Stateful Grid Resource Selection for Related Asynchronous Tasks

TR-08-02

Howard M. Lander
Robert J. Fowler
Lavanya Ramakrishnan
Steven R. Thorpe

April 25, 2008

RENCI Technical Report Series
http://www.renci.org/techreports

Stateful Grid Resource Selection for Related Asynchronous Tasks

Howard M. Lander and Robert J. Fowler
Renaissance Computing Institute, The University of North Carolina
Chapel Hill, North Carolina
{howard, rjf}@renci.org

Steven R. Thorpe
MCNC
Research Triangle Park, North Carolina
thorpe@mcnc.org

Lavanya Ramakrishnan
Indiana University
Bloomington, Indiana
laramakr@cs.indiana.edu

Abstract

In today's grid deployments, resource selection is based on the prior knowledge of the performance characteristics of the application on a particular resource and on real-time monitoring status of the resource such as load on the system, network bandwidth, etc. Any lag between a resource selection decision and the time the job appears in the system's monitoring facility will cause subsequent decisions to be based on incorrect information. If two or more jobs arrive within this hysteresis window, the incorrect assessment of system state can have negative consequences on job response time and system throughput. In this paper we describe a stateful resource selection protocol we designed to mitigate this problem for a real time storm surge modeling project. We present results from real experiments on a regional grid. We use emulation to compare and study the effect of our protocol under varying load conditions. Based on our evaluation we argue that the enhanced protocol should be made available as a globally-aware grid resource selection service.

1 Introduction

Grid resources are often shared by multiple user communities and can vary greatly in performance and load characteristics. Good resource selection techniques in such environments are critical to obtain reasonable Quality of Service(QoS) assurances. Methods used today make decisions by mapping application resource requirements onto a system model based on static system properties and system status information such as queue lengths, load factors, and network bandwidth. This real-time information is provided to application middleware by grid and cluster services such as Globus Monitoring and Discovery Service (MDS)[2, 11] or Network Weather Service (NWS)[25]. Typically, on-line resource selection uses "best-effort" and "stateless" methods to try to balance the application workload across a set of target clusters. In comparison, if the entire workload can be examined offline, as in the case of workflows and static ensembles, then offline planning techniques such as gang scheduling and related scheduling algorithms are used[13, 18].

Stateless resource selection and planning techniques used today have problems in certain production environments. Consider the case in which several jobs can arrive within a short time interval. In particular, there is a significant lag between the time when a resource selection decision is made and the time this selection is reflected in the monitoring data for the selected resource. Viewing resource selection as a system control problem, this lag induces hysteresis that causes decisions to be made with out of date information. For example, if several jobs arrive before system state information is updated, all of the decisions will be made with the same information. The consequence is that all the jobs may be sent to a single (overloaded) resource when there are other idle resources in the system. This impacts performance by increasing application response time and leaving other available resources under utilized. These effects are particularly bad for grid applications such as weather prediction, economic forecasting, hurricane prediction, *etc.* that have strict timeliness constraints.

In a previous paper[20] we described our experience in creating a distributed software infrastructure for a storm-surge modeling application that uses an *ad-hoc* set of grid resources. Storm-surge predictions are sensitive to various physical parameters that are not known with certainty. Thus, the accuracy and confidence of prediction is increased by running parameter sweep ensembles that cover the space

of possible inputs. Timeliness of the results is critical to emergency responders in coastal areas. The goal of our infrastructure is to harness compute cycles on grid infrastructure to run model simulations in parallel on multiple cluster resources. Input data sets for the model runs are derived from sensors and various simulations and arrive over time through a data distribution network. Hence, there is no *a priori* knowledge of the size or number of model runs in the ensemble. The resource selection protocol in this environment must therefore be able to accept a stream of job requests and adapt dynamically based on the measured state of the resources in the system as provided by grid monitoring services and on the state of other ensemble member runs that have entered the system.

