International Journal of Grid and Distributed Computing
Vol. 1, No.2, March, 2009

VJM: A Novel Grid Resource Co-Allocation Model for Parallel Jobs

Xiaohui WEI, Zhaohui DING, Shaocheng XING, Yaoguang YUAN
College of Computer Science and Technology, Jilin University, ChangChun, P.R.C.
weixh@jlu.edu.cn, zhaohui.ding@email.jlu.edu.cn

Wilfred W. LI
San Diego Supercomputer Center, University of California
wilfred@sdsc.edu

Abstract

The resources in a grid are distributed in multiple autonomic domains. Hence, it's always a time consuming process to perform the resource co-allocation in grids especially for parallel jobs due to the diversity of local domain policies and the dynamic resource workloads. Even worse, the resource competing among multiple parallel jobs may result in resource allocation deadlock. In this paper, a novel synchronized resource co-allocation model for cross-domain parallel jobs, called virtual job model (VJM), is proposed. VJM uses virtual jobs to co-reserve the earliest available resources for real parallel jobs with considering the local resource policies and the current resource availability in multiple domains. The co-reservation algorithm implemented in VJM is not only able to detect potential deadlocks, but able to relieve the resource competing via the resource reorganization mechanism. At last, we demonstrate that VJM is able to reduce the time cost of resource co-allocation significantly via experiments.

1. Introduction

Grid computing technology aims to realize the global resource sharing in the distributed, heterogeneous and autonomic environment, and assemble these resources to meet the requirements of the applications. With the rapid development of the grid computing technology, many people began to run large scale parallel applications on grids.

The resources in a grid are actually owned by different domains (organizations) and typically managed by local schedulers. Hence, the sub jobs of a grid parallel application will be submitted to multiple local schedulers first. Each local scheduler will decide when to start the sub jobs according to its scheduling policy and the local resource availability respectively. The whole parallel job cannot start up until all the sub jobs are scheduled to run by local schedulers. Therefore, the resource co-allocation for a parallel job could last considerable time or even fail. And the diverse local policies enforced by different domains and the dynamic changed resource availability make the resource co-allocation more challenge.

In traditional single domain environments, PVM [1], MPI [2] and OpenMP have been used widely for years. In grid environments, MPICH-G2 [3] is almost the only available parallel programming tools for now. However, the resource allocation in MPICH-G2 is low efficient and may cause deadlock.

Firstly, the host selection algorithm of MPICH-G2 is static. A static machine file need be predefined by user or admin, in which the potential available CPU number and the master hosts' IP addresses for each cluster (autonomic domain) are configured. MPICH-G2 just uses Round-Robin method to schedule parallel jobs to the clusters in the file.

Secondly, the resource co-allocation stage is mixed with the job execution. The resource co-allocation protocol used by MPICH-G2, DUROC protocol [4], is implemented inside MPI library like MPI_INIT(). MPI_INIT() is required to be the first MPI function called by MPICH-G2 programs. Therefore, the resource co-allocation cannot success until all the sub jobs startup and execute MPI_INIT() function. If one sub job fails to start, the DUROC protocol will fail. However, if all the sub jobs have startup to run already, the resource co-allocation seems unnecessary to perform. It is a dilemma caused by MPICH-G2's implementation.

In this paper, we proposed a novel synchronized grid resource co-allocation model, the Virtual Job Model (VJM). In the resource co-allocation phase, VJM sends virtual jobs (VJobs for short) instead of real parallel jobs to grid resources. Each VJob will report back to VJob manager after it successfully reserves the resource. Once enough VJobs return, the real jobs will be launched.

Many factors in local domains like local policies and workloads have strong impacts on grid jobs' execution. And some factors like scheduling policies are hard to evaluate. Based on queuing theory, VJM evaluates the overall capability that a local resource domain can provide through its history data, such as the average waiting time of virtual jobs in the local queue, and the average job execution time and so on. Based on the evaluation, VJM will decide which clusters should be selected for a parallel job and how to distribute the VJobs among them. Hence, VJM is able to co-reserve the earliest available resources in heterogeneous and dynamic grid environments. Moreover, the deadlock detection and resource reorganization mechanism is also implemented by VJM to relieve the resource competing among jobs.

