GridARS: An Advance Reservation-based Grid Co-allocation Framework for Distributed Computing and Network Resources

Atsuko Takefusa Hidemoto Nakada Tomohiro Kudoh Yoshio Tanaka
Satoshi Sekiguchi
National Institute of Advanced Industrial Science and Technology (AIST)
{atsuko.takefusa,hide-nakada,t.kudoh,yoshio.tanaka,s.sekiguchi}@aist.go.jp

Abstract

For high performance parallel computing on actual Grids, one of the important issues is to co-allocate the distributed resources that are managed by various local schedulers with advance reservation. To address the issue, we proposed and developed the GridARS resource co-allocation framework, and a general advance reservation protocol that uses WSRF/GSI and a two-phased commit (2PC) protocol to enable a generic and secure advance reservation process based on distributed transactions, and provides the interface module for various existing resource schedulers. To confirm the effectiveness of GridARS, we describe the performance of a simultaneous reservation process and a case study of GridARS grid co-allocation over transpacific computing and network resources. Our experiments showed that: 1) the GridARS simultaneous 2PC reservation process is scalable and practical and 2) GridARS can co-allocate distributed resources managed by various local schedulers stably.

1 Introduction

Grid technologies allow large-scale parallel computing, namely metacomputing, over distributed computing resources managed by different organizations. A crucial issue for achieving high effective performance of fine-grain message passing applications over Grid environments is Grid co-allocation of various distributed resources.

At this point, we perform Grid co-allocation as follows:

(1) Manual reservation and job execution by SSH

The user reserves distributed resources by human negotiations such as e-mail and phone for each resource manager and performs metacomputing over the reserved resources at the reserved time. Some academic Grid test beds apply this strategy, but the problems are: it is difficult to use resources effectively, someone might use the reserved resources, and it is unrealistic to expect to have a local account on all of the available resources in large-scale Grid environments.

(2) Manual reservation of resources managed by resource schedulers

The user reserves resources manually as in (1), and administrators of the corresponding cluster managed by a batch queuing system configure a reservation queue according to the requirements. Then, the user submits jobs to the queues and performs metacomputing. The Tera Grid project in the US[1] adopts this strategy, which allows management of resources based on each organization's policy. On the other hand, many manual configuration errors have been reported.

(3) Automatic reservation of resources managed by resource schedulers

Resources are managed by a local batch queuing system with an advance reservation capability and a global scheduler co-allocates distributed resources for user requirements. Then the user submits jobs to the reserved queue. This strategy allows resource management based on each organization's own policy, as well as (2) avoiding human configuration errors. However, there have been several technical issues standing in the way of automatic reservation by global schedulers, as described in Section 2

We propose GridARS (Grid Advance Reservation-based System framework), a Grid co-allocation framework for distributed resources, such as computers and network, and we developed a general advance reservation protocol over WSRF (Web Services Resource Framework)[2].

GridARS co-allocation architecture consists of a Global Resource Scheduler (GRS) and Resource Managers (RM, local schedulers), and automatically co-allocates required resources via WSRF. It enables a simultaneous reservation process for multiple resources by using a hierarchical two-phase commit (2PC) protocol between the User-GRS and GRS-RMs.

The main components of GridARS are the GridARS-Coscheduler and GridARS-WSRF. The GridARS-Coscheduler finds suitable resources for each user and co-allocates the resources by distributed transactions. GridARS-WSRF is an interface module for the proposed 2PC advance reservation protocol over WSRF. Our GridARS-WSRF implementation, called GridARS-WSRF/GT4, has been developed using Globus Toolkit 4 (GT4)[3].

To confirm the effectiveness of GridARS, we present the basic performance of our 2PC reservation process between GRS and 8 RMs over WSRF/GSI using GridARS, and describe a case study of GridARS Grid co-allocation of transpacific computing and network resources. Our experiments showed that: 1) the GridARS simultaneous 2PC reservation process is scalable and practical and 2) GridARS can co-allocate distributed resources managed by various local schedulers stably.

2 Issues for Grid Co-allocation

Various resources, such as computers, network, and storage, on Grids are generally used by local domain users. In Grid co-allocation, resource schedulers have to provide their resources for both local users and global users, and thus must aim for co-allocation over Grids, efficiently. To resolve this situation, there are the following issues:

Co-allocation of various resources Existing Grid global scheduling system, such as Moab[4] and CSF[5] actually address only computing resources. However, high performance parallel computing over distributed environments requires not only computing resources, but also network resources, such as bandwidth. A global scheduling system co-allocates various resources with assured performance.

Coordination with existing resource schedulers In order to use resources efficiently under differing domain policies, most Grid resources have been managed by resource schedulers such as GridEngine[6], TORQUE[7] or other commercial batch queuing systems. Global schedulers have to provide resources for global users in coordination with existing resource schedulers.

Advance reservation Local resource schedulers basically allocate each user job based on strategies such as FCFS. In this situation, it is difficult to estimate when a user job will start. To co-allocate resources without losing each local resource, an advance reservation capability is required for local and global schedulers.