Our resource scheduling protocol use two sources of system information. MDS (both the pre-web services and web services versions) reports the number of free CPUs on each cluster. NWS generates forecasts for future network bandwidth between nodes. Our resource selection protocol uses these data sources and knowledge of system capabilities to produce a ranked list of available cluster resources. We initially used a resource selection protocol in which each model run was independently directed to the most capable free resource without considering any other previously scheduled model runs. We discovered that the lag in the update of grid-wide system state allowed the resource selector to continue to direct many jobs to a single resource, leaving it with a high workload while other, less capable resources remained idle. This hysteresis in the decision process cannot be eliminated just by improving the measurement process because a significant part of the lag is due to the preparatory work (data staging, queuing, *etc.*) done between the time the decision is made and when the job actually occupies computational nodes on the target.

Although *a priori* offline decision rules are not possible in this context, the resource selector can generate an improved estimate of the future state of the system if it uses knowledge of its own actions in the process. Thus, we present a stateful resource selection technique that uses information about its recent job submission decisions that have not yet affected the reported system state. We evaluate the stateless and stateful implementations of resource selection in the context of our environment. We evaluate the performance of the stateful protocol under varying load conditions and make a case for a grid-level resource selection interface that implements our stateful protocol to improve the QoS for all applications. The main contributions of this paper are (a) a "stateful" resource selection protocol, (b) experimental data that validates the need for stateful resource selection over state of the art techniques, and (c) extended evaluation that shows the need for grid-level resource selection interfaces.

2 Background

Our driving application is part of the Southeastern Universities Research Association (SURA) Southeastern Coastal Ocean Observing and Prediction (SCOOP)[4] program. One goal of SCOOP is to advance the study of the effects of hurricane activity on coastal areas. An accurate and timely storm surge prediction system is an essential tool for planning for and responding to hurricanes. We describe the portion of the system needed to motivate our work on resource selection.

We have implemented a distributed software infrastructure used to run ADCIRC[1], a parallel finite-element storm surge model. Our infrastructure enhances SCOOP's study of hurricane activity by enabling multiple instances of the storm surge model to be executed in parallel. Each individual run of the storm surge model is forced by a wind field derived from forecast storm track generated by an ensemble of simulations and other sources. The approach is required to account for the uncertain nature of storm track prediction.

2.1 SCOOP Control Flow

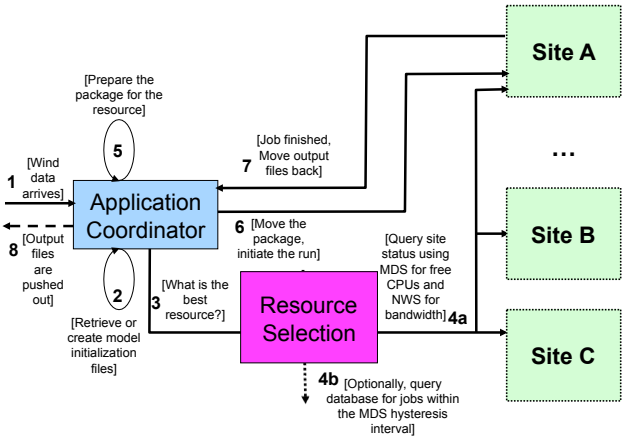

Figure 1. Resource Selection Interactions

Figure 1 summarizes the resource selection interactions within our system. A more complete description of the system is provided in [20]. When a forcing wind field arrives over an event-driven data distribution network using Unidata's Local Data Manager (Step 1), the "Application Coordinator" (AC) is invoked with a new job for this ensemble member. The application coordinator shepherds the job's flow across the grid. The AC publishes progress and status messages to a message broker using the WS-Messenger system[12]. The AC then retrieves all the required application files for the model execution from both local storage and remote archives (Step 2). Next the AC

uses the resource selection protocol to select a resource on which to execute this model run (Steps 3,4a,4b). It then prepares a self-extracting package that includes the binary and the input files for the model run customized for the selected resource (Step 5). The self-extracting package is sent to the batch queue system of the selected resource (Step 6). The delay between the resource selection and the actual submission can be substantial due to the need to copy binaries and process the input data. There is an additional delay between job submission and the start of execution. Another delay is added due to the the frequency at which the resource monitoring components such as MDS update their data. The selected resource monitors the status of the job and publishes messages both when the job begins execution and when the job has completed. Finally, after job completion the output files are retrieved and other clean-up operations are performed (Steps 7 and 8).