In this paper, VJM works with CSF4 [5] meta-scheduler to perform the resource co-allocation for cross domain parallel jobs. Actually VJM was implemented as libraries so that it can be linked with other meta-schedulers or resource brokers. The rest of this paper is organized as follows: section 2 discusses the related works; section 3 presents the architecture of VJM, and introduces the design and implementation of VJM in detail; in section 4, we evaluate the performance of VJM through experiments. At last, we conclude the paper and discuss the future work.

2. Related Works

[4] proposed two resource co-allocation protocols, GRAB and DUROC, based on two-phase commit protocol. GRAB works in atomic transaction mode. The protocol succeeds if all resources required by the application are allocated. Otherwise, the request fails and none of the resources are acquired. DUROC improves GRAB by using interactive transaction mode. The resources were classified into three types: required, interactive and optional. During the resource co-allocation, the optional resources do not participate in the commitment procedure: failure or timeout is ignored; the failure or timeout of the interactive resources result in a callback to the application, the protocol will try to allocate other resources as substitute; only when the required resources are not met, the protocol fails.

DUROC has also been used to construct of other Globus components. MPICH-G uses DUROC to start the elements of an MPI job. In this case, all DUROC calls are hidden in the MPI library. The DUROC protocol will be executed by each MPI sub job right after start up. Such implementation will cause the problems as we mentioned before. We argue that the

resource co-allocation should not be tied with the user applications but with the job scheduling process.

Neither GRAB nor DUROC can guarantee resource co-allocation successfully. The breakdown or heavy workload of some grid sites will cause the protocol's failure. The co-allocation and the synchronization of GRAB protocol often take tens of minutes or even more. And even worse, multiple parallel jobs may cause resource allocation deadlock. [8, 9] attribute the resource co-allocation deadlock to "the Dining Philosophers" problem and proposed global order-based prevention protocol.

To ensure the resource co-allocation success, [10] proposed GARA protocol based on advance reservation. GARA protocol splits the process of resource co-allocation into two phases: reservation and allocation. GARA need the user to indicate the required resources and the time of reservation, for example the begin time and end time of the reservation, to make advanced reservations on the grid sites. The reservation created in the first phase guarantees that the subsequent allocation request will succeed. Other works based on advance reservation include [11, 12].

However, users usually have little knowledge on the resource availability of the grid resources, it is hard for them to give out a good begin time. In [10], the begin time of a reservation was set to a random number between 0~2 hours, it is not reasonable. The end time is also hard to be given as well. As the users don't know their job's accurate runtime in most cases, they usually have to set an upper limit value to ensure the job's completion. This will aggravate the competing and conflict of resource allocation. The statistic data shows that the more reservation requests in a system, the longer average waiting time will be. Moreover, not all the cluster schedulers support advance reservation feature, such as SGE [13] and Open PBS [14] etc.

Same as GARA, VJM also separate the resource co-allocation phase from the job execution. However, the user does not need to specify the time of the resource reservation. VJM will automatically reserve the earliest available resources for the real jobs. VJM does not require the local schedulers to support resource advance reservation either. Hence, VJM can work with all kinds of local schedulers as long as they support GRAM protocol.

Actually a set of virtual jobs managed by VJM dynamically construct a cross-domains virtual cluster for a large scale parallel application to execute. In this context, VJM's virtual job mechanism is similar to the Condor Glidein mechanism [6], which is able to enlarge a Condor pool by dynamically sending the Condor daemon binaries to remote grid sites. After that, the real condor jobs can be started on the remote site. Another similar work is LSF MultiCluster [7].

3. Virtual Job Model

The parallel applications require synchronized resource co-allocation that can be satisfied only by simultaneously acquiring multiple resources from distributed locations. The diverse local domain policies and the unpredictable workload on the grid sites may result in long synchronization time. VJM uses VJobs to hold the resource for real parallel jobs in advance and synchronize the resource co-allocation. In this section, we will introduce the design and the implementation of VJM. Especially, the co-allocation algorithm and the deadlock detection mechanism will be discussed in details. The below sub section is the overview of VJM architecture.

