ISSN 2277-3061

Dynamic Fault Tolerance in Desktop Grids Based On Reliability

Geeta Arora[1], Dr. Shaveta Rani[2], Dr. Paramjit Singh[3]
[1]Computer Application, RBIM, Mohali.
mailtogeeta@gmail.com,
[2]Computer Sc. & Engg., PTUGZS Campus ,Bathinda.
garg_shavy@yahoo.com
[3]Computer Sc. & Engg., PTUGZS Campus, Bathinda.
param2009@gmail.com

ABSTRACT

Fault tolerance is an important issue to guarantee reliable execution of tasks in computational desktop grid environment where execution failures are frequently expected, requires the availability of efficient fault tolerant strategies able to effectively deal with resource failures and/or unplanned periods of unavailability. In this paper we present a Dynamic Fault Tolerant strategy that, rather than just tolerating faults as done by traditional fault-tolerant schedulers, exploit the information concerning size of task, resource speed and resource reliability by maintaining resource history to improve application performance. The performance of this strategy has been compared via simulation with those attained by traditional fault-tolerant strategy. Our results, obtained by considering a set of realistic scenarios modeled after real Desktop Grids, show that our approach results in better application performance and resource utilization.

Indexing terms/Keywords

Desktop Grid, Fault tolerance ,Reliability ,WQR-FT, DFTDT ,Delay Time.

Academic Discipline And Sub-Disciplines

Computer Science

SUBJECT CLASSIFICATION

Grid Computing

TYPE (METHOD/APPROACH)

Quasi-Experimental

Council for Innovative Research
Peer Review Research Publishing System
Journal: INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY
Vol 11, No 4
editor@cirworld.com
www.cirworld.com, member.cirworld.com

ISSN 2277-3061

1. INTRODUCTION

A desktop grid is usually built on the Internet infrastructure in which computing resources are unreliable and frequently turned off or disconnected. In addition, communication bandwidth between these resources is not settled. Two considerable challenges can be addressed in desktop grids[1]. The first is fault tolerance, where participating machines are unreliable and may simply be turned off, disconnected or even terminate the grid application during a task execution. This will lead to the need for rescheduling the interrupted task to another computing resource (Worker) and restart execution from scratch which is time consuming. The second is the limited and unsettled bandwidth between participating nodes. This is a common problem that is always shown up when resources are connected via Internet

In order to cope with resource failures, recent work has focused on fault-tolerant strategy [2,3, 4] relies on fault-handling mechanisms (such as task replication and checkpoint-and-restart) to deal with the occurrence of resource failures or unavailability. These strategies, however, being information-free, do not use any information concerning the tasks and the resources, with the consequence that their resource usage is suboptimal.

In this paper we propose an alternative approach to fault-tolerant strategy in Desktop Grids (named Dynamic Fault Tolerance in Desktop Grids) that, instead of just tolerating resource failures, tries to avoid them as much as possible by jointly exploiting the fault handling mechanisms, and uses the knowledge of the effective speed delivered by resources [5,6] and the size of the task [7] to improve Fault Tolerance. We show how this information can be exploited to improve both task selection (the choice of the next task to be executed) and machine selection (the choice of the machine on which it will be executed) with respect to fault-tolerant schedulers. More specifically, we have calculated the reliability of a resource by calculating its delay time & maintain a resource history table on the basis of delay time. In order to show the effectiveness of our approach, we have conducted a thorough simulation study that has shown that our strategies outperform fault tolerant schedulers for a variety of operational scenarios.

The rest of the paper is organized as follows. In Section II we place our work in the context of the related literature. In Section III we discuss existing fault-tolerant strategy. In section IV we discussed the proposed strategy i.e. Dynamic Fault Tolerant strategy in Desktop Grids. Finally, Section V concludes the paper and outlines future research work.

2. RELATED WORK

Most of the existing fault management mechanisms are reactive incomplete and application dependent. For example if a job execution machine fails during execution; jobs would be submitted on another machine from start. This technique is known as Retry. We cannot afford such kind of techniques for compute intensive jobs that require huge computational resources. An extension of this approach has been presented in [8] that uses task replication to reduce the effects of poor task assignments, and automatic restart (possibly coupled with checkpointing) to deal with resource failures.

