Int. J. Communications, Network and System Sciences, 2010, 3, 645-654
doi:10.4236/ijcns.2010.38087 Published Online August 2010 (http://www. SciRP.org/journal/ijcns)

A Load Balancing Policy for Distributed Web Service

Safiriyu Eludiora[1], Olatunde Abiona[2], Ganiyu Aderounmu[1], Ayodeji Oluwatope[1], Clement Onime[3], Lawrence Kehinde[4]
[1]*Department of Computer Science and Engineering*, *Obafemi Awolowo University*, *Ile-Ife*, *Nigeria*
[2]*Department of Computer Information Systems*, *Indiana University Northwest*, *Garry*, *USA*
[3]*Information and Communication Technology Section*, *Abdus Salam International Centre for Theoretical Physics*, *Trieste*, *Italy*
[4]*Department of Engineering Technologies*, *Texas Southern University*, *Houston*, *USA*
E-mail: {*sieludiora*, *aoluwato*}*@oauife.edu.ng*, *oabiona@iun.edu*, *gaderounmu@yahoo.com*, *onime@ictp.it*, *kehindelo@tsu.edu*
Received May 11, 2010; *revised June* 22, 2010; *accepted July* 28, 2010

Abstract

The proliferation of web services; and users appeal for high scalability, availability and reliability of web servers to provide rapid response and high throughput for the Clients' requests occurring at anytime. Distributed Web Servers (DWSs) provide an effective solution for improving the quality of web services. This paper addresses un-regulated jobs/tasks migration among the servers. Considering distributed web services with several servers running, a lot of bandwidth is wasted due to unnecessary job migration. Having considered bandwidth optimization, it is important to develop a policy that will address the bandwidth consumption while loads/tasks are being transferred among the servers. The goal of this work is to regulate this movement to minimize bandwidth consumption. From literatures, little or no attention was given to this problem, making it difficult to implement some of these policies/schemes in bandwidth scarce environment. Our policy "Cooperative Adaptive Symmetrical Initiated Dynamic/Diffusion (CASID)" was developed using Java Development Environment (JADE) a middle ware service oriented environment which is agent-based. The software was used to simulate events (jobs distribution) on the servers. With no job transfer allowed when all servers are busy, any over loaded server process jobs internally to completion. We achieved this by having two different agents; static cognitive agents and dynamic cognitive agents. The results were compared with the existing schemes. CASID policy outperforms PLB scheme in terms of response time and system throughput.

Keywords: Web Service, Load Balancing, Distributed Web Service, Job Migration, Quality of Web Service

1. Introduction

Recently, Web Services are widely used daily by more than six hundred million users over the Internet. The proliferation of web services; and users appeal for high scalability, availability and reliability of web servers to provide rapid response and high throughput for the Clients' requests occurring at anytime. Distributed Web Servers (DWSs) provide an effective solution for improving the quality of web services. A collection of web servers is used as a pool of replicated resources to provide concurrent services to the clients. The incoming requests can be distributed to servers according to specific load distribution strategies and thus the requests are processed quickly.

The DWSs can be organized in different ways, they can be integrated into a cluster of web servers linked via Local Area Networks (LANs) to act as a powerful web server, and they can also be deployed at different sites over open network (Internet).

The performance of a distributed system is enhanced by distributing the workload among the servers. Normally, load balancing at the server side assists to balance the load in distributed web servers. According to [1] the most excellent mechanism for achieving optimal response time is to distribute the workload equally among the servers. Traditional load balancing approaches on distributed web servers are implemented based on message passing paradigm. Presently, mobile agents' technology is used to implement load balancing on distributed web servers [2].

A mobile agent is defined as a software component that can move freely from one host to another on a network, transport its state and code from home host to other host and execute various operations on the site [2]. The mobile agent based approaches have high flexibility, low network traffic and high asynchrony. Load balancing (LB) is an efficient strategy to improve throughput or speed execution of the set of jobs while maintaining high processor utilization.

Basically, LB is the allocation of the workload among a set of cooperating servers. The demand for high performance computing continues to increase every day. LB strategies fall broadly into either one of two classes; static or dynamic.

Static load distribution, also known as deterministic scheduling, assigns a given job to a fixed server or node. Every time the system is restarted, the same binding task processor (allocation of a task to the same processor) is used without considering changes that may occur during the system's lifetime.

Moreover, static load distribution may also characterize the strategy used at runtime; in the sense that it may not result in the same task-processor assignment, but assigns the newly arrived jobs in a sequential or fixed fashion. For example, using a simple static strategy; jobs can be assigned to servers in a round-robin manner, so that each processor executes approximately the same number of tasks. However, dynamic load-balancing takes into account that the system parameters may not be known beforehand and using a fixed or static scheme will finally produce poor results. A dynamic strategy is executed several times and may reassign a previously scheduled job to a new server based on the current dynamics of the system environment. The paper is organized as follows, Section 1 introduced the readers to the issue web services' challenges in a distributed systems. Section 2 briefly discusses the intelligent agents and its potentials. The related works was discussed in Section 3. The proposed work was detailed in Section 4. The software model was explained in Section 5. Theoretical framework was discussed in Section 6. Section 7 narrates experimental results and Section 8 draws the conclusion and proposed future work.

2. Intelligent Agents

Intelligent agents exhibit the following characteristics: autonomy, social ability, reactivity, pro-activeness, mobility, learning, and beliefs. An intelligent agent is an independent entity capable of performing complex actions and resolving management problems on its own. Unlike code mobility, an intelligent agent does not need to be given task instructions to function, rather just the high-level objectives. The use of intelligent agents completely negate the need for dedicated manager entities, as intelligent agents can perform the network management tasks in a distributed and coordinated fashion, via inter agent communications. Many researchers believe intelligent agents are the future of network management, since there are quite some significant advantages in using intelligent agents for network management.