3.1. The VJM Architecture

VJM is designed as a grid resource co-allocator that is running between resource consumers and resources providers. Figure 1 illustrates the work principle of virtual job and the architecture of VJM.

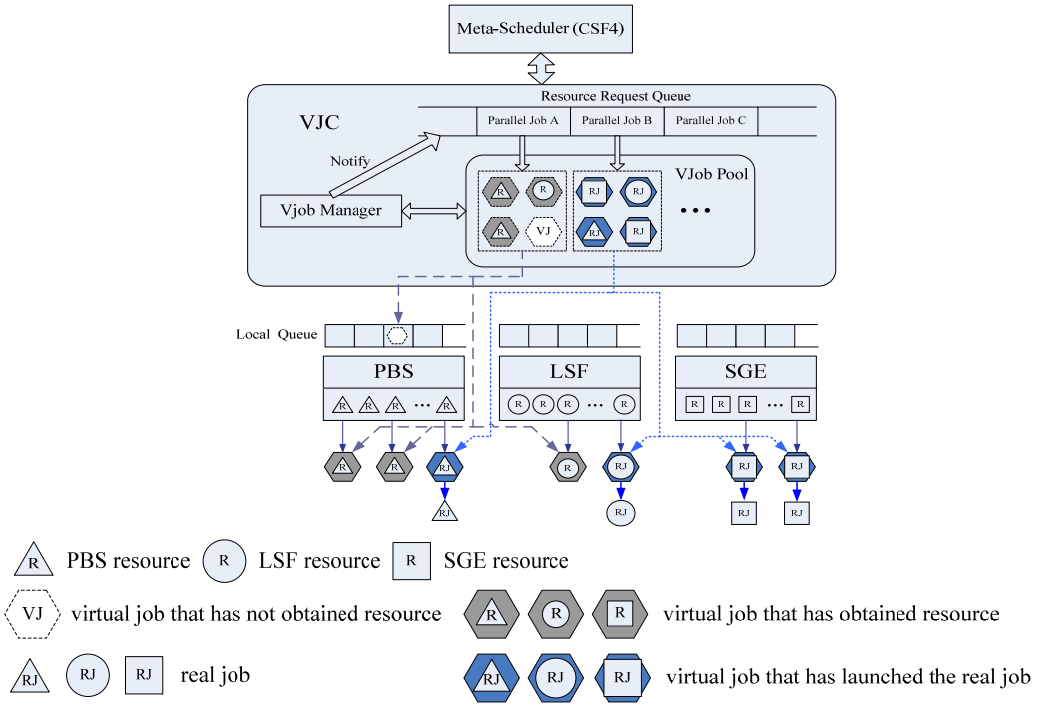

Figure 1. The VJM Architecture

VJM divides the process of parallel job into resource co-allocation phase and the job launch phase. In the resource co-allocation phase, VJM sends VJobs instead of real parallel jobs to grid resources via GRAM protocol (the resource selection algorithm will be described in section 3.2). A virtual job has same resource requirements with its corresponding real sub job. In Fig. 1, the VJobs will be dispatched to multiple distributed clusters. After that the VJobs will be queued and scheduled like normal serial jobs by local schedulers. When the virtual job startup, it will report back to VJC (virtual job center) that the resource for the sub job has been reserved. Then, such VJob will be registered in VJob Pool (virtual job pool) as a virtual resource. The virtual resource can be from various kinds of cluster (like LSF, PBS and SGE), and the API provided by the VJob Pool to the resource consumer is uniform, which hides the resource heterogeneity.

As all the virtual jobs registered successfully (i.e. all the virtual jobs obtained resource), VJC dispatches the real jobs to their corresponding virtual jobs. Then the virtual jobs transfer the resource access to the real jobs and synchronously launch the parallel job. VJM has no extra requirements for local sites. VJM neither require the local schedulers to support resource advance reservation nor need to install extra components on local grid sites.

