IJCSN International Journal of Computer Science and Network, Volume 2, Issue 3, June 2013
ISSN (Online) : 2277-5420 www.ijcsn.org

A New Checkpoint Approach for Fault Tolerance in Grid Computing

[1] Gokuldev S, [2]Valarmathi M

[1]Associate Professor, Department of Computer Science and Engineering
SNS College of Engineering, Coimbatore, Tamil Nadu, India

[2] PG Scholar, Department of Computer Science and Engineering
SNS College of Engineering, Coimbatore, Tamil Nadu, India

Abstract

Computational and Service grid are used to solve large-scale scientific application using grid resources. The main focus is on fault identification, fault rectification (fault tolerance) using checkpoint approaches. In order to achieve the fault tolerance, checkpoint approach can be used. Job check pointing is one of the most common utilized techniques for providing fault tolerance in computational grids. The effectiveness of check pointing depends on the choice of the checkpoint interval. A common technique for fault tolerance is dynamically adapting the checkpoint, in which all the failure information are maintained in the Grid Information Server. This requires a separate server for storage purpose in order to increase the execution time. The main goal of checkpoint approach is to minimize the overall execution time in grid system. In this work fault tolerant scheduling is achieved using kernel-level checkpoint. In case of resource failure, the Fault Index Based Rescheduling (FIBR) algorithm is used to reschedule the jobs to some other available resources. This ensures that the job is executed with minimized execution time.

Keywords: *Fault tolerant, Computational grid, Service grid, Checkpoint, Grid Information Server.*

1. Introduction

Grid is a system that coordinates resources that are not subject to centralized control using standard, open, general- purpose interfaces and protocols to deliver non-trivial qualities of service. A grid is a type of parallel and distributed system that enables the allocation, selection and aggregation of resources distributed across multiple administrative domains based on their (resources) availability, capacity, performance, cost and quality of service requirements.

A grid consists of shared heterogeneous computing and data resources networked across administrative boundaries. Grid computing (or the use of a computational grid) is applying the resources of many computers in a network to a single problem at the same time. Grid computing is the federation of computer resources from multiple administrative domains to reach a common goal. A grid is a collection of machine sometimes referred to as nodes, resources, donors, members, clients, hosts, engines and many other such terms. Overview of grid infrastructure shown in fig.1 contains number of resources, Grid Information Server (GIS) and Resource Broker (RB).

Grid environments are extremely heterogeneous and dynamic, with components joining and leaving the system all the time, more faults are likely to occur in grid environments. Also, the likelihood of errors occurrence is exacerbated by the fact that many grid applications will perform long tasks that may require several days of computation. This will lead to a number of new conceptual and technical challenges to fault-tolerance researchers. The most important one is the scheduling of user jobs to grid resources with meeting the user's Quality of Service (QoS) in existence of resource faults.

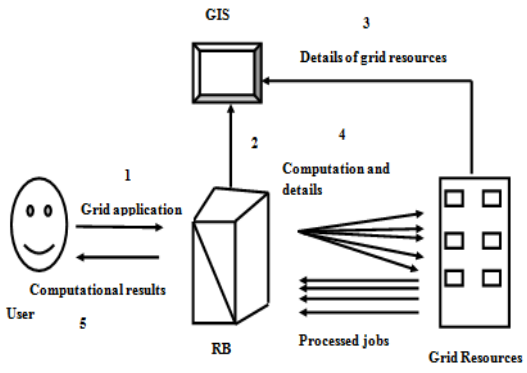

Fig.1. Overview of Grid Infrastructure

Computational grids can be defined as an environment that organizes geographically distributed and heterogeneous resources in different administrative domains with different security polices into a single computing system [1]. It enables users to use its resources for large-scale computing applications in science, engineering and commerce.

Fault tolerance is preserving the delivery of expected services despite the presence of fault-caused errors within the system itself. Errors are detected and corrected. Permanent faults are located and removed while the system continues to deliver acceptable services. In computational grids, fault tolerance is important as the reliability of grid resources may not be guaranteed.

Fault tolerance is the ability of a system to perform its function correctly even in the occurrence of faults. The

fault tolerance makes the system more dependable [2]. A complementary but separate approach to increase dependability is fault prevention. A failure occurs when an actual running system deviates from this specified behavior.

