International Journal For Technological Research In Engineering
Volume 3, Issue 7, March-2016 *ISSN (Online): 2347 - 4718*

OVERVIEW ON FAIR SCHEDULING AND OPTIMAL FAULT TOLERANCE APPROACHES TO INCREASE THE PERFORMANCE OF GRID ENVIRONMENT

P.B.Niranjane[1], D.G. Thakare[2]
[1]Asst.Prof. Department of CSE, Babasaheb Naik Collge of Engineering, Pusad, (MS)
[2]II Year ME Student, Department of CSE, Babasaheb Naik Collge of Engineering, Pusad, (MS)

ABSTRACT: Grid computing is becoming an important technology in distributed computing. The Fact is focused on grid computing is Load balancing, Fault tolerance and recovery from Fault. This paper study the Load balancing and Fault tolerance. Load balancing is the technique by which it maintain the workload of sites . This article designs a Grid Scheduler, which selects the Minimum Loaded Site for the candidate set of nearest sites, Execute the job within the Grid. Then the job is dispatched to the Fault Detector based on the availability of the site. The load balancing task in the grid environment will significantly improve the performance of the grid environment. Fault tolerance is a main technique in grid environment. This technique will used to execute the job from the processor failure. To achieve high throughput and resource utilization we propose a Fair scheduling algorithm and an optimal fault tolerences that trails for the proposed system are conducted using Grid Simulation Toolkit (GridSim)

Keywords: Distributed computing; Fault tolerance; Grid computing; Optimal Fault Tolerences GridSim; Load balancing; Scheduling

I. INTRODUCTION

An important characteristic of grid computing is resource sharing. The resources are shared among various applications. In this scenario load balancing plays an important role. Grid can offer an efficient load balancing effect by scheduling the incoming jobs. But the scheduling process will slow down the system performance and also cause the system overloaded. Load balancing is an important technique to maintain the workload in the sites. So there is a need for a fair scheduling approach. Scheduling is an important process to optimize the load in the system. The grid environment needs a proper scheduling and load balancing algorithms to increase the overall performance of the system Each site in the grid environment will provide its hardware information; time and resources available in each site are recorded for decision making purpose. The grid scheduler is a manager it will manage the state of sites to execute jobs. As arriving jobs are placed in the job queue, the load of the system is increased with increase in the queue length. While the load in the system is balanced, the grid scheduler will receive the job from the job queue and perform the scheduling process. The job queue length is used as the load indicator. The important concept in fault model is "Resource reclaiming" which is invoked when the primary site finished the job before the estimated time. The backup slot are removed and assigned for the new job which avoids the backup overloading. There are two types of load balancing policy in grid environment: Static load balancing policy and Dynamic load balancing policy. The static load balancing policy is not well suited to grid environment because the load may vary with respect to time. Based on the load at the time it allocates the job to the nodes. Here the work stations are not constantly monitored. But in dynamic load balancing policy the workstations are constantly monitored. The selection of the policy is done at run time and also uses the current load information for decision making. It become unavailable without any advance notifications. In software reliability engineering [2], there are four main approaches to increase system reliability, which are fault prevention, fault removal [1], fault tolerance, and fault forecasting [3]. Since source-codes and internal designs of Web services are unavailable to service users (usually developers of the SOA systems), it is difficult to use fault prevention and fault removal techniques to build fault-free service-oriented systems. Another approach for building reliable systems, software fault tolerance [4], makes the system more robust by masking faults instead of removing faults. One approach of software fault tolerance, also known as design diversity, is to employ functionally equivalent yet independently designed components to tolerate faults .Due to the cost of developing redundant components, design diversity is usually only employed for critical systems. In the area of service computing [5], however, it is possible to construct a fault-tolerant service-oriented system without having to pay the cost of developing diverse components. There are a number of functionally equivalent Web services already diversely implemented by different organizations on the Internet. These Web services can be employed as alternative components for building diversity-based fault-tolerant service-oriented system. Fault tolerance strategies can be divided into passive replication strategies and active replication strategies. Passive strategies [1], [2], [3] employ a primary service to process the request and invoke another alternative backup service when the primary service fails, while Active strategies [3],[4], [5], [6], [7] invoke all functionally equivalent services in parallel. In this paper, user requirements are formulated as local constraints and global constraints. A service-oriented system typically includes a set of tasks. Suitable Web services need to be selected to fulfil these tasks. This paper advances the current state-of-the-art in software fault tolerance for Web services by proposing a systematic and extensible framework for

selecting an optimal fault tolerance strategy for reliable service-oriented systems with local and global constraints. The main contributions of this paper include: (1) modelling the problem of selecting the optimal fault tolerance strategy as a specific optimization problem and designing a heuristic algorithm to efficiently solve the problem; (2) specifying the user requirements as local and global constraints, There are two types of load balancing policy in grid environment: Static load balancing policy and Dynamic load balancing policy. The static load balancing policy is not well suited to grid environment because the load may vary with respect to time. Based on the load at the time it allocates the job to the nodes. Here the work stations are not constantly monitored. But in dynamic load balancing policy the workstations are constantly monitored. The selection of the policy is done at run time and also uses the current load information for decision making. Dynamic Load balancing policy will not require the priori task information to allocate/reallocate the resource. Dynamic load balancing algorithm will give better performance then static load balancing algorithm.

