Checkpointing in Selected Most Fitted Resource Task Scheduling in Grid Computing

Rohaya Latip
College of Computer Science and Information Systems
Najran University, Najran, KSA.
Faculty of Computer Science and Information Technology
University Putra Malaysia
Serdang Malaysia
rohaya@fsktm.upm.edu.my

Lew Wai San
Faculty of Computer Science and Information Technology
University Putra Malaysia
Serdang Malaysia
Wai-San.Lew@teito.com

Fara Habib Chanchary
College of Computer Science and Information Systems
Najran University, Najran, KSA.
chanchary@gmail.com

Abstract— Grid applications run on environment that is prone to different kinds of failures. Fault tolerance is the ability to ensure successful delivery of services despite faults that may occur. Our research adds fault tolerance capacity with checkpointing and machine failure, to the current research, Selected Most Fitted (SMF) Task Scheduling for grid computing. This paper simulates one of fault tolerance techniques for grid computing, which is implementing checkpointing into Select Most Fitting Resource for Task Scheduling algorithm (SMF). We applied the algorithm of MeanFailure with Checkpointing in the SMF algorithm and named it MeanFailureCP-SMF. The MeanFailureCP-SMF is simulated using Gridsim with initial checkpointing interval at 20% job execution time. Results show that with MeanFailureCP-SMF has reduce the average execution time (AET) compare to the current SMF and MeanFailure Algorithm.

Keywords- Fault Tolerance, Checkpointing, GridSim, Job Scheduling, Grid Computing

I. INTRODUCTION

Task Scheduling in grid computing is complex because it is not subject to centralized control [1]. The grid has a pool of resources from different location enable it to become a powerful computing infra structure. A good scheduling is needed not just to handle the complexity of the resources but also to handle the fault tolerant when the machine fails while the machine is running a job. Having a proactive approach will reduce the execution time and number of jobs failure.

In this paper, we proposed a checkpointing method in the current approaches such as SMF and MeanFailure Algorithm to reduce the execution time for a job to be completed.

This paper is organized as follows; Section II discussed the related works, Section III elaborates the algorithm which is the MeanFailureCP-SMF algorithm. Section IV explains on the simulation design and implementations using GridSim. Section V discussed on the results and lastly, Section VI is concluding this paper.

II. RELATED WORKS

A. Select Most Fitting (SMF) Resource Algorithm

SMF is a dynamic scheduling algorithm, where it decides on most fitting resource for user's submitted job during scheduling. Global Information System (GIS) categorizes resources into *L* discrete levels according to the capability. We use speed of machine (MIPS) as job's resource requirement in our work whereas in [2], they use predicted execution time (PET). The calculation on PET consumes some time period and needs more works, hence we go for a simpler job's resource requirement. A task that assigned to machine must have a minimum requirement that belongs to the machine's level. SMF defines a closeness factor, *C* on job resource requirement, in order to get better prediction and assign jobs to closet machine, and avoid delay. The closeness factor plays important role for the stability in task average execution task (AET).

The steps of SMF:

1. Categorize resource into *L* discrete levels.

2. Based on task resource requirement and closeness factor, starting from smallest level, it searches for an available node that meets the needs.
3. If node found, the task is assigned. Else it will search for a free node upward, from level r_{j+1}, r_{j+2}, r_{j+3} to r_L, and assign task to the first free node found.
4. If node not found, it finds downward from level r_{j-1}, r_{j-2}, r_{j-3} to r_1, and assign task to the first free node found.
5. If task still not assigned, it has to wait in queue.

B. Checkpointing

As defined in [3], checkpointing is a process to save the state of a running application to a stable storage that can be used to resume execution of the application in case of any failures. Application can start from its last saved point instead of from the beginning, thus reduce the execution time.

Based on level of abstraction, Checkpointing can be categorized into system level, user/application level or hybrid. The Table 1 lists the merits and demerits between system level and application level check point. Another criterion to look at is whether to check point the entire application state or just save the modified state from program's last check point. The later is called incremental checkpointing. Besides, based on the scope of the check point, there are local and global check points. Check point of every separate process is called local check point whereas a set of each process's local check point is named global check point.

