Performance-Driven Load Balancing for Distributed File Systems in Clouds

Jasma Balasangameshwara
Associate Professor
Department of Computer Technology
School of Engineering
Dayananda Sagar University
Kudlu Gate, Hosur Main Road
Bengaluru, Karnataka, India

Chandrakala H. L.
Associate Professor,
Department of Computer Science & Engineering
HKBK College of Engineering
Manyata Tech Park, Nagwara
Bengaluru, Karnataka, India

ABSTRACT
Distributed file systems are the fundamental units for cloud applications where in the data node concurrently serves the computing and storage functions. In these file systems, a file is split by a master node into a set of file chunks and allotted to separate data nodes such that various jobs can be carried out in parallel across the data nodes. However, the unpredictability of the nodes and dynamism in the number of files raise the need for uniform re-distribution of files to prevent the adverse effects of load imbalance.

Hence, the latest enhancement to distributed file systems is a decentralized and asynchronous load rebalancing algorithm that exploits both heterogeneity and movement cost for file chunk allocation among data nodes. But, the load rebalancing protocol has its basis in a randomized method wherein the data node periodically collects and sorts the storage load status of an instance of arbitrary chosen data nodes without considering their computational capabilities or the physical proximity information thereby introducing not only considerable workload on the data nodes but also high overhead on message exchanges among data nodes thus leading to reducing scalability. Moreover, the distributed load re-balancing approach does not consider the additional redundant overhead on the data nodes from the federated, load imbalanced master nodes.

In the current study, a completely distributed **p**erformance-**d**riven **l**oad **b**alancing approach (PDLB) that employs Zero-Hop Hash Table (ZHT) and Modified Firefly Algorithm (MFA) is suggested for coping with the load imbalance issue on both master node and data node. The aim of PDLB is to arrive at data allocations among nodes that could achieve maximum resource utilization at optimized movement cost and minimized message exchanges and algorithmic overhead. The experimental results indicate that PDLB performs better than the earlier distributed protocol about overhead on message exchanges, scalability, movement cost, load imbalance factors as well as algorithmic overheads.

General Terms
Cloud Computing, Load Balancing, Resource Utilization, Distributed File Systems

Keywords
Map Reduce, Hadoop Distributed File System, Load Balancing, Zero-Hop Hash Table, Firefly Algorithm

1. INTRODUCTION
Cloud is an architecture which offers resources and services across the Internet. Storage clouds offer storage services, while data clouds provide data managing services and computing clouds provide computational services. Mostly these are layered for creating stacks of cloud services which serves as computing platforms to develop cloud-based application [1].

Distributed file systems are the fundamental units in cloud computing applications. A file in a distributed file system is split into a set of file chunks and allotted to separate data nodes such that various jobs such as chunk creation, deletion, replication and MapReduce jobs may be carried out in parallel among the data nodes [5].

But, due to the unpredictability of the nodes and dynamism in the data and number of files, there is a need for uniform re-distribution of data in distributed file systems to prevent the adverse effects of load imbalance. Hence, resource utilization of the node and load balancing among nodes are critical function for distributed file systems in a cloud.

The benefits of using cloud computing for distributed file systems as illustrated by Grossman et al [1] are; first the data in a storage cloud can be easily replicated and second once the information is stored in clouds, it can wait for the computing jobs [2]. The key enabling technologies for cloud based distributed file systems include MapReduce programming paradigm, distributed file systems virtualization among others [3]. These methods have an emphasis on scalability and comprises of resources that may randomly fail and join when maintaining system dependability.

1.1 Background
Current best distributed file systems in clouds are the Google GFS [4] and Hadoop HDFS [5]. Applications which function on these file systems have huge datasets and a generic file is gigabytes to terabytes in size. Hence these file systems ought to offer high aggregate data bandwidth [5].

HDFS provides interfaces for applications for moving themselves close to where data is positioned as it is efficient if computations demanded by applications is implemented nearer the information it functions on. This approach adopted by HDFS decreases network congestion while increasing the total throughput of the model [5].

However, “Google GFS” [4] and “Hadoop HDFS” [5] rely on a “centralized master node” for balancing the load of its data nodes. They distribute the file chunks to data nodes in a uniform manner so that MapReduce jobs may be executed in parallel. They also employ a standalone master node that gathers information on the file chunk locations and migrates excessive file chunks from data nodes. Due to this approach, the master node is under a great amount of workload which linearly scales with system size thereby becoming a

performance bottleneck as well as the one point of failure.

Also, as the centralized method does not regard the cost of load migrations [6], [7] it may be unsuccessful in Google GFS [4] and Hadoop HDFS [5] wherein the data nodes concurrently serve the computing and storage functions.

Google GFS [4] and Hadoop HDFS [5] employ periodic load information exchange policy. In this policy, the load status information is disseminated or collected at regular intervals. Even though this approach is simple, it is essential to determine the most appropriate dissemination period as overheads due to periodic communication increases system loads and reduces scalability. In periodic information exchange approach, a fixed quantity of state-collection overheads is induced in the model regardless of whether the information will be used. Too frequent updates induces high communication overhead and too few updates makes the system state outdated [14].

Distributed hash tables (DHT) are the important building blocks for large scale distributed systems [15] like Google GFS and Hadoop HDFS. ZHT refers to a zero-hop DHT [16]. It focuses on being a basic unit for future high-end computing models with the aim of providing availability, error tolerance, excellent throughputs, scalability, persistence as well as less latency. ZHT also possesses various significant attribute ensuring that it is a better than other DHTs as well as key-value stores. They are light weight, permitting data to join as well as leave in a dynamic fashion, effectively propagating events through the system, scalable and support for operations like offering lock-free concurrent keys/value alterations additional to inserts/lookups/removals [16].

1.2 Motivation

There has been a great amount of effort recently in the development of load balancing protocols for distributed files systems for realizing greater performance in cloud environments. However, practical samples of these issues are regarded as NP-complete [1]. Therefore, the current study's motivation lies in the requirement of excellent methods which consider cloud infrastructure, resource heterogeneity and volatility, communications overhead as well as movement cost. The primary aim is the reaching of solutions which have least make-span, effective network traffic as well as resource usage, a well-balanced load, excellent data center dependability as well as flexibility at the time of change to the files and nodes.

There is an inherent trade-off to arrive at the solution. For overload avoidance, the utilization of data nodes should be reasonably low in order to avoid possible overloads in the event that the resource requires increases at a later stage. For optimized resource utilization, the utilization of data nodes should be reasonable high to enhance the overall throughput of the system.

It is difficult to propose a load balancing strategy for huge-scale, dynamic as well as data-intensive clouds. This obstacle is the motivation to expand the load rebalancing technique suggested in [3] and combine it with the dynamic resource allocation approach [14]. Problems like metadata management, replication strategies and file consistency systems are not within the scope of the current work.

