Survey of Fault Tolerance Policy for Load Balancing Scheme in Distributed Computing

Akanksha Chandola Anthwal
Department of Computer Science
Kanya Gurukul Mahavidyalaya, Dehradun

Nipur,Ph.D
Department of Computer Science
Kanya Gurukul Mahavidyalaya, Dehradun

ABSTRACT

Cloud computing had opened a new horizon for utilization of resources and their computing. But as accessibility of cloud increases, one of the most important factors to be considered would be availability of resources and load balancing. Fault tolerance is another issue to deal with while providing Quality of Service in cloud environment thus enhancing the performance. This paper investigates about fault-tolerance in load balancing schemes in distributed environment. There are some more parameters influencing QOS but our main focus is on fault tolerance and load balancing.

Keywords

Cloud computing; fault tolerance; load balancing; Quality of Service (QOS).

1. INTRODUCTION

Cloud computing is Internet-based computing, whereby shared configurable resources (e.g., infrastructure, platform, and software) are provided to computers and other devices as services [1].Cloud computing make the resources available to the customer from common pool of distributed computing resources (storage, network, processing ability and infrastructure), from any location up to any duration. In this distributed computing environment some of important issue associated are security, dynamic load balancing or task scheduling, occurrence of failure, violation of Service Level Agreement (SLA) which are obstacle to provide better QOS values. Our main concern is Load Balancing along with Fault Tolerance for cloud.

In Cloud Computing the main concerns involve efficiently assigning tasks to the Cloud nodes such that the effort and request processing is done as efficiently as possible [2], while being able to tolerate the various affecting constraints such as heterogeneity and high communication delays. With the popularity of Cloud computing requirement of large and powerful data centers had occurred. Load balancing is another issue which should be taken in account in the virtual environment for efficient job execution and necessity for minimizing job turnaround time in distributed computing systems thus providing with improved QOS value. Till the time many load-balancing approaches were proposed for real-time scenario but, few of them had taken in account the fault-tolerance in load balancing mechanisms. An efficient load balancing mechanism is required for improving the system performance and Fault-tolerant network systems are designed to provide reliable and continuous services in distributed computing despite the failures of some of their components.

In this paper our prime focus is on load balancing and fault tolerance approaches and for the same, Section II; gives an overview of some commonly used Load balancing techniques. In our next section III; metrics and challenges for load balancing and fault tolerance in cloud environment are discussed. Section IV gives some load balancing schemes taken in account fault tolerant policy. In Section V above mention techniques are compare and contrast. At last paper is concluded with future scope of work in this particular area.

2. LOAD BALANCING TECHNIQUES REVIEW

Load balancing techniques are takes in account different types of approaches. Broadly these are classified as: *Static algorithms* which divide the traffic evenly between servers and *Dynamic algorithm* which search for the lightest server in the network and then designated appropriate weights on it. There are numerous load balancing techniques but few of them had been selected for review work.

Some of the commonly known *Static Load Balancing Algorithms* are;

2.1 Honey Bee Foraging

Honey Bee foraging [2] algorithm is derived from the behavior of honey bees for finding and reaping food. In order to check for fluctuation in demand of services, servers are grouped under virtual servers (VS), having its own virtual service queues. Each server processing a request from its queue calculates a profit or reward on basis of CPU utilization, which is corresponds to the quality that the bees show in their *waggle* dance and advertise on the advert board. Each of the servers takes the role of either a *forager* or a *scout*. A server serving a request, calculates its profit and compare it with the colony profit, if profit was high, then the server stays at the current virtual server and on the other hand if profit was low, then the server returns to the forage or scout behavior, thus balancing the load with the server.

2.2 Biased Random Sampling

According to Biased Random Sampling [3] approach the load on a server is represented as a virtual graph having connectivity with each node. Each server is symbolized as a node in the graph, where each in degree represents the free resources of the server. Whenever a node executes a job, an incoming edge is being deleted, thus indicating the reduction in the availability of free resource. After completion of a job, the node adds on an incoming edge, indicating an increase in the availability of free resource. Random sampling is used to increment and decrement processes. The last node in walk is selected for allocation of load; instead any other node based on certain criteria could also be preferred.