3. Related Works

An attempt to distribute workload among available processors to improve throughput or execution times of parallel computers in cluster computing environments was discussed in [3]. This work proposed an approach called DDLB (Distributed Dynamic Load Balancing) policy, this makes an attempt to balance load by splitting processes into separate jobs and then balance them to nodes. Mobile agent framework was proposed for DDLB policy and was implemented on PMADE. The performance evaluation shows that the multi-agent based approach outperforms the message-passing paradigm with large number of clients' requests in a heterogeneous cluster.

DDLB policy can only address issues on local LB there was little provision for global load balancing. Therefore, may not be able to address a very large heterogeneous distributed environment. They need to compare the performance of the DDLB with some other recent LB policies apart from MPI. Since this is a mobile agent based approach, it could be compared with some other mobile agent based approaches. Various LB policies were reviewed by [4], specifically MPI (message-passing interface). Its wide availability and portability makes it an attractive choice. However, the communication requirements are sometimes inefficient when implementing the primitives provided by MPI.

Other works presented a dynamic LB framework called Platform for LB (PLB). It uses MAs to implement reliable and scalable LB on distributed web servers. PLB is implemented on PMADE (A Platform for Mobile Agent Distributed and Execution). The performance evaluation shows that PLB based approach outperforms MPI when large number of servers and client requests are involved.

The PLB SSP adopted two strategies: Best-fit and First-fit. The best-fit will create two different speeds; the faster and the slower, this will affect the response time of the servers. The first-fit equally has some latency over a period of time. Overall, the two strategies adopted may not perform well when compared to other policies. This policy may not be efficient when compared with our work; we approach the load balancing by considering the heterogeneity of all nodes. We use broadcast approach to determine the node(s) that are available and ready for incoming jobs.

Load balancing on WAN is more time consuming [5], since it involves the iteration between remote servers for gathering load information, negotiating on load relocation and transporting the work-load. Also, moving a large amount of request requires high bandwidth and it is difficult if a node's load changes quickly in relation to the time needed to move requests. The authors proposed an approach that will address these problems. LDMA (Load Balancing using Decentralized decision-making Mobile Agents) strategy introduced the concept of decentralized decision making among the web servers in the cluster for processing requests.

Also WLDMA (WANLDMA) was introduced, this allow each server to process client requests independently and periodically interacts with others to share the workload.

The performance of load distribution on the three servers using WLDMA is two times better than the scheme without load balancing. The result of this work shows that the WLDMA scheme can improve the system throughput when increasing the number of servers.

The use of mobile agents implies high flexibility, low network traffic and high asynchrony. Also, the result shows that no replica remains idle at any time when the other replicas are processing more request of each nodes.

It is clear from this work that waiting time of the request will go up leading to increase in response time. This policy use SI (sender initiated) which can lead to poor performance of the network, because of varying value of state load information of other servers if applied to larger distributed and heterogeneous environments.

MALD (Mobile Agent based Load Balancing) framework that uses mobile agents technology was proposed to implement scalable load balancing on distributed web servers. To achieve this, two load balancing schemes were presented. Cluster of web servers and wide area network (www) web servers. The two schemes used MALD and tested in a well distributed environment. The load balancing scheme on www uses shared LIA (load information agent) for load information gathering [2].

The MALD framework performs well compared to message-passing paradigm and no load balancing. The system throughput, network traffic is better in LAN; while in WANs, the LIAs are restricted to domains. Series of simulations were done and the results are better than message passing paradigm. The drawback of this work is that, the waiting time of each task in the queue and within the entire system may be more, because LIA will continue to move from one server to the other until the best server is found. In this approach, the domain name system mentioned did not consider the minimum number of servers that must be in a domain, in essence the policy may not perform well in a system with large number of servers. The scalability and good response time may not be achieved. Our study address this problem by identify the server that is overloaded that needed to share with other server that is under-loaded or idle in the cluster. This is achieved by proposing two agents, mobile and stationary agents. The stationary agents periodically determine the status of each server.

The work in [6] addresses the escalating complexity and mobility of today's network. This has led to the increase in application of mobile agent paradigm; however load balancing (LB) is one of the important problems of computer heterogeneous network.

In that paper the authors introduced a decentralized algorithm for diffusion dynamic LB based on mobile agent paradigm. The architecture of three types of agents was employed to meet the requirements of the proposed diffusion LB algorithm. Packet format was suggested for each agent as a data communication packet. Two performance metrics were used in the proposed approach, they are: the average response time of the clusters and the variance of the load over the network. The algorithm was tested with the situation where there is NoLB. In addition it minimizes the bandwidth consumption and disruptive nature of wireless communication. This algorithm offers an alternative to client server based solution with appreciable bandwidth; it implies that it would be more appropriate in a wireless data communication application. The clusters size was not mentioned, nodes threshold was not stipulated and therefore there is no load inhibition, so in large networks it may lead to instability of the system SI (sender-initiated) was used in the research work. In our proposed algorithm we proposed threshold for the RAM and CPU. We set the two at 90% and 80% respectively.

Two of the existing static LB policies based on M/M/1 queues were reviewed [7]. This was done to address poor system performance due to loads imbalance. The authors proposed two new dynamic LB policies for multi-user jobs in heterogeneous distributed systems. The two dynamic policies proposed are: DGOS and DNCOOPC, the two static policies reviewed are GOS and NCOOPC. The result of the simulations shows that static has low communication overheads; the dynamic schemes show superior performance over the static policies. But as the overheads increases, the dynamic policy is expected to perform similar to the static policies.

