Advance Reservation-based Computational Resource Brokering using Earliest Start Time Estimation

Feng Liang, Shilong Ma
National Laboratory of Software Development Environment, Beihang University, China
Email: {liangfeng, slma}@nlsde.buaa.edu.cn

André Luckow and Bettina Schnor
Institute of Computer Science, University of Potsdam, Germany
Email: {luckow, schnor}@cs.uni-potsdam.de

***Abstract*—As a potential method for improving the resource utility, advance reservations is crucial for brokering computational resource in a predictable and efficient way. However, Conducting Advance Reservation efficiently in grid environment is not easy due to the insufficient support of the current grid architecture and the dynamic feature of grid resources. Based on a grid middleware Migol, this paper presents and evaluates a job broker architecture that addresses this deficit and provides support for advance reservation on grid-level. Using Earliest Start Time Estimation algorithm (ESE) to predict queuing delays at grid resources, this architecture is able to conduct advance reservation at lower cost and higher accuracy. Experiments show that this brokering architecture is scalable and the ESE algorithm is efficient in use.**

***Index Terms*—Grid Resource Brokering, Earliest Start Time Estimation, Advance Reservation, Local Jobs Preferred principle**

I. INTRODUCTION

Many applications have demanding resource requirements and usage modes that are not addressed by current grid resource management systems, which commonly only provide best effort guarantees. For example, deadline driven applications, such as severe weather and hurricane simulations, demand guaranteed completion times. Advance reservations are a common mechanism to provide such guarantees. An advance reservation is a contract between the resource owner and consumer that commits a certain resource for a defined time to the resource consumer. With advance reservations users can obtain execution guarantees from local resource managers without requiring detailed knowledge of current and future workloads or of the resource owner's policies.

Another interesting use case for advance reservations is applications that require simultaneous access to multiple resources within a Grid [1] or even cross different Grids [2]. Chero [3] – an application that analyzes biological networks – can e. g. greatly benefit from the availability of many distributed resources. The more resources are available, the larger networks can be computed and the lower the time-to-completion is. Nevertheless, in most cases it is necessary to at least partially co-allocate resources belonging to different sites. Advance reservations ensure that the requested resources are available at a specified time and are commonly used to implement co-allocation.

While many modern Local Resource Management Systems (LRMSs) support advance reservations on cluster-level, these capabilities are usually not accessible via grid platforms, such as Globus [4] or UNICORE [5]. Another concern for a grid brokering system is the selection of a suitable resource. In general, queuing times at the LRMS significantly contribute to the overall time-to-completion of grid jobs. Thus, it is often desirable to predict the start time of a grid job at the available resources. Based on a start time estimation, grid brokers can plan the placement of jobs and the execution of multiple, possibly dependent jobs. Unfortunately, common LRMSs do not expose queuing delays to grid users.

This paper proposes an architecture that addresses these deficits and provides support for advance reservation in grids. The Advance Reservation Service (ARS) offers a unified, WSRF-conform reservation service on top of different local resource managers, i. e. PBSPro, Torque/Maui and LSF. A core part of the ARS is the Earliest Start Time Estimator (ESE) – a queuing delay predictor. The ESE algorithm estimates job start times based on the current utilization of a grid resource. ESE has been designed to work independently of the underlying LRMS. The ARS is part of the Migol system [6]. Migol addresses the fault tolerance of long-running and computation-intensive grid applications. A main task of Migol is the automatic allocation of resources.

This paper is structured as follows: We begin with a discussion of related work in section II. An overview of the Migol brokering services and the presentation of the new Earliest Start Time Estimation algorithm are given in section III. Section IV presents experimental results that evaluate the scaling and effectiveness of the chosen architecture and algorithms. In section V we summarize

© 2012 ACADEMY PUBLISHER
doi:10.4304/jcp.7.6.1329-1336

the contributions of this paper and present some areas of future work.

II. Related Work

Advance reservation strategies and systems have been under intensive research for some time. We will discuss the most relevant papers in the following.

A. Advance Reservation Algorithms

[7] investigates the influence of resource heterogeneity on reservation-aware schedulers. The presented algorithm uses computational geometry to manage advance reservations. The simulation results show that resource heterogeneity may have a positive impact on the performance if it is taken into account in the design of the scheduling algorithm. The proposed data structures ensure that algorithm can scale to a large number of systems. We show that the real-world system proposed by this work provides a similar scalability. The JBS scheduling algorithm performs very well even in large grid systems that are loaded with hundreds of jobs.

[8] proposes the what-if algorithm for placing reservations. For each reservation request, the algorithm inserts a placeholder job into the schedule of the LRMS and simulates the processing of this schedule to determine the time when the job would be started. In contrast to our approach non-reservation jobs can be moveable, i. e. they are often postponed. This is a violation of the fairness principle and decreases the acceptance of grids in our opinion. The quality of the what-if algorithm also depends on the correctness of the estimated execution times, but this is not investigated.