II. RELATED WORK

In grid computing there are various job scheduling algorithm are used to utilize the resources effectively. This process will increase the execution performance and balance the system load. The resources which change over time are contributed by idle computers in the grid computing environment. This will happen when new resource join or old resource exit from the grid environment because grid is an dynamic environment the resources in the environment will change over time so designing a job scheduling algorithm to dynamically change according to the variation in resource requires considering numerous and complicated factor[1]

In grid the load balancing algorithm will follow three policies there are Information policy, transfer policy, location policy and selection policy. The information policy specifies what workload information to be collected, when it to be collected and from where it to be collected. The resource monitoring system will monitor the status of the resource based on the resource load information the transfer policy will decide whether the resource have the eligibility to act as a sender or receiver. Sender means it will transfer the job to the resource. Receiver means it will receive the job from another resource [2]. The load balancing mechanism will follow the fair distribution. It will distribute the load across the nodes in fairly manner. We mean that the difference between "heaviest loaded" node and "lightest loaded" node should be minimized [3]. In Oder to minimize the overhead of information collection, state information exchange is done by mutual information feedback. An advantage of mutual information feedback is that the rate of load dissemination is directly proportional to the job arrival rate. An increase in job arrival rate means load information is exchanged more frequently [4]. In cluster based load balancing algorithms [6], the computing nodes are partitioned into clusters on the basis of network transfer delay. In this proposed algorithm, each cluster and all computing nodes of its cluster are defined for making load balancing decisions thereby introducing considerable communication overhead In [5], a decentralized dynamic load balancing algorithm (ELISA) is proposed which neglected the overheads involved in collecting state information for load balancing. In this approach, the problem of frequent exchange of information is alleviated by estimating the load, based on system state information received Jobs will fail when the site where they are located fails due to hardware faults. The faults can be transient or permanent and are assumed to be independent. For each job, the backup is scheduled after its primary. There exists a fault detection mechanism such as fail-signal and acceptance test to detect processor and job failures [6].Resource reclaiming is invoked when the primary finishes the job before the estimated time; the backup slot is removed and assigned for the new job. The resource reclaiming will avoid the backup overloading

III. PRO POSED APPROACH

1] FAIR SCHEDULING APPROACH
2]LOAD BALANCING APPROACH
A. Static Load Balancing
B. Dynamic Load Balancing
C. Fault tolerant

A. FAIR SCHEDULING ALGORITHMS:

i) Component Ranking for ordinary application:

Initialize by randomly assigning a numerical value between 0 and 1 to each component in the component graph. Compute the significance value for each component. The significance values can be calculated either iteratively or algebraically. The iterative method is repeating the computation until all significance values become stable.

ii) Component Ranking for Hybrid application:

The components of a hybrid application are divided into two sets by their nature. One set for the components deployed in a private data centre, denoted as P, and the other for the components moved to the cloud, denoted as C. For each component calculate the significance value. The significance values can be calculated either iteratively or algebraically. The iterative method is repeating the computation until all significance values become stable.

B. Fault tolerance strategy selection algorithm:

First, the aggregated failure rate f, response-time t, and the resource cost r of each fault tolerance strategy candidate are calculated by using RB, NVP, Parallel and VM restart. And the strategies which could not satisfy the response-time constraints will be removed. Second, list the Top-K significant components according to the descending order of their significance value. Third, the strategy with minimum resource cost will be selected for each of the components as their initialization strategy to make sure all of them are fault-tolerant. Then for each component, select the candidate with the lowest aggregated failure rate as the optimal one. By repeating the last step until it meets the user resource cost constraints, the reliability-based design optimization can be achieved

IV. SYSTEM DESIGN PROCSS

Reliability optimization framework i.e. OPTIMAL GRID, includes three phases:

1] Fair scheduling analysis;

2]Optimal Fault Tolerances significance ; and

3] Fault tolerance strategy selection.

Legacy application analysis:

Both structure and failure information are extracted during the legacy application analysis phase.

