. Data Management. The formation and use of a decentralized data centre layer requires the consideration of two key issues:

1. **Data Source** For a BOINC client to become a data centre, it has to cache data that is initially provided by the main BOINC data server. When the BOINC client downloads data to process its job, it will cache this data to be available for other clients, who process the same job, thereby propagating the dataset on demand. Here, the BOINC data server is made the primary source of data when data is not available on the data centre layer. When data becomes available on other clients this will extend the source of data and the BOINC client can use them to download data and cache it.
2. **Data Downloading** As the data centres are ordinary internet users (who may be connected to the network using a variety of connection types, e.g. dial-up, DSL, wireless, etc), they can frequently become unavailable to provide data to other BOINC clients. This transient connectivity therefore needs to be addressed in the download algorithm and available data centres need to be dynamically updated as the network evolves. It must also deal with the case that no data centres are available, in which case, the BOINC client should switch to the main source e.g. BOINC data server to get data.

The general scheme for downloading data is outlined in algorithm 4.3.1, which shows that the trust algorithm provides the intelligence for each client to determine the best data centre at that point in time. Each client updates its empirically gathered parameters to feed back into the trust algorithm for the next iteration. In this way, the system learns dynamically to deal with the changing network conditions.

Algorithm 4.3.1 Download Algorithm

```
1: if data available in data centre layer then
2:     Identify trusted data centres using trust algorithm
3:     Download data
4:     Generate feedback for server
5: else
6:     Use BOINC data server for data download
7: end if
8: Cache data locally
9: Register data location with server
```

4.4. Bandwidth Throttling. Many internet users have limited bandwidth and may not be interested in offering this to distribute data – as this can slow their own use of the internet. This framework therefore must offer a throttle capability to a client on its bandwidth use e.g., if it has 1 MB/s connection it can offer 256KB/s to other clients for downloading data, thereby enabling a user to better plan how their capacity will be shared between other users. This enables a volunteered resources to continue to participate in the framework, whilst also enabling a resource owner to continue their own work. We believe such a mechanisms for bandwidth sharing is likely to increase contribution of resources to a project.

5. System Architecture. Our trust system has four general layers, which are provided in Figure 5.1. The bottom layer represents the participants of scientific projects who provide their resources to volunteer

computing projects. The next layer provides the P2P network capability – here we make use of AtticFS. This layer provides the core capability for volunteers to share data with other participants.

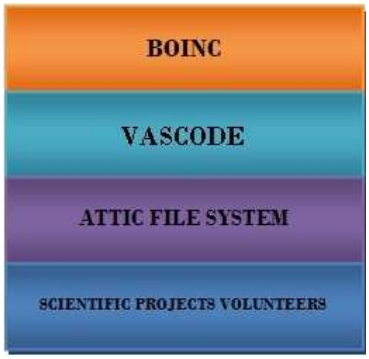


Fig. 5.1: FRAMEWORK LAYER

The third layer provides the *Volunteered Atomic Servers for Data Collection and Optimization in Distributed Environments* (VASCODE) trust framework for choosing he location of data download at each time step. In layer three therefore we integrate a data collection layer, which provides peers the necessary tools to select from which data server to download data, which peers to trust, and the throttling capabilities to manage their bandwidth. This layer is described in detail in the section 5.1.

Finally the last layer represents BOINC, which is the programming and Web interface that a project interacts with in order to the use the system. Since both AtticFS and VASCODE deal with HTTP endpoints, a simple `attic` URL scheme plug-in can be used to switch out the general BOINC URL endpoint with a dynamic AtticFS one to provide multiple possible endpoints for each dataset. Therefore, a project does not need to be aware of the use of VASCODE and AtticFS in order to have it enabled as the data distribution mechanism. The integration of AtticFS and VASCODE into existing projects is possible through a plug in that proxies the HTTP connections and resolves multiple endpoints available in AtticFS.

5.1. VASCODE. VASCODE is a layer built upon the capability provided by AtticFS to add the necessary functionality to its peers (e.g. worker or data centre) when they download data or distribute it. Figure 5.2 provides an overview of this functionality. Essentially, VASCODE provides the trust-based mechanisms within AtticFS, enabling the better integration of various components that make up the AtticFS system into BOINC. VASCODE enables user defined preferences to be taken into account when selecting data centres to download from, based on previous usage data that has been recorded about these data centers.

The three user defined preferences employed in this work to demonstrate the concept include: availability, data integrity (i.e. establishment of trust in that the data has not been altered) and connection speed of each data centre. Each peer in the system provides feedback on each interaction they have had with a data centre. This data is collected (through the data lookup nodes) and used in the trust algorithm to feedback into a selection criteria a client uses, to determine the most appropriate sets of data centres to use. By using this mechanism, the system continuously (i.e. on the completion of each interaction with a data centre) updates the aggregate statistics for each data centre. We can modify whether the aggregation must be undertaken after each interaction, or only after a pre-defined number of interactions, in order to reduce the computational overhead of calculating the aggregate value.

Fig. 5.2: VASCODE Layer on top of AtticFS

Figure 5.2 shows the three possible roles that a peer can perform in the BOINC-VASCODE integration: a conventional data worker, a data centre or a data lookup server. Further, each of these roles has various properties. For example, when the BOINC client is a data worker, it must also be capable of accessing the data centre layer. It also needs to make use of VASCODE to calculate the trustworthy of these data centres and select which data centres to get data from. Furthermore, it needs to then provide feedback to help other participants determine which is the best data centre at this point in time.

A BOINC client can use the VASCODE framework to interact with these functions. If a BOINC client wants to perform data centre capabilities, then they need to provides attributes such as, (i) how much bandwidth it wants to offer for data distribution, (ii) the maximum number of connections it will serve, and (iii) the keep alive time for these connections, for the general configuration of the data centre. Finally, when the BOINC client plays the role of a data lookup server, it is important to have the ability to collect feedback from clients and distribute these to other clients for use by the network as a whole. These capabilities are also offered through our VASCODE layer.

6. Integrating AtticFS with BOINC Projects. AtticFS uses a unique *ID* to identify data instead of using a file name. Data is first published using AtticFS to obtain the new ID and then the resolved `attic` URL using this ID is deployed in the generated BOINC work unit. A snapshot of a `DataDescription` of data published in the AtticFS is shown below:

```
<DataDescription xmlns="http://Attic.org">
    <id>
        f050a833-2fac-44c9-9b08-7c
    </id>
    <name>
        file10MB.dat
    </name>
            ......
            ......
</DataDescription>
```

The download URL in a work unit is changed to use an `attic` URL scheme instead of `http`. This indicates to the BOINC client when it parses the workunit description to use AtticFS to resolve the endpoint that will be used to download the actual data. This is illustrated in Figure 6.1.

1. The BOINC client contacts the Task Server to get a workunit.
2. Then it parses the work unit and resolves the data identifier by contacting the data look up server to get a list of data HTTP endpoint from data centres.
3. The BOINC client then uses VACSODE to determine the best endpoint to use at this point in time and contacts the data centres to start downloading the data.

A snapshot of a `workunit` using AtticFS is shown below

```
<file_info>
  <name>f050a833-2fac-44c9-9b08-7c</name>
  <url>
  attic://host/cplan/download/f050a833...9b08-7c
  </url>
  <md5_cksum>92d8c8...b0250e5</md5_cksum>
  <nbytes>10485760</nbytes>
</file_info>
<workunit>
  <file_ref>
    <file_name>f050a833...9b08-7c</file_name>
    <open_name>in</open_name>
</file_ref>
  .......
  .......
</workunit>
```

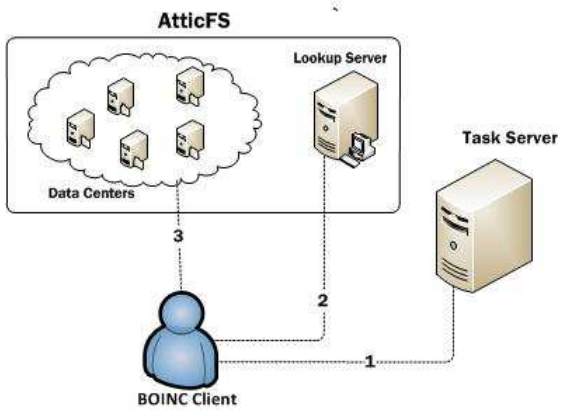


Fig. 6.1: Integrating AtticFS in BOINC Project

6.1. Baseline Comparison with BOINC. In this section, we make a comparative evaluation of BOINC with AtticFS to determine a baseline for the further experiments described later in the paper. We undertake two experiments to carry out this comparison and the result of this comparison are provide in Figure 6.2.

In the first experiment, we employ the use of a BOINC data server. A different number of data clients (1, 3 and 9) are used to download a 10 MB file. The network speed is set to 10 Mbps for the server and 100Mbps for the clients. We analysed the performance of the BOINC server as a baseline, the result of this part of the experiment shows that the average download time needed by all clients to download the 10 MB file increases as the number of clients increase.

In the second experiment, AtticFS is used to download a 10 MB file, published using a 1 MB chunk size. The number of data centres is set to 1, 3 and 9. The network speed in this experiment was 10 Mbps for the servers and 100 Mbps for the clients. We ran the experiment three times using a different number of clients (1, 3 and 9). The results of this part of the experiment show the download time for each client using a different number of data centres (1, 3 and 9).

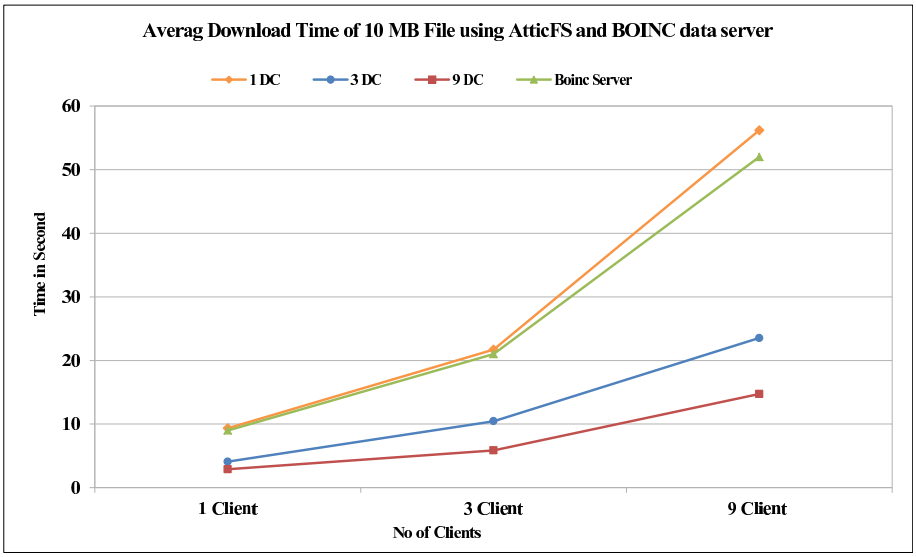


Fig. 6.2: Average Download Time 10 MB File Using AtticFS and BOINC Data Server

By using one data centre, the clients download time increases as more clients are added to the network, because of the limited network bandwidth of the server. The download time increases as more clients attempt

to download from the same resource creating a queue. Increasing the number of data centres by a factor of three enables clients to concurrently download chunks. Since the three data centres are shared between the clients this means the waiting time will decrease. For the nine data centres case, we see the largest improvements and the most consistency between the clients. By comparing the results of this experiment in Figure 6.2, we notice that AtticFS will give almost the same results as a BOINC data server when one data centre is used. In fact, BOINC is marginally more efficient, as it does not have the message overhead that AtticFS has, in terms of querying and prioritising endpoints before downloading commences. However, as AtticFS adds data centres, the download time decreases.

7. The VACSCODE Trust Framework. In comparison with the general BOINC system (which uses a pre-defined data server), a client in our system is required to identify a data centre prior to commencing data download. The selection of data centres is based on their reputation in the system, with reference to one or more metrics. The metrics are:

- the upload speed that a client obtained through a connection with a data server
- the availability of a particular data centre
- and the integrity of data supplied by the data centre

These metric are named in the framework as **DCspeed**, **DCAvailability** and **DCHonesty**, respectively. Each metric is independently calculated using feedback from multiple clients using our specialist tools. We have chosen these particular metrics because of their dominance in P2P literature, and in supporting job execution in volunteer computing systems.

Speed primarily relates to performance issues such as access time, latency and effective bandwidth. Availability relates to uptime and resilience, covering aspects such as downtime, and failure rate. Honesty covers aspects such as data integrity and quality, storage reliability and any malicious modification to the data. The trust framework makes use of the AtticFS (discussed in section 3.1) to support concurrent data downloads from multiple data centres. It utilizes the communication between the clients and the data lookup server to send feedback and receive data on the associated trust metrics.

The trust framework (Figure 7.1) involves components that operate on both the client and the server; the client generates feedback, processes trust values and selects data centres based on its preferences; the server collects client feedback, updates the reputation database and provides reputation data to the clients.

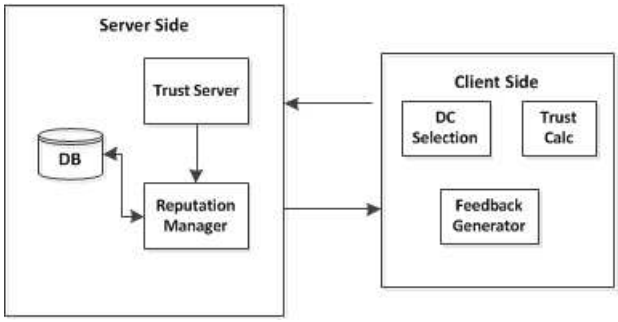


Fig. 7.1: Trust Framework extending AtticFS

In our framework trust is calculated by using feedback from the multiple clients that interact with a data centre using a Beta distribution [15], as outlined in equations 7.1 and 7.2, specifying whether the client was satisfied (r) or not satisfied (s) with the download from a data centre. The Beta distribution has been used to take into account this combined assessment (i.e. by considering both satisfied and not satisfied) from multiple clients, rather than only consider the positive outcomes (i.e. the number of times that a client has been satisfied with the download). After a client completes downloading data, it provides a subjective assessment of each of the three metrics (availability, honest and speed) for each data centre that has been used by this client. This public feedback can then subsequently be used by other clients, to support their decision about which data centres to trust, using equations 7.1 and 7.2.

$$x = \frac{r+1}{r+s+2} \tag{7.1}$$

$$T = a.TAvailability + b.THonesty + c.TSpeed,\ where\ a+b+c=1 \tag{7.2}$$

When a client calculates the total trust value of each data centre it uses algorithm 7.0.1. As outlined in the algorithm, a threshold referred to as `trustThreshold` is used to limit the number of data centres that have been returned as an outcome of a search. The client can either modify this parameter themselves or set the minimum number of data centres (referred to as `minDC` in algorithm 7.0.1) they would prefer to obtain to download from (i.e., the total number of data centres that match their particular trust criteria). In algorithm 7.0.1, the threshold value is set by a client to be 0.9. If an automated approach is used, i.e. where a client does not specify the threshold but instead identifies the minimum number of data centres they would prefer to download from, this threshold value is automatically adjusted.

Algorithm 7.0.1 Data Centre Selection

```
1: selectedDataCentres = 0;
2: trustThreshold = 1.0;
3: decrement = 0.1;
4: minDC=3;
5: loop
6: for each DataCentre[i] do
7:    if TotalTrustValue[i] ≥ trustThreshold then
8:       selectedDataCentres = selectedDataCentres + 1;
9:       return [i] ;
10:   end if
11: end for
12: if selectedDataCentres ≤ minDC then
13:   trustThreshold = trustThreshold - decrement;
14:   goto loop
15:   else exit();
16: end if
```

8. Experiments.

8.1. Availability Experiments. In a volunteer computing environment peers can enter and exist the network at any time. In AtticFS, as peers can also play the role of a data centre, the availability of these data centres can change over time. With the distributed servers appearing and disappearing, a mechanism is required to limit this variability in the network so that a clients download efficiency is maximised. We conducted a series of experiments to show how the download bandwidth is improved when the trust framework is utilised.

In the first experiment, Attic FS consists of a lookup server and 10 data centres. The 10 data centres have a 10Mb/s connection and are all deemed honest peers (i.e. no malicious behaviour) for this experiment. 20 modified AtticFS clients (running on 20 Linux machines) were used to mimic data requests, which were setup to request a 10MB file at periodic intervals. The clients report their download experience to the lookup server when they finish downloading data. This feedback to the server contains the status of each data centre and information about whether they were available, whether the download speed was satisfactory and whether they were honest. A Poisson distribution was used to simulate the availability of data centres – as this represents a realistic scenario from many existing P2P systems. Figure 8.1 shows the distribution of the data centres and when they are online and offline over a four-hour period. The total duration of the experiment was eight hours, only the first four hours have been shown here to demonstrate the overall availability trend.

We compare the behaviour of the AtticFS client with and without the trust framework. One instance of each type of client are used, each requesting data periodically every five minutes during the experiment. The modified

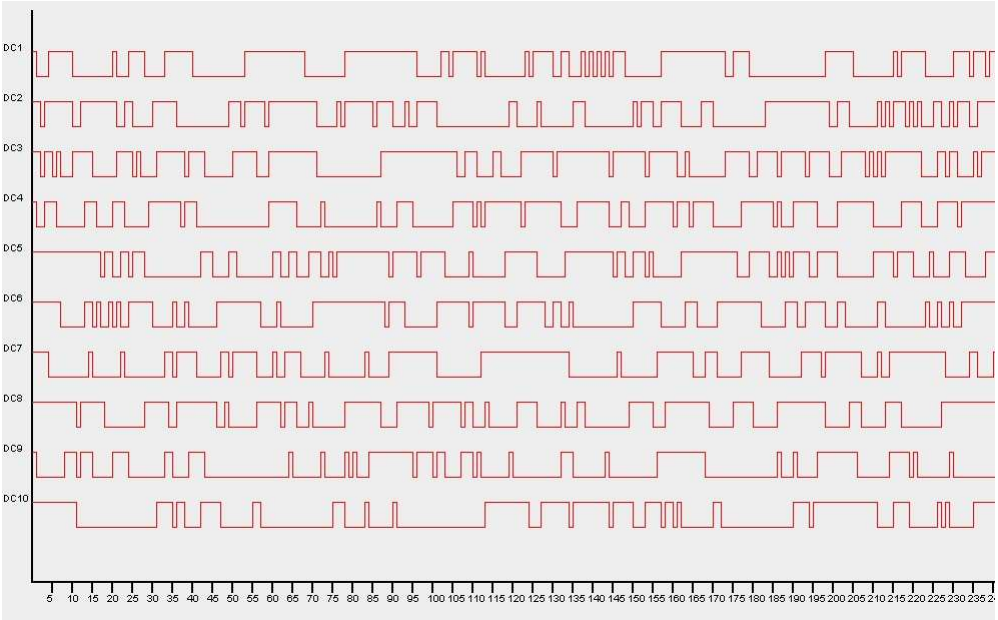


Fig. 8.1: Data centres Availability

AtticFS client is configured with the following parameters: Availability weight factor (AWF)=1, Honesty weight factor (HWF) = 0, and Speed weight factor (SWF)=0, as we are only interested in the availability of the 10 data centres, the other weight factors are set to zero.

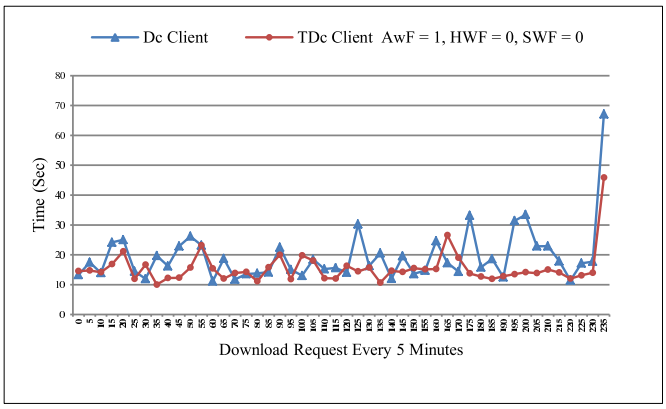


Fig. 8.2: Data Download using AWF in the first four hour period

This experiment had a duration of eight hours, the data centre availability in the first four hours is the same as in the second four hours as shown in (Figure 8.2), and we found that during the first four hours of the experiment, the download time of both clients is similar see (Figure 8.3). However, in the next four hours, our trusted data centre has an improved download time as the trust algorithm converges and learns the state of the network i.e. our trusted data centres learn to avoid the unavailable data centres as the experiment progresses. In Figure 8.3, it can be observed that the behaviour of nodes employing the use of the trust algorithm becomes smoother and more predictable during the last four hours of the experiment. We believe this convergence shows promise as the algorithm can adapt to network conditions and variability in node availability, which is a requirement of the volunteer computing environment as a whole.

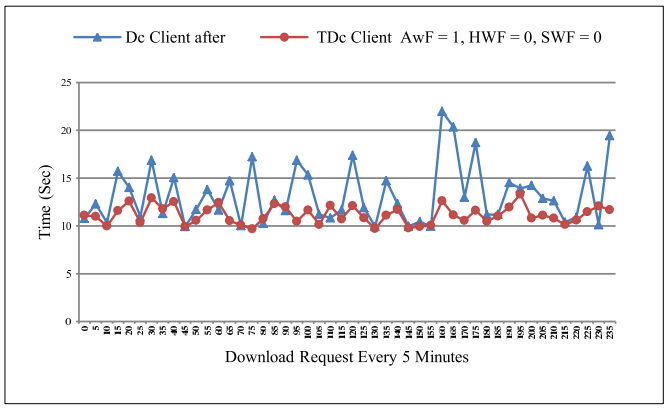


Fig. 8.3: Data Download using AWF in the second four hour period

8.2. Honesty Experiments. In this experiment, malicious data centres are injected into the network in order to disrupt the system. These nodes intentionally provide corrupted data to their clients in order to attack the system. We designed an experiment to show how our trust framework can become fault tolerant to this malicious behaviour and avoid the use of malicious data centres to recover and repair the corrupted network. This experiment focuses on the honesty of data centres and how this affects the download time. It aims to show how the client which uses our trust framework offers better stability and hence increased download efficiency than the ordinary AtticFS client. Note that since an MD5 hash is taken of the data, malicious peers can only slow down the network because if the hash of the downloaded file does not match the original hash of the data, it will be discarded and downloaded again. Our framework detects these malicious peers and effectively removes them from a clients download list thereby making significant gains overall.

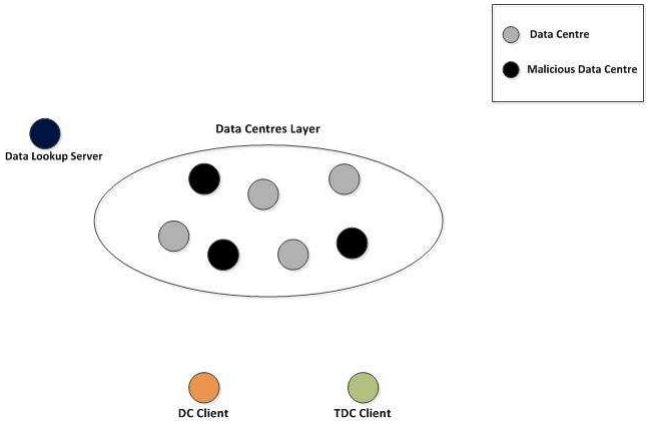


Fig. 8.4: Attic File System using Different Data centres Behaviour

Seven data centres are used in this experiment (Figure 8.4). Three of them are honest data centres and the other three are malicious and send corrupted data to their clients. Because we are interested in the honesty of these data centres, they are available throughout the duration of the experiment, this experiment is repeated three times every time the data centres uses the throttle functionality to configure their speed connection to (128 KB, 256 KB and 512 KB). The modified AtticFS client is configured with the following parameters (AWS = 0, HWF = 1, SWF = 0). As identified in Experiment 1 for availability, only the honesty weight factor is set to 1 in this experiment.

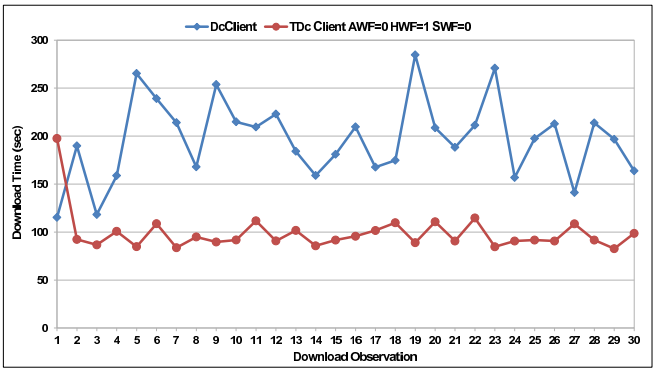


Fig. 8.5: 128 KB upload speed

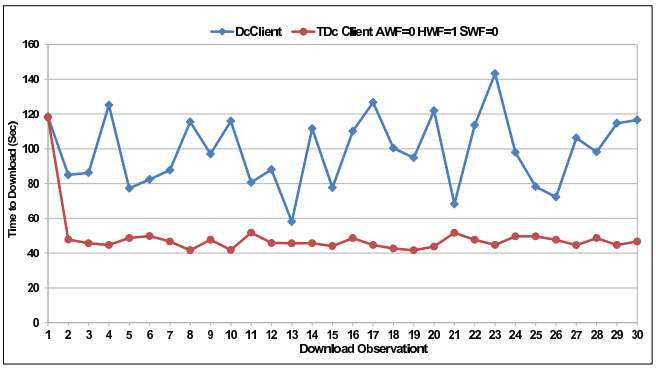


Fig. 8.6: 256 KB upload speed

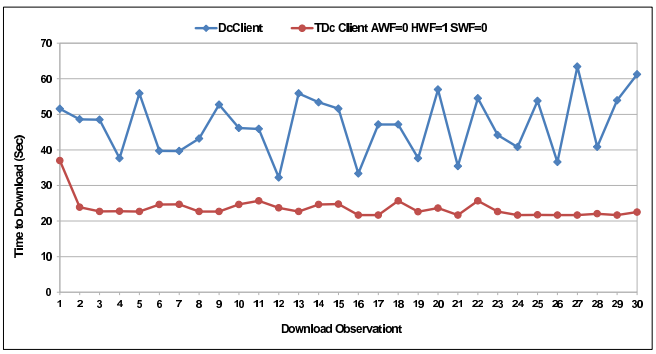


Fig. 8.7: 512 KB upload speed

Figures (8.5, 8.6, 8.7) show the result of this experiment. It can be observed that our trusted client has significantly better download time because it avoids the malicious data centres, while the ordinary AtticFS client uses all the data centres and therefore downloads a number of unnecessary corrupted chunks, which need to be reloaded – thereby incurring a download overhead. After a short period of convergence, our system performs on average 3 times better than the standard AtticFS approach. We believe that this shows a huge potential as it addresses one of the fundamental issues in volunteers distributing data, which is the ability to trust third party peers. This experiment indicates that our system can learn to avoid malicious peers and dynamically select more trusted peers in the network, which opens up the possibility for such a dynamic data distribution approach.

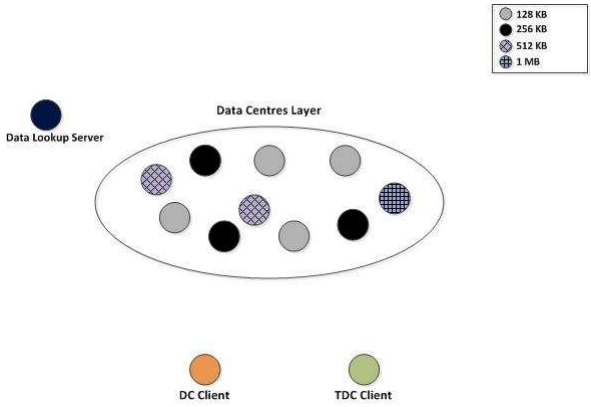


Fig. 8.8: Different upload speed

8.3. Speed Experiments. The data centres used in AtticFS can choose different upload speeds using the throttle functionality and this obviously affects the download time of each peer. Since clients are interested in getting data in the fastest possible way, they should obviously choose those data centres with high-speed connections. However, if all peers download from the same fastest peers then there will be a bottleneck, so any algorithm must dynamically optimise the distribution of clients connected to a data centre at each time-step during the operation of the system. Our trust framework provides such a mechanism by choosing data centres with high bandwidth at that point in time in order to optimise the throughput of the distributed system as a whole. In this experiment therefore we are interested in how the trust framework is used to choose the data centres which have the highest bandwidth connections at any point in time. For this experiment, 10 data centres are used (Figure 8.8). The upload speed of these data centres are configured using throttli ng functionality to configure the upload speed of each data centre as mentioned in (Table 8.1). The data centres have continuous availability and they all act honestly. The modified AtticFS client is configured with the following parameters (AWS = 0, HWF = 0, SWF = 1) to configure the system to only focus on the speed of the data centres.

No of Data centres	Bandwidth
1	1 MB
2	512 KB
3	256 KB
4	128 KB

Table 8.1: Different upload Speed

The results in Figure 8.9 show significant improvements by clients making use of the trust framework, compared to the conventional AtticFS approach, achieving download speeds several times faster overall. Even at this small-scale proof-of-concept testbed, these results demonstrate the benefit of the approach.

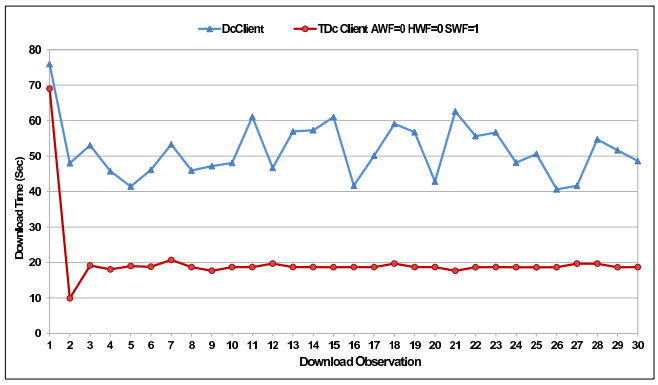


Fig. 8.9: Different upload speed

9. Conclusion and Future Work. The experiments described in this paper show how to extend and improve data distribution in volunteer computing projects using volunteers resources . Furthermore, they show significant improvements with respect to a clients download time when the trust model is used compared to the conventional AtticFS scheme. By comparing against AtticFS which makes use of multiple concurrent downloads using file chunks, rather than a conventional downloading mechanism based on a complete file, we are able to evaluate our approach against an already optimized system. Although the testbed we used is small, it employs the use of machines on a real network rather than taking a simulated approach where it is typically not possible to record a direct observation of the system. We believe these experiments show extremely encouraging results that can be investigated further. We are planning to perform experiments using this system using higher numbers of network sizes and also to investigate different combinations of the trust algorithms parameters in order to achieve an optimal balance of network behavior and performance.

REFERENCES

[1] D. P. Anderson, *BOINC: A System for Public-Resource Computing and Storage*. In: 5th IEEE/ACM International Workshop on Grid Computing, pp. 365–372. Pittsburgh, USA (2004)
[2] F. Cappello et al., *Computing on Large-Scale Distributed Systems: XtremWeb Architecture, Programming Models, Security, Tests and Convergence with Grid*. Future Generation Computer Systems 21(3), 417–437 (2005)
[3] S. Ghemawat, H. Gobioff, S. T. Leung ,*The Google File System*. SIGOPS Oper. Syst. Rev. 37(5), 29–43 (2003). DOI http://doi.acm.org/10.1145/1165389.945450
[4] L. Cherkasova, J. Lee, *Fastreplica: Efficient large file distribution within content delivery networks*. In: Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems (USITS). Seattle, WA, USA (2003)
[5] B. Cohen, *Incentives build robustness in BitTorrent*. In P2PEcon (2003).
[6] M. Witkowski, A. Aritikis, and J. Pitt, *Experiments in building experiential trust in a society of objective-trust based agents*. Trust in Cyper-societies, pages 111-132, 2001
[7] J. Sabater and C.Sierra,*Regret: a reputation model for gregarious societies*. Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agents Systems, 2002.
[8] A. Abdul-Rahman and S. Hailes. *Using Recommendations for managing trust in distributed systems*. Proceedings IEEE Malaysia International Conference on Communication, 1997.
[9] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. *The Eigentrust algorithm for reputation management in P2P networks*. Proceedings of the Twelfth International World Wide Web Conference, 2003.
[10] L. Page, S. Brin, R. Motwani, and T. Winograd, *The Pagerank citation ranking: Bringing order to the Web*. Stanford Digital Library Technologies Project, 1998.
[11] R. Zhou and K. Hwang, *PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing*, IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 4, pp 460–473, 2007.
[12] R. F. C. Castelfranchi, *Principles of Trust for Multi-Agent Systems: Cognitive anatomy*, social importance and quantification. Proceedings of the International Conference on Multi-Agent Systems, 1998.
[13] Ian Kelley and Ian Taylor, *Bridging the Data Management Gap Between Service and Desktop Grids*. In: Distributed and Parallel Systems In Focus: Desktop Grid Computing, Peter Kacsuk, Robert Lovas and Zsolt Nemeth (Editors), Springer, 2008.
[14] BOINC statistics, `http://www.boincstats.com`. Last accessed: April 2011.
[15] A. Jsang, R. Ismail , *The beta reputation system*, in: Proceedings of the 15th Bled Electronic Commerce Conference, Bled, Slovenia, 2002.
[16] The SETI@Home project, `http://setiathome.berkeley.edu`. Last accessed: March 2011.

[17] Entropia project, http://www.entropia.com. Last accessed: March 2011.
[18] Einstein@Home project, http://einstein.phys.uwm.edu. Last accessed: March 2011.
[19] The Climateprediction.net project, http://climateprediction.net. Last accessed: April 2011.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 209–226. http://www.scpe.org
ISSN 1895-1767

HBASESI: MULTI-ROW DISTRIBUTED TRANSACTIONS WITH GLOBAL STRONG SNAPSHOT ISOLATION ON CLOUDS

CHEN ZHANG* AND HANS DE STERCK†

Abstract. This paper presents the "HBaseSI" client library, which provides global strong snapshot isolation (SI) for multi-row distributed transactions in HBase. This is the first strong SI mechanism developed for HBase. HBaseSI uses novel methods in handling distributed transactional management autonomously by individual clients. These methods greatly simplify the design of HBaseSI and can be generalized to other column-oriented stores with similar architecture as HBase. As a result of the simplicity in design, HBaseSI adds low overhead to HBase performance and directly inherits many desirable properties of HBase. HBaseSI is non-intrusive to existing HBase installations and user data, and is designed to be scalable across a large cloud in terms of data size and distribution.

Key words: distributed transaction, cloud database, HBase, snapshot isolation

AMS subject classifications. 68U01, 68N01, 68M01, 68P01

1. Introduction. Column-oriented data stores (column stores) are gaining attention in both academia and industry because of their architectural support for extensive data scalability as well as data access efficiency and fault tolerance on clouds. Data in typical column stores such as Google's BigTable system [3] are organized internally as nested key-value pairs and presented externally to users as sparse tables. Each row in the sparse tables corresponds to a set of nested key-value pairs indexed by the same top level key (called "row key"). The second level key is called "column family" and the third level key is called "column qualifier". Each column in a row corresponds to the data value (stored as an uninterpreted array of bytes) indexed by the combination of a second and third level key. Scalability is achieved by transparently range-partitioning data based on row keys into partitions of equal total size following a shared-nothing architecture. These data partitions are dispatched to be hosted at distributed servers. As the size of data grows, more data partitions are created. In theory, if the number of hosting servers scales, the data hosting capacity of the column store scales. Concerning data access, at each data hosting server, data are physically stored in units of columns or locality groups formed by a set of co-related columns rather than on a per row basis. Column stores derive their name from this property. This makes scanning a particular set of columns less expensive since the data in other columns need not be scanned. Persistent distributed data storage systems (for example, with file replication on disk) are normally used to store all the data for fault tolerance purposes.

Column stores provide database-like table views, and it would be desirable if distributed transactions can be supported on them so that applications that used to be built around traditional database management systems (DBMS) can make use of cloud column stores for transactional data processing, with improved scalability. Indeed, many applications, such as a large number of collaborative Web 2.0 applications, would benefit from transactional multi-row access to the underlying data stores [1]. In fact, those modern applications pose high requirements on scalability and fault tolerance and there are currently no existing DBMS solutions (even parallel database systems) to fully cater to those requirements due to the overhead of managing distributed transactions and the fact that it is impossible for DBMSs to guarantee transactional properties in the presence of various kinds of failures without limiting system scalability and availability [1, 4, 7]. Unfortunately, no out-of-the-box support for transactions involving multiple data rows exists in column stores. This is mainly because multi-row transactions in column stores are intrinsically distributed transactions [6] and traditional approaches would suffer from similar problems as in existing distributed DBMS solutions.

This paper presents a novel light-weight transaction system with global strong snapshot isolation on top of HBase (which is a representative open source column store modeled after Google's BigTable system), without using traditional methods of handling distributed transactions, such as standard 2-phase commit protocols, consensus-based commits, atomic broadcast, or explicit data locking. A preliminary version of our system, providing weak SI for HBase, was presented in [11]. HBaseSI, described in this paper, recycles some of the design principles of the initial system from [11] but uses a different, more efficient solution for handling distributed synchronization, with added support for global strong (and not weak) SI and an efficient failure handling

*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada (c15zhang@cs.uwaterloo.ca).
†Department of Applied Mathematics, University of Waterloo, Waterloo, Canada (hdesterck@uwaterloo.ca).

mechanism. Our work in [11] described the first ever SI system for column-stores. Independently and at the same time, the Google Percolator system was presented in [8]. Percolator provides global strong SI for Google's column store system, BigTable. Percolator shares many design principles with our SI system, but there are also many important differences in design goals. The current paper extends and improves the system for SI in HBase that we presented in [11].

