International Journal of Grid Distribution Computing
Vol.7, No.5 (2014), pp.31-42
http://dx.doi.org/10.14257/ijgdc.2014.7.5.03

An Intelligent Agent based Grid Scheduler with Enhanced Fault Tolerance

Zhang Tienan [*]

School of Computer & Communication, Hunan Institute of Engineering
[*]*ztnan1974@126.com*

Abstract

In large-scale grid platforms, providing fault-tolerance for users is always a challenging task because of the uncertainty of network resources. In this paper, we present an intelligent agent based meta-scheduler, which is aiming at improving the fault-tolerance of grid systems when running user's application. The proposed meta-scheduler is designed as an extendable framework, which allows plugging in multiple scheduling policies for deal with different scenarios. The agent-based scheduling framework enables grid systems to deploy their local schedulers in a flexible manner. Extensive experiments are conducted to investigate the performance of the proposed meta-scheduler, and the results show that it is effective to provide enhanced dependability for grid users, especially when the system is across multi-organization.

Keywords: *Grid Computing, Meta-Scheduler, Quality of Service, Resource Allocation*

1. Introduction

In a grid environment, there are potentially thousands of resources, services and applications that need to interact in order to make possible the use of the grid as an execution platform. Since these elements are extremely heterogeneous, there are many failure possibilities, including not only independent failures of each element, but also those resulting from interactions between them. Because of the inherent instability of grid environments, fault detection and recovery is another critical component that must be addressed [1]. The need for fault-tolerance is especially acute for large parallel applications since the failure rate grows with the number of processors and the duration of the computation. Fault tolerance is the survival attribute of computer systems.

General speaking, fault tolerance is to preserve the delivery of expected services despite the presence of fault-caused errors within the system itself [2]. Errors are detected and corrected, and permanent faults are located and removed while the system continues to deliver acceptable service. As a result of the complex nature of heterogeneous networks, fault tolerance is the major concerns in the grid environment. There are various ways by which detection of such faults can be accomplished. When a fault occurs, it is important to rapidly determine exactly where the fault is, and then isolate the rest of the network from the failure so that it can continue to function without interference. After that, the system should reconfigure or modify the network in such a way as to minimize the impact of operation without the failed component or components, and then repair or replace the failed components to restore the network to its initial state.

Besides the issue of failure diagnosis and correction, there is also another interesting question to be considered in terms of fault treatment. It is important to investigate how to provide broader fault tolerance in grids since grid software (middleware and applications) is complex and prone to failures that are more dangerous than crashes such as timing or omission ones [3]. Traditional fault tolerance mechanisms such as replication and checkpointing-recovery were used in grid systems in the initial stages. Then fault tolerances of grid job and data replication services are designed with improvements, but scalability issues were not satisfactory. Grid workflow managers were designed then to initiate recovery from within workflows, an adaptive checkpoint and recovery scheme for grid workflows was then implemented which was found to be efficient than the traditional approach [4].

In this paper, we identify and build the fault tolerance mechanisms in grid environments. The rest of this paper is organized as following: In Section 2, related work is discussed; the overall proposed system is explained in Section 3. The design of the proposed meta-scheduler is described in section 4. Section 5 explains the experimental setup with conclusion in Section 6.

2. Related Work

Fault tolerance is the ability of a system to perform its function correctly even in the presence of faults [5]. The fault tolerance makes the system more dependable. A complementary but separate approach to increase dependability is fault prevention. This consists of techniques, such as inspection, whose intent is to eliminate the circumstances by which faults arise. A failure occurs when an actual running system deviates from this specified behavior [6]. Depending on the job and its running time, users may monitor the progress of their job and possibly change their mind about where or how it is executing. Historically, such monitoring is typically done by repetitively querying the resource for status information, but this is changing over time to allow easier access to the data [7, 8]. A two-level market solution for services composition optimization in mobile grid was done where participants in the mobile grid environment including grid resources, services and users can be represented as agents [9]. Two-level market converges to its optimal points; a globally optimal point is achieved. The system utility of mobile grid is maximized when the equilibrium prices are obtained through the service market level optimization and resource market level optimization.