TABLE 1. Comparison between system level and application level check point based on transparency, portability, check-point size, flexibility and check point generation.

Category	Merit	Demerit
System level	1. Transparent to user, little effort from programmer. 2. Easy to use. 3. Can generate check point forcefully; check point state can be obtained directly from main memory by separate thread/process.	1. Less portable. 2. Large check point size and performance overhead. 3. Less flexible.
User /Application level	1. Quite portable, check point can be saved in platform independent format. 2. Can be coded to reduce check point size and performance overhead, only store minimal check point needed for restart. 3. More flexible.	1. User-driven, require more effort from programmer. 2. Cannot force check point generation as program state can only be saved when check point generation code is executed.

In [3], there are different checkpointing models/algorithms proposed for job scheduling. New form of checkpointing concept lets application programmer, compiler and runtime system to evaluate the necessity of each check point, namely cooperative or adaptive checkpointing. The two cooperative checkpointing models suggested in [4] are: Last Failure Dependent Checkpointing (LastFailureCP) and Mean Failure Dependent Checkpointing (MeanFailureCP). LastFailureCP omits unnecessary checkpoints on stable resource by considering job estimated total execution time and last detected failure (LF_r) of resource. Whereas MeanFailureCP dynamically update checkpointing frequency/interval based on remaining job execution time and average failure interval (MF_r) of resource. In the experiment, both of them are compared with unconditional periodic job check-pointing (PeriodicCP) which performs identically on stable or volatile resource. Results show that LastFailureCP performs better for short check-pointing intervals but slightly worse than PeriodicCP for large check-pointing intervals. However, when failures occur quite periodically, LastFailureCP provides results close to PeriodicCP. MeanFailureCP is proved to be the most effective solution.

C. MeanFailureCP Algorithm

The algorithm schedules checkpointing interval in a dynamic way based on remaining job execution time and machine average failure interval:

1. Schedule initial checkpointing at short time period after job starts running.
2. Next scheduling interval is dependent on machine mean failure interval. Algorithm updates new checkpointing interval by topping up the initial checkpointing interval, $I_{new} = I_{old} + I_{init}$.
3. If inequality remaining execution time < mean failure interval and $I_{new} < \alpha$ * execution time is false, new chekpointing interval is decreased to $I_{new} = I_{old} - I_{init}$.

III. THE ALGORITHM

We applied fault tolerance capacity – checkpointing to SMF and combined it with MeanFailureCP. We named the algorithm MeanFailureCP-SMF. MeanFailureCP-SMF is a dynamic checkpointing interval approach.

The algorithm of MeanFailureCP-SMF is as below:

1. Resources register to GIS and machines are grouped into *L* discrete levels based on node speed MIPS.
2. User submits jobs with resource requirement and closeness factor.
3. Starting from smallest level, it searches for an available node that meets the needs.
4. If node found, the job is assigned. Else it will search for a free node upward, from level r_{j+1}, r_{j+2}, r_{j+3} to r_L, and assign job to the first free node found.

5. If node found, the job is assigned. Else it finds downward from level $r_{j-1}, r_{j-2}, r_{j-3}$ to r_1, and assign job to the first free node found.
6. If the job still not assigned, it will wait in queue.
7. When job starts running, it schedules initial checkpointing at a short time period.
8. During first checkpointing interval, executing job image is sent to resource's storage machine. Storage saves job image and informs resource upon completion.
9. Algorithm schedules next checkpointing based on remaining execution time and machine failure interval information kept at GIS.
10. Next scheduling interval is dependent on machine mean failure interval. Algorithm updates new checkpointing interval by topping up the initial checkpointing interval, $I_{new} = I_{old} + I_{init}$.
11. If the inequality remaining execution time < mean failure interval and $I_{new} < \alpha$ * execution time is false, new chekpointing interval is decreased to $I_{new} = I_{old} - I_{init}$.
12. During next intervals, instead of whole job image, resource only sends updated attributes/properties of the job to storage. Storage updates job attributes/properties, and sends back notification once finishes the process.
13. Resource triggers checkpoint remove event when job is completed. It notifies storage machine to remove the saved checkpoint to free storage space.
14. On machine failure or when the resource is out of order, storage machine notifies job's corresponding user whether to load saved checkpoint.
15. When user receives the load checkpoint request, it will use SMF to choose a fitting free machine in same or from other resources, to resume execution of the job.

