Co-Scheduling with User-Settable Reservations

Kenneth Yoshimoto, Patricia Kovatch, Phil Andrews
San Diego Supercomputer Center
University of California, San Diego
kenneth@sdsc.edu, pkovatch@sdsc.edu, andrews@sdsc.edu

Abstract

As grid computing becomes more commonplace, so does the importance of co-scheduling these geographically distributed resourcest. Negotiating resource management and scheduling decisions for these resources is similar to making travel arrangements: guesses are made and then remade or confirmed depending on the availability of resources. This "Travel Agent Method" serves as the basis for a production scheduler and metascheduler suitable for making travel arrangements for a grid. This strategy is more easily implemented than a centralized metascheduler because arrangements can be made without requiring control over the individual schedulers for each resource: the reservations are set by users or automatically by negotiating with each local scheduler's user-settable interface. The Generic Universal Remote is a working implementation of such a system and proves that a user-settable reservation facility on local schedulers in a grid is sufficient to enable automated metascheduling.

1. Introduction

A grid is a distributed computing and resource environment connected via software and hardware. The resources can be as diverse as electron microscopes or Terabyte-sized databases. Computational resources that are a component of a grid usually have many processors available for the use of scientists and researchers. These resources often have unique characteristics like a "large" amount of memory or access to "large" amounts of disk. Generally the compute resources have a local resource manager and scheduler to allow sharing of resources between the different users of the system.

In the case of this project, development was done within the context of a specific Grid, the TeraGrid[1], an NSF-funded project to create and link several supercomputer class systems across one of the world's fastest networks. The initial middleware suite used for coordination has been Globus[2,3], while recently there has been significant work done on the use of a Global File System[4], initially based on the General Purpose File System, GPFS[5], from IBM. A similar approach is being investigated by the European Grid community, DEISA[6].

A prior study by another group evaluated multi-site submission of single jobs [7]. This article describes a strategy for scheduling a single job over multiple sites.

A variety of applications can make use of these grid resources. Some applications use loosely-coupled communication resources while others require tightly-coupled communication resources. Some need to stage data from one location to another before it can compute and process additional data. Some applications can make use of distributed resources but synchronize some communications between the sites. Because of this synchronization, resources need to be available at multiple sites at the same time. Some applications want to run at the earliest time possible regardless of the machine type or geographical location of the resources.

Often the grid is compared to an electric utility: always available. But this does not necessarily apply to High Performance Computing (HPC) and grid computing, where demand exceeds supply. The electrical grid is planned to always have more electrical supply than demand. If demand ever exceeds supply, there are rolling brownouts. In this situation where supply exceeds demand, “on-demand” is easy: just plug into a wall outlet, because there’s enough slack in the system to provide the power necessary to run a radio. This can be contrasted with HPC and grid computing, where demand often exceeds supply. There are always more applications, more compute, more data and other resources needed to further the never -ending stream of research. The continual demand for capability requires the commitment of resources for specific jobs. The electric grid/”on demand” analogy breaks down for HPC. The scheduling strategies for these two situations are entirely different.

A better analogy is an industry where complex workflows cause excess demand for limited resources: travel. Every day, travel agents create multi-resource itineraries. An itinerary with round-trip airplane flights, rental car and hotel reservations corresponds very well to that of a grid job that needs to do pipeline computation. The travel agent works with a number of different independent resource providers to generate a valid itinerary. One thing that’s interesting about this model is that there is no requirement for global coordination. The airline does not need to know that there are rental cars available at the destination, for example. For the travel industry, there is a many-to-many supplier-to-consumer market. For the electric grid, there is one supplier through which all consumers in a region must go. The economics of the travel industry is a better model for HPC/grid computing.

The Generic Universal Remote (GUR) provides distributed resource management capabilities that make efficient use of HPC/grid computing resources based on a travel agent’s methods. It provides automatic negotiation of coordinated cross-site (pipeline and co-scheduled) and first possible run-time (Disneyland or load balanced) reservations allowing these types of jobs to run efficiently. These types of jobs are scheduled in an automatic way by taking advantage of local scheduler’s user-settable reservation capabilities. GUR requests and confirms reservations from independent local schedulers to generate a grid reservation. It uses a simple heuristic to generate a valid, but not necessarily optimal schedule. GUR does not require centralized control over all the grid resources, unlike other metascheduling solutions. GUR automatically reserves resources from a scientists’ laptop that not only makes it easier for the scientist, but it reduces load on the remote “login” nodes. This decentralized local scheduler works like a “universal remote” where it commands and coordinates access to multiple, heterogeneous distributed resources based on where you “point it”, much like a universal electronics remote.