VJC is the core component of VJM, it is in charge of the launching, monitoring and controlling of virtual jobs. It also maintains the VJob Pool that stores all the registered

resources information. According to the registered virtual job's lifetime, VJC detect whether there is potential resource co-allocation deadlock. A resource reorganization algorithm is also implemented to alleviate the resource waste caused by deadlock, which will be discussed in detail in section 3.3.

Although the resource co-allocation protocol usually works closely with job scheduling, the implementation of VJM is independent of any job scheduler. VJM provides a set of external APIs so that VJM can be embedded into a meta-scheduler (such as CSF4) or a resource broker to improve job scheduling. On the other hand, VJM can also be deployed as an independently co-allocation service in a grid system.

3.2. The Resource Selection Algorithm

The grid resources are distributed in many autonomic domains, in which various local policies are enforced. Here we distinguish two kinds of jobs for a domain, local jobs and external jobs, as they may not be treated equally by local policies. Local jobs are the jobs belong to local users, and the external jobs are the jobs submitted outside of the domain by grid users. Local jobs usually have higher priorities to access local resources than external jobs.

VJobs are the special external jobs to co-reserve resources. The less waiting time in the local queue for VJobs means the lower cost of the resource co-allocation. In particular, the VJob that acquires the required resource latest decide the parallel job's start time. Therefore, VJM should select such clusters for a parallel job that the resource co-reservation has the minimal makespan time. However, the domain policies are generally unknown and hard to measure to the grid users or meta-schedulers. Hence, VJM just regards these grid sites as the black boxes, and evaluate their resource capabilities via the average waiting time of the external jobs from history.

Besides makespan time, the number of clusters involved in the computing must be considered as well. Due to the communication cost, the users do not want their parallel applications span too many clusters. Therefore, VJM sets an upper limit of cluster number (M) for each parallel job. See below,

The eq.1 computes the expected waiting time of virtual job j to obtain the resource on Cluster C_i.

$$T_{avail}(C_i, VJob_j) = T_w(C_i) - T_{fjw}(C_i) + \Delta t(C_i) * QL(C_i) + \Delta t(C_i) * VJ_{seq}(C_i, j) \quad \text{Eq.1}$$

Where,

$Tw(C_i)$: the average waiting time of external jobs on Cluster C_i;

$Tfjw(C_i)$: the waiting time of the earliest submitted external job in the waiting queue of the Cluster C_i;

$QL(C_i)$: the queue length of external jobs on Cluster C_i , i.e., the number of the waiting external jobs on Cluster C_i;

$\Delta t(C_i)$: the average interval of two external jobs obtaining the resource on Cluster C_i;

$VJSeq(C_i,j)$: the number allocated to Cluster C_i of the $(j-1)$ virtual jobs.

The cluster selection algorithm:

(1) Pick the virtual job j from the undecided jobs set and remove it from the set, compute T_{avail} (C_i, $VJob_j$) for each cluster in available cluster set C (set C includes all the available clusters at initial time) using Eq.1, choose the cluster whose T_{avail} (C_i, $VJob_j$) is smallest as the candidate cluster for virtual job j;

(2) Repeat (1) until the undecided virtual jobs set is empty. Then we get cluster set C' which is the candidate cluster set for virtual jobs. After that, we compare the cluster number, L, in set C' with the upper limit M preferred by the user. If $L<=M$ (the maximum clusters that the parallel job can span), the algorithm ends. Otherwise, go to (3);

(3) For now, we get the candidate cluster set $C'=\{C_1, \ldots C_m, \ldots C_L\}$ where $L>M$. For each C_i ($i=1 \ldots L$) in C', there is a virtual job set $VJ(i)'$ represents the resources need be reserved in Ci. Then remove the cluster Cm who has the smallest virtual job set from C', $C' = C' - C_m$; update the available clusters set $C=C'$, and set the undecided jobs set to $VJ(m)'$; go to (2).