1.3 Contribution

The contributions of the current study are several. A load balancing method has been proposed for distributed file systems in data-intensive cloud computing environment called PDLB. PDLB employs Zero-Hop Hash Table (ZHT) and Modified Firefly Algorithm (MFA) for arriving at load balancing decisions. Given that cloud architectures are dynamic with differing network topologies; we consider the dynamic network topology through the generation of topologies with nodes of differing capabilities and differing bandwidth between the connecting links by considering their physical network proximity.

As far as is known, PDLB is the first that tackles all that which has been mentioned below:

1. A resource allocation approach using MFA that can obviate overloads in the system (master node and data node) efficiently while reducing the cost of message exchanges among/ between master/data nodes and algorithmic overhead.
2. PDLB considers distributed cloud infrastructure, resources as well as network heterogeneity, movement cost as well as resource unpredictability by configuring the system as a ZHT network.

The primary contributions of the current work to the previous literature are given below:

1. Mutual information feedback strategy as well as neighbor selection strategy suggested in [14] are improved even more.
2. Using ZHT for estimation of the presence as well as efficacy of nodes through the simplification of model of decision making.
3. **M**odified **F**irefly Optimization **A**lgorithm (MFA) is integrated to PDLB for arriving at optimal load allocation paths.
4. Analytical models are derived for validating the efficacy of PDLB.

2. RELATED WORK

State-of art distributed file systems in clouds depend on master nodes for managing metadata of the file systems as well as for balancing loads of data nodes based on the metadata [3]. However, as the quantity of data nodes and files rises linearly, the master node becomes a performance bottleneck.

Hence, the latest HDFS enhancement [23] was a distributed approach with multiple master nodes. However, HDFS statically assigns file system workloads to the master node and does not support adaptive load migration mechanism among master nodes. Also, the master nodes are independent of each other and the data nodes have to periodically send heartbeat messages to all master nodes in the cluster.

Hsiao et al [3] studied the load rebalancing issue in distributed file systems to huge-scale, dynamic as well as data-intensive clouds. The aim was the allocation of chunks of files in a uniform manner amongst the data nodes so that no data nodes oversees management of excess quantity of chunks. Additionally, the movement cost, resource heterogeneity and physical network locality was also taken into consideration for addressing the load imbalance issue among data nodes.

However, Hsiao assumed the following:

1. Randomized load re-balancing approach.
2. Each data node executes gossip-based aggregation algorithm for collecting storage load status of an instance of arbitrarily chosen data nodes.
3. There is one bottleneck resource i.e. storage for optimization.

4. Data nodes are arranged as a network based on DHTs.
5. Load re-balancing approach is not dependent on the DHT algorithms.
6. Re-balancing approach greatly relies on the data node arrival as well as departure operations for migrating file chunks amongst data nodes.
7. Periodic shifting of load among light-loaded data nodes
8. Pairing of under-loaded data nodes with multiple over-loaded data nodes
9. Load management of the master node is not considered

The assumptions in the load re-balancing approach [3] for distributed file systems leads to the following limitations:

1. The randomized load re-balancing approach only considers the storage capacity of the data node and does not take into consideration the data node's overall capability which practically ought to be a function of computation power storage space and network bandwidth [17], [18]. This is critical in distributed file systems as the data nodes concurrently serve both the computing and storage functions.
2. Periodic piggybacking of load status leads to accumulation of information regardless of whether the information will be used. Also, too frequent updates induce high communication overhead which inhibits system scalability and too few updates make the system state outdated causing adverse effects on the load balancing approach. Scalability and responsiveness are critical functions in huge-scale distributed file systems like Google GFS [4] as well as Hadoop HDFS [5].
3. Each data node executes gossip-based aggregation algorithm for collecting storage load status of an instance of arbitrarily chosen data nodes. The implication of small bounded message size as well as relatively slow periodic message exchange join to restrain the data carriage capability of the gossip-based aggregation protocol.
4. Each data node in the load re-balancing approach executes a DHT algorithm like Chord [20], Pastry [21] or Amazon's Dynamo [22]. DHT's Chord and Pastry scale logarithmically with the system scales. Amazon's Dynamo is a key value storage model which certain Amazon core services utilize for providing an 'always-on' experience. Dynamo is a zero-hop DHT. An important shortcoming of Dynamo is that it is an internal Amazon project that is not capable of being utilized outside Amazon's architecture. Also, the routing time for Dynamo is log (N) and it does not support append operation.
5. The Re-balancing approach greatly relies on the data node arrival as well as departure operations for migrating file chunks amongst data nodes and ignores the dynamism in the file chunks. Periodic shifting of load among light-loaded data nodes and pairing of under-loaded data node with multiple over-loaded data nodes adds additional algorithmic overhead.
6. As the master nodes in HDFS federation [23] are federated with no co-ordination among themselves, the data nodes have to periodically send heartbeat messages to all master nodes in the cluster. This approach lacks adaptive load migration among master nodes and adds additional messaging and movement overhead on the data nodes. Also, the load balancers in HDFS do not distinguish between various locations of remote clusters when balancing the load.

Hence, the objective of PDLB is to arrive at allocations as uniformly as possible among nodes by not only exploiting resource heterogeneity in terms of storage, compute and physical network proximity but also take into consideration the load transfer overhead, algorithmic overhead and movement cost so that no node must manage excess amount of data.

To achieve the objective, the nodes in PDLB are structured as a ZHT network and every node executes a ZHT algorithm. ZHT's has all the advantages of DHT's which are availability, error tolerance but simultaneously achieves the advantages of least latency typically related to idle central indexes [16] which is very essential for state-of-art distributed file systems.

Modified Firefly Algorithm (MFA) [35] has been integrated to PDLB for arriving at optimal load allocation paths for each node. It has been shown in [36] that MFA is better than Particle Swarm Optimization (PSO) in its applications.

The remaining sections are structured as follows: Section Remainder of this work is organized as follows: Section 3 reviews the load balancing problem. The overview of the system model is given in section 4. Section 5 discusses the proposed algorithm. Section 6 offers a performance comparison of PDLB as well as DLRA. Section 7 details the setup of the simulation and Section 8 gives the results of the experiments. Finally, section 9 concludes the work.

3. LOAD BALANCING PROBLEM

The aim in the current work is the designing of a load balancing protocol to re-allocate data in a uniform manner amongst nodes of a huge scale distributed file system by exploiting resource heterogeneity, physical network proximity, movement cost and communication cost [5], [23].

Load balancing in PDLB is dynamically and independently performed at three levels – the primary master node level, secondary master node level and data node level. Load balancing at the primary/secondary master node levels balances the storage at the file pool level. Load balancing at the data node level balances the storage of data nodes.

Note that with multiple master nodes, different categories of applications and users can be isolated to different namespaces [23]. PDLB employs this isolation strategy given by HDFS Federation at two levels namely the primary master node level and secondary master node level and allows only those primary/secondary master nodes that share the same category of application or user; to adaptively migrate the storage load among themselves.

Like HDFS Federation [23], PDLB balances only the data among the primary/secondary master nodes, taking into consideration the respective primary/secondary master node's namespace overhead but does not balance the namespace of the primary/secondary master nodes.