Within this control flow, the MDS data doesn't include the effect of the new job from the time resource selection is made until the job is actually initiated on the target. This is the time during which inappropriate resource selections may be made using our original, naive selection protocol. Our enhanced, "stateful" protocol queries a database for jobs that might be in preparation but not submitted (step 4b) to reduce the hysteresis effects on the resource selection. More details are provided in Section 3.

2.2 Resource Selection Requirements

Initially, SCOOP used an ensemble set of five different tracks. More recently, ensembles are approximately 33 tracks. Each of these runs, using the current finite element grid which represents the North Atlantic Basin, requires between 8 and 32 CPUs. Thus, with the increase in the number of ensemble members resource selection has become more critical in our system.

Using our first, stateless resource selection method, there was a strong tendency for all of the jobs of the ensemble to be sent to a small number of resources, whether or not those resources had sufficient available capacity to run the jobs simultaneously. In the worst case, all the runs would be dispatched to a single resource. This resulted in the ensemble being executed sequentially rather than in parallel. Going to the stateful resource selector greatly improves the timeliness of results.

3 Resource Selection Protocols

As previously mentioned, we first implemented a stateless resource selection protocol that uses the real-time status of the resources to make resource choices. This protocol is fairly common and used by many other application managers as well. We present this protocol in section 3.1, and the stateful enhancement in section 3.2.

3.1 Stateless Resource Selection Protocol

In our stateless, or naive, protocol, the application coordinator uses the Java CoG kit[24] to retrieve from MDS an estimate of the number of free CPUs on each resource. An estimated execution time is derived by fitting the number of free CPUs into an open source Rodbard curve fitting library[3, 7], along with the estimated complexity of the simulation (i.e. number of time steps in the job, finite element mesh of the simulation). Differing capabilities and speeds of the cluster resources are factored in using an empirically-determined "slowness" factor. The application coordinator also uses an NWS interface to obtain an estimate of network bandwidth between the machine on which the application coordinator is running and the resource. This number is used to estimate the total number of seconds it will take for uploading the job and later downloading the results. The protocol then uses the minimum completion time, including data transfers, of the job across resources as the metric for resource selection.

3.2 Stateful Resource Selection Protocol

The stateful version of the protocol extends the naive version by adjusting the reported MDS data to discount the reported idle nodes that the resource selector anticipated will be used by the jobs it has recently sent to each resource. This is information about the future that is inaccessible to MDS. As noted in Section 2, our system logs the progress of job execution through an event reporting system. The resource selection protocol queries the database for all model runs that have currently selected a resource but have not yet been reported as running on the resource and uses those results to update the MDS provided CPU count. Protocol 1 shows high-level pseudocode for the resource selection process.

In the SCOOP application, jobs arrive sequentially and in bursts. The lag between resource selection and the time that MDS reflects that selection is not short in relation to the frequency at which jobs arrive. Measurements indicate that the average time between resource selection and notification through the CoG listener mechanism that the job has been queued or is running is approximately 1.5 to 2.5 minutes. This is primarily the time it takes the system to select a resource, create the package to be uploaded and then upload the package using GridFTP. These numbers are still fairly optimistic since MDS will not instantly see the job running. By default MDS is set up to collect data every sixty seconds. On average this adds another half minute to the interval in which MDS data is out of date. Often sites will increase the intervals due to fear of the monitoring overheads over-

```
foreach site do
    Query MDS for free CPU count
    if using the stateful protocol then
        Query database for runs sent to this site that
        aren't yet reported by MDS
        Decrement site's free CPU count appropriately
    endif
    Query NWS for estimated bandwidth
    Estimate total time on this resource, based on 1)
    number of time steps in job, 2) finite element mesh of
    the simulation, 3) free CPU count, 4) resource slow-
    ness factor, and 5) bandwidth estimates
    if this site is the fastest found so far then
        This site becomes the current best site
    endif
endforeach
if the best site has enough CPUs to run the job then
    Return the best site.
endif
else
    Return a randomly chosen site.
endif
```

