Multifaceted Web Services: An Approach to Secure and Scalable Grid Scheduling

Sandeep Prabhakar sprabhak@vt.edu
Calvin Ribbens ribbens@vt.edu
Prachi Bora pbora@vt.edu

Department of Computer Science
Virginia Tech
660 McBryde Hall
Blacksburg, Virginia-24061

Abstract

A multifaceted or multi-interface web service is a web service that offers interfaces to clients and to other peer web services. The multifaceted web service uses a generic parameter-based approach developed in this paper to allow for collaboration between web servers. This parametric approach solves security and scalability problems and is found to be applicable in many situations. Such an approach is necessary to permit easy collaboration and secure resource sharing. We describe one instantiation of such an approach in the form of a global grid scheduler for data parallel (or Single Program Multiple Data) programs. This scheduler demonstrates the usefulness of our approach in current business environments where administrative policies are a major factor in scheduling decisions.

Keywords: Parameterized Scheduling, Secure Scheduling, Scalable Scheduling, Global Scheduling, Multi Interface Web Service, Application-Specific Scheduling

1. Introduction

A wide variety of application domains are using web services as a paradigm for enabling loosely coupled and extensible systems [20, 23]. In the web services framework, there are software components running on web servers, each providing a different service. Current web services are standalone entities that do not interact with other web services to better provide its own service. This means that current web services delegate complete modules of responsibility to other web services. If a service provided to a client requires the service of another module, it contacts the third party service to make use of its facilities. A typical example of this might be a vendor asking to use "Microsoft passport" for authentication during purchase of a product. But what if one web service required information from other services to perform its function effectively? There is a need for multiple web services of the ***same*** type to interact with each other within the constraints of administrative policies. Thus there is a need for multi-interface (or multifaceted) web services. These multi-interface web services offer one interface by means of which clients can request a service and receive replies. They offer another interface to peer components which they talk to in order to provide better service. Such a model of service is very similar to branch office - main office connectivity. If the branch office can provide the service by itself, it does so. But if it cannot due to lack of stock, then it would need to contact its peer branch offices or main office to verify if stocks are available.

Multi-interface web services can be used in grid scheduling. For example, suppose a user (or client) has an application to be run on the grid. He contacts the scheduler web service. If the scheduler web service can satisfy the request by itself, it does so. Otherwise it interacts with peer scheduler web services to satisfy the request. In this paper, we propose a method to facilitate peer interaction that is secure and scalable. It is secure because the web server in your domain that makes scheduling decisions for that domain. Thus, complete information about the structure of an organization's resources is not exposed in any way. Our method is scalable because each peer is not overwhelmed with resource information from all over the world. In our approach, schedulers decide which peers to contact by means of a few parameters extracted from each domain that represent how suitable a domain is for scheduling. Since the number of

parameters is small, the overhead of information gathering is low; hence, scheduling can take place on a truly global basis.

The rest of the paper is organized as follows. Section 2 relates our work to the enormous body of work being done in scheduling. This section shows how our work is different in a fundamental way with regard to tackling some of the basic security problems. Section 3 first gives general principles on which our solution rests and identifies the fundamental reasoning behind this solution. The rest of Section 3 applies these principles to the global scheduling problem constrained by security and administrative concerns. It gives the steps followed in getting a job scheduled and provides an intuitive argument as to why scheduling time is reduced. Sections 4 and 5 summarize our contributions and identify how it can be extended to other domains.

2. Related Work

Typically schedulers exist at two levels [14, 18]. The first level is that of the local scheduler. At this level, decisions are made for each individual resource. The second level is that of the meta-scheduler. At the meta-scheduler level, several parallel applications are scheduled and undesirable interaction between these applications eliminated [15]. There have been many efforts with regard to meta-scheduling. The most notable of those efforts is the scheduler produced by the GrADS project. As part of this project, application characteristics have been used [3] to produce heuristically good schedules. Also efforts like [4] focus on the interplay between many parallel programs and have carried out simplified experiments to see how preemption of parallel programs affects job completion time. Performance characterization of data parallel programs at the second level has been done by Walker [17]. Our work would serve as the third level of this hierarchy. The meta-scheduler in the second level presumes that information about all resources is readily available and that there will not be any problem collecting it. However, this is not always realistic, and a means to getting around this assumption is required. Also, all of the existing work fits nicely into one “zone” (defined in Section 3.2.1). Thus our work represents a higher level of abstraction. Existing meta-scheduling work can be leveraged to do scheduling a smaller scope, e.g., within a single administrative domain.