The solution presented in this paper is called "HBaseSI". HBaseSI targets the same type of OLTP(Online Transaction Processing) workloads as HBase, taking advantage of HBase's random data access performance comparable to open source database systems such as MySQL. It is implemented as a client library and does not require any extra programs to be deployed or running in addition to existing HBase servers. In addition, HBaseSI is non-intrusive to existing user data that have already been stored in HBase since it does not require modifications to existing user data tables. In HBaseSI, transactional management meta-data are written by each transaction to a separate set of HBase tables. There is no central "commit engine" that decides which of the transactions that are ready to commit can actually commit; instead, the transaction processes decide autonomously, in a distributed fashion, whether they can commit or have to fail, using the information stored in the additional meta-data HBase tables. As a result, little performance overhead pertaining to distributed synchronization is added by the transactional management logic. Many of HBase's desirable properties are directly inherited as well, such as fault tolerance, access transparency and high throughput. Concerning scalability, the design target of HBaseSI is to be fully scalable across a large cloud in terms of data size and distribution. In its current design, HBase does not target scalability in terms of the number of transactions per unit of time.

This paper is structured as follows: in Section 2 we introduce some background information about snapshot isolation and HBase. In Section 3 we describe the design and implementation of HBaseSI in detail. In Section 4 we evaluate the performance of HBaseSI. Section 5 gives comparison to related work, and section 6 concludes.

2. Background.

2.1. Snapshot Isolation. For our purposes, we can describe Snapshot isolation (SI) as follows.

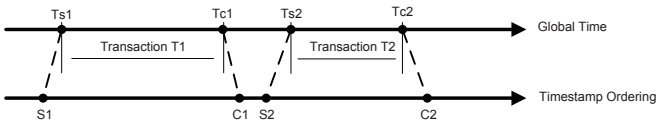


Fig. 2.1: Illustration of SI.

A transaction Ti acquires a start timestamp, S(Ti), at the beginning of its execution (before performing any read or update operations), and acquires a commit timestamp, C(Ti) at the end of its execution (after finishing any read or update operations). We will also use the shorthand notation Si=S(Ti) and Ci=C(Ti) in what follows. The timestamps Si and Ci are ordered: they inherit their ordering from the ordering of real global times Tsi and Tci to which they correspond (Fig. 2.1). This ordering implies in particular that all read and write operations of Ti happen (in real, global time) after the time corresponding to Si, and all write operations of Ti happen (in real, global time) before the time corresponding to Ci. Transactions Ti and Tj are called concurrent if their lifespan intervals (Si,Ci) and (Sj,Cj) overlap. A transaction Ti that commits successfully is called a successful or committed transaction.

Global Strong SI can then be described as follows. A transaction history H satisfies global strong SI, if its (successful) transactions satisfy the following two conditions: 1. Read operations in any transaction Ti see the database in the state after the last commit before Si. In other words, all updates made by the committed transaction Tj which has the last Cj <= Si are visible to Ti. However, read operations in transaction Ti that read data items that have previously been written by transaction Ti itself, see the data values that were last written by Ti; 2. Concurrent transactions have disjoint writesets.

We add the qualifier 'global' when we define strong SI because we want to investigate Snapshot Isolation for a distributed system in this paper, and want to stress that the definition above applies to the global system. Additionally, the above definition does not regulate the behavior when two concurrent transactions with overlapping writesets both try to commit. In many occasions, a rule called "first committer wins" is employed, which will cause the failure of the transaction that is second in attempting to commit. To better illustrate this rule, let's look at an example set of transactions shown in Fig. 2.2. T1 and T2 must have disjoint

writesets in order to both commit successfully. If they have overlapping writesets, only T1 will successfully commit and T2 will abort, because T1 attempts to commit before T2.

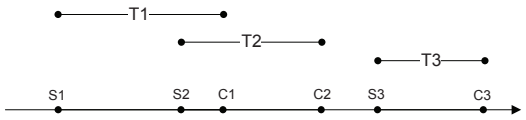


Fig. 2.2: An example SI scenario.

The strong notion of SI as defined above is different from the original definition of SI [2], which allows Si to be chosen corresponding to any time in the past before the first read or update operation in transaction Ti. This relaxed version of SI is also called weak SI in [5]. To illustrate this difference, we assume that T1 and T2 in Fig. 2.2 have disjoint writesets and both commit successfully. According to the definition of strong SI, T3 must see all the committed results as of timestamp S3, which include the commits for both T1 and T2. However according to weak SI, it is allowed that T3 use a snapshot between C1 and C2 that includes only the committed results from T1. Typically, versions of the strong notion of SI are implemented in stand-alone, non-distributed commercial databases. SI is not included in the ANSI/ISO SQL standard but versions of it are adopted by major DBMSs due to its better performance than Serializablity at the cost of having a potential write-skew anomaly [2].

2.2. HBase. HBase is a column-oriented store implemented as the open source version of Google's Big-Table system. Rows in each table are automatically sorted by row keys. The data value for each row-column combination is uniquely determined by the row key, column and timestamp. The timestamp facilitates multiple data versions. Timestamps are either explicitly passed in by the user when the data value is inserted, or implicitly assigned by the system. Each table is horizontally partitioned into row regions and each region is hosted by a distributed "region server" with region data stored in persistent storage (Hadoop HDFS, which stands for Hadoop Distributed File System). Currently, only simple queries using row keys and timestamps are supported in HBase, with no SQL or join queries. It is also possible to scan and iterate through a set of columns row by row within a row range. The scalability of HBase is attributed to the shared-nothing architecture of data regions hosted by distributed region servers. However, there could still be bottlenecks in the system in the case when a single region server gets overloaded by too many requests on the same data region. In fact, at each region server, all the read/write requests to a particular row in a table region are serialized.

Our choice of using HBase as the basis for investigating transactional SI solutions for clouds is not random. HBase enjoys several nice properties that are important for simple and efficient SI implementations. First, HBase offers a single global system view with access transparency, meaning that clients access all the HBase tables as if they are hosted at a centralized server without knowing that they are actually contacting different distributed region servers for fractions of data. This significantly reduces the complexity of transactional protocol implementation. Second, HBase provides multi-version data support distinguished by the timestamp the data item is written with. This feature can directly facilitate the SI protocol implementation. Third, HBase guarantees single atomic row operations (reads/writes) with strong data consistency in the global table view.

3. HBaseSI.

3.1. System Design. A major design principle of HBaseSI is to provide transactional SI for HBase with minimum add-ons to existing HBase installations and administration. It may also be an advantage if it is possible for HBase users with existing data tables to employ HBaseSI with minimum effort. To this end, HBaseSI is implemented as a client library in Java with no extra programs to be deployed. Applications that need to do transactions use the client library to interact with HBase instead of using the standard HBase API. Each transaction writes its own transactional meta-data (e.g. transaction ID, commit timestamp, commit request, etc.) to a set of global system HBase tables (separate from existing user data tables), and in the meantime, queries those tables to obtain information about other transactions. Based on the information obtained, and by accessing this information with atomic read/write operations provided by HBase, a transaction can autonomously decide to commit or abort. From a user's point of view, using HBaseSI requires no modification to any existing data tables.

Table 3.1: W counter table. W stands for "HBase write timestamp".

Row Key	Counter
W	86

Table 3.2: R counter table. R stands for "commit request ID".

Row Key	Counter
R	78

HBaseSI employs several HBase tables in addition to the user's data tables. These additional system tables are three Counter tables (Tables 3.1, 3.2 and 3.3) for providing globally unique counters, a CommitRequestQueue table (Table 3.4) that acts as a queue for transactions that are submitting requests to commit, a CommitQueue table (Table 3.5) that acts as a queue for transactions that are cleared to commit, and a Committed table (Table 3.6) that keeps track of successfully committed transactions and their writesets.

The Counter tables are intended to serve as a set of centralized locations for issuing globally unique IDs that may be used as well-ordered counters. Each of the tables is a single-row-single-column table. The HBase incrementColumnValue function is used on the column "Counter" to dispense globally unique and strictly incremental time labels to transactions atomically. The W Counter table (Table 3.1) issues a unique ID to each transaction at the start of the transaction. W stands for "write timestamp". This ID will be used as the unique ID for the transaction, and as HBase timestamp when writing data to HBase tables (note that in this paper we use two types of timestamps: "HBase write timestamps" are used as write timestamps for HBase to distinguish different data versions, and "transaction timestamps" are timestamps used for transaction ordering purposes). The order of the W counter is not important, as long as each W counter is unique. The R Counter table (Table 3.2) issues unique commit request ordering IDs dispensed to transactions that are attempting to commit, establishing an order among the transactions attempting to commit, which is, among other things, used for enforcing the "first-committer-wins" rule [2]. R stands for "commit request ID". The C Counter table (Table 3.3) issues the final unique commit timestamps, each of which is used as the actual commit timestamp of a transaction. Different from W counter values, the strict global ordering of the R and C counter values is very important to the correctness of the HBaseSI protocol.

The CommitRequestQueue table (Table 3.4) is used as a queue for ordering commit attempts and checking for conflicting updates among concurrent transactions that try to commit at almost the same time. A transaction Ti, when trying to commit, enters this queue table by first inserting a row containing its unique transaction ID Wi (obtained from the "W Counter table") as the row key and its writeset as the columns. (The writeset column names are unique identifiers for the data locations in the user data tables.) After this row is inserted, the transaction requests and obtains a commit request counter value Ri (from the "R counter table") and enters Ri into the "RequestOrderID" column of its row. The sequence of first inserting a row, then getting a Ri counter value, and finally putting it under the "RequestOrderID" column is essential for the queuing mechanism of our SI protocol as we will explain later. The transaction's writeset items are marked as "Y", and this information is used to detect conflicting updates. The "RequestOrderID" column is used to order the commit attempts and enforce the "first-committer-wins" rule.

The CommitQueue table (Table 3.5) is a queue for transactions that are already cleared for committing but just waiting for their turns to be actually committed according to the ordering of their commit timestamps. Each row in this table corresponds to a transaction and is indexed by the unique transaction ID obtained from the "W Counter table" (Table 3.1). The "CommitTimestamp" column stores the timestamp obtained from the "C Counter" table (Table 3.3) which is used as the commit timestamp of the transaction. Note that a transaction Ti first writes a row in this table with row key Wi, then requests and obtains its CommitTimestampCi, and finally adds Ci to its row. This sequence is again essential for the queuing mechanism to work properly, as explained below.

The Committed table (Table 3.6) stores the meta-data records for all the committed transactions. Each row in this table represents a successfully committed transaction indexed by the commit timestamp as the row key with the writeset data items as columns, containing the HBase timestamps used to actually write the data

Table 3.3: C counter table. C stands for "commit timestamp".

Row Key	Counter
C	54

Table 3.4: CommitRequestQueue table.

Row Key	writeset item 1	writeset item 2	writeset item 3	RequestOrderID
W1	Y		Y	R1
W2	Y	Y		

to the user's HBase data tables. In fact, for any transaction, successfully inserting a row into this table means that the transaction is committed atomically and the data becomes durable. Moreover, any row key of the table can identify a consistent snapshot because the rows in the table are strictly ordered and automatically sorted by row keys, and committed transactions are guaranteed to arrive in the Committed table in order due to the queuing mechanism, as explained below. The Committed table is also used by transactions in various functional ways, such as looking for the most recently committed version of data when reading, and checking for writeset conflicts at commit time against previously committed records. Note that HBase's sparse column nature is crucial here for efficiency: the table can contain many columns, but each column typically contains only few elements, and can be scanned efficiently.

In HBaseSI, each transaction sees a consistent snapshot of all the data in HBase user tables, identified by the start timestamp of the transaction. When a transaction Ti starts, it first gets its start timestamp by reading the last row of the Committed table at the time it starts, and uses the row key of that row Cj as the start timestamp. So we have Si=Cj, and Ti will see all data committed by Tj, and any transaction committed before Tj. Transaction Ti also obtains a unique ID Wi from the "W Counter" table as its transaction ID. Then it performs reads/writes based on the snapshot identified by the start timestamp. Data being read/written are first saved in in-memory readset/writeset data structures so that repeated reads can be efficiently served from memory, except for the first read/write of a certain data item. In this way, it is guaranteed that the transaction reads its own writes at all times. Writes are applied to the user data tables immediately (speculatively) using the transaction ID Wi as the unique timestamp to write to HBase (recall from Sect. 2 that a timestamp can be specified when writing data to HBase). At commit time, the transaction puts itself into the CommitRequest table, may wait for its turn if there are any conflicting commit attempts, then checks for conflicts with committed transactions, and finally enters the CommitQueue table if it is cleared to commit. It then waits for all the other concurrent transactions in the CommitQueue table with smaller RequestOrderID to commit, and finally commits by atomically inserting a simple record row into the Committed table to make its writes durable. The pseudocode of the protocol is provided in Listing 1.

```
1  Transaction {
   Writeset {(dataLocation(n),value(n))}; //containing N items
3  Readset {(dataLocation(m),value(m))}; //containing M items
   Long Wi, Si; //Wi is transaction ID, Si is start timestamp
5  Long Ri; //Ri is request order ID
   Long Ci; //Ci is commit timestamp
7
   //method called at the start of transaction
9  Start() { //transaction starts
     Wi = GetTimestamp (W counter);
11     Si = LastLineFromCommittedTable().getRowKey();
   }
13
   //method to read data value
15 Read(dataTable, dataRow, dataColumn) {
       dataLocation = dataTable + dataRow + dataColumn;
17     if (dataLocation in WriteSet) {read from WriteSet; return dataValue;} //read own writes
       if (dataLocation in ReadSet) {read from ReadSet; return dataValue;} //repeated read-only value
19   committedRecord = ScanForMostRecentRow (in Committed table, range [0, Si] containing a column named
          as dataLocation); //Scan in range [0, Si] (row keys are C counter values not less than 0),
        and return the last record in the list
       Wread = committedRecord.valueAtColumnDataLocation(); //find the latest data version in snapshot.
```

Table 3.5: CommitQueue table.

Row Key	CommitTimestamp
W1	C1
W2	

Table 3.6: Committed table.

Row Key	writeset item 1	writeset item2	writeset item3
C1	W1	W1	
C2	W2		W2

```
        If the data item is not in the Committed table, Wread will be set to null
21      dataValue = readData(in dataTable, in dataRow, in dataColumn, with Wread); //read data. If Wread
        is null, no timestamp will be specified in the HBase read (recall that it is optional to
        specify a timestamp in reading from HBase)
        ReadSet.add (dataLocation, dataValue);
23      return dataValue;
    }
25
    //method to write data value
27  Write(dataLocation, dataValue) {
      WriteSet.add(dataLocation, dataValue);
29    writeToDataTable (dataLocation, dataValue, using Wi); //directly write to data tables with HBase
        timestamp Wi
    }
31
    //method for commit attempt
33  boolean Commit() {
        EnqueueForCommitRequest(); //queue up for requesting to commit
35      CheckConflictsInCommittedTable (up to Si, with conflicting WriteSet); //do scan in the Committed
        table for writeset columns in range [Si + 1, +INFINITY) and verify that there are no
        writeset conflicts
        If (clearedToCommit) {
37        EnqueueForCommitting(); //when cleared to commit, queue up to finally commit
        } else {
39       doCleanup(); //abort transaction, remove rows in system tables and data items written to user
          tables
        }
41  }

43  //method to get a counter value
    GetTimestamp(HBaseTimestampTable) {
45    IncrementColumnValue (HBaseTimestampTable) //the mechanism to issue globally unique and well-
        ordered timestamps from a central HBase table
    }
47
    //method to enqueue for commit request
49  EnqueueForCommitRequest() {
        WriteHBaseTableRow (into CommitRequestQueue Table, row Wi, columns WriteSet);
51      Ri = GetTimestamp(R counter);
        WriteHBaseTableRow (into CommitRequestQueue Table, row Wi, column Ri);
53    PendingCommitRequests = GetRowsWithConflictingWriteSet(From CommitRequestQueue Table); //one-time
        scan
      while (PendingCommitRequests.isNotEmpty()) { //there exist requests to update conflicting data
55        select a row from PendingCommitRequests;
        if (row has disappeared) {
57            remove row from PendingCommitRequests; //the other transaction has completed
        } else {
59            wait until Ri appears in the row;
              if (row.Ri is larger than its own Ri) { //the other request is later than self
61                remove row from PendingCommitRequests; //no need to consider
              } else { //the other request is earlier than self
63                wait until row disappears; //wait till the other request is handled
          remove row from PendingCommitRequests;
65            }
          }
67    }
    }
69
    //method to enqueue for committing
71  EnqueueForCommitting() {
```

```
    WriteHBaseTableRow (into CommitQueue Table, row Wi);
73  Ci = GetTimestamp(C counter);
    WriteHBaseTableRow (into CommitQueue Table, row Wi, Ci);
75  PendingCommits = GetAllRows (From CommitQueue Table); //one-time scan
    while (PendingCommits.isNotEmpty()) {
77      select a row from PendingCommits;
      if (row has disappeared) {
79        remove row from PendingCommits; //the other transaction has completed
      } else {
81        wait until Ci appears in the row;
          if (row.Ci is larger than its own Ci) {
83          remove row from PendingCommits; //no need to consider
          } else {
85          wait until row disappears;
            remove row from PendingCommits;
87        }
      }
89 }
   //proceed to commit
91  WriteHBaseTableRow (into Committed Table, row Ci, columns WriteSet each containing value Wi); //
        atomic commit operation
    DeleteOwnRecordIn(CommitQueue table);
93  DeleteOwnRecordIn(CommitRequestQueue table);
   }
95
   Main() {
97  Start();
    ... //do reads and writes
99  Commit();
   }
101
   }
```

Listing 1: Pseudocode for the HBaseSI protocol.

It is important to understand in detail how HBaseSI handles distributed synchronization among concurrent transactions concerning the global ordering of transaction commit requests and commits. HBaseSI makes use of distributed queues to manage transaction commits and to guarantee the "first-committer-wins" rule, instead of using other traditional methods such as data locks or consensus-based protocols. The benefit is simplicity in design and implementation, which may in turn improve performance by avoiding the complexity of handling deadlocks and mandating complicated negotiation protocols for reaching consensus on transaction commit decisions between distributed data hosting servers involved in each transaction. HBaseSI makes use of two queues, implemented as two HBase tables. One is the CommitRequestQueue (Table 3.4); the other is the CommitQueue (Table 3.5). The protocol to ensure a correct sequence of entering and exiting a queue is the same for the two queues and therefore we explain the protocol using one queue, the CommitRequestQueue, as an example. Recall that when a transaction Ti makes a request to start the commit process, it first inserts a row indexed by its unique transaction ID Wi (obtained from the "W Counter table"), then gets a commit request counter value Ri and puts it under the "RequestOrderID" column of its row. The Ri value determines the order of Ti in the queue. This sequence of operations is of essential importance to guarantee that no concurrent transaction will leave the queue out of order, as we explain now. After transaction Ti inserts counter value Ri into its row in the CommitRequestQueue table, it reads all records in the table once. It then waits until all rows of transactions Tj it has read obtain Rj values in the table. This is essential to allow the queue to function based on the order of the R counter values: Ti is guaranteed to see any transaction Tj still in the queue that may have Rj<Ri, even if Rj appears in the table after Ri. This is so because Ti reads the table *after* it has obtained Ri, and any Tj still in the queue that may have Rj<Ri is guaranteed to have its row in the table at that time, because it inserted its row *before* requesting Rj. Ti will not proceed to the commit process until all Tj with Rj<Ri have left the queue, guaranteeing that transactions are processed in order and establishing the "first-committer-wins" rule. Based on the strict sequence of transactions entering the queue table, the protocol to ensure the ordering of exiting the queue is shown in Listing 1: pseudocode line 49 to 69. The pseudocode contains an optimization of the basic queuing protocol: transactions in the queue only need to wait for transactions that have a conflicting writeset. The same queuing protocol, using C counter values Ci, is also used to guarantee that transactions that are cleared to commit arrive in order in the Committed table, see lines 71-89 in the pseudocode. Using this queuing protocol, we can make sure that transactions follow the exact order as specified by their globally unique and well-ordered counter values. With the queuing mechanism, we can easily enforce a strict global ordering of transaction commits.

Table 3.7: Shop table.

Row Key	iPhone4	BlackBerry
Stock	1	3

Table 3.8: Committed table.

Row Key	Shop:Stock:iPhone4	Shop:Stock:BlackBerry
C6	W6	W6

Transactions in HBaseSI satisfy ACID properties as well as strong SI. Atomicity is provided by the underlying HBase atomic row write functionality because the final commit process only requires a single row write to the Committed table (Listing 1: pseudocode line 91). Durability is guaranteed by the underlying persistent data storage mechanism, i.e., Hadoop HDFS, because all the data in HBase are stored in HDFS. Consistency is maintained because only valid data is inserted into the HBase tables through the provided APIs and transactions never leave HBase in a half-finished state. The isolation level provided by HBaseSI is strong snapshot isolation. Strong SI requires that a transaction reads/writes in isolation upon a consistent snapshot of data identified by a start timestamp. Seen from the protocol above, our system guarantees that a transaction can see all the updates committed before it starts (start timestamps are row keys from the Committed table and any row key in the Committed table can identify a consistent snapshot containing all the previous committed updates). Our system also guarantees that transactions can only commit (atomically) if no conflicting updates have been inserted by previously committed concurrent transactions. Therefore strong SI holds.

3.2. Protocol Walkthrough by Example. We now describe the transactional SI protocol along with the system table usage in more detail by walking through the process of handling two concurrent transactions with conflicting updates under a concrete example scenario. In this example scenario, Alice and Bob intend to purchase smart phones from an online shop. They make their purchases by doing transactions involving several data tables of the shop stored in HBase, for example, item inventory, billing, etc. For simplicity, we limit their transactions to updating the same "Shop" table containing information about the number of available smart phones in stock.

Initially, the Shop table shows that the stock is updated with 1 iphone4 and 3 BlackBerrys (Table 3.7) by a transaction with unique ID W6 and commit timestamp C6 (Table 3.8). The Committed table contains a record for this stock update. Bob and Alice start transactions Ta and Tb concurrently, with start timestamps Sa=C6 and Sb=C6 (note that snapshots of different transactions can be the same, such as in this case). Now let's assume that Alice and Bob both read the stock of iPhone4 and BlackBerry, and then Alice decides to buy 1 iPhone4 while Bob would like to buy both an iPhone4 and a BlackBerry. Transactions Ta and Tb query the Committed table using the start timestamps Sa=C6 and Sb=C6 to get the most recently committed version of the stock data of both types of phones. They will both obtain HBase timestamp W6 and use W6 to read the stock from the Shop table and put the results into their readsets. (Listing 1: pseudocode line 15 to 23) After that they perform writes to update the stock and put data into their writesets (Listing 1: pseudocode line 27 to 31). Note that writes are applied to the Shop table immediately using timestamp Wa by Ta and Wb by Tb respectively, which is facilitated by the multi-version support of HBase (Sect. 2.2). We choose to write the data into the data tables speculatively to make the commit process faster. The writes become visible to other transactions only after the transaction has successfully committed.

When they are ready to attempt to commit, Ta and Tb use their transaction ID (Wa for Ta and Wb for Tb) as the row key to add a row to the CommitRequestQueue table with their writeset items as columns respectively (Table 3.9). Both transactions enter the CommitRequestQueue table a row with values for their writesets, and then request their commit request ID from the R Counter table. Then they put the commit request IDs, Ra and Rb, under the RequestOrderID column and perform a scan of the entire CommitRequestQueue table for all other row records with conflicting writeset items. This is to find any conflicting concurrent commit requests that may have Rj<Ra or Rj<Rb in the queue from transactions Tj. In our example, assume that Tb finishes inserting Rb into its row and that the row for Ta has not appeared in the table yet. Tb then scans the CommitRequestQueue

Table 3.9: CommitRequestQueue table.

Row Key	Shop:Stock: iPhone4	Shop:Stock: BlackBerry	RequestOrderID
Wa	Y		Ra
Wb	Y	Y	Rb

Table 3.10: CommitQueue table.

Row Key	CommitTimestamp
Wb	Cb

and finds no conflicts (Ta has not inserted its row yet). Then Tb can proceed to scan the Committed table to check if there are any conflicting committed transactions with commit timestamp larger than its start timestamp (C6). Assume there are none. Tb is now cleared for committing and atomically (line 91) adds a row with its transaction ID Wb as the row key to the CommitQueue table (Table 3.10). After adding the row, it requests and obtains a commit timestamp Cb and then puts it into its row under the CommitTimestamp column. It then waits in the CommitQueue for its turn according the CommitTimestamp to finally commit. This wait in the CommitQueue guarantees that all committed transactions Ti appear in the Committed table in the order of their commit timestamps Ci, and thus that all the records appearing in the Committed table are well ordered. After Tb finishes committing (see the resulting Committed table in Table 3.11), it will delete its row in both the CommitQueue and CommitRequestQueue (Listing 1: pseudocode line 92-93). In the meantime, assume Ta finishes inserting its row a bit later, and after it scans the CommitRequestQueue table for rows with conflicting columns, it sees that Tb has already entered the CommitRequestQueue with a conflicting writeset and RequestOrder ID Rb. Since Rb<Ra, Ta waits until row Tb disappears (meaning that Tb has either been committed or aborted) before proceeding (Listing 1: pseudocode line 54-65).

3.3. Read Optimization. An optimization for performance to the protocol above is necessary because the size of the Committed table grows linearly as transactions commit (each committed transaction creates a corresponding row that persists in the Committed table). Recall that when reading a data item, HBaseSI needs to scan all the rows in the Committed table up to the snapshot timestamp and iterate through the records in the result list of the scan to find the most recent data version. As shown in Fig. 4.4 below, the time it takes for scanning and iterating through the records grows linearly as the number of rows containing the target columns to scan increases. It would be good if only a small range of the committed table needs to be scanned by newly arrived transactions if the most recently known committed data version is kept somewhere globally visible. Following this idea, an extra system table called "Version table" is created (Table 3.12). Each row in the version table corresponds to a data item that has been written to, identified by its table, row and column name combination. Instead of using a centralized system component to constantly update the Version Table records, every transaction is responsible to update the records when new versions of data are read. With the Version table, when a transaction Ti tries to read any data item, it needs to query the version table first to see if there is a data version record. If there is a record and the commit timestamp Cj in the record is before Si, then Ti only scans the Committed table in the range [Cj, Si]. If no previous version is found or the version found is more recent than the snapshot time Si, a full scan of the Committed table up to the snapshot point Si is necessary. If a newer version is detected and read, the reading transaction updates the Version table record after reading the data item.

The adjusted pseudocode for reading with Version table is in Listing 2.

```
1 Read(dataTable, dataRow, dataColumn) {
      dataLocation = dataTable + dataRow + dataColumn;
3     if (dataLocation in WriteSet) {read from WriteSet; return dataValue;}
      if (dataLocation in ReadSet) {read from ReadSet; return dataValue;}
5     Cj = ScanVersionTable (dataLocation); //if the data item doen't exist in the Version table, Cj =
          0
      if (Cj <= Si) {
7     committedRecord = ScanForMostRecentRow (in Committed table, range [Cj, Si] containing a column
          named as dataLocation); //Scan in range [Cj, Si], and return the last record in the list
```

Table 3.11: Committed table.

Row Key	Shop:Stock: iPhone4	Shop:Stock: BlackBerry
C6	W6	W6
Cb	Wb	Wb

Table 3.12: Version table. For example, the most recently read version of the data item stored in user data location DataLocation1 was committed by the transaction with commit timestamp C17.

Row Key	CommittedTimestamp
DataLocation1	C17
DataLocationM	C8

```
    } else {
9     committedRecord = ScanForMostRecentRow (in Committed table, range [0, Si] containing a column
          named as dataLocation); //Scan in range [0, Si] (row keys are C counter values not less than
           0), and return the last record in the list
    }
11    if (committedRecord > Cj) {
        UpdateVersionTable (dataLocation, committedRecord);
13    }
      Wread = committedRecord.valueAtColumnDataLocation(); //find the latest data version in snapshot.
          If the data item is not in the Committed table, Wread will be set to null
15    dataValue = readData(in dataTable, in dataRow, in dataColumn, with Wread); //read data. If Wread
          is null, no timestamp will be specified in the HBase read (recall that it is optional to
          specify a timestamp in reading from HBase)
      ReadSet.add (dataLocation, dataValue);
17    return dataValue;
}
```

Listing 2: Read with Version table.

3.4. Handling Stragglers. In the protocol above, a transaction needs to wait in two queues, the CommitRequestQueue and the CommitQueue. Due to many possible failure conditions, transactions could stay in waiting forever if one or more of the previously submitted transactions get stuck in the commit process and never delete their corresponding rows in the above two queue tables. We call those transactions that do not terminate properly in a timely manner "stragglers". Measures must be taken to not only prevent such stragglers from hampering the other active transactions, but also to avoid any potential data inconsistency issues caused by re-appearing transactions that had been deemed to be dead.

HBaseSI handles stragglers by adding a timeout mechanism to the waiting transactions. More specifically, the waiting transactions can kill and remove straggling/failed transactions from the CommitRequestQueue or CommitQueue based on the clock of the waiting transaction if a preconfigured timeout threshold is reached. A problem associated with this method is that a straggler may come back to life and try to resume the rest of its commit process after its records in either queues are removed, which could cause data inconsistencies and incorrect SI handling. The solution to this problem is to use the HBase atomic CheckAndPut method on two rows at once in the Committed table when doing the final commit rather than only using a simple atomic row write operation on one row. The difference between CheckAndPut and simple row write is that the former method guarantees an atomic chain of two operations involving checking a row and writing to a possibly different row in the same HBase table, whereas the latter method only guarantees atomicity for a single row write operation. To use the CheckAndPut method, we first add an extra row called "timeout" in the Committed table (Table 3.13). When it starts, each transaction first marks the column named after its unique transaction ID Wi (obtained from the W Counter table) in the "timeout" row as "N", meaning that the transaction is not in timeout by default (a non-empty initial value "N" must be set because the CheckAndPut method does not work with empty column values). Later, in the commit process, if a transaction is deemed a straggler, other transactions will put a "Y" under the column named after the unique transaction ID of the straggler in the "timeout" row, and then delete the corresponding records of the straggler in both the CommitRequestQueue and the CommitQueue. (Note that the sequence of first marking the straggler in the Committed table and

Table 3.13: Committed table.

Row Key	writeset item 1	writeset item 2	W6	Wi	Wj
T6	W6	W6			
timeout			N	N	Y

only then deleting rows in the two queues is essential to the correctness of the SI mechanism). When a healthy transaction commits, it performs an atomic CheckAndPut: it checks for "N" in the "timeout" row, and if the check is successful, it puts its row into the Committed table. If the value under its corresponding column is still marked as "N", it can indeed successfully insert its row into the Committed table; otherwise it knows it has been marked as a straggler and should abort by deleting its records in both the CommitRequestQueue and the CommitQueue tables, if those records still exist. In this way, HBaseSI can make sure that no transaction can commit once it is marked as a straggler. There is no problem if after a transaction commits successfully by inserting a row into the Committed table, it fails to delete the corresponding rows in the queues on time; those records will be removed by waiting transactions after the timeout and SI is not compromised. Note that for garbage cleaning purposes, after a transaction successfully commits, it will remove the corresponding column value in the row "timeout".

3.5. Discussion. In this section, some further issues about the HBaseSI design and usage are discussed. First, there is no roll back or roll forward mechanism in HBaseSI and there is no explicit transaction log either. It is interesting to ponder on how HBaseSI supports ACID transactions, even in the face of failures, without those traditional mechanisms used in DBMSs. In fact, this all attributes to two very important HBase properties. The first one is that HBase stores many versions of data and allows reads/writes of data using a specific timestamp. This HBase property makes it possible for every concurrent transaction to write preliminary versions of data but only the successfully committed transactions get to publish the write timestamps they used in the Committed table for future reads. In other words, no roll back is necessary because uncommitted data won't be used in any case. The other property is the atomicity of the HBase row write and CheckAndPut methods. Using these atomic methods, HBase guarantees that once a row is inserted into the Committed table successfully, it becomes durable and is guaranteed to survive failures (media failure is handled by HDFS which stores data replicated across distributed locations).

Second, we discuss some design choices that affect performance such as scalability and disk usage. HBaseSI inherits many of the desirable properties of HBase because it is only a client library and imposes little overhead concerning system deployment. However, users need to be aware that in order to achieve several design goals, HBaseSI made some sacrifices for performance. For example, four important goals HBaseSI tries to achieve are: 1. global strong SI across table boundaries; 2. non-intrusive to user data tables; 3. non-blocking start of transactions with snapshots that are as fresh as possible (strong SI), and non-blocking reads; 4. strict "first-committer-wins" rule without lost transactions (transactions only abort when there is no chance they will be able to commit successfully). In order to achieve goal 2, HBaseSI is designed to use a separate set of system HBase tables for maintaining transactional metadata for all user tables instead of creating extra columns in each separate user table, which inevitably creates potential performance bottlenecks at the small number of global system tables. HBaseSI is therefore not designed to provide scalability in terms of the number of transactions per unit time, but its target is to provide scalability in terms of cloud size and user data size. HBaseSI makes the final commit process as short as possible and allows writes to insert preliminary data into the user data tables as the transaction proceeds rather than waiting till the commit time to apply all the updates (note that when a transaction aborts, it should remove its written items from user tables), avoiding chances for varyingly large waiting latency incurred by transactions with large writesets to be applied at commit time. In essence, HBaseSI trades disk space for high throughput in transaction commits. Additionally, it is important that the number of data versions HBase table locations can hold is set sufficiently high. Furthermore, a dedicated garbage cleaning mechanism should be created for optimized disk usage, with a policy on maximum transaction duration (such a policy is important to guarantee that the data that gets garbage cleaned is not needed by any long-running transactions).

Third, we discuss the efficiency of having transactions wait in queues when committing. Recall that in the

HBaseSI protocol, update transactions first wait in the CommitRequestQueue for the purpose of establishing an order in committing transactions and guaranteeing the "first-committer-wins" rule, and then wait in the CommitQueue after they are cleared for committing for the purpose of guaranteeing a correct global sequence of commits so that each row in the Committed table can identify a consistent snapshot of the data tables. Note that the first wait is only for transactions with conflicting writesets, but the second wait results in sequential processing of all concurrent transactions, no matter whether the writesets are in conflict or not. Although these two waits are essential for the commit queuing mechanism to work so that global strong SI can be achieved, it may sometimes be more efficient to relax the second wait to the extent that a transaction only waits for other transactions that use the same set of user tables. This would require transactions to declare in advance which groups of tables they use. This relaxation is reasonable in real-world applications. HBaseSI can be very easily adapted to such extended usage scenarios to make transactions more efficient in terms of minimizing unnecessary wait times in the CommitQueue. The decision of whether to use the extended scheme would be at the users' discretion.

Finally, we discuss the cost of adopting HBaseSI and the easiness of reverting back to non-SI default HBase. Normally, once one starts to use HBaseSI, all the read/write operations must be performed through the HBaseSI API rather than the default HBase API. Only through the HBaseSI API can a transaction find the correct timestamp used in writing the most up-to-date data, or make its committed updates accessible. However, it is very easy to write a small tool to help restore the user data tables back to a state that users can use their data tables in the default HBase manner. The tool only needs to write the latest version of committed data to all the user data tables once, without specifying timestamps (so that the HBase default timestamps are used). The next time users want to use HBaseSI again, they can directly use it without any required changes.

4. Performance Evaluation on Amazon EC2. The general purpose of this performance evaluation section is to quantify the cost of adopting the HBaseSI protocol in handling concurrent transactions. Therefore tests are performed on each critical step of the HBaseSI protocol, with comparison to the performance of bare-bones HBase when possible. Additionally, because HBaseSI is the first system that achieves global strong SI on HBase, there are no other similar systems to compare with for some of the properties. As a result, for those properties, the tests serve the purpose of showing the users the expected behavior of the system. Furthermore, as mentioned in Sect. 3.3 above, HBaseSI uses a set of global system tables that facilitate non-blocking reads and a strict "first-committer-wins" rule, but may become performance bottlenecks if accessed by many concurrent transactions. In order to make the performance effect of this design decision more apparent, we decided to deploy all systems tables at the same HBase region server by running all the HBase components on a single machine, mimicking the possible extreme real-world condition when concurrent transactions significantly outnumber the HBase servers. The test results are thus expected to show the system performance under heavy loads.

We use 20 Amazon machines in total to perform the tests and we are aware that performance variations may be observed in Amazon instances [9]. The test results may be affected by this to some extent but should be enough for proof-of-concept purposes. A high memory 64-bit linux instance with 15 GB memory, 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each) and high I/O performance is used to host both the Hadoop and HBase server components. Up to 19 other 64-bit linux instances with 7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each) and high I/O performance are used to run client transactions. All these machines are in the same Amazon availability zone so that the network conditions for each instance are assumed to be similar.

In the tests, each machine runs a single client program issuing transactions if the total number of clients is less than 19. If the total number of clients is more than 19, an equal number of concurrent clients are run at each machine instance. For example, each machine instance can run 1, 2, or more clients with the total number of transactions being 19, 38, etc. At each client, transactions are issued consecutively one after another. In other words, a new transaction will only be issued when the previous one has finished executing, having either committed or aborted.

The goal of Test 1 is to measure the performance of the timestamp issuing mechanism in terms of throughput. In the test, each client connects to the server and requests a new timestamp directly after being granted one. After a starting flag is marked in an Indicator table, all clients run for a fixed period of time and stop. The throughput is calculated by dividing the total number of timestamps issued by the length of the fixed time period. Figure 4.1 shows the result of this test. Apparently the server gets saturated at a total throughput of about 360 timestamps per second, or about 30 million timestamps per day. Note that the timestamp generating

mechanism currently used by HBaseSI is the most straightforward solution a user can get by using bare-bones HBase functionalities. Other more efficient timestamp generating mechanisms with much higher throughput can also be adopted if the user desires, such as the one used by Google's Percolator system [8] which generates 2 million timestamps per second from a single machine.