B. Advance Reservation Based Scheduling Systems

[9] proposes a standards-based resource brokering architecture. The system requires the deployment of a WS-Agreement service on each of the used resources. Further, Maui is the only supported LRMS. In contrast to the JBS/ARS, the service does not utilize a start time estimator and solely delegates the reservation to the LRMS. Since most LRMSs prioritize reservations higher than normal jobs, this approach again violates the fairness principle.

[10] introduces a reservation-based scheduling system for workflows as part of the ASKALON framework. The reservation framework provides a WS-Agreement based negotiation mechanism, which supports the agreement process between the scheduler and the resource managers. The system also provides rudimentary support for fairshare by supporting the configuration of a maximum node limit per user. Fairshare on LRMS level is not considered. Further, the system does not take into account jobs and reservations submitted via other mechanisms, e. g. via the LRMS or another grid scheduler, which can cause inaccuracies in the proposed algorithm in real-world settings.

The GridARS middleware [11] provides an advance reservation WSRF service. The service utilizes a two-phase-commit style protocol for the management of reservations. The user must explicitly specify a start time for the reservation and commit it after it has been accepted. Otherwise the resources are released again.

[12] proposes an advance reservation manager for heterogeneous grid environments. Again, the described architecture has some flaws: the service assumes that all jobs and reservations are routed through the reservation service. Also, the service does not enforce LJP between local jobs and reservation-based grid jobs.

C. Job Finish Time Estimation

As the user concerns mainly the job's finish time in brokering, so it is essential to estimate the job's finish time on each resource for selection. A job's finish time within Grid mainly depends on transferring time, running time and queuing time, and many researchs has been done to predict each of these time period.

QBETS [13] is a batch queuing delay predictor service based on the Network Weather Service. It uses job history data to predict job start times within a user-defined confidence interval. However, QBETS has a main limitation: It requires users to specify a queue for prediction. Often, grid users and brokers do not have detailed information about local resources, which makes it difficult to utilize QBETS. Also, some LRMSs, e. g. PBSPro, implement advance reservations using dynamically created queues. The prediction mechanism can be subject to certain inaccuracies in these cases. Further, QBETS is only available on TeraGrid resources.

[14] utilizes different machine learning techniques for predicting queue wait times. For this purpose, different algorithms for deriving future wait times from historical data are presented. In contrast to the work presented in this paper, the algorithms are solely validated using simulations and have not been integrated into a real world infrastructure.

Based on grid workload traces, [15] presents the results of different data mining methods in predicting the job runtime and the queue wait time within Grids. The simulation results show that job runtime prediction accuracy is on average low and the absolute prediction error is high. Besides, no dominant prediction methods are observed on accuracy, the most efficient prediction methods may differ from site to site due to different user behaviors and different job and system characteristics. The simulation results also show that the queue wait time prediction accuracy can be relatively high when using BMBP combined with workloads in good patterns, but the simulation experiments require that all LRMSs implements only FCFS policy without backfilling , which is hard to fulfill in production Grid environments.

III. Migol Brokering Architecture

Migol is a grid middleware that guarantees the correct and reliable execution of grid applications even in the presence of failures. The architecture of Migol follows the Open Grid Services Architecture (OGSA) vision of a Service-Oriented grid comprised of a set of loosely coupled services. Services are modular components that expose the capabilities of a resource via a standardized WSRF interface. The framework is based on the Globus

© 2012 ACADEMY PUBLISHER

Toolkit 4. Existing services, components and libraries, especially those of the underlying Globus framework are re-used if possible.

Figure 1 gives an overview of the Migol job brokering architecture. In our earlier work [16], we presented the Job Broker Service (JBS) and an initial version of the Advance Reservation Service (ARS). The ARS offers a unified, WSRF-conform reservation service on top of different local resource managers, i. e. PBSPro, Torque/Maui and LSF. A core part of the ARS is the Earliest Start Time Estimator (ESE) – a queuing delay predictor. The ESE algorithm is able to estimate job start times based on the current utilization of a grid resource. The JBS is a meta-scheduling service that is built on top of the ARS. The service supports among other things the automatic discovery of resources via the Globus Monitoring and Discovery Service (MDS) and different scheduling plugins. The reservation-aware scheduler of the JBS utilizes the ARS and the ESE algorithm for resource selection. In the following we describe the ARS and the ESE algorithm in detail.

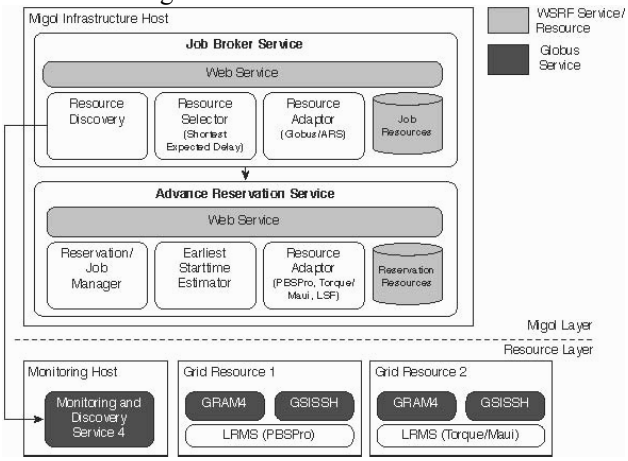


Figure 1: Migol Brokering Architecture

A.Job Broker Service

The JBS offers a unified access to resources managed by different grid and cluster LRMSs. Initially the JBS discovers available resources by querying the Globus Monitoring and Discovery Service (MDS). All obtained resources are matched against the application requirements. The JBS then uses a simple metric – the Shortest Expected Delay (SED) [17] – to pre-rank all resources, taking into account the transfer times for all input files, the queuing time (see next section) and the estimated run time of the application. Each factor of this metric is discussed in the following.