In order to schedule a parallel program, two pieces of information are required: information about resources and information about the application. Considerable work has been done in obtaining information about an application and making it of manageable proportions. There have been many efforts to characterize an application’s structure by means of compile-time and run-time analysis of the program [2, 5, 6, 7]. Information about resources has been a subject of interest and there are many existing systems that collect this information. There are tools like NWS that monitor and report information about network traffic. The globus [1, 2] project has developed the MDS protocol to store dynamic information about resources (processors and network) and GIIS [1] to provide a query interface to locate resources of interest.

Schopf [12, 16] defines “superscheduling” to consist of three phases: resource discovery, resource mapping and job startup. Our mechanism for scheduling fits into the resource discovery and resource mapping phases. We are not concerned with job startup; we make our framework generic enough to use any job startup mechanism. Our technique is a means to identify potential resources quickly and securely, even when the number of potential resources is large.

3. System Architecture

3.1 General Principles of the Architecture

The crux of the multifaceted web services solution rests on the identification of the parameters to be exchanged amongst the servers and the ranking of their importance. The parameters give a summarized view of the information in the administrative influence of a peer web server. Effective decisions will depend upon the choice of parameters. Once these parameters are identified, it is important to be able to prioritize them and weight them accordingly in calculations. This weighting of parameters can either be done statically by the system administrator or dynamically determined by means of feedback from the system. Static weighting is acceptable because we assume that such weighting will be done by experts in the field for the

class of applications under consideration. In case such a weighting is not available for the application under consideration, dynamic methods need to be used. Some form of reinforcement learning [22] algorithm could be used to reinforce the more important parameters. However, accurate dynamic feedback is hard to obtain, as there are sources of noise in the results.

3.2 Case Study: Scheduling of SPMD Parallel Programs

We show the credibility of the principles described above by means of a prototype implementation for one area that can benefit from a multifaceted web service: a global scheduler on the grid. We use applications whose data-flow graphs contain highly connected sub-graphs [11] as our motivating family of applications. These applications have a master and slave task graph. Each slave is actually a team of processes which communicates intensively within the team but relatively infrequently across teams.

3.2.1 Zones and Administrative Domains

A zone is defined to be one area over which complete information is collected and processed to obtain parametric information. Normally an administrative domain would correspond to a single zone. However, there might be many zones within one administrative domain if the administrative domain has a large number of resources. Each zone has a web server running the scheduler. We call each of these schedulers a zonal scheduler (ZS). This can be visualized as in Figure 1. A zonal scheduler needs some information about peer schedulers as it might need to schedule some processes whose resource requests cannot be satisfied locally. The zonal schedulers find out about the existence of other zonal schedulers by using a discovery mechanism. Such discovery services exist in practice and they use protocols like UDDI and GRRP [1]. The use of such a mechanism extends the concept of virtual organizations [8] to dynamic virtual organizations. This means that virtual organizations can be formed on the fly as and when there is a match between demand and supply. This would eliminate the need for any human effort to establish collaborations between organizations.

Once a zonal scheduler finds other interesting zonal schedulers, it registers with each of them. Registering with a zonal scheduler makes a request for the zone's parameters to be passed periodically. Additional authorization control schemes and policy information might be provided at this stage. Authorization entails the zone communicating information about which certificate authorities it trusts. Thus the user will be able to verify if he has a certificate from one of those authorities. Policy information communicates the priority of external processes and whether reservations can or cannot be made.

