International Journal of Engineering Trends and Applications (IJETA) – Volume 2 Issue 6, Nov-Dec 2015

RESEARCH ARTICLE OPEN ACCESS

Data Synchronization over Cloud Accounts

Aliraza Punjani [1], Singh Vijendra [2], Ankit Vasa [3], Prof. Sudhir Dhage [4]
Student [1], [2] & [3], 2Associate Professor [4]
Department of Computer Science and Engineering
University of Mumbai
Mumbai - India

ABSTRACT

Transferring data from one cloud account to another is a tedious task. It is necessary to download the contents that one needs to transfer to his/her computer before it can be uploaded to the destination cloud account. This process is not only time consuming but also requires that the end user's computer has enough resources to temporarily hold data that is to be transferred. Data Synchronization over Cloud Accounts is a web based application that attempts to connect technologies, like Google drive and Dropbox, that don't really get along very well, and make them work as one. Our application devises an effective load balancing algorithm to transfer data directly from Dropbox to Google drive or vice-versa without having to download them to end user's computer's first. The algorithm effectively breaks apart the user tasks into individual independent jobs which are allocated to proxy servers in order to complete the task at a faster rate while balancing the system load. Thus, we developed an effective system using divisible load balancing theorem to maximize or minimize different performance parameters such as throughput and latency for balancing the load on the server at a particular instant. We developed integrated measurement for the total load level of a server datacenter as well as the load level of each server. We have calculated and compared average response time of our load balancing algorithm with Honeybee Algorithm and Round-Robin Algorithm. Simulation results show that our algorithm has good performance with regard to total load level, average imbalance level of each server, as well as overall running time.

Keywords:- Data Synchronization, Google drive, Dropbox, load balancing, Honeybee Algorithm, Round-Robin Algorithm

I. INTRODUCTION

One of the main components of a distributed system is the distributed process scheduler that manages the resources of the system. The efficient usage of the large computing capacity of a distributed system depends on the success of its resource management system. A distributed process scheduler manages the resources of the whole system efficiently by distributing the load among the processors to maximize the overall system performance[1]. The distributed scheduler must perform the load distributing operations transparently, which means the whole system is viewed as a single computer by the users of it [2].

A distributed system consists of independent workstations connected usually by a local area network. Users of the system submit jobs to their computers at random times. In such a system some computers are heavily loaded while others have available processing capacity. The goal of the load distributing schema is to transfer the load at heavily loaded machines to idle computers, hence balance the load at the computers and increase the overall system performance.

It is a process of reassigning the total load to the individual nodes of the collective system to make resource utilization effective and to improve the response time of the job, simultaneously removing a condition in which some of the nodes are over loaded while some others are under loaded. A load balancing algorithm which is dynamic in nature does not consider the previous state or behavior of the system, that is, it depends on the present behavior of the system. The important things to consider while developing such algorithm are : estimation of load, comparison of load, stability of different system, performance of system, interaction between the nodes, nature of work to be transferred, selecting of nodes and many other ones . This load considered can be in terms of CPU load, amount of memory used, delay or Network load.

Thus we aim to design an effective load balancing mechanism for optimal resource utilization which helps the application to balance the load among the cluster machines. The application will thus allow users to move or copy files from one cloud storage (Dropbox) to another (Google Drive) or vice versa without using end user's personal computer's resources such as its central processing unit's processing time and bandwidth. Users can thus switch cloud storage providers and thus do not have to download and upload the files/folders on their machine rather than the task is accomplished by workstations in the underlying distributed network. No plug-ins or scripts are required to be installed in the browser and allows client to manage everything from applications web interface. Secure data transfer to maintain privacy of user's data.