Protocol 1. Live Grid Resource Selection

whelming the systems. This means that the MDS data we are using as an input to our protocol is actually out of date for approximately 2 to 3 minutes for each run of the application coordinator. In the live grid test runs we used for this paper, the job arrival rate in an ensemble averaged approximately one every 20 seconds.

There are several phenomena that might limit the accuracy of our stateful heuristic. There is no guarantee that by the time this model run begins to execute that any of the runs that we account for are still executing. Those runs could be quite short or they could fail. There may be competing jobs from outside the SCOOP application, or even competing SCOOP applications. These other jobs may start, fail, or finish during the hysteresis interval. We show in section 4 that the stateful protocol improves resource selection choices in these circumstances.

4 Evaluation

To evaluate the effect of using stateful resource selection, we performed live experiments over a regional grid and we used simulations using a system emulator. First, we used our live experiments to confirm the behavior of the enhanced protocol in a working grid. We used the simulations to compare several resource selection protocols to estimate the impact on response times and utilization. We also used the simulations with several assumptions about external loads to estimate the performance of the protocol in the presence of interference and to estimate its effect on other users of the system. These experiments support arguments in favor of implementing a grid-wide resource selection interface.

Test data: Our test bench is an ensemble of ten driving wind fields from tropical storm Alberto (June, 2006). Our live grid test methodology was as follows: cron jobs were set up to run at four and five minutes past the hour for 12 hours (a total of 240 jobs). Each cron job submitted 10 different ensemble members across the four clusters listed in Table 1, with a 30 second delay between each individual member's submission. This is similar to the bursty arrival pattern of the SCOOP application. Resource selection was done using the stateful protocol. Metrics recorded for each run included: resource pool ranking; available CPUs per resource reported by MDS; the available CPUs as estimated by the stateful protocol; selected resource; estimated and actual values for upload and download time, data transfer sizes and times, computation times, total turnaround time for the job, *etc*.

Grid Setup: The set of resources in our test bench is shown in Table 1. While our protocol has knowledge of its prior activities, it has no awareness of previously or about-to-be submitted jobs from other users. We study the effect of this in a controlled environment in our emulation. However in a real grid setting this affects our results and this might require further research to fully understand the impact.

Table 1. Test Bench Resources

Resource Name	Number of CPUs
canbc01.louisiana.edu	6
ci-team.acis.ufl.edu	4
mileva.hpc.odu.edu	16
scoops.itsc.uah.edu	8

Emulator setup: In order to compare protocols in a controlled environment with reproducible external loads, we simulated the system using a Maui-based grid emulator developed by Ramakrishnan *et. al.*[21]. This allows us to compare various stateless and stateful protocols under the same external conditions. Because we capture job and cluster state data on the live system, we can reconstruct the job flow with enough precision to allow a reasonable emulation. The modeling process also allows us to vary aspects of the selection protocol and observe the results in an efficient and controlled fashion.

The inputs to the simulator are summarized in Table 2. Most of these are self explanatory. The *resource selection* mode option controls whether the application simulates the naive or stateful protocol. There is also a mode that makes

random selections among the available resources. The *insufficient resource* mode option controls behavior when the estimate is that no site has enough idle CPUs to run the job. If this mode is set to "random", the protocol randomly selects a resource; if this mode is set to "best", the protocol selects the resource with the largest number of idle CPUs, even if that number is "negative" because multiple jobs have been dispatched to it. We also evaluated the impact of using our protocol with multiple competing job streams. The "system" option creates multiple job streams. Positive integers designate multiple instances of the protocol and a negative system number denotes an external load.