The runtime of an application is estimated based on a reference runtime run_{ref} and clock rate CPU_{ref} that is specified by the user in the application's resource description. In the first step, for each suitable machine M_i a speed factor α_i is calculated by the JBS scheduler:

$$\alpha_i = \frac{CPU_{ref}}{CPU_i} \tag{1}$$

In case of a heterogeneous machine, i. e. a machine consisting of CPUs with different speeds, the slowest clock rate is used. Since the slowest machine usually determines the time-to-solution of a parallel application, in particular for tightly coupled and strongly synchronized applications, this is a valid assumption.

$$\overline{\alpha} = \max(\alpha_1, \alpha_2, ..., \alpha_i) \tag{2}$$

The speed factor $\overline{\alpha}$ is now used to scale the runtime:

$$run_i = \overline{\alpha} \cdot run_{ref} \tag{3}$$

That means that if a resource is selected that is slower than the reference resource, a longer runtime is considered in the estimation. Currently, the CPU clock rate is the only information available in the Globus MDS which allows meaningful runtime estimations.

Another important factor in the SED metric is the waiting time at the LRMS. The waiting time for a job on a machine M_i depends on the current utilization of this machine. The following assumptions are made for the estimation of the waiting time: Each machine M_i provides a certain number of processor cores $available\_cores_i$. The LRMS manages these cores using a single FIFO queue. Each job is subject to a maximum runtime $walltime_i$. The waiting time for Machine M_i can then be approximated based on the resource requests, i. e. the number of cores requested by a job k ($requested\_cores_{J_k}$), of all waiting and running jobs ($J_1, J_2, \ldots, J_j$):

$$wait_i = \max\_walltime_i \cdot \left\lfloor \frac{\sum_{k=1}^{n} requested\_cores_{J_k}}{available\_cores_i} \right\rfloor \tag{4}$$

This simple formula provides only an approximation for the waiting time since it neglects possible resource fragmentations and complex LRMS configurations, e. g. with multiple queues or other intransparent allocation policies deployed by the resource owner. For example, in the case of fragmentations longer waiting times can occur since the LRMS is not able to utilize all cores at all times.

The algorithm also considers potential data transfer times for input files, e. g. application checkpoints. If network data is available, the transfer time for all n input files is computed based on the $bandwidth_i$, i. e., the available bandwidth between the file location and machine M_i

$$transfer_i = \frac{\sum_{k=1}^{n} file\_size_k}{bandwidth_i} \tag{5}$$

To estimate the overall delay for a machine M_i the transfer, waiting and runtime for each machine are aggregated:

$$delay_i = transfer_i + wait_i + run_i \tag{6}$$

Because of the dynamic nature of grids, the used metric is only a rough approximation. Grid brokers must be able to deal with uncertainty – no broker will be able to obtain all global information necessary to make a

© 2012 ACADEMY PUBLISHER

precise forecast. As described, the waiting time of a job is, due to insufficient information and intransparent allocation principles of the LRMSs, hardly predictable. Further, it is impossible to forecast the runtime of an application solely based on the CPU clock rate. Thus, the resulting list is solely considered as pre-qualification and called candidate list.

If the grid infrastructure supports advance reservation, the candidate list is used to send reservation requests to at most top three sites using the ARS. The ARS attempts to place the job in the desired time and returns for each successful reservation request a reservation id and the earliest start time committed by the LRMS. Using the returned start times, the earliest finish time of each resource is updated.

$$earlist\_finishtime(M_i) = earlist\_starttime_i + delay_i \quad (7)$$

For the top resource the JBS keeps the reservation. All other reservations are canceled. Finally, the job is submitted through the ARS using the assigned reservation ID. Details of the ARS are discussed in next section.

In the case of best effort jobs the JBS uses a very simple but practical approach to minimize the waiting time. The job is dispatched to the top three sites of the candidate list. This technique is referred to as Resource Manager Selective Flooding (RMSF). As soon as one of these jobs gets active, all other jobs are canceled. This approach avoids orphan jobs and ensures that the overhead caused by an allocation of more than necessary resources is minimized. However, there are some disadvantages: For example, since the LRMS decides when a job is started, no commitments with respect to a certain start time or deadline can be given. Thus, it is not possible to plan e. g. the staging of large files. Therefore, the JBS relies on advance reservation whenever possible.

