Achieving Fault Tolerance and Recovery in Computational Grid

Ravinder Mandhan[1], Vikas Yadav[2], Nisha Thakur[3]
Department of Computer Science & Engineering[1,2,3]
Manav Bharti University, Solan (H.P.), India[1,2,3]
ravimadhan@gmail.com[1] vikas8.yadav@gmail.com[2] thakurnisha71@gmail.com[3]

Abstract-Grid computing, most simply stated, is distributed computing taken to the next evolutionary level. The goal is to create the illusion of a simple yet large and powerful self managing virtual computer out of a large collection of connected heterogeneous systems sharing various combinations of resources. However, in the grid computing environment there are certain aspects which reduce efficiency of the system, job scheduling of the resources and fault tolerance are the key aspect to improve the efficiency and exploit the capabilities of emergent computational systems. Because of dynamic and distributed nature of grid, the traditional methodologies of scheduling are inefficient for the effective utilization of the resource available. The fault tolerance strategy proposed will improve the performance of the overall computational grid environment. In this paper we propose an efficient job scheduling algorithm to improve the efficiency of the grid environment. The simulation results illustrate that the proposed strategy effectively schedules the grid jobs and reduce the execution time.

Keywords: Middleware, Checkpoint recovery, Wide area replication

1. INTRODUCTION

1.1 Grid computing is a term referring to the combination of computer resources from multiple administrative domains to reach a common goal. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. What distinguishes grid computing from conventional high performance computing systems such as cluster computing is that grids tend to be more loosely coupled, heterogeneous, and geographically dispersed. Although a grid can be dedicated to a specialized application, it is more common as a single grid will be used for a variety of different purposes. Grids are often constructed with the aid of general-purpose grid software libraries known as middleware. The standardization of communications between heterogeneous systems creates the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, are driving a possibly equally large evolutionary step in grid computing.

1.2 Computational grids have become a popular approach to handle vast amounts of available information and to manage computational resources. Examples of areas where grids have been successfully used for solving problems include biology, nuclear physics and engineering. A Grid enables the sharing, selection, and aggregation of a wide variety of geographically distributed resources. Stand-alone system is prone to crash. These individual components in a distributed computing system may fail without stopping the entire computing system. So fault tolerance is an important property in Grid computing as grid resources are geographically distributed in different administrative domains worldwide. Also in large-scale grids, the probability of a failure is much greater than in traditional parallel systems. Since grid applications run in a very heterogeneous computing environment, fault tolerance is important in order to ensure their correct behavior. The development of correct grid applications is difficult with traditional software development methods. Hence, formal methods can be beneficial in order to ensure their correctness and structure their development from specification to implementation. One of the most difficult tasks in the design of a fault tolerant Grid environment is to verify that it will meet its reliability requirements. Performance models for two fault tolerance methods, checkpoint-recovery (CR) and wide-area replication (WR), have been considered. Application checkpoint-restart is the ability to save the state of a running application to secondary storage so that it can later resume its execution from the time at which it was last stored. Checkpoint-restart can provide many potential benefits, including Fault recovery by rolling back an application to a previous checkpoint, Better application response time by restarting applications from checkpoints instead of from scratch, and improved system utilization by stopping long running computationally intensive jobs during execution and restarting them when load decreases. An application can be migrated by check pointing it on one machine and restarting it on another providing further benefits, including fault resilience by migrating applications from the faulty hosts, dynamic load balancing by migrating applications to less loaded hosts, and improved service availability and administration by migrating applications before host maintenance so that applications can continue to run with minimal downtime. Many important applications consist of multiple cooperating processes. To checkpoint-restart these applications, not only must application state associated with each process be saved and restored, but the state saved and restored must be globally consistent and preserve process dependencies. Furthermore, for checkpoint restart to be useful in practice, it is crucial that it transparently support the large existing installed base of applications running on commodity operating systems. The ability to checkpoint a running application and restart it later can provide many useful benefits including fault recovery, advanced resources sharing, dynamic load balancing and improved service availability. However, applications often involve multiple processes which have dependencies through the operating system. A common method of ensuring the progress of a long running application is to take a checkpoint i.e., save its state on stable storage periodically. A checkpoint is an insurance policy against failures—in the event of a failure, the application can be rolled back and restarted from its last checkpoint—thereby bounding the amount of lost work to be recomputed. The state of a distributed application consists of the instantaneous snapshot of the local state of processes and communication channels. However, in an asynchronous distributed system with no global clocks or shared memory, we can only devise algorithms to approximate this global state [7]. A snapshot is deemed consistent if it could have occurred during the execution of an application [7, 4]. To yield a consistent snapshot, or checkpoint, an algorithm must ensure that all messages received by a process are recorded as having been sent.