2. Related work

Review of literature reveals that a large number of research efforts have already been devoted to tolerate faults in computational grids. job checkpointing and Job replication are the two often used techniques to accomplish fault tolerance in computational grids [1]. Job replication is based on the assumption that the probability of a single resource failure is much higher than of a simultaneous failure of multiple resources. It avoids job recomputation by starting several copies of the same job on different resources. With redundant copies of a job, the grid can continue to provide a service inspite of failure of some grid resources carrying out job copies without affecting the performance.

Fault tolerant measures in grid environment [8] are different from those of general distributed systems. 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. Therefore, fault tolerance is becoming a crucial area in grid computing.

In the grid atmosphere in case of a resource failure, an application is restart on another grid resource. If the application execution state is saved, then the application can be restart from its last successful state. To store the state of the application, the checkpoint files are required. The checkpoint files are stored in a checkpoint server.
Job checkpointing is the ability to save the state of a running job to a stable storage to reduce the fault recovery time. In case of fault, this save state can be used to restart execution of the job from the point in computation where the check-point was last registered instead of restarting the application from its very beginning.

This can reduce the execution time to a large extent. The effectiveness of checkpointing mechanism is strongly dependent on the length of the checkpointing period. The checkpointing period is the duration between two checkpoints. This paper focuses on job scheduling with check pointing based fault tolerant strategy along with the Kernel level checkpoint service for the computational and service grid environment. Grid jobs are performed by the computational grid as follows:

- Grid users submit their jobs to the grid scheduler by specifying their QoS requirements, i.e., deadline in which users want their jobs to be executed, the number of processors and class of operating system.
- Grid scheduler schedule user jobs on the best available resources by optimizing time.
- Result of the job is submit to user upon successful completion of the job.

Such a computational grid atmosphere has two major drawbacks:

- If a fault occurs at a grid resource, the job is reschedule on another resource which eventually results in failing to satisfy the user's QoS requirement i.e. deadline. The motivation is simple. As the job is re executed, it consumes more time.
- In the computational based grid environment, there are resources that fulfill the criterion of deadline constraint, but they have a tendency toward faults.

In such a scenario, the grid scheduler goes ahead to select the same resource for the mere reason that the grid resource promises to meet user's requirements of the grid jobs. This ultimately results in compromising the user's QoS parameters in order to complete the job.

The work on Grid fault tolerance can be divided into pro-active and post-active mechanisms. In pro-active mechanisms, the failure consideration for the grid is made before the scheduling of a job, and dispatch with hopes that the job does not fail whereas, post-active mechanisms handles the job failures after it has occurred [2].

3. Problem Formulation

The main objective of computational grids is to execute the user applications or jobs. Therefore, users submit their jobs to the Grid Scheduler (GS) along with their QoS requirements [1]. These requirements may include the deadline in which users want jobs to be executed, the type of the resources required to execute the job and the type of the platform needed.

The GS of the present scheduling systems allocates each job to the most suitable resource. In case of fault free, results of executing the job are returned to the user after end of the job. If the grid resource failed during execution of the job, the job is rescheduled on another resource which starts executing the job from scratch. This leads to more time consumed for the jobthan expected. Thus, the user's QoS requirements are not satisfied.

To address this problem, the job checkpointing mechanism is used. Using checkpointing, we can restore the partially completed job from the last checkpoint saved and then starting a job from scratch is avoided [7]. The main disadvantage of checkpointing mechanism is that it performs identically regardless the stability of the resource. This inappropriate checkpointing can delay the job execution and can increase the grid load. Commonly utilized checkpointing mechanisms use resource fault index to determine checkpoint interval. In the case of resource failure the fault index based rescheduling

algorithm reschedules the job from the failed resource to some other available resource with the least Fault-index value and executes the job from the final save checkpoint. This ensures the job to be executed within the deadline with increased throughput and helps in making the grid atmosphere trust worthy.

In computational grid environment, there are resources that assure QoS requirements but they tend to fail. To address this problem both the computational and service grid environment can be used. The GS of the present scheduling systems select resources according to the response time combined with the resource fault index to execute the job. If the selected resource is failed and it is the only available resource that can execute the job at that time, the job must hang around for that resource to join the system again and become available. This waiting time delays the job execution and reduces the throughput of the grid. To address this problem, the average failure time and mean failure time of the resource is taken into consideration when making scheduling decisions.

4. Checkpoint Approach

The checkpointing is one of the most popular technique to provide fault-tolerance on unreliable systems [4]. It is a record of the snapshot of the entire system state in order to restart the application after the occurrence of some crash. The checkpoint can be stored on temporary as well as stable storage. However, the efficiency of the mechanism is strongly dependent on the length of the checkpointing period. Frequent checkpointing may enhance the overhead, while lazy checkpointing may lead to loss of significant computation. Hence, the decision about the size of the checkpointing interval and the checkpointing technique is a complicated task and should be based upon the knowledge about the application as well as the system. Therefore, various types of checkpointing optimization have been considered by the researchers.