Table 2. Simulator Inputs

Type	Datum	Value
Job	Arrival Time	Time Job Arrived
	RunTime	Execution Time
	Hysteresis	Hysteresis Time
	CPU Count	Number of CPUS Used
	System	System for the Job Run
Site	Site Name	Name of the Site
	CPU Count	CPUs on the Site
	Site Speed	Site "Slowness" Factor
Modes	Res. Selection	Naive, Stateful or Random
	Insufficient Res.	Random or Best

Emulation Protocol: The grid emulator models a set of clusters of differing sizes and speeds. The emulator runs in a discrete time mode in which a virtual clock is explicitly advanced at one second intervals until all submitted jobs are completed.

Protocol 2 shows the emulation of the stateful resource selection procedure. This is similar to Protocol 1, with subtle differences to account for the emulation environment. The emulated naive resource selection protocol differs in two significant ways: it makes no attempt to account for any hysteresis; and in the case where no resource with sufficient CPUs is found, it selects a resource randomly.

foreach *site* **do**
  Query grid simulator for the free CPU count

  foreach *job that selected this site but is not yet running* **do**
    Decrease site's CPU count by the CPU count for the job. // This simulates the hysteresis compensation of the stateful protocol.
  endforeach
  foreach *job currently running on the site* **do**
    if *the current time is greater than job start time + MDS delay interval* **then**
      Increment the site CPU count by the CPU count for the job. // This compensates for the fact that the simulator has no MDS reporting delay.
    endif
  endforeach
  Use the relative resource speed to calculate the effective CPU count for the site
  if *this site is the fastest found so far* **then**
    This site becomes the current best site
  endif
endforeach
if *the best site has enough CPUs to run the job* **then**
  Return the best site.
endif
else
  Depending on configuration, return either the best site or a randomly chosen site.
endif

Protocol 2. Emulated Stateful Resource Selection

4.1 Impact of Enhanced Protocol

In our first experiment, we ran the enhanced protocol in a live grid setting to gauge its impact. In 154 of the 240 runs (64.2%), the selected resource was either changed due to the knowledge of prior submitted jobs available from the MySQL database (11 runs), or was chosen at random from among all the resources. In these 143 cases all resources were deemed full per the combination of MDS queries modified by the local state.

When all queues in the resource pool are non-empty, it is not possible for our protocol to reasonably estimate the total time a job might spend on its chosen resource; the Application Coordinator has no knowledge of what other jobs may be in the system and how long they might take.

4.2 Comparison of Protocols

We conducted several experiments using the emulator. Each experiment uses a job trace we captured during the live grid results. The results of these experiments are summarized in Figures 2 and 3. For protocols with a random component, these are the mean values over twenty runs.

Figure 2 summarizes the results of our first experiment. We simulated the performance of scheduling the canonical job trace with three different protocols. The protocols are the naive protocol, and two versions of the stateful protocol. If no site has sufficient free CPUs, one variant uses random resource selection and the other uses the "best" strategy, as

described above.

We ran additional experiments to investigate the effect of multiple competing job streams. If one subset of users adopts a strategy that improves the response for their job stream, does this adversely impact other users? In particular, what is the effect of running two groups of jobs, each using the stateful policy, as opposed to a single larger group? We therefore did emulations in which the canonical job flow is split into two parts and these two flows are managed independently using the stateful protocol (using the best resource configuration) on the same resource set.

The metrics reported are total time queued for all jobs, total compute time, and the time at which the last job completes. Figure 2 also includes the mean time in queue as well as the mean compute time. The performance of the naive selection protocol, measured by ensemble time to completion, is indeed worse than either of the stateful protocols. This confirms both intuition and the real world observations that led us to this work. The total running time metric is similar across all cases since the the performance characteristics of the emulated resources have little variation.