According to [8], challenge faced by these cluster-based applications is how to maintain desired real-time performance and service availability in the face of highly unpredictable work-loads in an open environment such as internet. The authors investigate the problem of multiprocessor utilization control in large scale real-time clusters.

The work proposed an effective approach that will dynamically enforce desired utilization bounds on all the processors in a cluster through online admission control. The authors present DUC-LB, a novel Distributed Utilization Control algorithm designed for large scale real-time clusters with LB. DUC-LB can provide robust utilization guarantees to all processors in a real-time cluster through feedback-based admission control. The result shows that DUC-LB is rigorously designed based on recent advance in distributed control theory, stability analysis and simulation results demonstrate that DUC-LB can dynamically enforce desired utilization set points under a range of dynamic workloads. FCU-LB was used as a baseline for performance comparison. In terms of utilization, DUC-LB performs better. In this study, the major drawbacks are:

- the topology
- the rejection of tasks
- communication cost

Challenges introduced by fine-grain services [9] for server workloads can fluctuate rapidly for those network services. Fine-grain services are sensitive to various overhead which is hard to accurately capture in simulations.

In recognizing this limitation the authors developed a prototype that its implementation is based on a clustering infrastructure and conducted evaluations on a Linux cluster. The study and evaluations identify techniques that are effective for fine-grain services and lead to the following conclusions:

• Random polling based load-balancing policies are well suited for fine-grain network services;

• A small poll-size provides sufficient information for load balancing, while an exclusively large poll size may even degrade the performance due to polling overhead;

• An optimization of discarding slow responding polls can further improve the performance up to 8.3%.

The drawback of this policy is stale load information which is sensitive to the fine-grain network services. In the literature, these are done locally, because it is delay sensitive. In their work, authors use poll size that must not be more than two. It implies that it can not support broadcast policy and partly support random policy.

Finally, this policy cannot support a largely distributed decentralized network environments.

This research work [10] focus on networks connecting the different nodes typically exhibit significant latency in the exchange of information and the actual transfer of loads. The load balancing policy rely on latest information about the system of nodes, which in turn may result in unnecessary transfer of loads while certain nodes remain idle as the loads are in transit. The strategy introduced by the authors put into account the following three aspects:

• the connectivity among the nodes;

• the computational power of each node and;

• the throughput and bandwidth of the system, the policy was described by defining some parameters.

Values of (C_i term) are critical to the stability and efficiency of the load balancing policy. The results show that this policy is more scalable. The method used here leads to optimal results, when the C_i/C_j has no delay factor. Transfer delays incurred when tasks are migrated from node to another may be unexpectedly large and result in a negative impact on the overall system performance. The traffic may be a little higher in a large heterogeneous distributed environment. The performance may not give optimal result. We take care of this in our work by introducing Job Migration regulatory mechanism.

In [11], the authors explained that modeling the dynamic load balancing behavior of a given set of tasks is difficult. They define an approximation problem using "bi-modal" task distribution, which serves as an accurate abstraction of the original task set. This approximation consists of only two task types and can therefore be tackled analytically. This work model the run time of the slowest processor as this will determine the overall run time of the parallel applications. All comparably under-loaded nodes will be probed before a suitable node will be located. The results of the experiments using synthetic micro-benchmarks and a parallel mesh generation application, demonstrate that this technique when used in-conjunction with the PREMA runtime toolkit can offer users significant performance improvements over several well known load balancing tools used in practice today. Little overhead is incurred using PREMA, when compared with others. The drawback of this work is the adaptive issue that was not taking care of. In essence, the decision of each node will be relatively poor, therefore the response time of each node will be prolonged and it will affect overall performance. Global load balancing cannot be easily achieved, because stale state information will be available to several nodes.

This work [12], overviewed the homogeneous and heterogeneous clusters. A general station model from a heterogeneous cluster was proposed.

The model used considered a station as characterized by the following elements:

• Unique ID
• Computing power
• Availability
• Task lists

Three network topologies were considered: mesh, trees and hyper-cube. The three configurations prove to be the least efficient. The hypercube seems to have best performance. The mesh topology is at the middle in terms of performance. Authors proposed a more intelligent routing technique as a future work. Point of failure could be experienced in tree topology. The success of this policy rely much on network connections and available bandwidth to really address the load transfer from neighbors to neighbors until it reaches a powerful node among them. It then implies that, the scheme is not conservative at all and it is not scalable in the overall performance. Our proposed work addresses this drawback by introducing cluster-based system that will allow server to join and exist at will. The cluster balances the loads by reducing the jobs intake through job migration regulation. This allows good quality of service (QoS) in the network. CASID policy is discussed in the next section.

4. CASID Policy

The proposed policy tagged Cooperative Adaptive Symmetrical Initiated Diffusion (CASID) consists of sender and receiver communicating through a communication networks links. The sender and receiver refer to as servers. They are web servers and database servers as shown in **Figure 1**. The load threshold of each server describes whether it is sender (overloaded) or receiver (under loaded). The main goal of this policy is to minimize response time and increase system throughput. CASID implements dynamic policy, where the server can become a sender or a receiver at anytime. In essence, the policy gives room for the server to interact and there is inter-communication among the servers. Also, CASID has adaptive properties whereby server can accept or transfer load to/from other servers. Symmetric-initiated diffusion affirmed the relationship among the servers. Diffusion can be used to describe dynamism of distributed networks system (DNS). The main problem of any load balancing policy is how to get the right server to establish the job exchange in the distributed networks system. The heterogeneous nature of dynamic policy explains whether a better performance will be achieved. CASID policy proposed a model that attempt to minimize the response time of the entire DNS to the user. The proposed policy will increase system throughput when compared with the existing policies. In this case, CASID policy considered the global load balancing within the DNS a priority.