- Full checkpointing or Incremental checkpointing
- Unconditional periodic checkpointing or Optimal (Dynamic) checkpointing
- Synchronous (Coordinated) or asysnchronous (Uncoordinated) checkpointing,
- Kernel checkpointing
- Application or User level checkpointing.

The economy base grid is a user centric, resource management and job scheduling approach [2]. It offers incentive and profits to resource owners as award of contributing their resources. On the other hand, it also provides user flexible environment to maximize their goal within their budget by relaxing QoS like deadline and budget. Fault tolerance in such environment is critical to consider because it effects the profit of both the parties, but it become more important because the possibility of fault in grid environment is much higher than a traditional distributed system due to lack of centralized environment, major execution of long jobs, highly dynamic resource availability, dissimilar geographical distribution of resources, and heterogeneous nature of grid resources.

Kernel-level check-pointing fault tolerance approach is used in this scenario to overcome above mentioned drawbacks. In this approach, checkpointing procedures are included in the kernel, checkpointing is transparent to the user and generally no changes are required to the programs to make them checkpointable [4]. While the system restart after breakdown, the kernel is responsible for managing the recovery operations. The required kernel-level code is provided in the form of dynamically loaded kernel module so that it is easy to use and install. The package is able to checkpoint multi-process programs.

A. Types of Checkpointing

(i) Full or Incremental Checkpoint

A full checkpoint is a traditional checkpoint mechanism which occasionally saves the total state of the application to a local storage. The drawback of this checkpoint is this can be time consumed to taking checkpoint, and also required very large storage to save.

Instead saving the whole process state incremental checkpoint mechanism allows to save the pages which reduce the checkpoint overhead. In the Incremental checkpoint method, the first checkpoint is typically a full checkpoint. After that, only modified pages are checkpointed at some predefined interval. This results in more expensive recovery cost than the recovery cost of the full checkpoint mechanism.

(ii) Uncoordinated or Coordinated Checkpointing

In uncoordinated checkpointing each process takes its checkpoint independently of the other processes, in this the processes may force to rollback upto the execution start. Since there is a chance for losing the whole computation, these protocols are not popular in practice.

Coordinated checkpoint protocols produce consistent checkpoints; hence, the recovery process is simple to implement. The drawback of this approach is these protocols must be consistent with each other.

(iii) Kernel or Low Level Checkpointing

Here checkpointing procedures are included in the kernel, checkpointing is transparent to the user and generally no changes are required to the programs to make them checkpointable. While the system restarts after failure, then the kernel is responsible for managing the recovery operation.

In low-level checkpointing, each checkpointing packages offers different functionality and interface. Because of technological issues the checkpointing packages impose

some limitations on applications that are to be checkpointed. The difficult task to integrate the low level checkpoint packages with the grid.

(iv) User Level Checkponting

In this approach, a user level library is provided to do the checkpointing. To checkpoint, application programs are linked to this library. This approach generally require no change in the application code; however explicit linking is required with user level library, which is also responsible for recovery from failure.

(v) Application Level Checkpointing

Here, the application is responsible for carrying out all the checkpointing functions. Code for checkpointing and recovery from failure is written into the application. It is expensive to implement but provide more control over the checkpointing process.

5. Fault Index Based Rescheduling

The job running on a resource is rescheduled to some other resource in case of resource failure. The Fault Index Based Rescheduling (FIBR) algorithm [5] is explained below:

Step 1: The user submits the job with its deadline, and estimated execution time. After allocating the job to the resource, the resource broker expects a response from the resource and communication latency between resource broker and the resource.

Step 2: If the resource could not get the result of execution within that time interval as specified by the grid manager, it realizes fault has occur, and increment the fault index of that resource by 1, or decrements by 1 on successful completion. This value is updated and stored in the Information Server.

Step 3: When there is a resource failure, the job executed on the failed resource is rescheduled by checking the fault index value of the available resources from the information server. The fault index value suggests the rate of tendency of resource failure. Lesser the fault index value, lesser is the failure rate of the resource.

Step 4: Based on the fault index value the job is rescheduled to some other available resources with least fault index value and executed from the last saved checkpoint. This increases the percentage of job execution.