Many works on fault tolerance use adaptive as well as load balanced techniques. However, they try to plan the flow of execution before the actual scenario starts. In real life environments, when there is sudden increase of load on the system, it tends to increase the number of failures, which eventually increases the system downtime. On demand fault tolerant techniques which can handle such burst in load and do not cause overhead in the absence of burst are required. To handle this situation [10] proposes grid system objective optimization scheduling that provides a joint optimization of objectives for both the resource provider and grid user, which combines the benefits of both resource provider objective optimization and user objective optimization. Since grid is difficult and complex to implement and manage, so there were differences in their performance under diverse experimental conditions. Most of the approaches of fault tolerance are based on the prediction of failure probability of the resources in a certain time interval. It is hard to achieve the resource failure prediction even with years of historic trace data. Hence, efficient techniques are required which do not base their decisions on the specific failure model and do not rely on the failure prediction accuracy are required. A mechanism proposed by [8] for efficient resource discovery with the help of an agent was a major contribution in the field of computational grid.

An agent-based artificial immune system approach for adaptive damage detection in distributed monitoring network was developed in [11]. The presented approach establishes a new monitoring paradigm by embodying desirable immune attributes, such as adaptation, immune pattern recognition, and self- organization, into monitoring. A mobile agent system embedded in sensor nodes supports the selective generation, migration, communication, and management of mobile monitoring agents automatically. The active structural health monitoring is achieved by distributing mobile monitoring agents to the sites where they are needed. Grids are highly dynamic in nature, failure in the resources must be monitored and how changes in the topology and computational capability of the grid resources affect the efficiency in terms of deadline of the tasks must also be studied [12, 13]. A monitoring structure was introduced in the work in which the above issue was taken care of. Agents were used as a monitoring tool for getting updates regarding each cluster. Two queues were maintained, one for new submissions (NJ) and another for submissions progress queue (CJ) [6]. The grid scheduler was more complex as it maintained the list of available processors in the neighboring cluster. This was done by using some preprocessing technique like complete sub-graphs.

3. Scheduling Framework Analysis and Design

3.1 Motivations

The goals for a fault tolerance approach for computational grid result from the following properties of agents and agent systems [13]: (1) Support for communicating agents; (2) Autonomy; (3) User transparency; (4) Efficiency; (5) No modifications to hardware, operating system, or run time environment; (6) Portability and reusability. The common thread underlies both agents and grids, namely, the creation of communities or Virtual Organizations bound together by a common goal or cause. Yet the two communities have focused on different aspects of this common problem. In the case of grids, the primary concern has been the mechanisms by which communities form and operate [14, 15].

Computational grids are now widespread, but their large-scale behavior is still poorly understood. Reports are available on some calculations of loading, scaling and utilization behaviors of computational grids, based on simulations [16]. In the present work agents were used to represent computational jobs, users and resources. The realistic behaviors were studied by endowing user agents with time-varying microscopic behavior patterns [17-19]. The static flow and dynamical macroscopic properties of the network including emergent pathological behaviors and other anomalies that arise when parts of the network become temporarily unavailable were also examined. Some benefits of using agents for fault tolerance in grid computing in general are flexibility, adaptability, maintainability, customization and reusability [8]. But when it comes to the real implementation, there are some challenges and issues that have to be resolved. Much of the research was devoted to the important topic of developing methods to make the computational grid fault tolerant.

Different types of users (typically scientists and engineers) load applications to execute on grid and expect results in more appropriate and timely manner [20-22]. The users need not have to bother where the job is scheduled, on which machine, how it is utilizing resources, or if there is a problem in system on which execution of application is going on etc. So, the challenge is to execute applications efficiently and robustly in grid environments, without placing these burdens on the programmer or the user. More specific, we summarize the designing goals of our meta-scheduler as following: (1) Waiting time should be minimized and results to be accurate; (2) Demands of tasks have to be reallocated quickly and automatically, in a completely transparent way from the user's point of view; (3) Separate recovery mechanism for different types of faults. Each time different types of faults may occur, the system needs some intelligence to deal with the new types; (4) List faults that are frequent or that are rare. If the fault has occurred once, the corresponding recovery mechanism has to be applied by the system so that waiting time is reduced to a grater extent; (5) Light weight agent can be used so that grid computing environment traffic does not have much impact due to agent communication.