4.1. Job Migration Regulatory Mechanism

Job migration determines the success of any load balancing policy, especially in the area of minimizing bandwidth consumption. Most of the previous works did not lay much emphasis on this bandwidth minimization as a key issue in any distributed networks environment. It is well known fact that bandwidth consumption must be minimized. There must be a control measure on job migration whenever LB is being deployed. As a result, the proposed policy will deal with this issue with more emphasis on the server status information whenever any initiation wants to be made by servers. In this study, server status information is gathered through the stationary agents. The stationary agents periodically check the status of the I/O networks, memory and CPU. This is done to have closed to real-time status information of every server. The mobile agents do the job transfer whenever there is need for such transfer. If this is done it will enhance better performance of this LB policy.

The essence of this regulatory mechanism is to achieve the following:

- reduce bandwidth consumption
- achieve local and global load balancing
- decrease response time of the users
- reduce waiting time of the users
- reduce congestion in the system

However, the proposed policy will address job migration mechanism. The server determines its load threshold by calculating the load on the queue and in service. The server status information will be broadcasted to other servers. Servers that meet such requirements will signify. The sending and receiving servers will establish communication and job migration can commence as illustrated in **Figure 2**. In a situation where there is no under-loaded server that meets up with the conditions set by the sender, such task will be processed by the sending server. Only profitable job migration will be allowed in this policy. This is further discussed in Section 5 of this paper.

5. System Model

This software model has some modules and each module

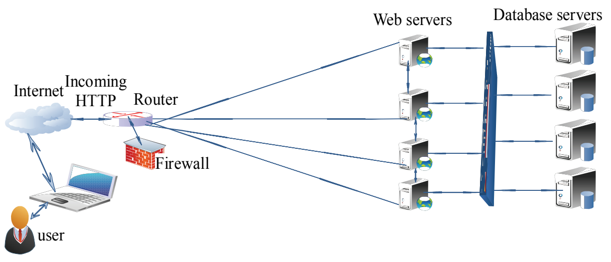

Figure 1. Proposed CASID policy architecture.

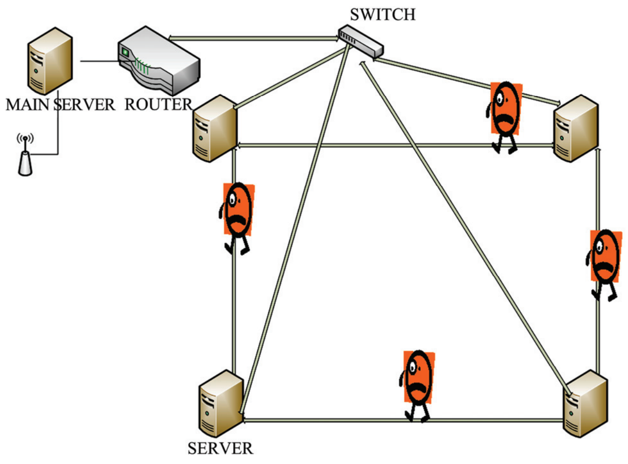

Figure 2. Job migration using dynamic (mobile) cognitive agents.

describes the operation of ServerAgents. Simple modeling language was used as illustrated in **Figure 3** to model the behavior of the agents on the server. This model will translate to the software that will be developed using JADE. Some of these modules are listed and discussed below.

IDLEServerBehaviour()

This module is used by the ServerAgent whenever it enters the IDLE state. This module is actually a sub-class of the ServerAgent class and it is used to interact with BUSY ServerAgents. As a class, this module has its own modules which are action and done. The action module is executed each time the ServerAgent is called on. While done is executed when the ServerAgent's state is to be changed to IDLE.

BUSYServerBehaviour()

This module is used by the ServerAgent whenever it enters the BUSY state. This module is actually a sub-class of the ServerAgent class and it is used to interact with IDLE ServerAgents. As a class, this module has its own modules which are action and done. The action module is executed each time the ServerAgent is called on. While done module is called when the ServerAgent's state is to be changed to IDLE.

registerDFServices()

This module is called when the ServerAgent is activated or instantiated. The module registers this agent with the Directory Facilitator (DF) which records the name, address, location of the agent, its host and the services the agent renders. This registration makes interaction with other agents possible and seamless.

MigrateJobCompleted()

This module is used to load migrated and completed jobs unto the ServerAgent (migrater) queue.

jobLoader()

This module loads new jobs unto the ServerAgent's queue.

jobLoaderFinishedMigrate(*jobSerialisable jbL*)

This module is used to get some statistics on migrate job like completion time, waiting time, migrate time, throughput, transfer rate of job between ServerAgents etc.

ServerIsIDLE()

This module returns the state of the ServerAgent (*i.e.* IDLE or BUSY).

5.1. Mobile Agent Modules

ServerInteractionBehaviour()

This module is used by the mobileAgent to interact with the IDLE ServerAgent during the mobileAgent's sojourn at the IDLE ServerAgent's domain. It is also used in interacting with the BUSY serverAgent.

registerDFServices()

This module is called when the ServerAgent is activated or instantiated. The module registers this agent with the Directory Facilitator (DF) which records the name, address, location of the agent, its host and the services the agent renders. This registration makes interaction with other agents possible and seamless.

sendJobToServer(*AID ServerAgent, String conID*)

This module is used by the mobileAgent to transfer jobs to ServerAgents either when the job is finished.