II. ARCHITECTURE

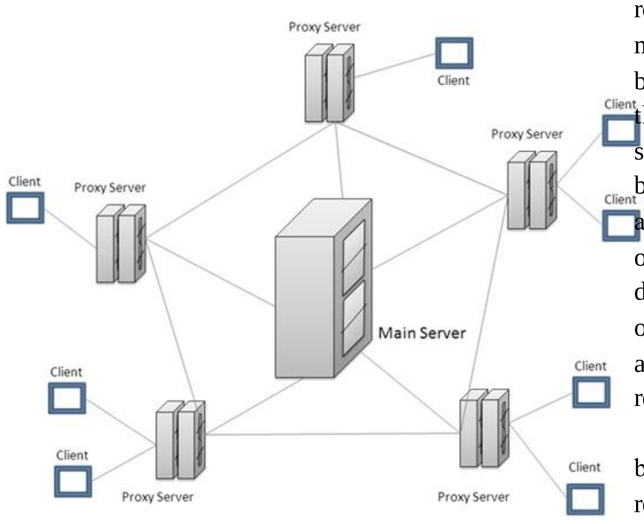

Fig.1 Proposed System Architecture

The major high-level components of the system include a central server, the proxy servers and the clients. The client interacts with the proxies using HTTP protocol. The central server interacts with the proxies using HTTP and FTP protocols. The proxy servers interact with each other using HTTP and FTP protocols. Canvas JS – as a monitoring tool. The end-user accesses the application using a web browser.

Initially the central server has all the applications while the proxy servers are empty. Now suppose a client hits a proxy to use one of the available applications. Firstly the client is authenticated. To do this it is checked if it is registered with this proxy. If no then this proxy asks the central server to see if it has registered with some other proxy and hence downloads his details. Now if this proxy server has the application then the client is allowed to proceed. Else this proxy asks its own neighbors if they have the requested application. If any one of them has it then whosever reply comes first, the client is redirected to that proxy. If none of them has then it downloads the application directly from the central server. In both the above cases, all the required details of the client are then transferred to the neighboring proxy.

Load Balancing is done periodically by the central server calculating the load at each of the proxy servers. A need for scale-up comes into picture if each of the proxy servers is overloaded and a new user request comes. Scale down is done if a significant number of proxies are under loaded.

III. IMPLEMENTATION

A. *Honey Bee Algorithm*

In [2] and [3], Honeybee algorithm is used for co-ordination of servers hosting Web Services. In load-balancing operation, each server takes a particular bee role with probabilities px or pr. These values are used to mimic the honeybee colony whereby a certain number of bees are retained as foragers – to explore (px) ; rather than as harvesters – to exploit existing sources. A server successfully fulfilling a request will post on the advert board with probability pr. A server may randomly choose a proxy server's queue with probability px(exploring), otherwise checking for an advert (watching a waggle dance). In summary, idle servers (waiting bees) follow one of two behaviour patterns: a server that reads the advert board will follow the chosen advert, then serve the request; thus mimicking harvest behaviour.

A server not reading the advert board reverts to forage behaviour; servicing a random proxy server's queue request. An executing server will complete the request and calculate the profitability of the just-serviced proxy server. The completed server (i.e. returning bee) influences system behaviour by comparing its calculated profit with the colony profit on the advert board, and then adjusts px (controlling the explore/exploit ratio) and colony profit accordingly. If the calculated profit was high, then the server returns to the current proxy server; posting an advert for it (waggle-dancing) according to probability pr. If profit was low, then the server returns to the idle/waiting behaviour described above. Initially, every server starts with explore/forage behaviour, and as requests are serviced, the advert waggle-dance-guided behaviour begins to emerge. Given a robust profit calculation method, this behaviour pattern provides a distributed and global communication mechanism; ensuring "profitable" proxy servers appear attractive to and are allocated to available servers.

B. *Proposed Algorithm*

Step 1: To calculate the average execution time of each subtask.

Step 2: If the required execution time of a subtask is less than or equal to the average execute time then carry out the subtask to execute normally.

Step 3: If the required execution time of a subtask is greater than the average execute time then executing time is set to ∞ (the execution time is too long so that cannot to be considered). The other nodes that had been executed will re-enter into the system to participate the execution of subtask.

Step 4: Repeat Step 1 to Step 3, until all subtasks have been executed completely.

The tasks can be assigned to execute quickly by the integrated scheduling algorithm and the effective service nodes can be chosen by the threshold in a three-level cloud computing network.

The two-phase scheduling algorithm integrates OLB and LBMM to assist in the selection for effective service nodes. First, a queue is used to store tasks that need to be carried out by manager (N_0), then the OLB scheduling algorithm within "threshold of service manager" is used to assign task to the service managers in second layer (N_1,N_2,N_3,N_4,N_5). However each consignation carries out of the task have different characteristic, so the restriction of node selection is also different. An agent is used to collect the related information of each node.

