ISSN 2277-3061

TheWorkQueue with Dynamic Replication-FaultTolerant Scheduler in Desktop Grid Environment

Jyoti Bansal [1], Dr. Shaveta Rani[2], Dr. Paramjit Singh[3]
[1] Associate Prof. & Dean Academic Affairs, B.F.C.E.T., Bathinda
erjyoti.2009@rediffmail.com
[2] Associate Prof. of CSE deptt, PTUGZS Campus ,Bathinda.
garg_shavy@yahoo.com
[3] Associate Prof. of CSE deptt, PTUGZS Campus ,Bathinda.
param2009@gmail.com

ABSTRACT

Desktop Grid is different from Grid in terms of the characteristics of resources as well as types of sharing. Particularly, resource providers in Desktop Grid are volatile, heterogeneous, faulty, and malicious. These distinct features make it difficult for a scheduler to allocate tasks. Moreover, they deteriorate reliability of computation and performance. Availability metrics can forecast unavailability & can provide schedulers with information about reliability which helps them to make better scheduling decision when combined them with information about speed. This paper using these metrics for deciding when to replicate jobs & how much to replicate. In particular our metrics forecast the probability that a job will complete uninterrupted & our schedulers replicate those jobs that are least likely to do so. Our policy outperforms other replication policies as measured by improved Total CPU Time & reduced Waiting Time & Failure count.

Indexing terms/Keywords

Desktop Grid Scheduling ,WQR, BoT, WQR-FT, WQDR-FT

Academic Discipline And Sub-Disciplines

Computer science and Engineering .

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

1. INTRODUCTION

Grid computing technology provides resource sharing and resource virtualization to end-users, allowing for computational resources to be accessed as a utility. By dynamically coupling computing, networking, storage, and software resources, Grid technology enables the construction of virtual computing platforms capable of delivering unprecedented levels of performance. However, in order to take advantage of Grid environments, suitable application-specific scheduling strategies, able to select, for a given application, the set of resources that maximize its performance, must be devised [1]. The inherent wide distribution, heterogeneity, and dynamism of Grid environments makes them better suited to the execution of loosely-coupled parallel applications, such as Bag-of-Tasks [2] (BoT) applications, rather than of tightly-coupled ones. Bag-of-Tasks applications (parallel applications whose tasks are completely independent from one another) are particularly able to exploit the computing power provided by Grids [3] and, despite their simplicity, are used in a variety of domains, such as parameter sweep, simulations, fractal calculations, computational biology, and computer imaging. Therefore, scheduling algorithms tailored to this class of applications have recently received the attention of the Grid community [3, 4, 5]. Although these algorithms enable BoT applications to achieve very good performance, they suffer from a common drawback, namely their reliance on the assumption that the resources in a Grid are perfectly reliable, i.e. that they will never fail or become unavailable during the execution of a task. Unfortunately, in Grid environments faults occur with a frequency significantly higher than in traditional distributed systems, so this assumption is overly unrealistic. A Grid may indeed potentially encompass thousands of resources, services, and applications that need to interact in order for each of them to carry out its task. The extreme heterogeneity of these elements gives rise to many failure possibilities, including not only independent failures of each resource, but also those resulting from interactions among them. Moreover, resources may be disconnected from a Grid because of machine hardware and/or software failures or reboots, network misbehaviors, or process suspension/abortion in remote machines to prioritize local computations. Finally, configuration problems or middleware bugs may easily make an application fail even if the resources or services it uses remain available [6].

In order to hide the occurrence of faults, or the sudden unavailability of resources, fault-tolerance mechanisms (e.g., replication or checkpointing-and restart) are usually employed. Although scheduling and fault tolerance have been traditionally considered independently from each other, there is a strong correlation between them. As a matter of fact, each time a fault-tolerance action must be performed, i.e. a replica must be created or a checkpointed job must be restarted, a scheduling decision must be taken in order to decide where these jobs must be run, and when their execution must start. A scheduling decision taken by considering only the needs of the faulty task may thus strongly adversely impact non-faulty jobs, and vice versa. Therefore, scheduling and fault tolerance should be jointly addressed in order to simultaneously achieve fault tolerance and satisfactory performance. Fault-tolerant schedulers [7, 8, 9] attempt to do so by integrating scheduling and fault management, in order to properly schedule both faulty and non-faulty tasks. However, to the best of our knowledge, no fault-tolerant scheduler with dynamic replication for BoT applications has been proposed in the literature. This paper aims at filling this gap by proposing a novel fault-tolerant scheduler with dynamic replication for BoT applications, in which resources will be selected not only on the basis of computation and memory power but also on the basis of resource reliability.