Let Υ be the ideal amount of data (i.e. number of files on primary/secondary master node or number of chunks on data

node) that a node i is needed to manage in a system-wide load balanced state, which is [3]

$$T_i = y_i \cdot P_i \quad (1)$$

Where "y_i is the storage load per unit capacity a node i ought to manage in the load balanced state and P_i is the storage capacity of the node" [3].

$$y_i = m/P_k \quad (2)$$

Wherein "*m* refers to the quantity of files (if *k* is a primary/secondary master node) or number of chunks (if *k* is a data node) stored in the rack. Hence, the basis of this design is to make sure that the amount of data managed by node *i* is proportional to its storage capacity".

PDLB's aim is to decrease the load imbalance factor in every node *i* as

$$|| L_i - T_i || \quad (3)$$

wherein L_i represents the current storage load of node *i* and ||.|| denotes the absolute value function [3]. It is to be noted that node collectively refers to primary/secondary master node and data node.

The movement cost $M_{c(i,j)}$ between node *i* and node *j* is computed thus:

$$M_{c(i,j)} = m/min(B_w, A_j) \quad (4)$$

Where B_w is network bandwidth between node *i* as well as *j* and A_j is the available I/O speed at node *j*. A similar Eq. can be found in [33].

In PDLB, it is possible that a set of distinct nodes intend to share the load of node *i* with the aim of minimizing the load imbalance factor as given in Eq. (3). Thus, *i* gives first preference to nodes belonging to same rack with minimum M_c. If a node in a cluster is unbalanced and if it cannot find another candidate node in the same cluster, then the node will look up for another node in another remote geographically close cluster with minimum M_c.

4. SYSTEM OVERVIEW

In this section, the original HDFS Federation structure is reformulated with a two-layer master node architecture as illustrated in figure 1. The extended master node structure given by Xiayu Hua et al. [33] is used for setting the architecture of PDLB. However, PDLB does not use in its architecture the cache support specified by Xiayu Hua et al.

4.1 Architecture

A huge-scale distributed file system comprising a set of data nodes *DN*, a set of primary master nodes *MN* and a set of secondary master nodes *SN* in a cloud is considered, where the cardinality of *DN*, *MN* and *SN* are *d*, *m* and *s* respectively. The *SNs* are applied to every rack of the *DNs* [33]. An *SN* is configured from an existing *DN* machine on each rack [33]. A scoring system is used by considering the storage space and computational capacity of *DNs* for selecting *DNs* eligible for acting as *SNs*. A *DN* with highest score is designated *SN* for a particular rack. To an *MN*, each *SN* is its *DNs* [33]. To a *DN*, *SN* in its rack is their *MN* [33]. The modifications needed in HDFS Federation structure for incorporating the structural extension are minimal and is taken from [33].

The secondary master nodes and primary master nodes are grouped into clusters (as given in HDFS Federation [23]) based on their physical network proximity. Each secondary master node registers with all primary master nodes in the cluster and the secondary master nodes are used as common storage by all primary master nodes of the cluster [23]. It is assumed that the data nodes can be arbitrarily upgraded, substituted and appended in the system.

A single primary master node offers no isolation in multi-user environments. For example, an experimental application can overload the primary master node and slow down production critical applications. Hence, with several primary master nodes, different groups of applications as well as users may be separated to different namespaces [23]. PDLB employs this isolation strategy given by HDFS Federation and allows only those primary master nodes that share the same category of application or user; to adaptively co-ordinate among one other.

In the system, a set of files are stashed in the *m* primary master nodes. The group of files is denoted as *F*. The files in *F* can be randomly generated, discarded or added. All files $f_i \in F$ are split into a set of disjoint, fixed-size chunks represented by c_f. The chunks are then allocated to distinct data nodes such that MapReduce tasks may be carried out in parallel across the data nodes.

A file pool is a set of files that belong to a single namespace [23]. Data nodes store file chunks for all file pools in the cluster. It is managed independently of other file pools.

Each chunk of a file has an identifier named using SHA1 [9]. Every data node, secondary master node, primary master node and cluster also has a unique ID. Cluster ID is added to identify all nodes in the cluster. The nodes of the distributed file system are organized as a ZHT network [10]. The ZHT network is initialized so that each two nodes with adjacent IDs are geometrically close. With the specified IP address of participant nodes (data/master) in the storage cloud, the space-filling curve technique [24] is used to designate IDs to the nodes, ensuring physically near nodes have adjacent IDs [3].

A data node comprises of ZHT instances as shown in Fig. 1. The ZHT instance handles requests from *MNs* or *SNs* during failure of its corresponding *MNs*. A data node can have several ZHT samples distinguished with the IP address and port. A ZHT instance is particular to the category of application of the *MN*. All ZHT instances belonging to a particular category of application of *MNs* are fully connected. Whenever a ZHT instance gets overloaded, it communicates with other ZHT instances for achieving PDLB.

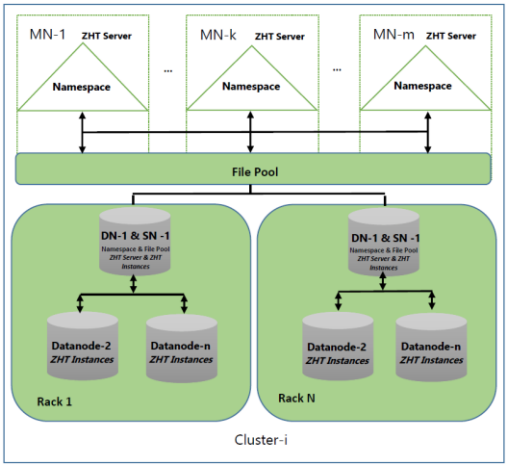

Fig. 1. PDLB High Level Cluster-Architecture

On *MN* & *SN* runs a ZHT server. ZHT server acts as a data management building block for its namespace and file pool as shown in Fig. 1. ZHT instances and ZHT servers runs the

PDLB.

For discovering file chunks, the ZHT lookup operation is carried out. ZHTs are used due to the following reasons [10]:

1. Optimization is carried out for high-end computational systems and is capable of surviving several failures and also ensuring least overhead.
2. It is flexible, supporting nodes joining and departing dynamically.
3. It has all the advantages of DHT's excellent availability and error tolerance, but simultaneously achieves the advantages of minimum latency typically related to idle centralized indexes.

4.2 Resource Heterogeneity & Fair Scheduling

PDLB employs the distributed monitoring system such as Ganglia for capturing and dynamically reporting the various system performance statistics at the rack, cluster and remote cluster level. Appendix A, in the supplementary material summarizes the most relative performance tuning parameters of HDFS used in PDLB. Note that "CPU utilization" and "compute load" represented as *PC* are interchangeable in this paper. PDLB assumes that there are four bottleneck resources for optimized load allocations to the nodes and a node's capacity is function of compute load, network bandwidth, current available I/O speed and storage load. The pluggable MapReduce fair scheduler [5] is employed by PDLB for optimized resource management. To overcome the drawbacks of periodic approach as discussed in the background section, event-driven approach is employed by PDLB to reload the allocation file. The number of maps and reduces that can run on a given data node is dynamically determined based on the storage load and compute load of the data node.