When the algorithm finish, the C' is the final candidate cluster set for J_a, and how many resources need be reserved in each cluster C_m are also decided. After that, VJM will dispatch the virtual jobs to the candidate clusters. If all the virtual jobs successfully reserve the resources and register back to VJC, then the resource co-allocation for J_a succeeds. Otherwise, there could be a deadlock happened.

3.3. Deadlock Detection and Resource Reorganization

When the algorithm finish, the C' is the final candidate cluster set for J_a, and how many resources need be reserved in each cluster C_m are also decided. After that, VJM will dispatch the virtual jobs to the candidate clusters. If all the virtual jobs successfully reserve the resources and register back to VJC, then the resource co-allocation for J_a succeeds. Otherwise, there could be a deadlock happened.

In a grid environment, as the resources are not totally controlled by a centralized "global" scheduler, the deadlocks caused by resource competing are inevitable. In this section, we will discuss the VJM deadlock detection mechanism and the resource reorganization mechanism used by VJM to minimize the resource wasting caused by deadlocks.

Assume $VS(J_a) = \{vj1, \ldots, vjn\}$ is the virtual job set for real parallel job J_a, and $C'=\{C_1, \ldots C_m, \ldots C_L\}(L<M)$ is the final candidate cluster set for J_a. According to resource selection algorithm (Eq.1 in section 3.2), the expected waiting time (T_{avail} in Eq.1) for each virtual job in VS is evaluated by VJM. For deadlock detection, VJM will set the lifetime for $VS(J_a)$ as below,

$$T_{life\_time}(VS(J_a))=2*Max\{T_{avail}(vj1), \ldots, T_{avail}(vjn)\} \quad \text{Eq. 2}$$

T_{life_time} $(VS(J_a))$ is used as time out for deadlock detection. If Ja does not run before T_{life_time} $(VS(J_a))$, then VJM considers there may exist a potential deadlock for J_a. Although releasing Ja's partially reserved resources could be a simple solution to break the potential deadlock, but the cost is too high. The released resources could be captured by local domain jobs or other meta-schedulers, and VJM cannot benefit from the incomplete resource co-reservation. To minimize such cost, the resources reserved for a job can be shifted to another by VJM. We call such mechanism as resource reorganization. Before describing the resource reorganization mechanism, we will give out two definitions first.

Definition 1: For any two pending parallel jobs, J_a and J_b, both of their sub jobs are dispatched to cluster C. If the virtual jobs of J_a have acquire the resources on C, and the virtual jobs of J_b on C do not obtain all the required resources yet, we call this "$J_a > J_b$ on C" or "$J_b < J_a$ on C"

Definition 2: For any two pending parallel jobs, Ja and Jb, both of their sub jobs are dispatched to cluster C. If neither of J_a and J_b successfully acquire all the required resources on C, we call this "$J_a = J_b$ on C".

Resource Reorganization Algorithm

If Ja does not startup before T_{life_time} *(VS(Ja)),* then VJM will do the following actions,

1. Check out the "indirect" potential deadlocks between Ja and other VJM jobs

VJC will go though all the pending jobs in VJM to check out if the following condition is satisfied,

$$\exists (J_a, J_b, C_m, C_n) \rightarrow (J_a < J_b \text{ on } C_m, J_a > J_b \text{ on } C_n) \quad \text{Eq. 3}$$

If there exists a pair of clusters C_m and C_n, where "$J_a < J_b$ *on* C_m" and "$J_a > J_b$ *on* C_n", then we say there could exist an "indirect" resource allocation deadlock between J_a and J_b. The resource competing may result in neither of J_a and J_b can run. In this case, VJM will shift the resources reserved by J_a on cluster "C_n" to J_b. If there are multiple such J_b jobs exists on cluster "C_n", the algorithm will choose one with earliest submission time. After that, VJM will seek new clusters as substitute for all such "C_n" clusters for J_a. Then the candidate cluster C' and T_{life_time} *(VS(J_a))* will be set to new values by the resource selection algorithm. (Note that J_a will not release the resources on the clusters other than such "C_n".)