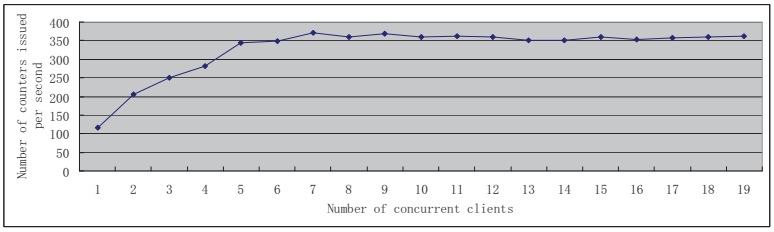


Fig. 4.1: Test 1, performance of the timestamp issuing mechanism through counter tables.

The goal of Test 2 is to measure the performance of the start timestamp issuing mechanism via the Committed table in terms of throughput, i.e., how many transactions can be allowed to start per second (in order for a transaction to start, a start timestamp must be issued first) with an increasing number of concurrent clients. Recall that the mechanism to obtain a start timestamp is different from getting a unique counter value from one of the counter tables. Instead, a transaction needs to read the last row of the Committed table at the time it starts and use the row key as its start timestamp. In this test, the clients all connect to the server first and then wait for a signal in the Indicator table to start at the same time. During the test, a program is run at the EC2 instance running the HBase server inserting a new row to the Committed table continuously, mimicking the real-world scenario where the Committed table keeps growing in size because of newly committed transaction records. The throughput is calculated in the same way as Test 1. Figure 4.2 shows the result for Test 2. The throughput stabilizes at about 420 timestamps per second due to server saturation, slightly higher than the result obtained from Test 1. The higher performance is expected because in Test 1 an atomic function call to increment a common column value is issued each time a counter value is to be obtained by each concurrent client, potentially causing a blocking write conflict at the HBase server, while in Test 2, only scanning the last row of the Committed table is necessary. The performance is thus satisfactory to the extent that the start timestamp mechanism is not the limiting bottleneck for starting new transactions even if the mechanism requires that every transaction should read from the Committed table at starting time.

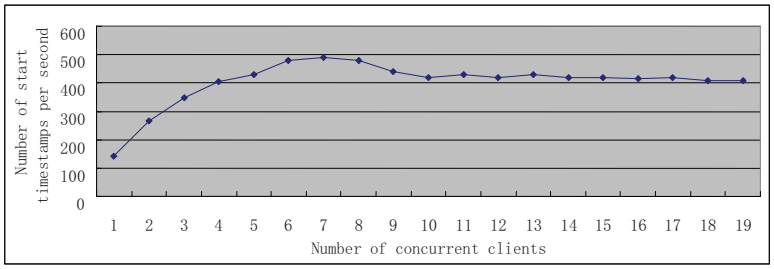


Fig. 4.2: Test 2, performance of the start timestamp issuing mechanism.

The goal of Test 3 is to study the comparative performance of transactions with SI that contain a set of read/write operations, against executions of the same number of read/write operations with bare-bones HBase, for varying numbers of operations per transaction. In the test, we run 1 client only, vary the number of operations per transaction and measure the time spent on each read/write operation. Additionally, in order to control the performance overhead associated with scanning a growing Committed table (recall from Sect. 3 that each SI read needs to scan the Committed table first to get the most up to date data version before actually reading the data), after each client run, the Committed table is manually cleaned. (In this test, no previous data versions exist, because the Committed table is cleaned up after each previous transaction execution and data locations are only written to once, but a quick scan is still executed for every read). The result of the test quantifies the performance overhead of transaction SI over bare-bones HBase. The results in Fig. 4.3 show the

startup/commit overhead of the protocol and how it can be amortized as the number of read/write operations per transaction grows. This indicates that the protocol is more efficient for transactions involving a larger number of operations per transaction or transactions with longer inter-operation intervals (user "think time" during user interactions) to better amortize the transaction startup/commit overhead.

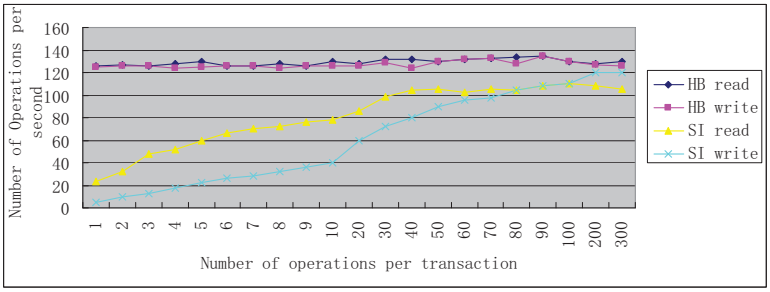


Fig. 4.3: Test 3, comparative performance of executing transactions with SI against bare-bones HBase without SI.

The goal of Test 4 is to measure the time needed to scan a column in a data table over a growing row range (each row contains a data value in the column scanned). The expected result is a linear growth of time corresponding to the number of table rows scanned. The result is used to show the necessity of using the Version table when performing reads in order to avoid costly full scans of the Committed table on every read. In this test, a single client is executed to scan a data table with a continuously growing row range. The test result is shown in Fig. 4.4 and is exactly according to expectation with linear growth in time.

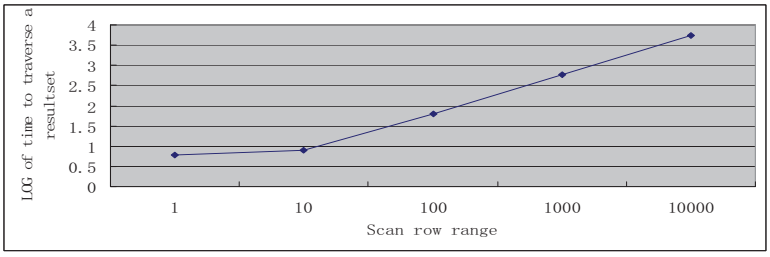


Fig. 4.4: Test 4, time to traverse a resultset against a varying number of rows to scan.

The goal of Test 5 is to measure the comparative performance of transactional SI with the use of the Version table on workloads complying with the TPC-W benchmark [13], which is used widely for evaluating database performance under OLTP loads. The TPC-W benchmark describes several different kinds of workloads with mixed read/write operations corresponding to real-world e-commerce scenarios, such as online shopping. A "browsing mix" is composed of transactions containing 95% read and 5% write operations; a "shopping mix" is composed of 80% reads and 20% writes; and an "updating mix" is composed of 50% reads and 50% writes. In the test, we run clients executing the above three kinds of workloads with a varying number of concurrent clients, each executing a random number of reads/writes according to the above specifications with an average of 15 operations per transaction, upon a table with 10,000 data rows. We measure two things: overall throughput (number of transactions per second) and average commit time for update transactions (the average time spent in the commit process). The overall throughput includes both successful and aborted transactions and shows the general system capacity in handling concurrent transactions. It is also interesting to see how much time is spent in the CommitRequestQueue and the CommitQueue separately because for different types of mixed workloads, the ratio of the number of update transaction requests and the number of actually committed transactions is different. The result for total throughput is shown in Fig. 4.5. An interesting point for this result is the comparative performance between these types of workloads. As we can see, as the number of concurrent clients grows, the shopping mix has the lowest throughput while the browsing and update mix have similar throughput. The reason why the shopping mix has the lowest throughput is because this mix actually has the most number

of successful update transactions processed among the three mix types: the browsing mix doesn't have many costly update transactions, and the updating mix doesn't have many successfully committed update transactions either because of the higher probability of having conflicts (and we count failed transactions in the throughput). The sharp drop in throughput, especially for the updating mix, when there are 95 concurrent clients is because of both the server saturation and the extra wait time in the CommitQueue.

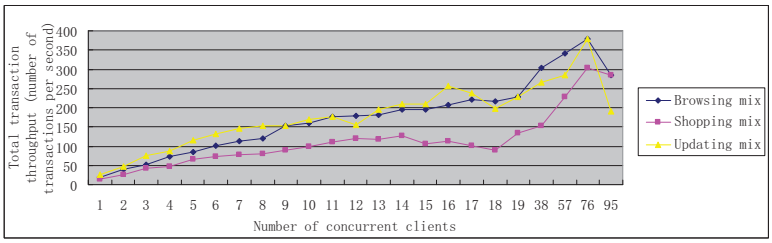


Fig. 4.5: Test 5, general performance of executing transactions with SI under TPC-W workloads.

Results for the average commit time for all three types of mixed workloads are shown in Figs. 4.6, 4.7 and 4.8, respectively. As for the browsing mix (Fig. 4.6), update transactions are relatively rare (5%). Therefore conflict probability is low. Transactions that get queued in the CommitRequestQueue are also likely to be able to commit successfully in the end. Therefore transactions tend to spend almost the same time on average staying in both queues. As for the shopping mix (Fig. 4.7), more update transactions are queued up for committing after passing the commit request checking stage at the CommitRequestQueue. Because there are almost no conflicts (the conflict rate will increase with a large number of concurrent clients, especially with respect to the total number of data items under shared access) and the CommitRequestQueue is basically skipped for most committing transactions, the wait time in the CommitQueue is much higher. As for the updating mix (Fig. 4.8), because there is a much higher conflict probability than for the other two mix workloads, more transactions are aborted at the checking stage in the CommitRequestQueue. Therefore the point at which the wait time in the CommitQueue outruns the wait time in the CommitRequestQueue comes later than for the shopping mix workload. The results in Figs. 4.6, 4.7 and 4.8 also indicate that the timeout threshold used in the straggler handling mechanism should be set according to the type of mix workload and may need to be adjusted according to the number of concurrent requests.

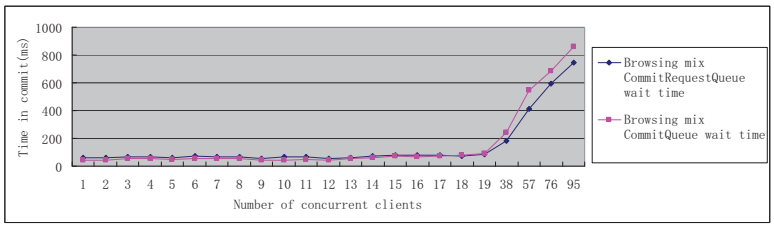


Fig. 4.6: Test 5, browsing mix wait time in both CommitRequestQueue and CommitQueue.

The goal of Test 6 is to test the effectiveness of the straggler handling mechanism. We use the "shopping mix" from Test 5 with 19 concurrent clients and add an abort ratio at the end of each transaction. With an increasing abort ratio, we measure the total throughput in terms of transactions per second. Because the artificially inserted aborts occur at the end of transactions while transactions wait in the CommitRequestQueue after completing all the reads/writes, we still count the aborted transaction into the calculation of the throughput. The failed transactions become stragglers in the CommitRequestQueue table that have to be removed by live transaction processes. The results show how random transaction faults affect the performance of the SI protocol. As seen in Fig. 4.9, the system achieves throughput similar to the case with no artificially inserted faults (because we also count the aborted transactions in the throughput calculation). We can also see from Fig. 4.10 that the duration of successful transactions stays almost constant in the face of failures, indicating that the straggler handling mechanism is effective in bounding healthy transaction duration.

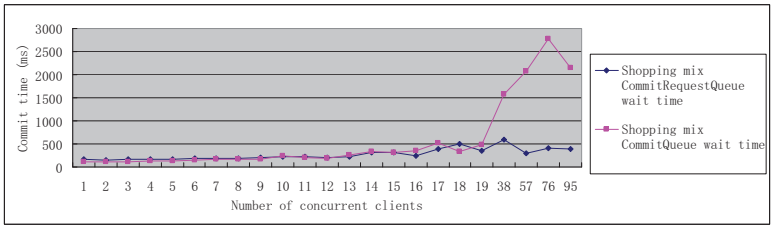


Fig. 4.7: Test 5, shopping mix wait time in both CommitRequestQueue and CommitQueue.

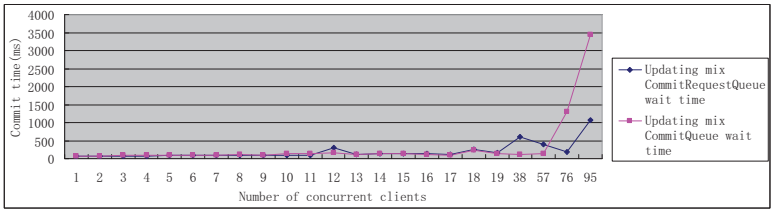


Fig. 4.8: Test 5, updating mix wait time in both CommitRequestQueue and CommitQueue.

5. Related Work. Several transactional systems exist for HBase, but none provide SI. The HBase project itself includes a contributed package for transactional table management, but it does not support recovering transaction states after region server failures. G-store [4] supports groups of correlated transactions over a pre-defined set of data rows (called "Key Group") specified for each group of transactions respectively, but assumes that the number of keys in a Key Group is small enough to be owned by a single node. CloudTPS [10] implements a server-side middleware layer composed of programs called local transaction managers (LTMs), but introduces extra overhead of middleware deployment, data synchronization, and fault handling.

Only recently two relevant papers were published independently at almost the same time about achieving snapshot isolation for distributed transactions, for HBase and for BigTable: we published a paper describing our preliminary system (the predecessor of the system described in this paper) to support transactions with SI on top of HBase [11], and Google published a paper about their system called "Percolator" [8] supporting transactions with SI on top of BigTable. The two systems share many design ideas yet are different in some major design choices.

HBaseSI is an extended and improved version of our preliminary system of [11]. It is similar to the preliminary system and similar to Google's Percolator [8] in that: all three systems are implemented as a client library rather than a set of middleware programs and allow client transactions to decide autonomously when they can commit (there is no central process to decide on commits); they all rely on the multi-version data support from the underlying column store for achieving snapshot isolation, and store transactional management data in column store tables; they all make use of some centralized timestamp issuing mechanism for generating globally well-ordered timestamps; and after starting using either of the systems, users must use the systems for all the subsequent data processing operations in order to guarantee data consistency. HBaseSI is superior to the preliminary system in that: HBaseSI is the first system on HBase to support global strong SI rather than the "gap-less" weak SI in the preliminary system; it uses a completely different mechanism in handling distributed synchronization (HBaseSI uses distributed queues to guarantee a correct sequence of transaction execution, while the preliminary system uses a complicated and inefficient mechanism to obtain snapshots); the preliminary system is inefficient because its PreCommit table grows without bound and has to be searched in its entirety by transactions attempting to commit; it provides a simple mechanism for handling stragglers, whereas handling stragglers for the system proposed in [11] would be overly complicated.

In addition to the similarities listed above, HBaseSI shares with Percolator its support of global strong SI. HBaseSI and Percolator are also very different in several other aspects: HBaseSI focuses on random access performance with low latency whereas Percolator focuses on analytical workloads that tolerate large latency; HBaseSI is non-intrusive to existing user data tables and stores the version information and transaction informa-

Fig. 4.9: Test 6, throughput seen at each client under a varying failure ratio.

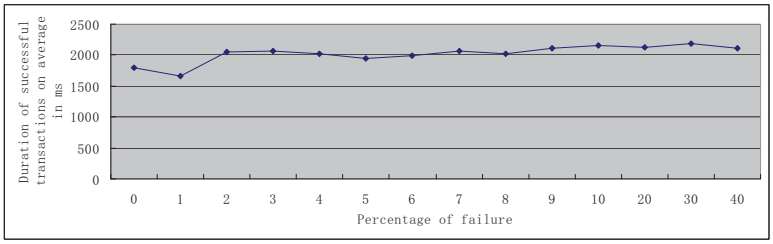


Fig. 4.10: Test 6, average duration of successful transactions under a varying failure ratio.

tion in extra system tables, whereas Percolator is intrusive to existing user data and stores the same information in two extra columns in every user tables (but this design decision of HBaseSI makes it less scalable than Percolator concerning the number of concurrent transactions); HBaseSI supports non-blocking starts of transactions and does not block reads, whereas Percolator may block reads while data is being committed which may harm performance; HBaseSI uses distributed queues in handling synchronization and concurrency rather than using traditional techniques such as data locks as in Percolator; and two concurrently committing transactions could unnecessarily both fail in Percolator but not in HBaseSI. In short, the two systems are designed with different purposes in mind and each may excel at one aspect and not another. Note also that the protocol described in Percolator cannot be trivially ported onto HBase, because HBase does not support BigTable's atomic single-row transactions, allowing multiple read-modify-write operations to be grouped into one atomic transaction as long as they are operating on the same row.

6. Conclusions and Future Work. This paper presents HBaseSI, a light-weight client library for HBase, enabling multi-row distributed transactions with global strong SI on HBase user data tables. There exists no other systems providing the same level of transactional isolation on HBase yet. HBaseSI tries to achieve several design goals: achieving global strong SI across table boundaries; being non-intrusive to existing user data tables; strictly enforcing the "first-committer-wins" rule for SI; supporting highly responsive transactions with no blocking reads; and employing an effective straggler handling mechanism. The performance overhead of HBaseSI over HBase is modest, especially for longer transactions involving a larger number of read and write operations per transaction. Future work includes implementing some helpful tools to optimize disk usage and possibly extending HBaseSI to increase its scalability by distributing the transactional metadata tables.

REFERENCES

[1] D. Agrawal, A. E. Abbadi, S. Antony, and S. Das, *Data management challenges in cloud computing infrastructures*, Proc. DNIS'10 (2010), pp. 1–10.
[2] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, *A critique of ANSI SQL isolation levels*, Proc. of SIGMOD (1995), pp. 1–10.
[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, *Bigtable: A Distributed Storage System for Structured Data*, Proc. OSDI, USENIX Association (2010), pp. 205–218.
[4] S. Das, D. Agrawal, and A. El Abbadi, *G-store: a scalable data store for transactional multi key access in the cloud*, Proc. SoCC '10 (2010), pp. 163–174.
[5] K. Daudjee and K. Salem, *Lazy Database replication with snapshot isolation*, Proc. of VLDB (2006), pp. 715–726.

[6] F. Farag, M. Hammad, and R. Alhajj, *Adaptive query processing in data stream management systems under limited memory resources*, Proc. of the 3rd workshop for Ph.D. students in information and knowledge management, PIKM '10 (2010), pp. 9–16.
[7] P. Helland, *Life beyond distributed transactions: an apostate's opinion*, CIDR (2007), pp. 132–141.
[8] D. Peng and F. Dabek, *Large-scale incremental processing using distributed transactions and notifications*, Proc. OSDI, USENIX Association (2010), pp. 1–15.
[9] J. Schad, J. Dittrich, and J. Quiané-Ruiz, *Runtime measurements in the cloud: observing, analyzing, and reducing variance*, Proc. VLDB Endow. (2010), 3, 1-2, pp. 460–471.
[10] Z. Wei, G. Pierre, and C.-H. Chi, *Scalable Transactions for Web Applications in the Cloud*, Proc. of the Euro-Par Conference (2009), pp. 442–453.
[11] C. Zhang and H. De Sterck, *Supporting multi-row distributed transactions with global snapshot isolation using bare-bones hbase*, Proc. of Grid2010 (2010).
[12] The Apache Software Foundation, *An open-source, distributed, versioned, column-oriented store. http://hbase.apache.org/*, retrieved April 15, 2011.
[13] The Transaction Processing Performance Council, *A transactional web e-Commerce benchmark. http://www.tpc.org/tpcw/default.asp*, retrieved April 15, 2011.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 227–238. http://www.scpe.org
ISSN 1895-1767

PARALLELIZATION OF COMPUTE INTENSIVE APPLICATIONS INTO WORKFLOWS BASED ON SERVICES IN BEESYCLUSTER

PAWEL CZARNUL*

Abstract. The paper presents an approach for modeling, optimization and execution of workflow applications based on services that incorporates both service selection and partitioning of input data for parallel processing by parallel workflow paths. A compute-intensive workflow application for parallel integration is presented. An impact of the input data partitioning on the scalability is presented. The paper shows a comparison of the theoretical model of workflow execution and real execution times. The execution of this distributed workflow is compared to a highly parallel approach using MPI. Finally, results for an integrated workflow/MPI approach are shown.

Key words: workflow management, service selection, data partitioning, parallel computing, numerical integration

1. Introduction. Parallel and distributed processing is now possible thanks to a variety of architectures, application models and technologies. First, these are available for both shared memory and distributed memory processing within HPC clusters [30]. Secondly, approaches such as grid, sky and cloud computing [5] allow integration of services at a high level.

Several models and frameworks have been proposed for complex distributed applications. Quality of Service (QoS) needs to be considered where resources are usually shared among various users requesting their own jobs [13, 29, 35]. Such approaches need to offer enough flexibility in constructing a distributed application and provide the ability to process input data in parallel. Ideally, the user should not be involved in low level details such as selection of services for particular tasks or selection of resources to execute the code. The user should define functional and QoS goals and rely on automatic selection to meet the goals.

2. Related Work. Parallel processing is currently being implemented at various levels within:

1. computing nodes using GPU programming [2], programming many processor cores,
2. local systems such as HPC clusters in which computing nodes are interconnected with a reasonably fast interconnect such as Infiniband, Gigabit Ethernet etc. [6, 30],
3. at a distributed level using e.g. grid [17], cloud computing [5].

One of practical approaches for modeling complex distributed processing based on services is done using workflow applications. A workflow application is modeled as an acyclic graph in which vertexes denote tasks to be executed while directed edges dependencies between the tasks. For *scheduling in utility grids* [34]/*workflow scheduling in grids* [31] for each task t_i there is a set of services S_i out of which one service is selected to execute t_i. Other attributes such as service costs are considered [33]. As proposed by [34, 35] the goal is to find the best assignment of $t_i \rightarrow (s_k, t_{ik}^{st})$ where s_k is a service able to execute task t_i and t_{ik}^{st} is the starting time of execution of task t_i using service s_k. Execution of t_i and t_j on one s_k must not overlap. The workflow execution time should be minimized while keeping the cost of selected services below a predefined minimum [7, 8]. In the context of typical business interactions, many more quality attributes are considered such as execution time, cost, availability [7, 27, 36], accessibility [27], fidelity [9] or conformance [27], security [27], reputation [36] is minimized. Usually no dependencies between or overlapping of services executing different tasks are considered in these applications.

From the infrastructure point of view, several MPI [26] implementations have been traditionally used for parallel programming. OpenMPI supports threads and MPI simultaneously. Additionally, several tools can be used for parallel programming within nodes such as CUDA [2], OpenCL, Pthreads [30] etc.

For the distributed workflow model, there are several workflow management systems for grid computing. Paper [32] provides a review and comparison of Gridbus, Kepler [23], Pegasus [16], Triana [25], P-GRADE [22], Directed Acyclic Graph Manager (DAGMan), ICENI, GridFlow, GrADS, Askalon, UNICORE, Taverna, GridAnt. These grid-oriented systems rely mainly on middlewares such as Globus Toolkit, Grid Application Toolkit or other resource management systems for starting and management jobs. Business oriented systems such as Meteor-S [3] incorporate support for BPEL for modeling workflows and use semantic service discovery and composition. As suggested by [24], BPEL can also be used for grid environments. Input data can often be

*Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland, (pczarnul@eti.pg.gda.pl) http://fox.eti.pg.gda.pl/∼pczarnul.

defined as values, files or data streams [19], as in Gridbus. Input data can drive workflows as in Kepler [1]. The input data from one or several preceding workflow tasks can be treated and processed in several ways [18].

3. Motivations. While this paper builds on the model presented in [14], the contribution of this paper is as follows:

1. assessment of the impact of input data partitioning of input data on the scalability for a workflow with parallel paths,
2. comparison of the theoretical model of processing in a workflow and real results,
3. assessment of the overhead of the parallelization using workflows with distributed services and a highly parallel solution using MPI,
4. integration of parallel processing by parallel workflow paths and parallelization within services using MPI.

4. Model of the Workflow Scheduling with Data Distribution. The model proposed by the author is based on the workflow model with service selection [13, 35] and is extended to consider data distribution for parallel paths of the workflow.

A directed graph $G(V, E)$ represents a workflow in which nodes V correspond to tasks while edges E represent task dependencies. At least one starting node with initial data and one termination node which terminates computations are distinguished. The starting nodes do not have predecessors. Each node should have a successor apart from the termination node. The model allows to define:

1. a sequence – a service assigned to the second task in a sequence successor is executed only after the service selected for the predecessor has completed (Figure 4.1a),
2. fork – services assigned to the tasks following the forked task are executed in parallel provided these were installed on separate processors (Figure 4.1b),
3. join – the service selected for the task to which other tasks are connected are executed only after each of the predecessors has finished (Figure 4.1b).

The following parameters are distinguished following [14]:

- $S_i = \{s_{i0}, s_{i1}, ..., s_{i(|S_i|-1)}\}$ – a set of services out of which one is selected to execute task t_i,
- c_{ij} – the cost of processing a unit of data by service s_{ij},
- N_{ij} – the node on which service s_{ij} was installed,
- sp_n – the speed of node n,
- P_{ij} – the provider of service s_{ij},
- d_{ij}^{in} and d_{ij}^{out} denote the sizes of the input and output data accepted and produced by service s_{ij}. These are linked with formula $d_{ij}^{out} = f_{t_i}(d_{ij}^{in})$ where f_{t_i} defines the size of output data for task t_i based on the input data size.
- d_i denotes the size of data processed by task t_i.
- d_{ijkl} denotes the size of data to be sent from service s_{ij} to service s_{kl}. d_{ij}^{out} can be sent and/or partitioned into input files of successors. In particular, all the data can be sent to all the successors or it can be partitioned into non-overlapping parts that will be distributed for parallel processing by the following tasks.
- $t_{ij}^{exec}(d_{ij}^{in})$ – the execution time of service s_{ij},
- t_{ijkl}^{tr} – additional time for data conversion between output/input formats if connected services are offered by various providers,
- $t_i^{st} : i \in |V|$ – the time at which service s_{ij} chosen to execute t_i starts processing it, we have $\forall_{i,k:(v_i,v_k)\in E}$ $t_k^{st} \geq t_i^{st} + \sum_j t_{ij}^{exec} + \sum_{j,l} t_{ijkl}^{comm} + \sum_{j,l} t_{ijkl}^{tr}$. t_{ij}^{exec} will be larger than 0 only for one j. Similarly, for the given i and k t_{ijkl}^{comm} and t_{ijkl}^{tr} will be larger than 0 only for one pair of l and k since only one service per node i and one per node k will be selected.
- $t^{workflow}$ – the workflow execution time i.e. $t^{workflow} = t_{termination}$ $not\exists_q(v_{termination}, v_q) \in E$.

Traditionally, several optimization goals can be considered such as minimization of $t^{workflow}$ with a bound on the total cost of selected services i.e. $\sum d_{ij}^{in} c_{ij} < B$ where B is the budget (problem MIN_T_C_BOUND) and $t^{workflow}$ is the time when the last service finishes. Problem MIN_TC is minimization of $v_{MIN\_TC} = \alpha t^{workflow} + \sum d_{ij}^{in} c_{ij}$. $\alpha > 0$.

It should be noted that some descriptions of services with respect to e.g. execution times or reliability may not be accurate. An adaptive technique for learning of service reliability were proposed by the author and his

team in [15].

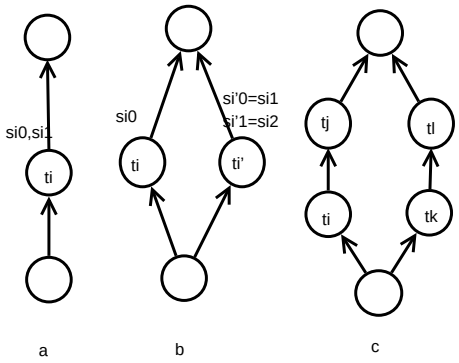


Fig. 4.1: Selection and Parallelization in the Model

4.1. Selection of Solutions and Data Parallelization. The proposed formulation allows to model several issues typical of workflows composed of independent services offered and managed by various users:

1. *selection of alternative solutions* (services) $s_{i0}, ..., s_{i(|S_i|-1)}$ to particular task t_i. This is possible thanks to the well known service selection concept i.e. selecting only one service out of the ones defined for the given task (one of several services is selected for task t_i in Figure 4.1a).
2. *data parallelism* – partitioning of input data for parallelization of computations on this data. This can be achieved in the model in several configurations, depending on the needs:
   (a) parallelization of task t_i where input data is divided and possibly processed by services s_{i0} and s_{i1} or s_{i0} and s_{i2} in parallel. Note that s_{i1} and s_{i2} are modeled as exclusive services which combines selection with parallelization. Task t_i is split into t_i and t'_i which are functionally equivalent (Figure 4.1b) and initial services are assigned to these two tasks.
   (b) parallelization of a complex task (composed of more than one task) by possibly execution of various sub-solutions in parallel. Input data is divided into flows so that $t_i \rightarrow t_j$ and $t_k \rightarrow t_l$ are executed in parallel. It can be that $t_i \neq t_k$ and $t_j \neq t_l$ (Figure 4.1c).

Assuming the user has access to services $s_{i0}, ..., s_{i(|S_i|-1)}$ for task t_i, parallelization where the algorithm determines partitioning of data between possibly all services, can be modeled as splitting services for many tasks (in fact functionally equivalent) as in Figure 4.2 where $\forall_{i,k} s^{'n}_{i0} = s_{in}$.

4.2. Real and Integer Data Sizes. Additional constraints on partitioning of data could be set e.g. $d_i \in Z$ $d_{ijkl} \in Z$ for integer values suitable for partitioning of a set of e.g. pictures of same size for processing in parallel. This makes the problem harder to solve but the algorithms proposed and discussed by the author in Section 5 can handle this case.

4.3. Synchronization. In Figure 4.2 all services are synchronized on the following task t_j which means that even after $s^{'2}_{i0}$ has finished processing its portion of data, t_j will not start until all $s^{'n}_{i0}$ have finished. If there are tasks t_i and t_j which need to process the input data subsequently, the proposed model can allow pushing a portion of data to the following task even if the previous steps on the other portions of data have not completed yet. This can be accomplished as in Figure 4.3. Namely, as soon as t'_i has finished processing its portion of data, it can send results for processing by t'_j without waiting for completion of processing by e.g. t_i.

The question arises into how many virtual tasks processing of the given task should be split. This should cover the number of services which can process the given task. In such a case, potentially all services for the original task t_i and t_j could participate in parallel execution of the corresponding task with the possibility of pushing parts of data from task t_i to task t_j even when execution of other parts of t_i's data is still in progress. Furthermore, this could be set up to the potential number of partitions the initial data could be divided into to allow pushing smaller portions of data sooner.

In one workflow there may be both tasks with multiple services (for selection of one service for task) and virtual tasks with one service assigned to each of them meant for data parallelization. The former may be used

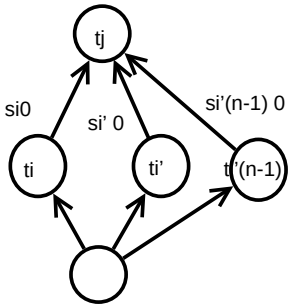


Fig. 4.2: Data Parallelization

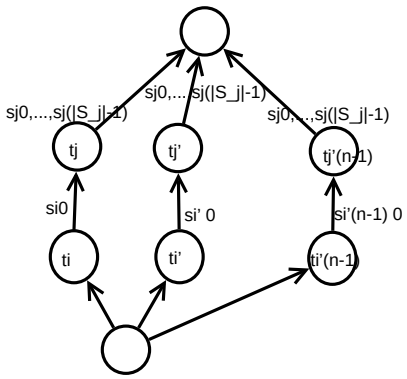


Fig. 4.3: Data Parallelization without Synchronization between Tasks

when there is a task to be executed on a portion of the input data where it is not possible to split data among services or it would be too costly compared to the execution by one service (e.g. a conversion of an image to another format). Data partitioning can be used when it is possible and beneficial to split the data and there are enough services to execute in parallel.

5. Algorithms. If the graph contains only tasks with one service for each, the problem becomes how to distribute data with possibly cost constraints. In this case and assuming that data can be divided into chunks of any size, fast linear programming [28] can be used where variables denote the sizes of data processed by particular services. This also assumes that the execution times for the serviceare linear functions of input data size which does not have to be the case. For service selection, the literature suggests introducing integer variables [3, 4, 36] that denote which service is selected for the particular task.

The author has proposed and implemented three different algorithms to solve the problem [14]:

1. genetic algorithm (GA) – in this case a solution is represented by a chromosome. It encodes both selection of particular services for the tasks and also order of execution for pairs of parallel tasks in the graph for which the order is not determined by the workflow graph.
2. mixed genetic algorithm and linear programming (MGALP) – in this case services are assigned to the tasks by a genetic algorithm and data distribution for the given schedule is determined by linear programming [28],
3. mixed integer linear programming (MILP) – similarly to the known approaches [36, 3, 4], integer variables mean which service is selected for a particular task. Additionally though, real or integer variables denote sizes of data flowing from task to task.

6. Management, Optimization and Execution of Workflow Applications in BeesyCluster. The author has created a workflow management environment for modeling, scheduling and execution of workflow applications, both for the proposed model [14] and also for dynamic selection of services at runtime [13].

The environment uses BeesyCluster as a middleware to access distributed services and is available at `https://lab527.eti.pg.gda.pl:10030/ek/Main` at Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology, Poland.

BeesyCluster allows its users to access accounts on distributed resources through SSH and compile and run applications, among others (Figure 6.1). Furthermore, such applications can be published as BeesyCluster services to which access may be granted to other users. In particular, costs of running services can be defined. BeesyCluster users have virtual purses from which can pay for services published by others. Similarly, users may earn from others running their own services.

Furthermore, the workflow management environment allows:

1. definition of a workflow using a graphical interface (Figure 6.2),
2. selection of services and defining data flows between tasks [10, 12],
3. actual execution of the workflow in a distributed environment [10, 12].

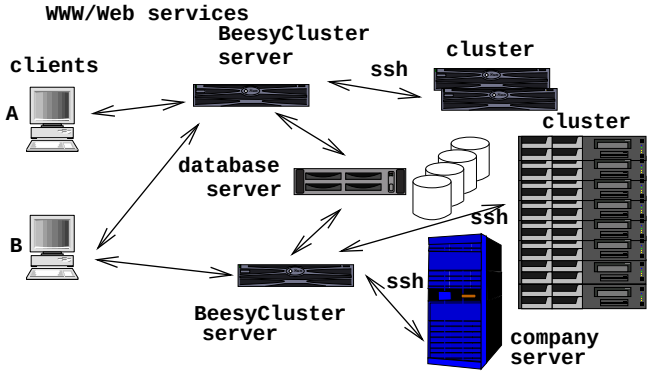


Fig. 6.1: BeesyCluster Architecture

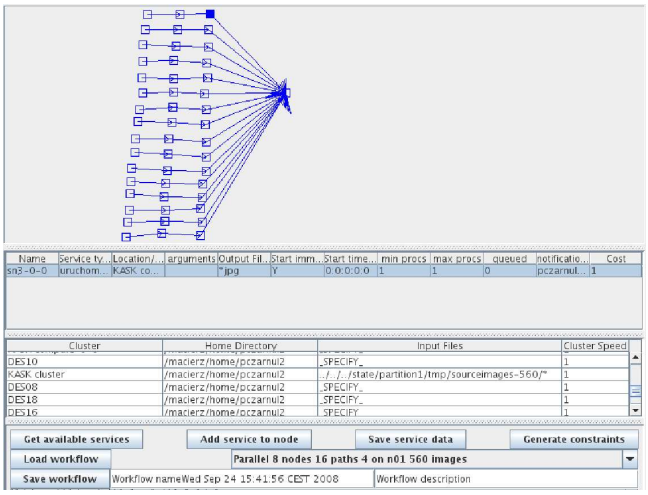


Fig. 6.2: Workflow Editor in BeesyCluster

6.1. Compute Intensive Scientific Workflow: Adaptive Quadrature Distributed Integration. As an example for demonstration of the goals defined in Section 3 a compute intensive application was implemented by the author and analyzed. This application computes a numerical integrate of a given function on the given range with a certain accuracy using the adaptive quadrature integration algorithm presented in [30].

6.1.1. Testbed Environment and Services. The testbed environment consists of 16 nodes installed at the Faculty of Electronics, Telecommunications and Informatics, Gdansk University of Technology. Each node along with services installed on them was accessed by BeesyCluster using SSH. The relative speeds of 8 nodes were 1 while the speeds of the other nodes were 0.575 for the application. Each node uses its local disks to store input and output data. As an example function $f(x) = sin(x)cos(sin(x)) + \frac{1}{x^2}$ was integrated on given ranges. The following services were developed:

partition *a b rpf filecount* – partitions range $[a, b]$ recursively into the given number of files *filecount* and subranges per file *rpf* so that the execution time per subrange is approximately similar. At the given step, for each subrange [a,b] 10 points $c_0, ..., c_9$ are selected so that $a < c_i < b$. Then the maximum of areas of the triangles formed by $(a, f(a)), (c_i, f(c_i)), (b, f(b))$ is selected for each subrange $[a, b]$. Out of all available subranges the one with the largest area is selected and partitioned into two subranges $[a, \frac{a+b}{2}]$, $[\frac{a+b}{2}, b]$. The procedure is repeated until the desired number of subranges is generated.

integration – an application written in C that integrates all subranges from input files in its directory and produces a single output file with extension `out` with the total integrate of the given subranges. The al-

gorithm divides the given subrange recursively like described above until the maximum area of triangles is smaller than 0.000000001 in which case the integrate is approximated by a sum of rectangles.

adder – adds the values supplied in files with `out` extension supplied in its directory

Figure 6.3 presents the workflow in which the initial range $[a, b]$ is partitioned into $filecount$ files each of which contains rpf subranges. Then the initial files with subranges are sent to separate nodes for parallel processing. It is important to note that filecount must be sufficiently large to distribute files with input subranges among parallel tasks for integration especially if services assigned to them run on nodes of different speeds. Similarly, rpf can be set larger than 1 to further improve load balance since it increases the number of initial subranges and makes their computation times more even. The model assumes that the processing time of each input file is the same.

