2009−ARC−182 (26)
2009−HPC−119 (26)
2009／2／28

Evaluation of Interactive Scheduling Architecture for GridRPC Applications

HAO SUN [†1] and KENTO AIDA[†2,†1]

GridRPC is known as an effective programming model to develop Grid applications. However, it still has difficulty to make efficient applications for non-expert users. For example, users need to monitoring remote computational resources and estimating application performance on selected computational resources. In this paper, we propose InterS, an interactive scheduling system for GridRPC applications. InterS provides three mechanisms, which help GridRPC application users to run their application in an efficient and robust way. First, the automatic scheduling mechanism provides resource allocation functionality and supports generation of budget constraint plans, which can be adopted by the application users. Second, the execution advice mechanism helps a user to improve performance of the application at run time while overload or failure on the resource(s) is detected. Third, the scheduling policy mechanism provides a user an interface in ClassAd format to define the scheduling policy applied in InterS. This paper also presents experiments to show the advantage of interactive scheduling and how they can be performed at run time.

1. Introduction

GridRPC[7] is known as an effective programming model to develop Grid applications. However, it still has difficulty to make efficient applications for non-expert users. The current implementation of GridRPC assumes that a user selects remote computational resources before run time; thus it forces the user to do hard work requiring expert knowledge, e.g. monitoring remote computational resources and estimating application performance on selected computational resources. Additionally, computational resources on the grid are unstable. Loads of the resources fluctuate and some resources may have failure. GridRPC users need to make their applications robust enough, so that those can accommodate the fluctuation and failure of remote computational resources.

Several mechanisms to reduce the complexity of running Grid applications have been proposed; some of them focused on GridRPC applications. Condor[4] and Nimrod/G[5] focuses on high throughput and economic feature of Grid application, respectively. Task Farming[3] and F-Omega[6] help GridRPC users by providing advanced mechanisms in terms of task scheduling and dynamic resource allocation at application run time. GridWay[1] meta-scheduler provides users adaptive scheduling features such as automatic reschedule when task fails or a better resource is discovered. Some of above mechanisms perform automatic scheduling, where the scheduling algorithm is implemented in the scheduling software or is provided by the user through APIs. Although there are many successes in automatic scheduling, it sometime fails to obtain satisfying performance due to fluctuation of resources or a complicated user policy to run the application. In this case, changing resource allocation (or even the scheduling algorithm) using the user's knowledge contributes to improve efficiency and robustness of the application run.

In this paper, we propose InterS, an interactive scheduling system for GridRPC applications. InterS provides three mechanisms, which help GridRPC application users to run their application in an efficient and robust way. The automatic scheduling mechanism provides resource allocation functionality and supports generation of budget constraint plans, which can be adopted by the application users. The execution advice mechanism helps a user to improve performance of the application at run time while overload or failure on the resource is detected. Some expert users may want to customize the scheduling policy for their applications. The scheduling policy mechanism provides a user an interface in ClassAd format to define the scheduling policy applied in InterS. It also enables to change the currently running scheduling algorithm to others during the run time.

The rest of this paper is organized as follows: in section 2, comparison between InterS and related works are presented. In section 3, design and implementation issues are discussed. Sec-

†1 Tokyo Institution of Technology
†2 National Institute of Informatics

tion 4 shows the experimental study and the results, and section 5 gives the conclusions.

2. Related Works

Condor and Nimrod/G are resource brokers, which dispatch user tasks to suitable computational resources. Both requirements from user applications and those from resources providers are specified in the ClassAds, and the matchmaking mechanism dispatches tasks to resources so as to satisfy the both requirements in Condor. Nimrod/G has the similar mechanism and it also enables task scheduling with budget constraints. Both Condor and Nimrod/G perform fully automatic scheduling. The users cannot change resource allocation during application run time.

Task Farming middleware provides users APIs for task scheduling and a fault tolerant mechanism. F-Omega is a programming framework, which enables flexible grid application development and its execution. Although Task Farming performs automatic scheduling, the implemented scheduling algorithm is simple one and users cannot change resource allocation at run time. F-Omega enables users to change resource allocation during application run time, however, the users need to select an initial set of computational resources.

GridWay is a job submission framework on the Globus toolkit. It performs automatic scheduling and enables users to change resource allocation during application run time.

The proposed InterS performs automatic scheduling and enables users to change resources allocation during application run time. Generally, changing recourse allocation requires users to have expert knowledge about both the applications and resources on the grid. InterS helps users by giving advices for changing resource allocation during application run time to solve this problem. Furthermore, InterS enables users to change the scheduling algorithm currently running during application run time. The advantage of InterS is to fully utilize the user's knowledge through an interaction with the user at application run time so as to achieve more efficient and robust application run. Table 1 summarizes techniques enabled in scheduling software. To the best of our knowledge, there is no software that enables "interactive scheduling" mechanisms as InterS.