2. FAULT TOLERANCE AND RECOVERY OF SCIENTIFIC WORKFLOWS

2.1 Large parallel computations, distributed data transfer and management and soft real-time constraints characterize complex scientific workflows. However, the computational grids on which these workflows may run, have distributed systems and software, virtual organizations and few guarantees of quality and availability of service. This necessitates the need for new approaches to scheduling complex scientific workflows on computational grids in order to meet the soft real-time constraints of deadline and reliability. Figure 1 shows our fault tolerance and recovery (FTR) service. It uses application performance models, network models, resource reliability models and batch-queue wait time predictions to decide what fault tolerance strategy viz. over-provisioning or migration to be adopted for each workflow step/task in a workflow. Then the FTR service either runs copies of the workflow step on multiple resources (over-provisioning) or migrates the workflow step to another resource in case of a failure (migration) to meet the soft real-time constraints of deadline and reliability (success probability).

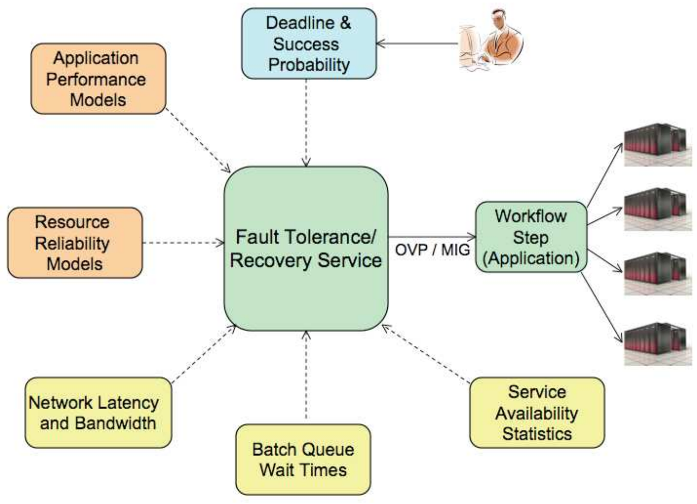
Deadline & Success Probability
Application Performance Models
Resource Reliability Models
Fault Tolerance/ Recovery Service
OVP / MIG
Workflow Step (Application)
Network Latency and Bandwidth
Batch Queue Wait Times
Service Availability Statistics

Fig. 1: Fault Tolerance and recovery service

2.1.1 Over-provisioning

Over-provisioning or replication is a fault-tolerance mechanism where multiple copies of an application (with the same input data-set) are executed in parallel. The idea is to maximize the probability of success for the application so that if one copy fails then another copy may succeed. This is particularly important in the presence of unreliable resources and where missing a deadline may incur a high penalty. The over-provisioning algorithm determines the best set of resources to replicate the application.

2.1.2 Migration

In migration, an application is progressively restarted from the last good checkpoint (if available) on a different resource in case of failures. The migration algorithm determines the best migration path. The FTR service uses the following application and resources estimates and finds the best set of resources for over provisioning or migration.

2.1.3 Application Performance Models

Application performance models are estimates of execution times of applications on different resources. Since grid resources are heterogeneous (in architecture, CPU speed and other system characteristics), the same application may have significantly different execution times on different resources. Hence, we take into account the performance models of the applications in the workflow to estimate the computational portion of the total expected completion time for the application.

2.1.4 Batch Queue Wait Times

Batch-queue systems like PBS and LSF, manage job prioritization and execution on most grid resources. An application may have to wait for a significant amount of time on a queue before it starts execution. Hence, an estimate of the batch-queue wait times is an essential component of the total expected completion time for an application. We use the methodology and software described in [15] for estimates of batch-queue wait times.

2.1.5 Network Latency and Bandwidth

Workflow execution may also involve moving large amounts of data between workflow steps, which takes significant amount of time. Hence, we also consider the data transfer time while calculating the total expected completion time for an application. To estimate the data-transfer time, we use the Network Weather Service (NWS) [16] to obtain estimates of network latency and bandwidth values between resources pairs on the grid. We also find out the size of the data to be moved from the inputs to the application. Combining data-size with NWS estimates gives us an estimated data-transfer time.

2.1.6 Deadline and Success Probability

Apart from these, the fault-tolerance algorithms also take into account the specified deadline for the application. This is important for workflows that have real-time constraints. Another input to the algorithm is the required success probability, which is the expected success probability from the scientist's perspective. It is the probability with which the scientist wants the workflow to run to completion. With application failures becoming more of a norm than exception on the grid, an expected success probability of 1 is not realistic. There is a trade-off between the value of required success probability and finding a fault-tolerance mechanism that can satisfy that value. A high value may result in the algorithms failing to find a resource set that can be used to run the application in a fault-tolerant way.