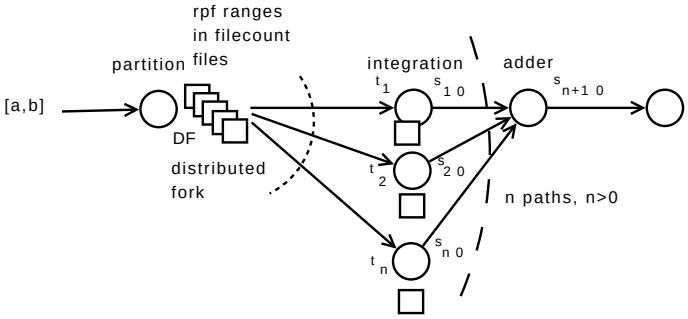


Fig. 6.3: A Parallel Integration Workflow

6.1.2. Workflow Configurations and Results. The algorithm is able to adapt data distribution to various speeds of processing nodes and also does it under budget constraints. The following configurations were tested (optimization goals are noted):

MIN_TC: granularity test (Figure 6.3) – for up to 8 nodes it is best to select 1 file per node as the nodes have same speeds but vary the number of subranges. On the other hand, for 12 and 16 processors we must increase the number of files to let the algorithm balance work between processors of different speeds. Figure 6.4 presents execution times for these settings for range [1,1000]. Setting the number of files or subranges too large causes too much overhead related to opening files, loop overhead with no further gain in load balance. Finally, best settings were selected for following tests.

MIN_TC:parallel (Figure 6.3) – execution times and corresponding speed-ups are shown in Figures 6.5 and 6.7 for three different ranges of $[1, 250]$, $[1, 1000]$ and $[1, 2000]$. It can be seen that the algorithm achieves very good speed-ups for larger ranges even though there are various node speeds. For a small run the overhead of the solution including preparation of directories on the nodes, copying of data take their toll. In fact the speed-up for range [1,1000] is slightly better than for range [1,2000] presumably because of some temporary load on one of the nodes during the runs. It also suggests that this is the upper limit on the speed-up for this configuration as increasing the size does not increase the speed-up.

MIN_T_C_BOUND: parallel under budget constraints (Figure 6.6) – we define the costs of services depending on the clusters they are installed on and the time of day according to Table 6.1. Three logical clusters are distinguished with 8, 4 and 4 nodes (with different costs) respectively. Figures 6.8 and 6.9 show the impact of decreasing the budget on the execution time of the workflow for two configurations: clusters 1 and 2 (total of 12 nodes), clusters 1, 2 and 3 (total of 16 nodes) for day and night. Cluster costs are the same as for the digital photography workflow since the same clusters were used. However, for this particular integration code clusters 2 and 3 (the speed of each node is 0.575) are slower than cluster 1 (the speed of each node is 1). The budget is varied from the minimum budget allowing full parallelization, then 0.9 and 0.8 of this value. Clearly tightening the budget increases execution times as the algorithm does not allow to pass input data to more expensive services causing the use of a smaller number of cheaper but slower paths.

6.2. Comparison of Theoretical Model and Simulation Results. The image processing workflow analyzed and tested in [14] is a good example for comparing theoretical execution times of the adopted model and

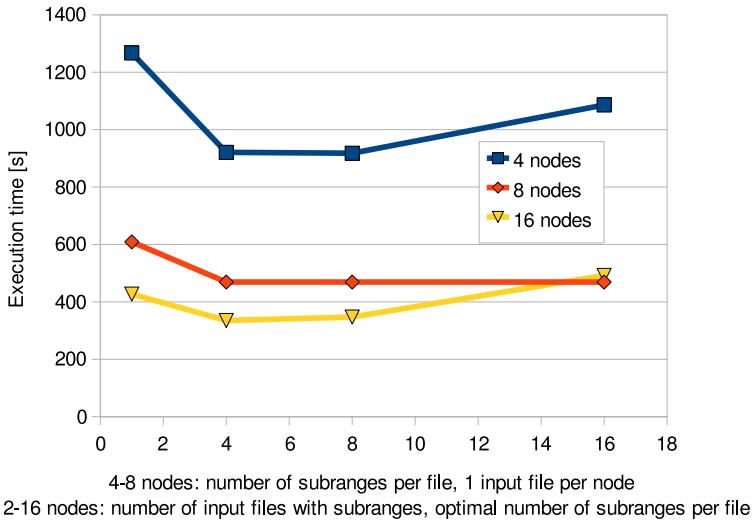


Fig. 6.4: Adaptive Integration on range [1,1000]: Execution Times [s] vs Number of Subranges per File (1-8 nodes) or Files Per Node (16 nodes)

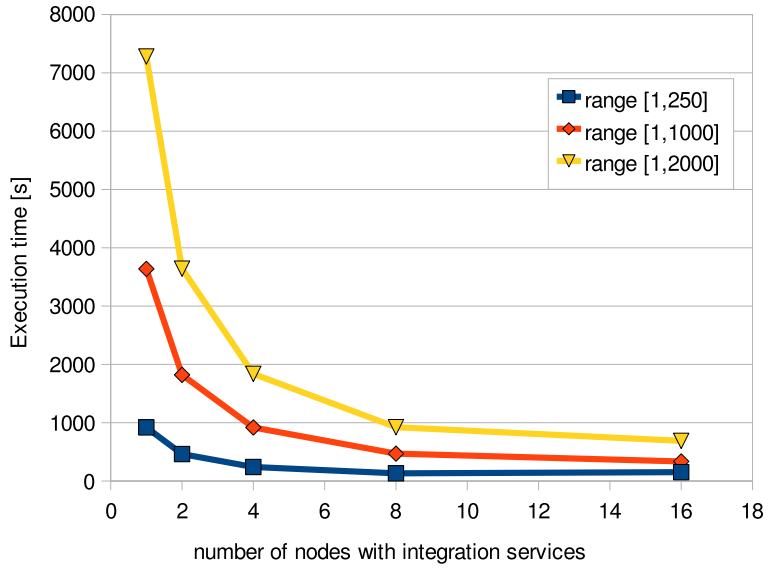


Fig. 6.5: Adaptive Integration: Execution Times [s]

			cost per second	
Cluster	services	node count	day	night
1	$s_{1\,0}$ to $s_{24\,0}$	8	20	10
2	$s_{25\,0}$ to $s_{36\,0}$	4	10	20
3	$s_{37\,0}$ to $s_{48\,0}$	4	15	15

Table 6.1: Services and Cost per Processor Second for Testbed Clusters

simulation results as it involves transfers of large portions of data of different sizes for the three configurations tested. The simulation results shown in [14] were compared to the theoretical model in Figure 6.10. The parameters of the model in this case were computed as follows. The execution time of each path of the workflow

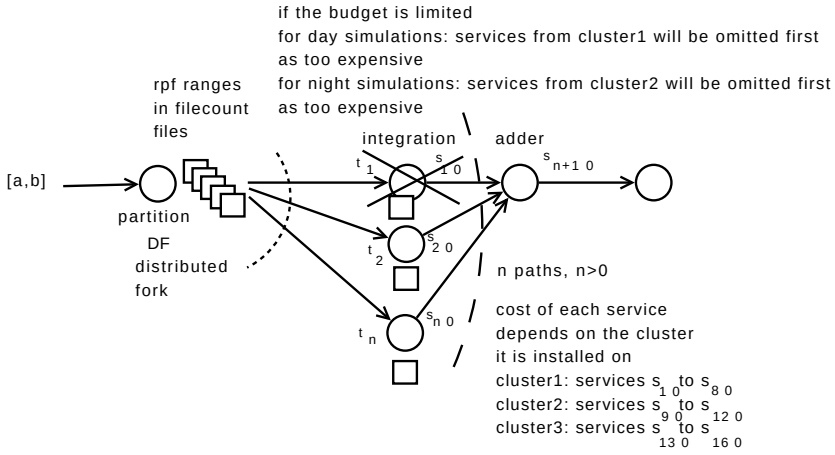


Fig. 6.6: A Parallel Integration Workflow with Budget Constraints

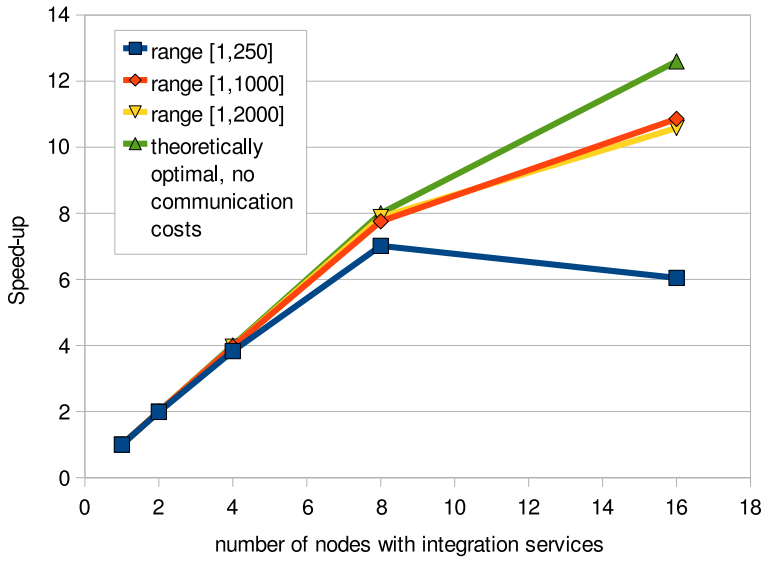


Fig. 6.7: Adaptive Integration: Speed-up

including web album generation as the last workflow task is as follows: execution time $= a + \frac{d}{p}(c + e) + df$ where d is the number of initial images, p the number of paths, a is an accumulated constant from startup times, overhead for preparation of directories for each workflow node, e and c correspond to computation and communication times and df corresponds to the execution time of Web album generation.

The execution times are known as are data sizes d. The execution times of the Web generation phase were read from the system logs and thus it was possible to determine f. The execution time of web album generation is always the same as requires the same number of images. Then linear regression was used to determine a and $c + e$.

The simulation results are very close to the model (Figure 6.10) being slightly too optimistic regarding performance. The source node is a bottleneck in copying data to several following nodes.

6.3. Integration of BeesyCluster and MPI and Overhead of the Solution. It is possible to assess the overhead of the workflow support in BeesyCluster mechanism by comparing execution times to highly dedicated parallel solutions run in the same environment. A C+MPI based implementation can be regarded as a lower bound on the execution time of the service-based BeesyCluster workflow. Obviously neither the

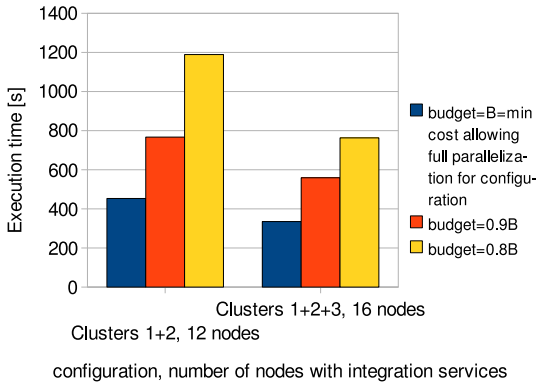


Fig. 6.8: Adaptive Integration: Execution Time [s] under Cost Constraints: Day

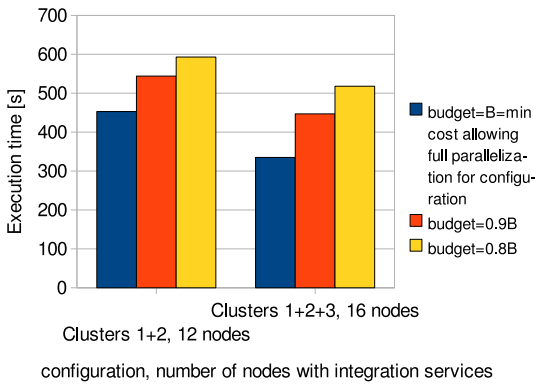


Fig. 6.9: Adaptive Integration: Execution Time [s] under Cost Constraints: Night

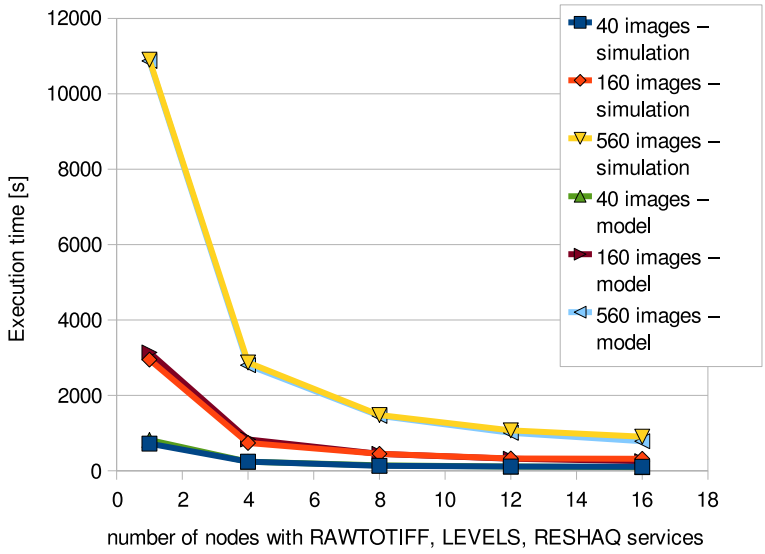


Fig. 6.10: Comparison of Theoretical Model and Simulation Results

standard MPI nor grid-enabled versions such as MPICH-G2 [20], PACX-MPI [21] or BC-MPI [11] offer easy integration of services with performance and cost aware optimization.

Distributed integration was used as an example in the following configurations:

MPI on 2 clusters. Two MPI environments each with 8 nodes were configured. The partitioning application divides the initial range into a predefined number of subranges and saves them to the number of files equal to the number of distinct MPI environments used (1 up to 8 nodes, 2 for 16 nodes in this case) considering relative speeds of the latter. Subsequently, MPI applications launched on the clusters divide the ranges between processes using MPI and compute results for their parts.

workflow in BeesyCluster with services using MPI. Two services were used each of which was implemented as a parallel C+MPI application. For 1-8 nodes one service implemented by one MPI application using from 1 to 8 nodes was used. For 16 nodes, two BeesyCluster services each of which ran an MPI application on 8 nodes just like in the previous example. It allows to assess the overhead of the BeesyCluster layer. Initial data was prepared as in the previous example.

workflow in BeesyCluster. The configuration considered in Paragraph 6.1 was used where the number of services is equal to the number of nodes. Each service is a sequential application.

Figure 6.11 presents a comparison of speed-ups between these three solutions. The reference run used 1

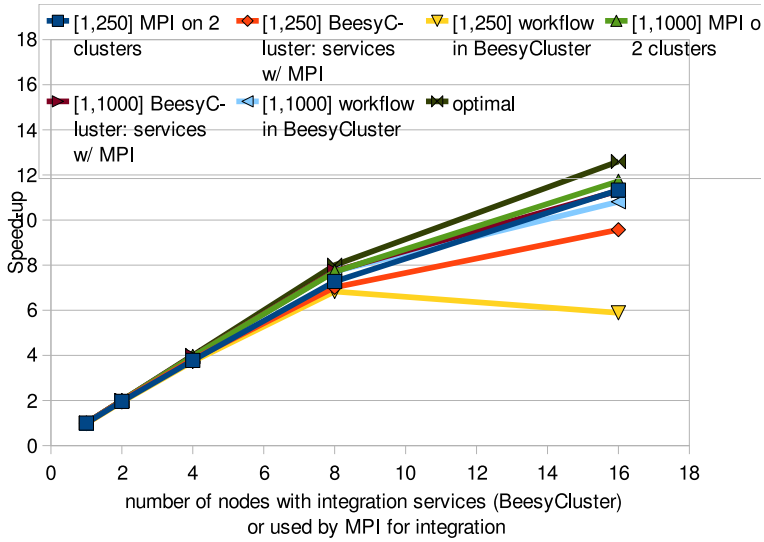


Fig. 6.11: Performance Comparison of BeesyCluster Workflow and MPI: Speed-up

processor.

It can be seen that the MPI implementation is not far below the theoretical best result while the BeesyCluster based implementations introduce slightly more overhead. It is not significant for the [1,1000] range but is visible for short runs for range [1,250] and 16 nodes. This is because the overhead introduced by the Java EE implementation is significant compared to the short execution time in this case. Nevertheless, for longer runs the performance of the workflow solution is very good. Secondly, as expected the overhead of the workflow with MPI-based services is much smaller than for a larger number of sequential services. The overhead of the workflow with MPI-based services compared to an MPI-only implementation on 16 nodes for range [1,1000] was 13 seconds. This encourages to use workflows with MPI-based services as it allows very easy integration of services with scheduling and data partitioning incorporating performance and costs which is not available in MPI implementations.

7. Summary. The paper formulated a problem on service selection and scheduling with data distribution encountered in the integration of distributed services offered by various providers. Both the model and the algorithm were implemented in BeesyCluster allowing to consume distributed services from various providers. Additionally, an easy-to-use workflow editor and an execution engine were implemented. As an example, a distributed numerical integration was constructed as a workflow application were prepared and executed a distributed service-based environment. The solution achieves good speed-ups. It is able to minimize the execution time and keep the total cost of selected services below a threshold. The paper demonstrates the impact of input data partitioning on the scalability of the approach. Secondly, simulation results are compared to the theoretical model which confirms its correctness. Finally, the overhead of the implementation is analyzed compared to a pure parallel MPI implementation. Results for an integrated workflow/MPI solution are also presented.

Acknowledgment. Research partially sponsored by research grant N N516 383534 "Strategies for management of information services in distributed environments".

REFERENCES

[1] *Kepler user manual*, May 2008.
[2] *CUDA Programming Guide 3.1.* `http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf`, June 2010.
[3] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, *Constraint driven web service composition in meteor-s*, in Proceedings of IEEE International Conference on Services Computing (SCC'04), 2004, pp. 23–30.

[4] R. Aggarwal, K. Verma, J. Miller, and W. Milnor, *Dynamic web service composition in meteor-s*, technical report, LSDIS Lab, Computer Science Dept., UGA, May 2004.
[5] S. A. Ahson and M. Ilyas, eds., *Cloud Computing and Software Services: Theory and Techniques*, CRC Press, 2011. ISBN 978-1-4398-0315-8.
[6] R. Buyya, ed., *High Performance Cluster Computing, Programming and Applications*, Prentice Hall, 1999.
[7] G. Canfora, M. D. Penta, R. Esposito, and M. Villani, *A Lightweight Approach for QoS-Aware Service Composition*. ICSOC 2004 forum paper, IBM Technical Report Draft.
[8] ———, *Qos-aware replanning of composite web services*, in Procs. of 2005 IEEE International Conference on Web Services, vol. 1, Res. Centre on Software Technol., Sannio Univ., Italy, July 2005, pp. 121–129.
[9] J. Cardoso, A. Sheth, and J. Miller, *Workflow quality of service*, tech. report, LSDIS Lab, Department of Computer Science, University of Georgia, Athens, GA 30602, USA, March 2002.
[10] P. Czarnul, *Integration of compute-intensive tasks into scientific workflows in beesycluster*, in Computational Science – ICCS 2006, vol. 3993 of LNCS, Springer, 2006, pp. 944–947.
[11] ———, *Bc-mpi: Running an mpi application on multiple clusters with beesycluster connectivity*, in Proceedings of Parallel Processing and Applied Mathematics 2007 Conference,, Springer Verlag, May 2008. Lecture Notes in Computer Science, LNCS 4967.
[12] ———, *A JEE-based Modelling and Execution Environment for Workflow Applications with Just-in-time Service Selection*, in proceedings of Grid and Pervasive Computing, Geneva, Switzerland, May 2009.
[13] P. Czarnul, *Modeling, run-time optimization and execution of distributed workflow applications in the JEE-based BeesyCluster environment*, The Journal of Supercomputing, (2010), pp. 1–26. 10.1007/s11227-010-0499-7, http://dx.doi.org/10.1007/s11227-010-0499-7.
[14] P. Czarnul, *Modelling, optimization and execution of workflow applications with data distribution, service selection and budget constraints in beesycluster*, in Proceedings of 6th Workshop on Large Scale Computations on Grids and 1st Workshop on Scalable Computing in Distributed Systems, International Multiconference on Computer Science and Information Technology, 2010, pp. 629–636. Wisla, Poland.
[15] P. Czarnul, M. Matuszek, M. Wjcik, and K. Zalewski, *BeesyBees — agent-based, adaptive & learning workflow execution module for BeesyCluster*, in Faculty of ETI Annals, Information Technologies vol. 18, Gdansk, Poland, 2010.
[16] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, *Pegasus : Mapping Scientific Workflows onto the Grid*, in Across Grids Conference, Nicosia, Cyprus, 2004. `http://pegasus.isi.edu`.
[17] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*, in Open Grid Service Infrastructure WG, June 22 2002. Global Grid Forum, `http://www.globus.org/research/papers/ogsa.pdf`.
[18] T. Glatard, J. Montagnat, D. Lingrand, and X. Pennec, *Flexible and Efficient Workflow Deployment of Data-Intensive Applications On Grids With MOTEUR*, International Journal of High Performance Computing Applications, 22 (2008), pp. 347–360.
[19] Gridbus Project, *Workflow language (xwfl2.0)*. `gridbus.cs.mu.oz.au/workflow/2.0beta/ docs/xwfl2.pdf`.
[20] N. Karonis and B. Toonen, *Mpich-g2 – a grid-enabled implementation of the mpi v1.1 standard*. `http://www.hpclab.niu.edu/mpi/`, Department of Computer Science at Northern Illinois University and Mathematics and Computer Science Division (MCS) at Argonne National Laboratory.
[21] R. Keller and M. Mller, *The Grid-Computing library PACX-MPI: Extending MPI for Computational Grids*. `www.hlrs.de/organization/amt/projects/pacx-mpi/`.
[22] Laboratory of Parallel and Distributed Systems, MTA SZTAKI, Hungary, *Parallel Grid Runtime and Application Development Environment, User's Manual, ver. 8.4.2*.
[23] B. Ludascher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao, *Scientific Workflow Management and the Kepler System*, Concurrency and Computation: Practice & Experience, Special Issue on Scientific Workflows, (2005).
[24] R.-Y. Ma, Y.-W. Wu, X.-X. Meng, S.-J. Liu, and L. Pan, *Grid-enabled workflow management system based on bpel*, Int. J. High Perform. Comput. Appl., 22 (2008), pp. 238–249.
[25] S. Majithia, M. S. Shields, I. J. Taylor, , and I. Wang, *Triana: A Graphical Web Service Composition and Execution Toolkit*, in IEEE International Conference on Web Services (ICWS'04), IEEE Computer Society, 2004, pp. 512–524.
[26] Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface Standard*, July 1997.
[27] C. Patel, K. Supekar, and Y. Lee, *A QoS Oriented Framework for Adaptive Management of Web Service based Workflows*, in Proceedings of the 14th International Database and Expert Systems Applications Conference (DEXA 2003), LNCS, Prague, Czech Republic, September 2003, pp. 826–835.
[28] M. M. Syslo, N. Deo, and J. S. Kowalik, *Discrete Optimization Algorithms*, Prentice-Hall, 1983.
[29] M. Wieczorek, A. Hoheisel, and R. Prodan, *Towards a general model of the multi-criteria workflow scheduling on the grid*, Future Generation Comp. Syst., 25 (2009), pp. 237–256.
[30] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, Prentice Hall, 1999.
[31] Yingchun, X. Li, and C. Sun, *Cost-effective heuristics for workflow scheduling in grid computing economy*, in GCC '07: Proceedings of the Sixth International Conference on Grid and Cooperative Computing, Washington, DC, USA, 2007, IEEE Computer Society, pp. 322–329.
[32] J. Yu and R. Buyya, *A taxonomy of workflow management systems for grid computing*, Journal of Grid Computing, 3 (2005), pp. 171–200.
[33] J. Yu and R. Buyya, *A budget constrained scheduling of workflow applications on utility grids using genetic algorithms*, in Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC 2006), Paris, France, June 2006.
[34] ———, *Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms*, Scientific

Programming Journal, (2006). IOS Press, Amsterdam.
[35] J. Yu, R. Buyya, and C.-K. Tham, *Cost-based scheduling of workflow applications on utility grids*, in Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science 2005), IEEE CS Press, Melbourne, Australia, December 2005.
[36] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Sheng, *Quality driven web services composition*, in Proceedings of WWW 2003, Budapest, Hungary, May 2003.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 239–255. http://www.scpe.org
ISSN 1895-1767

DEEPG: DUAL HEAP OVERLAY RESOURCE DISCOVERY PROTOCOL FOR MOBILE GRID

Y. MOHAMADI BEGUM* AND M. A. MALUK MOHAMED†

Abstract. Inherent characteristics of mobile devices make resource discovery in mobile grid challenging. Adding to the complexity is heterogeneity and hence multi-attribute management of these devices. Decentralized resource discovery using DHTs incur maintenance overheads when changes in attributes are continuous and rapid. The paper proposes a novel dual heap non-DHT overlay (DEEPG) for query resolution dealing with inherent mobile characteristics along with a mathematical model to estimate the problem size. Simulation results show that DEEPG reduces search bound on query resolution and results in limited maintenance overheads. The results also indicate that higher the query sum reduced is the number of lookups for exact match query resolution.

Key words: Mobile grid, Resource discovery, DHT, Multi-attributes, Overlay

1. Introduction. A computational grid constitutes a number of clusters, each of which may be under different virtual organizations (VO) [16]. Integration of mobile devices into grid computing infrastructure has been widely increasing. Mobile grid models [27, 52, 34] have evolved contributing to computational requirements of grid users. Such mobile grids find applications in domains such as e-health, disaster management, multimedia transfer and processing, and other context-sensitive applications. These applications demand resources involving a variety of attributes including the location of the mobile node. Attributes of grid resources are either static or dynamic. Static attributes of a resource include architecture type, memory configuration, CPU clock frequency, operating system, etc. Dynamic attributes include CPU queue length, memory utilization, CPU utilization, etc.

The volatile and dynamic environment of the mobile Grid calls for sophisticated mechanisms for resource discovery and selection [30]. Discovery process in mobile grid needs to handle issues like node dynamism, node mobility and heterogeneity with frequent attribute changes. Resource requirements of a job submitted to a grid can be determined a priori using historic data or appropriate prediction techniques [46]. Query-based approaches such as Globus MDS [15], Legion [44], Condor [33], etc. select candidate nodes from a pool of resources to satisfy requirements specified by grid users in the form of queries. Query resolution requires up-to-date information on widely-distributed resources. Organizing nodes in a mobile grid to facilitate query resolution is not trivial because of the highly dynamic availability of the participating mobile nodes and continuous, rapid change(s) in some of the attributes of the resources.

Centralized and hierarchical approaches for resource discovery in grid [12] are not suitable for resource discovery with multiple dynamic-query attributes. Such approaches are prone to single point of failure, lead to network congestion and also are not scalable [39]. Alternatively, P2P content distribution technology [2] is widely used for large-scale resource sharing and discovery. Unstructured P2P networks [31] have a variety of applications including resource discovery in grid [22]. In contrast to unstructured systems, DHTs (Distributed Hash Table) are used to design structured P2P systems to avoid flooding of query messages [39] thus reducing network traffic substantially. Some structured DHT-based systems [9, 13, 1] extend existing structured P2P routing substrate [24, 5, 48] for large-scale resource discovery. Systems such as [28, 4] are non-DHT overlays.

The best possible job-resource pair can be chosen only with sufficient information on multiple attributes and subsequent match-making. Query resolution is complex as the number of attributes describing a resource increases. A mapping is required to map attributes in a multi-dimensional space into a single-dimensional one. Space filling curves [51] perform such mapping to facilitate resource discovery. However, as the number of attributes increases, they do not perform well because of locality problem. Further, if the data distribution is skewed, it results in non-uniform query processing load for peers [39].

Various issues in existing DHT-based discovery protocols are as follows: (i) Few attributes of resources are extremely dynamic requiring frequent updates in DHTs. For example, CPU utilization tends to be a continuously changing attribute and sometimes bursty. The corresponding DHT needs to be updated to reflect the changes in this attribute contributing to overheads proportional to the structure of the DHT; (ii) Different types of attributes may require different indexing mechanisms. Therefore, systems especially those using multiple DHTs

*Software Systems Group, M.A.M. College of Engineering, Tiruchirappalli, India, Email: ssg_mohamadi@mamce.org
†Software Systems Group, M.A.M. College of Engineering, Tiruchirappalli, India, Email: ssg_malukmd@mamce.org

for multi-attribute management result in high maintenance overhead for DHT structures; (iii) A DHT peer acts only as an index to the appropriate resource and hence it needs to send the information on suitable resource to the requesting node. As the hash indices serve as only secondary index structures, additional mechanism is required for locating nodes in the presence of network delays; iv) DHTs require every resource-value pair to have a unique key, limiting its scalability when used for Grid resource discovery [36]. All these issues motivate the use of a non-DHT overlay. The heap overlay proposed in this paper is a non-DHT overlay used to organize the grid resources, instead of a separate hash index structure on it.

The rationale behind using heap data structure is analyzed here. The topology of the overlay network dictates how the participating nodes can communicate with each other in resolving a query. The topology should be robust enough to accommodate frequent joining and leaving of the nodes. In comparison with ring-based topologies, a heap serves better. N being the number of nodes in a VO, a ring requires O (N) for insertion, detection, and deletion. Insertion and deletion are done in constant time once the link to be modified is identified. However to identify the link, on an average N/2 inspections are required. For a heap the order of complexity is O (log (N)) and is better for large N. Of course, for very small values of N, the complexity of the procedure causes more time for heap compared with ring. This advantage of the ring when N is small can normally be ignored. Heap thus qualifies as the most appropriate overlay topology for large-scale resource discovery in comparison with other data structures. Such an organization of resources as a heap is vital because of complex search queries on large number of grid resources characterized by multiple attribute values.

Heap is a data structure with the ability to organize nodes based on arithmetic sum or individual values of its attributes. DEEPG creates a two-level heap. In sum-heap, resources are arranged based on their attribute sum and nodes with same attribute sum are clustered together. Resolving a query involves examining resources for individual attributes. All resources whose attributes make a lesser sum are not qualified for assignment to the job and hence not examined at all. This quickens query resolution process. At the second level is the attribute heap organizing resources based on one of the attributes. Resources qualified from the first-level sum-heap are examined in attribute heap to determine their fitness for the job. If these resources do not satisfy rest of the attributes, the system search for resources with higher sum and repeat the search process.

The paper proposes a novel protocol to determine suitable node(s) to host a process as a part of local scheduling of the cluster administered by a VO. DEEPG serves the problems associated with mobility and resolves exact match queries for multi-attribute resource discovery in a dynamic grid. In addition to the proposed protocol a mathematical model for determining the problem size is devised. The model calculates total number of unique processors with all possible combinations of attributes together contributing a given attribute sum. A single node serving as an index in the sum-heap, clusters resources whose attributes may differ in attributes individually, but collectively representing the same attribute sum. Simulation results presented show how DEEPG reduces the search bound by clustering all such resources with same attribute sum.

The rest of the paper is organized as follows. Section 2 presents the related work in this area of research. In section 3 the background and system design are discussed. Section 4 describes the DEEPG protocol. In section 5 an estimate of the problem size is presented. In section 6 various experiments and results obtained are evaluated for performance. Section 7 concludes and gives an insight into future enhancements.

2. Related Work. Various decentralized resource discovery techniques in grid, driven by P2P network model are investigated in [39]. Structured P2P resource discovery in grid is done by either using the existing DHTs or by augmenting DHTs to support additional functionality. DHT-based P2P resource discovery systems like Chord [24], CAN [47], Pastry [5] and Tapestry [8] are suitable for single-attribute queries and also fail to handle fast-changing resource attributes. Therefore, some systems augmented these DHT routing substrates to handle multiple-attribute queries. Such systems used either single DHT or multiple DHTs for resolving dynamic multi-attribute range queries. Some example systems are discussed below.

In AdeepGrid [42], a d-dimensional attribute space for both static and dynamic attributes is mapped to a single DHT network. Node dynamicity and changes in dynamic attribute(s) may sometimes map attribute space to a different node and therefore results in maintenance overheads. LORM [19] relies on a single DHT to distribute resource information among nodes in balance with its hierarchical structure and claims to incur low overheads. SWORD [1] also uses a single DHT to locate a set of machines matching user-specified constraints on both static and dynamic node characteristics, including both single-node and inter-node characteristics. SWORD uses the same principle as Mercury [4], although the latter is non-DHT based overlay. Mercury requires explicit load balancing mechanism as it does not apply hashing resulting in non-uniform data partitioning. Also

it creates a hub for each attribute and replicates data items on each hub. This is not suitable for a network with high degree of dynamism in nodes.

Xenosearch [13] uses one DHT per attribute and resolves multi-attribute query by aggregating results from every DHT. So it serves as a poor choice when number of attributes of resources is very high. Systems proposed in [3, 43] are based on space filling curves [51] and have drawbacks as discussed earlier. Also they are DHT-based in contrast to SOG [36] which does not use any DHT. However, SOG organizes grid nodes into groups based on their statistical characteristics and cannot be used effectively in a mobile grid where resources vary in their attributes quite often.

Adaptive [25] also supports updates and queries for both static and dynamic resource attributes, again using multiple DHTs. When the attribute value changes to the extent that it is mapped to a new node, the previous mapping is erased. Systems with multiple DHTs are affected by maintenance overheads and [53, 19, 45] attempt at reducing these overheads. Multi-attribute addressable network (MAAN) [9] extends Chord [24] to support multi-attribute and range queries. It maintains multiple DHTs, one per attribute. Nodewiz [41] maintains a single distributed index and hence the update and query traffic is independent of the number of attributes. DIndex [18] has a distributed indexing component in support of range queries.

CONE [6] is a distributed heap-based data structure layered on Chord [24]. CONE uses the DHT only for node joins and departures, and not for querying. A tree based scheduling with heap sort is described in [29]. In majority of these systems, the cost of maintaining the structure of DHT(s) in the presence of potentially frequent node joins and departures is a challenge. Further, in DHTs there is no direct support for complex queries including range queries, aggregate queries and nearest-neighbor queries. Also a majority of these systems employ consistent hashing [26] where removal or addition of a node changes only the set of keys owned by that node with adjacent nodes. This leaves all other nodes unaffected and is ideal for a dynamic system. However, in a heterogeneous mobile grid the adjacent node may have poor attributes. Such a node is forced to perform additional data management because its neighbor(s) departed. This calls for frequent load balancing. So consistent hashing may not hold good for a highly dynamic mobile grid.

Some non-DHT structured overlays attempt to handle the above issues by organizing overlays based on the attributes of the resources. SkipNet [35] enables systems to preserve useful content and path locality, while Skip tree graphs [17] support aggregation queries and broadcast/multicast operations. Mercury [4] is non-DHT-based and creates a routing hub for each attribute in the application schema while RCT [49] organizes resources on the basis of selected primary attributes. Other non-DHT overlays include ACOM [10], BATON [21] and multi-way tree [20] structures. In [50] an attribute-based overlay is explored where each peer is characterized by a single set of attributes and the peers satisfying a given range or k-nearest-neighbor query are looked up.

Challenges in resource management in mobile grid including resource discovery are addressed in [30, 32, 11]. The M-Grid approach [37] handles disconnected operation service in mobile grid, but fails to address the heterogeneity issue. A proxy-based approach [23] handles mobility by grouping mobile devices located on the same subnet and presenting the group as a single virtual resource. In [27] there is a central entity close to the Base Station (BS) or on the BS that handles instability in mobile grid. To the best of our knowledge we find no approaches that specify a structured topology for mobile nodes to facilitate discovery. DEEPG uses an overlay topology that is simpler and incorporates strategy to handle mobility of devices while performing resource discovery.

3. System Architecture.

3.1. System Model. The mobile grid for this work resembles the architecture as proposed in [38]. The grid as shown in Fig. 3.1, constitutes a collection of various service areas called cells occupied by mobile nodes (MN), each governed by a BS. Each cell is termed a Basic Service Set (BSS) as per the IEEE 802.11 based wireless LAN nomenclature. All BS are connected by a wired network enabling them to communicate to each other. A dedicated server of the grid called High-level Scheduler (HS) forwards job requests to BS after determining one among the multiple BS. The BS in turn, does local scheduling by locating an appropriate node using DEEPG to host the process or a migrated process, if any.