The structure information extraction consists of two sub processes: -

Component extraction:

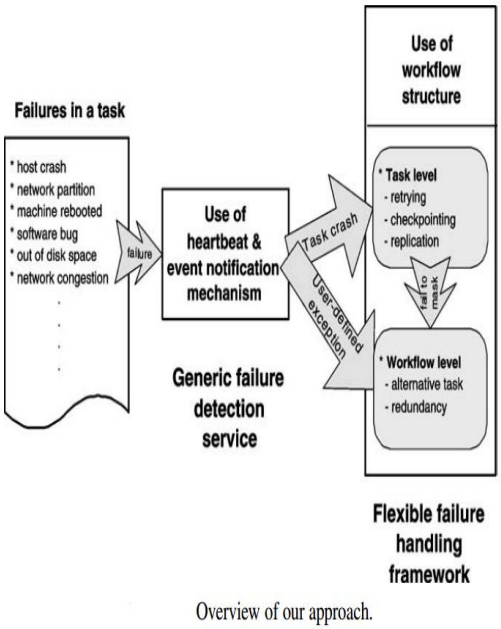
The structure information includes components and the invocation information. The components are extracted from legacy applications by source code and documentation analysis. The invocation information such as invocation links and invocation frequencies can be identified from application trace logs. Source codes and documentations are useful supplementary materials in addition to trace logs. All the information are represented in a component graph.

Invocation extraction:

Component failure rate and failure impact collection: The failure rate and failure impact information can be collected from the execution logs or test results of legacy applications. The failure information including failure rate and failure impact are collected from the execution logs and test results of the legacy application. Components with a failure rate higher than the threshold will be re -factored, and their reliability properties will be updated. A component graph is built for the legacy application based on the structure as well as the failure information.

Automated significance ranking:

In the automated significance ranking phase, two algorithms are proposed for ordinary applications that can be migrated to public cloud and hybrid applications that need to be migrated to hybrid cloud, respectively.



Overview of our approach.

Software fault tolerance strategies:

Recovery Block (RB), Optimal and Parallel are three widely used strategies in software fault tolerance. Since RB strategy invokes standby components sequentially when the primary component fails, its response time is the summation of the execution time of all failed versions and the first successful one. NVP strategy needs to wait for all n responses from the parallel invocations to determine the final result, thus its response time depends on the slowest version. While Parallel strategy employs the first returned response as the final result, its response time is the minimum one of all replications. So it can be concluded that the response time performance of Optimal is generally worse than that of Optimal , which in turn is worse than that of the parallel strategy. Since Optimal and Parallel use parallel component invocations and all the resources need to be allocated before the execution, while in RB extra resources will be allocated only when the primary component fails, the required resources of Optimal and Parallel are much higher than those of Optimal . All three strategies can tolerate crash faults, and Optimal strategy can also mask value faults

The virtual machine restart strategy will not affect resource allocation but can affect the response time if there is a failure. Employing a suitable fault tolerance strategy for the significant components can help achieve optimal resource allocation while improving application reliability. Each fault tolerance strategy has a number of variations, thus selecting an optimal strategy for each significant component is time consuming. An automatic optimal fault tolerance strategy selection algorithm is therefore required to reduce the workload of application designers. Four candidates are employed for fault tolerance which include recovery block, N-version programming, parallel, and virtual machine restart. These strategies can be employed to tolerate crash and value faults. Other types of fault tolerance mechanisms can be added to Optimal Grid without fundamental changes.

V. CONCLUSION AND FUTURE WORK

Authors presents a reliability-based design optimization framework for migrating legacy applications to the Grid environment. They proposes a component ranking framework for fault-tolerant Grid applications component ranking algorithms, the significance value of a component is determined by the number of components that invoke this component, the significance values of these components, how often the current component is invoked by other components, and the component characteristics. After finding out the significant components, System proposes an optimal fault-tolerance strategy selection algorithm to provide optimal fault-tolerance strategies to the significant components automatically, based on the user constraints. The Throughput and Performance of the grid environment will greatly improve by an optimal load balancing approach. Here a fair scheduling approach with equal opportunity to all the jobs is designed. The fair scheduling approach follows the hybrid scheduling by calculating the residue value for each job for a number of iterations until the residue gets down to zero. This approach is linear and iterative in nature

which eliminates the fluctuations in the response time so optimally it may possible to improve performance of Grid environment.

REFERENCES

[1] Qin X, Jiang H, "A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems", Journal of Parallel Computing, 2006;32:331–46.
[2] Grosu D, Chronopoulos , "Non-cooperative load balancing in distributed systems", Journal of Parallel and Distributed Computing 2005;65(9):1022–34.
[3] Penmatsa S, Chronopoulos , "Job allocation schemes in computational Grids based on cost optimization" In: Proceedings of 19th IEEE international parallel and distributed processing symposium, Denver; 2005.
[4] Alomari R, Somani AK, Manimaran G, "Efficient overloading techniques for primary-backup scheduling in realtime systems", Journal of Parallel and Distributed Computing, 2004;64:629–48.
[5] Hwang S, Kesselman C, "A flexible framework for fault tolerance in the Grid", Journal of Grid Computing, 2003;1:251–72.
[6] Buyya R, Murshed MM, "Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing", Concurr Comput: Pract Exp 2002;14:1175–220