A node on receiving a job, will execute it only if its current walk length is equal to or greater than the threshold value. Being lesser to the threshold value, the walk length of the job under consideration is incremented and another neighbor node is selected randomly. Again a new directed graph is formed and load balancing is achieved in fully decentralized manner, thus making it suitable for large network systems like cloud.

2.3 Active Clustering

Active Clustering approach [4] is considered as a self aggregation algorithm, works on the principle of grouping similar nodes and working together on these groups. A set of processes are iteratively executed by each node on the network. Initially any node can become an *initiator* and selects another node from its neighbors to be the *matchmaker* node satisfying the criteria of being a different type than the former one. The *matchmaker* node then forms a connection between neighbors of it which are similar to the initiator. The matchmaker node then removes the connection between itself and the initiator.

In order to work more efficiently in real scenario, *Dynamic Load Balancing Algorithms* are required; some of them are discussed as follows:

2.4 Join Idle Queue

JIQ [5] is a load balancing algorithm applicable for dynamically scalable web services. This technique involves a *dispatcher* to whom processors informs at the time of their idleness, without interfering with job arrivals. Thus removing the load balancing work from the critical path of request processing, system load is reduced; no communication overhead at job arrivals and no increment in actual response time.

In JIQ algorithm load balances idle processors across dispatchers, which is called the *secondary* load balancing problem. For solving *primary* load balancing problem concern with assigning jobs to the processors, first of all the secondary problem of assignment of idle processors to dispatchers is being solved, which in turn takes place in the reverse direction. While the primary problem concerns the reduction of average queue length at each processor, the secondary problem concerns the availability of idle processors at each dispatcher [5].

2.5 Divisible Load/Task Theory (DLT)

DLT [6] is inspired by level-balancing property of liquid. When water is poured into a cup on the horizontal surface, water (independent of amount) always reaches the equilibrium state, which means the water surface settles down to be level finally. Similarly, an unstructured P2P grid is a container and computational jobs are like water and jobs are unevenly distributed in girds, forced to move from overloaded nodes to lightly loaded nodes, distributing load evenly.

In DLT, a load can be arbitrarily partitioned into chunks for a group of processors with no priority relationship among obtained chunks. This approach assumes nodes in the grid to be homogeneous and during load balancing no task can insert the queue nor leave. A node only exchange and collect information with their nearest neighbors within one hop to make grids converge to the load balanced equilibrium state [6].

2.6 Load Balance Min-Min (LBMM)

LBMM scheduling algorithm [8] and new optimized *Load Balancing Max-Min-Max (LB3M)* [7] had main objective to minimize execution time of each task, also avoid unnecessary replication of task on the node thereby minimizing overall completion time. Opportunistic Load Balancing algorithm when combined with LBMM (OLB + LBMM) [8] keeps every node in working state to achieve load balance. Similar to LBMM, LB3M also calculate average completion time for each task for all nodes. Then mark the task with maximum average completion time. Next it dispatches the task of marked node to the unassigned node with minimum completion task, thus balancing the workload evenly among all nodes.

3. WORK LOAD METRIC AND FAULT TOLERANCE TECHNIQUE

While comparing different techniques of load balancing, certain metric had to take into account to check the implementation feasibility of the algorithm for a specific application. Some of metrics for load balancing are:

- **Throughput & Response Time** which are related with processing efficiency of the computing systems. Throughput and response time take in account *total turnaround time* calculate as sum of waiting time, expected execution time, time to input files and time taken for output. Throughput is number of task completed in unit time and response time is amount of time to complete unit task. For a good load balancing algorithm throughput should be high and response time should be low.

- **Overhead Associated** determines the amount of overhead involved while implementing a load-balancing algorithm. It includes inter process communication, network overhead, system monitoring and control, dues associated with movement of tasks. Overheads should be minimized for working efficiency of load balancing technique.