The proposed fault tolerant meta-scheduler should embed all the above characteristics to execute large-scale computations using remote clusters and high performance computing system. Since the primary purpose of grids is to provide computational power to the users, the intended application developer is a domain expert, typically a physicist, a biologist or a meteorologist, not necessarily a grid expert. So, most application developers are unaware about different types of failures that may occur in the grid environment and hence, fault tolerance becomes increasingly important.

3.2 Framework of Grid Scheduling Architecture

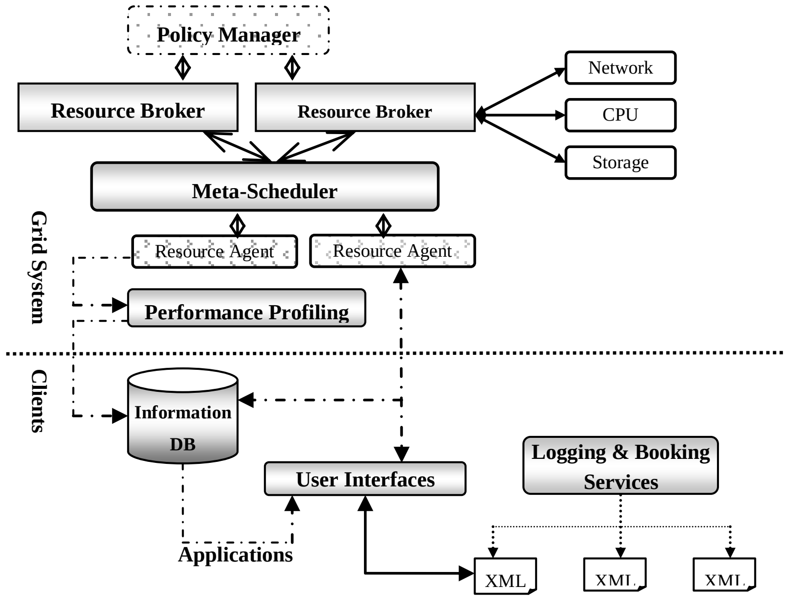


Figure 1. Framework of Agent-based Grid Scheduling Architecture

This work is aimed at building a fault tolerant computational grid using agent based scheduler. It was found that agent technology offers many advantages in terms of ease of deployment, usage and the ability to control the scheduling of grid applications at a higher level. The framework provided a generic mechanism for incorporating agents into grid applications. It was applied to the meta-scheduler with multi criteria approach. This scheduler enabled the easy integration of fault tolerance techniques to the computational grid. The fault tolerant techniques were more specific depending on the type of a fault. The framework and the different components are explained in detail is shown in Figure 1. The key components of this framework are described as following:

- **User Interface**: Service through which a user generates a request through jobs to the grid. All the parameters related to the user for particular applications or jobs are submitted through this interface. The final results are delivered to the user only through this user interface
- **Resource Broker**: It does job queuing, job scheduling and job dispatching like any other scheduler. Sometimes large jobs will be broken down into smaller jobs for easy mapping of resources. The scheduling decisions taken by the Scheduler depends on the information collected by the Information Service module about the grid resources. Usually the resource broker or scheduler will contain three modules Job queue, Job scheduler and Job dispatcher. The resource broker has the same role as any of the usual scheduler
- **Agent based Meta-scheduler**: The meta-scheduler design using agents is the heart of the proposed system. The scheduler reduces the workload of the inbuilt scheduler. Scheduler helps the jobs to be scheduled when there is a fault inside the grid computing environment. The Agents coordinate with the inbuilt scheduler and the information service for taking up the decision in mapping the resources to the jobs. They also frequently check the performance of the resource broker and add some intelligence in scheduling. Jobs with high priority or high weight in the job queue are separated by the multi approaches especially when there is a fault in the environment. The scheduler also makes sure that an agent does not cause any overhead or overload the traffic in fault conditions. From the user's point of view, the agents and the scheduler are completely transparent. Meta-scheduler can be reprogrammed or reconfigured easily by changing the weight of the different parameter in multi-criteria. There are different components the meta-scheduler is explained later in the design part of the meta-scheduler.
- **Logging and Book-keeping Service**: The service is used for basically recording the different types of faults and their corresponding recovery mechanism. Agents use this service for learning the previous faults and update their intelligence so that fault tolerant mechanism is time efficient. This service is frequently updated and analyzed. In fault tolerant mechanism, agents were able to act quickly only because of this logging service.
- **Job submission service**: As two schedulers are involved in the proposed system, the job submission service is made as separate service. Schedulers coordinate with each other, map the resources to the job and finally submit to the job submission service. Acts as an interface between the resources and the schedulers. It coordinates with the logging and book keeping service after before and after job submission.
- **Information Service**: The service contains all the information related to the jobs and the resources. Information provider to all other services in the system. It stores all the information in the databases and also has an updater module to frequently update the information. Agents collect the information from the grid and submit it to the information service. Meta-data about the resources and the jobs are also stored in the databases which will helpful for the fault tolerant mechanisms