2. Check out the "direct" potential deadlocks between J_a and other VJM jobs

VJC will go though all the pending jobs in VJM to check out if the following condition is satisfied,

$$\exists (J_a, J_b, C_n) \rightarrow (J_a = J_b \text{ on } C_n) \quad \text{Eq. 4}$$

If there exists a cluster C_n, where "$J_a = J_b$ *on* C_n", then we say there could exist a "direct" resource allocation deadlock between J_a and J_b. The resource competing may result in neither of Ja and J_b can acquire all required resources in C_n. In this case, VJM will shift the resources reserved by J_a on cluster "C_n" to J_b. If there are multiple such J_b jobs exists on cluster "C_n", the algorithm will choose one with earliest submission time. After that, VJM will seek new clusters as substitute for all such "C_n" clusters for J_a. Then the candidate cluster C' and T_{life_time} *(VS(J_a))* will be set to new values by the resource selection algorithm. (Note that J_a does not release the resources on the clusters other than such "C_n".)

3. If neither Eq. 3 nor Eq. 4 satisfied, then the time out should be caused by the resource competing from local jobs or other meta-scheduler jobs.

In this case, VJM will divide the candidate cluster set C' of Ja into two sub set C'_{cmp} and C'_{in_cmp}, where C'_{in_cmp} consists of the clusters that the resources are not reserved completely and C'_{cmp} consists of the clusters that the resources have been reserved successfully. Assume $C'_{in\_cmp} = \{C_1, \ldots, C_{[m/2]}, \ldots, C_m\}$, and $C_1, \ldots, C_m$ are sorted in ascending order by the mean waiting time of the virtual jobs that do not acquire the resources successfully in that cluster. Then, VJM will release the reserved resources in cluster $\{C_{[m/2]}, \ldots, C_m\}$, and seek for new cluster as substitute. The candidate cluster C'

and T_{life_time} *(VS(Ja))* will be set to new values by the resource selection algorithm. The virtual jobs of J_a will continue to acquire available resources in cluster $\{C_1, \ldots, C_{[m/2]-1}\}$.

4. Application and Performance Evaluation

In this section, we evaluated the overheads of VJM in the first. After that, we compared the parallel jobs' co-allocation of VJM with DUROC protocol.

4.1. Overheads of VJM

The process of co-allocation of VJM involved additional overheads, which includes resource selection algorithm, the initialization of virtual jobs and virtual job manager and the communications between virtual job and virtual job manager, etc. Figure 2 illustrates the average time cost of all the VJM stages.

The initialization of VJM will only be executed once, so it can be ignored. The *resource selection* algorithm will be executed for each real job(*RJob*). Once startup, a virtual job(*VJob*) will do the initialization, which includes reporting the reserved resource information back to VJC and completing the registration. Both of the *RJobs* and *VJobs* are launched via GRAM protocol, which costs around 0.5 seconds. Hence, the extra overheads of VJM for a job should consist of *resource selection algorithm*, *VJob deliver*, *VJob initialization*. The average total cost is 1.67 seconds. Compared with the execution time of real job and the benefits for resource co-allocation, the overhead is acceptable. (Note that the cost of *RJob launch* should not be included as the real job will be executed any way.)

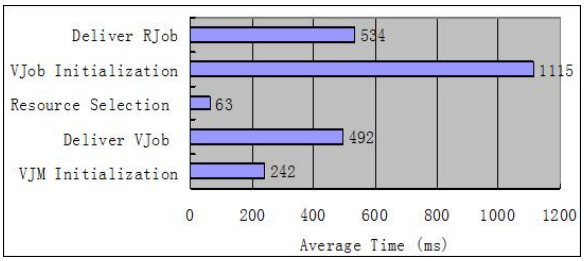

Figure 2. Extra Overheads of VJM

4.2. The co-allocation of VJM

To evaluate the performance of VJM, we compared VJM's average resource co-allocation time with DUROC. The computing environment is made up of 2 LSF clusters, Cluster A and Cluster B. The processor count of each cluster is 16. To simulate the difference of the workload of these clusters, the length of the schedule interval is used to simulate the job average waiting time, which can indicate the capability of external jobs execution of a cluster.