B. Advance Reservation Service

The ARS provides a unified interface for conducting reservations at different LRMSs: PBSPro, LSF and Maui. The LRMS specific code is encapsulated in a plugin and can therefore easily be extended. The ARS allows the user to manage the lifecycle of reservations and reservation-based jobs, i. e. it supports the creation, monitoring and destroying of reservations as well as the binding of jobs to reservations and the management of these jobs.

For each user request a LRMS specific shell script is generated conducting the respective operations at the LRMS, e. g. by calling pbs rsub to place a PBSPro reservation. The script is then executed on the specified resource using the GRAM service or GSISSH (depending on which service is available on the respective resource). This way the ARS is able to access the reservation capabilities of any supported LRMS via a standard grid protocol. Both GSISSH as well as the GRAM service ensure that all LRMS requests are securely executed using the privileges of the respective grid user.

To place a reservation several steps are conducted. First, the type of the remote LRMS, i. e. PBS, LSF or MAUI, is determined and information about the current resource utilization is gathered. Next, the earliest start time estimator (ESE) is used to obtain a start time for a reservation. Finally, the reservation is committed at the LRMS and a handle to the reservation is returned to the user.

Since the GRAM is not aware of reservations, it is not possible to bind GRAM jobs to a reservation. Thus, the ARS is also used to bind jobs to a reservation and to manage the execution of jobs. For this purpose, the same principle as for reservation requests is used. The responsible ARS plugin generates for each job request a shell script, which binds the job to the specified reservation and executes the script using the GRAM/GSISSH. This technique is also applied for job status queries or cancellations.

C. Earliest Start Time Estimation

The ARS uses the Earliest Start Time Estimation (ESE) module to estimate the start time for a job based on the current resource utilization taking into account: all currently running jobs, all currently waiting jobs and all committed reservations.

The required information is collected via different resource dependent collector scripts. Currently, Migol supports PBSPro, LSF and Maui. Using this data the ARS builds a schedule for the respective resource and attempts to place the requested reservation within this schedule.

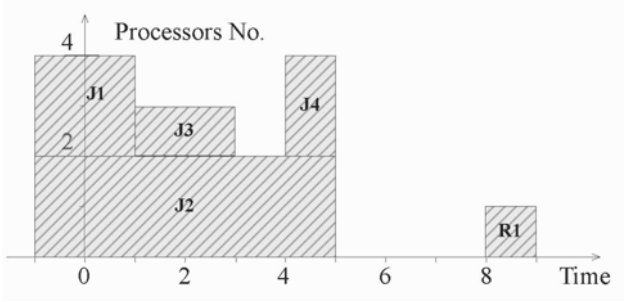
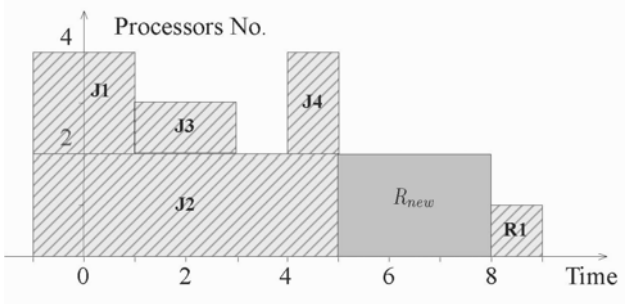


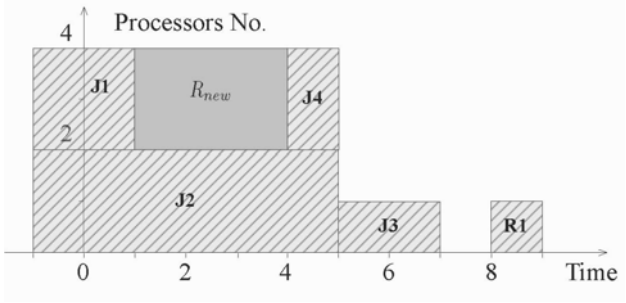
Figure 2: Cluster Resource Utilization

Different LRMSs such as PBSPro prioritize reservations higher than batch jobs. This is often the cause why local administrators forbid advance reservation. The ARS in contrast ensures the LJP principle and enforces that no local job is postponed by a reservation. By default, LJP is used; i. e. new reservations are placed into the schedule following the first-in-first-out principle without re-prioritizing or moving any previously queued job. Figure 2 shows e. g. the utilization of a cluster system with 4 processors using a Gantt chart for visualization purposes. The regions named Vi are idle slots. In Figure 3a) Rnew is placed after J1, J2, J3 and J4, while in the non-LJP case depicted in Figure 3b) the LRMS postpones J3 and places Rnew right after the job J1.

© 2012 ACADEMY PUBLISHER



a) With Local Job Preferred



b) Without Local Job Preferred

Figure 3: Local Job Preferred based Reservation

The ESE algorithm expects as input a reservation request, which consists of a ready time, i.e. the time when the job is ready to be started on the resource considering e. g. the necessary file transfer times, job duration and a deadline. Further, different resource information is required: the overall number of available processors and a list of all currently queued and running reservations and jobs. This data is obtained by the described information collector script. As output the algorithm returns the earliest possible start time for the reservation.