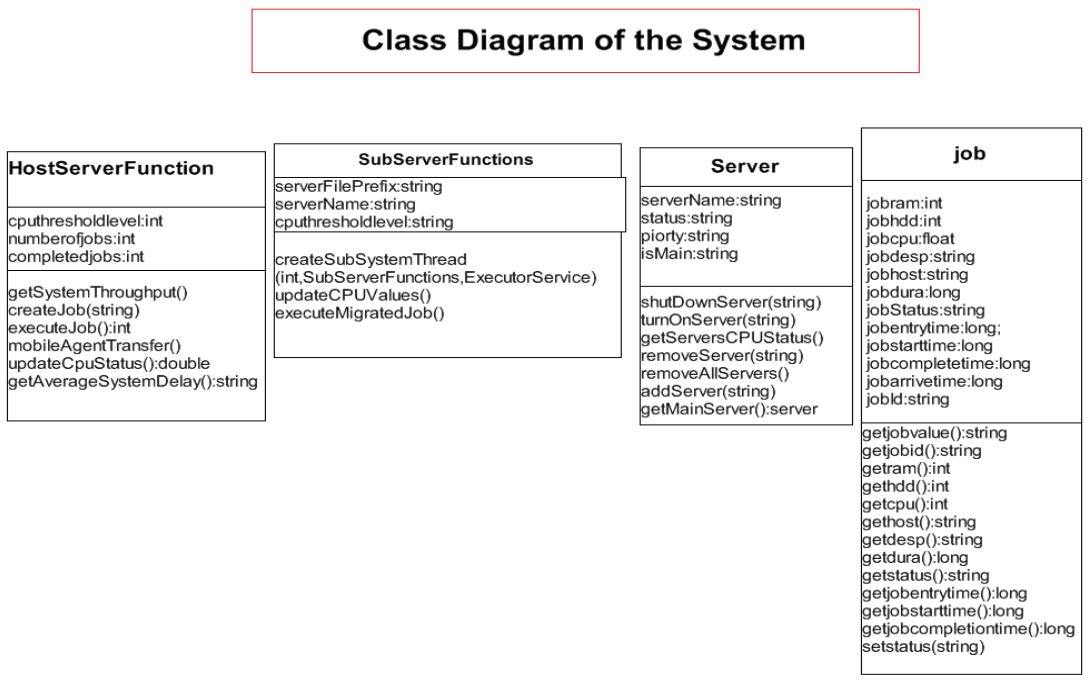

Figure 3. Class diagram of the system model.

6. Theoretical Framework

Having discussed related issues concerning this paper, we need to address the framework in which this paper was based on. We want to compute the network traffic, system throughput, mean response time and system utilization of CASID and compare with PLB.

We consider N exponential servers each having a different service rate μ_i with i € (1, N). Each server has its own local queue and the number of jobs in each queue is represented by q_i with Poisson distribution stream of strength λ (lamda). However, communication cost is considered to be significant, the arrival stream at each server is $\lambda + \delta_i$, where λ is a constant rate at which tasks are initiated at any node (user) and δ_i is the rate of transfer of jobs to server i.

The expected delay job i (D_i) will experience if sent to the j-th server is

$$D_j = (1+q_j)/\mu_j \tag{1}$$

where μ_i service rate and q_i no of jobs in the queue.

The selected (destination) sever will normally be the one for which D_j is minimal. We have another equation when communication delay is added to Equation (1).

Thus

$$D_j = \left[(1+q_j)/\mu_j\right] + t_{com(i,j)} \tag{2}$$

where μ_i service rate and q_i no of jobs in the queue. $t_{com(i,j)}$ represent communication delay experienced by jobs when it is being transferred from server i to server j.

If the job is processed locally then, $t_{com(i.i)} = 0$ in addition, the expected execution time is considered in Equation (3).

That is,

$$E\left[t_{exec}\right] = t_{com(i,j)} + (1/\mu_j) \tag{3}$$

where $(1/\mu_j)$ is the service rate.

The throughput of was determined using Equation (4)

$$T(j) = \left[\mu_j / \left(\lambda + \mu_j\right)\right]_j^{1+n} \tag{4}$$

where $^{n}{}_{j}$ is the total number of jobs in the j-th queue including any job in service. For each server j, the ratio $\mu_j/(\lambda + \mu_j)$ is less than 1. Thus the greater $^{n}{}_{j}$ is the smaller $T(j)$ will be and the less "recommendable" (exponentially) the server j becomes to be selected as a destination server.

$$1 + n_j = 1 + s_j + q_j + t_j \tag{5}$$

where q_j is the number of jobs in queue and t_i is the number of jobs being transferred to server j, that are expected to arrive before the next job initiation in the system. t_j is simply the exact number of jobs being transferred to server j at the instant that the decision is being made.

As for s_j, it is equal to zero if server j is idle. If server j is busy s_j is equal to 1, if the job currently in service at server j is not expected to be completed before the next job initiation *i.e* if $(1/\mu_j)$ – time(already spent in service) $\geq 1/\lambda$ (arrival rate). Otherwise, s_j is equal to zero [13].

The load size on each server was determined as

$$\begin{aligned} Load\_size = v_1{}^{*}CPU\_load + v_2{}^{*}\acute{K}/Ъ \\ + v_3{}^{*}free\_mem \end{aligned} \tag{6}$$

where CPU_load is the workload on the server, measured in the length of jobs in queue; $\acute{K}$ is the number of active connections on the server; Ъ is the maximum number of connections allowed to the server; *free_mem* is the percentage of free memory space;

$$v_1 + v_2 + v_3 = 1$$

System utilization was determined from Equation (7)

$$\alpha_{busy} + \alpha_{idle} = 1 \tag{7}$$

the parameter α_{busy} is the time averaged of busy servers in the system. A busy server is defined as a server having one job in service and zero or more jobs in queue. System utilization involves the percentages of CPU, Memory and I/O being used at a point in time [14].