.

2. Grid Scheduling Strategies

Many different types of applications make use of grid resources. Some typical scenarios include communication-, compute-intensive and combination-type jobs. Communication-intensive applications launch jobs to run on a single remote cluster. Computation –intensive jobs run on geographically distributed clusters. Communication- and computation-intensive jobs run on a single remote cluster and then possibly transfer the data to another site for visualization.

These applications represent three types of job scheduling scenarios that are common to grid computing:

1) Run x job on n nodes, where all n nodes are located at any one of a set

of sites, at the earliest possible time (Disneyland)

2) Run x job on n nodes, where n nodes may be distributed across multiple sites at the same time (co-scheduled)

3) Run x job at site A, then move output to site B for additional computation, visualization or database access (pipeline)

In the first scenario, a user wants to run as quickly as possible. For instance, the user has a communication-intensive type of job and doesn't care which single compute resource (out of a set of compute resources) the job runs on. For example, when you visit Disneyland with your friends, the object is to get on as many rides as possible. Since the rides are popular, there is a wait to get on the rides. Which means you and your friends will wait in line for a ride and then move to the next ride in sequential fashion. A more efficient way to get on more rides would be to split up with your friends and have each of you wait in line at different rides. Then, whoever gets to the front of the line first "wins" and you and your friends leave their different lines to join the "winning" person. This method allows you and your friends to ride as many rides as possible. Many scientific computation applications exhibit this behavior. A brain imaging application renders images from raw data is an example of a kind of application that uses this kind of scheduling scenario. It has been shown that this strategy does not necessarily reach the optimal solution, if the resources are heterogenous [7]. We have also not studied the effect on global and local utilization with such a strategy. Intuitively, a "market" of resource consumers and resource providers might be expected to emerge. Information, in the form of submitted jobs, would be available to resource providers. We speculate that this dissemination of information might allow more efficient global utilization, without a centralized scheduling mechanism.

Some kinds of computation-intensive jobs make use of geographically distributed resources. These jobs run within each separate site and communicate between sites. For example, to develop a full weather model for the ocean's atmosphere, one part of the job computes the ocean's effects on the atmosphere at one site and the other part of the model computes the effects of the atmosphere on the ocean. The distributed jobs then communicate at the end of the model to share the data to develop a complete model. This kind of job needs resources to be available simultaneously (co-scheduled).

Different sites offer different capabilities. Because of this, users want to compute at one site, move data to another site, visualize the data at another site and finally store the output at a last site. The storing of data can't start until the data is visualized. And the visualization can't start until the data is moved and so on. The space for the data needs to be available at the remote site. So a schedule of reservations is needed on resources to complete the job workflow. Each step in the process depends on the previous step. An example of this kind of job is an astronomy model that needs specific compute resources at one site and the data and visualization resources at another site. Another instance is a database server with a limited amount of space. In this limited amount of space, users can bring up their own database and stage data into it. A resource manager and scheduler for the database can grant space reservation requests to user. These reservations can be coordinated with regular compute reservations on a separate system. This staging of events is called a pipeline.

3. Grid User-settable Reservations and Catalina Scheduling

A local scheduler called “Catalina,” was developed to provide a user-settable reservation facility for IBM’s LoadLeveler, Portable Batch System (PBS) or any local resource manager that has an interface for an external scheduler. Catalina is a reservations-based, single-queue scheduler, much like the Maui scheduler. It prioritize jobs, based on a number of different characteristics. It calculates the expansion factor for how long a job is waiting and adjusts the priority on the job so it won’t starve. It also has backfill capabilities to keep the processors busy until the right number of nodes are available for a larger job with higher priority. When a system reservation is made, jobs that will complete before the system