4. Implementation of Agent-based Meta-Scheduler

The work focuses on a multi-agent framework for, fault tolerance, performance monitoring and tuning of computational jobs running in grid environment. The framework aims at providing adaptability to the runtime environment of a computer-intensive job running on a grid. The framework in Figure 2 proposed is general enough to cope with any distributed system with some extent of dynamism. The approach was aimed at providing protocols for discovering devices as they join the network and making their presence known to other members of the network. This also supports applications in heterogeneous, dynamic computing environments and it has been used to develop several agent based applications. Agents discover each other and communicate with each other using the grid services.

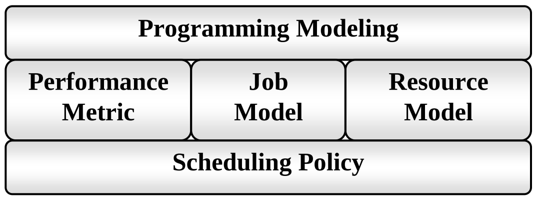


Figure 2. Architecture of Agent based Meta-Scheduler

It is also necessary to monitor the performance of a job on a regular basis so that the agreements between the clients and the resource providers are maintained. It needs to be seen that the clients get the services that they have been assured for and no over provisioning is made, as the clients may have to pay for the services. Whenever a fault is detected, middleware must take appropriate actions, which may change the execution environment of the job [9]. The agents act upon the middleware that is platform independent and supports byte-code for its code can be downloaded on any type of operating system or hardware and executed. So this system supports portability and interoperability. The key components of the above meta-scheduler framework are represented as following:

- **Consumer Agents**: They collect all the user related information particular to the job and submits the same to the information service. Information service based on the data by the customer agents, it generates the metadata like weight for each criterion, release- time, waiting time allowed, type of user etc. Metadata will be helpful for the meta-scheduler in taking the decisions during fault conditions. Using the multi criteria approach the jobs with high priority or high weight will be taken into consideration. In addition, they also act as a launching point for jobs to run and collect results after execution.
- **Broker Agents:** They perform write, read, take & notify operations between consumers and producers.
- **Producer Agents**: They are responsible for the Maintenance of core computational grid components and executing tasks/jobs on spaces and agents to support fault tolerance. They also collects all resource related information such as resource speed, resource availability, resource type and bandwidth *etc.*, particular to that grid environment and updates the information service frequently.

Based on the above the descriptions, the four kinds of agents are interacting as following steps:

1. Using the user interface the customer agents collects all the necessary information related to the Job.
2. The meta-data generation service is called by the customer agents to generate the meta- data for each job like date of submission, name of the machine submitted from, and the user's username on the submission machine, types of user (premium, regular or new).
3. Repeat the step 2 for multiple jobs and generate the meta- data information for all the jobs and update the same in the information service module.
4. Based on the meta data information from the previous step for each job, additional parameters are set for every scheduling policy and decisions, a minimum and maximum value of a decision attribute, standard deviation of a decision attribute, a mean error of previous predictions.
5. The information service is been updated or sorted according to the decision attribute that is been generated from the Meta-data information.
6. Steps 2-5 will be repeated for the resources. The meta-data information for the resources would be any special type of simulation available in that cluster or machine, last job completed , any fault occurred in that machine before *etc.*
7. The resources also ranked based on the decision attributes which is generated from the Meta –data information of the resources.
8. After the resource decision database and job decision database based on the meta-data is been constructed , then broker agents takes the help of the performance agents that works with logging and book keeping service and maps the resource to the job with appropriate recovery mechanism like cloning, migration, replication or checkpointing.
9. At the end of mapping or scheduling there will be at least one recovery mechanism to make the grid computing environment fault tolerable.