5. PERFORMANCE-DRIVEN LOAD BALANCING ALGORITHM

PDLB is applied to *MNs*, *SNs* and *DNs* for achieving optimal load balancing. *MNs* and *SNs* adopt the isolation strategy proposed in HDFS Federation [23] for further optimizing PDLB.

5.1 PDLB Policies

The 'Neighbor Node Selection Policy (NNSP)', 'Enhanced Mutual Information Feedback Policy (EMIF)', 'Load Status Monitoring Policy (LSMP)' & 'Performance Benefit Factor (PBF)' used in PDLB is detailed in Appendix A of the supplementary material of this article as these policies are adopted with minor changes from [3], [4], [5], [14], [23] and [27].

5.2 Brief Introduction to Modified Firefly Optimization Algorithm

By utilizing the load imbalance factor, a modified Firefly optimization algorithm *(MFA)* based load allocation algorithm is presented to decide the load allocation for nodes in the system.

The concept of Firefly algorithm *(FA)* was first introduced in [34]. FA is a nature inspired meta-heuristic protocol that owes its inspiration to the flashing behavior or fire flies [34]. The main aim for the firefly flash is to function as a single system for attracting other fire flies.

Xin-She yang developed FA through the following assumptions [34]:

1. That all fireflies are attracted to others,
2. Brighter fireflies attract less bright ones with attractiveness being proportional to brightness and
3. If no brighter fireflies are present, fireflies travel in arbitrary directions.

Surafel Luleseged Tilahun & Hong Choon Ong proposed a modified *FA* [35]. In modified *FA (MFA)*, the arbitrary movement of the brighter firefly (as stated above in iii.) is modified by generating random directions for determining the optimal direction wherein brightness increases.

In *FA*, the brightness is related to the objective function [34]. The designation of attractiveness in *MFA* is altered so that the impact of the objective function is magnified [35].

To apply *MFA* to the load balancing problem, the following are required:

1. Define objective function for both *DNs* and *MNs*.
2. Utilize the MFA method to search the solution space & arrive at a near optimal solution vector wherein the updating procedure of brightest firefly is altered for keeping optimal result throughout all iterations.
3. Define an optimization rule for both *DNs*, *MNs* and *SNs* which will either be maximized or minimized.

The optimization rules of PDLB are to

1. Maximize resource utilization of nodes
2. Minimize movement cost and message exchanges between nodes
3. Minimize load imbalance factor
4. Minimize algorithmic overhead

The solution vector is the load allocation plan which applies the optimization rules. As given in MFA, the brightest firefly is the firefly with current global best solution [35]. In PDLB approach, current load on the node i (L_i) is considered a firefly. The brightest firefly is the node i with highest performance benefit factor represented by β.

The following sub-sections outline the search space and the performance-driven load balancing function as the objective function for applying *MFA* to *DNs*, *MNs* & and *SNs*.

5.3 Modified Firefly Algorithm

The vector V_i is used for indicating the node selection policy of node i. $V = V_N + V_L$ (refer 'Neighbor Node Selection Policy' and 'Enhanced Mutual Information Feedback Policy' sections of Appendix A of the supplementary material). The neighbor vector V_N & local vector V_L consists of one entry per node with the node's ID as the index of each entry followed by the node's network address and its $\mathcal{T}$ as the entries for remote clusters and local clusters respectively.

$$F^i_k = (M_{c(i,k)} + (PC_k)^{-1})^{-1} - (L_k - \mathcal{T}_k) \quad (5)$$

Where F^i_k is the benefit factor of transferring load from node i to node k and PC_k is the compute load of node k. For simplicity, F^i_k represents the benefit factor of no load transfer from node i, which is in accordance with the assumption of MFA that if no optimal direction is generated, the firefly will remain in its current position.

$$\beta_i(L_k) = F^i_k - F^i_i \quad (6)$$

Where $\beta_i(L_k)$ represents the performance benefit factor of

transferring the load to node *k* from node *i*. The value of β is directly proportional to the brightness of the firefly.

$$V\beta_i[j] = k \text{ if } \beta_i(L_k) >= 0 \quad (7)$$

Where $V\beta_i[j]$ represents the performance benefit solution vector maintained by node *i*.

The objective is to find the optimal load transfer path from a node using the objective function of maximizing the performance benefit factor β.

$$Brightness = \sum_{r=1}^{n} \sum_{V\beta[i]=r}^{N[r]} f(i,r) \quad (8)$$

$$f(i,r) = log \frac{\beta(Li)}{N[r]} \quad (9)$$

Wherein *n* denotes the quantity of nodes in the system. *N[r]* refers to the quantity of nodes in a *Vß* of *r*.

As the higher brightness can bring current global best solution, Eq. (8) becomes the objective function for the MFA-based load allocation protocol.

To determine the optimal load allocation path of a node *k* with highest β, *m* unit vectors say $u_1, u_2, \ldots, u_m$ are randomly generated. An optimal path *U* is chosen from the randomly generated *m* paths such that the β of node *k* will increase if node *k* selects that path. Hence, the path of a node *k* with highest β is given by:

$$x := x + \alpha U \quad (10)$$

Wherein *x* represents *i* & *r* of *f(i,r)* in Eq. (8) and α refers to an arbitrary step length with $(0 \leq \alpha \leq 1)$. If such a path is not present amongst the arbitrary created solutions, then node *k* will not transfer any load as given in Eq. (7).

If β of node *k* is higher than the β of the node *x*, then node *x* will take the load allocation path towards node *k*. The index update of the node *x* in the system

$$x := x + A_0 e^{(-\lambda z^2)}(k-x) + \alpha\epsilon \quad (11)$$

Where $z = M_{c(k,x)}$, A_0 is the attractiveness at $z=0$, λ is the load absorption coefficient and ϵ vector of random numbers.

$$A_0 = e^{\Omega(k,x)} \quad (12)$$

$$\Omega(k,x) = F^k_k - F^x_x \text{ at } M_c = 0 \quad (13)$$

5.4 Performance Benefit Factor (β)

PDLB can be integrated as a pluggable with the existing huge-scale distributed file systems such as Google GFS [4] as well as Hadoop HDFS [5]. Particularly, for incorporating PDLB with the master nodes in Hadoop HDFS [5], every data node employs enhanced mutual information feedback policy to piggyback its locally hosted chunks' data to the master nodes, such that the master nodes can collect the locations of chunks in the system.

A master node will not assign further chunks to a heavy data node if the data node continues to be heavy for more than $(T_p)^q$ time where $q \neq 1$ as given in MFA. If for more than $(T_p)^q$ time, the number of heavy nodes that continue to be remain heavy exceeds $|d/q|$ or $|m/q|$, then a new node (master/data) request is initiated by PDLB.