- **Resource Awareness** is the feature of load balancing technique to have responsiveness about the capacity of server (including cpu, memory and I/O) and better utilization of resources. This factor helps in optimal resource allocation which is most important requirement of load balancing techniques.

- **Performance** in heterogeneous environment is also an important factor because of dynamic and diversity of real time applications. In order to deal with fluctuating demand of resources in a distributed environment, heterogeneity feature is a necessity to be implemented in these algorithms.

- **Fault Tolerance** is one of the metric which is consider to be most important since the resource failure affects job execution, throughput, response time and performance of system. Thus a fault tolerance policy is required to detect failures, resolve these failures, thus improving performance metric. Fault Tolerance policies are classified in two categories: *Reactive* and *Proactive*.

  Reactive fault tolerance policies are applicable on occurrence of fault to reduce its effect. Replication, check point/restart, retry, task resubmission, job migration, rescue workflow etc. are common reactive techniques. On the other hand proactive fault tolerance policies are based on fault avoidance, predicting failure points, or removing

suspected/slow components. Preemptive migration, software rejuvenating, self healing are some proactive techniques.

- **Migration** is the metric which is associated with fault tolerance. In that case if one of the nodes of distributed system fails in executing tasks assigned to it, jobs had to shift to some other nodes. The time to migrate the jobs or resources from one node to other is *Job migration* time which should be minimized in order to enhance the performance of the system.

- **Implementation Complexity** as the name suggests is the difficulty for implementation of any algorithm to the distributed system. Load balancing algorithms are preferred to be less complex in terms of implementation and operations. The higher implementation complexity would lead to a more complex process which could cause some negative performance issues.

4. LOAD BALANCING SCHEME INCLUDING FAULT TOLERANCE

4.1 Ant Colony Optimization (ACO)

ACO [9] is an improved version of load balancing mechanism based on Ant Colony and Complex Network Theory (ACCLB) [10] in an open cloud computing federation. Both algorithms make use of ants' pheromone to gather and update information about the cloud thus selecting a specific node in order to assign the task however evenly distributing work among nodes.

The ants in proposed algorithm continuously originate from the Head node and traverse all around the network making forward and backward movement to find the under loaded and overloaded nodes. In ACO two types of pheromones are used *Foraging Pheromone* (FP) used to explore overloaded node by forward movement of ants while *Trailing Pheromone* (TP) used to discover its path back to the under loaded node. In order to limit the number of ants in the network, they would commit suicide once it finds the target node [10].

4.2 ESWLC (Exponential Smooth Forecast based on Weighted Least Connection)

ESWLC [11] is an improved form of Weighted least-connection (WLC) along with its features, it also taken into account time series and trials. However WLC counts the connections of each server and reports the appropriate server based on the multiplication of a server weight and its count of connections, ESWLC algorithm concludes assigning a certain task to a node only after getting to know about the node capabilities. ESWLC builds the decision based on the experience of the node's CPU power, memory, number of connections and the amount of disk space currently being used. ESWLC then predicts which node is to be selected based on exponential smoothing [11].

4.3 Map Reduce

Map Reduce [12] is a programming model consisting of two functions- Map and Reduce and improved Map Reduce Fault Tolerance [13] use passive replication on top of re-execution in the cloud. It involves two operations Map process which processes a block of input producing a sequence of (key, value) pairs, and Reduce that process all values of a given key and emit one or more (key, value) pairs. A Map Reduce algorithm consists of three phases: map phase, shuffle phase and reduce phase. The map function operates on request entity i.e. a series of key/value pairs, processes them and emits output key/value pairs. In shuffle phase, each output key/value pair is saved into a hash table and sorted, grouping all values associated with a particular key. The reduce phase is associated with processes of all values associated with given key and emitting the one or more new key/value pairs.

In Map Reduce Fault Tolerance, the master first attempts to assign a map task (in the queue) whose data is on that machine (*data locality*) provided that the machine is free to process the request. In case of failure in the execution, it attempts to assign a map task whose data is located (on a machine) on the same network switch with that machine (*rack locality*). Therefore, on occurrence of failure complete map tasks need to be re-executed, but completed reduce tasks does not. To ensure that a failed job can be recovered and is being scheduled with a guaranteed time period, a threshold value is used whereby beyond it, the failed job will be scheduled on the next available machine in spite of data locality [13].