In this way a simple test bed for grid is formed, but providing fault tolerance is the work of agents which resides in the nodes consumer, producer or broker. There were various states of agents like Available, Busy, Locked, Transfer, Death, Save point and Pending. Each state plays a vital role in executing applications on the grid. The agency which is provided as wrapper, allows the agents to communicate and pass status messages. As grid applications require tremendous computation power and perform long tasks which may run for many days on different heterogeneous nodes, this requires powerful mechanisms to handle failures. Partial failures are a major issue in grid environment. There are many mechanisms to deal with failures while a node, application or network crashes: (1) Retrying: stopped and started from scratch; (2) Checkpointing: repository the execution states and restores at point of failure; (3) Replication: simultaneously executing task on n different nodes; (4) Checkpointing & Replication: This mechanism is highly required for high-end tasks.

Whenever a consumer (user) wants to execute a job, he puts the job into broker (space) subdivided into small tasks. The producers which get notification and see into space will take the tasks and execute them. Number of producers is the strength of the grid and its computation power. After the execution is completed, the results are written to space and if some more tasks are there, again producer the will take and execute it. The broker is a rich shared memory location which is capable of handling tasks and executed results. It notifies the results back to the consumer. A central issue in this system is how to find services. The broker provides the lookup services typically by service type and/or service attributes. In this current system, whenever a service is registered in the LUS with a unique service ID, a new agent is deployed with a unique agent ID (ID is created by combination of host address and system current time in milliseconds). So, no other agent can have the same identity (ID). The lookup service uses the required service interface and attributes during service matching in its lookup method. When connecting to the system, producers also need to locate lookup services and then find brokers. If at least one broker is available, the consumer side agent will download the broker proxy and use it to perform computation directly on producer or indirectly by asking the broker to perform the task.

5. Experiments and Performance Evaluation

5.1. Experiment Settings

In the evaluation of the scheduling algorithm, the test-bed for evaluation, the property limits of the task, and the choice over more than one scheduling algorithm could verify the change in the time needed to finish the work according to the amount of work. The performance time and the size of input and output data produced a random number in uniform distribution or normal distribution. As a result, to obtain the exact measured value, the average value related to the designated number of repetitions should be acquired, to eventually get the final performance time. The cluster configuration that was simulated is shown in Table 1. All the clusters offered similar type of services. Some unique services were also offered by each cluster. The services could also be interoperable; the system1.vitgrid used the services which were available in vishwa vitgrid. Here system1 nodes loaded the job for execution and vishwa grid executed it.

Table 1. Configurations of Test-bed Platform

Cluster ID	Resource Type	Number of PE	MIPS Rating	OS
system1.vitgrid	system1.vitgrid	25	337	Solaris
vishwa.vitgrid	vishwa.vitgrid	25	280	Linux
system3.vitgrid	system3.vitgrid	50	377	Solaris

5.2 Performance Analysis

Various time measures were used as metrics to measure the performance of methods applied. In particular, the scheduling time (during which the whole job is scheduled) and make span (the completion time of the last job) was an important metric for the evaluation of the efficiency of the proposed approach. The observation shows that the checkpointing outperforms others as regards both the scheduling time and the make span time Figure 3 shows the average scheduling time for 100 machines versus jobs; The difference concerning the scheduling time for checkpointing was more significant than the difference between values of make span for migration and cloning. By using the agents, many long queue waiting times that increase the maximal and mean scheduling time were reduced to a grater extent. Additionally, the agent based job scheduler turned out to be more profitable for the scheduling times and make span time when the numbers of machines were more. Figure 4 shows the average scheduling time versus the no of jobs for 150 machines, the graph clearly shows more differences compared to 100 machines.