Also note that in the distributed load re-balancing approach proposed by Hsiao et al [3], storage load-based sorting of data nodes in the system is done periodically. In PDLB, such sorting is done whenever data is added/deleted, or the node is heavy/added/deleted.

The distributed load re-balancing approach proposed by Hsiao et al [3] requires to periodically send load status messages to data nodes that are not physically close thereby introducing additional overhead on message exchanges, whereas PDLB exchanges load status messages among nodes from their local and neighbor vector only.

As PDLB extends the approach in HDFS Federation [23] at two levels i.e. primary master nodes level and secondary master nodes level, the failure of the master node does not prevent the data node from serving other master nodes in the cluster because each data node registers to each distinctive category master node and to one master node from the set of shared category master nodes.

Note that when a master node is deleted, the corresponding file pool at the data nodes is also deleted by PDLB.

Algorithm 1: Algorithm PDLB with MFA

Begin

1. Objective function: $f(x)\ x=(x_1,x_2,\ldots,x_n)$ Eq. (8)
   Where *x* refers to *i* & *r* of *f(i,r)*
2. For each *SN*, *DN* & *MN* sets
   For each different application category of *MN*
   Create an initial population of fireflies
   x_i $(I = 1,2,\ldots n)$
3. Attractiveness *A* associated with f(x) as $A = f(x)$
4. Define load absorption coefficient λ as 2

While (t < 500)

for i = 1 to n (all n fireflies)

for j = 1 to m (m fireflies NNSP)

Call LSMP (Refer Appendix A of Supplementary)

Choose optimal path [Eq. (10)]

If $(A_j > A_i)$

Move firefly i toward j [Eq. (11)]

Differ attractiveness with M_c *[Eq. (12)]*

Evaluate new solutions and update Attractiveness

[Eq. (6) & Eq. (7)]

End If

End for j

End for i

Rank fireflies & find the current best

Call EMIF

End While

6. PERFORMANCE COMPARISON OF PDLB and DLRA

Table 1 compares the time complexity of PDLB and **D**istributed **L**oad **R**ebalancing **A**lgorithm in short (DLRA) [3].

Table. 1. Performance Comparison of PDLB and DLRA

Comparison Parameter	**Symbol Description**	**PDLB**	**DLRA [3]**
Routing Time	C = Max. size of the network	0 to 2	O(c)
Number of algorithmic rounds in expectation so that system comprises no light nodes	K is the initial number of heavy data nodes	O(log log K)	O(log K)
A light data node gathers a number of vectors per algorithmic round	Per algorithmic round, n_v number of vectors are gathered by a light data node. Each data node contacts a set of chosen data nodes in the system and generates a vector V	O(|V|log|V|)	O(n_v|V|log|V|)
Spreading of a message in a network and rounds of communication	Each data node contacts a set of chosen data nodes in the system and genreates a vector V	O(log|V|loglog V) rounds of communication and O(|V| loglog|V|) messages	O(log|V|) rounds of communication and O(|V|log|V|) messages

Thus, from Table 1, it can be concluded that PDLB leads to dramatically lower network congestion and achieves quicker convergent rate about the quantity of algorithmic rounds required to ensure that the model contains no light nodes as compared to DLRA. The basic algorithm of PDLB is available in Appendix B of the online supplementary material.

7. SIMULATIONS

7.1 Simulation Setup

The performance of PDLB is tested through Java based computer simulations. In the simulations, PDLB is carried out by using ZHT_Sim [28], a zero-hop distributed hash table simulator for extreme scale system services written in Java. The number of files and file chunks originally hosted by a node in the simulations follow the geometric distribution, permitting stress tests as proposed in [3]. PDLB is compared with **D**istributed **L**oad **R**ebalancing **A**lgorithm in short (DLRA) [3]. The number of random directions considered is 20.

Table 2 reveals the values of the variables utilized in the simulations and heterogeneous system configurations. A cloud network topology connecting the storage nodes is simulated in a 2D torus direct network [3], [29].

Table. 2. Simulation Parameters (tu=time unit, pt=percent)

Simulation Parameter	**Value**
Set Cardinality | . |	-
Absolute Value Function || . ||	-
Size of system, s	1010
The number of primary & secondary master nodes, m	10
The number of data nodes, d	1000
The period for periodic information exchange, T_p	10 tu
Number of file chunks, nfc	10,000
Number of clients	4
Average storage capacity (ASC) among nodes, S_c	11
Average processing power (APW) among data nodes, P_p	11
Maximum and minimum storage capacities is	110 , 2
Random number that each node contacts, r	70-100
Chunk allocator factor, q	2
File size, fs	10
Chunk size, cs	1
Mean transfer delay, μ	0.05 tu
Standard deviation of transfer delay, Ω	50pt
Storage load per unit capacity, γ	0.5
power-law distribution, α	2

Fig.2 displays the cumulative distribution function (CDF) of the data in the simulation, where workload-I represents the variation of the geometric distribution. Workload-I indicates that a small quantity of nodes originally processes a huge amount of data.

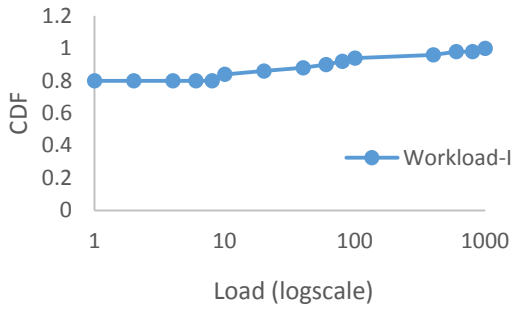

Fig. 2. The Workload Distribution

7.2 Simulation Results

Fig. 3 gives the simulation outcomes of the storage load distribution after performing the DLRA and PDLB. The simulation outcomes indicate that PDLB performs better than DLRA with regard to the storage load imbalance factor because PDLB employs MFA to ensure that a heavy data node does not overload a light data node during the storage load migration and also the master nodes do not further assign chunks to overloaded data nodes. On the contrary, in DLRA, the storage load migration from a heavy data node to light data node may overload the light data node.

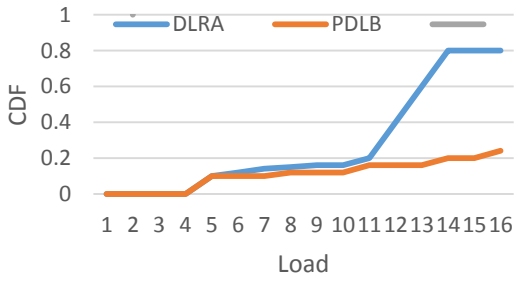

Fig. 3. The Load Distribution

Fig. 4 shows the movement cost of DLRA and PDLB. The movement cost of PDLB is less than that of DLRA. This is because, DLRA matches the top light data nodes with the top-heavy data nodes and a light data node is required to shed its load to its successor data node in order to accept storage load from a heavy data node. In PDLB, a light data node need not shed its storage load rather accept the storage load from a heavy data node.