To realize a mobile grid, P2P system is an attractive architectural alternative to the traditional client-server computing. A number of critical, real-time, computationally high-end applications can be successfully implemented on a mobile P2P grid. Further, P2P network is self-organizing, which is a key advantage for a dynamic network. They offer efficient search/location of nodes. The mobile grid here adopts a super-peer network model, in which the BS acts as super-peer and all MNs in its BSS are peers. The HS submits a typical

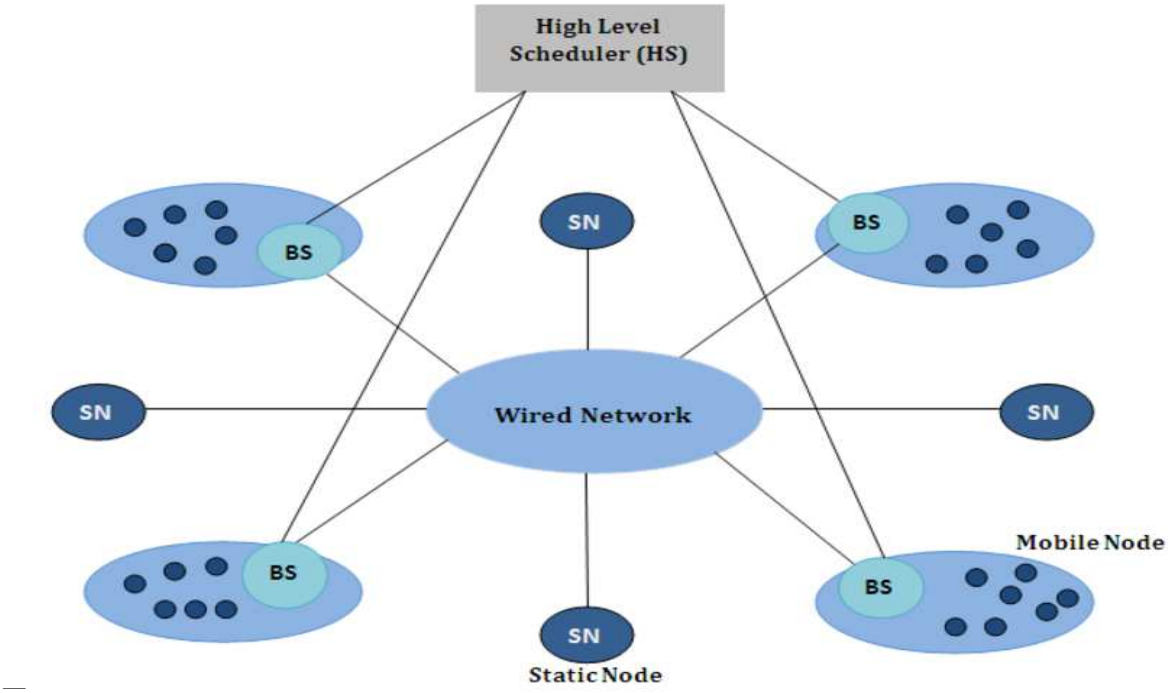


Fig. 3.1: A Mobile Grid connecting HS and BS

job-resource request pair to the BS to determine the suitable host. The BS uses DEEPG to allocate the process peer(s) that matches the specified resource requirements.

3.2. Overlay Design. DEEPG is built on the notion of organizing the participating nodes of a VO to perform local scheduling in it, in the form of a non-DHT overlay. A non-DHT overlay organizes various peers in a structure, so as to facilitate logical connection among the peers. The overlay structure is derived using the attribute information. The peers periodically inform and update their attribute information to their super peer, the BS. The BS is kept aware of the node joins and departures. This information on peers gathered at the BS is broadcast to all peers so that every peer updates its overlay information.

DEEPG helps in solving the exact match, dynamic multi-attribute resource discovery problem. The attributes considered are dynamic and assumed to be in numeric form. The structure of the overlay is dictated by the principle of heaps. A min-heap is a complete binary tree in which at every node the data stored at the node is no more than the data at either child. There are two min-heaps here, one is the sum-heap and the other is the attrib-heap. The sum-heap acts as centralized index server at the BS and the attrib-heap acts as distributed heap in which nodes are arranged based on any one of the attributes, called primary attribute. Each node is responsible for storing its own attributes and change its neighboring peers as and when required.

In DHT-based P2P networks, it is easy to keep multiple single attribute DHTs and select that parameter with the least records to start checking on other attributes. If there are k attributes to be checked, the time bound is k x n where n is the total number of nodes while searching for each attribute. The protocol attempts to reduce this bound with the overlay design. If attributes are treated with equality, then the following heuristic reduces the search space. When multiple objectives are to be satisfied, a weighted sum [14] approach helps and it can be supplemented with optimization. Using the weighted sum approach, the sum-heap is constructed as follows. For example, a unit of one is assigned for each of the attributes of the node. For every participating node, the sum of attributes Sum_a is obtained. A *centralized* min-heap is constructed at the BS using Sum_a as the value and individual nodes such that any node has a sum less than at either child. Table 3.1 shows some sample nodes with three attributes each.

Let the second heap called the attrib-heap, be based on one of the attributes, say CPU frequency and nodes be arranged in the form of a min-heap. This heap is *distributed* so that each node of this heap except the

Table 3.1: Nodes with 3 attributes

Node No.	CPU(GHz)	Memory(GB)	Bandwidth (MBps)	Sum_a
1	1	3	1	5
2	7	4	3	14
3	3	2	2	7
4	4	1	1	6
5	6	3	4	13
6	4	0	0	4
7	4	2	2	8
8	3	1	0	4
9	5	3	3	11
10	0	4	0	4
11	5	4	4	13
12	2	3	2	7
13	7	4	6	17

root node and leaf nodes are responsible for maintaining information on at most three neighbors namely the parent and children if any. Each of the nodes obtained from comparison of sum in the first heap, now point to attrib-heap. This results in a two-level heap. Figs. 3.2 and 3.3 represent corresponding sum and attribute heaps for Table 3.1. In the sum-heap, multiple nodes with same sum are shown as a linked list and other nodes simply as nodes of the heap. In attrib-heap, the nodes are arranged to preserve the ordering of a min-heap based on their CPU capacity.

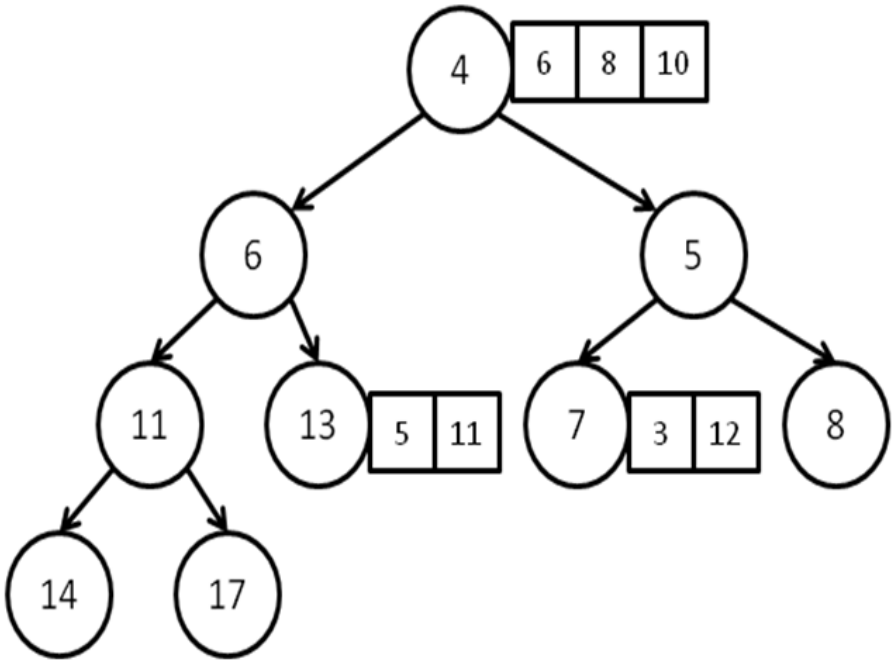


Fig. 3.2: Sum Heap for Table 3.1

If there is a requirement for a node with attributes, 5 GHz clock, 2 GB memory, and 1 USB2 port, the sum is $5 + 2 + 1 = 8$ (For simplicity, fraction values in attributes are rounded). Now it is easy to show that all nodes with a sum of 7 or less are not suitable. However, a node with sum greater than or equal to 8 need not satisfy all the requirements. For example, a node with 10 GHz clock but with 1 GB memory and no USB2 port has a

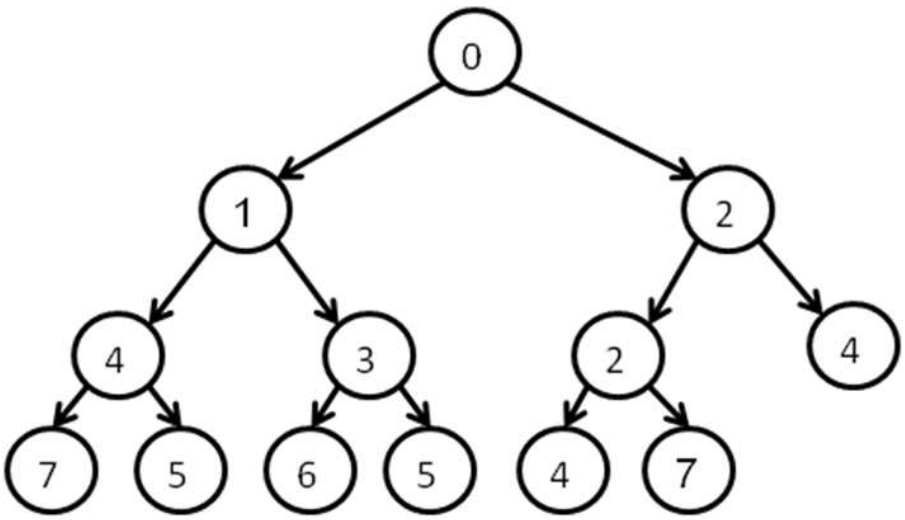


Fig. 3.3: Attribute Heap for Table 3.1

sum of 11 but fails on the memory and USB2 port. This justifies the need for attrib-heap to be constructed. The next section describes how DEEPG utilizes these two heaps for resolving queries.

4. DEEPG Protocol. A mobile grid is basically a finite set of clusters and DEEPG accomplishes resource discovery in one such cluster of the grid. A node entering a BSS and which intends to participate in grid indicates its willingness to the concerned BS. The BS in turn fixes the new node in the existing heaps (sum and attribute) through heap sort. Similarly, any node that leaves the BSS again causes an update in the heaps. Being organized as a heap, the join and leave operations consume O (log N) time. Similarly any change in attributes of the nodes is periodically reflected in the heaps and is possible with the same time complexity. In sequel, the resource discovery process of DEEPG is described. Let N be the total number of nodes in a cluster, participating in the grid. The nodes are denoted as S_i and each node is characterized by k attribute values namely v_1, v_2... v_k. DEEPG protocol comprises two phases for resolving an exact match query and employs one heap per phase. Fig. 4.1 shows the query getting forwarded to the two heaps. Upon receipt of exact-match-query from HS, the BS computes Sum_q, the sum of the attributes specified in the query. Then it examines the sum-heap in Phase-1. Those nodes whose Sum_a equals Sum_q, that is S $=\{s_1, s_2...s_{N'}\}$ are located, discarding nodes whose Sum_a is less than the Sum_q. Thus the nodes to be examined is reduced from N to $N^{'}$, where $N^{'} <$ N. Thus DEEPG filters the candidate resources to a manageable number using the first heap.

In Phase-2, the second min-heap is considered where nodes are arranged in their increasing order of attribute values. Nodes in S are now visited in the order as suggested in this attrib-heap (S reorganized as $S^{'}$) and begin the search. The first node in $S^{'}$ is located in the heap to make further comparisons so as to satisfy other attributes. If this node is unable to satisfy the query criteria, the next node in $S^{'}$ is examined. The query thus gets propagated until the desired node is located or until the sum set $S^{'}$ is exhausted.

If the resource needs are not satisfied by any node examined in $S^{'}$, the next higher value in sum is considered by visiting the subsequent nodes (neighbors) in the sum-heap and repeating the search. The difference in query sum and next sum examined is termed as *slack*. Thus slack represents the degree by which query sum varies.

Table 4.1 depicts a sample of 10 nodes each with an attribute pair. Suppose the query involves search for a processor with attribute pair (1, 4). Sum_q is 5 and search in the sum heap reduces the search set S to 3 nodes. S has 3 nodes with attribute pairs: (3, 2), (4, 1), and (2, 3) whose Sum_a equals 5. Let the attrib-heap be organized based on v_1 and the set S is reordered now as $S^{'}$ = (2, 3), (3, 2) and (4, 1). Considering (2, 3) first, it is found to satisfy processor speed, but fails on memory. Then the next faster processor (3, 2) also fails

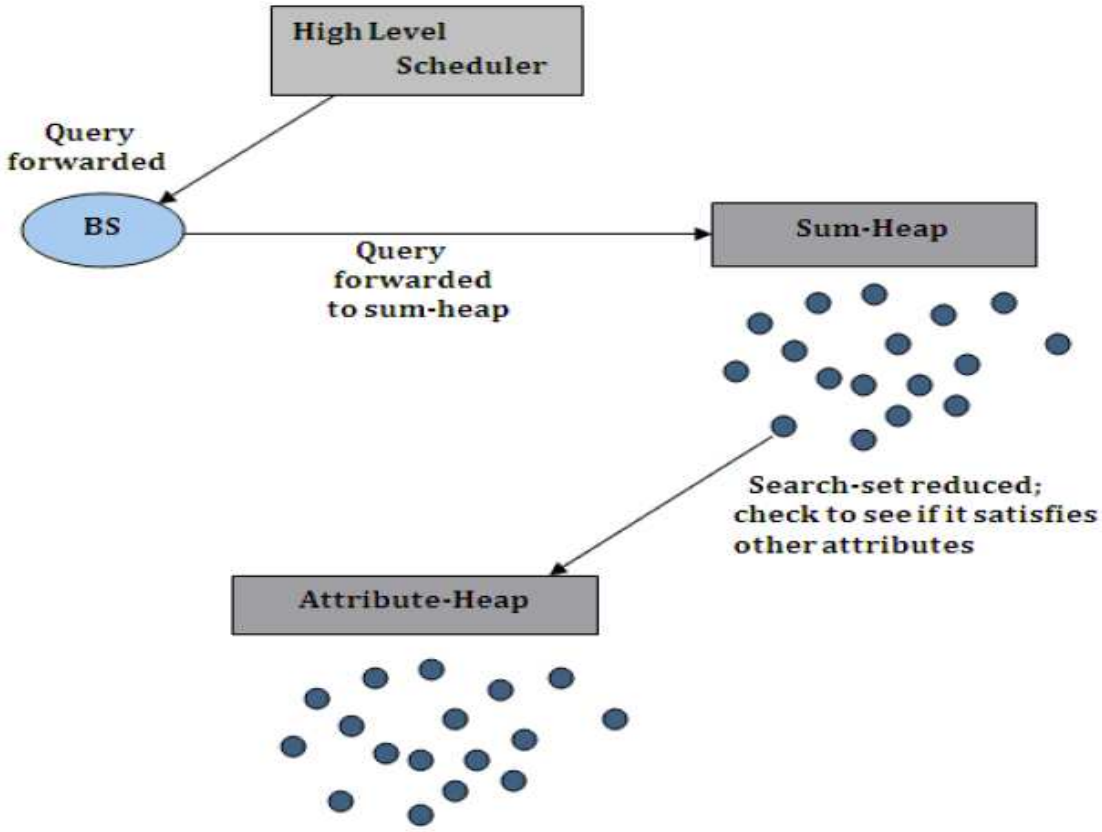


Fig. 4.1: Query forwarded to heap overlays

Table 4.1: Ten nodes with two attributes and their sum

Node No.	CPU(GHz) (v_1)	Memory(GB) (v_2)	Sum_a
1	1	3	4
2	7	4	11
3	3	2	5
4	4	1	5
5	6	3	9
6	4	2	6
7	5	3	8
8	5	4	9
9	2	3	5
10	7	4	11

at memory. Similarly processor with attribute pair (4, 1) fails on memory. So sum-heap is revisited to go one level higher to examine processors with a sum of 6. Here the slack value is 1. This leads to a single processor (4, 2) which also fails to satisfy the query. Then the processors with the sum of 7 are tried and again do not qualify the search. Considering the next higher sum 8 there is just one processor (5, 3) which also fails. The next higher sum 9 has 2 processors (6, 3) and (5, 4). To optimize resource use, pair (5, 4) is tried first as per the reorganized sum set $S^{'}$ and this reports success.

There are two typical cases in these searches. First, there may be multiple nodes satisfying either Sum_q or Sum_q+ slack with same attribute values. In that case, only the first node that satisfies the requirements is selected. Second, a suitable node may not be found even after exhaustively searching all nodes reaching the leaves of the heap. In this case, a failure is reported. Alternatively, the HS may redirect the query to another

BS. Thus there can be one of three outputs: success without slack (exact match), success with slack and failure.

The DEEPG protocol works well for a grid as discussed above and takes care of node dynamism and heterogeneity. If DEEPG is used in mobile grid, it should also handle node mobility and disconnections prevalent in a mobile grid. MN could be self location-aware like a cell phone with GPS capability. Such a MN has the facility to handle location specific activities built-in. The GPS devices have the map of the area of use built-in. Any location specific query is answered by interrogating the built-in map. Therefore, it is sufficient to consider the location specific activity of non-GPS MN alone. The issue of mobility especially for location specific activities of non-GPS MN can be handled by having the location of the device linked to the BS. The location of the non-GPS MN is best known to be within the signal range of the BS only and a change in location can be an attribute updated by the BS whenever the device moves away from its range. Similarly, the BS of the non-GPS MN into whose range the MN moves would update the location of the MN and handle location specific activities. Making location specific activity thus anchored to the BS would solve the issue of mobility. Further, mobility may result in unstable network leading to intermittent connections and poor bandwidth. Bandwidth being an attribute, its changes is reflected periodically in the heaps. Intermittent connections and forced disconnections are out of the scope of this work.

Mobile nodes may frequently operate in a doze mode or disconnect entirely from the network. In doze mode, a mobile host is reachable from the rest of the system and thus when required, can be induced by the system to resume its normal operating mode [7]. Therefore BS simply needs to inform the MN to change from its doze mode to active mode, when it finds that a particular query request can be satisfied by that node. When voluntarily disconnected from the network, the node prior to its departure from grid informs BS which in turn updates the two heaps accordingly.

5. Estimating Problem Size. DEEPG has been proposed with a view to support resource discovery on resources with multiple attributes. As the number of attributes increases, it is expected that total number of resources to be compared for resolving a query increases. Here the relationship between these two factors is examined so as to understand how the proposed protocol minimizes the number of comparisons.

In the sum-heap all nodes whose Sum_a is less than Sum_q are rejected. The efficiency of the protocol can thus be estimated by knowing how many nodes share same Sum_a. The number of nodes with the same Sum_a can be recursively calculated. Thus the theoretical maximum of number of resources with k attributes contributing same Sum_a can be estimated. The recursive algorithm is devised as follows. Let NOP be the number of processors. Consider for example, nodes whose Sum_a is 4 with k attributes, where k is equal to 3. Table 5.1 lists all resources with these combinations of attributes.

Table 5.1: Resource List with $Sum_a = 4$ and k = 3

Node No.	CPU(GHz)	Memory(GB)	USB2Ports
1	4	0	0
2	3	1	0
3	3	0	1
4	2	2	0
5	2	1	1
6	2	0	2
7	1	3	0
8	1	2	1
9	1	1	2
10	1	0	3
11	0	4	0
12	0	3	1
13	0	2	2
14	0	1	3
15	0	0	4

From Table 5.1, it can be observed that for any Sum_a, there can be a deterministic number of unique combinations of a set of attributes. Further recursion is terminated when any of these two conditions are met: (1) NOP with 1 attribute is the number of attributes, which is 1 (as in Eq.5.2); (2) NOP with 0 attribute is the number of attributes, which is 0 (as in Eq.5.3); NOP with Sum_a on k attributes is given by

$$NOP(Sum_a, k) = \sum_{i=Sum_a}^{0} NOP(Sum_a - i, k-1) \tag{5.1}$$

$$NOP(Sum_a, 1) = 1 \tag{5.2}$$

$$NOP(Sum_a, 0) = 0 \tag{5.3}$$

Using the recursive equation Eq.5.1 the function for k attributes and any Sum_a can be obtained. NOP can thus be calculated for any number of attributes and sum. Table 5.2 lists different NOP values for various values of Sum_a and k. As an example, it can be observed that there are 15 unique processors with $Sum_a = 4$ and k = 3 while 5 processors with $Sum_a = 4$ and k = 2; 66 unique processors with $Sum_a = 10$ and k = 3. That is, if the query sum is 15 there can be 800 nodes each with 4 attributes which need to be examined as the worst case. *This implies that the complexity of query resolution increases with the increasing number of attributes as well as the query sum.*

Table 5.2: NOP for various Sum_a and k=2, 3, and 4

Sum_a	NOP		
	k=1	k=2	k=3
0	1	1	1
1	2	3	4
2	3	6	7
3	4	10	16
4	5	15	30
5	6	21	50
6	7	28	77
7	8	36	112
8	9	45	156
9	10	55	210
10	11	66	275
11	12	78	352
12	13	92	442
13	14	105	546
14	15	120	665
15	16	136	800

The recursive equation thus helps in estimating the problem size and therefore its complexity. By selecting at the Sum-heap as many processors as NOP are considered in case nodes exist with all possibilities. In a real case, it is very rare that number of nodes (with same $Sum_a$and k) equal NOP. Assuming the rare case, the actual benefit would be less than the theoretical maximum. Also it should be noted that replicas of resources are not considered while calculating NOP.

6. Experiments and Results. A node in a P2P network that cannot satisfy the query criteria in general, forwards the query to other nodes by unicast, multicast, flooding, etc. The node receiving the request forwards the query to other nodes in case it is unable to answer the query. Thus irrespective of whether the search is in structured or unstructured P2P, efficiency of resource location policy is closely dependent on the request forwarding strategy. To evaluate the proposed request forwarding strategy using DEEPG, we are interested to find answers for the following questions:

1. DEEPG filters the candidate resources to a manageable number using the sum-heap depending on the query sum. What is the relation between query sum and number of resources ignored from examination for successful attempts (with or without slack) and failed attempts? How to generate queries that will fall under these categories? Is there any relation between slack value and number of resources examined?
2. What is the relation between number of attributes and the query resolution time?

3. How does the query resolution time vary depending on whether the resource sought is located in the sum-heap at higher levels, mid-levels, or as the leaves?

4. Can the proposed dual heap withstand/support frequent node joins and departures?

5. What is the impact of frequency of attribute changes over query resolution time?

The proposed resource location protocol DEEPG was experimented using GridSim [40], a Java-based grid simulation toolkit. Using GridSim, a mobile computational grid was realized. The resulting environment consists of multiple users and resources with multiple attributes. Each user has different requirements of resources which are sent as a query to one of the nodes designated as a BS. Depending on the experiment, these queries are either specified or generated randomly. The broker entity in GridSim is emulated as BS and it is delegated the responsibility of super-peer. Although a grid machine can have more than one CPU, for simplicity it is assumed as *one Processing Element per machine per resource*. Henceforth, the terms resource, processor, device and node are interchangeably used.

6.1. Successful and Failed Attempts. Given N nodes in a cell, a query may be resolved by comparing a minimum of 1 and a maximum of N number of nodes. The result of query resolution can be either a successful attempt with or without slack or it can be a failure. For experiment and analysis purpose, framing sample queries that will fall under any of the above three categories is discussed below.

Say for example, when generating nodes with attribute units GHz, GB, and USB2, for every sum greater than 2, a minimum value of 1 is assigned for each of the attributes. Therefore, a sum of 4 has combinations namely, (2, 1, 1), (1, 2, 1) and (1, 1, 2). Every other sum vector would have at least one zero. These combinations thus get dropped. While generating queries, let the queries specify a minimum of 1 for each attribute. These are queries satisfied *without slack (QWS)*. Queries with at least one zero in their attributes are generated in order to get queries that do not get satisfied of a given sum. This query demands Sum_a from 2 attributes, whereas all processors have a maximum of (Sum_a-1) from 2 attributes. Thus these queries would cause failure at the initial sub tree and hence we have to look for nodes with at least (Sum_a+1) sum. These are queries that are satisfied *with a slack value (QS)*.

All queries satisfying the restriction that each attribute is at least 1 are generated to yield success as discussed above. But in practice, it is not necessary that an application needs some GHz. A simple store and read application does not require processing speed as it is controlled by the speed of the communication link. An application that does not require storage of results when the MN gets switched off does not require USB2 port. An application that copies a file from the BS to a USB2 attached drive does not need GB memory. Encouraged by these examples, another set of queries was generated including a zero in one of the attributes. These queries would demonstrate the performance of the protocol under failure mode from the immediate sub-tree.

To generate a query that would eventually fail (QF) after looking at all sums in the complete heap, first the maximum GHz, GB, and USB2 attribute values are recorded. A query that demands one more than the maximum recorded can be generated. Such query requirements therefore cannot be satisfied resulting in failure. For example, if all the processors have a maximum of 2 USB2 ports, looking for a processor with 3 USB2 ports would result in *global failure (QF)*.

Using the above procedure sample queries were generated for searching from a random set of resources. Figs. 6.1 and 6.2 plot various query sum values against the number of resources discarded for all the three cases of queries QWS, QS, and QF. From these graphs we observe that the number of resources discarded increases with the increase in query sum. In the sum-heap, with the increase in the query sum, the distance between the node satisfying the sum and the root node increases. This leads to an increased number of nodes getting discarded from examination. If the query sum is the highest, the node satisfying the same is located in one of the leaves of the sum-heap. A drastic increase in number of nodes ignored from examination and hence a reduction in query resolution time is found in such cases.

Using the above procedure sample queries were generated for searching from a random set of resources. Figs. 6.1 and 6.2 plot various query sum values against the number of resources discarded for all the three cases of queries QWS, QS, and QF. From these graphs we observe that the number of resources discarded increases with the increase in query sum. In the sum-heap, with the increase in the query sum, the distance between the node satisfying the sum and the root node increases. This leads to an increased number of nodes getting discarded from examination. If the query sum is the highest, the node satisfying the same is located in one of the leaves of the sum-heap. A drastic increase in number of nodes ignored from examination and hence a reduction in query resolution time is found in such cases.

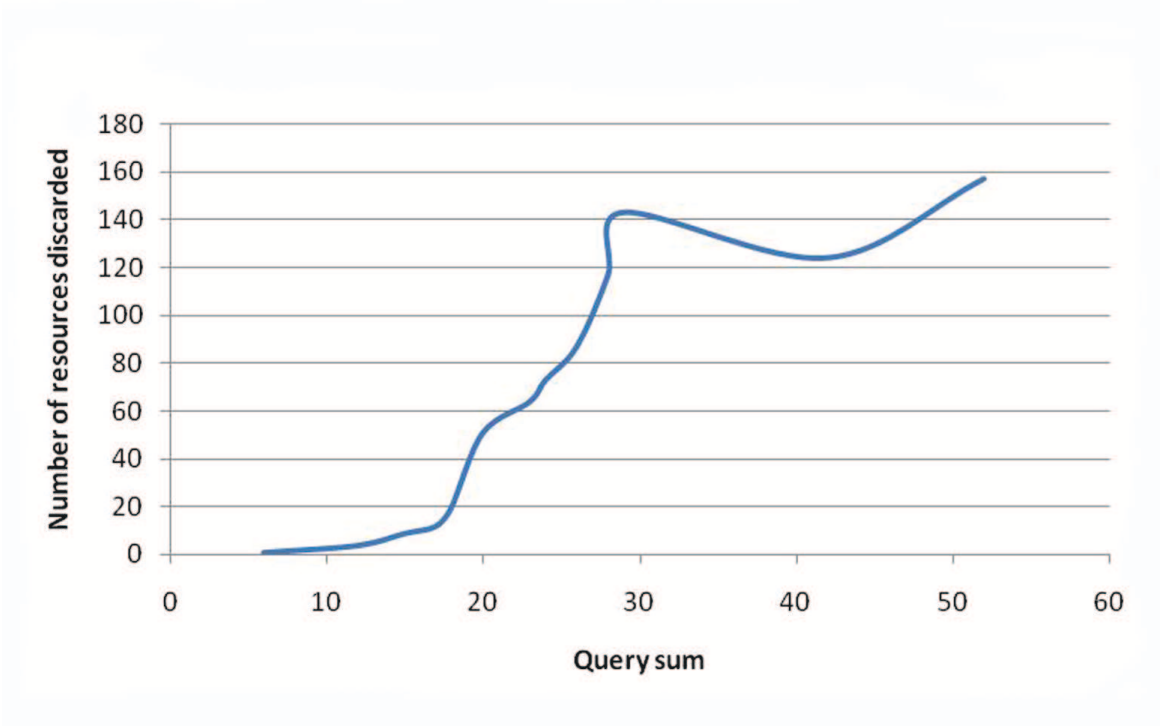


Fig. 6.1: Success without slack: query sum vs number of resources discarded

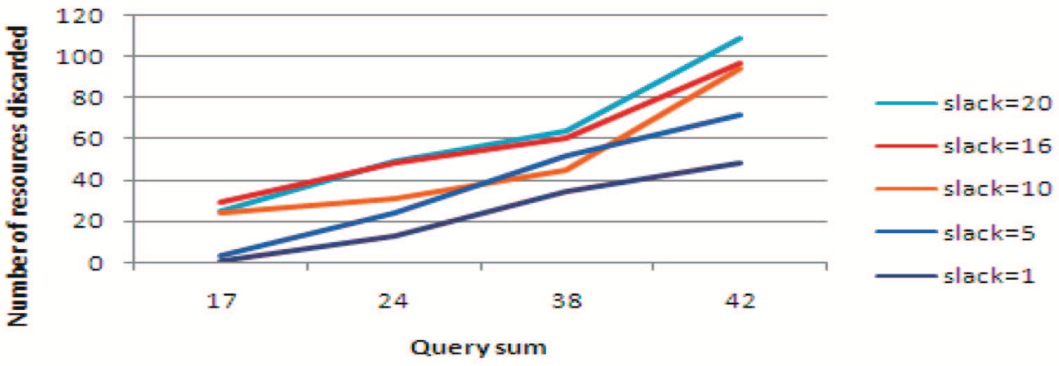


Fig. 6.2: Success with slack:query sum vs number of resources discarded

In the case of QS type of queries, the slope we find in Fig. 6.2 is a function of slack. Experiments were conducted with different slack values. As the slack value increases, the number of resources discarded increases with increasing query sum. Further, higher the slack value, more is the number of resources discarded. In Fig. 6.3 for QS type of queries, number of resources is plotted against number of resources examined for different slack values. Interestingly, we find that there is again a linear relationship between these two values. This indicates that slack value has an effect on both resources examined as well as resources discarded.

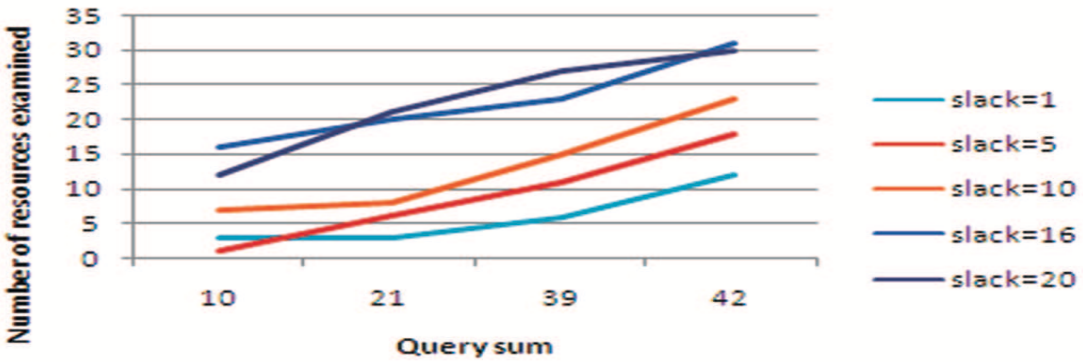


Fig. 6.3: Success with slack:query sum vs number of resources examined

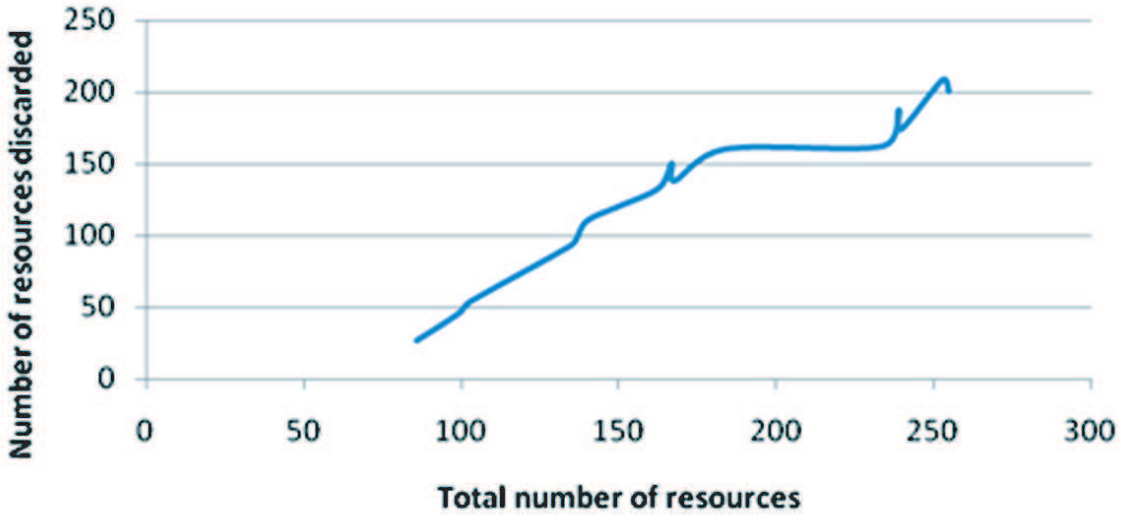


Fig. 6.4: Failure to satisfy a query: number of resources vs number of resources discarded

In the case of QF type of queries, query sum has no effect on query resolution time. This is because failure is reported only after traversing the entire sum-heap. However, we infer from Figs. 6.4 and 6.5 that both number of resources discarded as well as examined is proportional to total number of resources. This is again justified because irrespective of the position of the node representing query sum in sum-heap, search continues and failure is reported only after exhaustive checking.

6.2. Number of Attributes and Query Resolution. The number of attributes that describe a resource affects query resolution time. This is evident from Fig. 6.6 that for k=1, number of resources examined are less and this quantity increases as the k value increases. A resource that satisfies one attribute specified in a query need not satisfy rest of the attributes. Therefore, other resources are examined so as to evaluate their fitness with respect to all other attributes. Thus number of resources examined increases with the increase in value of k as we increase the total number of resources. As shown in Fig. 6.6 there are some values that do not satisfy

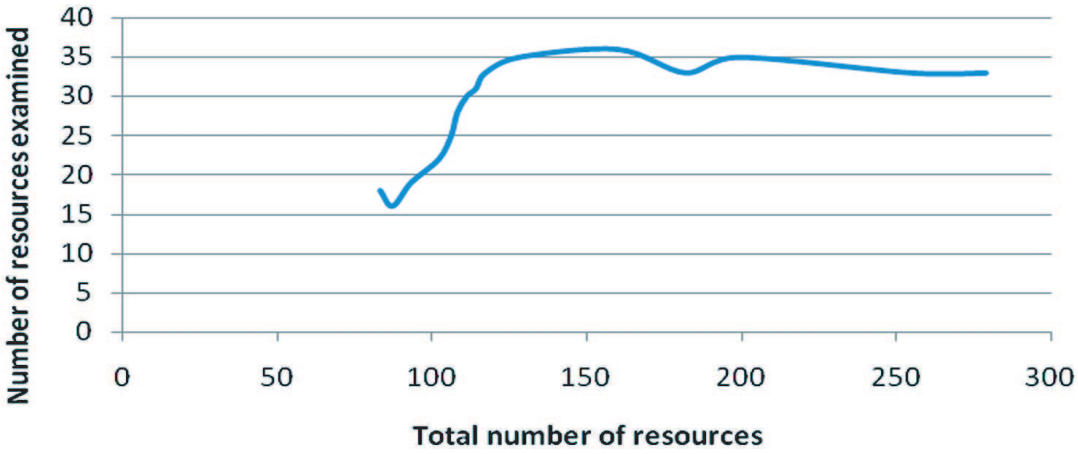


Fig. 6.5: Failure to satisfy a query: number of resources vs number of resources examined

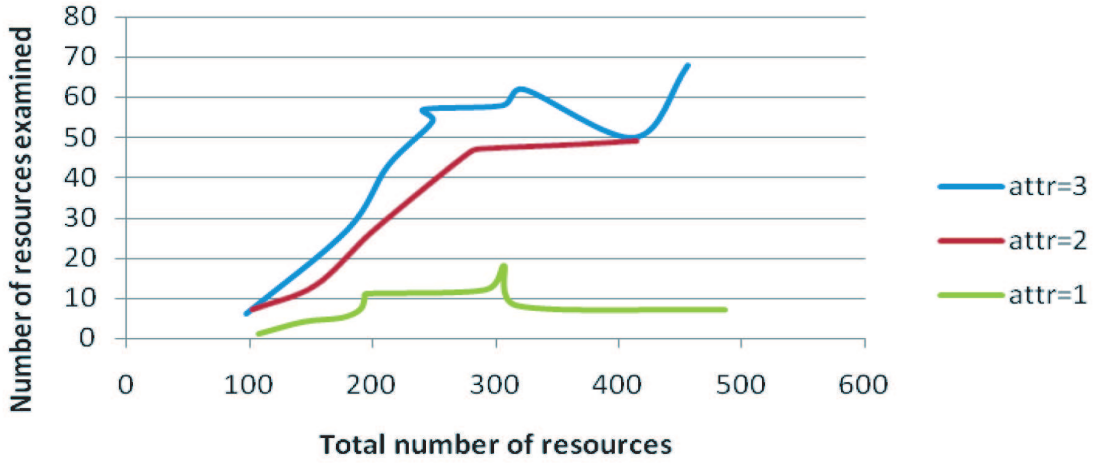


Fig. 6.6: Effect of number of attributes

this linearity. This is due to the fact that there are instances where a resource examined may satisfy more than one attribute at the first instance itself without calling for further resources to be examined.

6.3. Position of the Resource in a Heap. In the attrib-heap where the nodes are arranged on one parameter, say effective processing speed, query resolution time is saved by rejecting all nodes with processing speed less than the required speed. After these close-to-root processors are rejected, we are forced to check every node until success or we exhaust all nodes and declare failure. This exhaustive search has a time bound of N, the number of nodes with the same sum. Therefore, the time required is log (N) at the sum-heap and (N) at the attrib-heap. Fig. 6.7 shows a comparison of query resolution time with resources available in the different levels of the sum-heap. Values are plotted by varying number of resources as 10, 20 and 30. We find that number of comparisons required increases with the position of resource in the sum-heap. That is, if the resource is in the higher levels close to the root node in sum-heap, number of comparisons is less and vice versa.