WSRF/GSI WSRF is a standard interface for stateless services. Most resource schedulers provide a command line interface or a graphical interface. To provide resources for various global users, who usually do not access resource scheduler hosts by SSH or other schemes, resource schedulers and global schedulers should provide a standard WSRF interface with secure communication, such as GSI (Grid Security Infrastructure).

Two-phase commit Resource schedulers should support a two-phase commit (2PC) reservation interface so that global schedulers can allocate distributed resources simultaneously based on distributed transactions. As shown in Fig. 1, we assume modification of reservation time on reserved resources managed by distributed resource schedulers using a one-phase commit (1PC) protocol, which most resource schedulers support. (1) After User sends a modification request to the global scheduler, called Co-allocator, Co-allocator sends the request to related resource managers. (2) In this case, RM2 has failed to modify the reservation time but the other RMs have succeeded. Then, (3) User and Co-allocator send a rollback request of the reservation time to RM0, RM1, and RM3. But at (4), the rollback has failed fatally because the rollback on RM1 has failed due to another reservation being inserted in advance.

3 GridARS Grid Co-allocation Framework

In order to resolve the above issues, we propose, and have developed, a GridARS (Grid Advance Reservation-based System framework) co-allocation framework, which allows co-allocation of widely-distributed resources managed by various organizations and resource schedulers.

An overview of the GridARS co-allocation framework is shown in Fig. 2. GridARS consists of a

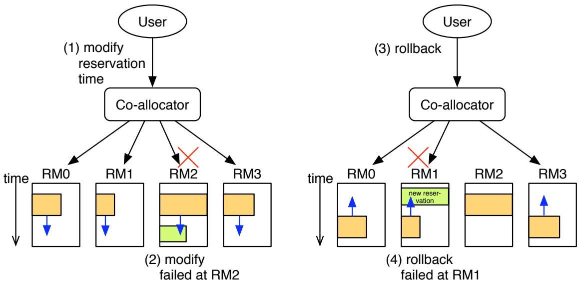

Figure 1. An example of rollback failure by one-phase commit.

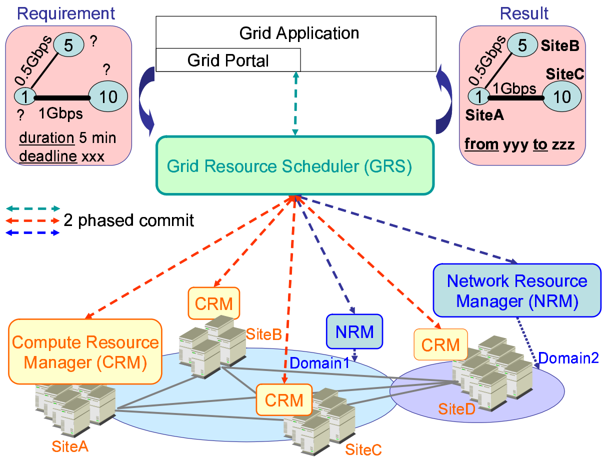

Figure 2. Overview of the GridARS co-allocation framework.

Global Resource Scheduler (GRS) and Resource Managers (RM) for computers (CRM), network (NRM), and other resources. In each RM, existing resource schedulers manage a reservation table of their resources for advance reservation. A User sends requirements on resources and reservation time to GRS, and then GRS co-allocates suitable resources in coordination with related RMs.

The dotted lines between User-GRS and CRS-RMs in Fig. 2 indicate a two-phase commit (2PC) advance reservation process so that GRS can book distributed resources simultaneously based on distributed transactions. As shown in Fig. 2, GridARS provides a hierarchical 2PC process so that GRS can be one of the resource managers, because it is easy to coordinate with other global schedulers.

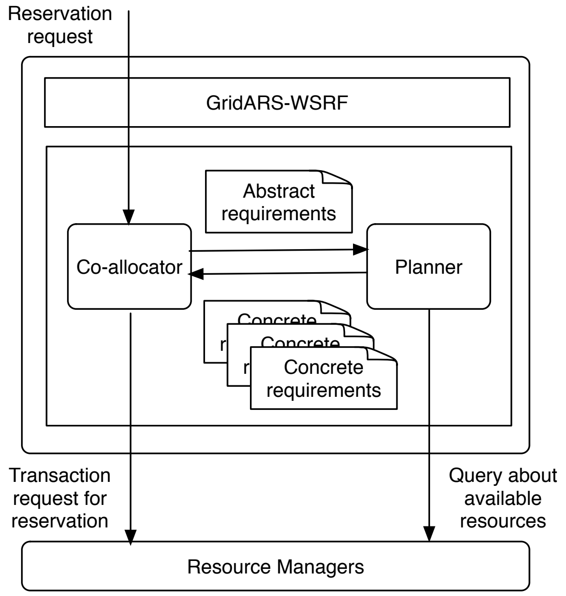

Figure 3. GridARS-Coscheduler System Architecture.