IV. SIMULATION DESIGN

GridSim is a Grid simulator based on SimJava, a general purpose discrete-event simulation package implemented in Java. The new release of GridSim version 4.0 adds package *gridsim.resFailure* for resource failure detection that uses pull mechanism [5]. Therefore this package was used in our simulation.

We define Million Instructions per Second (MIPS) as job requirement in our experiments. GIS is responsible to group the nodes register to it into discrete levels based on different MIPS. We use the same simulation parameters as in [2], refer Table 2. We generate a set of speed of machine and a set of task length to be used by all of our experiments.

TABLE 2. Simulation Parameters

Parameter	Value
Number of Task	1000
Number of Resource Nodes	200
Speed of Resource Node (MIPS)	1000 – 4000
Workload of Task / Task Length in Million Instruction (MI)	300000 – 500000

In our simulation, each user submits jobs to resource entities at an interval of every 100 seconds. Each machine is configured to have just one processing element. We use a Regional GIS and all the entities are from the same region. No job retries policy. Only independent tasks and resource /machine failures are considered in our experiments. Machine failure time generations are based on a set of constant values for all simulations, and the pattern is every 150 seconds.

Each User has its local scheduling agent. Resource registers to GIS and GIS asks for resource characteristics, e.g. free machine from resource. User retrieves available resource list from GIS. If user received returned state from resource that indicates resource is out of order, user will inform GIS to update the available resource list. User sends and receives jobs from resource. Resource notifies user on job status, and machine state. Polling mechanism is applied in the communication between resource and its checkpoint storage node. Resource triggers task checkpointing event to storage at some time intervals, and storage notifies resource on completion of process.

The flows and steps of the simulation and experiments are as following:

1. We start simulate SMF with 0% machine failure rate.
2. Secondly, we run an experiment to get the best resource requirement closeness factor to be used in SMF. The 'best' will go for the smallest deviation time, the largest execution time minus the smallest execution time for our jobs. Since 40% closeness factor has the lowest deviation time, we take this.
3. Thirdly, we simulate environments of different machine failure rates, and examine on AET and number of failed jobs. This is to get the best machine failure rate that SMF can perform. 20% of machine failure rate achieves most stable AET and lowest total AET. Hence we pick 20% as the machine failure rate for all our following experiments. Figure 1 shows the AET over the simulation time for SMF with 0% machine failure and SMF with 20% machine failure.
4. We plug in *MeanFailureCP* algorithm to SMF. We inspect on AET and failed jobs reduced rate. The metric 'failed jobs reduced rate' is calculated by number of successfully resumed checkpoint / number of failed jobs.

V. RESULT AND DISCSSIONS

Paper [2] simulated SMF but without any machine failure. Therefore with a little effort, the SMF algorithm was simulated with machine failure of 20% and the results are shown in Average Execution Time (AET). 20% machine failure is required for analyzing the results with other algorithms where they have 20% machine failure embedded. The AETs shown in Figure 1 are lesser compared to the simulation result in [2], this may due to time spent on calculation of Predict Execution Time(PET) and grouping of tasks into discrete categories for the usage of PET. After applying 20% machine failure rate to SMF, the graph still maintain the stability, but it had increased the total AET to 1.8%.