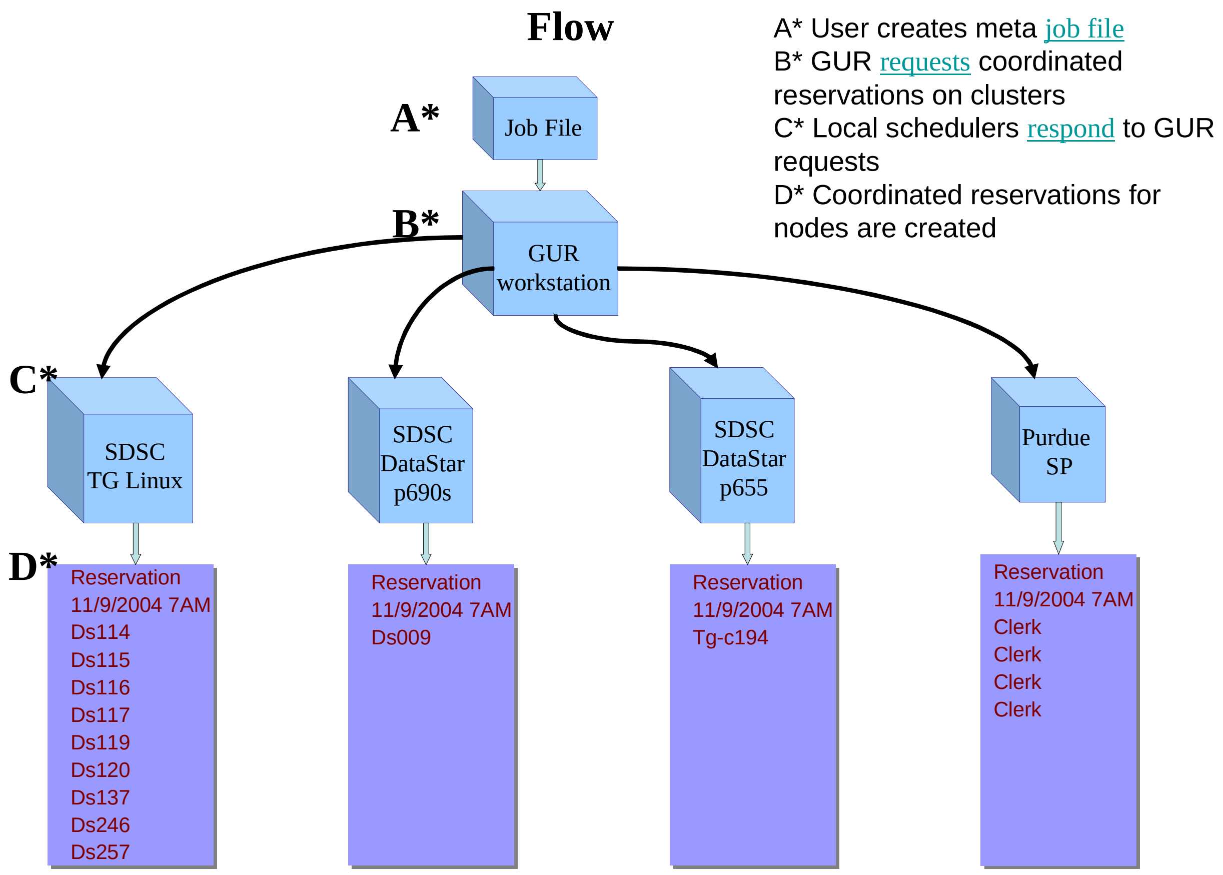

Flow
A* User creates meta job file
B* GUR requests coordinated reservations on clusters
C* Local schedulers respond to GUR requests
D* Coordinated reservations for nodes are created
A*
Job File
B*
GUR workstation
C*
SDSC TG Linux
SDSC DataStar p690s
SDSC DataStar p655
Purdue SP
D*
Reservation 11/9/2004 7AM Ds114 Ds115 Ds116 Ds117 Ds119 Ds120 Ds137 Ds246 Ds257
Reservation 11/9/2004 7AM Ds009
Reservation 11/9/2004 7AM Tg-c194
Reservation 11/9/2004 7AM Clerk Clerk Clerk Clerk