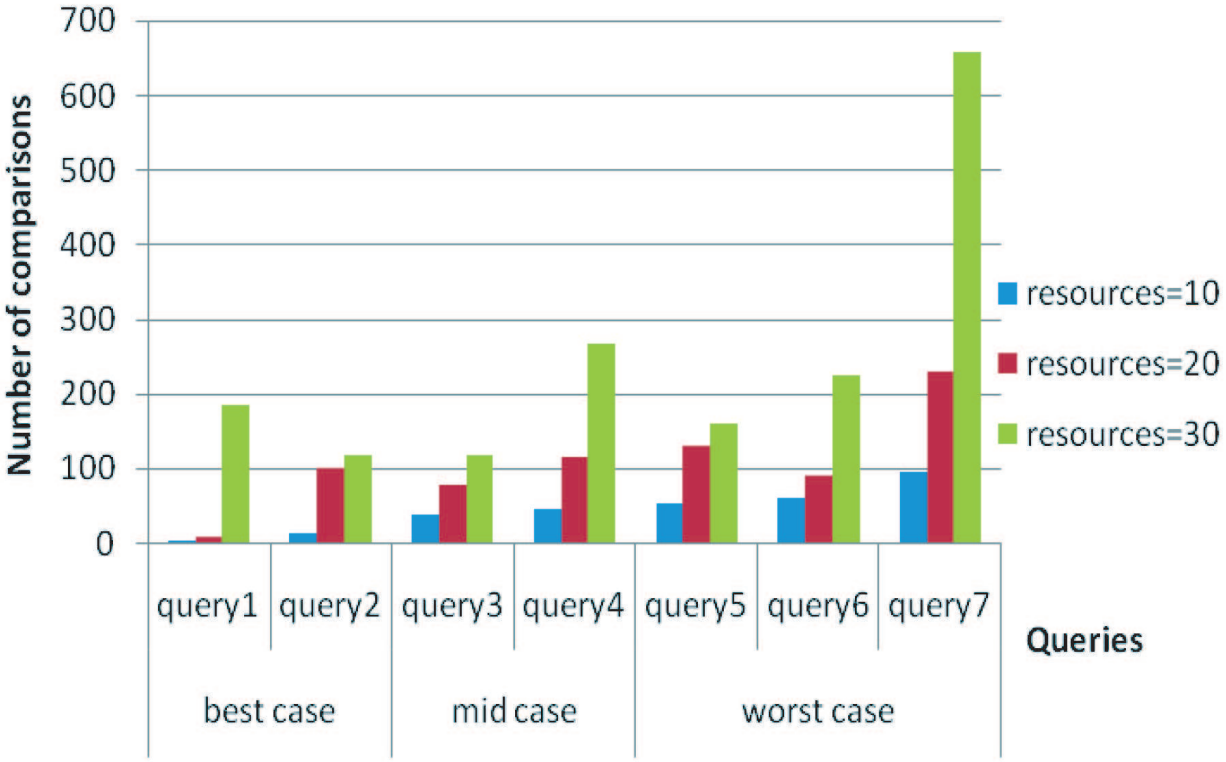


Fig. 6.7: Effect of number of attributes

6.4. Dynamism of nodes. To speed up the process, every node retains the attributes used earlier when the attributes change calling for a new arrangement. Let the attrib-heap be built based on processing speed of nodes. For example, if a node had effective free processing speed of 5.0 GHz and it becomes 5.5 GHz as it finished one task, it uses the earlier attributes to locate the current nodes in both the sum-heap and the attrib-heap. Then we find the new position using the new attributes. Now the links are adjusted. Link adjustments consume constant time in a heap. The search complexity in sum-heap is log (N) and that of attrib-heap is (N). Since we keep the nodes in each attrib-heap small, it could be treated as a constant. Thus the time complexity is log (N).

As regards a topology such as a ring, it requires O (N) for insertion, detection, and deletion. The insertion and deletion are constant time once the link to be broken is identified. However to identify the link, on an average N/2 inspections are required. For a heap the order is O (log (N)) and is better for large N. For very small N, the complexity of the procedure causes more time for heap compared with ring. This advantage of the ring when N is small is normally ignored.

6.5. Frequency of attribute changes. The analysis given above holds good for attribute changes also. Hence the order of complexity again is log (N).

7. Conclusion. There is an interesting property in our approach to resource discovery. In the sum-heap we select all nodes satisfying Sum_q. On matching attributes of these selected nodes with individual attributes in Sum_q, we may find that at least one attribute may not match the query requirement. So we may fail to find a match. So we move to the node in sum-heap, with sum more than the sum of attributes. The difference is called slack. If we need x bandwidth, in the sub tree we check only mobile nodes with bandwidth values x, x+1 ...x+slack. If we take a mobile node with (x + slack + 1) or more bandwidth, at least one other attribute may

be smaller. DEEPG can be modified to check for this condition and thus we need not search the sub tree until we reach the leaves. We may reach some leaves and very rarely all the leaves of the sub tree. Undoubtedly this will improve the efficiency of the protocol.

In this paper, DEEPG has been proposed to locate resources with exact and approximate attribute value (when no resources satisfy the requirement exactly). DEEPG can be modified to locate all resources with attribute values larger (or smaller) than a certain number depending on the QoS demands of the grid user. Resource discovery systems built on Chord [24] for N number of nodes, incur a worst-case time complexity of O (N) to locate a node and also O ((log N) 2) to rearrange to accommodate joins and leaves. In contrast, the proposed heap-based overlay has a time complexity of O (log N). Further, when peers in DHTs such as Chord [24], CAN [47], Pastry [5] and Tapestry [8] maintain an index of O (log N) peers, the proposed heap is simpler in that it has a maximum of only three neighbors including its parent and possibly two children. Another factor that makes DEEPG superior is that there is no need for load balancing as done in other DHTs using consistent hashing or any other explicit mechanisms. This is because, we do not create a distributed index like other DHTs, but we simply organize the overlay based on either the sum of attributes or one of the attributes. DEEPG can be extended to handle range queries also. Further, it requires only a small modification to consider static attributes for query resolution.

Acknowledgments. Special thanks to Prof. M.Ibramsha for many substantive discussions on this material and comments on the text.

REFERENCES

[1] J. Albrecht, D. Oppenheimer, A. Vahdat, and D. A. Patterson, *Design and implementation tradeoffs for wide-area resource discovery*, ACM Transactions on Internet Technology, 8 (2008).

[2] S. Androutsellis-Theotokis and D. Spinellis, *A survey of peer-to-peer content distribution technologies*, ACM Computing Surveys, 36 (2004).

[3] A. Andrzejak and Z. Xu, *Scalable, efficient range queries for grid information services*, in Proceedings of the Second IEEE International Conference on Peer to- Peer Computing (P2P 02), 2002, p. 33.

[4] A.R.Bharambe, M.Agarwal, and S.Seshan, *Mercury: supporting scalable multi-attribute range queries*, in In SIGCOMM04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, 2004, pp. 353–366.

[5] A.Rowstron and P.Druschel, *Pastry: Scalable, decentralized object location and routing for large-scale peer-to peer systems*, in Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms, 2001, pp. 329–359.

[6] R. Bhagwan, P. Mahadevan, V. G, and G. M. Voelker, *Cone: A distributed heap-based approach to resource selection*, Tech. Report CS2004-0784, UCSD, 2004.

[7] A. B.R.Badrinath and T.Imielinski, *Impact of mobility on distributed computations*, ACM SIGOPS Operating Systems Review, 27 (1993), pp. 15–20.

[8] B.Y.Zhao, J.D.Kubiatowicz, and A.D.Joseph, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*, Tech. Report UCB/CSD-01-1141, UC Berkeley, Apr 2001.

[9] M. Cai, M. Frank, J. Chen, and P. Szekely, *Maan: A multi-attribute addressable network for grid information services*, J. of Grid Computing, (2004).

[10] S. Chen, B. Shi, S. Chen, and Y. Xia, *Acom: Any-source capacity-constrained overlay multicast in non-dht p2p networks*, IEEE Transactions on Parallel and Distributed Systems, 18 (2007).

[11] D. Chu and M. Humphrey, *Mobile ogsi.net: Grid computing on mobile devices*, in Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (Grid2004), Nov 2004.

[12] D. Cokuslu, A. Hameurlain, and K. Erciyes, *Grid resource discovery based on centralized and hierarchical architectures*, International Journal for Infonomics, 3 (2010).

[13] D.Spence and T.Harris, *Xenosearch: Distributed resource discovery in the xenoserver open platform*, in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03), 2003, p. 216.

[14] A. A. Fatos Xhafa., *Computational models and heuristic methods for grid scheduling problems*, Future Generation Computer Systems, 26 (2010), pp. 608–621.

[15] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, International Journal of Supercomputer Applications, 11 (1997), pp. 115–128.

[16] I. Foster and C. Kesselman, *The grid: Blueprint for a new computing infrastructure*, Morgan Kaufmann Publishers, 1998.

[17] GonzaLez-BeltraN, P.Milligan, and P.Sage, *Range queries over skip tree graphs*, Computer Communications, 31 (2008), pp. 358–374.

[18] M. Hentschel, M. Li, M. Ponraj, and M. Qi, *Distributed indexing for resource discovery in p2p networks*, in 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[19] H.Shen, A.Apon, and C.Xu, *Lorm: Supporting low-overhead p2p-based range-query and multi- attribute resource management in grids*, in Proceedings of ICPADS, 2007.

[20] H.V.Jagadish, B.C.Ooi, K.L.Tan, Q.H.Vu, and R.Zhang, *Speeding up search in peer-to-peer networks with a multi-way tree structure*, in Proceedings of SIGMOD2006, 2006, pp. 1–12.

[21] H.V.Jagadish, B.C.Ooi, and Q.H.Vu, *Baton: A balanced tree structure for peer-to-peer networks*, in Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), 2005, pp. 661–672.

[22] A. Iamnitchi, I. Foster, and D. C. Nurmi, *A peer-to-peer approach to resource discovery in grid environments*, in Proceedings of the 11th Symposium on High Performance Distributed Computing, 2002, p. 419.

[23] S. Isaiadis and V. Getov, *Integrating mobile devices into the grid: Design considerations and evaluation*, in Proceedings of the International Euro-Par Conference (Euro-Par 2005), 2005.

[24] I.Stoica, R.Morris, D.Karger, M.F.Kaashoek, and H.Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, in SIGCOMM 01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, 2001, pp. 149–160.

[25] J.Gao and P.Steenkiste, *An adaptive protocol for efficient support of range queries in dht-based systems*, in Proceedings of the 12th IEEE International Conference on Network Protocols, 2004, pp. 239–250.

[26] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, , and D. Lewin, *Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web*, in Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997, pp. 654–663.

[27] S. Kurkovsky, Bhagyavati, A. Ray, and M. Yang, *Modeling a grid-based problem-solving environment for mobile devices*, in Proceedings of the IEEE International Conference on Information Technology: Coding and Computing (ITCC-04), 2004.

[28] L.Gong, *Jxta: a network programming environment*, IEEE Internet Computing, 5 (2001), pp. 88–95.

[29] F. Li, D. Qi, L. Zhang, X. Zhang, and Z. Zhang, *Research on novel dynamic resource management and job scheduling in grid computing*, in Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2006), 2006.

[30] A. Litke, D. Skoutas, and T. Varvarigou, *Mobile grid computing: Changes and challenges of resource management in a mobile grid environment*, in Proceedings of the 5th International Conference on Practical Aspects of Knowledge Management (PAKM 2004), Dec 2004.

[31] Q. Lv, P. Cao, and E. Cohen, *Search and replication in unstructured peer to peer networks*, in Proceedings of the 16th international conference on Supercomputing (ICS 02), June 2002, pp. 84–95.

[32] D. Millard, A. Woukeu, F. Tao, and H. Davis, *Experiences with writing grid clients for mobile devices*, in Proceedings of the 1st International ELeGI Conference, 2005.

[33] M.Litzkow, M.Livny, and M. Mutka, *Condor a hunter of idle workstations*, in Proceedings of the 8th Int. Conf. on Distributed Computing Systems (ICDCS 88), June 1988, pp. 104–111.

[34] M. Mohamed, *An object based paradigm for integration of mobile hosts into grid*, International Journal of Next-Generation Computing, 2 (2011).

[35] N.J.A.Harvey, M.B.Jones, S.Saroiu, M.Theimer, and A.Wolman, *Skipnet: A scalable overlay network with practical locality properties*, in Proceedings of Fourth USENIX Symposium on Internet Technologies and Systems (USITS03), 2003, pp. 113–126.

[36] A. Padmanabhan, S. Ghosh, and S. Wang, *A self-organized grouping (sog) framework for efficient grid resource discovery*, Journal of Grid Computing, 8 (2010), pp. 365–389.

[37] S.-M. Park, Y.-B. Ko, and J.-H. Kim., *Disconnected operation service in mobile grid computing*, in Proceedings of the International Conference on Service Oriented Computing (ICSOC 2003), 2003.

[38] P.Ghosh, N.Roy, and S.K.Das, *Mobility-aware efficient job scheduling in mobile grids*, in First IEEE International Workshop on Context-Awareness and Mobility in Grid Computing (held in conjunction with CCGrid 2007), 2007, pp. 701–706.

[39] R. Ranjan, A. Harwood, and R. Buyya, *Peer-to-peer-based resource discovery in global grids: a tutorial*, IEEE Communications Surveys & Tutorials, 10 (2008), pp. 6–33.

[40] R.Buyya. and M.Murshed, *Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing*, Concurrency and Computation: Practice and Experience, 14 (2002), pp. 1175–1220.

[41] S.Basu, S.Banerjee, P.Sharma, and S.Lee, *Nodewiz: peer-to-peer resource discovery for grids*, in Proceedings of the Fifth IEEE international Symposium on Cluster Computing and the Grid (CCGrid'05), 2005.

[42] S.Cheema, M.Muhammad, and I.Gupta, *Peer-to-peer discovery of computational resources for grid applications*, in Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid05), 2005, pp. 179–185.

[43] C. Schmidt and M. Parashar, *Flexible information discovery in decentralized distributed systems*, in Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC03), 2003.

[44] A. S.Grimshaw and W. A.Wulf, *The legion vision of a worldwide computer*, Communications of the ACM, 40 (1997), pp. 39–45.

[45] H. Shen and Z. Li, *Spps: A scalable p2p-based proximity-aware multi-resource discovery scheme for grids*, in Proceedings of IEEE Military Communications Conference (MILCOM 08), 2008, pp. 1–7.

[46] S.Hotovy, *Workload evolution on the cornell theory center ibm sp2*, in Job Scheduling Strategies for Parallel Proc. Workshop (IPPS 96), 1996, pp. 27–40.

[47] S.Ratnasamy, P.Francis, M.Handley, R.Karp, and S.Schenker, *A scalable content-addressable network*, in In SIGCOMM01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, 2001, pp. 161–172.

[48] S.Rhea, D.Geels, T.Roscoe, and J.Kubiatowicz, *Handling churn in a dht*, Tech. Report UCB//CSD-03-1299, UC Berkeley, Dec 2003.

[49] H. Sun, J. Huai, Y. Liu, and R. Buyya, *Rct: A distributed tree for supporting efficient range and multi-attribute queries in grid computing*, Future Generation Computer Systems, 24 (2008), pp. 631–643.

[50] M.-T. Sun, C.-T. King, W.-H. Sun, and C.-P. Chang, *Attribute-based overlay network for non-dht structured peer-to-peer lookup*, in International Conference on Parallel Processing (ICPP 2007), 2007.

[51] T.Asano, D.Ranjan, T.Roos, E.Welzl, and P.Widmayer, *Space-filling curves and their use in the design of geometric data structures*, Theoretical Computer Science, 181 (1997), pp. 3–15.

[52] T.Phan, L.Huang, and C.Dulan, *Challenge: Integrating mobile wireless devices into the computational grid*, in Proceedings of the 8th ACM Int. Conf. on Mobile Computing and Networking, (MobiCom 02), 2002.
[53] Z. Xu, R. Min, and Y. Hu, *Reducing maintenance overhead in dht based peer-to-peer algorithms*, in Proceedings of the Third International Conference on Peer-to-Peer Computing (P2P03), 2003.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 257–264. http://www.scpe.org
ISSN 1895-1767

GREEN DESKTOP-GRIDS: SCIENTIFIC IMPACT, CARBON FOOTPRINT, POWER USAGE EFFICIENCY

BERNHARD SCHOTT *AND AD EMMEN †

Abstract. Desktop Grids take their place in the e-Science distributed computing infrastructure - scaling into the millions of PCs. Desktop-Grids collect CPU cycles from PCs contributed by donors, by volunteers who are willing to support science and research. The Green key advantage of Desktop-Grids over service Grids and data centers based on clusters of servers is the minimal heat density. Compute Clusters without energy intensive aircondition would run into thermal disaster within minutes. PCs participating in Desktop-Grids usually do not make use of any aircondition. In return and with raising energy prices, data centers have implemented lower costs air-conditioning means like e.g. free cooling, improving their thermodynamic-efficiency and PUE rating. This paper, based on [24], investigates whether Desktop-Grids still have a Green advantage over Service-Grids and describe several distinct Green Methodologies to optimize compute unit specific energy consumption. Green-IT metrics as Carbon Footprint and PUE are analyzed for their relevance and applied to Desktop-Grids. Pragmatic implementation steps to Green-Desktop-Grids are described.

1. Desktop-Grid: Scientific Impact of large scale DCIs. During the past years, volunteer Desktop Grids have become a regular part of the computational infrastructure for e-Science. Although they already form an impressive computational power, the EDGeS infrastructure for instance connects about 150.000 computers in Desktop Grids to the European Grid Infrastructure (EGI), this is only the beginning, as there are hundreds of millions of computers, alone in Europe that could be connected. Desktop Grids take their place in the e-Science distributed computing infrastructure - scaling into the millions of PCs [23].

Compute time harvested from resource owners, typically individuals at home but also institutions and companies, does not request large upfront investment by the scientist; it is a low cost approach towards significant scientific output. Typically implemented using BOINC [17], sometimes XtremWeb [19], OUR-Grid [49], or other packages, Desktop-Grids are found among the largest Distributed Compute Infrastructures (DCI) [1]. Also known as Volunteer-Computing, Desktop-Grids have been around since the very early days of Grid computing [18]. An outstanding illustration of Desktop-Grid scientific impact was delivered these days by Einstein@Home. The new binary radio pulsar J1952+2630 was detected in data recorded at the Arecibo telescope back in 2005 [35].

The aggregations of so many machines result in significant performance well beyond Petaflop/s for selected applications. For example: BOINC network averages about 5.1 Petaflop/s as of April 21, 2010 [2]. Key difference to service Grids like EGEE (now EGI) is the voluntary character of the resources citizens contribute their PCs compute time to the Desktop-Grid projects in order to support scientific challenges of their choice. The FP7 project DEGISCO [20] supports Desktop-Grid deployments in and beyond Europe, especially countries that strongly collaborate with the European Union. DEGISCO recently published a first version of the Desktop Grids for eScience Road Map [21, 22], a guide to prepare and implement successful Desktop-Grid roll-outs, second release scheduled for June 2011. DEGISCO is accompanied by the EDGI project that continues to maintain and further develop the 3G-Bridge [3], a gateway transparently connecting gLite, Unicore, and KnowArc based infrastructures (Service-Grids) to Desktop-Grids by automated translation of the job-languages.

Sustained development and success of this combination of Desktop-Grids and Service-Grids has been proven at the EGI User Forum 2011 in Vilnius, Lithuania, by command-line submission of 10,000 jobs to a Desktop Grid from and through gLite, one of the main EGI Grid stacks [26].

Fascinating progress in material science using Desktop-Grids was reported by O. Gatsenko et al at the CGW2010 [27, 28, 29, 30], proving even complex problems to be solvable on a distributed Desktop-Grid platform.

2. The need for Green Desktop-Grids. One core topic of DEGISCO is the energy efficient handling of Desktop-Grid workload and management of resources, provided as configuration advice to Desktop-Grid operators. The need for Green-Desktop-Grids derives from sheer size: Desktop-Grids can aggregate hundred-thousands of machines per project. Power consumption of such large amounts of devices should be considered when making use of them. Indeed, when used for computation, energy consumption of PCs (like of any other computer) goes up [5,31]. The contributor, the volunteer, who allows and enables the use of her or his machine,

*AlmereGrid and VCOdyne SAS, Le Chesnay, France, E-mail: bernhard.schott@vcodyne.com
†AlmereGrid, Almere, Netherlands, E-mail: ad@almeregrid.nl

not only provides compute time for free but also pays for the additional electrical energy to cover the computation induced power consumption during a potentially increased uptime.

Key advantage of Desktop-Grids is the minimal power density compared to conventional data centres. Typically, PCs participating in Desktop-Grids in Europe are not hosted in air-conditioned environments. Without the energy burden of air-conditions, Desktop-Grid are intrinsically greener than data centre based clusters and thereof built Service-Grids. Data centers have been improving their power efficiency in the recent years, so we want to investigate how Green they are compared to Green-Desktop-Grids.

3. PUE a Green IT metric. PUE Power Usage Efficiency has been broadly accepted as metric to determine how green data centers are [32]. PUE is defined as "Total Facility Energy divided by the IT Equipment Energy". The best possible value is 1,0 = all energy provided is used in the IT equipment. PUE values are to be measured over a full year timespan to have winter temperatures with lower cooling efforts balance summer heat. PUE typically ranges between 1.5–2.5.

The Code of Conduct on Datacenters [4] quotes that most (older) European data centres are actually worse: they consume more than 200% of the IT related energy (PUE 3) for cooling, UPS and power distribution losses.

Recently the Facebook Open Compute Project [33] published PUE 1.07 which was criticized to be a snapshot value, calculated over a limited period during perfect conditions [34]. In order to try a comparison between Desktop-Grid and Service-Grid Green level, we transfer the PUE concept[32] from data center boundary to a system boundary including Desktop-Grid servers and Desktop-Grid-Clients. To clarify this fact, we call it the System-PUE. In our example we take 150000 PCs with partial-PUE (PPUE) of 1.0 + 15 BOINC servers with a PPUE of 1.8 deliver a System-PUE of 1.00020 (Fig. 3.1).

Desktop-Grid nodes			IT-Energy		System-Energy
	# nodes	[W]/node	Sums [kW]	PPUE	Total [kW]
Clients	150.000	150	22.500	1,0	22.500
Servers	15	380	6	1,8	10
		Sum	22.506	Sum	22.510
				PUE	**1,00020**

Fig. 3.1: Calculating a Desktop-Grid "System"-PUE

Transferring the PUE metric from data center to Desktop-Grid seems adequate for the similarities in thermodynamic definitions of systems. PUE for data centers is defined with regards to the data center boundaries. Still this is not a thermodynamic ensemble, not a closed system. The key advances in data center efficiency are based on introducing free-cooling [36] and direct-air-cooling, both implying open system thermodynamics as cold water or air is opportunistically fed into the data center - which is quite similar to the fundamental air-condition method of Desktop-Grid-Clients: "open the window". Intel has exercised the open window approach successfully in a production data center even to the point where servers became dust covered, exposed to varying humidity. The whitepaper [37] reports results like saving 2,8 million USD in power consumption annually with only minimal increase in server failure rate. Another example [38] keeps the air quality to ETSI standards but cools directly with air also.

3.1. Limited applicability of PUE. The resulting amazing PUE values derived from open window and similar approaches are questionable. IT equipment is stressed to the full extend of safe operations temperature range. Pointed out in the chapter Difficulties of temperature measurements below, built in fans are temperature controlled now running at maximum speed, causing raised energy consumption. By definitions of PUE, this additional cooling consumption does not worsen but improves the PUE value a clear mistake.

3.2. More Green Desktop-Grids needed. The need for compute power, the progress of Computer Aided Science (CAS), is massive and unstoppable. Researchers from all sectors are urgently looking for more

compute power. By June 2010 and well ahead of operations start, PRACE resources were already oversubscribed by a factor of 5 [6]. Although Desktop-Grid suitable workload is not HPC and only the lean-data-fraction of all HTC, significant scientific output has been, is, and will be produced with their help [7]. With the growing importance of Desktop-Grids in the scientific process and the significantly growing deployments, the need to optimize the use of energy is obvious both for general environmental considerations as well as for the attraction of contributors. Citizens providing their machines are interested in finding their contribution used in the most optimal way, producing more science and less waste-energy.

4. Questioning the Aims of Green IT: CO2 Footprint and the Energy mix. The original aim and claim of Green IT was formulated as "Reduction of IT activities CO2 footprint". As extended scope reduction of energy consumption in general and especially thermal emissions in metropolitan areas, both impacting a) global climate b) local (micro) climate and c) human quality of life. Production of CO2 and accordingly the reduction of CO2 footprint are difficult to measure from the perspective of a concrete IT activity like computation.

Even if energy consumption as such is accounted for, it depends on the local energy mix how much CO2 this is equivalent to.

Zero CO_2 electricity sources by 2007					
Energy Mix	Total	**Nuclear**		**Renewable**	
Electricity	[TWh]	[TWh]	%	[TWh]	%
Germany	607,0	167,1	**27,50%**	58,2	9,60%
France	572,2	448,2	**78,30%**	66,0	11,50%
Denmark	40,5	0	0,00%	10,2	25,20%

Fig. 4.1: EC Data sheet: Zero CO2 electricity sources by 2007

The electrical energy mix, the combination of electrical energy sources, depends on national specifics. The electrical energy mix (Fig. 4.1) in Germany [9] includes a nuclear energy portion of 27.5%, scheduled to be phased out, while the French [10] one (78.3%) is stable on a higher level. Denmark [11] produces 25% of its electricity consumption from wind sometimes up to 150% (when strong winds produce more electricity that the Denmark needs) causing negative energy prices at the spot market [12].

In Desktop-Grids, main energy consumption takes place at the client, compare Fig. 3.1. Accordingly the individual PC owner is in control which energy source, which energy tariff is primarily used. The Desktop-Grid server operator respectively the BOINC project manager can target to recruit donors from specific regions with their specific regional energy mix.

Typical for concurrent Green-IT discussion, it avoids mentioning nuclear power. For example the whitepaper on data center metric "Carbon Usage Effectiveness" (CUE, TheGreenGrid) [42]) states: "...the electricity may have been generated from varying CO2-intensive plants. Coal or gas generate more CO2 than hydro or wind. " but skips nuclear.

As pointed out in the following chapter on Green Desktop-Grid Methodologies it is possible to direct workload towards a region by recruiting volunteers from those specific geographies. In order to reduce Carbon Footprint, the Desktop-Grid-Project could recruit donors from France for example, automatically assuring 90% CO2 free computing (=80% nuclear + 10% renewables.), a CO2 value much lower than for example for Denmark not owning any nuclear power plant.

With a more general approach towards environmental protection - latest under the impression of Fukushima - it is difficult to prefer nuclear power plants with their intrinsic security threads and unresolved radioactive waste issues over natural gas powered electricity generation, although emitting carbon dioxide. Nevertheless it is beyond the scope of this paper to discuss alternative energy production strategies as we are focusing on alternative energy uses.

Desktop-Grid operators and donors together can implement environmental friendly policies:

- Energy tariff choice. Green energy tariffs excluding nuclear power are generally available for private households, companies, and institutions and meanwhile price-wise acceptable as examples from UK and Germany indicate [39, 40, 41];
- Reduce energy consumption in general.

As CO2 footprint alone fails to reflect energy production reality, it seems adequate rephrasing the core aim of Green IT from "Reduce CO2 footprint" to "Save energy".

4.1. €- metrics for Green IT success. In order to measure the effectiveness of energy saving policies and methods, we need to introduce a metric that can be "metered". The obvious advantage of "kWh" as the base metric for Green IT is the simplicity of measurement: electricity is metered everywhere. Different from data centres and conventional Service-Grids, policies and methods are applied and executed in Desktop-Grids mainly by the volunteer effort of the resource contributor. As success metric for Green IT, the translation into cost, into money, is helpful to connect to business considerations and propel motivation. With €(for kWh) as metric, contributors can relate their choice of workload and policy-compliance to the personal electricity bill. Green Desktop-Grids help the planet and your budget may express this motivation appropriately.

5. DEGISCO Green Desktop-Grid Methodologies. DEGISCO investigates conceptually different methodologies, based on technology means like Desktop-Grid-Client based ambient metrics, exploitation of natural ambient conditions, and more. Some of those methodologies are technological, some are purely organizational. DEGISCO promotes innovative Desktop-Grid deployments through the International Desktop-Grid-Federation [8] especially by the Desktop Grids for eScience Road Map. The Desktop-Grid-Federation will offer consulting and advice based on Green methodologies, continuing the roadmap process to reflect and integrate future findings and developments. One focus topic in the roadmap process is the application of Green Methodologies to achieve reduction in energy consumption of research infrastructures.

5.1. Seven Green Desktop-Grid Methodologies. DEGISCO has started with a shortlist of 7 methodologies which are a collection of best practices, techniques and policies:

- Ambient metrics based Green optimization;
- Cool strategy: avoid air-condition use;
- Energy profiling of applications;
- CPU speed steps;
- Exploitation of natural ambient conditions;
- Time-of-day dependent energy tariffs;
- Management of unused resources in a local Desktop-Grid.

In the course of the roadmap process [21, 22] these methodologies are challenged, refined or replaced, according to feedback and feasibility tests supported by contributors.

5.1.1. Ambient metrics based Green optimization. In order to tune DEGISCO connected Desktop-Grids towards saving of energy suitable configurations and parameters are to be identified enabling the Desktop-Grid client to intelligently select adequate workload. A regular PC [13] almost doubles its power consumption from idle 160W to 300W under full CPU load. BOINC general preferences [45] allow to specify that computation consumes a certain portion of the machine, e.g. 50% of CPU time, by this reduces heat dissipation.

Ambient temperature measurement or at least estimation could be used to control and potentially prevent download of workload items if the PC and its environment are too hot for comfortable or safe operations. The measurement and observance of ambient conditions, mainly temperature, is essential for several advanced Green Methodologies, too.

5.1.2. Cool strategy: avoid air-condition use. Desktop-Grids are the real Green Grids: lower energy density than clusters results in less energy wasted for cooling. However, this may not longer be true if air-conditions are used to assure proper operation of Desktops. According to the principles of thermodynamics, the energy consumption by air-conditions for cooling range from 30% to >200% (PUE: 1.3..>3) of the energy dissipated by the IT device. The wide range is a direct result of the cool-reservoir temperature the heat pump can utilize to get rid of the heat. The prime advice to configure Green-Desktop-Grids: avoid air-condition use.

Selection criteria for the "maximum temperature" as described above could be that temperature which would just not yet trigger the start of the local air-condition. In case the use of air-condition is unavoidable, the recommendation to participate in Desktop-Grids depends:

- if the additional workload by Desktop-Grids would cause proportional air-condition power consumption, a change of strategy should be considered. Maybe by restricting the acceptance of workload to night times would help, configurable in the BOINC client settings [44] and preferences [45];
- if the air-condition is in full power use anyway – like in tropical ambient – the additional heat dissipation during compute load processing may not impact the total energy balance too much. Still the additional heat dissipation can be controlled as described above [45].

Example: light building structures with poor thermal insulation and continuously running air-conditions are de-facto standard in sub-tropical and tropical regions globally. If we assume a 3.5kW air-condition (2-3 room flat, small house) to run non-stop in order to keep the ambient temperature 15°C below the 40°C outside, additional heat dissipation of a standard office PC (60W idle, 120Watts fully loaded) would raise the ambient temperature by ≈1°C (assumed 50% efficiency of the air-con). The 120 Watts compare to the 100W approximate basal metabolic rate [47] + 20-40W brain activity of the human body so the user of the PC will raise the ambient temperature for another 1°C while awake and thinking. The raise in room- or ambient-temperature is minimal since the thermal balance in this example is dominated by the heat flow through the building structure. Massively higher impact on the room-temperature is caused by cooking activities (in the multi-kW range).

5.1.3. Energy profiling of applications. Different applications and codes consume more or less CPU at any given time, resulting in different energy consumption per time interval, specific energy profiles. They behave differently in raising machine and ambient temperature. According to our findings within the DEGISCO available pool of applications (see Desktop Grid Application Super-Repository: [14]), these are could be classified accordingly with a heat index as +, ++ and +++ for example. We refrain from using "green", "orange", and "red" at this point: The +++ index marks an application that makes maximum use of a given machine, is raising its temperature, but finishes the computation quickly. This behaviour may total in less energy consumed/computation than the application which creates less heat/time but runs longer. Still heat/time is an important parameter from a green operations point of view. As PC owners can select the project and by this the application they want to contribute to, they can take into account their specific knowledge of local operations conditions, primarily how much additional heat they can accept. If energy profiling of applications does not supply sufficient control range, the BOINC native method of setting general preferences [45] to specify CPU% utilization may be used.

5.1.4. CPU Speed Steps. A similar effect could be achieved by exploiting processor speed steps, avoiding additional preparation work on the application side. Current processors provide multiple steps (8-16) for CPU speed, thus controlling energy consumption. Gruber and Keller discuss the use of "SpeedStep" among other methods in order to use the minimal CPU frequency to run an application at full memory bandwidth [15]. Different from the application, the OS and tools installed at the PC are under control of the Desktop-Grid contributor, placing the management of methods like "SpeedStep" into the volunteers hand.

5.1.5. Exploitation of natural ambient conditions. DEGISCO investigates another completely independent green strategy: exploitation of natural ambient conditions. A specific project advantage facilitates the aggregation of partners from various different geographies, a fact that allows benefiting from differences in regional weather situations in order to save energy. Workload indexed as "+++" may systematically be offered to contributors located in low temperature areas while those in sunny summer weather will be offered to contribute for "+" workload. Different locations yield different climates:

- Kazakhstan, Amaty: `http://worldweather.wmo.int/070/c00152.htm`
- Russia, Moscow: `http://worldweather.wmo.int/107/c00206.htm`
- Hungary, Budapest: `http://worldweather.wmo.int/017/c00060.htm`
- Denmark, Copenhagen: `http://worldweather.wmo.int/173/c00190.htm`
- Spain, Zaragoza: `http://worldweather.wmo.int/083/c01240.htm`

Recruiting regions with opposite weather conditions allows to counteract actual weather conditions. E.g. if there is only "+++"-workload-type available today, Copenhagen may be preferred over Zaragossa. DEGISCO partners from Kazakhstan, Russia, and Spain confirmed the weather conditions reported on "worldweather" as already averaged – the peak temperatures exceed both into heights and lows significantly.

5.1.6. Time-of-day or weather dependent energy tariffs. The value and price of electrical energy is changing according to the conditions of generation as well as by changing consumption. Accordingly, the tariffs

for electricity are changing over time of day and year. While in Germany electricity prices are high during lunch time, in Kazakhstan the energy prices go up in the evening – in both cases dependent on consumption.

Energy prices at the Spot markets vary depending on excess production capacity. Since wind energy can deliver significant amounts of energy, these spot market prices can even turn negative [12].

To improve the energy cost situation and to take advantage of excess Green electricity, advice could be given to contributors how to configure their Desktop-Grid-Clients to prefer workload during low tariff times [45, 46].

5.1.7. Management of unused resources in a local Desktop-Grid. A seventh Green strategy can be reported from the OUR-Grid project, presented e.g. at OGF30, Brussels: Lesandro Ponciano and Francisco Brasileiro have focussed on sleeping- and wake-up-strategies for Desktop-Grid-Clients from various modes like off, suspend, or hibernate, and the according impact on responsiveness of a campus Desktop-Grid [49]. Wake-up-strategies like WOL (Wake on LAN) are usually not applicable for Internet scale deployments but work well in local networks.

6. Desktop-Grid the loosely coupled virtual data centre. A major difference between a data centre situation and volunteer based Desktop-Grids is the almost complete lack of central control over the compute resources. Further, Desktop-Grid applications are executed as user with limited permissions (no root rights). Accordingly, installation of support tools, in our case for temperature and energy consumption measurement, is not possible without active voluntary contribution by the resource owner, installing tools with administrator (aka "root") rights. This cannot be done as regular Grid job: different from service Grids, Desktop-Grids implement highest security standards also on the execution side. Applications that are downloaded and executed on the contributed Desktop-Grid client are security validated and, dependent on the Desktop-Grid technology used, even rewritten to execute exactly that computation as described and nothing more. Any activity beyond the sandbox, e.g. accessing local HW devices like sensors, is off limits for Desktop-Grid jobs. When DEGISCO is looking to gather detailed temperature and energy consumption data we are asking for volunteers to download and install tools and allow the upload of resulting metric data.

6.1. Difficulties of temperature measurements. In order to apply the Green Methodologies described, it is very helpful to adequately understand the ambient conditions of the PC, especially the ambient temperature. It seems obvious to utilize the PC's built in temperature sensors – but there are difficulties to overcome. The temperature sensors included in PCs and laptops are optimized to support energy management of the PC and its components – not to provide ambient conditions, the kind of information we need. Mainly the position of the sensor determines what is measured. On-die temperature sensors may reflect the CPU internal temperature quite precisely while "system" temperature sensors are placed "somewhere" on the mainboard – delivering temperature measurements that cannot be interpreted meaningful without precise and detailed knowledge of the individual board. Although this seems doable in a lab situation, it is completely beyond scope for real world deployed Desktop-Grids. The situation is not very much better in regular data centres: depending on the placement of the temperature sensor in the rack, a hot spot will be detected or not. A detailed temperature measurement at several positions in the rack is still not commonly found. For safety reasons, the single temperature sensor is placed to detect the (known or anticipated) hot spot, caused by poor local airflow – delivering information misleading with regards to average, typical, or total (i.e. = full rack) energy consumption. Further complication is deriving from the application of temperature aware fan speed controls embedded in the systems. Originally developed for Desktops in order to keep their operations noise level convenient for living room conditions, meanwhile regular servers are controlling their fan speeds to provide exactly that amount of cooling needed to keep board temperature within the targeted operations range while enhancing the lifetime of the fans. The "Code of Conduct on Datacenters" [4] explicitly requests control of fan speeds also on the data centre level. The result for our aim to understand the ambient conditions of a machine by reading its temperature sensors gets complicated by these features.