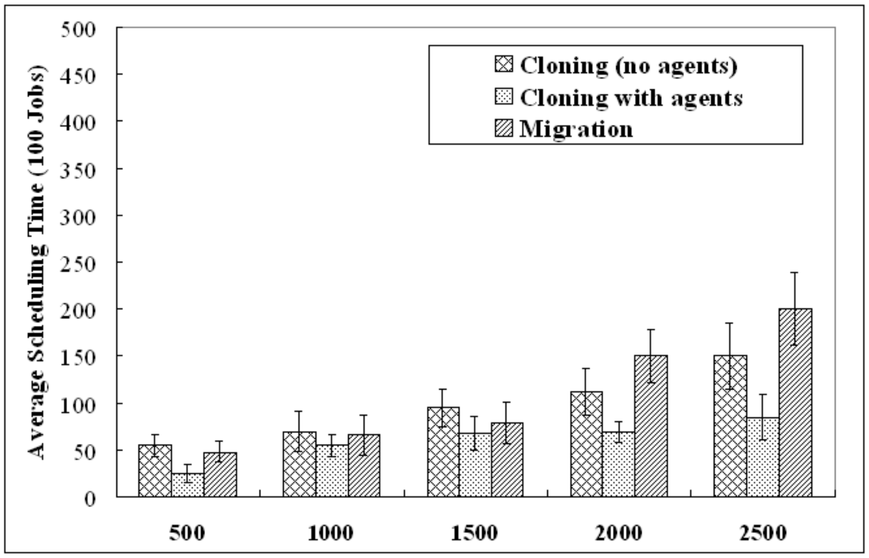


Figure 3. Average Scheduling Time for Jobs (100 machines)

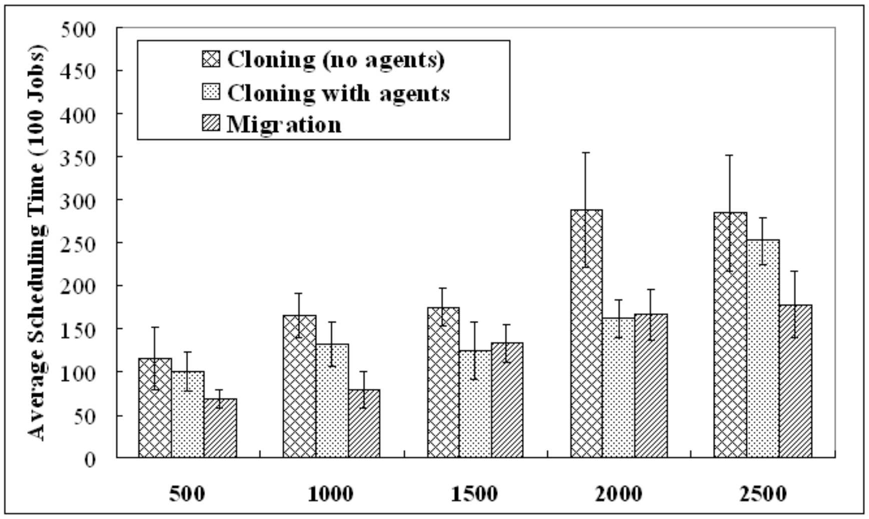
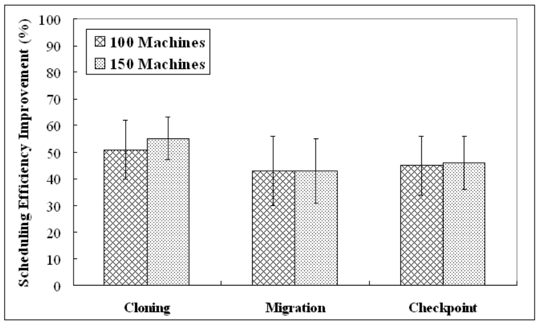


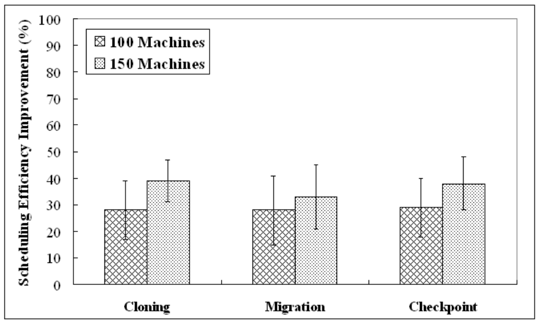
Figure 4. Average Scheduling Time for Jobs (150 machines)

The big scheduling time for the migration resulted as the agent state could not be transferred immediately. Even after completion of most of the jobs it was impossible to reduce a load without the use of a job migration. From the graph it can be observed that the average scheduling time for 250 jobs for migration was much higher than for checkpointing, cloning and replication for same amount of jobs because of the fact that the reschedule was very difficult while migration was done. For the lower number of jobs, the three recovery mechanisms were found to be closer to each other. The performance was same even when the number of machines was increased up to 150. The advantage of the agent for scheduling during fault tolerance revealed itself mainly when the test machines were saturated with incoming jobs, which means that new jobs submitted to scheduler could not be started. However, an excessive number of jobs decreased the performance of agents, even for checkpointing. The obvious reason for this was that jobs could not be scheduled to any other hosts for lack of free resources.