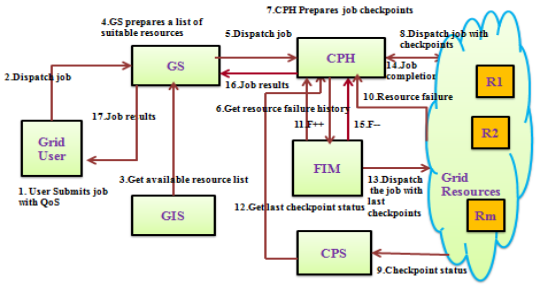

Fig. 2 Fault Tolerance Checkpoint System Architecture

A grid contains multiple grid resources that provide computing services to users. The main component of the Fault tolerance checkpoint system is the Grid Scheduler (GS) fig.2 It receives jobs with their information from users. Job information includes job number, job type, and job size. Also, the user submits QoS requirements of each job such as the deadline to complete its execution, the number of required resources and the type of these resources.

The main function of GS is to find and sort the most suitable resources that can execute the job and satisfy user QoS requirements. In order to perform this task, the GS connects to the Grid Information Server (GIS) to get information of available grid resources that can execute the job [6]. GIS contains information about all available grid resources. It maintains details of the resource such as memory available, load, processor speed and so on. All grid resources that join and depart the grid are monitored by GIS. Whenever a scheduler has jobs to implement, it consults GIS to get information about available grid resources. The GS uses response time, resource failure rate and resource failure time to construct the list of suitable resources that can execute the job.

CheckPoint Server (CPS) receives and stores partially executed results of a job from the resource in intervals specified by the CheckPoint Handler (CPH). These intermediate results are called checkpoint position. For each job, there is only one record of checkpoint status. When CPS receives a new checkpoint status it overwrites the old one. If CPS receives a job completion message from the resource it removes the record of such job. On each checkpoint set by the checkpoint manager, job position is reported to the checkpoint server. Checkpoint server save the job status and return it on demand i.e., during job/resource breakdown. For a particular job, the checkpoint server discards the result of the previous checkpoint when a new value of checkpoint result is received.

CPH is an important component of Fault tolerance checkpoint system. The main functions of CPH are determining the number of checkpoints and determining the checkpoints interval for each job. CPH receives a job with its assigned list of resources from GS. It connects to GIS to get information about the failure history of grid

resources assigned to the job. Based on breakdown rate of the resource, the CPH determines the number of checkpoints and the checkpoint intervals for each job. Then, it submits the job to the first grid resource in the resources list.

Fault Index Manager (FIM) maintains the fault index value of each resource which indicates the failure rate of the resource. The fault index of a grid resource is incremented every time the resource does not complete the assigned job within the deadline and also on resource breakdown. The fault index of a resource is decremented whenever the resource completes the assigned job within the deadline.

6. Performance Evaluation

Grid is a complex environment and the behavior of the resources in the grid is unpredictable. So, it is difficult to build a grid on a real scale to validate and evaluate scheduling and fault tolerant systems. Therefore, simulation is often used. There is a number of well-known grid simulators, such as GridSim [9], SimGrid [10] and NSGrid [11]. However, none of these simulators support the development of fault-tolerant scheduling algorithms. So, in order to carry out this work, we used grid simulator.

The system models of these approaches are designed and tested in GridSim Toolkit. The gridsim libraries are added to the platform of Eclipse, which is an integrated development environment (IDE) for java. The gridsim libraries are available freely as java runtime environment (jre), and they are linked to eclipse platform as external jre. Different numbers of Gridlets are created to evaluate these approaches. Gridlet is define in term of length (in Million Instruction), input file size (in byte), and output file size. Total number of gridlet successfully competed is plotted in fig 3 and fig 4. To measure the performance, gridlets are assigned to grid in which fault tolerance approach is used and to grid in which dynamic adaptive checkpoint fault tolerance approach is used. In dynamic adaptive checkpoint approach we need one separate server in order to avoid this kernel level checkpoint can be used. Thus kernel level checkpoint base approach is better than the adaptive checkpoint approach.

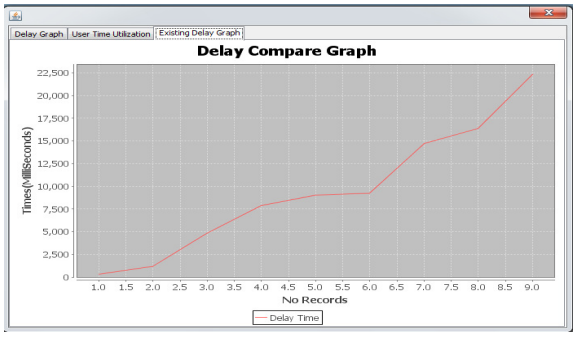

Fig 3: Dynamic adaptive Checkpoint

In dynamic adaptive checkpoint the performance can be seems to be contain with number of variations. Due to these variations the proposed kernel-level checkpoint can be used. The result is shown in fig 4.