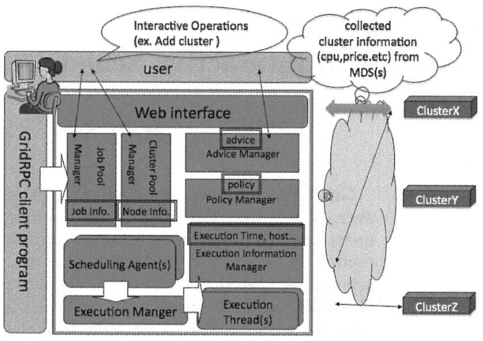


Fig. 1 InterS architecture overview

3. Design and Implementation

Fig. 1 illustrates software architecture of InterS. The user first writes a GridRPC client program using the InterS client APIs. The "addJob" method (in Table 2) creates GridRPC tasks first then stores them into the job pool through the job pool manager. Three ways of adding resources to InterS are supported: the API, the web interface and the policy file. GridRPC executables, or tasks, are executed on remote computational resources selected by InterS. At run time, the user starts the application and performs interactive scheduling through the web or policy file interface.

3.1 Client Interface

A user of InterS first writes a GridRPC client program using the InterS client APIs. The user gives information about the application, or remote GridRPC executables, through the APIs. Currently, APIs for Java & Groovy[10] are available. Table 2 summarizes the InterS Client APIs. Fig. 2 presents an example of InterS client program in Groovy language. First, the user needs to decide which scheduling policy to use for automatic scheduling, and then binds the remote executables with the scheduling agent by creating a RemoteFunction instance. The "FCFS" scheduling policy is adopted for utilizing local clusters. When tasks finish their execution, the results are stored into "results", which is an array of String arrays (defined at line 3). After submitting tasks to InterS, user can call waitAll method to block the client program until all tasks finished. In this example the user decides to add a commercial resource later, which shows better performance. So the user changed the "FCFS" scheduling agent to "BudgetConstraint" for the

Table 1 comparison of scheduling techniques in scheduling software

	Condor	Nimrod/G	Task Farming	F-Omega	GridWay	InterS
automatic scheduling	Y	Y	Y	N	Y	Y
scheduling budget constrains	N	Y	N	N	N	Y
execution advice	N	N	N	N	N	Y
changing resources allocation	N	N	N	Y	Y	Y
configuring scheduling algorithm	N	N	N	N	Y	Y
changing schedule algorithm	N	N	N	N	N	Y

Table 2 List of APIs in InterS

	Task initiation APIs
ScheduleAgent	A Java class implementing the scheduling agent(s) in InterS. It takes agent type as argument in terms of FCFS, round-robin and BudgetConstraint
RemoteFunction	A Java class implementing the GridRPC remote executable(s) in InterS. It takes name of the executable and scheduling agent, a default agent setting for this executable, as arguments.
Cluster	A java class, which stores the resource information(s).
	Task submission APIs
addJob()	Submitting user tasks to InterS. Tasks are stored in the job pool and InterS decides the resource allocation. A task id will be returned for further job handling.
addJobWith()	Same as addJob except that resources are selected by users.
waitAll()	Blocking until all tasks finished.
waitFor()	Blocking until a certain task finished. The only argument is the task id.
waitAnd()	Blocking until all tasks in a certain group finished. The arguments are a list of task ids.
waitOr()	Blocking until one of the tasks in a certain group finished. The arguments are a list of task ids.
	Execution control APIs
reschedule()	Rescheduling tasks, which do not start

```
// INITIALIZE CLIENT PROGRAM
def schedulingAgent =
   new ScheduleAgent( type:''FCFS'' );
def remoteFunc = new RemoteFunction(
   name:''NPB/EP'', agent:schedulingAgent );
def scheduler, results = [], taskids = [];
// SUBMIT TASKS
for(i=0;i<N;i++) {
 def aResult = new String[];  results << aResult;
 taskids <<
     scheduler.addJob(remoteFunc,arg1,arg2,aResult );
}
scheduler.waitAnd( ** subset of taskids **);
// CHANGE SCHEDULING AGENT
schedulingAgent =
    new ScheduleAgent( type:''BudgetConstraint'' );
remoteFunc.agent = schedulingAgent;
remoteFunc.save();
scheduler.reschedule();
scheduler.waitAll();
// PROCESS RESULTS
for(i=0;i<N;i++) { **use results[i]** }
```

Fig. 2 InterS client program

cost management, after waiting for the end of tasks specified in waitAnd. Finally, all tasks finish execution, user can access values stored in the "results" list.