Another problem faced is the cognitive problem. It becomes very difficult to detect, identify, isolate and recover from failures. An extension to the classification [9,10, 11] of errors, failures and faults with their expected occurrence at appropriate grid layer has already been presented in [12].

A proactive approach regarding job scheduling in computational grid has been proposed in [13]. It schedule jobs based on history maintained in grid information service about grid resources.

An alternative proactive approach to BoT scheduling in Desktop Grids has been proposed in [14] that, exploit the information concerning resource availability to improve application performance.

According to our knowledge, none of the techniques or frameworks deals with failures both in proactive and reactive way. Our strategy exploit task replication and checkpoint-and-restart, but their usage of task and resource information allows them to greatly reduce the amount of wasted CPU cycles that are instead used to improve application performance.

3. EXISTING FAULT TOLERANT STRATEGIES

The applications of Fault Tolerant Strategies on a Desktop Grid are a non trivial task, even for simple applications like those belonging to the BoT paradigm. As a matter of fact the set of Grid resources may greatly vary over time (because of resource additions and/or removals), the performance a resource delivers may vary from an application to another (because of resource heterogeneity), and may actuate over time (because of resource contention caused by applications competing for the same resource). Achieving good performance in these situations usually requires the availability of good information about both the resources and the tasks, so that a proper scheduling plan can be devised. Unfortunately, the wide distribution of Grid resources makes obtaining this information very difficult, if not impossible, in many cases. Thus, the so called knowledge-free Strategies, that do not base their decisions on information concerning the status of resources or the characteristics of applications, are particularly interesting.

3.1 WQR-FT: The WorkQueue with Replication – Fault Tolerant Strategy

WorkQueue with Replication Fault Tolerant (WQR-FT) is a knowledge free Scheduler for Bag-of-Tasks Grid applications on desktop grid. WQR-FT is able not only to guarantee the completion of all the tasks in a bag, but also to achieve performance better than alternative scheduling strategies to obtain fault tolerance. WQR-FT is obtained by adding checkpointing and replication to the WorkQueue with Replication (WQR) scheduling algorithm [15] with the purpose of keeping the number of running replicas of each task above a predefined replication threshold R[4]. In particular, when a replica of a task t dies and the number of running replicas of t falls below R, WQR-FT creates another replica of t that is scheduled as soon as a processor becomes available, but only if all the other tasks have at least one replica running.

Automatic restart ensures that all the tasks in a bag are successfully completed even in face of resource failures. However, each time a new instance must be started to replace a failed one, its computation must start from the beginning, thus wasting the work already done by the failed instance. In order to overcome this problem, WQR-FT uses checkpointing, that is the state of the computation of each running replica is periodically saved with a frequency set as indicated in [16] (we postulate the existence of a reliable checkpoint server where checkpoints are stored). In this way, the execution of a new instance of a failed task may start from the latest available checkpoint. In this paper we assume that an application may be restarted from its latest checkpoint only on a machine having the same operating system of that on which the checkpoint was taken. If such a machine cannot be found when a new replica is created (to replace a faulty one), its execution will start from the beginning.

Following algorithm details the behavior of WQR-FT.

…… Data structures and functions ……..

1. Q {is the queue of the tasks}
2. R {is the set of resources}
3. RPTH {is the maximum number of replicas for each task }
4. RPR(t) {returns the number running instances of task t}
5. resourceFree(R) {returns a resource available }
6. deleteInstances(t) {deletes all running instances of task t}
7. assign(r,t) {assigns task t to resource r}
8. assignChkpnt(r,t) {assigns task t to resource r but in this case the execution of task t on resource r will start from the checkpoint}
9. chkpnt(t,r) {returns true if task t has a checkpoint compatible with resource r}
10. processwaitingqueue(){process jobs of waiting queue}

…… WQR-FT algorithm……..