Figure 1. Job Flow Schematic

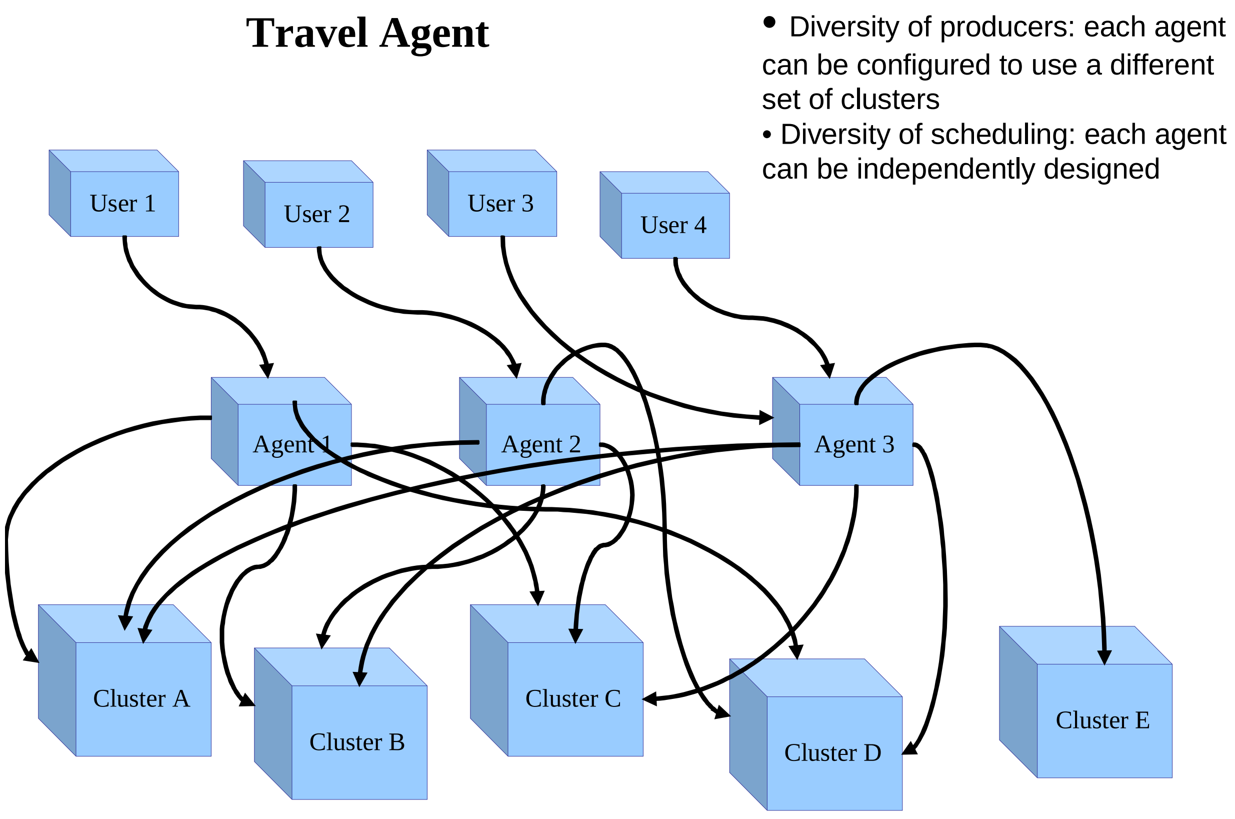
reservation starts are scheduled to run. It can schedule any kind of resource including data, database or compute. Catalina consists of 10,000 lines of Python with some functions written in C. The user-settable reservation facility consists of a command-line client run by an unprivileged user. Parameters provided to the user include:

1) Allocation charge account
2) Exact or maximum number of nodes requested
3) Duration of the reservation
4) Earliest time at which the reservation can start
5) Latest time at which reservation may end
6) Email address for failure notification

To keep users from severely disrupting the batch schedule, reservations are restricted by the following policies:

1) User-settable reservations are made only after all currently queued jobs are scheduled
2) Number of reservations per account can be limited
3) Number of nodes and duration of each reservation can be limited
4) Global limit on the number of node-seconds devoted to user-settable reservations in a configurable time window