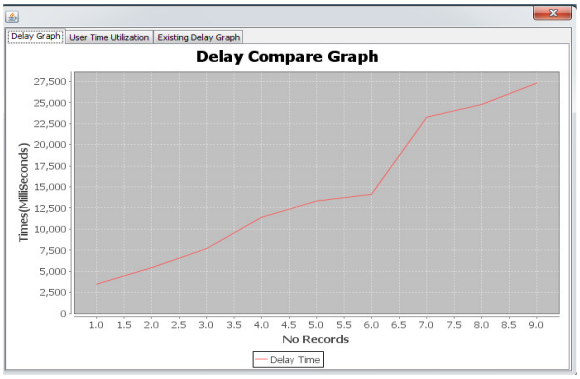

Fig 4: Kernel-level checkpoint

7. Conclusion and Future Work

Fault tolerance techniques are most important for grid systems**.** A proper comparison should be carried out to analyze the performance of different checkpoint approaches. In the proposed work A Fault Index Based Rescheduling (FIBR) algorithm is used to compare dynamic checkpoint and kernel-level based checkpoint. The Fault Index Based Rescheduling (FIBR) algorithm is used to reschedules the job to some other available resource. It increments the fault index value when the failure is detected and decrement after the completion of the job. This will ensures that the job is executed within minimized execution time. The proposed kernel-level checkpoint is used to minimize execution time. Thus the system proposes a new scheme that analyzes the failure ratio in computational grid as well as service grid. The future work includes the kernel-level checkpoint applicable for various scheduling algorithm.

References

[1] Amoon " A Fault Tolerant Scheduling System Based on Check pointing for Computational Grids," International Journal of AdvancedScience and Technology,Vol. 48, November, 2012.

[2] Pankaj gupta "Grid computing and checkpoint approach," IJCSMS International Journal of Computer Science & Management Studies, VOL. 11, Issue 01, May 2011 ISSN (Online): 2231– 5268.

[3] Malarvizhi Nandagopal and Rhymend Uthariraj.V. "Fault Tolerant Scheduling Strategy for Computational Gri Environment," International Journal of Engineering Science and Technology, VOL. 2, 2010.

[4] Ritu Garg and Awadhesh Kumar Singh "Fault Tolerance in Grid Computing: State of the Art and open issues," International journal of Computer Science & Engineering Survey, VOL 2, No 1, February 2011.

[5] Antony Lidya Therasa.S, Antony Dalya.S and Sumathi.G "DynamicAdaptation of Checkpoints and Rescheduling

in Grid computing," International Journal of Computer Application, VOL.3, May 2010.

[6] Fangpeng Dong and Selim G. Akl "Scheduling Algorithms for Grid Computing: State of the Art and Open Problems," Technical Report No.2006-504.

[7] Maria Chtepen, Filip H.A. Claeys and Bart Dhoedt "Adaptive Task Checkpointing and Replication: Toward Efficient Fault-Tolerant Grids," IEEE Transaction on Parallel and Distributed Systems, VOL. 20, NO.2, February 2009

[8] P. Latchoumy,P. Sheik Abdul Khader "Survey on Fault tolerance in Grid Computing," International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.4, November 2011

[9] Rajkumar Buyya, Manzur Murshed "GridSim: a toolkit for the modeling and simulation of distributed resource management and Scheduling for Grid Computing," concurrency and computation: practice and experience Concurrency Computat.: Pract. Exper. 2002; 14:1175–1220 (DOI: 10.1002/cpe.710).

[10] A. Legrand, L. Marchal and H. Casanova, "Scheduling Distributed Applications: The SimGrid Simulation Framework", Proc. Third Int'l Symp. Cluster Computing and the Grid (CCGrid '03), (2003), pp. 138-145

[11] P. Thysebaert, B. Volckaert, F. De Turck, B. Dhoedt and P. Demeester, "Evaluation of Grid Scheduling Strategies through NSGrid: A Network-Aware Grid Simulator", J. Neural, Parallel and Scientific Computations, special issue on grid computing, vol. 12, no. 3, (2004), pp. 353-378.