International Journal of Management, IT & Engineering
Vol. 9 Issue 6, June 2019,
ISSN: 2249-0558
Impact Factor: 7.119Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

A STUDY ON HYBRID POLICY FOR FAULT TOLERANT LOAD BALANCING WITH A REFERENCE TO GRID COMPUTING

KOMAL SINGLA
Research Scholar, Department of Computer Science
Sri Satya Sai University of Technology & Medical Sciences, Sehore (MP)

Guide Name:
Dr. Jitendra Sheethlani
Sri Satya Sai University of Technology & Medical Sciences, Sehore (MP)

ABSTRACT

When it comes to tackling complex, large-scale scientific and engineering problems, grid computing has emerged as the next-generation distributed computing paradigm. A huge number of enterprises, each with a local administration and scheduling body, are sharing these resources across Wide Area Networks (WANs), which connect them all (WAN). A heterogeneous network connects large-scale Grid systems, where bandwidth across resources varies between links.

Inter-node communications and load exchange in many of today's distributed computing environments (DCEs) are impeded by latency and bandwidth limitations inherent in the communication medium. While the DCEs are predictable in terms of processor capabilities, the Grid environment is unpredictable in terms of processor capabilities and computer connectivity. The current paper highlights the hybrid policy for fault tolerant load balancing.

KEYWORDS:

Fault, Tolerant, Load, Balancing

International Journal of Management, IT & Engineering
Vol. 9 Issue 6, June 2019,
ISSN: 2249-0558
Impact Factor: 7.119Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

INTRODUCTION

Computer and data grids are two types of grid systems. While in data grids, the primary resource being managed by the resource management system is data spread across geographical locations, in compute grids, the primary resource being managed is compute cycles (i.e. processors). The type of grid system in which the resource management system is installed affects the system's design and functions.

The grid system's resource management system is its most important component. Its primary functions are to take requests from users, match those demands to available resources, and schedule those resources. Resource management and scheduling are critical grid services, and many grids struggle with challenges of job allocation and load balancing, which are typical in most grids. In a computational grid, the goal is to efficiently assign user-defined workloads while meeting deadlines and maximizing the utilization of all available resources at any given moment. Scientists can employ various job-scheduling methods to make efficient use of computing resources by using them in diverse scientific disciplines. This improves system load balancing and enhances execution performance. But even in a grid computing environment, inactive machines are still contributing resources. Resources that have been donated will change over time. This occurs when resources are added or removed from the grid. Job scheduling algorithms in this dynamic and time-varying environment must take into account a wide range of complex parameters in order to dynamically adjust to changes in resources.

As a result of the more difficult scheduling, the system's performance will be significantly impacted. In the grid context, a suitable work scheduling algorithm is critical. Some nodes may

be overburdened, while others may be underutilized as a result of differential task arrival patterns and computational capacity. This method is designed to distribute the load among all nodes of a network in an effort to maximize their efficiency, throughput, and response time.

Load balancing must be "fair," meaning that the disparity between the "heaviest-loaded" node and the "lightest-loaded" node should be reduced, in order to fulfil these goals. When it comes to the location of the load-balancing choices, algorithms can be classed as centralized, decentralized, or hierarchical. Centralized systems have one resource that serves as the central controller in a distributed system. If the system load is too high, it can select how to distribute work between the available resources.

As the system grows in size, it becomes increasingly difficult to keep track of all of the system's properties, and the central controller ends up serving as a bottleneck and single point of failure. Decentralized load balancing involves all of the distributed system's resources in the decision-making process. Obtaining the dynamic status information of the entire system is expensive since load balancing choices are dispersed. Due to this, it is common for most algorithms to employ just a suboptimal judgement based on the little information available in the local resource. The schedulers are arranged in a hierarchy in a hierarchical paradigm.

Resources at the top of the scheduler hierarchy get priority scheduling, whereas resources at the bottom of the hierarchy get priority scheduling. In this architecture, both centralized and decentralized approaches are used simultaneously.

Static, dynamic and adaptive load balancing algorithms are further divided based on the type of information used to make the judgments. It's common for static algorithms to assume that all the

information needed to complete a task, such as the job's qualities and the resources available, is already known. This method is easy to implement and requires no additional resources to execute. However, there are two big drawbacks to this method. Firstly, it is impossible to estimate the workload distribution of many programs before they are run.