GRS consists of GridARS-Coscheduler and GridARS-WSRF. Grid-Coscheduler selects suitable resources for user requirements for resources and co-allocates the resources based on distributed transactions. GridARS-WSRF is an interface module of this 2PC WSRF reservation process. Each RM consists of GridARS-WSRF and an existing local resource scheduler.

3.1 GridARS-Coscheduler

GridARS-Coscheduler consists of a Co-allocator and a Planner, as shown in Fig. 3. Co-allocator receives user resource requirements via GridARS-WSRF and sends the requirement to Planner. From the user request and current resource status, Planner determines candidates from among concrete resources and then returns the planning results to Co-allocator. One solution to get distributed resource information is a centralized global information service to collect and provide local resource information. However, a commercial resource manager cannot expose resource information, and the amount of reservation timetable information is larger than current resource information as managed by current information services, such as Ganglia[8]. Therefore, GridARS GRS requests resource information from each RM directly. Planner is replaceable for

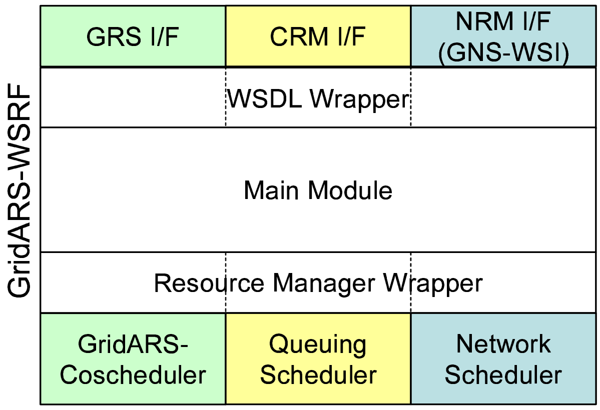

Figure 4. GridARS-WSRF System Architecture.

each manager strategy or user requirement.

Then, Co-allocator negotiates with the related RMs and books the resources selected by Planner simultaneously based on distributed transactions. Details of the reservation process will be described in Section 4. After the reservation process has finished, Co-allocator monitors the status of the reserved resources periodically.

3.2 GridARS-WSRF

GridARS-WSRF is a polling-based 2PC interface module for advance reservation. In a polling-based situation, the number of communications between client and server will increase, and the client detects a change of resource status behind the actual change. On the other hand, this enables asymmetric communication, e.g., a client does not have global address or firewall problems. WS-Notification[9] has been proposed for notification over web services and it also requires polling from the client side in order to detect network or server failures. Therefore, GridARS is based on polling and applies WS-Notification, optionally.

GridARS-WSRF consists of a WSDL Wrapper, a Main Module, and a Resource Manager Wrapper. Fig. 4 is an example of GRS, with CRM for the computing resource, and NRM for the network resource. WSDL Wrapper is in between the various resource interfaces and the Main Module. GridARS applies a common advance reservation protocol for reservation, modification, and release, and different resource parameter representations for each resource, because some resource representations such as JSDL[10] have already been standardized.

Main Module enables a polling-based 2PC reservation process for reservation, modification, and release. When a client invokes the reserve operation, Main Module returns a response to the client in a non-blocking manner, and sends the reserve request to resource schedulers or to the GridARS-Coscheduler. After pre-reservation has finished, it completes the reservation using the client commit request. A non-blocking manner is important for distributed systems. It avoids hang ups because of server or client side troubles, and enables recovery of each process from the failure, easily. Main Module also checks the status of reserved resources managed by the resource scheduler periodically in a polling-based manner, so that the client can get the status via the WSRF interface.

Resource Manager Wrapper provides an API for the GridARS-Coscheduler or resource schedulers. Implementing this API, existing schedulers can provide a GridARS WSRF interface without complicated WSRF coding.

4 Design and Implementation of GridARS-WSRF

The advance reservation protocol of GridARS-WSRF is based on GNS-WSI (Grid Network Service - Web Services Interface)[11] version 2 (GNS-WSI2)[12]. GNS-WSI has been defined by the G-lambda project[13], which is a collaboration of AIST, KDDI R&D Laboratories, NTT, and NICT. It is a web services-based interface for network resources for Grid middleware and applications. While the version 1 is based on pure web services, GNS-WSI2 is based on WSRF.

GridARS-WSRF provides the following services:

ReservationFactoryService Receives registration requests to book Grid resources. It also returns information on resources available on the Grid.

ReservationService Receives reservation, modification, and release requests. It also manages current status of reserved resources.

ReservationCommandService Supports 2PC. It manages the status of pre-reserve, -modify, and -release processes, and abort or commit for each process by order of users.

ReservationResource and *ReservationCommandResource* are service instances for *ReservationService* and *ReservationCommandService* for each user request, respectively.

Table 1. Service operations related to reservation, modification, and release.