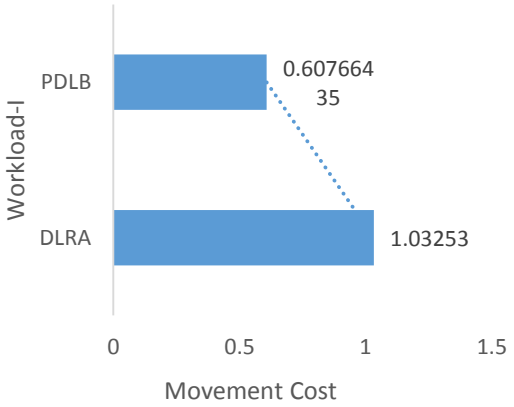

Fig. 4. The Movement Cost

Fig. 5 reveals the total quantity of messages created by a PDLB and DLRA. In DLRA, each data node probes a set of other data nodes in the system periodically and may reallot its storage load from/to the probed data nodes, introducing more messages. On the contrary, in PDLB, each data node gathers partial system knowledge from its local and neighbor data nodes based on the EMIF policy. Only a heavy data node reallocates some of its storage load to one of the probed light data node through EMIF thereby introducing less messages. Also, in PDLB, load status exchange is performed among geographically close nodes thereby preventing the messages that travel between two nodes, to traverse a long physical distance through various physical network links. Also, a data node is required to only send its load status to each distinct category master node and one geographically close same category master node thus again reducing the number of messages.

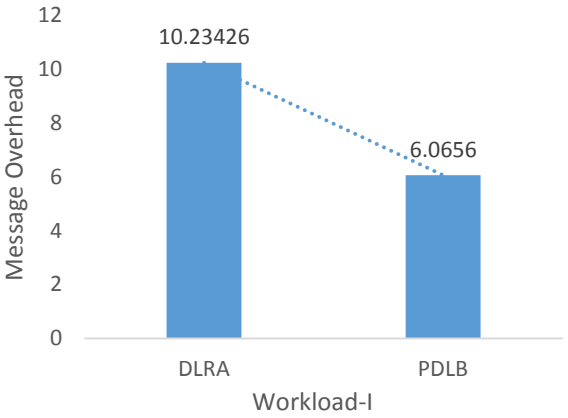

Fig. 5. The Message Overhead

Both DLRA and PDLB depend on DHT network in the simulations. But, in DLRA the data nodes can depart from or join the network for load re-balancing thereby raising the overheads needed for maintaining DHT structures. Hence, the quantity of re-joining operations of PDLB is further investigated.

Fig. 6 illustrates the simulation results. It is seen that in DLRA, only light data nodes rejoin the system as successors of heavy data nodes. DLRA tries to pair light and heavy data nodes precisely. PDLB pairs a heavy data node with compatible light data node thus preventing light data nodes from shedding their

storage load or rejoining the heavy data nodes. Note the PDLB is comparable with the centralized algorithm used in Google GFS and Hadoop HDFS. This is because, like the centralized algorithm, PDLB does not introduce rejoining overhead as data nodes in PDLB need not self-organize or self-heal for rejoining operations.

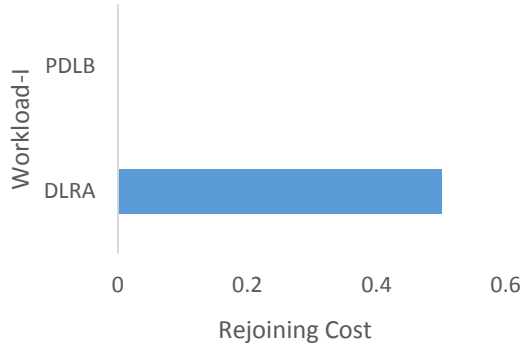

Fig. 6. The Rejoining Cost

Fig. 7 illustrates the weighted communication cost (WCC) for DLRA and PDLB by investigating the network traffic introduced in them. The weighted communication cost is defined as follows

$$\sum_{i \in M} size^i * link^i \quad (14)$$

Where M denotes the load status messages, $size^i$ is the size of the message i and $link^i$ represents the communication cost incurred by message i . In the simulations, it is presumed that the size of each message is identical, i.e. $size^i=1$ *for all* $i \in M$ with no loss of generosity. Thus, based on (4), the higher the WCC, the more physical links utilized for load status message exchanges. The simulations reveal that PDLB performs better than DLRA in terms of WCC. This is because, each light data node in DLRA first finds several matched heavy data nodes from its vector. While doing so, DLRA does not exploit physical network proximity thus leading to higher WCC as compared to PDLB.

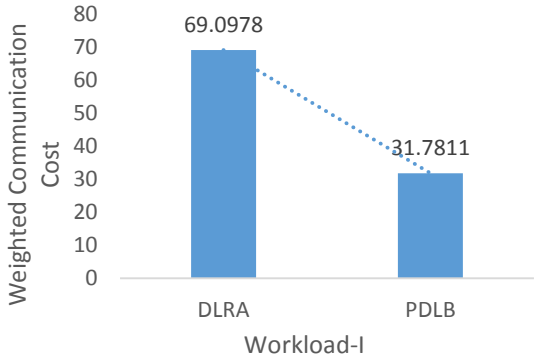

Fig. 7: The WCC

The effect of node heterogeneity in terms of storage and computational power is investigated. In this experiment, the storage and compute capabilities of the nodes follow the power-law distribution, which is the zipf distribution [3], [30],[31] and [32]. In, fig. 8, the ratio of the amount of data hosted by every node *i*, to its storage capacity represented by *P* is assessed [3]. Node *i* tries to make minimum ||P- γ || for approaching the load balanced state [3]. The simulations denote that PDLB outperforms than DLRA. This is because in DLRA, light data nodes may need to offload their loads to their successor data nodes, so that the light data node can accept storage load from a heavy data node.

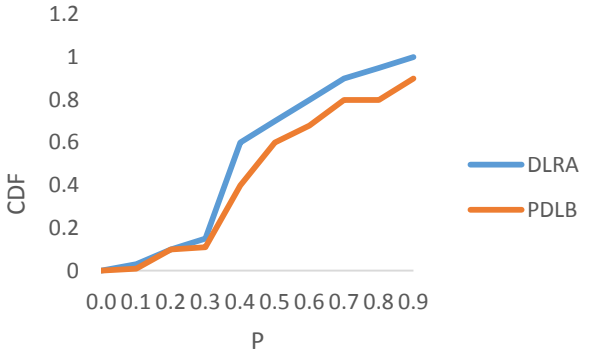

Fig. 8. The Effect of Data Node Heterogeneity

By default, the quantity of data nodes and the quantity of chunks in the experiments are *d=1000* and *nfc=10,000* respectively. Fig. 9 illustrates the effect of varying *nfc* by having *nfc*=10,000, 20,000 and 80,000 for DLRA and PDLB with *d*=1000. As shown in fig. 9, PDLB adapts well as compared to DLRA disregarding the quantity of chunks in the system.