The network traffic was determined using Equation (8)

$$R = [\gamma_{ij}] \tag{8}$$

where γ_{ij} is the average transmission rate from source (server i) to destination (server j). In some cases, we define the requirement matrix as $R = \rho R^{'}$, where $R^{'}$ is a known basic traffic pattern and ρ is a variable scaling factor usually referred to as traffic level [15]. In general, R or $R^{'}$ cannot be estimated accurately a *priori*, because of its dependence upon network parameters (e.g. allocation of resources to computers, demand for resources etc which are difficult to forecast and are subject to changes with time and with network growth.

7. Experimental Results

To analyze the proposed load-balancing algorithm, it was implemented on a cluster-based web server. For evaluation of CASID algorithm, throughput and average response time were selected as criteria [16,17]. To give a better evaluation, a network traffic analysis of the algorithm was also provided.

We implement two of the commonly used load-balancing algorithms PLB and NO-LB, we compare the performance of the proposed algorithm with them. The PLB is a commonly used agent-based algorithm that uses the average CPU threshold to determine whether a server is busy/idle. PLB uses the same update intervals as used in the CASID to collect CPU and memory loads from web server nodes.

The following sections describe the implementation setups which are used in our experiments and finally the results are analyzed.

7.1. Implementation Setup

We implement the experimental test bed with the hardware and software configurations as described below.

7.1.1. Hardware Setup

The clustered web server consists of four (4) machines configured as follows. Each web server is connected to a dedicated database server node, so four (4) machines are used as database servers. Also, variable numbers of machines were used as client emulators to simulate real users' interactions with the web site. The web server nodes and corresponding database server nodes are having Pentium 4 processors (ranging from 1 GHZ to 2 GHZ) with minimum of 512 MB of DDR RAM.

7.1.2. Software Setup

All the machines in the cluster run windows xp 2007 operating system; we developed our software using JADE as discussed in the previous section to perform the experimental test. We implement a client-browser emulator to generate load against the cluster. We vary the loads on the cluster by varying the number of concurrent clients. We use to determine which machine become overload and ready to perform job migration. The overloaded server broadcast its status to other servers in the cluster. Since the developed software are installed and configured on each server. The server that is under-loaded or idle will receive the message and acknowledge the message and the jobs are transferred. Matlab [18] was used to plot the graphs and the data were generated from the results from the simulations

7.2. Implementation Results

In the following sections, we demonstrated the results of our experiments with the CASID benchmark (non-standard). We change load intensity, machines and measure the cluster throughput, response time and network traffic for the two load-balancing algorithms PLB and CASID).

7.2.1. Network Traffic Evaluation

Figure 4 compares CASID policy network traffic with PLB scheme. CASID policy performs better than the PLB scheme; it reduces congestion in the network, because it introduces job migration regulatory mechanism where loads cannot migrate to other server(s) without acknowledgement from other servers. If no acceptance from other servers, such server will process the tasks till completion.

We consider situations where all the servers are idle and busy. The CASID scheme shows that when all the servers are busy the network traffic was low or insignificant. As the number of busy and idle servers increase in PLB scheme the traffic increase accordingly, because of non-implementation of job migration regulatory mechanism as in CASID policy. CASID policy allows an overloaded server to broadcast its status before it can send jobs to other servers. The receiving server will acknowledge before the jobs is being transferred to the receiver (server). PLB scheme will send to under-loaded server irrespective of volume of jobs at hand and processing capability of that server. We work with maximum of 2 Mbps network within the average of 90 minutes.

7.2.2. Throughput Evaluation

System throughput was evaluated and Analyzed in **Figure 5**, we generate requests through requests emulator, the number of requests/sec range from 0 to 100 and the requests range from 0 to 1200. The two schemes were subjected to the same environments and conditions. The

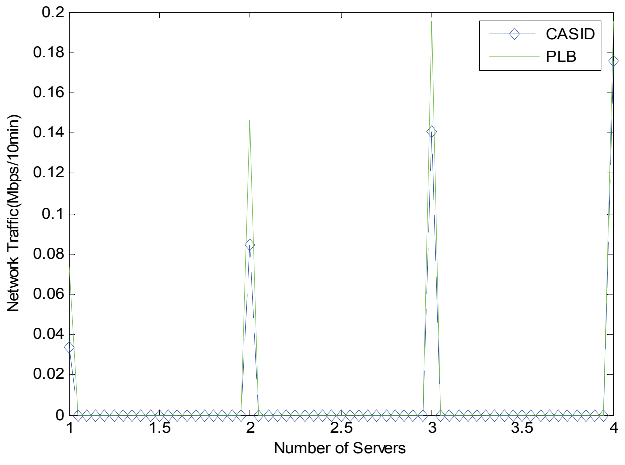

Figure 4. Network traffic (average 90 minutes).