Note in Figure 2 that the queueing and completion metrics of the first run of two independent stateful streams are close to those of the unified stateful protocols. Further investigation revealed that the canonical job stream had been divided to assign the two longest running jobs to one of the systems, so we created a more even partitioning and ran the emulation again (i.e. Run 2). The fact that slight changes in the input partitioning results in such different results confirms the expectation that that multiple independent stateful systems will not be as effective as a single unified system.

4.3 Impact of External Load on the Protocols

Figure 3 summarizes the behavior of the naive and stateful protocols in response to increasing levels of external load. External jobs are assigned statically to the cluster resources. The effect is to reduce the number of free nodes seen by the resource selectors and to disrupt the accuracy of the stateful predictor.

External load is added in four discrete levels, represented by the X axis on Figure 3. In level one two small jobs are started on one of the clusters at the beginning. At level four each of the clusters is fully subscribed by external load throughout the the majority of the resource selection period of the canonical job trace. The total run time of the external jobs increase from 400 seconds at level one to 3400 seconds at level four.

The metric we use in this experiment is the initial latency time for all resources scheduled by our protocol. This measures the effectiveness of the resource allocator at identifying and using idle resources. On each resource, this measures the "lost capacity" in the form of the integral of unused "job slots" from the start of simulation until the resource is running at maximum capacity. Figure 3 illustrate three qualitative observations. First, the stateful "best" protocol dominates the naive protocol. Also, as we add external load to the system, the metric decreases because the external jobs are already running. Lastly, the performance of the two stateful protocols converge as external load increases, but the naive protocol continues to perform poorly. The fact that, even under substantial externally generated load, the stateful protocols outperform the naive protocol is a strong argument for its use in a grid-level resource selection interface.

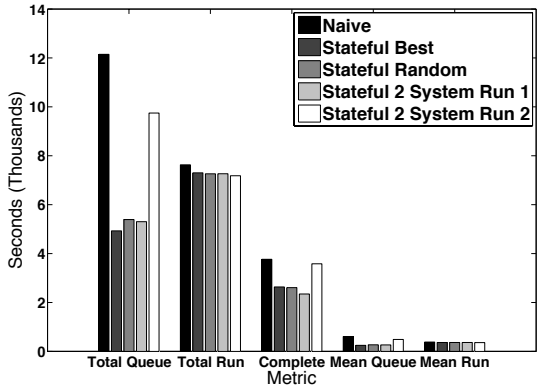

Figure 2. Summary for Multiple Runs

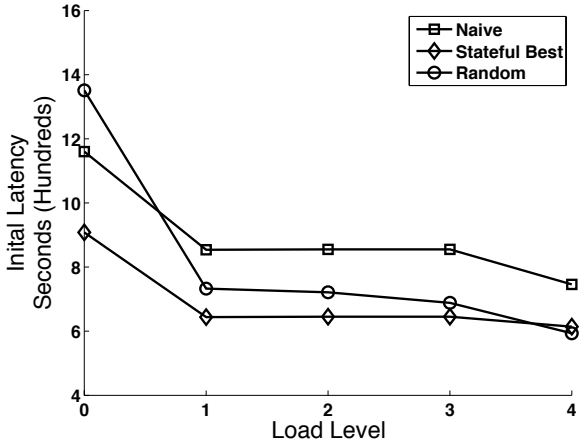

Figure 3. Initial Latency Until System Saturation, Under Increasing External Load

5 Related Work

Today's grid resource management tools provide mechanisms to discover and select resource services in support of grid applications' Quality of Service (QoS) requirements. More recently there has been work on applying offline methods to optimize resource selection for a Directed Acyclic Graphs (DAGs). However, offline methods are appropriate for single applications and/or multi-step workflows. While batch queue systems like PBS, Maui, Condor[17] focus on the problem of optimizing a set of jobs that are already in a centralized queue, the policies are resource-provider centric and intended for use at a single site. In contrast, our approach addresses online resource selection for a stream of loosely-connected jobs over multiple sites

The Network Weather Service's batch queue prediction service (QBETS)[6] offers queue delay predictions for individual jobs. Compute resources can be added to the QBETS service wherever the site allows one to inspect the local queuing system's batch queue prediction log files, then queue delay predictions can be made using similar mechanisms as the NWS network predictions.

There are a variety of resource management systems[19] providing application scheduling and adaptation on grid systems. Monitoring tools are used to evaluate system and application performance and to aid in scheduling and/or rescheduling decisions. AppLeS provides a framework for adaptive scheduling on the grid through distinct steps for resource discovery and selection, schedule generation and selection, application execution and schedule adaptation. Various site selection policies and meta-schedulers such as Grid Service Broker, GridWay, Nimrod/G, *etc*.[5, 10, 22, 23] are being explored in the context of the grid. These provide interfaces for submitting jobs to multiple sites using information collected using standard monitoring tools.

Heuristic techniques are often used to qualitatively select and map resources to available resource pools. Mandal *et. al.*[18] propose a heuristic strategy using performance model based in-advance scheduling for optimal load-balancing on grid resources using the GrADS[16] infrastructure. Blythe *et. al.*[9] identify and evaluate two resource allocation strategies for workflows: task-based and workflow-based. While these are good strategies for optimizing data-dependent application steps, they do not address the online case.

The GrADS workflow scheduler[8] uses a performance-model based workflow scheduling, rescheduling by stop-restart and rescheduling by process swapping. The Virtual Grid Execution System (vgES)[14] provides an integrated resource selection and binding approach to resource allocation allowing higher tolerance to lower resource availability[15].

While there is a need for such resource selection and mapping techniques to aid scheduling at the batch queue level and at the workflow level to optimize multiple applications, these methods do not address the online case of loosely connected jobs. None of these will completely solve the problem we are addressing, which is what to do when multiple jobs arrive in a short interval of time - something that will happen in many systems. That is the need we have addressed in this paper.

6 Conclusions

Job response time and efficient resource utilization are crucial goals for effective grid enabled software infrastructure. In this paper we described and evaluated an enhanced resource selection protocol that minimizes the problems caused by rapid, online arrival of jobs combined with delays between resource allocation decisions and when their effects appear in systemwide monitoring data. Driven by the requirements of the SCOOP application we developed a stateful resource selection protocol to address these problems. Our evaluation shows that the stateful protocol dominates the stateless resource selection approach that is used in today's grid deployments.

Our approach improves the response time of the system and also addresses utilization and throughput. Although it still works well in the presence of competing and external workload streams, our experimental results indicate that the best place for this stateful protocol would be within a globally-aware resource selection and submission interface. There has been recent work on grid-level service infrastructure such as meta schedulers that provide multi-site scheduling capabilities. Our protocol can be easily implemented in one or more of these grid-level services.

7 Acknowledgments

This study was carried out as a component of the "SURA Coastal Ocean Observing and Prediction (SCOOP) Program", an initiative of the Southeastern Universities Research Association (SURA). Funding support for SCOOP has been provided by the Office of Naval Research, Award N00014-04-1-0721 and by the National Oceanic and Atmospheric Administration's NOAA Ocean Service, Award NA04NOS4730254.

We would also like to thank the various SCOOP partners listed at http://scoop.sura.org/partners.html, as well as various member institutions of the SuraGrid listed at http://www.sura.org/programs/sura_grid.html who provided compute resources for this study.

References

[1] ADCIRC website. http://www.adcirc.org.

[2] Globus toolkit 4.0: Information services. http://www.globus.org/toolkit/docs/4.0/info.

[3] Imagej website. http://www.rsb.info.nih.gov/ij.

[4] SCOOP website. http://scoop.sura.org.

[5] Coordinated harnessing of the irisgrid and egee testbeds with gridway. *Journal of Parallel and Distributed Computing*, 66(5):763–771, May 2006.

[6] Network weather service: Batch queue prediction. http://nws.cs.ucsb.edu/ewiki/nws.php, 2007.

[7] M. Abramoff, P. Magelhaes, and S. Ram. Image processing with imagej. *Biophotonics International*, 11(7):36–42, 2004.

[8] F. Berman, H. Casanova, A. Chien, K. Cooper, H. Dail, A. Dasgupta, W. Deng, J. Dongarra, L. Johnsson, K. Kennedy, C. Koelbel, B. Liu, X. Liu, A. Mandal, G. Marin, M. Mazina, J. Mellor-Crummey, C. Mendes, A. Olugbile, M. Patel, D. Reed, Z. Shi, O. Sievert, H. Xia, and A. YarKhan. New grid scheduling and rescheduling methods in the grads project. *International Journal of Parallel Programming (IJPP)*, Volume 33(2-3):pp. 209–229, 2005.

[9] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, and K. Kennedy. Task scheduling strategies for workflow-based applications in grids. In *CCGRID*, pages 759–767, 2005.

[10] R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture of a resource management and scheduling system in a global computational grid. *In Proceedings of 4th International Conference on High Performance Computing in ASIA-Pacific Region, IEEE Computer Press*, cs.DC/0009021, 2000.

[11] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[12] Y. Huang, A. Slominski, C. Herath, and D. Gannon. Wsmessenger: A web services-based messaging system for service-orient ed grid computing. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 166–173, Washington, DC, USA, 2006. IEEE Computer Society.

[13] M. A. Jette. Performance characteristics of gang scheduling in multiprogrammed environments. In *In Proceedings of the ACM/IEEE Supercomputing 1997 Conference (SC'97)*, 1997.

[14] Y.-S. Kee, D. Logothetis, R. Huang, H. Casanova, and A. Chien. Efficient resource description and high quality selection for virtual grids. In *Proceedings of the 5th IEEE Symposium on Cluster Computing and the Grid (CCGrid'05), Cardiff, U.K.* IEEE, 2005.

[15] Y.-S. Kee, K. Yocum, A. A. Chien, H. Casanova, and H. Casanova. Improving grid resource allocation via integrated selection and binding. In *International Conference on High Performance Computing, Network, Storage*, 2006.

[16] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torczon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C. Kesselman, R. Aydt, D. Reed, J. Dongarra, S. Vadhiyar, and R. Wolski. Toward a framework for preparing and executing adaptive grid programs. In *Proceedings of NSF Next Generation Systems Program Workshop (International Parallel and Distributed Processing Symposium 2002), Fort Lauderdale, FL*, April 2002.

[17] M. Litzkow and M. Livny. Experience with the Condor distributed batch system. In *Proceedings of the IEEE Workshop on Experimental Distributed Systems*, Huntsville, AL, October 1990.

[18] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson. Scheduling strategies for mapping application workflows onto the grid. In *High Performance Distributed Computing (HPDC 2005).*, pages 125–134. IEEE Computer Society Press, 2005.

[19] J. W. E. Nabrzyski, J.M. Schopf. *Grid Resource Management*. Kluwer Publishing, 2003.

[20] L. Ramakrishnan, B. O. Blanton, H. M. Lander, R. A. Luettich, Jr, D. A. Reed, and S. R. Thorpe. Real-time Storm Surge Ensemble Modeling in a Grid Environment. In *Second International Workshop on Grid Computing Environments (GCE), Held in conjunction ACM/IEEE Conference for High Performance Computing, Networking, Storage and Analysis*, November 2006.

[21] L. Ramakrishnan, L. Grit, A. Iamnitchi, D. Irwin, A. Yumerefendi, and J. Chase. Toward a Doctrine of Containment: Grid Hosting with Adaptive Resource Control. In *Proceedings of the ACM/IEEE SC2006 Conference on High Performance Computing, Networking, Storage and Analysis*, Tampa, Florida, November 2006.

[22] S. Vadhiyar and J. Dongarra. A metascheduler for the grid. In *Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing*, July 2002.

[23] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling e-science applications on global data grids: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(6):685–699, 2006.

[24] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java commodity grid kit. *Concurrency and Computation: Practice and Experience*, 13(8–9):645–662, /2001.

[25] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5–6):757–768, 1999.