Still the information retrieved may well be sufficient for our aims:

1. Understand values delivered by PC internal temperature sensors as non-linear non-calibrated relative information on machine cooling effectiveness;
2. For ambient temperature use meteorological data by independent sources;
3. To calibrate and QA the methods, call for participation by contributors in a temperature measurement campaign.

Even qualitative temperature information is suitable to distinguish condition "too hot for workload" from "cool and ready to work". To verify our understanding on ambient conditions, we started working Fraunhofer Institute ITWM, Kaiserslautern to reuse a simple and low cost temperature sensor [16] that can be connected to the desktop or laptop (USB) and delivers proper ambient metrics. This temperature sensor may be offered to Desktop-Grid volunteers by mail-order, requesting the commitment to provide temperature measurement data for automated upload.

Desktop-Grids have been used for sensor applications frequently, like the project "Quake": Quake-Catcher Network Seismic Monitoring [48]. The Quake project tried to use the built-in sensors primarily, but offers external sensors too.

7. Conclusion and outlook on International-Desktop-Grid-Federation. We need to progress on wise usage of donated compute time and the accompanied energy, otherwise future willingness of donors is questionable. Desktop-Grids are positioned well as the comparison with data centers show – but improvements are possible and looked for like improved energy aware scheduling interconnected with user friendly project specific energy aware client preferences. This will need to be implemented on the Desktop-Grid technology (client, server) in an easy to manage, easy to operate way, something that could be targeted by an upcoming research project. For immediate use and implementation, DEGISCO provides the roadmap document to assure Green-Desktop-Grid success.

In order to improve Desktop-Grid services for e-science and to sustain Desktop-Grids as regular DCI (Distributed Compute Infrastructure) the International-Desktop-Grid-Federation (IDGF, [14]) takes over from DEGISCO and EDGI. The IDGF is becoming the crystallization point for new projects and advances in Desktop-Grids and especially Green-Desktop-Grids.

Acknowledgements DEGISCO is supported by the FP7 Capacities Programme under grant agreement nr RI-261561.

REFERENCES

[1] Folding@Home, *Client statistics by OS*, [Online] http://www.boincstats.com/stats/project_graph.php?pr=bo
[2] BOINC, *Credit overview*, [Online] [Cited: 21 04 2010.] http://www.boincstats.com/stats/project_graph.php?pr=bo
[3] Farkas, P. Kacsuk, Z. Balaton, G. Gombas, *Interoperability of BOINC and EGEE*, Future Generation Computer Systems,Volume 26, Issue 8, Pages 1092-1103, October 2010, http://dx.doi.org/10.1016/j.future.2010.05.009
[4] Institute for Energy,, *European Commission Joint Research Centre. EU Code of Conduct for Data Centres*, http://re.jrc.ec.europa.eu/energyefficiency/html/standby_initiative_data_centers.htm. http://re.jrc.ec.europa.eu/energyefficiency/pdf/CoC%20DC%20new%20rep%20form%20and%20guidelines/Best %20Practices%20v2.0.0%20-%20Release.pdf
[5] Schott, Bernhard, *Energy optimization of existing datacenters*, OGF25 Catania, [Online] http://www.ogf.org/OGF25/materials/1654/Energy+Optimization+of+Existing+Datacenters+-+Bernhard+Schott+-+Platform.pdf
[6] Lippert, Thomas, *Contributions to HPC 2010 Cetraro*, High Performance Computing, GRIDS and clouds, June 21–25, 2010, Cetraro, Italy. [Online] 21-25 06 2010. http://www.hpcc.unical.it/hpc2010/ctrbs/lippert.pdf.
[7] Robert Lovas, Tamas Kiss, *Integrated service and desktop grids for scientific computing*, In: Conference proceedings of DCABES 2009. The 8th international symposium on distributed computing and applications to business, engineering and science. Wuhan, China, 16–19, October, 2009. DCABES, pp. 251-255. ISBN 9787121095955 http://dcabes.meeting.whut.edu.cn/DCABES2009/Files/DCABES%202009%20Proceedings.pdf
[8] Federation, *International Desktop Grid. International Desktop Grid Federation*, [Online] http://desktopgridfederation.org/
[9] Comission, European, GERMANY Energy Mix Fact Sheet. [Online] http://ec.europa.eu/energy/energy_policy/doc/factsheets/mix/mix_de_en.pdf
[10] Comission, European, FRANCE Energy Mix Fact Sheet. [Online] http://ec.europa.eu/energy/energy_policy/doc/factsheets/mix/mix_fr_en.pdf
[11] Comission, European, DENMARK Energy Mix Fact Sheet. [Online] http://ec.europa.eu/energy/energy_policy/doc/factsheets/mix/mix_dk_en.pdf
[12] Spot, Nord Pool, *Nord Pool Spot implements negative price floor in Elspot from October 2009.* [Online] http://www.nordpoolspot.com/Market_Information/Exchange-information/No162009-Nord-Pool-Spot-implements-negative-price-floor-in-Elspot-from-October-2009-/
[13] PCWelt, *CPU-Leistungsexplosion Intel Core i7 Prozessor. Stromverbrauch und Energieeffizienz*, [Online] http://www.pcwelt.de/start/computer/prozessor/tests/185273/intel_core_i7_prozessor/index3.html
[14] IDGF, *Applications available on the EDGI/DEGISCO infrastructures*, International Desktop Grid Federation [Online] http://desktopgridfederation.org/applications
[15] Gruber, Ralf and Keller, Vincent, *HPC Green IT*, Berlin Heidelberg: Springer-Verlag, 2010. p. 184ff. DOI 10.1007/978-3-642-01789-6_1
[16] Dalheimer, Mathias, *FHG ITWM. USBTemp: Continuous Temperature Monitoring*, [Online] http://gonium.net/md/2009/01/03/usbtemp-continuous-temperature-monitoring/

[17] BOINC, *Open-Source Software fr Volunteer Computing und Grid Computing*, [Online] http://boinc.berkeley.edu/
[18] GIMPS, *Great Internet Mersenne Prime Search*, [Online] http://mersenne.org/various/history.php
[19] XtremWeb, *XtremWeb: the Open Source Platform for Desktop Grids*, [Online] http://www.xtremweb.net/
[20] DEGISCO, *DEGISCO project website*, [Online] http://degisco.eu/introduction
[21] DEGISCO press release, *DEGISCO to release first version of Desktop Grids for eScience Road Map*, [Online] http://degisco.eu/press-release-20110204
[22] DEGISCO, *Desktop Grids for eScience Road Map*, [Online] http://desktopgridfederation.org/documents/10508/57919/RoadMapD.pdf?version=1.0
[23] Emmen, Ad, *Desktop Grids take their place in the e-Science distributed computing infrastructure*, Cracow Grid Workshop 2010 [Online] http://www.cyfronet.pl/cgw10/keynote-abs.html#n1
[24] Schott, Bernhard and Emmen, Ad, *Degisco Green Methodologies in Desktop Grids*, International Multiconference on Computer Science and Information Technology, [Online] http://proceedings2010.imcsit.org/pliks/191.pdf
[25] Schott, B. and Emmen, A., *Green Methodologies in Desktop-Grid*, [Online] http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5679621
[26] Versweyveld, Leslie, *EDGI submits 10,000 jobs to Desktop Grid at EGI User Forum*, [Online] http://degisco.eu/start/-/blogs/edgi-submits-10-000-jobs-to-desktop-grid-at-egi-user-forum
[27] Gatsenko, Oleksandr et al., *Statistical Properties of Deformed Single-Crystal Surface under Real-Time Video Monitoring and Processing in the Desktop Grid Distributed Computing Environment*, Key Engineering Materials, 2011 [Online] http://www.scientific.net/KEM.465.306
[28] O. Gatsenko, O. Baskova, and Yu.G. Gordienko, *Desktop Grid Computing in Materials Science Lab - Example of Development and Execution of Application for Defect Aggregation Simulations*, Proc. of Cracow Grid Workshop, Cracow, Poland, 2010, pages 264-272
[29] O. Gatsenko, O. Baskova, and Yu.G. Gordienko, *Scaling-up MATLAB Application in Desktop Grid for High-Performance Distributed Computing Example of Image and Video Processing*, Proc. of Cracow Grid Workshop, Cracow, Poland, 2010, pages 255-263
[30] O. Gatsenko, O. Baskova, and Yu.G. Gordienko, *Enabling High-Performance Distributed Computing to e-Science by Integration of 4th Generation Language Environments with Desktop Grid Architecture and Convergence with Global Computing Grid*, Proc. of Cracow Grid Workshop, Cracow, Poland, 2010
[31] BOINC, *Heat and energy considerations*, [Online] http://boinc.berkeley.edu/wiki/Heat_and_energy_considerations
[32] Azevedo Dan, Cooley Jud, Patterson Michael and Blackburn Mark, *Data Center Efficiency Metrics*, The Green Grid Technical Forum 2011 [Online] http://www.thegreengrid.org/ /media/TechForumPresentations2011/Data_Center_Efficiency_Metrics_2011.ashx?lang=en
[33] Higginbotham, Stacey, *Facebook Open Sources Its Servers and Data Centers*, GIGAOM [Online] http://gigaom.com/cloud/facebook-open-sources-its-servers-and-data-centers/
[34] Fehrenbacher, Katie, *How a Snapshot of a Green Data Center Can Be Misleading*, GIGAOM [Online] http://gigaom.com/cleantech/how-a-snapshot-of-a-green-data-center-can-be-misleading/
[35] Eistein@home, *Arecibo PALFA Survey and Einstein@Home: Binary Pulsar Discovery by Volunteer Computing*, [Online] http://einsteinathome.org/
[36] Wikipedia, *overview article on "Free cooling"*, [Online] http://en.wikipedia.org/wiki/Free_cooling
[37] Don Atwood, John G. Miner, *Reducing Data Center Cost with an Air Economizer*, [Online] http://www.intel.com/it/pdf/reducing_data_center_cost_with_an_air_economizer.pdf
[38] Erik Vandenmeersch and Johan Vanderhaegen, *Free Air Cooling Proof of Concept*, [Online] http://wikis.sun.com/display/freeaircooling/Free+Air+Cooling+Proof+of+Concept
[39] Greenpeace Energy, [Online] http://www.greenpeace-energy.de/index.html
[40] Naturenergieplus, [Online] http://www.naturenergieplus.de/index.php
[41] Green Electricity Marketplace, [Online] http://www.greenelectricity.org/index.php
[42] Belady Christian et al., *WP#32-Carbon Usage Effectiveness (CUE): A Green Grid Data Center Sustainability Metric*, TheGreenGrid [Online] http://www.thegreengrid.org/ /media/WhitePapers/CarbonUsageEffectivenessWhitePaper20101202.ashx?lang=en
[43] BOINC Project-Options, [Online] http://boinc.berkeley.edu/trac/wiki/ProjectOptions
[44] BOINC Client configuration, [Online] http://boinc.berkeley.edu/wiki/Client_configuration
[45] BOINC General preferences, [Online] http://boinc.berkeley.edu/wiki/Preferences
[46] BOINC Heat and energy considerations, [Online] http://boinc.berkeley.edu/wiki/Heat_and_energy_considerations
[47] Wikipedia, *Basal animal metabolic rate* [Online] http://en.wikipedia.org/wiki/Basal_metabolic_rate
[48] Quake, *Quake-Catcher Network Seismic Monitoring* [Online] http://qcn.stanford.edu/sensor/
[49] Ponciano Lesandro and Brasileiro Francisco, *New developments in DGs: Prof. Brasiliero: Our-Grid Power management On the Impact of Energy-saving Strategies in Opportunistic Grids* [Online] http://www.ens-lyon.fr/LIP/RESO/e2gc2_2010/slides/e2gc2_lponciano.pdf

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 265–274. http://www.scpe.org
ISSN 1895-1767

HYBRID PARALLEL PROGRAMMING FOR BLUE GENE/P

MADS R. B. KRISTENSEN, HANS H. HAPPE, AND BRIAN VINTER*

Abstract. The concept of massively parallel processors has been taken to the extreme with the introduction of the BlueGene architectures from IBM. With hundreds of thousands of processors in one machine the parallelism is extreme, but so are the techniques that must be applied to obtain performance with that many processors. In this work we present optimizations of a Grid-based projector-augmented wave method software, GPAW, for the Blue Gene/P architecture. The improvements are achieved by exploring the advantage of shared and distributed memory programming also known as hybrid programming and blocked communication to improve latency hiding. The work focuses on optimizing a very time consuming operation in GPAW, the stencil operation, and different hybrid programming approaches are evaluated. The work succeeds in demonstrating a hybrid programming model, which is clearly beneficial compared to the original flat programming model. In total an improvement of 1.94 compared to the original implementation is obtained. The results we demonstrate here are reasonably general and may be applied to other stencil codes.

Key words: GPAW, HPC, Hybrid-programming, Multicore platforms

1. Introduction. Grid Based Projector Augmented Wave (GPAW) [8] is a simulation software, which simulates many-body systems at the sub-atomic level. GPAW is primarily used by physicists and chemists to investigate the electronic structure, principally the ground state, of many-body systems. The GPAW users often have a desire to increase the system size and resolution to the point where the simulation time escalates to weeks and sometimes even months. A massively parallel implementation of GPAW, which is able to fully utilize a supercomputer, is therefore highly desirable.

The performance profile of GPAW dependence almost entirely on the electronic structures that are being simulated. Therefore, it is difficult to measure the general performance of GPAW. However, a significant part of any GPAW computation consists of a distributed stencil operation. Thus an optimization of this stencil operation will result in an improvement of the general performance of GPAW. The main object of this paper is to optimize the stencil operation for the Blue Gene/P [9] (BGP) architecture.

The current trend in HPC hardware is towards systems of shared-memory computation nodes. The BGP also follows this trend and consists of four CPU-cores per node. Furthermore, it is quite possible that future versions of the Blue Gene architecture will consists of even more CPU-cores per node.

To exploit the memory locality in shared-memory computation nodes a paradigm that combines shared and distributed memory programming may be of interest. The idea is to avoid communication between CPU-cores on the same node. Unfortunately, it is not trivial to obtain good performance when combining shared-memory programming with distributed memory programming. Even though inter-CPU communication is avoided, it is often the case that the sole use of MPI [5] outperforms a combination of threads and MPI when computing on clusters of shared-memory computation nodes [6, 7, 10].

We evaluate two different hybrid programming approaches. One approach in which inter-node communication is handled individually by every thread and another approach in which one thread handles the inter-node communication on behalf of all the other threads in a node. The work shows that, on the Blue Gene/P, the first approach is clearly superior the latter. In [3] the authors concludes that, on a well balanced system, a loop level parallelization approach, corresponding to our second hybrid approach, is unfavorably compared to a strict MPI implementation. Our first hybrid approach was developed on the basis of that conclusion.

2. GPAW. GPAW is a real-space grid implementation of the projector augmented wave method [2]. It uses uniform real-space grids and the finite-difference approximation for the density functional theory calculations.

A central part of density functional theory and a very time consuming task in GPAW, is to solve Poisson and Kohn-Sham equations. Both equations rely on stencil operations when solved by GPAW. When solving the Poisson equation, a stencil is applied to the electrostatic potential of the system. When solving the Kohn-Sham equation, a stencil is applied to all wave-functions in the system. Both the electron density and the wave-functions are represented by real-space grids. A system typically consists of one electron density and thousands of wave-functions. The number of wave-functions in a system depends on the number of valence electrons in the system. For every valence electron there may be up to two wave-functions.

*Niels Bohr Institute, Copenhagen, Denmark. {madsbk, happe, vinter}@nbi.dk

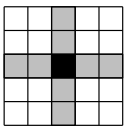


Fig. 2.1: A stencil operation on a 2D grid.

Table 3.1: Hardware description of a Blue Gene/P node

Node CPU	Four PowerPC 450 cores
CPU frequency	850 MHz
L1 cache (private)	64KB per core
L2 cache (private)	Seven stream prefetching
L3 cache (shared)	8MB
Main memory	2GB
Main memory bandwidth	13.6GB/s
Peak performance	13.6 Gflops/node
Torus bandwidth	$6 \times 2 \times 425$MB/s = 5.1GB/s

The computational magnitude of a GPAW simulation depends mainly on three factors: The world size, simulation system resolution and the number of valence electrons. The world size and resolution determine the dimensions of the real-space grids and the number of valence electrons determines the number of real-space grids.

A user is typically more interested in adding valence electrons to the simulation than to increase the size or resolution of the world. The real-space grid size will ordinary be in the interval 100^3 to 200^3 where as the total number of real-space grids will be greater than thousand.

2.1. Stencil Operation. A stencil operation updates a point in a grid based on the surrounding points. A typical 2D example is illustrated in Fig. 2.1 where points are updated based on the two nearest points in all four directions.

Stencil operations on the real-space grids (3D arrays) are used for the finite-difference approximation in GPAW. The stencil operation used is a linear combination of a point's two nearest neighbors in all six directions and itself. The stencil operations do normally use periodic boundary condition but that is not always the case.

If we look at the real-space grid A and a predefined list of constants C, a point $A_{x,y,z}$ is computed like this:

$$\begin{aligned} A'_{x,y,z} \quad = \quad & C_1 A_{x,y,z} + C_2 A_{x-1,y,z} + C_3 A_{x+1,y,z} + \\ & C_4 A_{x-2,y,z} + C_5 A_{x+2,y,z} + C_6 A_{x,y-1,z} + \\ & C_7 A_{x,y+1,z} + C_8 A_{x,y-2,z} + C_9 A_{x,y+2,z} + \\ & C_{10} A_{x,y,z-1} + C_{11} A_{x,y,z+1} + \\ & C_{12} A_{x,y,z-2} + C_{13} A_{x,y,z+2} \end{aligned}$$

3. Blue Gene/P. Blue Gene/P consists of a number of nodes interconnected with three independent networks: a 3D torus network, a collective tree structured network, and a global barrier network. All point-to-point communication goes through the torus network and every node is equipped with a direct memory access (DMA) engine to offload torus communication from the CPUs. The collective tree structured network is used for collective operation like the MPI *reduce* operation and the global barrier network is used for barriers.

Table 3.1 is a brief description of a BGP node. One thing to highlight is the ratio between the speed of the CPU-cores and the main memory. Since the CPU-cores are relatively slow and the main memory is relatively fast compared to today's standard, the performance of the main memory is not as far behind the CPU as usually. Furthermore, the torus bandwidth is only three times lower than the main memory bus when all six connections are used. The von Neumann bottleneck [1] associated with main memory and network is therefore reduced.

The CPU-cores can be utilized by normal SMP approaches like pthread or OpenMP, with the limitation that BGP only supports one thread per CPU-core. The BGP addresses the problem of utilizing multiple CPU-cores by supporting a virtual partition of the nodes. From the programmers point of view the four CPU-cores

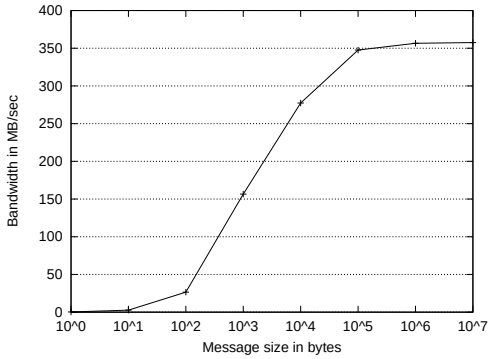


Fig. 3.1: A bandwidth graph showing how the message size influence the bandwidth. In this experiment, one MPI message is send between two neighboring BGP nodes.

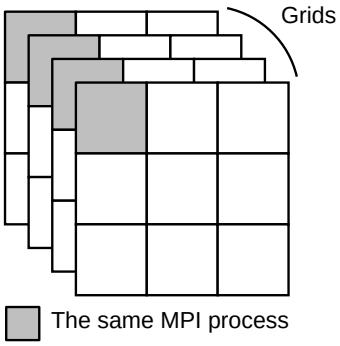


Fig. 4.1: Four 2D grids distributed over nine processes.

would then look like four individual nodes with each 512MB of main memory. This virtual partitioning is called virtual mode.

3.1. MPI. BGP implements the MPICH2 library, which comply with the MPI-2 specification [4]. MPI-2 specifies different levels of threaded communication. BGP supports the fully thread-safe mode called `MULTIPLE` that allows any thread to call the MPI library at any time. Since there is an overhead associated with `MULTIPLE` (e.g. locks), it is also possible to use the more restricted `SINGLE` mode that do not allow concurrent calls to MPI.

The MPICH2 implementation is tailored to utilize the BGP's DMA engine which means that non-blocking MPI communication is handled asynchronously with minimum CPU involvement.

BGP supports the `MPI_Cart_create` function, which tells BGP to reorder the MPI ranks in order to match the torus network. We make use of this function in all the following.

To investigate how much the message size influence point-to-point bandwidth, we have performed an experiment in which one MPI message is send between two neighboring BGP nodes (cf. Fig. 3.1). The result of the experiment clearly shows that in order to maximize the bandwidth, a message size greater than 10^5 bytes is needed, while half the asymptotic bandwidth is achieved at approximate 10^3 bytes.

4. The GPAW Implementation. GPAW is implemented using C and Python. The intention is that the users of GPAW should write the model description in Python and then call C and Fortran functions from within Python. It is in this context a user would apply the C implemented stencil operation on one or more real-space grids.

The parallel version of GPAW uses MPI in a flat programming model and the parallelization is done by simple domain decomposition of every real-space grid in the simulation. That is, every MPI process gets the same subset of *every* real-space grid in the simulation. This is important because some part of the GPAW

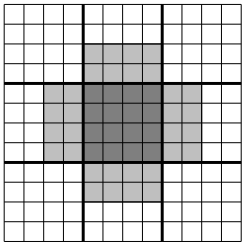


Fig. 4.2: 2D grid distributed over nine processes. A process needs some of its neighbor's surface points, to compute its own surface points.

computation, like the orthogonalization of wave-functions, requires the same subset of every real-space grid in the simulation. This domain decomposition is illustrated in Fig. 4.1 with 2D real-space grids instead of 3D grids.

The grids are simply divided into a number of quadrilaterals matching the number of available MPI processes. If no user-defined domain decomposition is present, GPAW will try to minimize the aggregated surface of the quadrilaterals. A real-space grid is represented as a three dimensional array where every point in the grid can be a real or complex value (8 or 16 bytes)

4.1. Distributed Stencil Operation. Generally, it should be easy to obtain good scalability for a distributed stencil operation since computation grows faster than communication. If we look at a 3D grid of size $n \times n \times n$ the aggregated computation is $O\left(n^3\right)$ where as the aggregated communication is only $O\left(n^2\right)$. The operation should scale very well when n grows at the same rate as the number of CPUs. In GPAW, however, scalability is very hard to obtain since the grid size will ordinarily not exceed 200^3. Thus, the n is smaller than 200 even when parallelizing over thousands of CPUs.

The fact that the number of independent grids grows linearly with the number of valence electrons that a simulated would normally make the problem embarrassingly parallel. Each MPI process could compute a whole grid without the need of any communication, since no communication between grids is required in GPAW. However, this is not possible because GPAW requires that every MPI process gets the same subset of every grid (cf. Fig. 4.1).

One feature in GPAW, which makes it easier to parallelize, is the fact that the input grid and the output grid used in the stencil operation is always two separate grids. We need therefore not consider the order in which the grid-points are computed.

Applying a stencil operation on a grid involves all MPI processes. It is possible for an MPI process to compute most of the points in the sub-grid assigned to it. However, points near the surface of the sub-grid, *surface points*, are dependent on remote points located in neighboring MPI processes. This dependency is illustrated in Fig. 4.2.

The straightforward approach, and the one used in GPAW, for making remote points available, is to exchange the surface points between neighboring MPI processes before applying the stencil operation. The serialized communication pattern looks like this:

1. Exchange surface points in the first dimension.
2. Exchange surface points in the second dimension.
3. Exchange surface points in the third dimension.
4. Apply the stencil operation.

5. Optimizations. In order to make GPAW run faster on the BGP, we have explored different optimizations. In this section, we will discuss the optimizations that have been beneficial for the overall performance.

The most obvious optimization is to exchange surface elements simultaneously in all three dimensions by using the following non-blocking communication pattern:

1. Initiate the exchange of surface points in all three dimensions.
2. Wait for all exchanges to finish.
3. Apply the stencil operation.

The idea is to fully utilize the torus network in all six directions simultaneously, see Table 3.1.

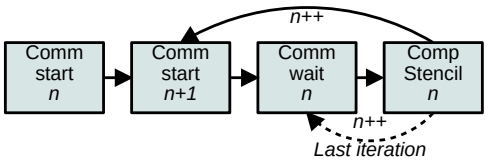


Fig. 5.1: Flow diagram illustrating double buffering. The n'th iteration is expressed with a n and Comm and Comp stands for communication and computation, respectively. n++ is an iteration to n's successor.

Another important performance aspect is how to map the distributed real-space grids onto the physical network topology. The 3D torus network is used for point-to-point communication in MPI, thus it is the network, we should attempt to map the distributed real-space grids onto. Since the grids have the same number of dimensions as the torus network, and since the stencil operation may use periodic boundary condition, a torus topology is a perfect match to our problem. However, the BGP requires a partition with 512 or more nodes to form a torus topology. A partition under 512 nodes can only form a mesh topology.

5.1. Multiple Real-space Grids. Double buffering and communication batching are two techniques which can improve the performance of the stencil operation. Both techniques requires multiple real-space grids but the stencil operation is typically applied on thousands of real-space grids.

5.1.1. Double Buffering. Double buffering is a technique that makes it possible to overlap communication and computation. The following communication pattern illustrates how (cf. Fig. 5.1):

1. Initiate the exchange of surface points in all three dimensions for the first grid.
2. Initiate the exchange of surface points in all three dimensions for the second grid.
3. Wait for all exchanges of the first grid to finish.
4. Apply the stencil operation on the first grid.
5. Initiate the exchange of surface points in all three dimensions for the third grid.
6. Wait for all exchanges of the second grid to finish.

The performance gain is dependent on the ability of the MPI library and the underlying hardware to process non-blocking send and receive calls. On the BGP, progress in non-blocking send and receive calls will be maintained by the DMA engine and increased performance is therefore expected.

5.1.2. Batching. An way to obtain critical packet size is to pack real-space grids into batches; inspired by the message size experiment (cf. Fig. 3.1).

Continuously dividing the grids between more and more MPI processes reduces the number of surface points in a single sub-grid. That is, at some point the amount of data send by a single MPI call will be reduced to a size in which the MPI overhead and network latency will dominate the communication overhead. The idea is to send a batch of surface points in each MPI call, instead of sending surface points, individually. This will reduce the communication overhead considerably, as the size of the sub-grids decreases. The number of grids packed together in this way, we call *batch-size*.

When using double buffering, it is important to allow the CPUs to start computing as soon as possible. Combining a large batch-size with double buffering will therefore introduce a penalty as the initial surface points exchange cannot be hidden. One approach to minimize this penalty, is to increase the batch-size continuously in the initial stage. For instance a batch-size of 128 could be reduced to 64 in the initial exchange. This technique we call *sloped batching*.

The amount of time used by waiting on non-hidden communication depends on many factors – some related to the runtime system and some related to the implementation. A general expression of the relationship is given in figure 5.2, which can be used to find the optimal batch-size and the optimal number of initial batch-size increasements when doing sloped batching. The CPU overhead associated with a implementation of double buffering and sloped batching is not included in the expression likewise the memory access time associated with the stencil computation is also not included.

6. Programming Approaches. Different approaches exist when combining threads and MPI. To preserve control we have chosen to handle the threading manually in pthread.

l : latency
B : bandwidth
C : computation time of one stencil element
t : total stencil size
b : batch-size
n : number of batch-size increasements initially

$$WaitTime = l + \frac{b}{2^n B} + \sum_{i=1}^{n} \max\left(0, l + \frac{b}{2^{i-1}B} - \frac{Cb}{2^i}\right) + \max\left(0, l + \frac{b}{B} - Cb\right)\left(\frac{t}{b} + 1 - 2^{n+1}\right)$$

Fig. 5.2: Formula of the amount of time used by waiting on non-hidden communication when using double buffering and sloped batching. The first line represents the initial communication, which can not overlap computation. The second line represents the sloped bashing, in which the block-size is doubled in each iteration and the third line represents the rest of the iterations, in which the block-size remains constant.

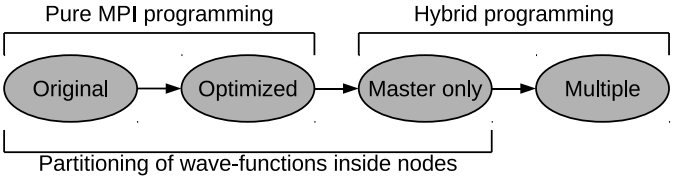


Fig. 6.1: A illustrates of the relationship between the four programming approaches – going from the Flat-original approach to the Hybrid-multiple approach.

The following is a description of different programming approaches that we have investigated. Every programming approach except the Flat-original uses the optimizations described in Sect. 5.

Flat-original is the approach originally used in GPAW. It uses the BGP's virtual mode, where the four CPU-cores are treated as individual nodes, to utilize all four CPU-cores. Therefore, it is not necessary to modify anything to support the BGP architecture.

Flat-optimized is an optimized version of the original approach and just like the Flat-original it uses the virtual mode.

Hybrid-multiple does not use the virtual mode. Instead, one hardware thread per CPU-core is spawned. Every thread handles its own inter-node communication. The node will distribute the real-space grids between its four CPU-cores, not by dividing the grids into smaller pieces but by assigning different grids to every CPU-core. Because of this no synchronization is needed until all grids are computed and the synchronization penalty is therefore constant. This way of exploiting multiple grids is the main advantage of this approach.

Hybrid-master-only also spawns one thread per CPU-core, but only one thread, the *master thread*, handles inter-node communication. Since we have to synchronize between every grid-computation, each grid-computation will be divided between the four CPU-cores. The synchronization penalty thus become proportional to the number of grids. On the other hand, this approach does work in `SINGLE` MPI-mode and the overhead associated with `MULTIPLE` is therefore avoided.

Fig. 6.1 illustrates the relationship between the four programming approaches – from the original approach, in which pure MPI programming is used and the wave-functions are partitioned inside the nodes, to the hybrid approach where hybrid programming is used and the wave-functions are shared inside the nodes.

7. Results. A benchmark of each implementation has been executed on the Blue Gene/P (Sec. 3). 16384 CPU-cores or 4096 nodes or 4 racks were made available to us. Every benchmark graph compares the different programming approaches of the stencil operation in GPAW and a periodic boundary condition is used in all cases.

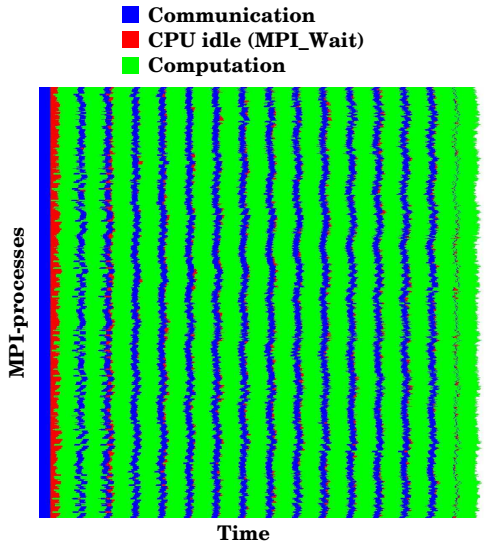


Fig. 7.1: Profile of the communication and computation pattern when computing 1024 real-space grids on 1024 CPU-cores and the Hybrid-multiple approach is used. A line represents a MPI-process and the length of the line represents the progress of time.

Fig. 7.2 is a classic speedup graph comparing every implemented approach with a sequential execution. It is a relatively small job containing only 32 real-space grids. But because of the memory demand, it is not possible to have more than 32 grids running on a single CPU-core.

The result clearly show that the best scaling and running time is obtained with Flat-optimized and Hybrid-multiple both using a batch-size of 8 grids. Since the job only consists of 32 grids a batch-size of 8 is the maximum if all four CPU-cores should be used. Another interesting observation is that the advantage of batching is greater in Hybrid-multiple than in Flat-optimized. This indicates that if a job consist of more grids, the Hybrid-multiple approach may become faster than Flat-optimized.

7.1. Communication and Computation Profile. The communication and computation profile becomes very important when scaling to a massive number of processes. As the number of MPI processes increases the communication time has a tendency to increase due to network congestion. It is therefore essential that communication is spread evenly between the CPU-core and that the diversity of the communication and computation time is minimized.

Fig. 7.1 is a profile of the Hybrid-multiple approach executing on 1024 CPU-cores. It shows a distinct pattern in which the communication and the computation phase are aligned throughout the execution. From that it is evident that Hybrid-multiple actually do execute in a fairly synchronized manner and no ripple effect of waiting processes is observed.

7.2. Multiple Real-space Grids. As the number of grids grow there is a corresponding linear growth in the computation required in the stencil operation. It is therefore possible to create a Gustafson graph by increasing the number of grids in the same rate as the number of CPU-cores (cf. Fig. 7.3). It is important to note that the required communication per node increases faster than the needed computation. This is due to the increased surface size associated with the additional partitioning of the grids. To illustrate this communication increase, the right graph in Fig. 7.3 shows the needed communication per node for Flat-optimized and Hybrid-multiple respectively.

If we, for example, look at a computation of a grid with a size of 192^3 using 1024 nodes, the grid will either be divided between 1024 MPI processes when using Hybrid-multiple or 4096 MPI process when using Flat-optimized. Flat-optimized needs to communicate approximately 140KB more data per node than Hybrid-multiple. Note that this is only for a single real-space grid, the different will grow linearly with the number of grids in the computation.

At 512 CPU-cores Hybrid-multiple is faster than Flat-optimized. The main reason is the difference in the

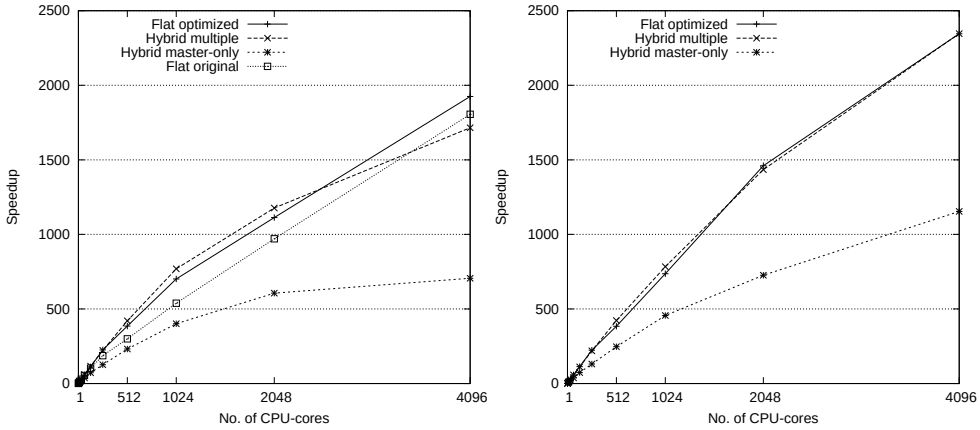


Fig. 7.2: Speedup of the stencil operation. The job consist of only 32 real-space grids all with a size of 144^3. In the left graph batching is disabled and in the right graph batching is enabled using a batch-size of 8.

needed communication. Flat-optimized divides the grids four times more than the Hybrid-multiple. We did not see this effect in the speedup graph, Fig. 7.2, because of the small number of grids. Furthermore, Hybrid-multiple is better to exploit an increase in grids because of the thread synchronization overhead. The overhead is small and constant, but since the total running time is very small for 32 grids (9 milliseconds with 2048 CPU-cores), the impact of the synchronization overhead is drastically reduced when the number of grids, and thereby the total running time, is increased.

To investigate the scalability of a large job with many real-space grids, we have made a scalability graph beginning at 1k CPU-cores, which allows for a 2816 grid job (cf. Fig. 7.4). Again Hybrid-multiple has the best performance - going from 1k to 16k CPU-cores gives a speedup of approximately 12.5 where 16 would be linear but unobtainable due to the increase in the needed communication. If we compare the running time of Hybrid-multiple with Flat-original, we see a 94% performance gain at 16384 CPU-cores.

To further investigate the performance difference between Hybrid-multiple and Flat-optimized, we have made a small experiment. We modifies Flat-optimized to statically divide the real-space grids into four sub-groups. It is now possible for all four CPU-cores to work on its own sub-group and the real-space grids will be divided into the same level as in Hybrid-multiple. The only difference between the two approaches is that Flat-optimized uses the virtual mode in Blue Gene/P and Hybrid-multiple uses threads. It should be noted, however, that in a real GPAW computation this modification does not work, since GPAW requires that every MPI process gets the same subset of every real-space grid, see Sect 4. The experiment is not included in any of the graphs since its performance is identical with the Hybrid-multiple. Because of the identical performance, we find it reasonable to conclude that the level of real-space partitioning is the sole reason for the performance difference between Hybrid-multiple and the non-modified Flat-optimized.

8. Conclusions. Overall this work has managed to improve the performance of a domain specific stencil code when scaling to a very high degree of parallelism. The primary improvements are obtained through the introduction of asynchronous communication which, even in a well balanced system such as the Blue Gene, efficiently improves processor utilization. Furthermore, two hybrid programming approaches have been explored: the hybrid multiple and the master-only approach.

The hybrid programming approach, in which inter-node communication is handled individually by every thread, has shown a positive impact on the performance. By allowing every thread to handle its own inter-node communication, the overhead for thread synchronization remains constant and the application becomes faster than the non-hybrid version.