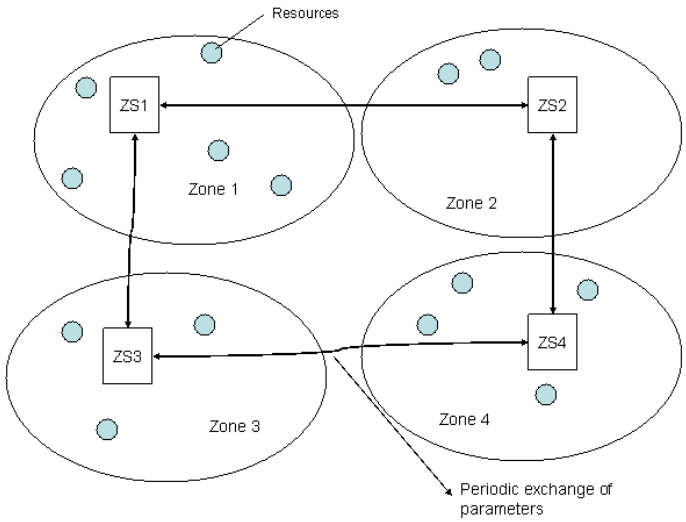

FIGURE 1: Partitioning of resources into zones

After the registration stage, communication takes places between the zonal schedulers to exchange certain parameters representing the state of resources in the zones. These parameters are sufficient to make informed decisions about whether that zone is fertile to be scheduled on. These parameters do not reveal any of the internal structure of resources in that zone. This is important to protect the privacy of information about resources within one zone or administrative domain.

The important question to answer is when and how frequently will the parameters be communicated. The mechanism used is event-driven. During setup, the parameters are communicated between zones. After this stage, re-communication takes place only when the state of the resources within a zone changes by over a threshold limit. If no such change takes place then a simple "keep alive" message is communicated to peer-zones interested in it.

3.2.2 Parameter Identification

The parameters that represent the state of a zone must meet the following criteria: they must be small, they must facilitate identifying the fertility of a zone and they must not divulge resource structure information. Based on parallel application characterization experience, we identified the following parameters.

- Distance Factor (DF): This gives an idea of how far the "target" ZS is from the "home" ZS. A home ZS is defined to be the zone in which the program and input data are present and to which the output data will go. If separated by a large network distance, i.e., high latency and low bandwidth, staging files and bringing program and input files to that zone will be costly. Another reason why such a factor is important is that tasks in parallel programs might be scheduled on different zones. Thus there will be some communication between zones, even though such a situation will be reduced as far as possible by the scheduling algorithm. For tightly coupled applications this may not always be possible and the scheduler might be forced to schedule them on different zones. This parameter will make zones between which there is large latency or low bandwidth less desirable to the zone selector. A high value of this factor makes a zone less desirable for scheduling.

  Distance Factor = Delay-Bandwidth product between home ZS and remote ZS

- Resource Density (RD): This parameter represents the intensity of computing power per unit communication bandwidth. The lower this value, the more bandwidth between every pair of nodes. This signifies that the resources in that zone are tightly coupled. For parallel programs that have a communicator in which a small group of processes communicate a lot, a zone with a low value of RD is important. For example, a SMP will have low RD whereas a network of workstations will have high RD. A similar parameter has been used to represent the computation to communication ratio in schedulers of parallel programs [19].

  Resource Density = $$\frac{\sum \mathrm{Pr}\,ocessorSpeed}{\sum CommunicationBandwidth}$$

- Load Factor (LF): This gives the overall load at some instant in that zone. This is important to take care of the computation component of the parallel program. Thus parallel processes that have a high computation aspect compared to communication would prefer a better value for LF than for RD.

  Load Factor = $$\sum_i \mathrm{Percentage\ Utilization\ of\ Processor_i}$$

These parameters would be number calculated from information about the state of resources in a particular zone.

Since XML is used in the transfer of zone state parameters, if a new parameter is identified, or a certain parameter cannot be revealed for security reasons, modifying the model and scheduler code is trivial since we can simply adapt the XML parser appropriately.

3.2.3 Identification of the Best Zone

Each zonal scheduler has a set of other viable zones with the state of resources in those zones. The ZS ranks the zones using the parameters obtained. One instance of a static weighting is described in Section 3.3. The internal structure of how such a zone selector will work is shown in Figure 2. Currently we use a statically weighted parameter set in determining the relative importance of a zone. This is suited for applications in which the structure of communication and computation is well known. Supplying this information would be done by experts in the field only once for the class of applications under consideration and hence would not be a tremendous overhead. Learning algorithms can be used to dynamically assign weights to parameters for cases in which the application structure is not known.