Catalina is the scheduler running on the production supercomputers at the San Diego Supercomputer Center. For instance, it's the scheduler for Blue Horizon, an 1100 processor IBM SP2. It interfaces with LoadLeveler, IBM's proprietary resource manager. It's been running successfully for over three years. In this time, none of the 2000+ user accounts on Blue Horizon interrupted, interfered or delayed the overall batch scheduling process with user-settable reservation capability. Catalina is consistent with the Global Grid Forum Advanced Reservations API. More information on Catalina can be found at http://www.sdsc.edu/catalina .


Travel Agent

- Diversity of producers: each agent can be configured to use a different set of clusters
- Diversity of scheduling: each agent can be independently designed

Figure 2. Scheduling Communication

3. Travel Agent Method

When a travel agent makes reservations for a trip, the agent starts with specific dates and start/end point locations in mind. Then the agent makes a starting guess based on those dates and locations for a specific time for the reservation. The agent may check several airlines (resources) for availability that match the traveler's parameters. If the flight meets the traveler's needs, the reservation is made. If several flights meet the traveler's needs, the first available flight in those times is booked. If no flights meet the traveler's need, the agent makes another guess about flight times to match the traveler's next best time. If the traveler's needs are met, the reservation is made. If not, another guess is made, and so on. This trial and error strategy generates resource reservations in an acceptable amount of time since there are a finite number of resources available in a specific time period.

Once the airline is reserved, the agent will make the car reservation based on the airline arrival and departure times. And the hotel reservation is dependent on the airline and car rentals times and so on. A draft itinerary is created and checked with the traveler before being booked permanently.

Making reservations for distributed grid resources can be done in a similar way: resources can be scheduled sequentially for jobs requiring staged multiple resources or in parallel for co-scheduled resources. Both user-controlled (manual) and automatic reserved resource acquisitions make efficient use of grid resources since system administrator time is not required to check and make reservations at different sites. This strategy makes reservations for the three different types of jobs requests outlined in the Grid Usage Scenarios section.

4. Generic Universal Remote (GUR)

Traditionally, only system administrators make reservations for resources with local schedulers. With this approach, creating Disneyland, co-scheduled or pipeline reservations require communication with the local system administrators at each of the sites the user wants to run. Contacting individual system administrators at each site is a time consuming process that undoubtedly requires multiple iterations Traditional approaches to metascheduling require a centralized scheduler that controls the schedulers at each of the sites. This is an impractical approach on the grid, where each site has it's own security and local scheduling policies.

When the user-settable reservation capability is available to users, then users can "self-schedule" and request resources when needed. Users act as their own travel agent and reserve co-scheduled jobs without manual system administrator intervention. This manual process is automated with GUR.

Using the Travel Agent Method, the GUR negotiates reservations with local schedulers. It probes the local schedulers to find potential suitable times and then makes a guess at possible times for the job to execute. Once a time is found, the reservation is made. In order for this to happen, user settable reservations need to be available on the local schedulers.

GUR is a metascheduler that was developed to automatically created coordinated reservations on local schedulers, using the user-settable reservation facility. A user submits a metajob to GUR providing information such as:

1) Total number of nodes needed
2) Minimum and maximum number of nodes needed for each local system
3) Job duration
4) Earliest start time
5) Latest end time
6) Usage scenario
    a) Single system (Disneyland)
    b) Multiple systems (co-scheduled)
    c) Multiple systems (pipeline)

A GUR job request would look something like this:

```
[metajob]
total_nodes=12
machine_preference=datastar655,purduesp
machine_preference_reorder=yes
duration=7200
earliest_start=07:00_11/09/2004
latest_end=09:30_11/09/2004
usage_pattern=multiple
machines_dict_string = {
 'datastar655' : {
 'username_string' : 'kenneth',
 'account_string' : 'sys200',
 'email_notify' : 'kenneth@sdsc.edu',
 'min_int' : 1,
 'max_int' : 158
 },
 'purduesp' : {
 'username_string' : 'kenneth',
 'account_string' : 'TG-STA040001N',
 'email_notify' : 'kenneth@sdsc.edu,
 'min_int' : 1,
 'max_int' : 2
 }
 }
```

For job submission, architecture-specific requirements are abstracted into a GUR configuration file. Jobs request required resource features by specifiying a 'machine', such as 'datastar655'.This would mean p655 nodes on DataStar. At least one node must be used on each cluster. No more than 158 can be used on 'datastar655', and no more than 2 can be used on 'purduesp'.