Operation name	Action	Input / Output
	ReservationFactoryService	
create	Creates *ReservationResource*	- / *rsvEPR*
getAvailableResources	Provides available resource information	conditions / available resource information
	ReservationService (Accessed using *rsvEPR*)	
reserve	Makes resource reservation	Requirements on resources and reservation time / *cmdEPR*
modify	Modifies reserved resources	Requirements on resources and reservation time / *cmdEPR*
release	Releases reserved resources	- / *cmdEPR*
getReservationStatus	Returns reserved resource status	- / reserved resource status
getResourceProperty (GridResources)	Returns reservation result	Resource property name / Reserved resource information
	ReservationCommandService (Accessed using *cmdEPR*)	
commit	Completes reserve/modify/ release process	- / -
abort	Destroys reserve/modify/ release process	- / -
getReservation-CommandStatus	Returns current status of (pre-)reserve/modify/release	- / status of the pre-process

4.1 Service Operations

Table 1 shows service operations related to reservation, modification, and release for each GridARS-WSRF service. *ReservationFactoryService* creates *ReservationResource* which manages each set of reservation information and provides a query operation which provides information on available resources. The *create* operation returns EPR(End Point Reference) to the created *ReservationResource* . We call this EPR *rsvEPR*.

ReservationService provides operations for resource reservation / modification / release and acquisition of reserved resource status and reserved resource information. For *reserve* and *modify*, *ReservationService* receives requirements on resources, such as the number of clusters and CPUs, and bandwidth and reservation times, such as duration, deadline, or exact start and end time. At this point, *ReservationService* just returns an EPR called *cmdEPR* for *ReservationCommandResource* which manages the reserve / modify / release process. *reserve*, *modify*, and *release* are triggers of each command, and the actual process is managed by *ReservationCommandResource*.

ReservationCommandService provides notification of each command status and completes or destroys the command by order of the user. *ReservationCommandService* enables the 2PC WSRF reservation process.

4.2 Resource status transition and advance reservation protocol

ReservationStatus is a property of *ReservationResource* and represents the current reservation status for each reservation request. The *ReservationStatus* transition process is shown in Fig. 5. The *ReservationStatus* transition process consists of the following:

Created *ReservationResource* is created.

Reserved Requested resources are booked.

Activated The resources are activated.

Released The resources are released.

Error Errors have occurred.

create, *reserve*, *modify*, and *release* in Fig. 5 indicate operations of Table 1 invoked by a client. S and F represent success and failure or destruction by the client of each command. The gray squares represent status changes at the server side.

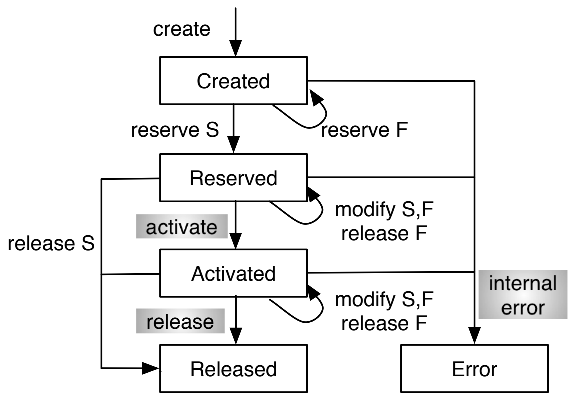

Figure 5. The ReservationStatus transition process.

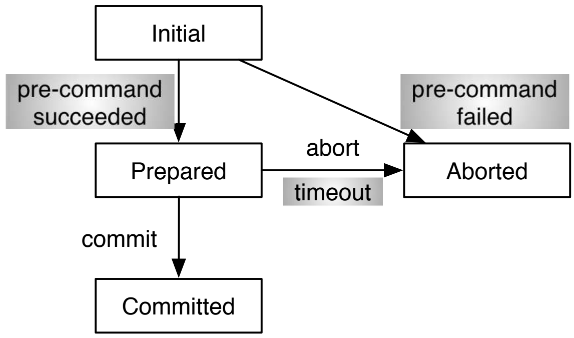

Figure 6. The ReservationCommandStatus transition process.

ReservationCommandStatus is a property of *ReservationCommandResource* and represents the current command status of each *ReservationCommandResource* created by a reservation-related operation such as *reserve*, *modify*, or *release*. The *ReservationCommandStatus* transition process is shown in Fig. 6. The *ReservationCommandStatus* transition process consists of the following:

Initial *reserve*/*modify*/*release* command has been sent to an actual resource manager, but the request has not been completed yet.

Prepared The requested command has been prepared.

Committed The command has been completed.

Aborted The requested resources are not available or the pre-command has expired.

commit and *abort* in Fig. 6 are invoked by the client, and the gray squares also represent status changes at the server side. After *ReservationCommandStatus* has changed to *Prepared*, the client invokes *commit* and *abort*.

We use a modified two-phase commit protocol. Fundamentally, a two-phase commit is a blocking protocol. If a coordinator fails after a *reserve* request, *ReservationCommandStatus* may be left in the *Prepared* state until the coordinator is repaired and the requested resources are blocked for that duration. Moreover, a coordinator and its cohorts are loosely coupled on the Grid, and the coordinator may not issue a *commit* or *abort* request.

We applied an automatic “time out” to the transit from *Prepared* to *Aborted*. In our system, *Prepared* waiting for a commit or abort request times out at $T_{timeout}$ as follows:

$$T_{timeout} = T_{transit} + \epsilon \quad (1)$$

$T_{transit}$ indicates the state transit time from *Initial* to *Prepared*.