11. For all jobs t in the Queue Q
12. WaitEvent();
13. if (event =resourceFree(R)) then
14. r=getResourceFree(R);
15. t=popfront(Q);
16. if (RPR(t) < RPTH) AND chkpnt(t;r) then
17. assignChkpnt(r,t); {assigns task t to resource r so that execution will start from checkpoint}
18. else if (RPR(t) < RPTH) AND NOT chkpnt(t;r) then
19. assign(r,t); {assigns task t to resource r}
20. end if
21. else
22. Push t into waiting Queue
23. end if
24. if (event=jobComplete) then
25. deleteInstances(t);
26. processwaitingqueue();
27. end if
28. end for

The Strategy WQR-FT checks after selecting a resource(r) and a task(t) if the execution can start from a saved checkpoint (line 16 and 17). If a compatible checkpoint does not exists the execution of task t on machine r starts from the beginning (line 19).

4. PROPOSED STRATEGY

Although WQR-FT has shown good results, a knowledge-free strategy that do not use any information concerning the tasks & the resources but is not able to fully exploit the computational power of Desktop Grids. For this reason, we have proposed new fault tolerant strategy based on WQR-FT that uses the information concerning size of task, resource speed and resource reliability to improve Fault Tolerance. We show how this information can be exploited to improve both task selection (the choice of the next task to be executed) and machine selection (the choice of the machine on which it will be executed) with respect to fault-tolerant schedulers

4.1 (DFTDT) :Dynamic Fault tolerance on the basis of of delay time of resource

In our proposed strategy the selection of resources will be based upon the average of ratio of delay time of resource with respect to the task and size of task. Delay time is the difference between estimated time and actual time taken by task for its execution. Hence delay time of jth resource with respect to ith task can be calculated as:

$$D\ i\ j = (E\ i\ j - A\ I\ j)/ \text{Size}\ (T_I) \text{ where}$$

D i j is the delay time of ith task on jth resource

E i j is the estimated execution time of ith task on jth resource

A i j is the actual execution time of ith task on jth resource

Thus average delay time of Jth resource is :

$$D_j = \sum D_{ij} / N$$

Here i=1...N
J=1...M

Following algorithm details the behavior of DFTDT.

…… Data structures and functions ……..

1. Q {is the queue of the tasks}
2. R {is the set of resources}
3. RPTH {is the maximum number of replicas for each task }
4. RPR(t) {returns the number running instances of task t}
5. getResourceList();{returns all available resources}
6. resourceFree(R)) {returns a resource available }
7. deleteInstances(t) {deletes all running instances of task t}
8. assign(r,t) {assigns task t to resource r}
9. assignChkpnt(r,t) {assigns task t to resource r but in this case the execution of task t on resource r will start from the checkpoint}
10. chkpnt(t,r) {returns true if task t has a checkpoint compatible with resource r}
11. processwaitingqueue(){process jobs of waiting queue}
12. getResourceHistorytable(){returns the resource history table according to delay time }
13. manageResourceList() {Sort resourceList in ascending order according to average delay time from resource history table}
14. updateHistoryTable(){updates the resource history table}

…… DFTDT algorithm ……

15. getResourceList();
16. getResourceHistorytable();
17. manageResourceList();
18. For all jobs t in the Queue Q
19. WaitEvent();
20. if (event =resourceFree(R))
21. r=getResourceFree(R);
22. t=popfront(Q);
23. if (RPR(t) < RPTH) AND chkpnt(t;r) then
24. assignChkpnt(r,t); {assigns task t to resource r so that execution will start from checkpoint }
25. else if (RPR(t) < RPTH) AND NOT chkpnt(t;r) then
26. assign(r,t); {assigns task t to resource r}
27. end if
28. else
29. Push t into waiting Queue
30. end if
31. if (event=jobComplete)
32. updateHistoryTable()
33. deleteInstances(t);
34. processwaitingqueue();
35. end if
36. end for

As a matter of fact our strategy will manage resource list (line 17) as per the resource history table by sorting them on the basis of average delay time of all resources that has been calculated and the resources having minimum average delay will assigned the tasks first.The DFTDT algorithm is able not only to guarantee performance (Total CPU time) better than WQR-FT, but also reduce the relative waiting time and failure count occurred during execution of gridlet .