As a second point, it presupposes that all information is constant in a grid setting, which may not be the case. For new jobs, dynamic load-balancing algorithms may make an instantaneous and direct judgement without any prior information, so they don't spend any time. In order to offset the higher costs, however, it is necessary to restrict the amount of overhead associated with collecting and preserving load information. It is a subclass of dynamic algorithms known as adaptive load-balancing algorithms.

As the condition of the system changes, they change their parameters or even their policies, and this allows them to modify their operations accordingly. When a load imbalance is found, the four policies that regulate the algorithm's actions are information, transfer, location, and selection. It is the responsibility of the information policy to retain up-to-date information on the load of each resource in the system.

HYBRID POLICY FOR FAULT TOLERANT LOAD BALANCING WITH A REFERENCE TO GRID COMPUTING

System dynamics are the focus of a transfer policy. In order to determine when a resource is qualified to operate as a sender or receiver, it leverages the resources' load information (retrieve a job from another resource). A company's location policy determines which partners are available

for a job move. The location policy looks for a receiver resource to receive the jobs if the resource is an eligible sender. In order to find an eligible sender, the location policy checks if the recipient is eligible.

To determine which of the queued jobs should be transmitted, a selection policy is invoked if a resource qualifies as a sender or recipient. Polling data is used to identify resources as sender/overloaded, receiver/underloaded or OK in a stable symmetrically launched adaptive algorithm (resources having manageable load). Each resource maintains a data structure that includes a sender list, a receiver list, and an OK list to keep track of the current status of the resources. These lists are maintained using an efficient scheme and list manipulation activities, such as shifting a resource from one list to another or determining which list a resource belongs to.

Regardless of the quantity of resources in the system, these acts place a tiny and persistent burden on the system's performance. It's therefore a good fit for large-scale distributed systems. The jobs in the queue are evaluated based on a variety of selection criteria. Selecting the work that decreases local load, costs the least to transfer, and has a good affinity to its destination is its primary purpose in this process. In grid systems, resource failures (processors/links) are common and can have a negative impact on applications.

Fault tolerance has become increasingly important as a result of this trend. Fault tolerance may be achieved in multiprocessor systems by scheduling copies of work on several processors.

Replication can be accomplished in one of two ways, as discussed in the following sections:

1. Active replication:

There are numerous copies of each work assigned to various processors that are executed in parallel to accept a certain number of failures with this approach. A fault-finding mechanism isn't needed with this method.

2. Passive replication:

To put it simply, if something goes wrong while the primary task is running, the backup job gets launched. When a processor fails, it is assured that all impacted tasks will be recovered. Only two copies of the work are scheduled to run on different processors in this arrangement (space exclusion and time exclusion).

For a grid where fault diagnosis is difficult, this technique is particularly valuable since one can uncover a grid processor failure about which they had no idea it was a hardware platform model that had existed. When scheduling primary and backup copies of a work, there are two methods that may be used. In order to maximize the usage of available processor time, backup overloading involves scheduling backups for several primary jobs in the same time slot, and de-allocating resources designated for backup operations after the associated primary processes complete successfully.

FAIR SCHEDULING APPROACH FOR LOAD BALANCING AND FAULT TOLERANT IN GRID ENVIRONMENT

The IT infrastructure is growing more and more dependent on grid technologies. Using a grid, we are able to share resources such as hardware and software, but this sharing is not the same as

file-sharing. Grid computing is based on the idea that we can't just acquire more resources; instead, we may use those that are already accessible. Grids and clusters are both distributed systems, however there are a few differences between the two types. Because the master node controls the executor nodes in a centralized grid, the system is not a grid. There are no central points of control in the grid.

As a result of this decentralized method, the scalability issues are solved, as well as the capacity to exchange goods and services "all to one" and "one to all." The sharing of resources is a key feature of grid computing. Several programs share the same resources. A significant role is played by load balancing in this circumstance. By timing the incoming operations, grid can provide an efficient load balancing effect.