Then GUR probes each system to make a rough guess at each system's queue load. It does this by setting test reservations on each system and checking the delay for each reservation. This approach is not optimal for all possible queue states, but it does provide usable information on the status of each system.

In order of the least loaded system, GUR makes reservations consistent with the minimum and maximum nodes for each system and the total number of nodes requested. Nodes are preferentially distributed to the least loaded systems. If the initial distribution of nodes to systems is not possible in the requested time frame, a new distribution is generated, giving more nodes to the more heavily loaded clusters. As soon as a solution is found, GUR stops. GUR does not have a 24 hour hold on reservations or a two phase commit. It makes reservations and cancels them, if they are no longer required. While finding the initial optimal distribution of times and nodes, GUR makes a "sliding reservation window" time based on the requested running parameters of the earliest start time and latest end time. It binds the job to the reservation and vise versa.

In addition, GUR is "generic" and works with any local scheduler (with user-settable or manually-settable reservations) that schedules any kind of compute, data or instrumental resource. GUR can perform a series of scripted tasks. For instance, a job can automatically compile and execute based on GURs instructions. It can also stage data or an executable. This enables a "universal" grid roaming capability, where jobs can be launched from a laptop regardless of the operating system on the destination resource. It also reduces the load on "login" nodes because users don't need to login to submit jobs. If the user performs a grid-proxy-init and specifies a GSI-enabled SSH, then GUR will use that to contact the remote systems. If no GSI-enabled SSH is available, or the remote system's gatekeeper is down, GUR will use regular SSH, setting up an agent for the user.

If a reservation cannot be fulfilled, perhaps due to hardware failure or unplanned maintenance,then the local scheduler is expected to email notification to the user.

If any local scheduler is unable to provide the minimum number of nodes within the sliding reservation window, GUR informs the user that the reservation is not possible. The user may then expand the time window, reduce the number of resources requested or both and resubmit.

GUR also gives your local computer the capacity to act like a television "remote" where it can control the geographically distributed resources on the grid behind it by providing a single, uniform, easy-to-use interface for the user.

GUR works with any local scheduler that takes user-settable reservations (such as Catalina) or has an interface for reservations. It can also work with the Maui Scheduler with manual system administrative assistance. GUR conforms to Global Grid Forum Advance Reservations API. GUR does not currently support pipeline jobs, but it would be easy to extend the multiple usage pattern to pipeline by providing a time offset to each resource so they happen in sequential rather than simultaneous order.

More information on GUR can be found at http://www.sdsc.edu/~kenneth/gur.html . GUR can be downloaded from http://www.sdsc.edu/scheduler/gur.html .

Real-Life Experiences with Metascheduling and GUR

Several previous experiences explored the use of coordinated reservations. At a previous SuperComputing conference, a user manually created reservations through the General-purpose Architecture for Reservation and Allocation (GARA) to the Portable Batch System Professional (PBSPro). The co-scheduled metajob ran across sites using MPICH-G.

In another demonstration, a user made reservations on an SDSC IA-32 Linux cluster running the Portable Batch System (PBS) resource manager and on SDSC's Blue Horizon, an 1100 processor SP, running the LoadLeveler resource manage r. The co-scheduled metajob ran between the machines using MPICH.

Co-scheduled reservations were made with the Silver metascheduler on Blue Horizon's LoadLeveler/Maui and an SP at Pacific Northwest Laboratories running LoadLeverl/Maui. ECCE/NWChem was the application. Silver is a centralized metascheduler that only works with the Maui scheduler.

Additionally, SP clusters at SDSC, University of Michigan – Ann Arbor and University of Texas – Austin were reserved and scheduled a centralized metascheduler during a separate SuperComputing demonstration. These clusters all shared the same UID/GID space. All the clusters used LoadLeveler and the Maui scheduler. Again, the centralized metascheduler made the reservations from a privileged account for a co-scheduled metajob.