4.4 Virtual Machine Mapping (VM Mapping)

VM Mapping [14] is based on multi-dimensional resources to achieve overall load balance. This algorithm adopts the centralized control architecture comprises of scheduling controller and resource monitor as core elements of the system. The scheduling controller is responsible for VM lifecycle management and fulfilling allocation policy while the resource monitor collects the information about resources from physical hosts [14].The processes involved in VM mapping policy goes through following four phases: firstly; accepting the request for virtual machine on FCFS principle, secondly; obtaining resource information which in turn is maintain by resource monitor, thirdly; VM initial placement by scheduling controller, finally; user can remotely access the application on cloud.

In the initial phase weights for each dimension resources are calculated, by taken in account all the resources occupied and VM are compared. Next step is based on light load first, where each node is being scored on bases of above calculated weights, which would be inverse propositional to its utilization. Finally probability scheme is used for selection of node using Roulette Wheel approach. This algorithm also gives the provision of VM migration policy for dynamically adjusting load according to the threshold during the running period. It eases the problem of load crowding to a certain extent and ensures load balance in the virtual environment.

4.5 *Dual Direction FTP (DDFTP)*

DDFTP [15] is a *dual-direction* downloading algorithm from FTP servers and its modified version in [16], introducing efficient fault-tolerance and load balancing with minimal communication and coordination overhead while executing services in parallel over shared and dynamic heterogeneous distributed cloud infrastructure. The main idea of algorithm is to splits the file into two half and task is being executed on two servers, such that each server starts processing the task in an opposite direction from the other, one server starts processing from the beginning in an incremental order while other starts the file downloading from the last block in decrement order. The task is considered to be finished when the two servers download two consecutive blocks meeting at consent. As a result, both servers will work independently, but will end up downloading the whole file to the client in the best

possible time, giving the performance and properties of both servers [16]. Moreover, attributes such as network load, node load, network speed are automatically taken into consideration, while no run-time monitoring of the nodes is required, yet it maintain good load balancing among all participating server. In addition, if one of the servers fails before completion of task, second one continues the task till it reaches to the point where the other gets fail.

4.6 Fault Tolerance Policy on Dynamic Load Balancing (FTDLB)

FTDLB [17] is fault tolerance policy that could tolerate the node's permanent failures while balancing load of real-time applications on P2P grids. For improving the system reliability, FTDLB duplicates jobs into different sites and adaptively adjust the load of real-time applications to achieve the job's minimal turnaround time. FTDLB algorithm works as follows; each site regularly sends "heartbeat" messages to its neighbor site which includes the cpu utilization, memory usage, job status, etc. When receiving of "heartbeat" messages stops, within a fixed period, it indicates failure of neighbor site thus triggering fault tolerance policy.

The proposed policy calculates the turnaround time of jobs in the Request Queue (REQ), and selects the site's Running Queue (RQ) to minimize job turnaround time maintaining load balancing among sites. When the job is dispatched from REQ, job redundant mechanism is triggers to finds the primary and secondary site provided turnaround time would be smallest and then sends the redundant jobs to backup storages of both. When the original site to fails execute the job, the primary site transfers the redundant job from the backup storage to REQ in primary site, and notifies the backup storage in the secondary site to remove the copy. When the original site as well as its primary site is failed, the secondary site transfers the job from the backup storage to REQ for execution [17]. In this policy the initial task of allocation of job to the original site, selection of primary and secondary site achieve load balancing while triggering job redundancy and backup storage add on fault tolerance to the system.