The rest of the paper is organized as follows. In section 2, we review some related works. In Section 3 we discuss the performance of a Knowledge free Scheduler called Work Queue with Replication(WQR) is an extension of the classical WorkQueue(WQ) Scheduling algorithm & performance of a knowledge free fault –tolerant scheduler called Work Queue with Replication and Fault Tolerant(WQR-FT). In Section 4 we discuss how it is possible to build The WorkQueue with Dynamic Replication – Fault Tolerant Scheduler able to outperform The WorkQueue with Replication - Fault Tolerant Scheduler(WQR-FT).Finally, Section 5 concludes the paper & outlines future research work.

2. RELATED WORK

Existing algorithms for scheduling BoT applications on Desktop Grids can be classified along two dimensions, namely (a) their reliance on task/resource information (i.e., we have knowledge-free and knowledge-aware strategies), and (b) the way they handle resource failures (i.e., we have fault-agnostic and fault-aware strategies). Although this classification gives rise to four different combinations, the literature provides examples belonging to only three of them.

knowledge-free schedulers [10] adds task replication to the classical Workqueue(WQ) scheduler to avoid task failures near the end of the application, and unpredictably slow hosts can cause major delays in application execution. Using the replication approach, hosts are assigned to execute replicas of tasks that are still running. Tasks are replicated until a predefined maximum number of replicas are achieved. When a task replica finishes, its other replicas are canceled. This policy has the drawback of wasting CPU cycles (due to the replicas that do not contribute to the completion of the tasks), which could be a problem if the Desktop Grid is to be used by more than one application. Knowledge-based fault-agnostic schedulers[11] rely on resource/task information, but are based on the implicit assumption that resources never fail. Schedulers in this class assume the knowledge of the execution time of individual tasks, and exploit various type of static [12], [13] or dynamic [11], [14] resource information to perform machine selection. Knowledge-free, fault-tolerant schedulers[7,15] improve over their knowledge free counterparts by using task replication to reduce the effects of poor task assignments, and automatic restart (possibly coupled with checkpointing) to deal with resource failures.

An alternative approach to BoT scheduling in Desktop Grids named fault-aware schedulers has been proposed in [16] that, rather than just tolerating faults as done by traditional fault-tolerant schedulers, exploit the information concerning resource availability to improve application performance.An extension of this approach has been proposed in [17] that uses three general techniques for resource selection: resource prioritization, resource exclusion, & task duplication. we used these technique to instantiate several scheduling heuristics.

A decentralized scheduler for BoT applications on desktop grids has been proposed in [18] which ensures a fair and efficient use of the resources. It aims to provide a similar share of the platform to every application by minimizing their maximum stretch, using completely decentralized algorithms and protocols.

3. EXISTING SCHEDULERS

Scheduling applications on a Grid is 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 schedulers, that do not base their decisions on information concerning the status of resources or the characteristics of applications, are particularly interesting.

3.1 The Standard WQR Scheduler

In the classical WorkQueue (WQ) scheduling algorithm, tasks in a bag are chosen in an arbitrary order and are sent to the processors as soon as they become available. WQR adds task replication to WQ in order to cope with task and host heterogeneity, as well as with dynamic variations of the available resource capacity due to the competing load caused by other Grid users.WQR works very similarly to WQ, in the sense that tasks are scheduled the same way. However, after the last task has been scheduled, WQR assigns replicas of already-running tasks to the processors that become free (in contrast, WQ leaves them idle). Tasks are replicated until a predefined replication threshold is reached. When a tasks replica terminates its execution, its other replicas are canceled. By replicating a task on several resources, WQR increases the probability of running one of the instances on a faster machine, thereby reducing task completion time. As shown in [3], WQR performance are equivalent to solutions that require full knowledge about the environment, at the expenses of consuming more CPU cycles.

3.2 The WorkQueue with Replication – Fault Tolerant Scheduler

In its original formulation, WQR does not do anything when a task fails. Consequently, it may happen that one or more tasks in a bag will not successfully complete their execution. In order to obtain fault tolerance, we add automatic restart, with the purpose of keeping the number of running replicas of each task above a predefined replication threshold R[15]. 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 [19] (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.

Following algorithm details the behavior of WQR-FT.

```
1: ----------- data structures and functions --------------
2:  Q {is the queue of the tasks (Qi is the ith task)}
3:  R {is the set of resources (Ri is the ith resource)}
4:  RLTH {is the maximum number of available replicas for each task}
5:  RLR(t) {returns the number running instances of task t}
6:  getRscFree(R) {returns a resource available}
7:  deleteInstances(t) {deletes all running instances of task t}
8:  allocate(r,t) {assigns task t to resource r}
9:  allocateChk(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: --------------WQR-FT algorithm ---------------
12: while Q is not empty do
13: wait(event)
```

```
14: if (event == "RscFree") then
15: r=getRscFree(R); {r is a resource available}
16: t=pop front(Q); {extracts the first task t of the queue}
17: if (RLR(t) < RLTH) AND chkpnt(t; r) then
18: allocateChkpnt(r,t); {assigns task t to resource r}
19: else if (RLR(t) < RLTH) AND NOT chkpnt(t; r) then
20: allocate(r,t); {assigns task t to resource r}
21: end if
22: pushback(Q,t); {adds task t to the end of the queue}
23: else {event=="TaskDone"}
24: deleteInstances(t); {deletes all running instances of task t}
25: end if
26: end while
```

As a matter of fact, once that a resource and a task has been selected (lines 15 and 16), the scheduler checks if the execution can start from a saved checkpoint (line 17 and 18). If a compatible checkpoint does not exists the execution of task t on machine r starts from the beginning (line 20).

4. PROPOSED SCHEDULER

WorkQueue with Replication is a knowledge-free scheduling algorithm that adds task replication to the classical WorkQueue scheduler. WQR-FT, adds both automatic restart and checkpointing to WQR, and properly coordinates the scheduling of faulty and non-faulty tasks in order to simultaneously achieve fault-tolerance and good application performance.

Our scheduler, WQDR-FT: A Fault-Tolerant Scheduler with dynamic replication for BoT Applications adds dynamic replication to WQR-FT, in which resources will be selected not only on the basis of computation and memory power but also on the basis of resource reliability.

4.1 The WorkQueue with Dynamic Replication – Fault Tolerant Scheduler

Although WQR-FT has shown good results, we believe that knowledge free schedulers cannot exploit the full potential of Desktop Grids, as these algorithms usually require the use of much more resources than necessary in order to tolerate some bad decisions made when these algorithms usually are using same replication threshold for all tasks.

Following algorithm details the behavior of WQDR-FT.

```
1: ----------- data structures and functions ---------------
2:  Q {is the queue of the tasks (Qi is the ith task)}
3:  R {is the set of resources indexed according to Success Rate (Ri is the ith resource)}
4:  T { is the total no of successful jobs)
5:  N {Total no of jobs submitted for execution}
6:  SR  { Success rate }
7:  RLTH {is the maximum number of available replicas for each task calculated dynamically}
8:  RLR(t) {returns the number running instances of task t}
9:  getRscFree(R) {returns a resource available}
10: deleteInstances(t) {deletes all running instances of task t}
11: allocate(r,t) {assigns task t to resource r}
12:  allocateChk(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}
13: chkpnt(t,r) {returns true if task t has a checkpoint compatible with resource r}
14: getresourcelist()
15: getResourceHistory()
16: --------------WQDR-FT algorithm ---------------
```

```
17: getResourcelist();
18:getResourceHistory();
19:sort resource list according to Resource history.
20:while Q is not empty do..
21: wait(event)
22: if (event == "RscFree") then
23: r=getRscFree(R); {r is a resource available}
24: t=pop front(Q); {extracts the first task t of the queue}
25: SR=T/N;
26: RLTH=1+1/SR;
27: if (RLR(t) < RLTH) AND chkpnt(t; r) then
28: allocateChkpnt(r,t); {assigns task t to resource r}
29: else if (RLR(t) < RLTH) AND NOT chkpnt(t; r) then
30: allocate(r,t); {assigns task t to resource r}
31: end if
32: push back(Q,t); {adds task t to the end of the queue}
33: else {event=="TaskDone"}
34:updateResourceHistoryTable();
35: deleteInstances(t); {deletes all running instances of task t}
36: end if
37: end while
```

Our algorithm tries to find a node with the most suitable resource for a task by calculating the Success Rate of all the resources & then this value will be used to sort the resources as per the Resource History table (lines 19) and the resources having maximum relibility will assigned the tasks first & then Replication Threshold(RLTH) value(line 26) for each task will be calculated to make the replication of tasks on the resources dynamic.