4.3 Protocol Sequence of the Advance Reservation Process

We describe the protocol sequence of our advance resource reservation process in Fig. 7. As described in Section 3.2, each operation is non-blocking and based on a polling method.

User calls the *create* operation provided by GRS *ReservationFactoryService*, *ReservationResource* is created and the EPR (*rsvEPR*) is returned to User. After User calls *reserve* using *rsvEPR*, GRS starts to co-allocate the requested resources.

GRS collects available resource information, such as CPUs and bandwidth, by the *getAvailableResources* operation provided by RMs. Using the information obtained, GRS selects suitable resources and co-allocates the resource in coordination with related RMs based on distributed transactions. The bold lines in Fig. 7 represent a simultaneous process by transactions between GRS and RMs.

The reservation process between GRS and each RM is performed in the same manner between User and GRS.

After all of related RMs’ *ReservationCommandStatus* have been changed to *Prepared*, GRS’s *ReservationCommandStatus* is changed to *Prepared* and GRS waits

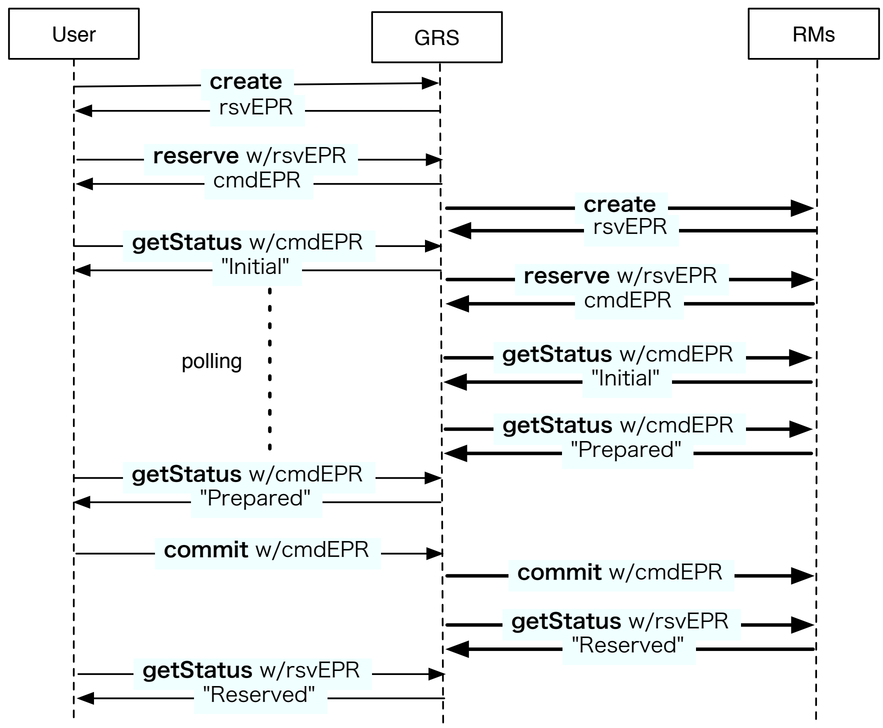

Figure 7. Protocol sequence of advance reservation process.

for a user *commit* or *abort* request. If User detects a *Prepared* status, User sends GRS the *commit* request and then GRS sends *commit* to the related RMs.

After the reservation process has completed at the related RMs, *ReservationStatus* of GRS and the appropriate RMs is changed to *Reserved*. User can search for success for the resource reservation to check the *ReservationStatus* via the *getReservationStatus* operation. Then, User acquires the reserved resource information.

4.4 Reference Implementation of GridARS-WSRF

We have developed a reference implementation of GridARS-WSRF named GridARS-WSRF/GT4 using Globus Toolkit4 (GT4). GridARS-WSRF/GT4 allows user authentication and authorization by GSI (Grid Security Infrastructure) as provided by GT4. GSI supports capabilities of authentication by certificates based on PKI (Public Key Infrastructure) and authorization by the `grid-mapfile` which maps global user name in the certificate on local user name. GRS also adopts a GSI delegation capability and books resources by each user authority.

We apply JSDL for computing resources and GNS-WSI2 for network resources, and extend them to represent advance reservation requirements.

5 Performance Measurement

The elapsed time of simultaneous resource reservation processes based on distributed transactions, compared to the number of RMs is shown in Fig. 8, Fig. 9, and Fig. 10. In these experiments, we emulate an actual Grid environment in our cluster, where all the hosts of GRS and eight RMs are deployed. User is located on the GRS host. Latencies between the hosts of GRS and RMs are 200 [us] in this cluster. In the experiments in Fig. 9 and Fig. 10, we configured additional 186 [ms] latencies on the paths to one RM or all RMs. 186 [ms] equals the latency between Tokyo and North Carolina, where the GRS and one of the RMs in US were located in the experiment described in Section 6. It takes 2 [sec] for each pre-reservation and 1 [sec] for completion of each requested reservation at each RM.