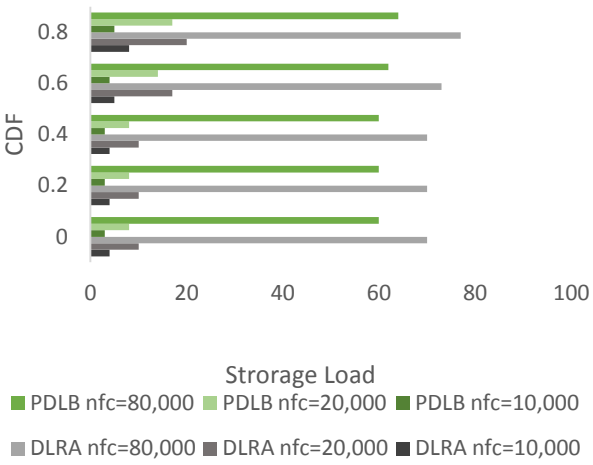

Fig. 9: The Effect of Varying nfc (Workload -I)

The performance effect of different range of random number *r* is investigated. The simulation results are depicted in fig. 10 where different value of *r*=10, 100 and 1000 is studied for workload-I. As shown in fig. 10, without global knowledge PDLB performs very well for *r*=100 and 10,000 as compared to DLRA.

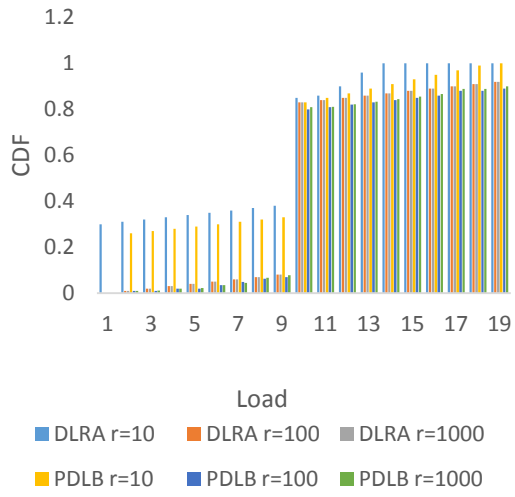

Fig. 10: The Effect of Sampling Quality

8. EXPERIMENTATION

8.1 Experimental Environment Setup

The performance is PDLB is evaluated by setting up an inter-connected cluster running Ubuntu 10.10 with eight heterogeneous systems, with each system having a minimum of two cores. The specifications of the cluster are given in Table 3 On each system, the Google Protocol buffers C++ and C bindings are installed.

Table. 3. Cluster Specification

Sl. No.	Node	Processor	RAM
1.	Master	Intel Core 2 Duo	3 GB
2.	Master	Intel Centrino	3GB
3.	Master	Intel i7	8GB
4.	Data	Intel Centrino	2GB
5.	Data	Intel Core 2 Duo	3GB
6.	Data	Intel Pentium	3GB
7.	Data	Intel Pentium	2GB
8.	Data	Intel i3	4GB

ZHT using MPI protocol is then deployed on the cluster. Hadoop HDFS Federation version 2.3.0 is installed on the cluster with three dedicated master nodes and 5 data nodes. Out of the three dedicated master nodes, two master nodes share a common application. PDLB is implemented on the cluster and its performance is assessed against the load balancer in HDFS Federation and DLRA.

Four multi-threaded client programs are established in the multi-threaded environment for issuing requests to the master nodes. Requests are commands for creating directions with arbitrarily assigned names and removing randomly chosen directories. For emulating the loads of the master node in a production system as well as for investigating the impact of the master node's loads on the performance of PDLB, the processor cycles available for the master node is limited by differing the maximal processor utilization represented by P_U, P_U=8%, 16%, 32%, 64% and 99%.

Total quantity of files and chunks distributed in file the system in our experiment is restricted to 16 and 128 respectively. All the nodes are linked with a 100 Mbps fast Ethernet switch. The size of a chunk is set to 16MB. Hence, transferring the chunks takes no more than (16*128*8)/100 = 2.8 minutes if the network bandwidth is completely used. The original placement of 16 files and 128 chunks follow the geometric distribution.

For every experimental round, the time elapsed to finish PDLB, DLRA and HDFS Federation load balancer is measured. Eight runs are performed for a particular P_U and the average time needed for implementing the three algorithms is computed. Note that each data node randomly selects three samples.

8.2 Experimental Results

The experimental results are demonstrated in Fig. 11a, reveals the setup of the experiment. Fig. 11b, shows the time required for carrying out PDLB, DLRA and HDFS Federation load balancer. PDLB clearly outperforms HDFS Federation load balancer and DLRA.

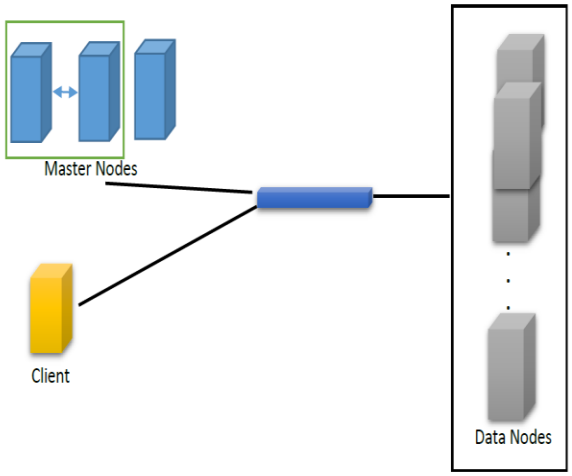

Fig. 11a: Experimental Environment Setup

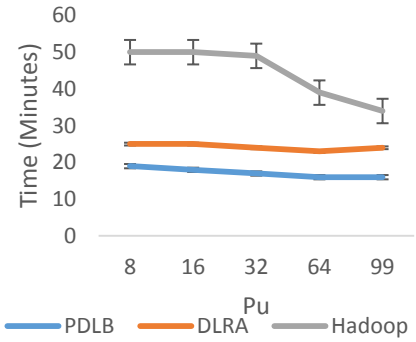

Fig. 11b: Time vs. PU

9. CONCLUSION

In the current study, a performance-driven load balancing protocol using modified firefly algorithm for dealing with the load imbalance issue in huge-scale, dynamic as well as distributed emerging file systems in the clouds has been suggested. PDLB strives for balancing the storage load of both the master nodes as well as the data nodes by taking into consideration physical network locality, communication cost,

movement cost and node heterogeneity in terms of both storage and compute.

If no representative real workloads are present in terms of distribution of chunks in a storage system in the public domain, the performance of PDLB is investigated against DLRA and Hadoop HDFS through synthesized probability distribution of chunks. The synthesis workloads stress test the protocols through the creation of a few data nodes and master nodes which are heavily loaded.

The computer simulation results indicate that PDLB outperforms DLRA and Hadoop HDFS Federation load balancer about load balance factors, communication and movement costs as well as algorithmic overheads. Furthermore, PDLB exhibits a fast-convergent rate as compared to DLRA and HDFS. The efficacy of PDLB is additionally proved with real-implementations with small-scale cluster environments.