4.7 O-Ring (Overlapped Ring)

O-Ring [18] is a novel architecture that provides fault tolerance and load balancing for distributed and dynamic scenario. O-Ring use the approach of data replication (mirroring) and data distribution in order to provide both fault tolerance and load balancing in well- organized manner. In the initial phase data items are replicated on the neighboring peers on the ring in order to achieve fault tolerance and each peer also stores the address of its predecessor and successor. Every ring had a Directory Service which is responsible for routing of requests like; data retrieval, updates, insertions and deletions on appropriate peers. As copy of data is already being replicated for backup on another peer short-term fluctuations are addressed by moving the boundaries of responsibility between peers without the need to move the data itself. Thus, redistributing the load in forward and backward direction in order to balance the load faster, and minimizing interferes with concurrent query processing. Any types of fluctuations, that require the movement of data, are addressed by moving the backup copies of the data in the background, without disturbing the primary copy of the data that is being used to handle requests for the data. Along with less expensive load balancing of O-Ring also achieves higher throughput, as it can balance the load with lower overheads and can respond rapidly to load imbalances.

5. DICUSSION & COMPARISION

Table I show the comparison of above mentioned Load Balancing schemes on bases of metric mentioned in section III.

Biased Random Sampling and Active Clustering are the algorithm which supports heterogeneity partly means as system diversity increases these algorithm degrade their efficiency, thus inferring that these technique would be suited to lesser diversify and dynamic environment. ACO is the dynamic load balancing technique which do not support heterogeneity but had the provision of fault tolerance technique and is easy to implement to Internet, cloud or grid computing systems. DLT had job migration provision for load balancing only and does not consider the faulty scenarios while FTDLB and VM Mapping shift the jobs to other nodes in order to achieve load balancing and on the occurrence of fault, for improving performance of the system. O-Ring as compare to FTDLB has lesser implementation complexity thus with less overhead it reduces the implementation cost and perform fast. VM Mapping had higher overhead associated than DDFTP as the former had resource monitoring system as core element of the system.

6. CONCLUSION

In this paper, main focus is on fault tolerance policy based load balancing algorithm. In this survey starting with common load balancing techniques in cloud computing and further investigated techniques having fault tolerance provision in load balancing scheme also included some approaches implemented to grid computing as both are type of distributed computing.

By comparing the techniques on different metric and tried to find the scope for improving fault tolerance policy in load balancing schemes. In near future research could be conducted on development of load balancing algorithm for cloud, taking in account fault management and also minimizing migration time of job in case of failure of node occurs and further guaranteeing optimal performance of system. More load balancing algorithm could be developed which take into account proactive technique of fault tolerance in cloud computing for enhancing the efficiency and providing quality of service value with the increase of demand of resources on cloud for vital applications.

Technique/ Metric	Heterogeneity	Resource Awareness	Implementation Complexity	Distributed Environment	Fault Tolerance	Job Migration	Overhead Associated
Honey Bee Foraging	YES	YES	NO	Cloud	NO	NO	NO
Biased random sampling	YES (Partly)	YES	NO	Cloud	NO	NO	NO
Active Clustering	YES (Partly)	NO	NO	Cloud	NO	NO	NO
Join Id Queue	YES	NO	LESS	Cloud	NO	NO	NO
DLT	YES	YES	NO	P2P Grid	NO	NO	YES
ACO	NO	YES	NO	Cloud/P2P Grid	Reactive	NO	YES
ESWLC	YES	YES	HIGH	Cloud	Reactive	YES	YES
Map Reduce	YES	NO	HIGH	Cloud	Reactive	NO	NO
VM Mapping	YES	YES	HIGH	Cloud	Reactive	YES	YES
DDFTP	YES	YES	LESS	Cloud/P2P Grid	Reactive	NO	LESS
FTDLB	YES	YES	HIGH	P2P Grid	Reactive	YES	YES
O Ring	YES	NO	LESS	P2P Grid	Reactive	NO	NO

Table I: Comparison of Load balancing Techniques

7. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," *Technical Report, EECS-2009-28*, University of California, Berkeley, 2009.

[2] Randles, M., D. Lamb and A. Taleb-Bendiab, "A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing," in Proc. IEEE 24th International Conference on Advanced Information Networking and Applications Workshops (WAINA), Perth, Australia, April 2010.