The scheduling procedure, on the other hand, causes the system to run slowly and to become overloaded. There are several ways to keep the work load balanced at the various sites. In other words, a fair scheduling strategy is required. System load optimization necessitates the use of scheduling. In order to improve the overall system performance, the grid environment requires effective scheduling and load balancing techniques. The grid scheduler performs the scheduling process in grid computing. Before sending the job to the scheduler, it will run a resource monitoring procedure to keep track of all available resources. There is a grid scheduler for each location.

The customer submits their request, and grid schedulers are in charge of assigning each task to the appropriate resource (processors). In a grid system, each site will offer its hardware

information; the time and resources available in each site are recorded for decision-making purposes.

The grid scheduler acts as a manager, keeping tabs on the status of several sites in order to carry out tasks. The strain on the system increases as the number of jobs in the queue grows longer. The grid scheduler will take the work from the queue and schedule it as soon as the system load is balanced. The length of the job queue is used as a measure of demand. If a job fails because of a problem with one or more of the grid's sites, it should be accepted. Each task has a copy of its data backed up at the main site. Backups always work even if the original site does not.

When the primary site completes the task ahead of schedule, the fault model's "Resource reclaiming" idea comes into play. To prevent backups from being overloaded, the new operation is given the old backup slot. Load balancing policies in a grid system may be divided into two categories: Static and Dynamic. A grid environment's dynamic load distribution makes a static load balancing scheme ineffective. Based on the current workload, the system assigns the job to the appropriate nodes. Workstations are not frequently checked at this location. Workstations are regularly monitored under dynamic load balancing policies.

The policy is chosen on the fly, and the present load is taken into account in the decision-making process. This policy does not require prior task information to allocate/reallocate the resource dynamically. Using a dynamic load balancing method instead of a static load balancing approach will result in greater performance.

International Journal of Management, IT & Engineering
Vol. 9 Issue 6, June 2019,
ISSN: 2249-0558
Impact Factor: 7.119Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

FAULT TOLERANCE OPTIMAL NEIGHBOR LOAD BALANCING ALGORITHM FOR GRID ENVIRONMENT

Virtual Organizations (VOs) are dynamic, multi-institutional organizations that share resources and collaborate to solve complex problems. The Grid is a distributed computing system that allows users to do work on either local or remote computers. A good scheduling and effective load balancing throughout the Grid is needed to improve the system's performance because of its diverse resources. The average reaction time to tasks is a common performance statistic. A task's reaction time is the amount of time that elapses between when it is started and when it is finished.

Load balancing frequently aims to reduce the average response time. This is a measure of the amount of work a computer system is doing. Some computers will get overloaded if their workloads are greater than those of others, or if some processors take longer to complete a job than others. The long-term goal of the load balancing is to ensure that all processors receive the same amount of work. Load balancing algorithms, in general, are made up of two main rules. A transfer policy and a location policy are required. The transfer policy determines whether or not the system needs to implement load balancing.

Workload information is used to assess whether a node may send (move a job to another node) or receive (receive a task from another node) (retrieve a task from another node). Using this policy, nodes may transmit and receive workloads to and from other nodes, which helps the system as a whole perform better. There are three basic categories of location-based policies:

sender-initiated, receiver-initiated, and symmetrically initiated policies. Static, dynamic, and adaptive are all types of load-balancing algorithms.

Static algorithms use a preset policy to schedule tasks. The current condition of the system is not taken into account during scheduling. A dynamic algorithm, on the other hand, changes its choice based on the current state of the system. Load scheduling algorithms might be classed as either centralized or decentralized based on another categorization that considers the degree of centralization.

There is just one processor that handles load scheduling in a centralized system. There's no way around it: centralized algorithms are always going to be less trustworthy than decentralized ones. Because of this, as more vital applications move to the grid, ensuring their high availability is critical. This is done through load balancing and fault tolerance.

Failure tolerance is the inability of interacting with other nodes due to hardware or system failure or a very busy environment. The greater the number of grid system components, the greater the risk of grid computing failures than in traditional parallel computing.