5. CONCLUSIONS AND FUTURE WORK

In this paper we propose optimum resource selection framework based on its reliability of resource to achieve maximum fault tolerance and performance. We have presented (DFTDT)- Dynamic Fault tolerance on the basis of delay time of resource with respect to the task, a fault-tolerant scheduling algorithm for Bag-of-Tasks Grid applications based on the WQR-FT algorithm. DFTDT is able not only to guarantee the completion of all the tasks in a bag, but also to achieve performance better than alternative Fault Tolerant Strategies; we can conclude that DFTDT outperforms these Strategies when Total CPU time, Waiting time and Failure Count of resources are taken into account.

As future work, we plan to study the behavior of DFTDT in more complex scenarios, such as task reliability and system load to carry on dynamic fault tolerance .We are planning to extend the DFTDT with dynamic replication and checkpointing .

REFERENCES

[1] D. Kondo, A. Chien, and H. Casanova. 2004. Resource management for rapid application turnaround on enterprise desktop grids. Proc. of Super Computing Conference.

[2] J. Abawajy. 2004. Fault-tolerant scheduling policy for grid computing systems. Proc. of 18th Int. Parallel and Distributed Processing Symposium.

[3] J. Weissman and D. Womack. September 1996. Fault tolerant scheduling in distributed networks. Department of Computer Science, University of Texas, San Antonio, Tech. Rep. TR CS-96-10.

[4] C. Anglano and M. Canonico. 2005. Fault-tolerant scheduling for bag-of-task grid applications," in Proc. of European Grid Conference. EGC 2005.

[5] P. A. Dinda. 2001. Online prediction of the running time of tasks. IGMETRICS/Performance, pp. 336–337.62.

[6] R. Wolski, N. T. Spring, and J. Hayes. 2000. Predicting the CPU availability of time-shared unix systems on the computational grid. Cluster Computing, vol. 3, no. 4, pp. 293–301.

[7] Brevik, D. Nurmi, and R. Wolski. 2004. Automatic methods for predicting machine availability in desktop grid and peer-to-peer systems. Proc. of 4th Int. Workshop on Global and Peer-to-Peer Computing.

[8] C. Anglano and M. Canonico. 2005. Fault-tolerant scheduling for bag-of-task grid applications" in Proceedings of European Grid Conference. EGC 2005.

[9] Y. Derbal. 2006. A New Fault-Tolerance Framework for Grid Computing. Journal of Multiagent and Grid Systems. vol.2, no. 2, pp. 115-133.

[10] M. T. Huda, H W. Schmidt, I D. Peake. 2005. An Agent Oriented Fault tolerant Framework for Grid Computing. Proc. of the js International Conference on e-Science and Grid Computing (e-Science'05), Melbourne, Australia, pp. 304-311.

[11] K. Vaidyanathan and K. S. Trivedi. 2001. Extended Classification of Software Faults based on Aging. 12th International Symposium on Software Reliability Engineering. Hong Kong, pp. 99.

[12] S Haider, M. Imran, I. A. Niaz, S. Ullah and M.A. Ansari. 2007. Component based Proactive Fault Tolerant Scheduling in Computational Grid. Proc. of the IEEE 3rd International Conference on Emerging Technologies (ICET 2007). Rawalpindi, Pakistan.

[13] B. Nazir and T. Khan. 2006. Fault Tolerant Job Scheduling in Computational Grid . Proc. of the IEEE 2nd International Conference on Emerging Technologies (ICET2006), Peshawar, Pakistan, pp. 708-713.

[14] C. Anglano, J. Brevik and et al. Sept. 2006. Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids In Proc. of 7th IEEE/ACM International Conference on Grid Computing, Barcelona, Spain,. IEEE Press.

[15] D.P. da Silva, W. Cirne and et al. 2003. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In Proc. of EuroPar 2003, volume 2790 of Lecture Notes in Computer Science.

[16] J.W. Young. 1974. A First-order Approximation to the Optimum Checkpoint. Communications of the ACM, 17.