The two level hierarchical scheduling played a vital role in the allocation of resources. As both resources and jobs were ranked and separate allocation was done when there was a fault, the detection time was considerably reduced when the load was very small and as the load increased the overhead also increased which to a certain extent affected the response time also. The make span time steadily increased even though the fault and the nature of the recovery mechanism did not much affect the make span time. The system was fault tolerant at various levels and it did not show much difference when it came to a large grid computing system. Even though there was a little distortion in the beginning, as a whole the proposed system performance did not degrade to a larger extent. Figure 5(a) shows the improved percentage efficiency over the traditional methods. To make the comparison of the proposed work more useful and detailed, improved percentage efficiency for all iterations were performed. The influence of agents played a vital role in making the grid computing environment fault tolerant. The criteria such as runtime and resource requirements, *i.e.*, "the size" of the job, were a major impact. Results presented in Figure 5(b) confirm that the work was satisfactorily user transparent.



(a) Average Scheduling time



(b) Makespan

Figure 5. Improved Percentage Efficiency

The use of agents was inferred to bring the greatest benefits if it was used for relatively small rather than long jobs. Observation showed that the especially good performance gain was attained when the numbers of machines were performing more incoming job set consisting of many small long-lasting jobs which were followed by the submission of jobs having large run time. Comparing these results with the analysis of other metrics presented in this section, it can be inferred that the best performance of the approach was connected with reasonable resource utilization: not too great (to avoid a machine overload) and not too small (to avoid excessively long waiting times for jobs in the queue).A migration that was performed due to the insufficient amount of free resources required by incoming jobs was explored. Application-level checkpointing was used in order to provide full portability in the heterogeneous grid environment.

The amount of free physical memory was used to determine whether there were enough available resources to submit the pending job. Nevertheless, the proposed approach was generic; other criteria such as free processors, disc space, or even a network bandwidth could be easily incorporated. Cloning techniques turned out to be very useful, especially for applications of a reasonable size and duration. Make span, Scheduling time and efficiency percentage metrics were particularly improved in the case of sets of applications characterized by reasonable memory requirements and sufficiently long execution times.

In general, the use of the job migration and cloning methods led to the significant improvement of overall performance. Furthermore, this approach may be the only way to provide requested performance for end-users if job execution times and resource usage are not known a priori, are assumed in experiments. Dynamic rescheduling based on application-level checkpointing and migration is a strategy that should improve resource brokering and scheduling in heterogeneous and non-dedicated environments. Therefore these mechanisms were particularly useful for grids. Checkpointing turned out to be useful and more efficient than the remaining two policies. Thus benefits were achieved by means of the reduction of queue waiting times. The migration of applications to the best available resources outstripped overheads connected with writing and reading the checkpoint files and application startup times.

6. Conclusion

In this paper, we present an intelligent agent based meta-scheduler, which is aiming at improving the fault-tolerance of grid systems when running user's application. The proposed meta-scheduler is designed as an extendable framework, which allows plugging in multiple scheduling policies for deal with different scenarios. The agent-based scheduling framework enables grid systems to deploy their local schedulers in a flexible manner. Experimental results show that the proposed meta-scheduler performs well when the grid system uses different fault tolerant policies. In the future, we are planning incorporate some real-time guarantee mechanisms in the current implementation. In addition, we are planning to re-design it for cloud-based systems, in which virtual mechanism is considered as the basic scheduling units.

References

[1] R. Entezari-Maleki and A. Movaghar, "A probabilistic task scheduling method for grid environments," Future Generation Computer Systems, vol. 28, no. 3, **(2012),** pp. 513-524.

[2] W. D. Gropp and E. Lusk, "Fault Tolerance in MPI Programs", International Journal of High Performance Computer Applications, vol. 18, no. 3, **(2004),** pp. 363-372.

[3] P. Zhao, "libELC-A portable library enabling fault tolerance of MPI programs in heterogeneous environments", A thesis submitted for the degree of M.Sc. in National University of Ireland, Dublin, **(2005)**.