3.2 Automatic Scheduling

The automatic scheduling mechanism selects computational resources that satisfy requirements of the user application. Multiple scheduling algorithms, or policies are implemented in InterS. Currently, the FCFS policy, the round robin policy and the budget constraints policy are available in InterS. FCFS and round robin are the simple heuristics, which allocate resources without any cost concern. On the other hand, budget constraint algorithm offers scheduling plans to the user. Various allocation variations as well as the cost and performance estimation are given in the plans. The user then chooses the suitable one for his/her preference. Also, the user can implement the customized policy through the scheduling policy mechanism. The scheduling agent presented in Fig. 3 performs scheduling. An instance of the scheduling agent is generated for each scheduling policy, that is, three scheduling agent instances (FCFS, round robin, budget constraint) are implemented in the default setting. The user can change the scheduling policy at application run time by switching the scheduling agent instances. The scheduling agent submits tasks to the execution manager following the scheduling policy. The execution manager invokes tasks to remote computing resources through execution threads, where an execution thread is created for each task.

Information of remote computational resources, e.g. CPU specifications, available memory sizes and unit prices for computation, is required in scheduling. InterS has ways to obtain the information. The first way is col-

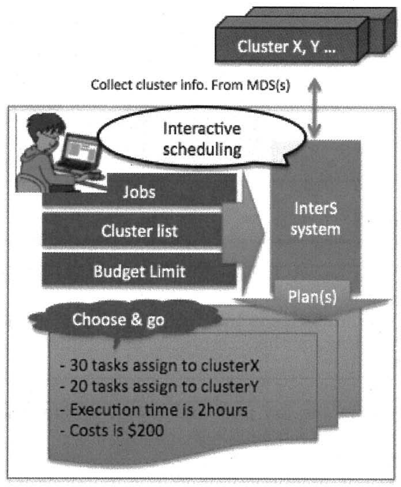
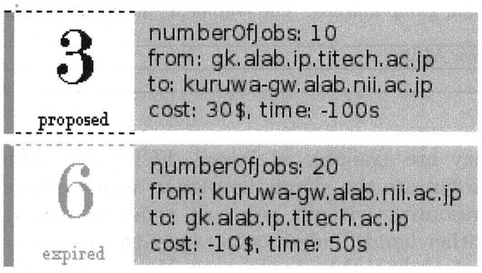

Cluster X, Y ...
Collect cluster info. From MDS(s)
Interactive scheduling
Jobs
Cluster list
Budget Limit
InterS system
Plan(s)
Choose & go
- 30 tasks assign to clusterX
- 20 tasks assign to clusterY
- Execution time is 2hours
- Costs is $200

Fig. 3 An example of automatic scheduling



3
proposed

```
numberOfJobs: 10
from: gk.alab.ip.titech.ac.jp
to: kuruwa-gw.alab.nii.ac.jp
cost: 30$, time: -100s
```

6
expired

```
numberOfJobs: 20
from: kuruwa-gw.alab.nii.ac.jp
to: gk.alab.ip.titech.ac.jp
cost: -10$, time: 50s
```

Fig. 4 An example of execution advice

lecting resource information from the MDS[8]. The resource information is collected automatically and the user does not have to take care about the information. The second way is that the user provides the information.

Estimation of task execution time on remote computational resources is an important issue to make better scheduling plan. InterS has a mechanism to estimate task execution time on remote computational resources by running test jobs. The user can control the test by e.g. defining the number of test jobs to run, through the InterS interface. Running test jobs is not acceptable in some cases due to performance problems or budget constraints. In this case, the user can give estimated task execution time to InterS.

3.3 Execution Advice

The execution advice mechanism gives advice for changing resource allocation or scheduling plan during application run time, when it finds a better plan. For instance, when new computational resources are available during the application run time. InterS detects it and estimates the application performance by utilizing the new resources. If InterS estimates that adding the new resources improves the application performance, InterS gives the advice for utilizing the new resources to the user. The user can use the new resources through InterS interface if the user accepts the advice.

Fig.4 presents an example of the execution advice. The numbers 3 and 6 are the advice ids. User can handle these ids through the web interface to the advice manager which advice is applied. The execution advice shows users the number of migration tasks, cluster names, cost and performance changes. Advice 3 recommends the user to migrate 10 tasks from the resource "gk" to the resource "kuruwa-gw". The advice also shows that the migration costs $30 more but makes execution time 100 seconds shorter. Execution advices are managed in three statuses: proposed, used and expired. Advice 6 is in the expired status because less than 20 tasks are available in cluster "kuruwa-gw".