For all graphs, the horizontal axis indicates the number of RMs invoked simultaneously in a reservation re-

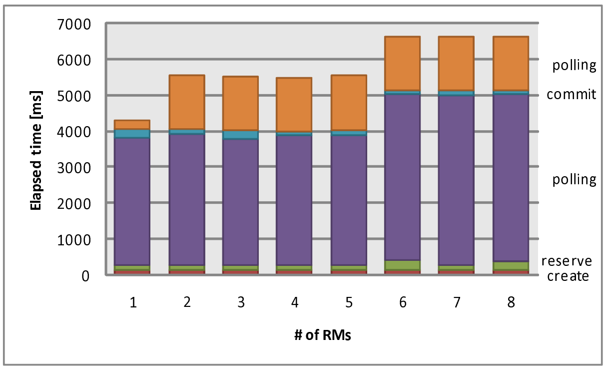

Figure 8. Elapsed time of simultaneous resource reservation processes (no additional latency).

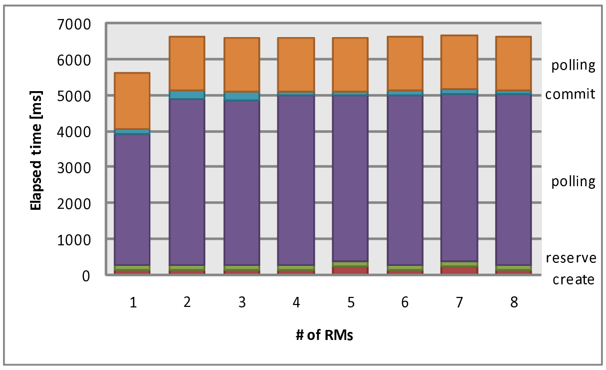

Figure 9. Elapsed time of simultaneous resource reservation processes (additional latency on the path to RM1).

quest, and the vertical axis indicates the elapsed time of the entire resource reservation process. Details of elapsed times are shown on the right hand side. create / reserve / polling / commit / polling in all graphs correspond to User's invocation in Fig. 7. All of the results show the shortest elapsed time for ten trials over WSRF/GSI, respectively.

Comparing the three graphs, the elapsed times of Fig. 9 and Fig. 10 are comparable, and longer than those of Fig. 8; this is because the latest reservation process at an RM determines the total elapsed time in transactions. On the other hand, when the number of RMs increases, the elapsed times increase because of the load at GRS, but they are around 6.7 [sec]. Therefore, the GridARS co-allocation framework works efficiently on Grids on which GRS and RMs are widely-distributed.

6 Case Study: a Trans-pacific Experiment using GridARS

We conducted a demonstration [14] at GLIF2006[15] and SC06[16]. In this demonstration, a user booked trans-pacific computing and network resources managed by different organizations, and we performed operations an actual parallel applications over the reserved resources. The demonstration was in cooperation with G-lambda and the EnLIGHTened Computing project[17].

In this experiment, a user submits requirements on resources from the portal system, GridARS makes corresponding reservations, and then the user invokes a parallel application via WS GRAM of GT4. The parallel application starts at the reserved time, automatically. We use QM/MD simulation developed using GridMPI[18] for the application program.

QM/MD simulation simulates a chemical reaction process based on the Nudged Elastic Band (NEB) method[19]. In this simulation, the energy of each image is calculated by combining classical molecular dynamic (MD) simulation with quantum mechanics (QM) simulation, in parallel. MD and QM simulations were performed on distributed clusters in Japan and the US using GridMPI, which is a Grid-enabled reference implementation of MPI.

The experimental environment is as follows:

- # of sites (clusters) = 10 (7 sites in Japan and 3 sites in the US)
- # of network domains = 4 (3 domains in Japan and 1 domain in the US)
- CRM composition : GridARS-WSRF/GT4, PluS[20] and GridEngine[6] (Japan), Maui[21] and TORQUE[7] (US)
- NRM composition : NRMs developed by KDDI R&D Labs, NTT, *EnLIGHTened Computing* , and AIST, respectively. *EnLIGHTened Computing* and AIST NRMs were developed using GridARS-WSRF/GT4.

We booked computing and network resources in the US via HARC (Highly-Available Resource Co-allocator)[22] developed by EnLIGHTened. The EnLIGHTened and G-lambda teams developed wrappers to enable interoperability across our middleware stacks, so that GRS could book resources in the US with our

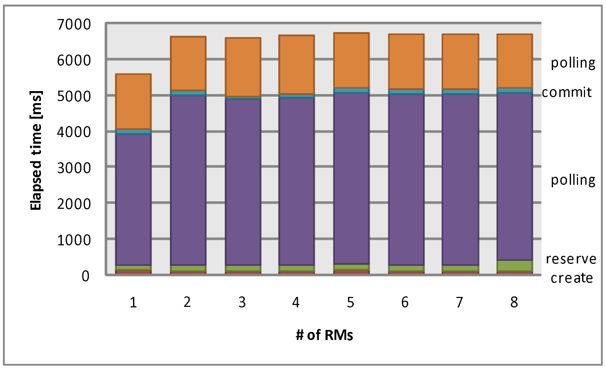

Figure 10. Elapsed time of simultaneous resource reservation processes (additional latencies on the paths to all RMs).

distributed transactions. PluS and Maui are plugin schedulers which allow advance reservation on existing batch queuing systems.