All the clusters are idle, and we set the schedule interval of A to 60 seconds and the schedule interval of B to a dynamic value from 10 to 120 seconds. The bigger the schedule interval, the more time the job will have to wait to be scheduled.

The test application is *cpi*, the MPICH-G2 version of calculation program of π. The required processor number of our test is 10. The jobs were submitted by CSF4 (for VJM) and MPICH-G2 (for DUROC).

The figure 3 shows the comparison of the resource co-allocation performance of VJM and DUROC in a dynamic environment. The execution of parallel job requires all the resources are synchronously co-allocated, so the co-allocation cost depends on the longest waiting time of all the sub jobs. As MPICH-G2 uses Round-Robin policies to allocate resource for the jobs, its co-allocation time keep even when the cluster B's mean waiting time become less than cluster A's. In this scenario, MPICH-G2's co-allocation time is decided by cluster A's mean waiting time, 60 seconds. On the contrary, VJM completes the resource co-allocation faster while decreasing cluster B's mean waiting time. From the tests, VJM's resource selection algorithm has a significant better performance than MPICH-G2 in a grid environment with dynamic and imbalanced workloads.

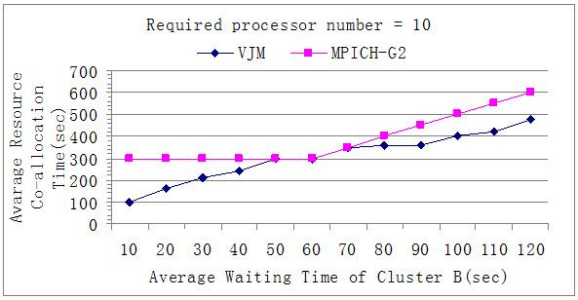

Figure 3. Resource Co-Allocation Time of VJM and MPICH-G2

5. Conclusion and the Future work

The paper proposed a novel synchronized resource co-allocation model VJM. Using virtual job mechanism, VJM is able to co-reserve the earliest available resources for real parallel jobs with considering the local resource policies and the current work loads in different domains. For now, VJM was implemented as a library and embedded in CSF4 meta-scheduler, but it can work with other resource brokers as well. The VJM resource selection algorithm is able to optimize the resource set of a parallel job by computing the waiting time of each sub job to minimize the whole resource co-allocation time. In addition, the co-reservation algorithm of VJM is not only able to detect potential deadlocks, but is able to relieve the resource competing via a resource reorganization mechanism.

The experiment results show that the resource co-allocation time using VJM is much shorter than using DUROC protocol in an imbalanced workload environment. Moreover, VJM is able to improve the execution of workflow jobs as well by reserving the resources in advance.

The current resource reorganization in VJM are constrained to the virtual jobs belong to same users due to the resource access control issue. In the near future, we are planning to remove such limitation so that it can support community users, which is widely used by some grid toolkits such as VOMS [16]. We are also planning to release a stable updated version of CSF4, and deploy VJM on some real grid testbeds like PRAGMA grid.

6. Acknowledgement

The authors would like to acknowledge support from the China NSF Grant No.60703024 for the CSF4 project and Jilin Department of Science and Technology Grant No.20070122 and 20060532.

7. References

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, B. Manchek and V. Sunderam PVM:Parallel Virtual Machine-A User's Guide and Tutorial for Network Parallel Computing MIT Press, Cambridge, MA, 1994.

[2] Message Passing Interface Forum, "MPI: A message-passing interface standard", Internat. J. Supercomput. Appl. 8(3/4), 165-414, 1994.

[3] N.T. Karonis, B. Toonen, I. Foster "A MPICH-G2: A Grid-enabled implementtation of the Message Passing Interface," Journal of Parallel and Distributed Computing, 2003.

[4] K. Czajkowski, I. Foster, and C. Kesselman. Resource Co-Allocation in Computational Grids. Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 219-228, 1999.

[5] X. Wei , Z. Ding and S. Yuan "CSF4: A WSRF Compliant Meta-Scheduler" International Conference 06' on Grid Computing and Applications, June 26-29, 2006.