The advice manager presented in Fig. 4 performs the execution advice. The advice manager periodically communicates with the execution information manager, the policy manager and the cluster pool manager presented in Fig. 1. The execution information manager collects past task execution records, which include errors, performances and costs to run tasks on remote computational resources. When a new resource becomes available, the cluster pool manager notifies the advice manager the information of the new resource.

3.4 Scheduling Policy

The scheduling policy mechanism enables the user to give the customized scheduling policy. Two interfaces to give the user's policy, the web interface and the policy file in ClassAd format are available in InterS.

The policy manager presented in Fig. 1 actuates change of the user's policy to scheduling at application run time. It includes adding new computational resources, change budget constraints and configuration of test jobs and etc.

Fig. 5 shows an example of the policy file. Every change is detected and handled by the policy manager. "CostLimitation" stands for the limitation of the entire application run. "TestJobCostLimitation" specifies the totally cost upper bound of test jobs. The user may

```
costLimitation = 1700;   // Budget constraints
testJobCostLimitation = 30;   // Test job cost constraints
numOfJobsPerCall =  // Bundle tasks into one GridRPC request
    ( CLIENT_LOAD_AVERAGE <= 0.1 )? 3 : ( CLIENT_LOAD_AVERAGE <= 0.5 )? 5:20;
cluster2 = [   // Cluster definition
    name = "gk.alab.ip.titech.ac.jp";
    price = 0.02;
    cpuInfo = (RMOTE_NODE_CPUINFO is undefined)? 1263.475 : RMOTE_NODE_CPUINFO;
    memInfo = (RMOTE_NODE_MEMINFO is undefined)? 1010 : RMOTE_NODE_MEMINFO;
    numOfNodes = (RMOTE_NODE_NUMBER is undefined)? 4 : RMOTE_NODE_NUMBER;
    corePerCPU = (RMOTE_NODE_NUMOFCORE is undefined)? 2 : RMOTE_NODE_NUMOFCORE;
    onFailReduceRatio = numOfJobsPerCallReduceRatio;
];
```

Fig. 5 A policy file example

Table 3 experimental environment

	DRM*1	CPU (vendor/MHz)	nodes×cores	OS	exec. time(s)*2	price($/(*core* · *s*))
blade	SGE	Pentium/1266	4×2	Redhat 7.2	210	0.02
kuruwa	SGE	AMD/2412	2×4	CentOS 5.0	116	0.13

1.DRM: distributed resource manager, such as SGE, PBS, Condor

2.Average execution time of EP(Class A) benchmark per core in each cluster

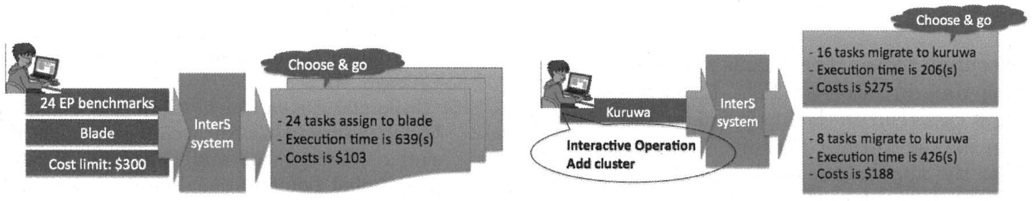


Fig. 6 Application scenario used in the experiments

change these options to affect the behavior of the budget constraint scheduling agent. Cluster information is written by ClassAd expressions; for example, RMOTE_NODE_NUMOFCORE is assigned to corePerCPU, if InterS can retrieve the number of cores in “gk” from MDS. Expression of numOfJobsPerCall is in charge of task bundle, which helps to reduce the heavy load of client machine.

4. Experimental Study

This section presents experiments to see effect of interactive scheduling provided by InterS. We implemented InterS and conducted experiments using PC clusters located in two sites. In the experiments, we verify an effect of interactive scheduling by InterS using an application scenario (Fig.6).

4.1 Experimental Setting

Table 3 shows PC clusters used in the experiments. Two clusters, blade and kuruwa, are located in Yokohama and Tokyo, respectively. The execution time in the table shows the average benchmark execution time of each core in each PC clusters; EP benchmark (Class A) in NAS Parallel Benchmarks[9] is used here. We assume that a unit price to run computation on the PC clusters, price in Table 3, is announced from resources providers.