In a new approach, GUR was used to automatically schedule jobs with various characteristics on several heterogeneous compute platforms. At SDSC, Disneyland and co-scheduled jobs were scheduled between Blue Horizon and a Linux cluster. Both of these resources are working, production supercomputers with real user jobs scheduled and running at the time of the tests.

User job requests were made from each compute platform to GUR. These jobs requested a various number of processors, run times and executables. Some jobs requested to run at the same time on both platforms. And other jobs requested to run at the earliest possible time. All of these tests worked. This tests shows that grid metascheduling can be performed with local schedulers with user-settable reservations and a decentralized metascheduler.

In the latest demonstration of GUR capability,at SC2004, co-scheduled reservations were made across three platforms. These were SDSC DataStar (IBM SP with Catalina), SDSC Linux cluster (ia64/Myrinet with Catalina), and Purdue SP (IBM SP with PBSPro). The SDSC machines are running production workload, while the Purdue machine was a test system. GUR was able to successfully create a set of synchronized reservations across the three clusters. GUR reserved 10 nodes on SDSC DataStar, one node on SDSC Linux cluster, and one node on the Purdue SP. These reservations were all scheduled to start at 7am Nov 9, 2004 PST.

GUR is similar to Silver in that it depends on reservations created on the local schedulers. It differs in using user-settable reservations rather than privileged reservations. GUR resembles Condor-G, since GUR can submit jobs to diverse compute resources. GUR makes use of user-settable reservations to provide synchronization of job starts, which Condor-G[8] does not do.

5. Conclusion

Resources can be easily scheduled on a grid by deploying an automatic scheduler that mimics the human travel agent's process for making reservations. Reading the user's request for resources, making a guess at the best possible times and sliding the window of those times until a match is found reserves resources in a reasonable amount of time. Various types of typical HPC/grid computing jobs can be scheduled in this manner, including Disneyland, co-scheduled and pipeline jobs.

A working version of this metascheduler, GUR, along with a local scheduler with user-settable reservations, Catalina, proves that this approach is possible. The Catalina-GUR system demonstrates that a user-settable reservation facility is sufficient to enable automatic, coordinated metascheduling. User settable reservations are practical since they are controlled by policies that restrict users from interrupting the flow of the batch system. The ability to request any number of nodes up to a maximum makes it much easier to explore the node distribution space throughout the grid.

This architecture allows many alternate metaschedulers to participate in the grid. It also allows additional flexibility since it can be run from a laptop, giving unprecedented, ubiquitous access to users of the grid. In addition, it also removes the requirement for a centralized metascheduler, which is difficult to coordinate and make secure. The combination of user-settable reservations along with an automated, de-centralized metascheduler is an approach that allows many new types of HPC applications to run on a grid.

6. Acknowledgments

We wish to thank Wendy Lin of Purdue for development of code to enable user-settable reservations on the Purdue SP.

7. References

[1] Catlett, C. The **TeraGrid**: A Primer, 2002. www.**teragrid**.org
§
[2] I. Foster and C. Kesselman, “Globus: A Metacomputing Infrastructure Toolkit,” *Intl. J. Supercomputer Applications*, Vol. 11, No. 2, pp. 115-128, 1997,

[3] Foster, I., Kesselman, C., Nick, J.M. and Tuecke, S. Grid Services for Distributed Systems Integration. *IEEE Computer*, *35* (6). 37-46. 2002

[4] A Centralized Data Access Model for Grid Computing, Phil Andrews, Tom Sherwin, and Bryan Banister, Twentieth IEEE Symposium on Mass Storage Systems, (April 2003)

[5]GPFS: A Shared-Disk File System for Large Computing Clusters , Frank Schmuck

and Roger Haskin, Conference Proceedings, FAST (Usenix) 2002

[6] http://www.deisa.org/

[7]Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment, Gerald Sabin, Rajkumar Kettimuthu, Arun Rajan and P. Sadayappan, Proceedings of 9th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2003), June 2003

[8]"Condor-G: A Computation Management Agent for Multi-Institutional Grids", James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke,Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001.