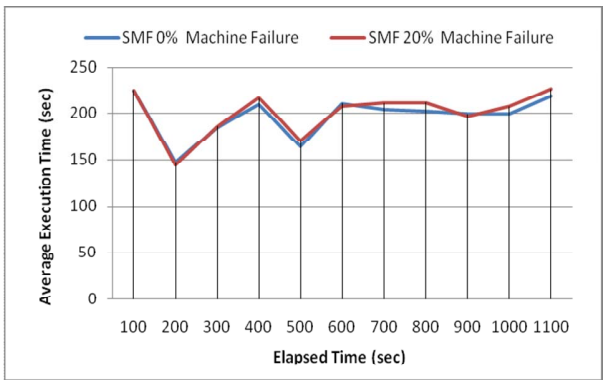

Figure 1: SMF with 0% machine failure and SMF with 20% machine failure.

Figure 2 shows the results of AET on SMF, MeanFailureCP and our algorithm, Mean FailureCP-SMF. Comparing the current algorithm, MeanFailureCP with SMF, the MeanFailureCP is 3.4% higher but by combining both algorithms it has reduce the AET to 0.9% lower. If we compare the current algorithm, MeanFailureCP with our algorithm it has reduced to 4.1% of AET.

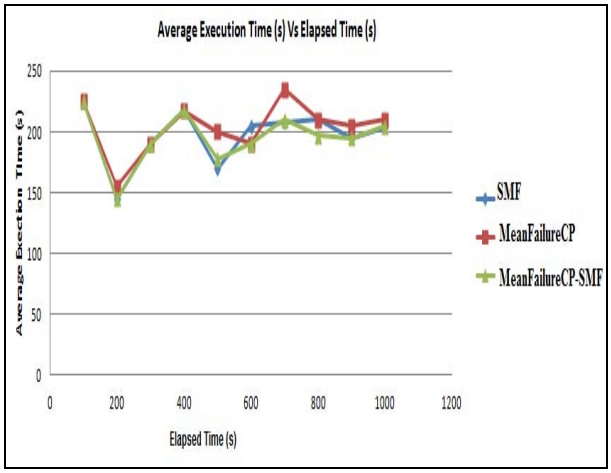

Figure 2: Results of AET for SMF, MeanFailureCP and MeanFailureCP-SMF.

VI. CONCLUSIONS

SMF was better than MeanFailureCP in term of Average Execution Time (AET) where the MeanFailureCP is 3.4% higher compare to SMF. By combining the SMF and MenaFailureCP, it has reduced the AET to 4.1%. By reducing the execution time, the task scheduling is enhanced and making it more reliable for job submission even there is machine failure in a complex infrastructure called grid computing.

In the Future, our algorithm should be improved to be able to search for free holes in a more efficient way. It should broadcasts and moves waiting jobs into execution as soon as resource has completed jobs, without having to wait for next round of scheduling event.

ACKNOWLEDGMENT

We would like to thank you Najran University, Kingdom of Saudi Arabia for providing the research funding for this project. We would also like to thank you University Putra Malaysia, Serdang Malaysia for giving the opportunity to continue doing the research and support us until we successfully completed the project.

REFERENCES

[1] G. Sudha Sadasivam, V. Viji Ranjendran,"An Efficient Approach to Task Scheduling in Computational Grids", Int. Journal of Computer Science and Applications, Vol. 6(1), pp.53-69, 2009.

[2] R.S. Chang, C.F. Lin, J.J. Chen, "Selecting the Most Fitting Resource for Task Execution", Future Generation Computer Systems, Vol 27, pp. 227-231, 2011.

[3] S.T.L. Anthony, G.Sumathi, S.Anthony Dalya, "Dynamic Adaptation of Checkpoints and Rescheduling in Grid Computing", Int. Journal of Computer Applications, Vol. 2(3). pp. 95-99, 2010.

[4] M. Chtepen, et. al. "Adaptive Task Checkpointing and Replication : Toward Efficient Fault Tolerent Grids", IEEE Transaction on Parallel and Distributed Systems, Vol. 20(2). Pp. 180-190, 2009.

[5] A. Caminero, A. Sulistio, B. Caminero, C. Carrion, R. Buyya, "Extending GridSim with an Architecture for Failure Detection," International Conference on Parallel and Distributed Systems, pp. 1-8, 2007.