[4] C. Dabrowski, "Reliability in grid computing systems," Concurrency and Computation-Practice & Experience, vol. 21, no. 8, **(2009),** pp. 927-959.

[5] F. G. Khan, K. Qureshi and B. Nazir, "Performance evolution of fault tolerance techniques in grid computing system", Journal of Computer & Electrical Engineering, vol. 36, no. 3, **(2010),** pp. 1110-1122.

[6] T. Altameem and M. Amoon, "An Agent-Based approach for dynamic adjustment of scheduled Jobs in Computational Grids", Journal of Computer & Systems Sciences International, vol. 49, no. 5, **(2010),** pp. 765-772.

[7] M. Amoon, "A fault-tolerant scheduling system for computational grids," Journal of Computers & Electrical Engineering, vol. 38, no. 2, **(2012),** pp. 399-412.

[8] L. Chunlin and L. L. Li, "Applying agents to build grid service management", Journal of Network & Computer Application, vol. 26, no. 4, **(2003),** pp. 323–340.

[9] L. Chunlin and L. Layuan, "Two-level market solution for services composition optimization in mobile grid", Journal of Network & Computer Applications, vol. 34, no. 2, **(2011),** pp. 739-749.

[10] L. Chunlin, Z. J. Xiu and L. Layuan, "Resource Scheduling with Conflicting Objectives in Grid Environments: Model and Evaluation", Journal of Network & Computer Applications, vol. 32, no. 3, **(2009),** pp. 760-769.

[11] C. Boo, "Agent-based artificial immune system approach for adaptive damage detection in monitoring networks", Journal of Network & Computer Application, vol. 33, no. 6, **(2010),** pp. 633-645.

[12] A. Goldman and C. Queiroz, "A model for parallel job scheduling on dynamical computer Grids", Concurrency - Practice and Experience, vol.16, no. 5, **(2004),** pp. 461- 468.

[13] L. Han and D. Berry, "Semantic-supported and agent-based decentralized grid resource discovery", Future Generation Computer Systems, vol. 24, no. 8, **(2008),** pp. 806-812.
[14] G. Ali, N. Shaikh and Z. Shaikh, "Integration of Grid and Agent Systems to Perform Parallel Computations in a Heterogeneous and Distributed Environment," Australian Journal of Basic and Applied Sciences, vol. 3, no. 4, **(2006),** pp. 3857-3863.
[15] L. Chunlin and L. Layuan, "Agent framework to support computational grid", Journal of Systems and Software, vol. 70, no. 1, **(2010),** pp. 177-187.
[16] A. I. Saleh, A. M. Sarhan and A. M. Hamed, "A New Grid Scheduler with Failure Recovery and Rescheduling Mechanisms: Discussion and Analysis," Journal of Grid Computing, vol. 10, no. 2, **(2012)** June, pp. 211-235.
[17] L. Tomás, A. C. Caminero and O. Rana., "A GridWay-based autonomic network-aware metascheduler," Future Generation Computer Systems, vol. 28, no. 7, **(2012),** pp. 1058-1069.
[18] Y. Huang, N. Bessis and P. Norrington, "Exploring decentralized dynamic scheduling for grids and clouds using the community-aware scheduling algorithm," Future Generation Computer Systems, vol. 29, no. 1, **(2013),** pp. 402-415.
[19] P. Xiao and Z. Hu. "Workload-aware reliability evaluation model in grid computing," Journal of Computers, vol. 7, no. 1, **(2012),** pp. 141-146.
[20] P. Xiao, Z. Hu and X. Qu. "Hybrid-policy co-allocation model in computational grid," Journal of Software, vol. 7, no. 2, **(2012),** pp. 382-388.
[21] P. Xiao, D. Liu and X. Qu, "Three-side gaming model for resource coallocation in grid Computing," Journal of Software, vol. 7, no. 9, **(2012),** pp. 2125-3132.
[22] N. Mansouri and G. H. Dastghaibyfard, "Job scheduling and dynamic data replication in data grid environment," Journal of Supercomputing, vol. 64, no. 1, **(2013),** pp. 204-225.

Authors

Zhang Tienan, he was born in 1973. He received his master degree in Xiangtan University at 2008. Current he is a lecturer of Hunan Institute of Engineering. His research directions include software engineering, distributed computing, network integration.