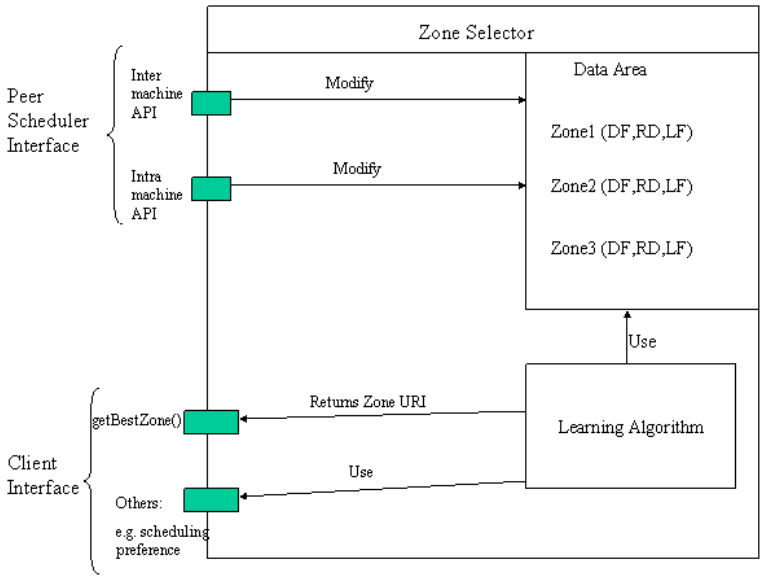

FIGURE 2: Internal Structure of the Zone Selector

3.2.4 Structure of the Global Scheduler Service

The scheduling takes place as follows. The client contacts the ZS on the zone in which it is present with a request to schedule a parallel program. XML is used as a means to communicate application characteristics to the ZS. The ZS does the scheduling and returns a mapping of processes to processors. This mapping is generic and represented in XML format. This XML can be translated at the client into any existing job submission language, e.g., ground RSL [1]. The overall structure of the global parametric scheduler is shown in Figure 3.

The characteristics of the application are important to the scheduler. There have been many efforts in the past to characterize the application's structure [2, 5, 6, 7, 24, 25]. For the purpose of the prototype global scheduler, we use a very simple model to characterize the parallel application. There are two components of interest in Single Program Multiple Data (SPMD) type of applications: computation of each process and communication between each pair of processes. Thus our definition of an application's characteristics consists of a vector

representing computation of each process and a two-dimensional matrix representing communication between every pair of processes. The scheduler uses the vector and the matrix to first select a zone that can provide the required resources and then to select machines within the selected zone that best satisfy the application requirements (through communication with remote schedulers). There have been lots of efforts to characterize an application based on computation. However, some recent efforts take into account communication patterns in addition to computation [9, 10, 11, 17]. We feel that in a wide-area environment both these factors will be important. Since the vector and matrix represent the relative proportions of the computation and communication, algorithms can use them to identify which of these is important. This would be used in appropriately weighting the various parameters.

Three separate components, each providing pieces of information that together form a cogent picture of the application's characteristics, will help in obtaining the vector and matrix data. First, the compiler [7] can extract certain information like the static task graph. The problem here is that at compile time not all information about branch direction and number of times loops are executed is known. Second, the run-time system [6] can actually profile the application and obtain the dynamic task graph. The problem here is that there might be noise on both processors and networks that make inferring information difficult. Third, and perhaps most important, is the programmer. A graphical tool is needed by means of which he or she specifies the structure of the application, if known. This graphical tool will give some indication as to what the application's structure is. The reason to include the programmer in this phase is that he knows the high level structure of the program. Using a combination of all 3 components will provide the best results. However, this implies that the application is of a parameter-sweep type where information about previous runs is available. Since run time information is not available for non-parameter-sweep type applications, we have to use programmer or compiler-supplied information.