4.2 Results

Fig. 6 shows the execution scenario used in this experiments. The user requests to run the EP benchmark (Class A) in NPB 24 times within a budget of $300. The scheduling agent for budget constraints is adopted for the resource allocation. In this scenario, we assume that the kuruwa cluster is not available at this time due to the heavy load of external jobs. InterS offers the plan to run all tasks to the blade cluster with the cost of $103, and the user accepts this plan. When the first eight tasks are running, the kuruwa cluster becomes available. InterS detects it and gives the user an advice with new scheduling plans (right in Fig. 6). Here, the user chooses the first plan with higher performance.

Fig. 7 shows the results of resource allocation and execution times for the above 24 tasks. The X-axis indicates the task numbers. Note that InterS run one test job for estimation of the

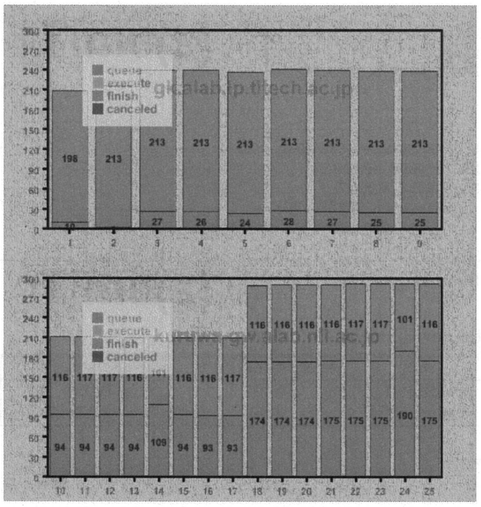


Fig. 7 results of experiment with re-planning

execution time. The Y-axis shows the execution time in seconds. Fig. 7 shows that first eight tasks run on the blade cluster and 16 tasks run on the kuruwa cluster. The results show that the application finishes in 292 [sec], while it takes more than 600 [sec] if the user runs the application without the execution advice by InterS.

5. Conclusions

This paper proposed InterS, an interactive scheduling system for GridRPC applications. InterS fully utilizes the user's knowledge through an interaction with the user at application run time so as to achieve more efficient and robust application run. We implemented three mechanisms, the automatic scheduling mechanism, the execution advice mechanism and the scheduling policy mechanism on the testbed and evaluated the effectiveness of the interactive scheduling using the application scenario.

The experiments presented in this paper are limited to those with the simple application scenario. We plan to evaluate the advantage of InterS through experiments with more scenarios.

Acknowledgments A part of this work is supported by Japan Society for the Promotion of Science (JSPS) within the framework of Global COE Program "Photonics Integration-Core Electronics".

References

1) E.Huedo, R.S.Montero and I.M.Llorente.: "A Framework for Adaptive Execution on Grids," Intl.J.of Software - Practice and Experience (SPE), 2004
2) Yoshio Tanaka, Hidemoto Nakada, Satoshi Sekiguchi, Toyotaro Suzumura, and Satoshi Matsuoka.: "Ninf-g: A reference implementation of rpc-based programming middleware for grid computing," Journal of Grid Computing, 2003.
3) Yusuke Tanimura, Hidemoto Nakada, Yoshio Tanaka and Satoshi Sekiguchi.: "Implementation of A Task Farming API over GridRPC Framework," IPSJ SIG Technical Reports, HPC-103, 2005
4) Ra jesh Raman, Miron Livny, and Marvin Solomon.: "Matchmaking: Distributed Resource Management for High Throughput Computing," Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.
5) Rajkumar Buyya, David Abramson, Jonathan Giddy.: "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid," hpc,pp.283, The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region, 2000
6) Hiromasa Watanabe,Shoichi Hirasawa, Hiroki Honda.: "F-Omega: A Framework for GridRPC Application with Adaptive Server Use," IPSJ SIG Technical Reports, HOKKE-2007, 2007
7) Keith Seymour, Hidemoto Nakada, and et al.: "Overview of GridRPC: A Remote Procedure Call API for Grid Computing," GRID COMPUTING, GRID 2002, LNCS 2536, 2002
8) K. Czajkowski, S. Fitzgerald, I. Foster, an d C. Kesselman.: "Grid Information Services for Distributed Resource Sharing, in Proc. of 10th IEEE Int. Symp. on High Performance Distributed Computing, San Francisco, CA, USA, 2001.
9) D. Bailey, J. Barton, T. Lasinski, and H. Simon (Eds.).: "The NAS Parallel Benchmarks, NAS Technical Report RNR-91-002, NASA Ames Research Center, Moffett Field, CA, 1991.
10) Groovy: An agile dynamic language for the Java Platform http://groovy.codehaus.org/