According to the property of each task, the threshold value of each node is evaluated and a service node will be assigned. However, in order to avoid the execution time of some is too long and affect system performance, "threshold of service node" is used to choose the suitable service node to execute subtask. [4]

C. *Pseudo Code*

Input: User request (Tasks)
Output: Balanced Load

```
Begin
1. Initialize  si in Vj serving Qi, Revenue rate T,
Advert: posting prob p, reading prob n, read interval T
2. While(true)
        While Q not empty do   //service queue

          Serve request();

        if T expired  then
           compute revenue rate;
        adjust n from lookup table;
3. If Flip(p) == True then Post Advert;
        If T expired  && Read(ri)==True  then
        If forager then Select/Read advert id Vk
        //randomly  select
        Else proxy server id Vk //randomly select
          If Vk  Not.Eq Vj  then Switch (Vk)
          //migrate to proxy server
4. End while
End

Service request()
{
Loop for i from 1 to n
Loop for j from 1 to m
          timeChart=calculateTime(task[i],node[j])
          //get time estimates of each task
end loop
end loop

while(tasks exists)
unblock all nodes    //unblock all nodes for the new
iteration
minPair[]={all  min task-node pairs}
 //get the minimum  task-node pair

loop for i from 1 to n
if(minPair[i].task.time<=task.threshold)
 //execute the task only if doesn't exceed the
 // threshold of service node
minPair[i].task.execute()
minPair[i].node.block()
end if
end loop
end while
}
```

Parameters:

timeChart	Task-node time estimation table
minPair	Minimum value from the set
task.time	Estimated time for the task
task.threshold	Threshold time for the task

IV. APPENDIX

Process transfer policies for transferable load

There are three strategies that can be implemented to man-age transferable load. First strategy requires that that each load be time stamped before load balancing initiates. The timestamp for the previous balancing iteration must also be available. With these two parameters new processes can be distinguished from the older ones and load balancing can be carried out assuming the older processes to be non-transferable.

The second strategy doesn't differentiate between old pro-cesses and processes which have arrived in the current itera-tion. All the processes are candidates for a transfer and no effort is made to avoid transfer of older processes. A draw-back of this strategy is that it causes unnecessary transfer of resident older processes which do not require transferring thus incurring unnecessary overheads associated with the transferring of processes from one to another.

The third and the final strategy aims at minimizing the transfer of processes from one node to another by transfer-ring only an optimal number of processes which help achieve a balanced state of the system. At the same time it aims at eliminating the overhead involved with time stamp-ing strategy.
The strategy uses a Load Deviation Factor (σ)

$$\sigma = \sqrt{\frac{1}{Np}\sum_{i=1}^{N_p}(l_i - \mu)^2}$$

Where

l_i = individual load
Np = total number of processes
$\mu = \frac{1}{Nn}\sum_{i=1}^{N_p}(l_i)$

Condition for balanced node:

$$(\mu - \sigma) \leq Np(l) \geq (\mu + \sigma)$$

Nodes having loads in the range $\mu \pm \sigma$ are assumed to be balanced and their loads are not transferred. Nodes having loads above or below the specified region have their exceeded loads as candidates for transfer.

V. RESULTS

In order to compare the described algorithms, a simulation model was set up to allow as direct a comparison of results as possible. For the Honeybee Foraging algorithm experiment, the server colony consisted of M proxy server types with N servers (representative of bees). The round robin method is used to control the advert board reading process. The probability (px) that a server reads the advert board (i.e. forages) is initially set to 0.2 whilst the probability (pr) that a successful server writes to the advert board (i.e. performs a waggle dance) is 0.8. An advertisement's lifespan on the advert board is equal to 10 ticks.

For the remaining approach, Biased Random Sampling, the parameters are identical to the previous experiments. Here, the node with the greatest free resources on a walk is preferred; to receive the new job, its resources must be greater than or equal to those of the last node on the random sampling. The results that receive scrutiny in the following section are based on two phases of experiments as described above. The first phase measured throughput against diversity; all experiments ran for N=1000, and an increasing value of M (10 to 800) for each iteration. The second phase measured throughput against available resources; experiments ran for M=10, and increasing values of N for each iteration (100 to 1000).