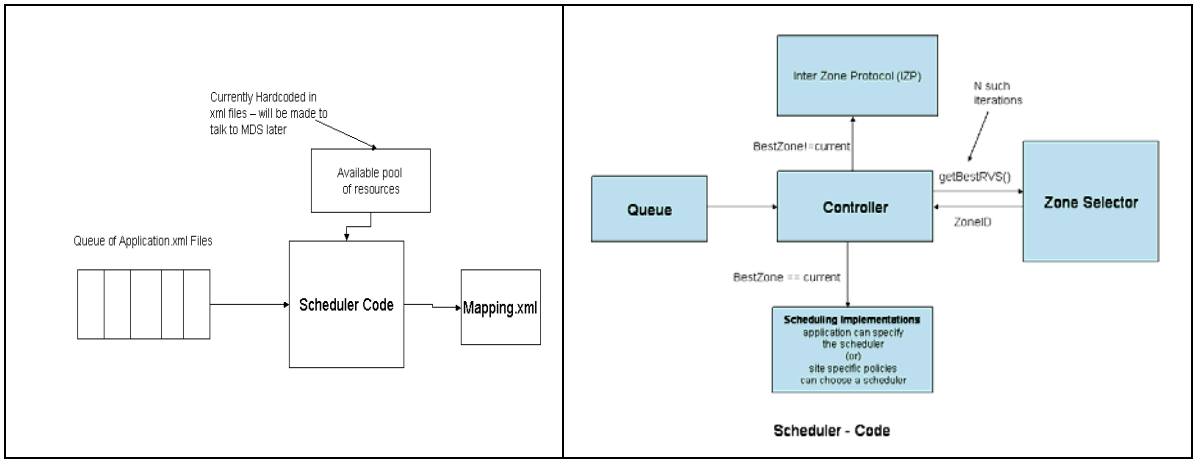

FIGURE 3: (left) Overall Structure of the Scheduler and (right) Internal Structure of the Scheduler

The scheduler does the scheduling as follows (see Figure 3). When an application request arrives, it is queued. The scheduler processes a single application's request at a time. The controller module contacts the Zone Selector module, which returns the best zone to schedule the next x processes. The number x represents one team or communicator in the parallel program that communicates a lot within itself. For the current application's requirements, the learning algorithm in the Zone Selector selects the best zone where the x processes could be scheduled. If the zone selected happens to be the current zone, the scheduler invokes some local scheduler and obtains a mapping of resources in the current zone. This can be done because the scheduler knows the details of resources in the current zone. If the zone selected is some other zone, the scheduler contacts the ZS of that zone through some inter zone protocol and requests scheduling of x processes on that zone. Since only the remote zone's scheduler makes the decision about which resources to schedule the processes on, complete security of each zone's resources is guaranteed. The other zone returns the mapping of processes to resources if it can

satisfy the request. This entire process is repeated till all the processes in the application are scheduled.

Once all the processes have been assigned physical resources, the current scheduler now reserves or locks those resources in order that it is guaranteed what it asked for. Deadlock is resolved by means of a mechanism shown in the next paragraph. If reservation is not present, a best effort scheduling is done, but no guarantees on job completion time can be obtained.

Each zonal scheduler has a unique Uniform Resource Identifier (URI). This is used in many places in the design of our system. First, it is used to publish the existence of a ZS in a directory to be used during discovery. Second, zonal schedulers use it to exchange parameters and inter zone protocol messages. Third, the URI is used to resolve deadlocks in scheduling. Since a URI is unique, deadlocks can be avoided by simple means of performing reservations in increasing order of sorted URIs. This is similar to the Siena event system [21] that uses a URI to uniquely identify event servers.

3.3 Steps to getting the SPMD program scheduled

1. Get application's structure from programmer/ compiler/ runtime system (in XML).
2. Send application structure to get it queued at the scheduler.
3. Scheduler selects best zone on which to run groups of processes of the application
    a. If best zone is current zone, run scheduling algorithm and obtain mapping of tasks to resources.
    b. If best zone is not current, contact peer scheduler in that zone and get the mapping in that zone.
    c. Repeat 3a and 3b for all groups of processes in the SPMD program.
4. Return mapping to client (in XML).
5. Client converts the mapping to RSL or any other job submission specific format.

Our most important contribution in this sequence of steps is the selection of the best zone. We illustrate this process by means of weighting parameters in one specific physics application "transport". This program is used to find the conductance of molecules in various orientations. This program uses MPI as its message-passing library. The application structure is a master-slave team structure, where each slave team performs large matrix operations using the SCALAPACK library. Thus, a slave team corresponds to a group which is communication intensive and there is relatively little communication amongst groups. At each iteration of step 3, the number of processes sent to the zone selector corresponds to the number of processes in the current slave team. Since each team internally communicates a lot, the resource density parameter is weighted the most. Solving sets of linear equations is computation intensive and hence the load factor is also given a high weight. This is an example of static weighting where we use the intuition of the programmer to weight parameters depending on application characteristics.

3.4 Performance Issues

The performance of a global scheduler is measured based on three metrics that have to be obtained from benchmark problems like the SCALAPACK application:

- Information Gathering time = $T_{end\_collection} - T_{start\_collection}$
- Scheduling time = $T_{mapping\_done} - T_{application\_arrival}$
- Job Completion time = $T_{output\_arrives} - T_{job\_startup}$

The best scheduling algorithm is one which minimizes total time, where

Total Time = Information Gathering Time + Σ(Scheduling Time$_i$ + Job Completion Time$_i$)

3.4.1 Information Gathering Time

Since we require no global collection of information about resources, the amount of information transferred is reduced. Information is also timely because the resources are closer to the local zone schedulers. As a result, more complete information or more frequent resource information updates are possible.