5. CONCLUSION AND FUTURE WORK

In this paper we have presented WQDR-FT, a fault-tolerant scheduler with dynamic replication for Bag-of-Tasks Grid applications based on the WQR-FT algorithm. By considering the dynamic threshold value for replication, WQDR-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. As a matter of fact, being WQR-FT able to attain performance higher than other (non fault-tolerant) alternative strategies, and being WQDR-FT to achieve performance better than WQR-FT, we can conclude that WQDR-FT outperforms these strategies when resource failures and/or unavailability's are taken into account.

There are a number of ways in which this work can be extended. For example, checkpoint policies can be improved. First of all, in our study we assumed that tasks submitted always terminate, but in the case of buggy task that never terminates because it cores dump half way through, the scheduling algorithms proposed never going to terminate. In order to avoid this situation, we can consider a timeout technique: a task will be forced to terminate if it has not completed yet within a certain amount of time. In this way, the BoT will always finish its execution, and the broker can execute the next BoT.

Also the checkpoint policy can be improved considering a specific process on each machine that carry out to save the checkpoint in order to reduce the suspension of the task execution. Moreover, each time the process must save its checkpoint, it should be able to decide if it is effectively useful to save the checkpoint or it is better to retrieve a newer checkpoint.

REFERENCES

[1] F. Berman, R. Wolski and et al. April 2004. Adaptive Computing on the Grid Using AppLeS. IEEE Trans. on Parallel and Distributed Systems, 14(4).

[2] W. Cirne and et al. Sept. 2003. Grid Computing for Bag of Tasks Applications. In Proc. of 3rd IFIP Conf. on E-Commerce, E-Business and E-Government, Sao Paulo, Brazil.

[3] 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.

[4] H. Casanova, F. Berman and et al. 2000. The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid. In Proc. of Supercomputing 2000. IEEE CS Press.

[5] H. Casanova, A. Legrand and et al. 2000. Heuristics for Scheduling Parameter Sweeping Application in Grif Environments. Proc. of Heterogeneous Computing Workshop. IEEE CS Press.

[6] R. Medeiros, W. Cirne and et al. Nov. 2003. Fault in Grids: Why are they so bad and What can be done about it?. In Proc. 4th Int. Workshop on Grid Computing (Grid 2003). IEEE-CS Press.

[7] J.H. Abawajy and et al. April 2004. ault-Tolerant Scheduling Policy for Grid Computing System. Proc. of 18th Int. Parallel and Distributed Processing Symposium, Workshop on IEEE-CS Press.

[8] J. Weissman and et al. Sept. 1996. Fault Tolerant Scheduling in Distributed Networks. Technical Report TR CS-96-10, Department of Computer Science, University of Texas, San Antonio.

[9] X. Zhang, D. Zagorodnov and et al. Sep. 2004. Fault tolerant Grid Services Using Primary-Backup: Feasibility and Performance. In Proc. IEEE Int. Conf. on Cluster Computing. IEEE-CS Press.

[10] 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 Proceedings of Euro-Par 2003, volume 2790 of Lecture Notes in Computer Science.

[11] F. Berman, R. Wolski and et al. 2004. Adaptive computing on the grid using apples. IEEE Trans. on Parallel and Distributed Systems, vol. 14.

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

[13] H. Casanova, A. Legrand and et al. 2000.Heuristics for scheduling parameter sweep applications in grid environments. Proceedings of the 9th Heterogeneous Computing Workshop.

[14] H. Casanova, F. Berman and et al. 2000. The apples parameter sweep template: User level middleware for the grid . Proc. of Supercomputing 2000.

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

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

[17] D. Kondo, A. Chien and et al. 2007. Scheduling Task Parallel Applications for Rapid Application Turnardound on Enterprise Desktop Grids. Journal of Grid Computing.

[18] Javier Celaya and et al. 2010. A Fair Decentralized Scheduler for Bag-of-Tasks Applications on Desktop Grids. CCGRID '10 Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing . IEEE Computer Society Washington, DC, USA .

[19] J.W. Young and et al. 1974. A First-order Approximation to the Optimum Checkpoint. Communications. ACM, 17.