To efficiently support a query for the earliest start time, a vacancyList, which stores all idle time slots, is used. Each element of the vacancyList is referred to as vacancyElement. Initially, the vacancyList contains a single vacant slot of infinite length and an available resource count set to the overall number of processors. For each running and queued job as well as for each committed reservation the vacancyList is modified to reflect the respective job/reservation. For this purpose a worst case approximation for the duration is used, i. e. for all queued jobs the walltime and for all running jobs the remaining walltime is used.

Figure 4 illustrates the ADD_QJOB procedure, which is used to adjust the vacancyList for each queued job. Similar functions are used for adding running jobs and reservations. The algorithm simply iterates over the vacancyList. If a vacant slot with a suitable number of processors and duration is found, the procedure removes the respective vacancy element V from the list and splits it into V' and V''. Both elements are then re-added to the vacancyList. After processing all jobs and reservations the vacancyList of the schedule depicted in Figure 3 contains e. g. the elements V_1 to V_5.

Listing 1 ADD_QJOB procedure

```
procedure ADD_QJOB(vacancyList, job)
    for each vac in vacancyList do
        if job.duration ≤ vac.duration and job.numberProcessors ≤
vac.availProcessors then
                                    ▷ Adjust vacancy list to accommodate new job
            remove V from vacancyList
            split V into V' and V''
            add V', V'' to vacancyList
            return V.startTime
        end if
    end for
end procedure
```

Figure 4: ADD_QJOB procedure.

In the final step the requested reservation R_{new} is processed. For this purpose, the procedure iterates over all vacancies in the vacancyList structure to search for a suitable slot. As is shown in Figure 4a), V_3 represents a suitable slot for R_{new}. Hence, V_3 is split into the new vacancy elements V_3' and V_3''. Since V_3'' has a duration of 0 it is dropped. The start time of V_3 is then returned to the user.

The algorithm has a linear time complexity of $O(N)$, where N is the number of running and queued jobs and reservations. While the ARS relies on up-to-date information, the estimation can only be considered as a rough approximation. However, the experiments presented in the next section will demonstrate that ESE is in particular for meta-scheduling well-suited.

IV. Experimental Results

To evaluate the performance and overhead of the Migol job management services, several experiments have been conducted. In the first experiment we analyze the submission time of the services GSISSH, the GRAM, the ARS, and the JBS. The second experiment evaluates the accuracy of the earliest start time estimation module.

A. Experiment Testbed

The testbed comprises of two parts: two dedicated machines (Flotta and Stronsay) on which the Migol services are deployed and different cluster resources. Flotta is equipped with a 1.66 GHz AMD processor and 1GB RAM, Stronsay with a Intel Core 2 Duo 2.66 CPU and 2GB memory. Both are running a Debian Linux with 2.6.18 Kernel. The services were deployed in a Globus Toolkit 4.0.5 container running on a Tomcat 5.5.23 with Java 5 Update 14.

As compute resources the Siemens and Highland clusters at the Potsdam University are used. Siemens consists of 2 front and 10 compute nodes managed by PBSPro. Each node provides a 2.6 GHz Intel Xeon processor and 1GB of memory (resp. 2GB on the front nodes). Highland is managed by Torque 2.3.0 and Maui 3.2.6. It offers 10 compute nodes and a single front node. All nodes are equipped with an AMD Opteron 1.8 GHz dual core processor.

Our test application is a real-world application called Chero [3]. Chero is a simple master-worker MPI application written in C++, which is used to study the robustness of biological networks. For this purpose Chero relies on Dijkstra's shortest path algorithm, which is used

© 2012 ACADEMY PUBLISHER

to evaluate the impact of each node on the overall biological network. Since this work can be well partitioned into independent sub-tasks – each task computes the robustness with a different missing node – the application is very scalable and suited for grid environments.

B. Job Submission Times

In this experiment, the submission times and overheads of the Globus GRAM4, GSISSH, the ARS, and the JBS are analyzed. As described in section 3, a JBS submission typically involves several calls to the ARS and its ESE module, which again requires multiple remote calls to the respective resource. In the following we carefully analyze the performance and overhead of each layer.

For this experiment a basic workload environment consisting of a random number of jobs and reservations (between 1 and 10) is created. Having created the environment we repeatedly (at least 10 times) submitted the Chero application using the different services, i. e. GSISSH, the GRAM4, the JBS and the ARS. Figure 5 shows the mean submission times measured during this experiment.