3.4.2 Scheduling Time

The time complexity of a scheduling algorithm is proportional to the number of resources and the number of tasks being scheduled, i.e.,

Scheduling Time = $\Omega(nk)$, where n = number of resources and k = number of tasks.

Because only one zone is considered at a time, the number of resources 'n' is always kept within bounds and since the input to the zone selector is a subset of processes that can be scheduled at each step, the number of application processes 'k' is also under control. The algorithm in the previous section is an instance of divide and conquer strategy. Thus the scheduling time for the zone based algorithm is,

Scheduling Time = $\Omega(n_1k_1) + \Omega(n_2k_2) + \ldots + \Omega(n_xk_x)$ + overhead of contacting other zones
<= $\Omega(nk)$

where,
n_i = number of resources in the i^{th} zone
k_i = number of application processes requested to be scheduled in the i^{th} zone

The overhead of contacting other zones is negligible compared to scheduling time. This overhead can be further reduced by sending out a request to more that one zone at a time. For example, the request could be sent out to the best three zones, and the zone that replies the fastest could be used.

The optimal value for number of resources to be present in a zone is to be determined by further experimentation. However, there might be problems with having an optimal number of resources within a zone due to administrative policies and management issues.

3.4.2 Job Completion Time

In order to measure this some form of feedback is required. Noiseless experiments are required to be able to measure job completion time accurately. This project is a work in progress and these experiments represent the next step that would be required to evaluate the approach suggested here.

4. Future Work

We have analyzed the model for scheduling parallel applications and have obtained parameters for collaboration of peer schedulers. The parameters identified in Section 3.2.2 are a preliminary version of the set of parameters that would be required to identify the state of resources in a zone. Further research is required to identify how other parameters like cost [13] and presence of requisite software libraries affect scheduling decisions. The effects of these and other parameters need to be further analyzed.

We have tried to make the framework for global scheduling as generic as possible. Different scheduling algorithms might require different representations of the application's characteristics. Using XML for representing this and defining an XML schema for it, allows the framework to accommodate schedulers requiring application characteristics in a different format.

As noted earlier, the underlying problem structure is found in many practical situations. We illustrate how this approach could be used in other problem domains. The key to the approach is in reducing large sets of information to a small number of parameters. These

parameters would then be used to select peer services to which work can be delegated. Dynamic software detection is one such situation. Suppose a user wants to use a specific piece of software to perform some task on a set of data. Where does such a search start? A "software detection service" could be used to have a complete index of such software and all searches could be routed there. A more elegant way of performing such a search might be to have parameters that indicate how likely it is to find software from specific companies running in that domain. Thus peer software detection servers would delegate such responsibility of finding the required software to other domains where the likelihood of that software being present is high.

We have striven to make our solution fit in with existing job startup mechanisms, e.g., DUROC. This may not be the best strategy, especially when scheduling applications for reservation-based execution in the future. In such a setting a just-in-time RSL generation should take place where a zone scheduler promises a certain number of resources and the actual binding to physical resources takes place only at execution time.

5. Conclusions

We strive to provide two improvements to scheduling on the grid. The first is to provide a scheduler that takes into account security of resources in a domain. This improvement is qualitative and no quantitative information can be given to prove it. This is an important issue and the concept of virtual organizations cannot be realized without tackling this problem. The second improvement is to provide for scalability of meta-schedulers. Even if information about all resources in the world could be accumulated in one place (a tremendous undertaking since the information has to be timely), the scheduling time would become large as scheduling algorithms run in time proportional to the number of resources and the number of tasks. In the case of a global grid and massive applications, both of these numbers are huge. Using zones, the number of resources and the number of tasks under examination are reduced, thereby reducing task scheduling time. We also believe that there are many similar situations where the principles applied to deriving a solution to global grid scheduling can be applied. The same principles can be applied effectively if a small set of parameters that represent the problem under question can be obtained.

6. References

[1] Czajkowski, K., Fitzgerald, S., Foster, I., Kesselman, C., (2001) Grid Information Services for Distributed Resource Sharing, *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing*, IEEE Press.