On the other hand, the alternative hybrid programming approach, in which one thread handles the inter-node communication on behalf of all threads in the process, cannot compete with the non-hybrid version. That is explained by the overhead that is introduced by thread synchronization which grows proportional to the number of grids in the computation.

When comparing our fastest implementation compared to the original implementation, the hybrid program-

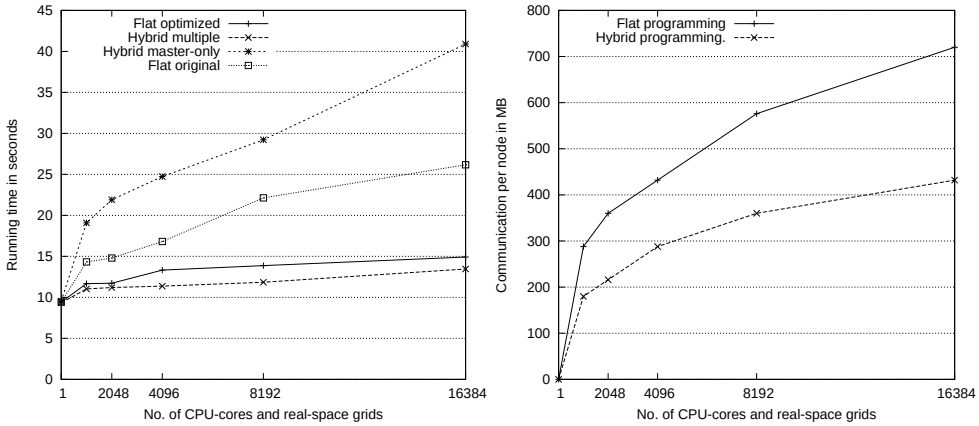


Fig. 7.3: Gustafson graphs showing the running time of the stencil operation and the needed inter-node communication when the number of real-space grids is increasing in the same rate as the number of CPU-cores - one grid per CPU-core. The left graph shows the running time and the right graph shows the needed inter-node communication. The grid size are 192^3 and the best batch-size has been found for every number of CPU-cores.

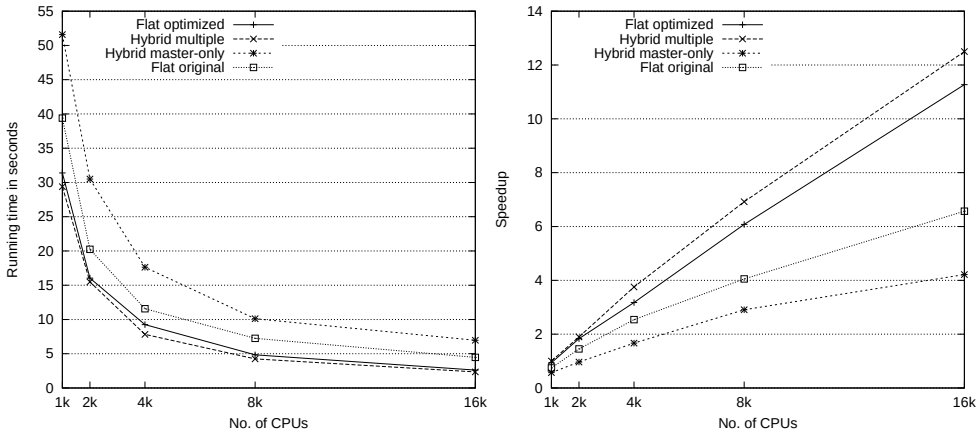


Fig. 7.4: A scalability graph starting at 1024 CPU-cores running the stencil operation. In the left graph the running time of every approach is shown and in the right graph every approach is compared against the fastest approach on 1024 CPU-cores namely the Hybrid-multiple. All jobs consists of 2816 real-space grids all size of 192^3, and the best batch-size has been found for every number of CPU-cores.

ming approach combined with the latency-hiding techniques is 94% faster at 16384 CPU-cores. Translated into utilization this means that CPU utilization grows from 36% to 70%.

While latency-hiding is the primary factor for the improvement we observe, the hybrid implementation is still 10% faster than the non-hybrid approach.

8.1. Further Work. Overall we are satisfied with the performance of the new implementation of the stencil operation, still a lot of work remains if the entire GPAW computation should utilize latency-hiding and hybrid programming. It may not be worth the hard work that is needed to rewrite most of GPAW.

Acknowledgments. The authors would like to thank The Danish Agency for Science, Technology and Innovation and the GPAW team at the Technical University of Denmark in particular Jens J. Mortensen and Marcin Dulak. Furthermore we would like to thank Argonne National Laboratory for giving us access to the Blue Gene/P.

REFERENCES

[1] J. Backus, *Can programming be liberated from the von neumann style?: A functional style and its algebra of programs*, Communications of the ACM, 16 (1978), pp. 613–641.
[2] P. E. Blochl, *Projector augmented-wave method*, Phys. Rev. B, 50 (1994), pp. 17953–17979.
[3] F. Cappello and D. Etiemble, *Mpi versus mpi+openmp on the ibm sp for the nas benchmarks*, SC Conference, 0 (2000), p. 12.
[4] W. Gropp, S. Huss-Lederman, A. Limsdaine, E. Lusk, W. Saphir, and M. Snir, *The Complete Reference: Volume 2, the MPI-2 Extensions*, MIT Press, 1998.
[5] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI Portable Parallel Programming with the Message Passing Interface*, The MIT Press, 1994.
[6] D. S. Henty, *Performance of hybrid message-passing and shared-memory parallelism for discrete element modeling*, Supercomputing, ACM/IEEE 2000 Conference, (2000), pp. 10–10.
[7] M. Hipp and W. Rosenstiel, *Parallel Hybrid Particle Simulations Using MPI and OpenMP*, Springer-Verlag Berlin Heidelberg, 2004, pp. 189–197.
[8] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen, *Real-space grid implementation of the projector augmented wave method*, Physical Review B, 71 (2005), p. 035109.
[9] I. B. G. TEAM, *Overview of the ibm blue gene/p project*, IBM Journal of Research and Development, 52 (2008).
[10] B. Vinter and J. M. Bjørndalen, *A comparison of three mpi implementations*, in Communicating Process Architectures 2004, I. R. East, D. Duce, M. Green, J. M. R. Martin, and P. H. Welch, eds., 2004, pp. 127–136.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

Scalable Computing: Practice and Experience
Volume 12, Number 2, pp. 275–282. http://www.scpe.org
ISSN 1895-1767

SOME GEOMETRIC PROBLEMS ON OMTSE OPTOELECTRONIC COMPUTER

SATISH CH. PANIGRAHI* AND ASISH MUKHOPADHYAY†

Abstract. Optical Multi-Trees with Shuffle Exchange (OMTSE) architecture is an efficient model of an optoelectronic computer. The network has a total of $3n^3/2$ nodes. The diameter and bisection width of the network are $6\log n - 1$ and $n^3/4$ respectively. In this note, we present synchronous SIMD algorithms on an OMTSE optoelectronic computer for the following problems in computational geometry: Convex Hull, Smallest Enclosing Rectangle, All-Farthest/All-Nearest Neighbors, Closest/Farthest pair, Maximal Points. The strength of the proposed algorithms over the existing algorithms on OMULT has also been discussed.

Key words: Parallel Algorithms, Optoelectronic Computer, Computational Geometry, OTIS Mesh, OMULT

1. Introduction. Optical interconnections are superior in power, speed with less crosstalk properties as compared to electronic interconnections when the interconnection distance is more than a few millimeters [1, 6]. Motivated by these observations, some new hybrid optoelectronic computer architectures utilizing both optical and electronic technologies have been proposed and investigated by several researchers [8, 10, 13, 15]. In these architectures, both the electronic link and the optical link are done where the former is being considered within the same physical package (e.g. chip) where as the latter is for the pair of processors that are kept in different packages.

A number of parallel algorithms on these optoelectronic computers have been addressed and studied extensively [3, 4, 5, 7, 8, 9, 10, 14]. In this paper we present some computational geometry algorithms such as Convex Hull, Smallest Enclosing Rectangle, All-Farthest/All-Nearest Neighbor, Closest/Farthest pair, Maximal Points, on OMTSE optoelectronic computer [8, 10]. Irrespective of different factor network of OMTSE than OMULT, here in this paper we show that Convex hull and Smallest Enclosing Rectangle problem for n points can be solved on OMTSE in $O(\log n)$ time with the same time complexity as on OMULT [2]. Here it is worth noting that the total number of processors of OMULT and OMTSE respectively be $\delta 1 = n^2(2n-1)$ and $\delta 2 = n^2(\frac{3n}{2})$ (we have $\delta 1 < \delta 2$, as because of their topological nature we can assume that $n \geq 4$). Islam et al. in [2] stated that algorithm for empirical cumulative distribution, all nearest neighbor can be implemented on OMULT in $O(\log n)$ time for n number of points. In this paper we explore this line of work farther and implement the algorithms such as All-Farthest/All-Nearest Neighbor, Closest/Farthest pair among n^2 points in $O(n \log n)$ time and also provide an algorithm for maximal points among n^3 data points in $O(\log n)$ time.

The rest of the paper is organized as follows. In section 2 we briefly present the topological property of the OMTSE System. In section 3, we describe our propose algorithms and finally we conclude in section 4.

2. Topology of OMTSE. The factor network used in OMTSE topology constitutes two layer Trees with Shuffle Exchange (TSE) network. The TSE is nothing but an interconnection network containing a group of $2^k, k \geq 1$, complete binary trees of height one and the roots of these binary trees are connected with Shuffle-Exchange fashion. The OMTSE interconnection system consists of n^2 TSE networks, which are organized in the form of an $n \times n$ grid in matrix form. We denote the TSE network placed at i_{th} row and j_{th} column of this matrix by $G_{ij}, 1 \leq i, j \leq n$. Each TSE network has n nodes at layer 2 and $n/2$ nodes at layer 1 which results in $N = 3n^3/2$ processors in total. The nodes within each TSE network are interconnected by usual electronic links, while the nodes at layer 2 (i.e. the layer having leaf processors) of different TSE networks are interconnected by optical links according to the rules defined below. Let us label the nodes in each TSE network $G_{ij}, 1 \leq i, j \leq n$, by distinct integers from 1 to $3n/2$ in reverse order, i.e., the nodes at both layer 2 and 1 of TSE network are numbered from 1 to $3n/2$ in order from left to right. The node, k, in a TSE network G_{ij} will be referred as the processor $P(i, j, k), 1 \leq i, j \leq n, 1 \leq k \leq 3n/2$. We can now define the optical links interconnecting only leaf nodes in different TSE networks in the following way.

*School of Computer Science, University of Windsor, Canada(panigra@uwindsor.ca).
†School of Computer Science, University of Windsor, Canada(asishm@cs.uwindsor.ca)

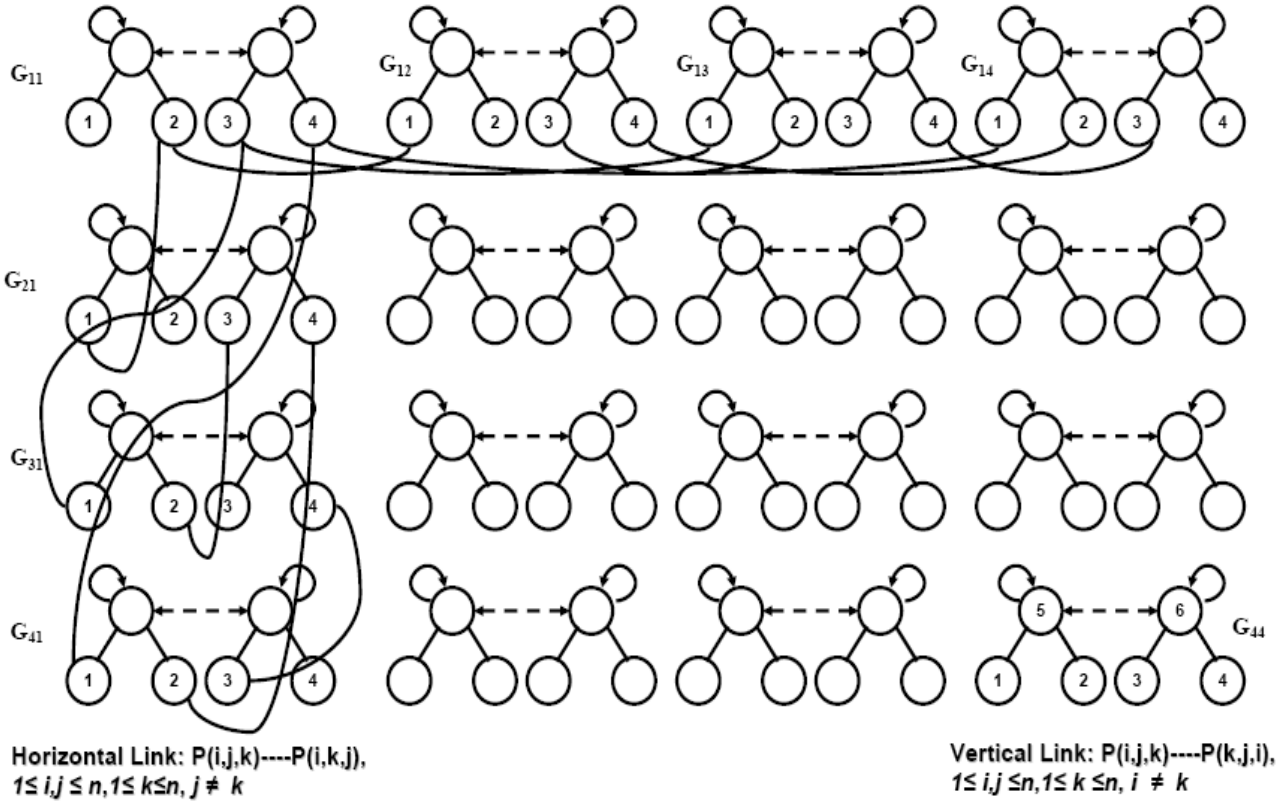


Fig. 2.1: An example of OMTSE topology with n = 4

(1) Processor $P(i,j,k), 1 \leq i,j,k \leq n, j \neq k$, is connected to the processor $P(i,k,j)$ by bi-directional optical link called horizontal inter-TSE link.

(2) Processor $P(i,j,k), 1 \leq i,j \leq n, i \neq k$, is connected to the processor $P(k,j,i)$ by bi-directional optical link called vertical inter-TSE link.

The diameter of a network is defined as the maximum distance between any two processing nodes in the network. If we start from a node $P(i,j,k), 1 \leq i,j \leq n, 1 \leq k \leq 3n/2$, we can reach another node $P(i',j',k'), 1 \leq i',j' \leq n, 1 \leq k' \leq 3n/2$, of the OMTSE interconnection system by traversing the path

$P(i,j,k) \rightarrow P(i,j,j') \rightarrow P(i,j',j) \rightarrow P(i,j',i') \rightarrow P(i',j',i) \rightarrow P(i',j',k')$

It can easily be seen that the diameter of OMTSE topology is $6\log n - 1$ which is $O(\log n)$ comprising of $6\log n - 3$ electronic links and 2 optical links. Similarly we can find out the bisection width of OMTSE topology is equal to $n^3/4$. An Example of OMTSE topology for $n = 4$ with partial links is shown in FIG. 2.1.

3. Proposed Algorithms.

3.1. Convex Hull. The convex hull [11] of a set of points S in the plane is smallest convex polygon P that encloses S, smallest in the sense that there is no other polygon P' such that $P \supset P' \supseteq S$. To find the convex hull for a given set of points S on a plane we need to identify the extreme points, in particular, what constitutes constructing the boundary. Suppose $|S| = n$ and assume that no three points in S are collinear then our algorithm employs the result of the following theorem discussed in [14].

THEOREM 3.1. *For any point $p_i \in S$, let $p_{j0}, p_{j1}, ..., p_{jn-2}$ be the points in $S - p_i$ (i.e. $p_{jk} \neq p_i, 0 \leq k \leq n-2$), sorted by the polar angle made by the vector $\overrightarrow{p_i p_{jk}}, 0 \leq k \leq n-2$. The point p_i is an extreme point of S iff there is a $k, 0 \leq k \leq n-2$, such that counterclockwise angle between p_{ik} and $p_{i(k+1)mod(n-1)}$ is more than π.*

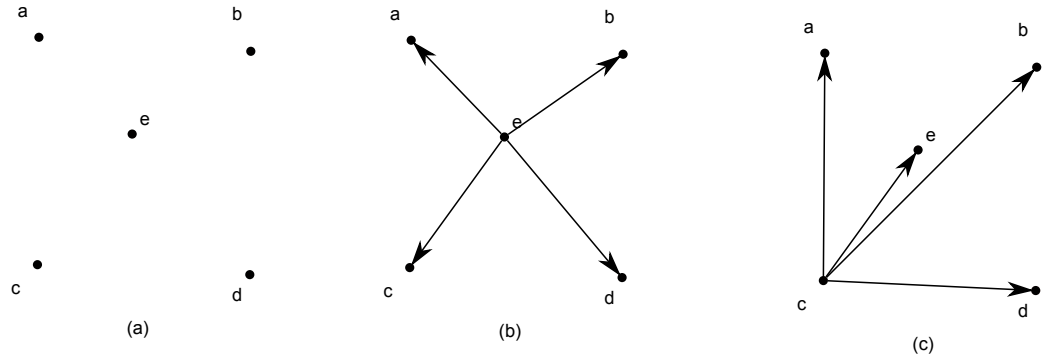


Fig. 3.1: Example for Theorem 3.1:(a) Original Layout, (b) $p_i = e$, (c) $p_i = c$

We assume that each leaf processor $P(i,j,k)(1 \leq i,j,k \leq n)$ has three registers, represent by $A(i,j,k)$, $B(i,j,k)$ and $C(i,j,k)$. We have a set of points $S = p_1, p_2, ..., p_n$ in which no three points are collinear. The coordinates of all n points are initially stored in the A-register of the leaf nodes G_{11}.

Algorithm: $ConvexHull()$

Input: $\forall k, 1 \leq k \leq n$
$A(1,1,k) \leftarrow p_k$
Output: $\forall k, 1 \leq k \leq n$
Extreme points ← B(1, 1, k)

Step 1: $\forall i,j; 1 \leq i,j \leq n$, do in parallel
Broadcast all these n points to the A-register of the respective leaf nodes of G_{ij} [9].
Step 2: $\forall i,j,k; 1 \leq i,j,k \leq n$, do in parallel
Broadcast the point in the A-register of $P(i,j,i)$ to all the $B(i,j,k)$ of G_{ij}.
Step 3: $\forall i,j,k; 1 \leq i,j,k \leq n$, do in parallel
Compute the polar angle of the vector $p_i\vec{p}_{ik}$ at $P(i,j,k)$ of G_{ij} and store in $C(i,j,k)$ along with the zero vector.
Step 4: $\forall i,j,k; 1 \leq i,j,k \leq n$, do in parallel
Sort the n vectors $p_i\vec{p}_{ik}$ stored in the C-register of the leaf nodes of each G_{ij}. After this step we assume the sorted order list given by each G_{ij} is $p_ip_{i1}, p_ip_{i2}, ..., p_ip_{in}$ (i.e. in each G_{ij} the vector $p_i\vec{p}_{i1}$ always represent the zero vector.)
Step 5: $\forall i,j,k; 1 \leq i,k \leq n$, and $2 \leq j \leq n$, do in parallel
Broadcast the content of $C(i,j,j)$ to $A(i,j,k)$.
Step 6: $\forall i; 1 \leq i \leq n$, do in parallel
i) $\forall j, 2 \leq j \leq n-1$,
Calculate the counter clockwise angle between $p_i\vec{p}_{ij}$ and $p_i\vec{p}_{i(j+1)}$ at each G_{ij} and store the result in $C(i,j,j+1)$
ii) $\forall j, j = n$,
Calculate the counter clockwise angle between $p_i\vec{p}_{in}$ and $p_i\vec{p}_{i2}$ at each G_{in} and store it in $C(i,n,2)$.
Step 7: $\forall i; 1 \leq i \leq n$, do in parallel
i) $\forall j; j = 1$
$C(i,j,1) \leftarrow 0$.

ii) $\forall j; 2 \leq j \leq n-1$
if $((C(i,j,j+1) > \pi))$
$C(i,j,1) \leftarrow 1.$
else
$C(i,j,1) \leftarrow 0.$
iii) $\forall j; j = n$
if $((C(i,n,2) > \pi))$
$C(i,n,1) \leftarrow 1.$
else
$C(i,n,1) \leftarrow 0.$

Step 8: $\forall i,j; 1 \leq i,j \leq n$, do in parallel
$C(i,1,j) \leftarrow C(i,j,1)$. /* through the horizontal optical link content of $C(i,j,1)$ is moved to $C(i,j,1)$ */

Step 9: $\forall i,k; 1 \leq i,k \leq n$, do in parallel
if $(C(i,1,k) == 0)$
$B(i,1,1) \leftarrow NULL$. /* if the content of C-register of all leaf nodes of G_{i1} is 0 then reset $B(i,1,1)$ to NULL value */

Step 10: $\forall i, 1 \leq i \leq n$, do in parallel
$B(1,1,i) \leftarrow B(i,1,1)$. /* through the vertical optical link content of $B(i,1,1)$ is moved to $B(1,1,i)$ */

Hence the extreme points of the convex hull can be taken from the B-register of all leaf nodes of G_{11} excluding the NULL entries. In order to analyze the time complexity of the above algorithm we also consider the data movements along the both electronic link and optical link. For the complete group broadcast [9] the step 1 needs $4\log n - 2$ electronic moves and 3 optical moves. For the required intra-group group broadcast [8] the step 2 and 5 need $2\log n - 1$ electronic move. For the basic assignment and geometry operations we can assume that the steps 3, 6, 7 and 9 need $O(1)$ time. The required sorting (see appendix) of n points at corresponding $G_{ij}, 1 \leq i,j \leq n$ the step 4 needs $7\log n - 1$ electronic move and 5 optical move. In addition, for the required inter-group data movement the step 8 and 10 need one optical move. Thus overall, we need $O(\log n)$ to compute the convex hull.

THEOREM 3.2. *Algorithm PCH requires $O(\log n)$ time to compute the convex hull of n points.*

The above algorithm can be extended for the smallest enclosing rectangle of n points within $O(\log n)$ time as discussed in [2]. But it would be interesting to devise algorithm for convex hull and smallest enclosing rectangle among n^2 data points on both OMULT and OMTSE optoelectronic computer.

3.2. All-Nearest/All-Farthest Neighbor. All-Nearest(All-Farthest) Neighbor problem can be stated as follows: given a set $S = \{p_1, p_2, ..., p_q\}$ of q points, for each point $p_i \in S$ we wish to determine a point $p_j \in \{S - p_i\}$ such that the Euclidean distance $\|p_i - p_j\|$ is minimum(maximum).

In order to implement All-Nearest Neighbor (All-Farthest Neighbor can be dealt analogously) problem for n^2 points, we assume that each leaf processor $P(i,j,k), 1 \leq i,j,k \leq n$, has four registers A, B, C and D; where as each non-leaf processor $P(i,j,k), 1 \leq i,j \leq n, n+1 \leq k \leq \frac{3n}{2}$, has two registers A and B. Initially, the points $p_{(i-1)+k}$ is stored in the A(i, i, k)of all the diagonal leaf nodes of $G_{ii}, 1 \leq i \leq n$, where as all the D-registers of OMTSE system are set to zero. Set a counter variable c to zero at B-register of each non leaf processor of OMTSE optoelectronic system. Here we describe the algorithm in the following steps

Algorithm $AllNearestNeighbor()$
Input: $\forall i,k, 1 \leq i,k \leq n$
$A(i,i,k) \leftarrow p_{(i-1)+k}$
Output: $\forall i,k, 1 \leq i,k \leq n$
Nearest Neighbor of $p_{(i-1)+k} \leftarrow C(i,i,k)$

Step 1: Perform a column group broadcast [8].
Step 2: While $(c < n)$ do

Step 2.1: $\forall i, j, 1 \leq i, j \leq n$, do in parallel
if $(c == 0)$
Broadcast the content of $A(i, j, i)$ to B-registers of all leaf nodes of G_{ij}.
else
Broadcast the content of $B(i, j, 1 + (j\%n))$ to B-register of all leaf nodes of G_{ij}.
Step 2.2: $\forall i, j, k, 1 \leq i, j, k \leq n$, do in parallel
$D(i, j, k) \leftarrow \|A(i, j, k) - B(i, j, k)\|$
if $(D(i, j, k) == 0)$
$D(i, j, k) \leftarrow \infty$
Step 2.3: $\forall i, j, k, 1 \leq i, j, k \leq n$, do in parallel
Compute the minimum of values stored in each D-register of G_{ij} and store the result in $C(i, j, 1 + ((j + c - 1)\%n))$.
Step 2.4: $\forall i, j, k, 1 \leq i, j, k \leq n$,do in parallel
Perform horizontal optical move on the content of B-registers so that the data from each side move to the corresponding leaf nodes.
Step 2.5: $\forall i, j, k, 1 \leq i, j \leq n, n + 1 \leq k \leq \frac{3n}{2}$, do in parallel
$c = c + 1$.

Step 3: $\forall i, j, k, 1 \leq i, j, k \leq n$ and $j \neq k$, do in parallel
Perform horizontal optical move on the content of C-registers so that the data from each side move to the corresponding leaf nodes.

Step 4: $\forall i, k, 1 \leq i, k \leq n$, do in parallel
Compute the minimum of values stored in each C-register of G_{ki} and store the result in $C(k, i, i)$.

Step 5: $\forall i, k, 1 \leq i, k \leq n$ and $i \neq k$, do in parallel
$C(i, i, k) \leftarrow C(k, i, i)$/* Vertical Optical Move */

For the required column group broadcast the step 1 requires $2 \log n - 1$ electronic moves and 3 optical moves. The Step 2.1 requires $2 \log n - 1$ electronic moves for intergroup broadcast. To find the minimum in each group G_{ij}, the step 2.3 and step 4 require $O(\log n)$ time. Again the Step 2.4, Step 3 and Step 5 require one optical move each. For the basic increment and distance measure we can assume that the Step 2.2 and Step 2.5 require $O(1)$ time. Since we have n iterations of while loop in Step 2, the overall complexity of the algorithm is $O(n \log n)$ for n^2 points.

3.3. Closest-Pair/Farthest-Pair of Points. This problem can be defined as follows: given a set $S = \{p_1, p_2, ..., p_q\}$ of q points, $\exists\{p_i, p_j\} \in S$ such that euclidean distance $\|p_i - p_j\|$ is minimum(maximum). The closest pair of points can be found by first solving the All-Nearest neighbor problem and then determining the closest pair among the nearest problem of each point. Here we describe the basic algorithm for n^2 points in following steps

Algorithm: $ClosestPairPoints()$

Input: $\forall i, k, 1 \leq i, k \leq n$
$A(i, i, k) \leftarrow p_{(i-1)+k}$
Output: Closest-pair $\leftarrow C(1, 1, 1)$

Step 1: $AllNearestNeighbor()$
Step 2: $\forall i, 1 \leq i \leq n$
Compute the minimum at each G_{ii} and store the result in $C(i, i, 1)$
Step 3: $\forall i, 1 \leq i \leq n$
$C(1, i, i) \leftarrow C(i, i, 1)$
Step 4: $\forall i, 1 \leq i \leq n$
$C(1, i, 1) \leftarrow C(1, i, i)$
Step 5: Compute the minimum at G_{11} and store the result in $C(1, 1, 1)$

The algorithm *ClosestPairPoints* require additional $3\log n - 1$ electronic moves and 2 optical moves which will be subsumed by the $O(n \log n)$ of *AllNearestNeighbor* algorithm.

3.4. ECDF. In ECDF (empirical cumulative distribution function) problem [14], we are given a set $S = \{p_1, p_2, ..., p_q\}$ of q distinct points. For $\{p_i(x_i, y_i), p_j(x_j, y_j)\} \in S$, we will say p_i dominates p_j iff $x_i \geq x_j$ and $y_i \geq y_j$. For all $p_i \in S$, we are going to determine the number points it dominates in set S. In the FIG 3.2 we have illustrated a dominating relationship between three points p_1, p_2, and p_3. In this case, the number of points dominated by p_1, p_2 and p_3, respectively are 1, 1, and 0.

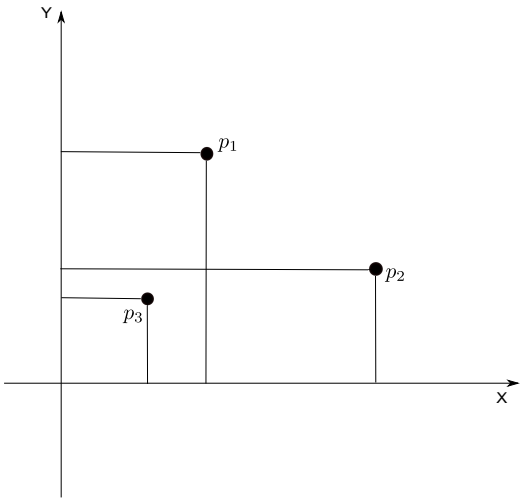


Fig. 3.2: Example of dominating relation

The algorithm to implement ECDF for n^2 is quite similar to the *AllNearestNeighbor* algorithm. Here in order to get dominating value of point $p_{(i-1)+k}$ at corresponding $C(i, i, k)$, in step 2.2 of *AllNearestNeighbor* algorithm the D-register value is set to 1 if B-register point dominates its A-register point. Then in Step 2.3, we need to compute the summation of all D-register value with in that group and store the result in $C(i, j, 1 + ((j + c - 1)\%n))$. After this the D-register is reset to zero and contiue the loop while $c < n$. Further, in Step 4 we need to compute the summation of values stored in each C-register of G_{ki} and store the result in $C(k, i, i)$. Finally in Step 5 we get the dominating value of point $p_{(i-1)+k}$ at corressponding $C(i, i, k)$.

To compute the summation in shuffle exchange network [12] takes the same complexity as to compute the minimum. Thus the time taken to implement ECDF for n^2 points is same as that of *AllNearestNeighbor* algorithm i. e. $O(n \log n)$.

3.5. Two-Set Dominance. The two set dominance problem can stated in this way: We have given two sets $S_1 = \{p_1, p_2, ..., p_p\}$ and $S_2 = \{q_1, q_2, ..., q_q\}$, for each point $p_i \in S_1 (or\ q_j \in S_2)$ we wish to determine the number points in $S_2 (or\ S_1)$ is dominated by $p_i (or\ q_j)$. This is quite similar to the ECDF and can be achieved with $O(n \log n)$ for $\|S_1 + S_2\| = n^2$ points.

3.6. Maximal Points. A point $p \in S$ is maximal iff it dominates all the points in S. This is quite simple and can be achieved by $O(\log n)$ time for $\|S\| = n^3$ points as follows.

Algorithm: *MaximalPoint*
Input: Arbitrarily assign the n^3 points to n^3 leaf processors of OMTSE optoelectronic computer.
Output: Maximal point $\leftarrow A(1, 1, 1)$
Step 1: $\forall i, j, 1 \leq i, j \leq n$,
Each group G_{ij} determine the maximal point with in that group and store the result in $A(i, j, 1)$
Step 2: $\forall i, j, 1 \leq i, j \leq n$,

$A(i,1,j) \leftarrow A(i,j,1)$

Step 3: $\forall i, 1 \leq i \leq n$,
Each group G_{i1} determine the maximal point with in that group and store the result in $A(i,1,1)$

Step 4: $\forall i, 1 \leq i \leq n$,
$A(1,1,i) \leftarrow A(i,1,1)$

Step 5: The group G_{11} determine the maximal point with in that group and store the result in $A(1,1,1)$

For finding the local maximal points with in a group, the Step 1, 3 and 4 requires $O(\log n)$ electronic moves each. Further, for the inter group communication we require one optical move each for the Step 2 and 4. Thus overall we have $O(\log n)$ algorithm with exactly $3\log n$ electronic moves and 2 optical moves. Now if we define minimal points analogous to maximal points, the above algorithm can be improved slightly to get both the maximal and minimal points out of $n(n-1)^2$ points with $4\log n + 4$ electronic moves and 3 optical moves as discussed in [10].

4. Conclusion. We have shown that several computational geometry problems can be solved on OMTSE optoelectronic computer efficiently. It would be interesting to devise the discussed algorithms for n^3 number of points on OMTSE and OMULT system.

Acknowledgments. This research is supported by an NSERC Individual Discovery Grant to second author.

REFERENCES

[1] M. R. Feldman, S. C. Esener, C. C. Guest, and S. H. Lee, *Comparison between optical and electrical interconnects based on power and speed considerations*, Appl. Opt., 27 (1988), pp. 1742–1751.

[2] R. Islam, N. Afroz, S. Bandyopadhyay, and B. P. Sinha, *Computational geometry on optical multi-trees (OMULT) computer system*, in CCCG, 2005, pp. 150–154.

[3] P. K. Jana, *Improved parallel prefix computation on optical multi-trees*, in India Annual Conference, 2004. Proceedings of the IEEE INDICON 2004. First, 20-22 2004, pp. 414 – 418.

[4] P. K. Jana, *Polynomial interpolation and polynomial root finding on otis-mesh*, Parallel Computing, 32 (2006), pp. 301–312.

[5] P. K. Jana and K. Sinha, *Permutation algorithms on optical multi-trees*, Comput. Math. Appl., 56 (2008), pp. 2656–2665.

[6] A. V. Krishnamoorthy, P. J. Marchand, F. E. Kiamilev, and S. C. Esener, *Grain-size considerations for optoelectronic multistage interconnection networks*, Appl. Opt., 31 (1992), pp. 5480–5507.

[7] D. K. Mallick and P. K. Jana, *Parallel prefix on mesh of trees and otis mesh of trees*, in PDPTA, H. R. Arabnia and Y. Mun, eds., CSREA Press, 2008, pp. 359–.

[8] S. C. Panigrahi, S. Paul, and G. Sahoo, *OMTSE - an optical interconnection system for parallel computing*, in Advanced Computing and Communications, 2006. ADCOM 2006. International Conference on, 20-23 Dec 2006, pp. 626 –627.

[9] ———, *Parallel prefix computation, sorting and reduction operation on OMTSE architecture*, in ICACC 2007 International Conference, 9-10 Feb 2007, pp. 616 –622.

[10] S. C. Panigrahi and G. Sahoo, *An MIMD algorithm for finding maximum and minimum on OMTSE architecture*, Scalable Computing: Practice and Experience, 9 (2008), pp. 69–75.

[11] F. P. Preparata and M. I. Shamos, *Computational geometry: an introduction*, Springer-Verlag, New York, 1985.

[12] M. J. Quinn, *Parallel computing (2nd ed.): theory and practice*, McGraw-Hill, Inc., New York, NY, USA, 1994.

[13] B. P. Sinha and S. Bandyopadhyay, *OMULT: An optical interconnection system for parallel computing*, in Euro-Par, M. Danelutto, M. Vanneschi, and D. Laforenza, eds., vol. 3149 of Lecture Notes in Computer Science, Springer, 2004, pp. 856–863.

[14] C.-F. Wang and S. Sahni, *Computational geometry on the OTIS-mesh optoelectronic computer*, in ICPP, IEEE Computer Society, 2002, pp. 501–.

[15] F. Zane, P. Marchand, R. Paturi, and S. Esener, *Scalable network architectures using the optical transpose interconnection system (OTIS)*, J. Parallel Distrib. Comput., 60 (2000), pp. 521–538.

Appendix

For the sake of explaining the basic idea, in this appendix we discuss how the sorting of n distinct elements can be performed in OMTSE optoelectronic computer. Let's assume that each processor $P(i,j,k), 1 \leq i,j,k \leq n$, has two registers $R1(i,j,k)$ and $R2(i,j,k)$. Initially we have n distinct elements $\{a_1, a_2, a_3, ..., a_n\}$ stored in R1-register of n leaf nodes of G_{11}. We can sort these elements by finding rank of each element in the list. Thus the objective of the algorithm is to place the element of rank $r, 1 \leq r \leq n$ in the processor $P(1,1,r)$.

Algorithm $Sort()$

Step 1: Perform a column broadcast [8] so that the list of elements stored in the leaf nodes of G_{11} broad casted to the corresponding leaf nodes all $G_{i1}, 1 \leq n$.
Step 2: $\forall i, 1 \leq i \leq n$, do in parallel
Broadcast the element a_i to R2-register of all leaf nodes of G_{i1}. Set a Flag as 1 if a_i greater than other element in R1-register of same leaf node. Otherwise set Flag as zero. The value of the Flag variable can be kept in R2-register which may overwrite previous entries.
Step 3: $\forall i, 1 \leq i \leq n$, do in parallel
Compute the summation of all Flag values stored on each leaf nodes of G_{i1}, which is the rank(r) of the element a_i in the given list.
Remark: As a result of summation in the shuffle exchange network [12] the rank value will reflect in all nodes of shuffle exchange layer of G_{i1}.
Step 4: $\forall i, 1 \leq n$, do in parallel
if the rank of a_i is r then the element a_i is moved to $R1(i, 1, r)$.
Step 5: $\forall i, 1 \leq n$, do in parallel
$R1(r, 1, i) \leftarrow R1(i, 1, r)$ /* Vertical optical link */
Step 6: $\forall i, 1 \leq n$, do in parallel
$R1(r, 1, 1) \leftarrow R1(r, 1, i)$
Step 7: $\forall r, 1 \leq r \leq n$, do in parallel
$R1(1, 1, r) \leftarrow R1(1, 1, r)$ /* Vertical optical link */

For the complexity analysis of the above algorithm we also consider the data movement along the electronic and optical link. The above algorithm needs $(7 \log n - 1)$ communication steps along electronic links and 5 communication steps along optical links [8] giving overall $O(\log n)$ time algorithm. The idea can be extended to sort n^2 data values in $O(n \log n)$ time but this is beyond the scope of this paper.

Edited by: Dana Petcu and Marcin Paprzycki
Received: May 1, 2011
Accepted: May 31, 2011

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.