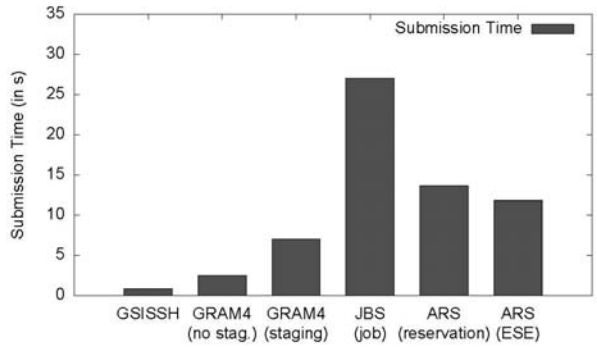


Figure 5: GRAM 4, GSISSH, ARS and JBS performance

A GRAM4 job submission including the staging of a graph file with the size of 72 kb using the Reliable File Transfer (RFT) service required about 7.5 sec. In comparison, the execution of a GSISSH task requires only about 1 sec, but does not involve any interaction with the LRMS.

Since the ARS requires the direct interaction with the LRMS, the most efficient way is the utilization of GSISSH for executing LRMS scripts. The ARS overhead of about 13.7 sec is mainly caused by two factors: the Web service and security overhead (authentication, encryption, credential delegation) associated with the remote call and the time required for conducting the start time estimation. For each reservation request, several tasks must be executed: Resource information, i. e. the current active jobs and reservations must be collected and the reservation respectively job request must be executed.

The JBS introduces additional overhead. For each JBS job multiple ARS invocations must be conducted: First, multiple reservations must be conducted to obtain an earliest possible time at different resources. As previously discussed, an ARS reservation requires in average 13.7 sec. Secondly, the job must be submitted to the remote site. Thus, the measured JBS overhead is reasonable taking into account the additional Web service calls and site ranking tasks that have to be conducted by the JBS.

C. Scaling of ESE Algorithm and JBS

As described in section 3.3, the performance of the earliest start time estimation algorithm depends on the current resource utilization, i. e. the number of active jobs and reservations at a resource. In Figure 6 we analyze the correlation between the resource utilization and the JBS response time. Obviously, the JBS submission time increases with the number of currently active reservations. But with an increase of the number of active reservations by a factor of four, the submission time only doubled. This reflects that the JBS submission time is dominated by the Web service call overhead and not by the performance of the ESE algorithm (even in the presence of a large number of 700 reservations). Since the number of sites that is used for negotiating advance reservations is bound to three, it can be expected that the JBS scales well also in larger grid environments.

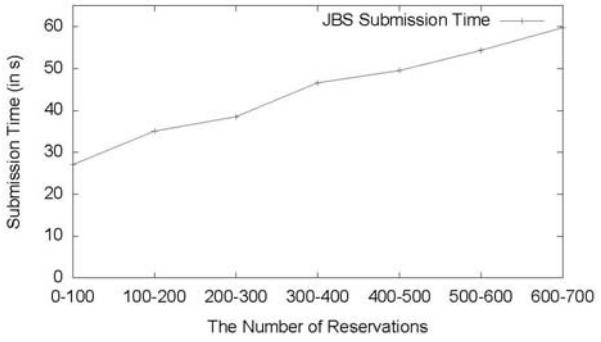


Figure 6: JBS Job Submission under Different Resource Utilizations

D. Evaluation of the ESE Algorithm

The start time estimation algorithm depends on the specified job walltime, which is used as worst-case approximation for the run time of active jobs. The quality of the specified walltime depends on the user. However, users commonly overestimate walltimes [18]. Further, failures can lead to early terminations. In the following, we investigate the impact of the specified walltime on the accuracy of ESE. For this purpose, we compare the real start time of a job with the estimated start time using different load scenarios.

Based on the real start time rt and the estimated start time et, the accuracy ac is computed as follows:

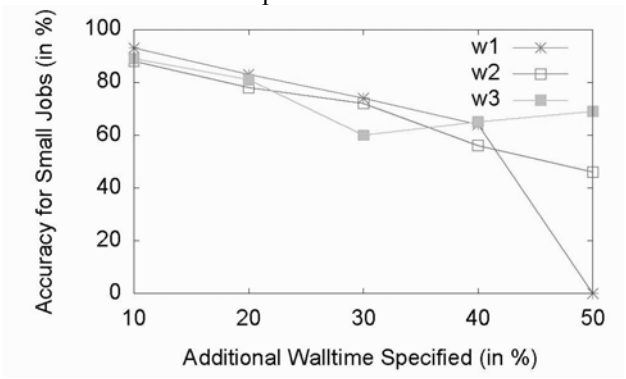
$$ac = \begin{cases} 1 - \dfrac{|et - rt|}{rt} & if\ |et - rt| <= rt \\ 0 & otherwise \end{cases} \quad (8)$$

For deviations larger than rt the accuracy is set to 0. This experiment was conducted for all 3 workloads W1, W2, and W3 using different walltime deviations for the background jobs: The walltime of these jobs is set from 110% up to 150% of the actual runtime, which reflects a moderate overestimation.