[2] Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S., (2002) SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems, *8th Workshop on Job Scheduling Strategies for Parallel Processing*.

[3] Dail, H., (2002) A Modular Framework for Adaptive Scheduling in Grid Application Development environments, *Masters Thesis, University of California, San Diego, Department of Computer Science*.

[4] Vaidhiyar, S. S., Dongarra, J. J., (2002) A Metascheduler for the Grid, *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11*.

[5] Chang, F., Karamcheti, V., Kedem, Z., (2000) Exploiting Application Tunability for Efficient, Predictable Resource Management in Parallel and Distributed Systems, *Journal of Parallel and Distributed Computing* 60, 1420-1445.

[6] Adve, V. S., (1993) Analyzing the Behavior and Performance of Parallel Programs, *Computer Sciences Technical Report #1201, University of Wisconsin-Madison*.

[7] Adve, V. S., Vernon, M. K., (2002) Parallel Program Performance Prediction Using Deterministic Task Graph Analysis, *ACM Transactions on Computer Systems.*

[8] Foster, I., Kesselman, C., Tuecke, S., (2001) The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *International Journal of Supercomputer Applications*, 15(3).

[9] Orduna, J. M., Arnau, V., Ruiz, A., Valero, R., (2000) On the Design of Task Communication-Aware Task Scheduling Strategies for Heterogeneous Systems, *Proceedings of the 2000 International Conference on Parallel Processing*.

[10] Arnau, V., Duato, J., (2000) Characterization of Communications Between Processes in Message-Passing Applications, *Proceedings of the IEEE International Conference on Cluster Computing*.
[11] Taura, K., Chien, A., (2000) A Heuristic Algorithm for Mapping Communicating Tasks on Heterogeneous Resources, *Heterogeneous Computing Workshop* 102-115.
[12] Schopf, J. M., (2001) Ten Actions When Superscheduling, *Scheduling Working Group, Scheduling Request For Comments*.
[13] Buyya, R., (2002) Economic Based Distributed Resource Management and Scheduling for Grid Computing, *PhD Thesis, Monash University*, Melbourne, Australia.
[14] Weissman, J. B., Grimshaw, A. S., (1996) A Federated Model for Scheduling in Wide Area Systems, *Proceedings of the IEEE High Performance Distributed Computing*.
[15] Raman, R., Linvy, M., Solomon, M., (1998) Resource Management through Multilateral Matchmaking, *Proceedings of the 7th IEEE Symposium on High Performance Distributed Computing*.
[16] Schopf, J. M., (2002) A General Architecture for Scheduling on the Grid, *Submitted to Special Issue of JPDC on Grid Computing*.
[17] Walker, M., (2001) A Framework for Effective Grid Scheduling of Data Parallel Applications in Grid Systems, *Masters Thesis, University of Virginia*.
[18] Krauter, K., Buyya, R., Maheshwaran, M., (2000) A Taxonomy and Survey of Grid Resource Management Systems, Technical Report 2000/80: Mannitoba University and Monash University.
[19] Chingchit, S., Kumar, M., Bhuyan, L.N., (1999) A Flexible Clustering and Scheduling Scheme for Efficient Parallel Computation, *13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*.
[20] Coyle, F. P., (2002) XML, Web Services and the Data Revolution, *Addison-Wesley Information Technology Series,Indianapolis*.
[21] Carzaniga, A., Rosenblum, D. S., Wolf, A. L., (1998) Design of a Scalable Event Notification Service: Interface and Architecture, *University of Colorado, Department of Computer Science, Technical Report CU-CS-863-98*.
[22] Sutton, R. S., Barto, A. G., (1998) Reinforcement Learning: An Introduction, *MIT Press, Cambridge, MA*.
[23] Tuecke, S., Czajkowski, K., Foster, I., Frey, J., Graham, S., Kesselman, C., (2002) Grid Service Specification draft 3, http://www.gridforum.org/ogsi-wg.
[24] Aggarwal, A., Chandra, A. K., Snir, M., (1990) Communication Complexity of PRAM, Theoretical Computer Science.
[25] Culler,D.E., Karp, R.M., Patterson, D.A., Sahay, A., Santos, E.E., Schauser, K.E., Subramonian, R., von Eicken, T., (1996) LogP: A Practical Model of Parallel Computation, Communications of the Association for Computing Machinery.