Fig. 11 shows the reservation resource monitor service display and the simulation results output at the experiment.

In this demonstration, we sent 10 [min] reservation requests, submitted a QM/MD simulation into local scheduler queues in the reserved sites, and performed the simulation, continuously. Although the reservation cycle is shorter than that of general use cases, GridARS worked stably during the demonstrations.

7 Related Work

There have been several global schedulers which allow metacomputing over distributed computing environment. In Moab Grid Suites[4], the Moab Grid Workload Manager can co-allocate distributed computing resources managed by the Maui Cluster Scheduler and TORQUE Resource Manager. Moab is a commercial Grid scheduling suite, and it also provides monitoring and reporting tools and a portal system for end users In general use, only administrators can make reservations, but users can submit a reservation request and their jobs from the portal.

CSF4 (Community Scheduler Framework)[5] developed using GT4 is a WSRF-based scheduling framework for computing resources. The CSF MetaScheduler can submit user jobs to queuing systems, Platform LSF[23], GridEngine, and Open PBS[24]. CSF supports an advance reservation capability for LSF clusters. CSF is open source and provides a Portlet GUI,

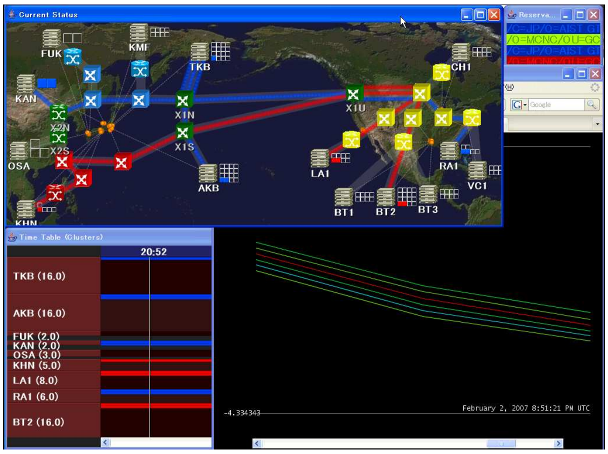

Figure 11. The reservation resource monitor service display and the simulation results output at the experiment.

but LSF is a commercial queuing system.

GUR[25] is a global scheduler which supports advance reservation. It is offered in cooperation with the Catalina external scheduler, and can work with TORQUE and LoadLeveler. GUR finds and books available resources to communicate with Catalina schedulers, one by one. Communication between GUR and Catalina is SSH or GSI-enabled SSH.

While Moab, CSF, and GUR support only computing resources, the VIOLA MetaScheduling Service (MSS)[26] can co-allocate both computing and network resources as well as work with GridARS. MSS works on UNICORE[27]-based Grid environments. The communication between MSS and the other components will be based on WS-Agreement[28] for establishing agreement between a service provider and consumer.

HARC developed by the EnLIGHTened Computing project is a co-allocation system, which consists of Acceptors and Resource Managers. HARC applies the Paxos commit protocol[29] to enhance fault-tolerancy of the Acceptor (coordinator) side. Each requirement on resources is represented by an XML documents and sent to the HARC Acceptors and Resource Managers by REST-styled HTTP messaging.

However, there are no other co-allocation systems which support a safe transaction process by 2PC over the standard WSRF interface and that satisfies all the requirements as described in Section 2.

8 Conclusions

We propose the GridARS Grid co-allocation framework for management of various distributed resources such as computers and network, and we developed a general 2PC advance reservation protocol over WSRF.

The GridARS co-allocation architecture consists of Global Resource Scheduler (GRS) and Resource Managers (RM) and automatically co-allocates required resources, simultaneously. The main components of GridARS are GridARS-Coscheduler and GridARS-WSRF. GridARS-Coscheduler finds suitable resources for each user and co-allocates the resources based on distributed transactions. GridARS-WSRF is an interface module for the proposed advance reservation protocol.

Using a reference implementation, called GridARS-WSRF/GT4, we investigated the basic performance of the 2PC reservation process over WSRF/GSI on emulated Grid environments. The results showed that the GridARS co-allocation framework and the simultaneous reservation process worked efficiently on Grids on which GRS and RMs are widely-distributed.

Also, we described a case study of Grid co-allocation for transpacific computing and network resources using GridARS-WSRF/GT4. The experiment shows GridARS can co-allocate distributed computing and network resources managed by various multiple-domain local schedulers, stably.

For future work, we plan to make the reservation protocol more practical and investigate suitable co-allocation algorithms for multiple resources. We also plan to collaborate with other Grid co-allocation systems, such as VIOLA MSS.

Achknowledgements

We thank all of the members of G-lambda and the EnLIGHTened Computing project. This work is partly funded by the Science and Technology Promotion Program's "Optical Paths Network Provisioning based on Grid Technologies" of MEXT, Japan.