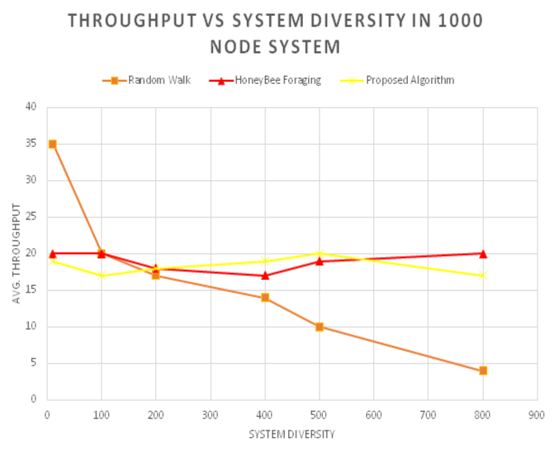

Fig.2 shows the comparative performance on a simulated heterogeneous system, with the x axis showing the effect of increased system diversity on performance: This graph demonstrates that the honeybee algorithm and proposed algorithm performs consistently well as system diversity increases. However, despite performing better with high resources and low diversity, both the random sampling walk degrades as system diversity increases.[5]

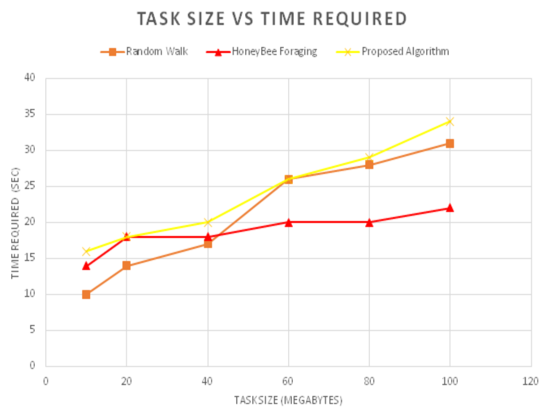

Fig.3 shows the comparative performance on a simulated system where as the load increases shows the effect of increased time required to complete the task:
This graph demonstrates that the Proposed algorithm performs better than Honey bee load balancing and Random sampling.

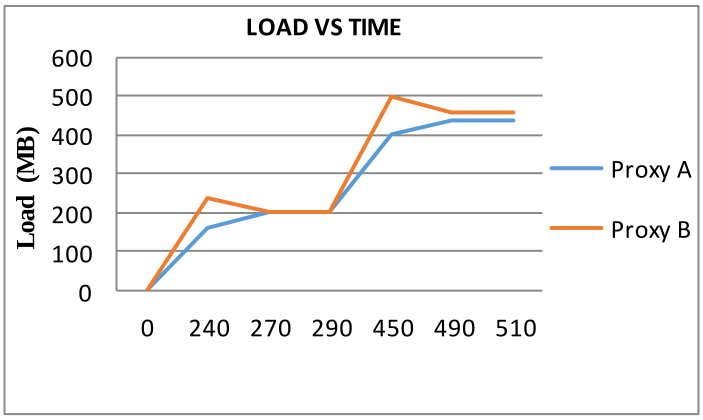

Fig.4 shows results of load balancing between two proxy servers

VI. CONCLUSION

We have shown through experimental results that our task division algorithm functions is the most suitable for balancing the type of load generated by data synchronization application. Thus we have successfully implemented Data synchronization between cloud accounts with divisible load balancing mechanism to efficiently balance the load generated by the application.

REFERENCES

[1] Singhal M., Shivaratri N., 1994, Advanced Concepts In Operating Systems, McGraw Hill

[2] Tanenbaum, A., 1995, Distributed Operating Systems, Prentice Hall

[3] Martin Randles, A. Taleb-Bendiab and David Lamb, Cross Layer
Dynamics in Self-Organising Service Oriented Architectures.
IWSOS, Lecture Notes in Computer Science, 5343, pp. 293-298, Springer, 2008.

[4] Shu-Ching Wang, Kuo-Qin Yan, Wen-Pin Liao and Shun-Sheng Wang ,Towards a Load Balancing in a Three-level Cloud Computing Network

[5] Martin Randles, David Lamb, A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing , 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops