International Journal of Distributed and Parallel Systems (IJDPS) Vol.10, No.2/3, May 2019

ADVANCED DIFFUSION APPROACH TO DYNAMIC LOAD-BALANCING FOR CLOUD STORAGE

Eman Daraghmi[1] and Yousef-Awwad Daraghmi[2]

[1]Department of Applied Computing, Palestine Technical University Kadoori (PTUK), Tulkarm, Palestine
[2]Department of Computer Systems Engineering, Palestine Technical University Kadoori (PTUK), Tulkarm, Palestine

ABSTRACT

Load-balancing techniques have become a critical function in cloud storage systems that consist of complex heterogeneous networks of nodes with different capacities. However, the convergence rate of any load-balancing algorithm as well as its performance deteriorated as the number of nodes in the system, the diameter of the network and the communication overhead increased. Therefore, this paper presents an approach aims at scaling the system out not up - in other words, allowing the system to be expanded by adding more nodes without the need to increase the power of each node while at the same time increasing the overall performance of the system. Also, our proposal aims at improving the performance by not only considering the parameters that will affect the algorithm performance but also simplifying the structure of the network that will execute the algorithm. Our proposal was evaluated through mathematical analysis as well as computer simulations, and it was compared with the centralized approach and the original diffusion technique. Results show that our solution outperforms them in terms of throughput and response time. Finally, we proved that our proposal converges to the state of equilibrium where the loads in all in-domain nodes are the same since each node receives an amount of load proportional to its capacity. Therefore, we conclude that this approach would have an advantage of being fair, simple and no node is privileged.

KEYWORDS

Load balancing, cloud storage, Heterogeneous, Simulation, Task assignment

1. INTRODUCTION

Load-balancing techniques have become a critical function in cloud storage systems that consist of hundreds of independent storage nodes (or nodes for short). In such systems [1], nodes simultaneously serve computing and storage functions where a file is partitioned into a large number of disjointed and fixed-size pieces (or file chunks), and each file chunk is assigned to a different cloud storage node so that the load of a node is typically proportional to the number of file chunks the node possesses. Thus, it is possible to improve the overall performance of the cloud storage system by balancing the load among the distributed nodes. In general, load-balancing algorithms are designed to distribute the loads over multiple nodes in a way that ensures expanding resource utilization, maximizing throughput, minimizing response time, and avoiding the overload situation where one node is heavily loaded with excess of loads while another node is lightly loaded or idle.

Practically, distributed file systems for clouds [2], such as GFS, utilize the centralized approach to simplify the design as well as the implementation of a distributed file system, to manage the metadata information of the systems and to balance the loads of storage nodes based on that metadata. However, when increasing the number of storage nodes, the number of files to be stored and the number of files to be accessed, central nodes become a bottleneck. Additionally, if the central nodes fail, then the whole file system fails as well. As a solution, many studies have proposed a number of dynamic load-balancing algorithms to eliminate the dependence on central

DOI:10.5121/ijdps.2019.10301

nodes by allowing the storage nodes balance their loads spontaneously. The main objective of these previous studies was to propose a better algorithm and to develop a new approach to remedy shortcomings in previous efforts. In fact, previous algorithms are designed to be scalable, portable, easy to use and more improved. Improvements include the derivation of a faster algorithm that transfers less work to achieve a balanced state than other algorithms, and a mechanism for selecting and transferring the loads to other machines in order to improve the algorithm performance. They found that the performance of any load-balancing algorithm as well as its convergence rate deteriorated as the number of nodes in the system, the diameter of the network and the communication-overhead increased. They concluded that increasing the number of nodes in the system, from one hand; make it not feasible for a node to collect the load-information from all nodes in the system and, from the other hand, leads to difficulties in using the collected load-information as most of this information will be out of date which result in lower performance. In other word, the more complex the system is, the less performance achieved. Therefore, our proposed solution attempts to bolster the previous approaches efforts by applying the going-vertical principle for the dynamic diffusion load-balancing technique to consider both the virtual structure of the system and the variables of the load-balancing algorithm. In other words, this principle aims at scaling the system out not up – allowing the system to be expanded by adding more nodes without increasing the system complexity nor the need to increase the power of each node while at the same time increasing the overall performance of the system. The key idea is to simplify the structure of the system by breaking down the entire complex heterogeneous network into simpler domains or clusters of homogeneous nodes based on the property of each node in the system. As a result, the number of nodes, the diameter of the network as well as the communication overhead are decreased and thus the overall performance is increased.

In summary, the objectives of this paper are: (1) to improve the performance of cloud storage systems by applying dynamic load-balancing technique that employs the going on vertical principle; (2) to propose an algorithm that re-balancing tasks to storage nodes by allowing the storage nodes balance their loads spontaneously such that each node obtains load proportional to its capacity and thus achieving the fairness state which in turn eliminates dependence on central nodes. Our proposal was evaluated through computer simulations as well as mathematical analysis, and it was compared with the centralized approach and the original diffusion technique. Results show that our solution outperforms them in terms of throughput and response time. Finally, we proved that our proposal converges to the state of equilibrium where the loads in all in-domain nodes are the same since each node receives an amount of load proportional to its capacity. Therefore, we conclude that this approach would have an advantage of being fair, simple and no node is privileged.

2. RELATED WORK

Presented here is a summary of work related to the approaches and techniques used in this paper [2, 4, 5, 6, 7]. In complex huge systems, it is not feasible for a node to collect the information of all other nodes in the system. Thus, the Going-Vertical principle is a technique aims at converting the complex heterogeneous system, by breaking it out, to several simple clusters of homogeneous nodes. This technique considers both the properties of each node in the system, such as its functionalities and/or capacity, as well as the objective of the system that will execute the load-balancing algorithm to group those nodes who have similar properties into clusters or domains and thus virtually simplifying the structure of the complex system. As mentioned before, load-balancing becomes harder when more loads need to be balanced across a larger system. Moreover, the performance of any load-balancing algorithm (i.e. throughput) as well as the convergence rate of it deteriorated as the number of nodes, the diameter of the network and the communication overhead increased; thus, employing this technique to any load-balancing algorithm aims at scaling the system out not up means allowing the system to be expanded by

adding more nodes while at the same time increasing the performance of the system load-balancing algorithm without the need to increase the power of each node, and maintaining the homogeneity property in-domain. Fig.1 shows the concept of the principle.

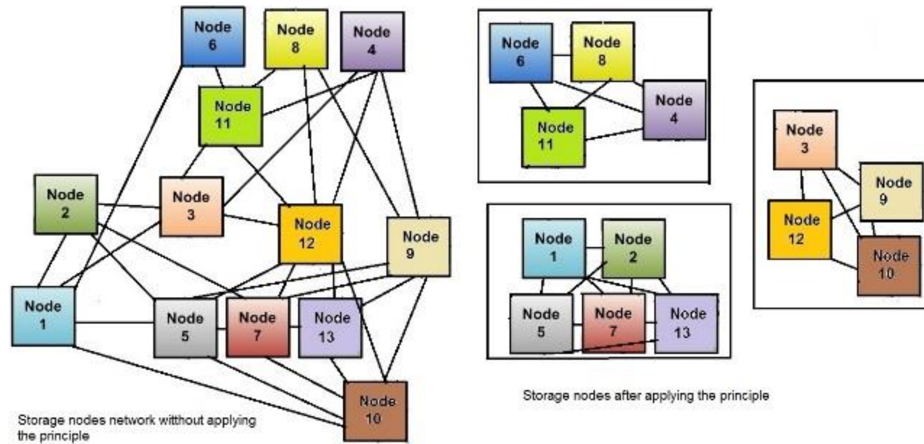

Figure.1. The concept of the going vertical principle

The diffusion approach is a dynamic load balancing technique that allows the nodes to communicate and migrate tasks with other nodes. Each node balances the load among the other nodes in the hope that after a number of iterations the whole system will approach the balanced state. In the diffusion approach, each node simultaneously sends the excessive load to its under-loaded neighbor nodes and receives loads from its neighbor nodes with higher load. Under the synchronous assumption, the diffusion method has been proven to converge in polynomial time for any initial load distribution given the quiescent assumption that no new load is generated and no existing load is completed during execution of the algorithm. Since it is not necessary to have a global coordinator, the diffusion approach is inherently 1oca1, fault tolerant and scalable. Hence this approach is a natural choice for load balancing in a highly dynamic environment [8]. In 1989, Cybenko [9] proposed the first diffusion scheme for dynamic load balancing on a message passing multiprocessor networks. According to his method the load distribution at time t is quantified by a vector $L_i^t = (l_1^t, l_2^t, \ldots, l_n^t)$ where l_i^t is the load of node i at time t ≥ 0. In each round t, node i and node j compare their load and node j sends α tokens to node i if node j has more loads than node i. Cybenko method requires ld(n) steps, where n is the number of processors and ld denotes the logarithm to base 2. The method utilizes the topology of the hypercube machine for its efficiency, but ignores any dependencies between the individual items of data moved. In 1990, Boillat [10] et al. presented an approach to solve the load balancing problem for parallel programs. They presented a fully distributed load balancing algorithm, consisting of the same process running in parallel on each processor of a given network. No assumption has to be made concerning the structure of the underlying network. They show the number of iterations in several cases to be of the form $O(n^2)$ where n is the number of processors. Practically, neighborhood load balancing algorithms are diffusion algorithms that have the advantage that they are very simple and that the vertices do not need any global information to base their balancing decisions on. Another advantage is that balancing with neighbors has the tendency to keep load items initiated by one vertex in the neighborhood of that vertex.

3. LOAD BALANCING PROBLEM FORMULATION

Formally, a large-scale cloud storage file system is modeled as an undirected graph $G = (V, E)$ where V represents the set of chunkservers or nodes and E describes the connections among them. The cardinality of V is $|V| = n$ where n can be one thousand, ten

thousand, or more. The n chunkservers in the system $v_i \in V$ stores a number or a set of files F where any file $f \in F$ is partitioned into a number of disjointed, fixed size chunks denoted by C_f. For example, in Google File System GFS, each chunk has 64 Mbytes [11]. Since each chunkserver i hosts a number of fixed size files chunk, the load L_i of a chunkserver is proportional to the number of chunks hosted by the server. To simulate the worst case, we assume that the chunkservers are heterogeneous in which each server has different capacity. Moreover, the files in F may be arbitrarily created, deleted, and appended. The net effect results in file chunks not being uniformly distributed to the chunkservers. Our objectives in this paper are to design a load-balancing algorithm to reallocate the file chunks such that the chunks can be distributed to the system as uniformly as possible and each chunkserver hosts a number of file chunk proportional to its capacity.

Definition 1 (Going Vertical Principle).

Given a network of heterogeneous chunkservers or nodes $G=(V,E)$ such as each node has its capacity c and with any assigned file-chunks, a principle or a relation R to be found that classifies the nodes based on their capacities into domains or clusters of homogeneous nodes and then the load will be transferred among only nodes in the same domain is the going vertical principle. More formally, the semantic of the relation is defined as follows:

$GV(R) = N \div D = \{t[a_1, \ldots, a_n] : t \in N \text{ and } \forall d \in D((t[a_1, \ldots, a_n] \cup d) \in N)\}$ where $[a_1,\ldots,a_n]$ is the set of attribute names unique such as capacity to N and $t[a_1,\ldots,a_n]$ is the restriction of t to this set. It is usually required that the attribute names in the header of D are a subset of those of N because otherwise the result of the operation will always be empty.

Definition 2 (Dynamic Load-Balancing Problem).

Given a large scale distributed file system $G=(V,E)$ of $|V|=n$ heterogeneous chunkservers and a set of files F such that each file is partitioned into fixed size chunks, the dynamic load balancing problem is to employ the going vertical principle to convert the set of heterogeneous chunkservers into several clusters or domains of homogeneous chunkservers so that the load-balancing algorithm efficiently redistribute the file chunks among the in-domain chunk servers such that if G is stable in a sufficient time period, the file-chunks allocated at each chunkserver in one domain n_i; is fair, that is, $L_1 = L_2 = \ldots = L_n$. Considers that for all $v_i \in V$ when the load in all one domain nodes are equal $L_1 = L_2 = \ldots = L_n$. When this happens, the domain G is said to have achieved local fairness. Obviously achieving the local fairness in all domains means the entire system achieves the fairness state.

4. OUR PROPOSAL

Our proposed algorithm is shown in algorithm1. NeighborLB. Each node n_i in the system G that executes the same algorithm in parallel has a unique node id and a capacity, which defined as the maximum number of file chunk that can the chunk server host. First, the structure of the system is simplified by applying the going-vertical principle; thus, all the chunkservers that have the same capacity form one local-domain or cluster such that chunkservers in one cluster are neighbors and they can exchange their metadata load information and a file chunk can only be migrated to another chunkserver in the domain. As a result, the graph diameter, the number of nodes that will exchange the load information and communication overhead is decreased. In this paper, because of the size limitation, we only focus of the load-balancing algorithm not the management of the domains. Following sub-sections illustrate the proposed algorithm in details.

4.1. Initialization

Each chunkserver v_i hosts a number of fixed size file-chunks $CF_i = \{f_1c_1, f_ic_2, \ldots, f_nc_m\}$. Each chunkserver v_i initializes its state (initialization stage) in steps 1 through step 3. First, by applying the going vertical principle, all nodes who have the same capacity c form one domain. This pre-initialization step means converting the heterogeneous system into several clusters of homogeneous nodes. Step 1: Each chunkserver v_i defines a set info to store its information and the in-domain chunkservers neighbors $\text{info} = \{< v_i, L_i >\}$, where v_i is the node id and L_i is its load i.e. the number of chunks the node hosts. Step 2: Each chunkserver v_i defines an array $to.mig$ to store the amount of the migrated load that node n_i will transfer to its in-domain neighbors. Step 3: Each node v_i computes its initial load $L_i = \sum |\text{CF}_i|$

4.2. Information Broadcasting

Step 4: Each chunkserver v_i then broadcasts its initial state to all its in-domain neighbors. Note that, each node maintains a FIFO message queue which holds the incoming messages. Each message has the format $< v_f, L_f, "T", g >$ where the message came from v_f, its load L_f, T is the type of the message, and g is the migration information. There are three types of messages:

1. Request message (“R”): v_i receives a message to be informed that additional load submitted to it.

2. Load Migration message (“G”): v_i sends a “G”-message to v_j to tell it that v_i wants to migrate g units of load to v_j.

3. Broadcast message (“B”): broadcast the status (i.e. id, load to all in-domain neighbors).

Step 5: The main part of the algorithm starts when the node takes the first message from the queue and processes the message according to its type. Initially, first messages received by each node v_i are “B” type messages.

4.3. Computing the Average In-Domain-Load and Finding In-Domain Assistant Neighbors

Step 6: After receiving the information of all in-domain neighbors, each node n_i computes the average in-domain load in which a node is located. The average in-domain load is defined as $L_{avg} = \frac{\sum_{i \in \text{info}} |\text{Cf}_i|}{|\text{info}|}$. Step 7: According to the set info of node n_i and the average in-domain-load, node n_i defines a set of assistant neighbors N_{lower} whose loads is less than the in-domain average load.

The transferring strategy

Step 8: The decision of calling a procedure LB to migrate the excess files chunks or not depends on the difference between the current load of the node v_i and the average in-domain load. Therefore, the requests will be migrated if the load difference is positive. Hence, we will show later that the local domain will rapidly converge to a state where $L_i - L_{avg} \le 0$ for all edges. The pseudo-code of the procedure Load-Balance is given in Procedure LB.

$Algorithm 1. NeighborLB$
n_i : The node where the algorithm is executed.
$Adj(n_i) = \{< n_i >\}$ The set of in-domain neighbors
$Cf_i = \{f_1c_1, \ldots, f_mc_s\}$ The set of hosted file chunks by node i

Begin

1.Let info $= \{< n_i, L_i >\}$

2. Let mig(n_j)=0 for all $n_j \in Adj(n_i)$

3.Compute the initial Load: $L_i = \sum |Cf_i|$

4.For each node $n_j \in Adj(n_i)$ do

send message<n_i, L_i,"B",0>

5.Read messages from the messages queue

a. if T="B" then info= info $\cup \{< n_f, L_f >\}$

b. if T="G" then

info= info $\cup \{< n_i, L_i + g >, < n_f, L_f - g >\}$

For each node $n_j \in Adj(n_i)$ do

send message<n_i, L_i+ g,"B",0>

6.Compute the average in-domain-load $L_{avg} = \dfrac{\sum_{j \in info} L_j}{|Adj(n_i)|}$

7.Define the set of Assistant Neighbors

For each node $n_j \in Adj(n_i)$ do

if $L_j < L_{avg}$ then $N_{lower} = N_{lower} \cup n_j$

8.Let load-difference $LD = (L_i - L_{avg})$

9.If $LD \leq 0$ then exit;

else

$LB(Cf_i, N_{lower}, LD)$

EndBegin

$Procedure$ $LB(Cf_i, LD_i, N_{lower})$

Begin

1. Sort the nodes in N_{lower} in acsending order
2. for nodes in N_{lower}

compute $\delta = L_{avg} - L_j$

if $\delta > LD_i$ then send message <n_i,L_i,"G",LD_i>

else

if $\delta < LD_i$ then send message <n_i,L_i,"G",δ>

$LD_i = LD_i - \delta$

Endfor

EndBegin

4.4. Load-Balancing Mechanism

In the procedure LB, the load difference LD, the set of in-domain assistant neighbors and the set of the hosted file chunks are formed the procedure input parameters. In this step all the over-loaded nodes call the procedure LB to migrate the excess file chunks to the under-loaded nodes. Each overloaded node sorts the set of assistant neighbors in ascending order. The file chunks that will be migrated from the overloaded node is the load difference between the load of the overloaded node and the average in-domain load or the difference between the average load and the assistant neighbor. In addition, this amount spread to the assistant neighbors ensures the node who will receive the file chunk maintains the under-loaded status. Figure.4 shows the load-balancing mechanism.

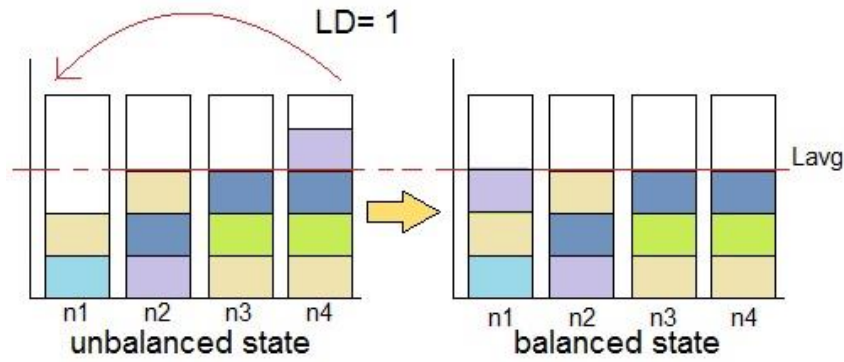

Figure. 2. Load-balancing example

5. ALGORITHM ANALYSIS

5.1. Time Complexity Analysis

Most of the operations in the proposed algorithm take $O(1)$ time. Since the broadcasting operations have time complexities $O(d_i)$, the idea of our approach is to improve the performance of load-balancing algorithm by considering both the algorithm parameter and the virtual structure of the system that will execute that algorithm and thus the number of in-domain neighbors $d_i = |Adj(n_i)|$ of each node is decreased. That has benefits in reducing the communication delay and the amount of out of date information. Also, if "info" and "to.mig" objects are implemented as arrays, then steps 1 and 2 have time complexities $O(d_i)$ and thus each individual update takes $O(1)$ time. Moreover, the sorting steps for the LB procedure has the worst case time complexity of $O(c \log c)$ where c is the number of in-domain assistant neighbor, suppose that merge sort is used. The for-loop only takes $O(c)$ since each entry in sorted is referenced only once. So NeighborLB algorithm runs only at $O(c \log c)$ time.

5.2. The Convergence

In this section we prove that the NeighborLB algorithm converges to the state of fairness given sufficient time.

Lemma 1. Given $L^t = (l_1^t, l_2^t, \ldots, l_n^t)$ is the loads array of the in-domain nodes at time t where l_1^t is load of node 1 at time t. In time t, if there is at least one overloaded node (i.e. $LD > 0$) then L^{t+1} is lexicographically greater than L^t means the lightly-loaded nodes at time t will receive load at time t+1.

Proof:

Let $X \neq \phi$ be the set of overloaded nodes in domain (i.e. nodes with $LD > 0$) who needs to migrate some loads to other nodes at time t. In reality, a node $i \in X$ will also host additional loads in time t. Thus, the nodes that migrate loads in time t will reduce their load at time t+1. Let γ be the node that has lowest load at time t+1. Assume that γ occupies the kth position of the array L^{t+1} where L^{t+1} similarly is the load array of in-domain nodes at time t+1 sorting in ascending order. Let

$Q_t = (l_1^t, l_2^t, \ldots, l_{k-1}^t)$ be the array of the loads in first k-1 positions of L^t. In order to prove this lemma we have to consider two cases: A set Q_{t+1} contains a node i that received loads in time t. Thus, node i belongs to both Q_t and Q_{t+1}, and its load value is increased in time $t+1$ since it will received some migrated load. Therefore, there will be at least one load value in set Q_{t+1} strictly greater than one value in Q_t. Accordingly, L^{t+1} is lexicographically greater than L^t.There are nodes in which located in Q_{t+1} and did not migrate or receive loads at time t. In this case, the load value at k^{th} position at time $t+1$ is strictly greater than the load value in the same position at time t which has received load from X^t and therefore, L^{t+1} is lexicographically greater than L^t.

Theorem 1 Convergence. In heterogeneous system, if each domain of nodes executes the NeighborLB algorithm, then the system converges to a balanced state.

Proof: Given a heterogeneous unbalanced system. Assume that the going-vertical principle, first, applied to this system to spread the nodes into several domains and some of these domains are unbalanced. Each domain separately execute the NeighborLB algorithm in order to achieve the convergence state. Consider one domain N. Let i be the most heavily loaded node in that domain, i.e. $l_i - l_{avg} > 0$, and all other nodes in-domain $j \in N$ who have $l_j \leq l_i$ and $l_j < l_{avg}$ form the set of assistant neighbors of node i. When $l_i \geq l_{avg}$ and thus $l_i - l_{avg} > 0$. Thus, the result of Lemma 3 shall be used, which guarantees that the array of loads sorted in ascending order, in the next time moment, is lexicographically greater than the array of the current step. Given that the NeighborLB algorithm is executed in some domains in a given time t. Let $S \subseteq N$ be the domain of nodes executed the algorithm in time t. Let L^t be the array of loads in one domain sorted in ascending order in time t. It has been proven in lemma 1 that L^{t+1} is lexicographically greater than L^t. Let $S_{\min}$ be the lightly loaded node in time t. There exists at least one node $v \in S_{\min}$ which is in-domain neighbor to node k such that $l_k^t > l_v^t$. Now by using the proposed algorithm, node k migrates a portion of its excess load to node v, but v does not migrate any loads in time t because v is under loaded when compared to the average load of the domain. In time $t+1$ the effective-load of node k decreases; however, its load value never become less than the load value of node v which is given by $l_k^{t+1} \geq l_{avg} \geq l_v^t$. Thus, L^{t+1} is lexicographically greater than L^t meaning that the sorted array of load values of nodes in time $t+1$ are lexicographically greater than the sorted array of the load values of nodes at time t.

6. SIMULATIONS

The performance of our proposed algorithm was examined through a computer simulation that was implemented with CloudSim [12,13] and was compared to the original diffusion neighborhood method and the centralized approach. Initially, the test of our proposed method was based on two parameters: the number of chunk files and the number of storage nodes. The measurement of the performance of the proposed algorithm was based on two metrics: throughput and the response time. Only one parameter was changed each time so that any changes in the performance would be based solely on this parameter. Therefore, two tests were produced for each parameter to allow a rough average and standard deviation to be obtained. In fact, results achieved from these tests were used to study: (1) the behavior of different load-balancing algorithms under the same condition; (2) the behavior of the algorithms for random systems with

different number of storage nodes; (3) the behavior of the algorithms for different load distributions.

6.1. Changing the Number of File Chunks

To study the effects of changing the number of file chunks on the average response time and the throughput, the number of file chunks was varied from 1000-10,000 chunks and the distribution of the chunks among the storage nodes were carried in the following manner.

- The initial distributions varying 25% from the in-domain average load to represent a situation where all nodes have similar loads at the beginning and those loads are close to the in-domain average load; in other word, the initial situation is quite balanced.
- The initial load distributions varying 50% from the average load to constitute the intermediate situations.
- The initial load distributions varying 75% from the average load to constitute the higher intermediate situations.
- The initial load distributions varying 100% from the average load to constitute the advanced unbalanced situations.

6.1.1. Average Response Time

The total time taken for the three algorithms increased as the number of file chunks was increased as shown in Figure.3. This is expected as the more files to be stored, the longer it takes to complete the storing tasks. However, it was observed that our proposed method performed better than the centralized scheme and the original diffusion algorithm. In addition, when comparing the results of the method and the centralized algorithm, it is observed that the gap between these two curves was widening as the assigned loads was increased. This shows that the method actually reduced the completion time by a considerable amount (greater speedup) in comparison to the centralized algorithm as amount of loads increased.

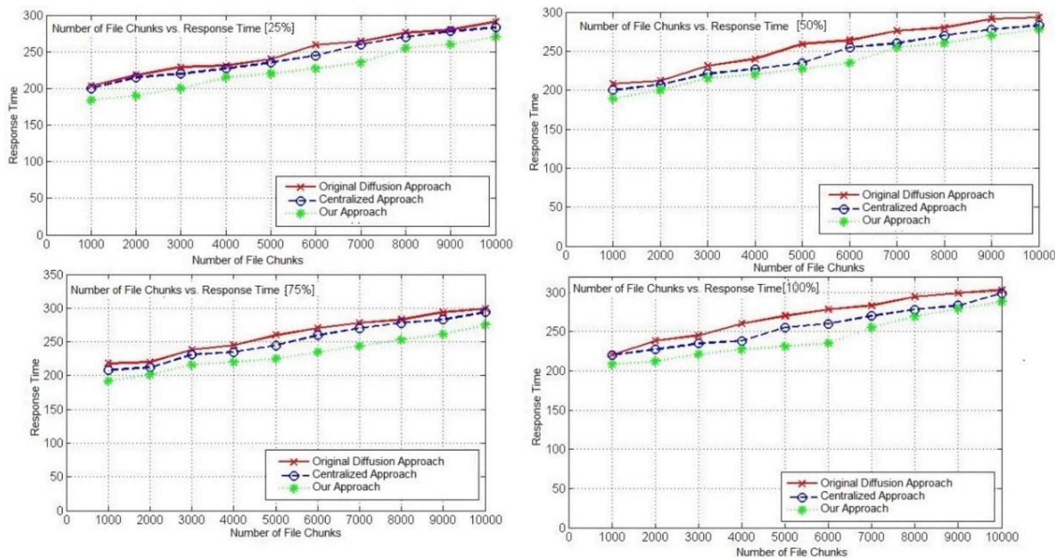

Figure. 3. Number of File Chunks vs. Response Time

6.1.2. Throughput

As shown in Figure.4, our method outperformed the original diffusion neighborhood algorithm in terms of the system throughput in all loads distribution cases. The throughput using our method was in the range of 89-98 percent while the nearest neighborhood algorithm only had a utilization of 80-94 percent.

6.2. Varying the Number of Storage Nodes

To study the effects of changing the number of storage nodes on the average response time and the throughputs, the number of nodes were varied from 10– 100 nodes and the distribution of the overloaded nodes were carried in the following manner.

- 25% of storage nodes are idle, 75% of storage nodes are overloaded.
- 5o% of storage nodes are idle, 50% of storage nodes are overloaded.
- 75% of storage nodes are idle, 25% of storage nodes are overloaded.

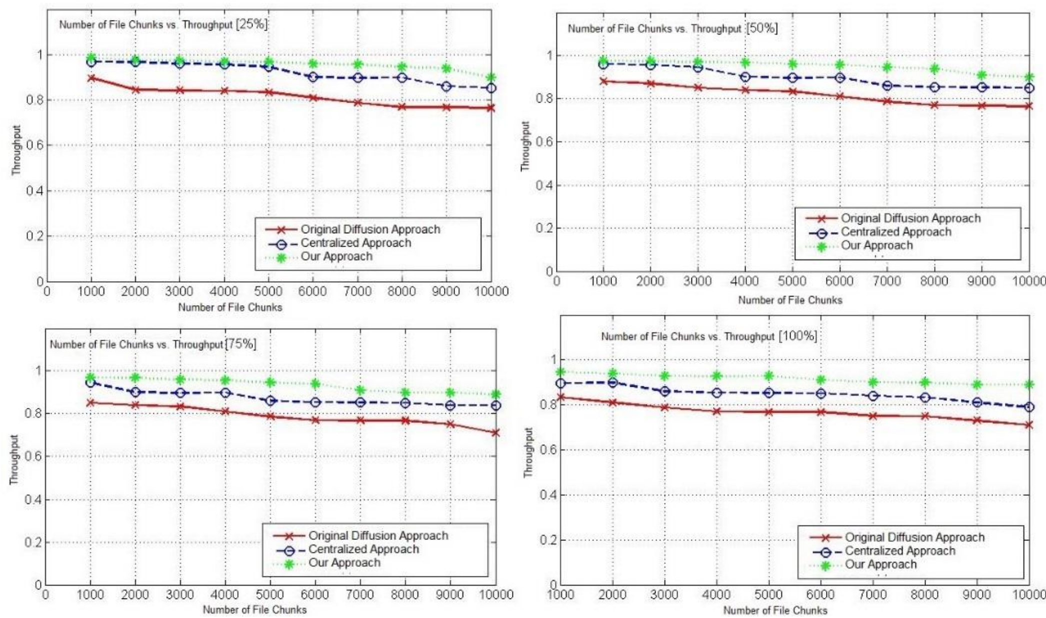

Figure. 4. Number of File Chunks vs. Throughput

6.2.1 Response Time

Figure. 4 shows that the response time improved when the number of nodes was increased. However, this improvement was mainly caused by the fact that more nodes were used for larger domain. Therefore, even though there were more loads to be scheduled in each round, the extra load was easily handled by the additional nodes.

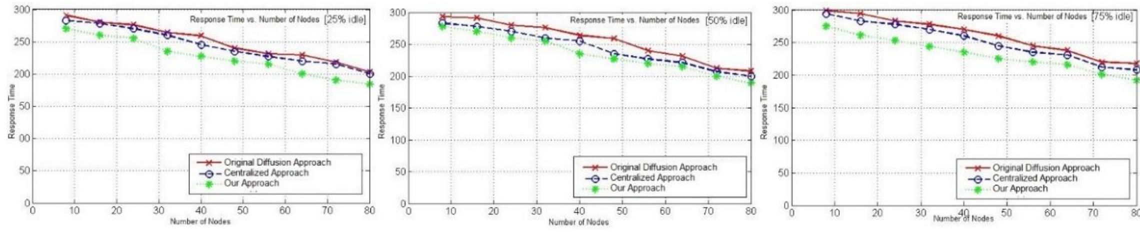

Figure. 5. Number of nodes vs. Response time

6.2.2. Throughput

Figure. 5 shows that the throughput in the original neighborhood algorithm decreased as the number of nodes in the system increased. However, in our proposed method, the number of nodes is divided into several domains keeping the number of node in a domain reasonable. This shows that load-balancing is harder when more tasks are to be balanced out across a larger system.

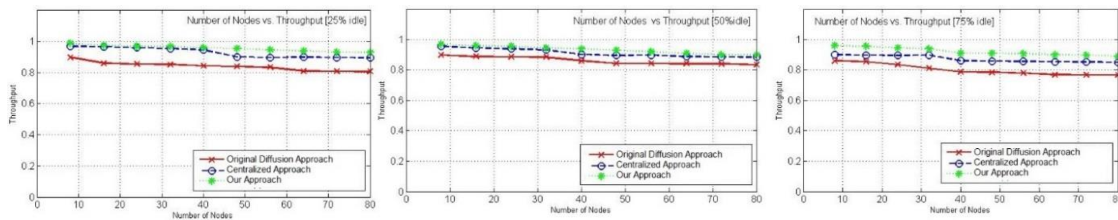

Figure. 6. Number of Nodes vs. Throughput

6.3. Discussion

This section summarized the performance of the proposed solution as compared to the original diffusion method and the centralized scheme (see table 1 below). Each of the test used the response time and the throughput as performance measures. The performance of these methods was compared in many cases by changing the parameters of the algorithm. The parameters varied (one at a time) were the assigned loads to be executed and the number of nodes. It was observed that our proposed method performed better than the other approaches. The number of nodes, the network diameter and the communication delay affect the convergence rate of any load-balancing algorithm as well as its performance. It is intuitive that a graph or a system with longer diameter will take longer time to converge as the number of iterations to propagate the loads to all assistant neighbors is proportional to the network diameter. In addition, more communication delays lead to out of date information. Our proposed method considers both the structure of the network that will execute the algorithm and the algorithm parameters. It works, first, by simplifying the structure of the system which in turn decreases, from one hand, the communication overhead between the in-domain neighbors which lead to faster response time and, from the other hand, the time need to choose the assistant neighbors and the target node that will receive the migrated loads. This effect appears clearly when the assigned loads and the number of nodes increased. Moreover, reducing the communication delay improves the load evaluation since the effect of out of date information will be decreased. Also, considering the processing speed and thus the processing capacity of each node leads to more accurate average load evaluation which improves the algorithm performance. The importance of the average load appears when deciding the amount of loads to be migrated; if the migrated loads to one node is too small, then the load distribution will take longer and so the convergence rate. In contrast, if the migrated loads to one node is too large, then the overloaded node may transfer too much load to its neighbor and thus this overloaded node will not have sufficient load to transfer to the remaining neighbors. Thus, by using the in-domain average load, each node obtains an amount of load proportional to its

capacity and thus no node is privileged. This indicates reliable performance of the method when the assigned loads increases that is very valuable from a practical point of view.

Table 1. A comparison between our proposed solution and both the centralized approach as well as the diffusion method

	Centralized	**Original Diffusion**	**Our Method (diffusion + applying the GV principle)**
Pros	Simple design, simple implementation, good performance.	Good performance, solve the bottleneck and the failure problems in centralized approach.	Good performance even with large systems, simplify the network structure, solve the cons of the original diffusion method and the centralized approach.
Cons	Bottleneck problem, failure possibility	Communication overhead, low convergence rate and low performance for large heterogeneous systems.	Need a good skills to define the properties for each node.

7. CONCLUSION

This paper considered load-balancing mechanism in cloud storage systems. As the convergence rate of any load-balancing algorithm as well as its performance deteriorated as the number of nodes in the system, the diameter of the network and the communication overhead increased, our proposal that employed the going-vertical principle has been very effective especially in the case of a large number of nodes and dense loads. In fact, a going-vertical based scheme works better when the number of nodes is large since the key idea of the proposed method is that the communication occurs between only the in-domain node reduces the impact of communication delay on freshness of the load information which in turn allows the method to handle all load-balancing information and thus all load-balancing decisions with minimal inter node communication. In other words, we aimed at not only considering the parameters that will affect the algorithm performance but also simplifying the structure of the network that will execute the algorithm. Finally, we proved that the proposed algorithm under this approach converge to the state of equilibrium where the load in all nodes is the same since each node receives an amount of load proportional to its capacity. Therefore, we conclude that this approach would have an advantage of being fair, simple and no node is privileged.

REFERENCES

[1] H.-C. Hsiao, H.-Y. Chung, H. Shen and Y.-C. Chao, "Load Rebalancing for Distributed File Systems in Clouds," IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 5, pp. 951-962, 2013.

[2] E. Y. Daraghmi and S. M. Yuan, "In-domain neighborhood approach to heterogeneous dynamic load balancing in real world network," in 14'th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'13), Taipei, Taiwn, 2013.

[3] C. P. A. a. P. Berenbrink., "Distributed selfish load balancing with weights and speeds.," in The 2012 ACM symposium on Principles of distributed computing, New York, USA, 2012.

[4] J. Bahi, R. Couturier and F. Vernier, "Synchronous Distributed Load Balancing on Totally Dynamic Networks," in Parallel and Distributed Processing Symposium, 2007.

[5] E. Luque, A. Ripoll and A. C. a. T. Margalef, "A distributed diffusion method for dynamic load balancing on parallel computers," in Euromicro Workshop on Parallel and Distributed Processing, 1995.

[6] P.Neelakantan, "Decentralized Load Balancing In Heterogeneous Systems Using Diffusion Approach," International Journal of Distributed and Parallel systems (IJDPS), vol. 3, no. 1, pp. 229 - 239, 2012.

[7] C.-C. Hui and S. Chanson, "A hydro-dynamic approach to heterogeneous dynamic load balancing in a network of computers," in Proceedings of the 1996 International Conference on Parallel Processing Software., 1996.

[8] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," Journal of Parallel and Distributed Computing, vol. 7, no. 2, pp. 279-301, 1989.

[9] J. E. Boillat, "Load balancing and Poisson equation in a graph," Concurrency: Practice and Experience, vol. 2, no. 4, pp. 289-313, 1990.

[10] "Google File System," [Online]. Available: http://en.wikipedia.org/wiki/Google_File_System.

[11] R. N. Calheiros, R. Ranjan, A. Beloglazo, C. A. F. D. Rose and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud," SOFTWARE – PRACTICE AND EXPERIENCE, vol. 41, no. 1, pp. 23-50, 2010.

[12] R. M. I. Stoica, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan, "Chord: a Scalable Peer-to-Peer Lookup Protocol for Internet Applications," in Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols, San Diego, California, USA, 2001.