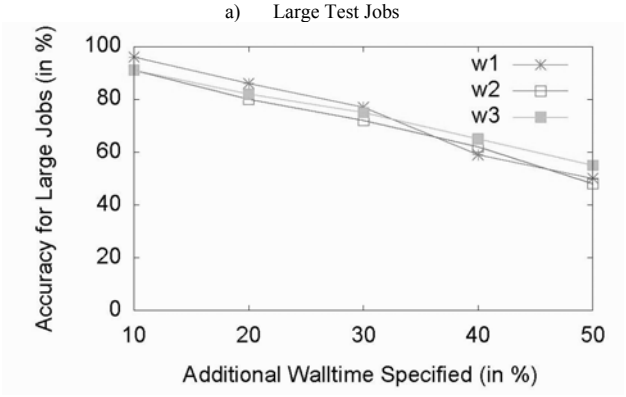
Figure 7 summarizes the results of this experiment. Obviously, the accuracy of the estimation decreases with the quality of the walltime specification. The more the

© 2012 ACADEMY PUBLISHER

walltime differs from the actual runtime, the less accurate is the estimation. Figure 7 b) also shows the effect of backfilling. Workload W1 contains some longer jobs that are likely to fragment the cluster system. This gives the LRMS the opportunity to backfill smaller jobs, which leads to an increased unpredictability in particular if jobs terminate earlier than expected.



a) Large Test Jobs



b) Small Test Jobs

Figure 7: Estimation Accuracy

Since the main use case of the ESE algorithm is the support of the resource ranking mechanism of the JBS, we evaluate the rankings produced by ESE in the following experiment. For this purpose, we compare two clusters (Einstein and Highland) with different background workloads (W1/W2, W2/W3, W1/W3). We use different walltime specification errors for the background workload to set the walltime values between 110% to 150% of the actual job runtime. Again, two test jobs are utilized – a small and a large job.

V. Conclusion and Future Work

Using advance reservations the predictability of application runs can be significantly improved. While many local resource management systems provide support for reservations, these capabilities are not supported by current grid systems. In this paper, we introduce Migol's Advance Reservation Service (ARS) and the Earliest Start Time Estimation (ESE) algorithm utilized to obtain queue time predictions. The ARS is designed as extensible, WSRF-conform grid service. New resources adaptors and selectors can be provided without modifying the core of the ARS. Resources are accessed via the standard WSRF interfaces provided by Globus. In general, it is not necessary to deploy custom code on the used grid resources.

Our experiments demonstrate that the overhead exposed by our services is acceptable and that the system scales under load. The start time estimations produced by the ARS depend on the accuracy of the walltime specification given by the user. Our experiments show that even in the case that the deviation between the walltime and the actual runtime is 50 %, the JBS is able to make the right placement decisions for more than 60 % of all submitted jobs. In total, the rankings produced by ESE were correct in 75% of all cases.

In the future we are going to adapt our work to more recently emerging standards, such as WS-Agreement [19], GLUE [20] and the OGSA Basic Execution Service [21].

References

[1] Chuan-Wen Chiang. Applying An Improved Fast Ant System To DAG Scheduling For Computational Grids. Journal of The Chinese Institute of Engineers, 2008, 31(7):1113–1125.

[2] Chao-Tung Yang,Wen-Jen Hu, and Kuan-Chou Lai. Implementation of Cross Grid Information Services for Resource Broker on Computational Multi-Grid Environments. Journal of The Chinese Institute of Engineers, 2009, 32(7, Sp. Iss. SI):975–984, November 2009. 9th International Conference on Algorithms and Architectures for Parallel Processing,.

[3] Potsdam University Operating Systems and Distributed System Group. Chero homepage, 2010. http://www.cs.uni-potsdam.de/bs/chero/.

[4] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. In IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pages 2–13, 2005.

[5] Valentina Huber. Unicore: A grid computing environment for distributed and parallel computing. In PaCT '01: Proceedings of the 6th International Conference on Parallel Computing Technologies, pages 258–265, London, UK, 2001. Springer.

[6] Andre Luckow and Bettina Schnor. Migol: A Fault-Tolerant Service Framework for MPI Applications in the Grid. Future Generation Computer Systems– The International Journal of Grid Computing, 24(2):142–152, 2008.

[7] Claris Castillo, George N. Rouskas, and Khaled Harfoush. Efficient resource management using advance reservations for heterogeneous grids. In Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium, pages 1–12, April 2008.

[8] Thomas Roeblitz and Krzysztof Rzadca. On the placement of reservations into job schedules. InWolfgang E. Nagel,Wolfgang V.Walter, andWolfgangLehner, editors, Euro-Par 2006 Proceedings, volume 4128 of Lecture Notes in Computer Science, pages 198–210. Springer, 2006.

[9] Erik Elmroth and Johan Tordsson. A standards-based grid resource brokering service supporting advance reservations, coallocation, and cross-grid interoperability. Concurrency and Computation: Practice and Experience,21(18):2298–2335, 2009.