2.1.7 Resource Reliability Models

We use a simple model for modeling resource reliability. We assume that resource failures are independent over time, implying that resource failures in the current time interval are independent of the failures in the previous time interval. Hence, we can assume resource failures to follow a binomial distribution.

3. PROPOSED WORK

We present a transparent mechanism for commodity operating systems that can checkpoint multiple processes in a consistent state so that they can be restarted correctly at a later time. We introduce an efficient algorithm for recording process relationships at system-level with checkpoint/restart implementation for Linux clusters that targets typical High Performance Computing applications, including MPI. Application process state including shared resources and various identifiers that define process relationships such as group and session identifiers are saved and restored correctly. The process will be migrated from a failed node to a spare node instead of restarting the application using

checkpoint policy. We try to achieve the maximum fault tolerance by using Reliability consideration. Reliability defects generally are failures that might occur in the future inside a system that has been working well so far. Therefore, reliability must be regarded as a ratio expressed in terms of units of time. If a system is well designed and carefully configured, and thus homogeneously reliable, the accumulated failure rate of the system, F(t), is proportional to the time period of acceptable life to job assigned to the node, including on and off time[5]

$F(t)=1-R(t)=1-e- \lambda t \Xi \lambda t$

where R is reliability function, λ is failure rate, and t is time requirement of the job. By using the above formula we can calculate the number of spare nodes based on the Failure Rate.

4. CONCLUSIONS

The study has surveyed the progress in making grid systems more reliable. It has been found that, today, efforts for making grid systems more reliable have centered on developing methods for fault tolerance. Owing to the complexity of computational grids, executing complex workflows reliably is a challenge. We have developed a fault tolerance and recovery (FTR) service that uses over-provisioning and migration for reliable execution of workflows on computational grids.

REFERENCES

[1] Oren Laadan Jason Nieh, Transparent Checkpoint-Restart of Multiple Processes on Commodity Operating Systems, USENIX Annual Technical Conference [2007].

[2] J. H. Abawajy, Fault-Tolerant Scheduling Policy for Grid Computing Systems, IEEE [2004].

[3] Babar Nazir, Taimoor Khan, Fault Tolerant Job Scheduling in Computational Grid, IEEE, pp 708-713, [2006].

[4] Mattern, F., Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation, Journal of Parallel and Distributed Computing, pp. 423-434, 1993.

[5] D. Ryu, Quality, product quality, and market share increase, Int J Reliab Appl 2 (2001) (3), p. 163 174,182.

[6] Paul Stelling, Ian Foster, Carl Kesselman, Craig Lee, Gregor von Laszewski , A Fault Detection Service for Wide Area Distributed Computations

[7] Chandy, K. M., Lamport, L., Distributed Snapshots: Determining Global States of Distributed Systems, ACM Transactions on Computer Systems, pp. 63-75, February [1985].

[8] I. Foster, C. Kesselman and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. The International Journal of Supercomputer Applications,15(3), [2001].

[9] I. Foster, C. Kesselman, J. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems

Integration. Open Grid Service Infrastructure WG, Global Grid Forum, [2002].

[10]URL:http://www.globus.org/alliance/publications/papers/ogsa.pdf

[11] Jin Liang, Tong WeiQin, Tang JianQuan, Wang Bo, A Fault Tolerance Mechanism in Grid, 0-7803, IEEE pp. 457-461,[2003].

[12] J. H. Abawajy and S. P. Dandamudi. Parallel job scheduling on multi-cluster computing systems. In Proceedings of the IEEE International Conference on Cluster Computing (Cluster 2003), Hong Kong, China, December 1-4 [2003].

[13] A. Avizienis, "The N-version Approach to Fault-Tolerant Software" - IEEE Transactions on Software Engineering - vol. 11 [1985].

[14] T. Thanalapati and S. Dandamudi. An efficient adaptive scheduling scheme for distributed memory multicomputer. IEEE Transactions on Parallel and Distributed Systems, 12(7):758–768, July [2001].

[15] J. Brevik, D. Nurmi, and R. Wolski, "Predicting Bounds on Queuing Delay for Batch-scheduled Parallel Machines," Proceedings of the Eleventh ACM SIGPLAN Symposium on Principle and Practice of Parallel Programming, [2006].

[16] G. E. Fagg, E. Gaberiel, G, Bosilca, T. Spring, Z. Angskun, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," Future Generation Computer Systems, vol. 15(5-6), pp. 757-768, [1999].