10. REFERENCES

[1] Grossman, Robert L., Yunhong Gu, Michael Sabala, and Wanzhi Zhang. "Compute and storage clouds using wide area high performance networks."Future Generation Computer Systems 25, no. 2 (2009): 179-183.

[2] Xiao, Zhen, Weijia Song, and Qi Chen. "Dynamic resource allocation using virtual machines for cloud computing environment." Parallel and Distributed Systems, IEEE Transactions on 24, no. 6 (2013): 1107-1117.

[3] Hsiao, Hung-Chang, Hsueh-Yi Chung, Haiying Shen, and Yu-Chang Chao. "Load rebalancing for distributed file systems in clouds." Parallel and Distributed Systems, IEEE Transactions on 24, no. 5 (2013): 951-962.

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," Proc. 19th ACM Symp. Operating Systems Principles (SOSP '03), pp. 29-43, Oct. 2003.

[5] Hadoop Distributed File System, http://hadoop.apache.org/hdfs/, 2012.

[6] L. M. Ni and K. Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," IEEE Trans. Software Eng., vol. 11, no. 5, pp. 491–496, May 1985.

[7] L. M. Ni, C.-W. Xu, and T. B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," IEEE Trans. Software Eng., vol. 11, no. 10, pp. 1153–1161, Oct. 1985.

[8] Zhang, Qi, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L. Hellerstein. "Dynamic energy-aware capacity provisioning for cloud computing environments." In Proceedings of the 9th international conference on Autonomic computing, pp. 145-154. ACM, 2012.

[9] Eastlake, Donald, and Paul Jones. "US secure hash algorithm 1 (SHA1)." (2001).

[10] Li, Tonglin, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, and Ioan Raicu. "ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table." In Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 775-787. IEEE, 2013.

[11] McNett, Marvin, Diwaker Gupta, Amin Vahdat, and Geoffrey M. Voelker. "Usher: An Extensible Framework for Managing Clusters of Virtual Machines." In LISA, vol. 7, pp. 1-15. 2007.

[12] C. A. Waldspurger, "Memory resource management in VMware ESX server," in Proc. of the symposium on Operating systems design and implementation (OSDI'02), Aug. 2002.

[13] Abu-Libdeh, Hussam, Paolo Costa, Antony Rowstron, Greg O'Shea, and Austin Donnelly. "Symbiotic routing in future data centers." ACM SIGCOMM Computer Communication Review 41, no. 4 (2011): 51-62.

[14] Balasangameshwara, Jasma, and Nedunchezhian Raju. "Performance-driven load balancing with a primary-backup approach for computational grids with low communication cost and replication cost." Computers, IEEE Transactions on62, no. 5 (2013): 990-1003.

[15] Naor, Moni, and Udi Wieder. "A simple fault tolerant distributed hash table." InPeer-to-Peer Systems II, pp. 88-97. Springer Berlin Heidelberg, 2003.

[16] Li, Tonglin, Xiaobing Zhou, Kevin Brandstatter, Dongfang Zhao, Ke Wang, Anupam Rajendran, Zhao Zhang, and Ioan Raicu. "ZHT: A light-weight reliable persistent dynamic scalable zero-hop distributed hash table." In Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, pp. 775-787. IEEE, 2013.

[17] H. Shen and C.-Z. Xu, "Locality-Aware and Churn-Resilient Load Balancing Algorithms in Structured P2P Networks," IEEE Trans. Parallel and Distributed Systems, vol. 18, no. 6, pp. 849-862, June 2007.

[18] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," Performance Evaluation, vol. 63, no. 6, pp. 217-240, Mar. 2006.

[19] Birman, Ken. "The promise, and limitations, of gossip protocols." ACM SIGOPS Operating Systems Review 41, no. 5 (2007): 8-13.

[20] I. Stoica, R. Morris, 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," IEEE/ACM Trans. Networking, vol. 11, no. 1, pp. 17-21, Feb. 2003.

[21] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms Heidelberg, pp. 161-172, Nov. 2001.

[22] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's Highly Available Key-Value Store," Proc. 21st ACM Symp. Operating Systems Principles (SOSP '07), pp. 205-220, Oct. 2007.

[23] HDFS Federation, http://hadoop.apache.org/common/docs/r0.23.0/ hadoop-yarn/hadoop-yarn-site/Federation.html

[24] Sagan, Hans. Space-filling curves. Vol. 18. New York: Springer-Verlag, 1994.

[25] Wolski, Rich, Neil T. Spring, and Jim Hayes. "The network weather service: a distributed resource

performance forecasting service for metacomputing." Future Generation Computer Systems 15, no. 5 (1999): 757-768.

[26] Heger, Dominique. "Hadoop Performance Tuning-A Pragmatic & Iterative Approach." *CMG Journal* (2013).

[27] Balasangameshwara, Jasma, and Nedunchezhian Raju. "A hybrid policy for fault tolerant load balancing in grid computing environments." Journal of Network and Computer Applications 35, no. 1 (2012): 412-422.

[28] Wang, Ke, Abhishek Kulkarni, Michael Lang, Dorian Arnold, and Ioan Raicu. "Using simulation to explore distributed key-value stores for extreme-scale system services." In *Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis*, p. 9. ACM, 2013.

[29] Raicu, Ioan, Ian T. Foster, and Pete Beckman. "Making a case for distributed file systems at exascale." In *Proceedings of the third international workshop on Large-scale system and application performance*, pp. 11-18. ACM, 2011.

[30] Zhu, Yingwu, and Yiming Hu. "Efficient, proximity-aware load balancing for DHT-based P2P systems." *Parallel and Distributed Systems, IEEE Transactions on* 16, no. 4 (2005): 349-361.

[31] Shen, Haiying, and Cheng-Zhong Xu. "Locality-aware and churn-resilient load-balancing algorithms in structured peer-to-peer networks." *Parallel and Distributed Systems, IEEE Transactions on* 18, no. 6 (2007): 849-862.

[32] Hsiao, Hung-Chang, Hao Liao, Ssu-Ta Chen, and Kuo-Chan Huang. "Load balance with imperfect information in structured peer-to-peer systems." *Parallel and Distributed Systems, IEEE Transactions on* 22, no. 4 (2011): 634-649.

[33] Hua, Xiayu, Hao Wu, Zheng Li, and Shangping Ren. "Enhancing throughput of the Hadoop Distributed File System for interaction-intensive tasks." Journal of Parallel and Distributed Computing 74, no. 8 (2014): 2770-2779.

[34] Yang, Xin-She. Nature-inspired metaheuristic algorithms. Luniver press, 2010.

[35] Tilahun, Surafel Luleseged, and Hong Choon Ong. "Modified firefly algorithm." Journal of Applied Mathematics 2012 (2012).

[36] Chatterjee, A.; Mahanti, G. K.; Chatterjee, A. (2012). "Design of a fully digital controlled reconfigurable switched beam conconcentric ring array antenna using firefly and particle swarm optimization algorithm". Progress in Elelectromagnetic Research B 36: 113–131.