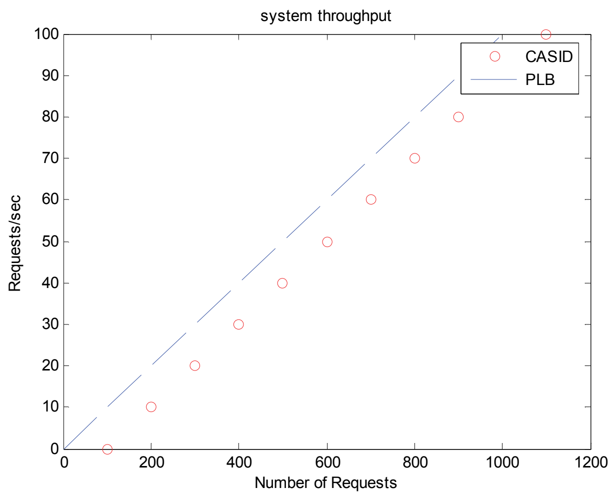

Figure 5. System throughput.

schemes performance was close. CASID still has better performance, because the servers are able to manage their resources by ignoring load transfer messages.

We used **Figure 6** to explain the throughput of the system without load balancing and compare it with our CASID policy. We discover that the job done when servers are many only have little improvement compare with when server is one. The mobile agents assisted our policy couple with our job migration scheme. The CASID perform well than when the system was not load balancing at all.

7.2.3. Mean Response Time Evaluation

Response time was evaluated and analyzed in **Figures 7-9**, these figures analyze the high, medium and low heterogeneous of the servers used. The inter-arrival of the requests to the system was processed by different server with their processing capabilities. This determined the mean response time of the servers at different time interval. Inter-arrival of the requests was plotted against response time. The CASID, PLB and NO-LB schemes were compared. CASID and PLB have better performance. The CASID has improved performance over PLB, because of its job regulatory mechanism scheme. Different machines were used as servers to confirm the effect of heterogeneity of the nodes on the mean response time of the cluster.

8. Conclusions and Future Work

This paper addresses un-regulated jobs/tasks migration among the servers. Considering distributed web services

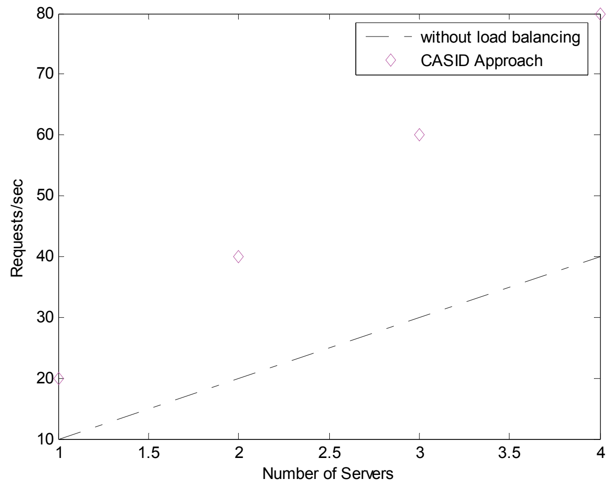

Figure 6. System throughputs of the CASID policy and the case without load balancing.

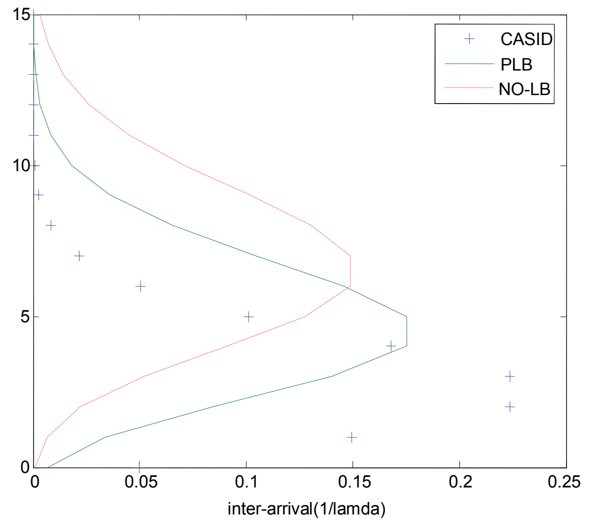

Figure 7. Mean Response time of a highly heterogeneous system.

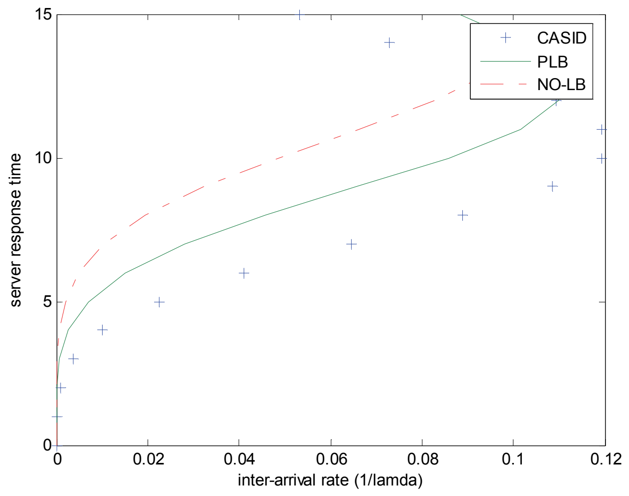

Figure 8. Mean Response time of a system with medium heterogeneity.