[3] O. Abu- Rahmeh, P. Johnson and A. Taleb-Bendiab, "A Dynamic Biased Random Sampling Scheme for Scalable and Reliable Grid Networks", INFOCOMP - Journal of Computer Science, ISSN 1807-4545, 2008, VOL.7, N.4, December, 2008, pp. 01-10.

[4] F. Saffre, R. Tateson, J. Halloy, M. Shackleton and J.L. Deneubourg, "Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications." The Computer Journal, March 31st, 2008.

[5] Yi Lua, Qiaomin Xiea, Gabriel Kliotb, Alan Gellerb, James R. Larusb, Albert Greenbergc, " Join-Idle-Queue: A Novel Load Balancing Algorithm for Dynamically Scalable Web Services" Volume 68 Issue 11, November, 2011, pp:1056-1071, Elsevier Science Publishers, 2011.

[6] V. M. B. Veeravalli, D. Ghose and T. Robertazzi, "Scheduling Divisible Loads in Parallel and Distributed Systems," *IEEE CS Press*, 1996.

[7] Che-Lun Hung, Hsiao-hsi Wang and Yu-Chen Hu "Efficient Load Balancing Algorithm for Cloud Computing Network", International Conference on Information Science and Technology (IST 2012), April 28-30, pp; 251-253.

[8] S. Wang, K. Yan, W. Liao, and S. Wang, "Towards a Load Balancing in a Three-level Cloud Computing Network", Proceedings of the 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT), Chengdu, China, September 2010, pages 108-113.

[9] Nishant, K. P. Sharma, V. Krishna, C. Gupta, KP. Singh, N. Nitin and R. Rastogi, "Load Balancing of Nodes in Cloud Using Ant Colony Optimization." In proc. 14th International Conference on Computer Modelling and Simulation (UKSim), IEEE, pp: 3-8, March 2012.

[10] Zhang, Z. and X. Zhang, "A load balancing mechanism based on Ant Colony and Complex Network Theory in Open Cloud Computing federation." In proc. 2nd International Conference on. Industrial Mechatronics and Automation (ICIMA), IEEE, Vol. 2, pp:240-243, May 2010.

[11] Ren, X., R. Lin and H. Zou, "A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast" in proc. International Conference on. Cloud Computing and Intelligent Systems (CCIS), IEEE, pp: 220-224, September 2011.

[12] J. Dean and S.Ghemawat, "MapReduce: simplified data processing on large clusters", Communications of the ACM, 107-113 (2008).

[13] Qin Zheng, "Improving MapReduce Fault Tolerance in the Cloud", Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium, April 2010.

[14] Ni, J., Y. Huang, Z. Luan, J. Zhang and D. Qian, "Virtual machine mapping policy based on load balancing in private cloud environment," in proc. International Conference on Cloud and Service Computing (CSC), IEEE, pp: 292-295, December 2011.

[15] Al-Jaroodi, J. and N. Mohamed. "DDFTP: Dual-Direction FTP," in proc. 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), IEEE, pp: 504-503, May 2011.

[16] Jameela Al-Jaroodi, Nader Mohamed, and Klaithem Al Nuaimi, "An Efficient Fault-Tolerant Algorithm for Distributed Cloud Services," in proc. 2012 IEEE Second Symposium on Network Cloud Computing and Applications, pp:1-8.

[17] Tian-Liang Huang, Tian-An Hsieh, Kuan-Chou Lai, Kuan-Ching Li, Ching-Hsien Hsu, and Hsi-Ya Chang, "Fault Tolerance Policy on Dynamic Load Balancing in P2P Grids", in proc. International Joint Conference of IEEE TrustCom-11/IEEE ICESS-11/FCST-11, 2011, pp:1413-1419.

[18] P. M. Melliar-Smith and Louise E. Moser, "O-Ring: A Fault Tolerance and Load Balancing Architecture for Peer-to-Peer Systems", Proceedings of International Conference of the Chilean Computer Science Society 2009, IEEE Computer Society, pp:25-33.