Computed-intensive grid applications can take days or even weeks to complete, and the grids' vulnerability to failure, including process failure, node crash, and network breakdowns are often exacerbated by this. The control of faults in computational grids is a significant and challenging issue for grid application developers. A fault tolerance service is especially crucial in computational grids since the failure of resources impacts task execution fatally, along with load balancing, and grid applications require fault tolerance services that identify and fix identified failures.

The quality of service (QoS) of grid services is also frequently required for a desired functioning. For a number of computation-intensive applications, grid computing has recently emerged as an interesting computing paradigm. In computing, the optimum solution is the execution of a job with the utmost efficiency while avoiding any errors.

Check point recovery and job replication are two of the most used methods for ensuring the integrity of distributed systems. When an application is running, it can be restarted from where it left off by saving the state of the application to a secondary storage device. With checkpoint-restart, you can avoid re-creating applications every time a problem arises, improve response time, and increase system efficiency.

CONCLUSION

Fault recovery can be facilitated by reverting an application to a previous checkpoint, while better response time can be achieved by starting applications from checkpoints rather than starting from scratch. In the case of a failure, the program may be rolled back and resumed from its latest checkpoint, thereby limiting the amount of lost work that must be recomputed. Synchronous, asynchronous, and quasisynchronic checkpoint algorithms are all subsets of the broader category of checkpoint algorithms. Each process takes its own checkpoints in asynchronous check pointing.

Choosing checkpoints for distinct processes is determined by their casual dependencies. The set of checkpoints would be consistent if they were all taken at the same time. Because it is difficult to build globally synchronized clocks, processes may take checkpoints at regular intervals.

Before taking checkpoints in a synchronous check pointing procedure, synchronize using a message passing interface.

REFERENCES

1. Shukla, Anju & Kumar, Dr Shishir & Singh, Harikesh. (2019). Fault tolerance based load balancing approach for web resources. Journal of the Chinese Institute of Engineers. 42. 1-10. 10.1080/02533839.2019.1638307.

2. Behera, Itishree & Tripathy, Chita & Sahoo, Satya. (2012). An Efficient Method of Load Balancing With Fault Tolerance for Mobile Grid. www.ijcst.com. 3.

3. Indhumathi, V. & Nasira, G.. (2016). Service oriented architecture for load balancing with fault tolerant in grid computing. 313-317. 10.1109/ICACA.2016.7887972.

4. Nanthiya, D. & Periasamy, Dr. Keerthika. (2013). Load balancing GridSim architecture with fault tolerance. 2013 International Conference on Information Communication and Embedded Systems, ICICES 2013. 425-428. 10.1109/ICICES.2013.6508306.

5. Balasangameshwara, Jasma & Raju, N. (2010). A Fault Tolerance Optimal Neighbor Load Balancing Algorithm for Grid Environment. Proceedings - 2010 International Conference on Computational Intelligence and Communication Networks, CICN 2010. 428 - 433. 10.1109/CICN.2010.136.

6. Legrand, A, Marchal, L & Casanova, H 2013, 'Scheduling distributed applications: The SimGrid Simulation Framework', 3rd International Symposium on Cluster Computing and the Grid, IEEE Computer Society, Los Alamitos, CA, USA, pp. 138-146.

7. He, L, Jarvis, SA, Spooner, DP, Chen, X & Nudd, GR 2014, 'Hybrid Performance-based Workload Management for Multiclusters and Grids', IEE Proceedings - Software, vol. 151, no. 5, pp. 224-231.

8. Yang Gao, Hongqiang Rong & Joshua Zhexue Huang 2015, 'Adaptive Grid Job Scheduling with Genetic Algorithms', Future Generation Computer Systems, vol. 21, Issue 1, pp. 151-161.

9. Stefka Fidanova & Mariya Durchova 2006, 'Ant Algorithm for Grid Scheduling Problem', LSSC 2005, LNCS 3743, 2006._c SpringerVerlag Berlin Heidelberg, pp. 405-412.

10. Siriluck Lorpunmanee, Mohd Noor Sap, Abdul Hanan Abdullah & Chai Chompoo-inwai 2007, 'An Ant Colony Optimization for Dynamic Job Scheduling in Grid Environment', World Academy of Science, Engineering and Technology, vol. 1, no. 5, pp. 1343-1350.