[6] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. Concurrency and Computation: Practice and Experience, 17, 2005

[7] Web page, online, http://www.platform.com/Products/platform-lsf-multicluster.

[8] Driss Azougagh, Jung-Lok Yu, Seung Ryoul Maeng, "Resource Co-Allocation : A Complementary Technique that Enhances Performance in Grid Computing Environment", 11th International Conference on Parallel and Distributed Systems (ICPADS'05), vol. 1, July 2005.

[9] Jonghun Park. A Scalable Protocol for Deadlock and Livelock Free Co-Allocation of Resources in Internet Computing. 2003 Symposium on Applications and the Internet (SAINT'03), January 2003.

[10] W. Smith, I. Foster, V. Taylor. Scheduling with Advanced Reservations. Proceedings of the IPDPS Conference, May 2000.

[11] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan. Scheduling of parallel jobs in a heterogeneous multisite environment. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, Job Scheduling Strategies for Parallel Processing, pages 87-104. Springer Verlag, 2003.

[12] Dean Kuo , Mark Mckeown. Advance Reservation and Co-Allocation Protocol For Grid Computing. First International Conference on e-Science and Grid Computing (e-Science'05), December 2005.

[13] Sun Microsystems, Inc. Sun Grid Engine 5.3 Administration and User's Guide[M]. http://gridengine.sunsource.net/project/gridenginedownload/ SGE53AdminUserDoc.pdf, April, 2002.

[14] Specification Altair PBS Pro 5.3[M]. http://www.mta.ca/torch/pdf/pbspro54/pbsproers.pdf, March 2003.

[15] Z. DING, X. WEI, etc. "VJM- A Deadlock Free Resource Co-allocation Model for Cross Domain Parallel Jobs", HPCAsia2007, Korea, Sep 9-12, 2007.

[16] Alfieri, R., et al., From gridmap-file to VOMS: managing authorization in a grid environment. Future Generation of Computer Systems, 2005. 21: p. 549-558.

Authors

Xiaohui Wei, ACM/IEEE member, Ph.D. of Computer Software and Theory. He had worked with Platform Computing Inc. (Canada) from 1999 to 2004. Since 2003, he is a professor of the College of Computer Science and Technology in Jilin University, China. He is also the vice dean of the College of Computer Science and Technology and the director of the Grid Computing and Information Security Lab. His research interests include distributed system, grid computing, fault tolerance and information security.

Zhaohui Ding was born in 1979. He received the BS and MS degrees in Computer Science from Jilin University, in 2001 and 2006, respectively. He is currently a Ph.D. student of College of Computer Science and Technology of Jilin University, P.R.C. He is the administrator and the developer of the open source metascheduler project CSF4. His research fields include grid computing, meta-scheduling and distributed system.

Shaocheng Xing was born in 1983. He received the BS degrees in Computer Science from Jilin University in 2006. He is currently a master student of College of Computer Science and Technology of Jilin University, P.R.C. His research fields include grid computing and resource co-allocating.

Yaoguang Yuan was born in 1983. He received the BS degrees in Computer Science from Jilin University in 2006.He is currently a Master student of College of Computer Science and Technology of Jilin University, P.R.C. His research fields include grid computing and parallel job co-scheduling.

Wilfred Li received his Ph.D. in Biochemistry and Molecular Biology from the University of Southern California in 1994 where he studied the molecular chaperone GRP78/BiP as a model system for gene regulation and protein-DNA interaction. While a postdoctoral fellow at the University of California, San Diego (UCSD), he contributed to the understanding of protein phosphatases in cell growth and apoptosis in the JNK/SAPK pathway. He became a senior fellow in Bioinformatics at the San Diego Supercomputer Center, UCSD, in 1999. He is currently Executive Director of the National Biomedical Computation Resource (NBCR at http://nbcr.net). He is actively involved in the Pacific Rim Applications and Grid Middleware Assembly (PRAGMA at http://www.pragma-grid.net) activities. His current research interests include cluster and grid computing, cyberinfrastructure, high throughput proteome annotation and virtual screening, multiscale modeling, and data mining.