[10] Marek Wieczorek, Mumtaz Siddiqui, Alex Villazon, Radu Prodan, and Thomas Fahringer. Applying advance reservation to increase predictability of workflow execution on the grid. In E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science

© 2012 ACADEMY PUBLISHER

and Grid Computing, pages 82–90, Amsterdam, Netherlands, 2006. IEEE Computer Society.
[11] Atsuko Takefusa, Hidemoto Nakada, Tomohiro Kudoh, Yoshio Tanaka, and Satoshi Sekiguchi. GridARS: An advance reservation-based grid coallocation framework for distributed computing and network resources. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, Job Scheduling Strategies for Parallel Processing, 13th International Workshop, JSSPP 2007, volume 4942 of Lecture Notes in Computer Science, pages 152–168, Seattle, WA, USA, 2007. Springer.
[12] Changtao Qu. A grid advance reservation framework for co-allocation and co-reservation across heterogeneous local resource management systems. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Wasniewski, editors, Parallel Processing and Applied Mathematics, 7th International Conference, PPAM 2007, volume 4967, pages 770–779, Gdansk, Poland, 2007. Springer.
[13] Daniel Charles Nurmi, John Brevik, and RichWolski. QBETS: queue bounds estimation from time series. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, Job Scheduling Strategies for Parallel Processing, 13th International Workshop, JSSPP 2007, volume 4942 of Lecture Notes in Computer Science, pages 76–101, New York, NY, USA, 2007. Springer.
[14] Hui Li, Juan Chen, Ying Tao, David Gro, and LexWolters. Improving a local learning technique for queuewait time predictions. Cluster Computing and the Grid, IEEE International Symposium on, 0:335–342, 2006.
[15] Ozan Sonmez, Nezih Yigitbasi, Alexandru Iosup, and Dick Epema. Tracebased evaluation of job runtime and queue wait time predictions in grids. In Proceedings of the 18th ACM international symposium on High performance distributed computing, HPDC '09, pages 111–120, New York, NY, USA, 2009. ACM.
[16] Janin Jeske, Andre Luckow, and Bettina Schnor. Reservation-based resourcebrokering for grid computing. In Proceedings of German E-Science Conference 2007, pages 1–10, Baden-Baden, Germany, 2007.
[17] Bettina Schnor and M. Gehrke. Dynamic-sed for load balancing of parallel applications in heterogeneous systems. In Hamid R. Arabnia, editor, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pages 442–449, Las Vegas, Nevada, USA, July 1997. CSREA Press.
[18] Georg Birkenheuer, Andr´e Brinkmann, and Holger Karl. The gain of overbooking. In Eitan Frachtenberg and Uwe Schwiegelshohn, editors, JSSPP, volume 5798 of Lecture Notes in Computer Science, pages 80–100. Springer,2009.
[19] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Open Grid Forum Document GFD.107, 2007.
[20] S. Andreozzi, F. Ehm, L. Field, and B. K´onya. GLUE Specification. ogf.org/documents/GFD.147.pdf, 2009.
[21] I. Foster, A. Grimshaw, P. Lane,W. Lee, M. Morgan, S. Newhouse, S. Pickles,D. Pulsipher, C. Smith, and M. Theimer. OGSA Basic Execution Service –Version 1.0, 2007.

Feng Liang Feng Liang is currently a Ph. D student in National Laboratory for Software Development and Environment, Beihang University with the research interests in grid computing and cloud computing, he has worked on Migol Grid Project from 2007-2009 in Potsdam University, Germany as an academic visiting student.

Shilong Ma Ph.D. professor, doctoral supervisor, standing deputy director of the National Key Lab for Software Development Environment of the school of computer science and engineering in Beihang University, member of the 10th expert appraisal panel under Department of Information Science of the National Natural Science Foundation Committee, member of executive Committee of Asian Software Foundamental Federation, standing director of China Artificial Intelligence Society. He has undertaken many projects from 973 Program, 863 Program, and National Natural Science Foundation.

André Luckow André Luckow received his Doctor's Degree from University of Potsdam in 2009 in fault tolerance of Grid Computing and Cloud Computing, He currently works in BMW IT Innovation and Development. He has been an academic visitor in Center for Computation & Technology in Louisiana State University in USA and he is also the main contributor to the Grid Project Migol. His research interests include Grid Computing, Cloud Computing and Mobile Computing Applications.

Bettina Schnor Bettina Schnor studied Mathematics and Computer Science at the TU Braunschweig, Germany, where she also received her Ph.D. degree in 1990. From March 1990 to February 1996 she was research scientist at the Institute of Operating Systems and Computer Networks at the TU Braunschweig. There, she established the research field "Distributed Systems" and the working group "Failure Tolerance and Load Balancing in Distributed Systems". During this time, Bettina Schnor did also lecture at the Justus-Liebig-University Giessen (October 94 - July 95). From March 1996 to September 1996 she was a lecturer at the BTU Cottbus for Operating Systems. From October 1996 to March 2000 she was researcher and lecturer at the Institute for Telematics at the Medical University of Luebeck, Germany. Since April 2000 she is head of the Department of Operating Systems and Distributed Systems at the University Potsdam. Her research interests are distributed systems, fault tolerance, cluster and grid computing, and Assisted Living Applications.

© 2012 ACADEMY PUBLISHER