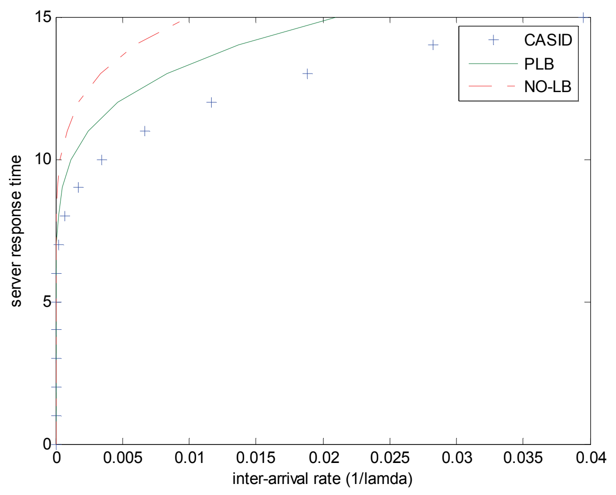

Figure 9. Mean response time of a system with low heterogeneity.

with several servers running, a lot of bandwidth is wasted due to unnecessary job migration. It is important to develop a policy that will address the bandwidth consumption while loads/tasks are being transferred among the servers. This work attempts to regulate this movement to minimize bandwidth consumption. The results of this first phase of the experiment as illustrated in the graphs show that our policy performs better. The second experiments that will be carried out in future will consider the system utilization, response time and throughput that will be experienced by the job while running the software on LANs and open networks (Internet).

9. References

[1] W. Winston, "Optimality of the Shortest Line Discipline," *Journal of Applied Probability*, Vol. 14, No. 1, 1977, pp. 181-189.

[2] J. Cao, Y. Sun, X. Wang and S. K. Das, "Scalable Load Balancing in Distributed Web Servers Using Mobile Agents," *Journal of Parallel and Distributed Computing*, Vol. 63, No. 10, 2003, pp. 996-1005.

[3] N. Nehra, R. B. Patel and V. K. Bhat, "A Framework for Distributed Dynamic Load Balancing in Heterogeneous Cluster," *Journal of Computer Science*, Vol. 3, No. 1, 2007, pp. 14-24. http://www.scipub.org/fulltext/jcs/jcs31 14-24.pdf

[4] R. B. Patel and N. Aggarwal, "Load Balancing on Open Networks: A Mobile Agent Approach," *Journal of Computer Science*, Vol. 2, No. 4, 2006, pp. 337-346. http://www.scipub.orga/fulltext/jcs/jcs24337-346.pdf

[5] M. Aramudhan and V. R. Uthaiaraj, "LDMA and WLDMA: Load Balancing Strategies for Distributed LAN and WAN Environments," *International Journal of Computer Science and Network Security*, Vol. 6, No. 9B, September 2006. http://www.ijcsns.org/04_journal/200609/200609B 11.pdf

[6] M. Saeb and C. Fathy, "Modified Diffusion Dynamic Load Balancing Employing Mobile Agents," Computer Engineering Department, Arab Academy for Science, Technology & Maritime Transport, Alexandria, 2001. http://www.magdysaeb.net/images/wseapaper3.pdf

[7] S. Penmatsa and A. T. Chronopoulos, "Dynamic Multi-User Load Balancing in Distributed Systems," Proceedings of IEEE International Parallel and Distributed Processing Symposium, Long Beach, 26-30 March 2007, pp. 122-131.

[8] Y. Fu, H. Wang, C. Y. Lu and S. Ramu, "Distributed Utilization Control for Real-Time Clusters with Load Balancing," *Proceedings of the* 27*th IEEE International Real-Time Systems Symposium*, Rio de Janeiro, 5-8 December 2006, pp. 137-146.

[9] L. Chu, K. Shen and T. Yang, "Cluster Load Balancing for Fine-Grain Network Services," *Technical Report TR-CS*2002-02, Department of Computer Science, University of California, Santa Barbara, 2002.

[10] J. Ghanem, C. T. Abdallah, M. M. Hayat, J. Chiasson and J. D. Birdwell, "Distributed Load Balancing in the Presence of Node Failure and Network Delays," 2006. http://www.eece.unm.edu/Ib/papers/dynamicLB2.pdf

[11] K. Barker and N. Chrisochoides, "Practical Performance Model for Optimizing Dynamic Load Balancing of Adaptive Applications," *Proceedings of the* 19*th IEEE International Parallel and Distributed Processing Symposium*, Denver, Vol. 1, 4-8 April 2005, pp. 28-37.

[12] F. Leon, M. H. Zaharia and D. Galea, "A Simulator for Load Balancing Analysis in Distributed Systems," In: A. Valachi, D. Galea, A. M. Florea, M. Craus, Ed., *Thenologii Informationale*, Editura Unvierisitatii, Suceava, 2003, pp. 53-59.

[13] K. Y. Kabalan, W. W. Smari and J. Y. Hakimian, "Adaptive Load Sharing in Heterogeneous Systems Polices, Modifications, and Simulation," 2009. http://ducati.doc.ntu.ac.uk/uksim/journal/Vol-3/No%201&2%20Special%20issue%20karatza/kabalan/kabalan.pdf

[14] K. S. Chatrapati, K. P. Chand, Y. R. Kumar and A. V. Babu, "A Novel Approach for Effective Dynamic Load Balancing by Using Tendensive Weight," *International Journal of Computer Science and Network Security*, Vol. 8, No. 6, June 2008, pp. 42-48.

[15] M. Gerla and L. Kleinrock, "On the Topological Design of Distributed Computer Networks," *IEEE Transactions on Communications*, Vol. 25, No. 1, 1977, pp. 48-60.

[16] M. Andreolini and E. Casalicchio, "A Cluster-Based Web System Providing Differentiated and Guaranteed Services," *Cluster Computing*, Vol. 7, No. 1, 2004, pp. 7-19.

[17] M. Andreolini and S. Casolari, "A Distributed Architecture for Gracefully Degradable Web-Based Services," *Proceedings of IEEE International Symposium on Network Computing and Applications*, Cambridge, 24-26 July 2006, pp. 235-238.

[18] MathWorks, Inc., 2008. http://www.mathwork.com