References

[1] TeraGrid: `http://www.teragrid.org/`.

[2] OASIS Web Services Resource Framework (WSRF) TC: Web Services Resource 1.2 (WS-Resource) Committee Specification (2006).

[3] Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems, *IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779*, pp. 2–13 (2005).

[4] Moab Grid Scheduler (Silver) Administrator's Guide, version 4.0: `http://www.clusterresources.com/products/mgs/docs/`.

[5] Community Scheduler Framework: `http://sf.net/projects/gcsf`.

[6] Grid Engine: `http://gridengine.sunsource.net/`.

[7] TORQUE Resource Manager: `http://www.clusterresources.com/resource-manager.php`.

[8] Ganglia Monitoring System: `http://ganglia.info/`.

[9] OASIS Web Services Notification (WSN) TC: Web Services Base Notification 1.3 (WS-BaseNotification) Public Review Draft 02 (2005).

[10] A. Anjomshoaa and F. Brisard and M. Drescher and D. Fellows and A. Ly and S. McGough and D. Pulsipher and A. Savva: Job Submission Description Language (JSDL) Specification v1.0 (2005).

[11] Takefusa, A., Hayashi, M., Nagatsu, N., Nakada, H., Kudoh, T., Miyamoto, T., Otani, T., Tanaka, H., Suzuki, M., Sameshima, Y., Imajuku, W., Jinno, M., Takigawa, Y., Okamoto, S., Tanaka, Y. and Sekiguchi, S.: G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS, *Future Generation Computing Systems*, Vol. 22(2006), pp. 868–875 (2006).

[12] Takefusa, A., Hayashi, M., Hirano, A., Okamoto, S., Kudoh, T., Miyamoto, T., Tsukishima, Y., Otani, T., Nakada, H., Tanaka, H., Taniguchi, A. and Sameshima, Y.: GNS-WSI2 Grid Network Service - Web Services Interface, version 2, OGF19, GHPN-RG (2007).

[13] The G-lambda project: `http://www.g-lambda.net/`.

[14] Thorpe, S. R., Battestilli, L., Karmous-Edwards, G., Hutanu, A., MacLaren, J., Mambretti, J., Moore, J. H., Sundar, K. S., Xin, Y., Takefusa, A., Hayashi, M., Hirano, A., Okamoto, S., Kudoh, T., Miyamoto, T., Tsukishima, Y., Otani, T., Nakada, H., Tanaka, H., Taniguchi,

A., Sameshima, Y. and Masahiko Jinno: G-lambda and EnLIGHTened: Wrapped In Middleware Co-allocating Compute and Network Resources Accross Japan and the US, *Submitted to GridNets2007*.

[15] GLIF: Global Lambda Integrated Facility: http://www.glif.is/.

[16] SC06: http://sc06.supercomputing.org/.

[17] The EnLIGHTened Computing project: http://enlightenedcomputing.org/.

[18] GridMPI: http://www.gridmpi.org/.

[19] Ogata, S., Shimo, F., Kalia, R., Nakano, A. and Vashisha, P.: Hybrid Quantum Mechanical/Molecular Dynamics Simulations on Parallel Computers: Density Functional Theory on Real-space Multigrids, *Computer Physics Communications*, p. 30.

[20] Nakada, H., Takefusa, A., Ookubo, K., Kishimoto, M., Kudoh, T., Tanaka, Y. and Sekiguchi, S.: Design and Implementation of a Local Scheduling System with Advance Reservation for Co-allocation on the Grid, *Proceedings of CIT2006* (2006).

[21] Maui Cluster Scheduler: http://www.clusterresources.com/pages/products/maui-cluster-scheduler.php.

[22] HARC: The Highly-Available Robust Co-allocator: http://www.cct.lsu.edu/~maclaren/HARC/.

[23] Zhou, S.: LSF: Load sharing in large-scale heterogeneous distributed systems, *Proceedings of Workshop on Cluster Computing* (1992).

[24] OpenPBS: http://www.openpbs.org/.

[25] Yoshimoto, K., Kovatch, P. and Andrews, P.: Co-scheduling with User-Settable Reservations, *Job Scheduling Strategies for Parallel Processing*, Springer Verlag, pp. 146–156 (2005). Lect. Notes Comput. Sci. vol. 3834.

[26] Barz, C., Pilz, M., Eickermann, T., Kirtchakova, L., Waldrich, O. and Ziegler, W.: Co-Allocation of Compute and Network Resources in the VIOLA Testbed, TR-0051, CoreGrid (2006).

[27] UNICORE: http://www.kfa-juelich.de/unicore/.

[28] Andrieux, A., Czajkowski, K., Dan, A., Keathey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S. and Xu, M.: Web Services Agreement Specification (WS-Agreement). https://forge.gridforum.org/sf/docman/do/downloadDocument/projects.graap-wg/docman.root.current_drafts/doc6090 (2005).

[29] Gray, J. and Lamport, L.: Consensus on Transaction Commit, MSR-TR-2003-96, Microsoft Research (2004).