ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

A Scalable Distributed File System for Cloud Computing

Gangam Soumya[1], Indira Priyadarshini [2]

[1]Student, BHARAT INSTITUTE OF TECHNOLOGY & SCIENCE FOR WOMENS, Mangalpally(V), Ibrahimpatnam(M),R.R Dist Hyderabad, AP, INDIA

[2]Associate Professor, BHARAT INSTITUTE OF TECHNOLOGY & SCIENCE FOR WOMENS, Mangalpally(V), Ibrahimpatnam(M),R.R Dist Hyderabad, AP, INDIA

Abstract: Distributed file systems area unit key building blocks for cloud computing applications supported the Map Reduce programming paradigm. In such file systems, nodes at the same time serve computing and storage functions; a file is partitioned off into variety of chunks allotted in distinct nodes in order that Map Reduce tasks are often performed in parallel over the nodes. However, during a cloud computing atmosphere, failure is that the norm, and nodes is also upgraded, replaced, and more within the system. Files may also be dynamically created, deleted, and appended. This ends up in load imbalance during a distributed file system; that's, the file chunks aren't distributed as uniformly as attainable among the nodes. Rising distributed file systems in production systems powerfully rely on a central node for chunk reallocation. This dependence is clearly inadequate during a large-scale, failure-prone atmosphere as a result of the central load balancer is anesthetize goodish work that's linearly scaled with the system size, and will so become the performance bottleneck and also the single purpose of failure. during this paper, a totally distributed load rebalancing rule is given to address the load imbalance downside. Our rule is compared against a centralized approach during a production system and a competitive distributed answer given within the literature. The simulation results indicate that our proposal is comparable the present centralized approach and significantly outperforms the previous distributed rule in terms of load imbalance issue, movement price, and algorithmic overhead.

Keywords: Map Reduce, Distributed file system, rebalancing, node, chunk.

I.INTRODUCTION

Distributed Computing Environment (DCE), developed at IBM Transarc research laboratory, provides a user with the ability to store and access knowledge at remote sites, just like the techniques used with Network classification system (NFS). Structurally, DCE DFS could be a assortment of many file systems that square measure mounted onto one virtual classification system area with a single namespace. The top user has direct access to all or any files during this distributed file system without knowing wherever the physical files reside. Putting file systems onto different servers so as to supply the optimum service for the top users, moreover as optimize the use of accessible resources, is load equalization of DFS servers. Load balancing for distributed systems represents mapping or

Remapping of labor to different processors with the intent of distribution every processor AN equal quantity of labor. Load equalization of information is already a lot of economical in DFS than in customary non-distributed file systems. One reason is that the use of replication, that provides an alternative for read only. DCE filesets to be replicated on multiple machines. Requests for files from frequently used .read-only. Filesets square measure then unfold across completely different machines, preventing anyone machine from changing into burdened with knowledge requests.

Our goal during this chapter is to gift a brand new methodology for managing read write. Filesets across the DFS cell. The planned methodology employs data mining techniques and graph theory algorithms to accomplish the specified results of improved employment distribution between DFS servers. The information mining approach generates association

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

rules distinguishing existing file access patterns, whereas graph analysis helps optimize relocation selections and suggest fileset transfers. By implementing the planned methodology, we have a tendency to extend and improve the load balancing techniques presently gift in DFS by augmenting them with the improved management of .read-write. DCE DFS filesets (in addition to read-only. filesets) across multiple DFS file servers. Our methodology is intended to make intelligent selections on mapping .read-write. Filesets to multiple DFS file servers.

II. RELATED WORK

Existing solutions to balance load in DHTs incur a high overhead either in terms of routing state or in terms of load movement generated by nodes incoming or outward the system. during this paper, we have a tendency to propose a group of general techniques and use them to develop a protocol supported Chord, called Y0, that achieves load equalization with tokenism overhead underneath the everyday assumption that the load is uniformly distributed within the symbol house.

In explicit, we have a tendency to prove that Y0 can do near-optimal load equalization, whereas moving very little load to keep up the balance and increasing the scale of the routing tables by at the most a relentless issue. exploitation in depth simulations supported real-world and artificial capability distributions, we have a tendency to show that Y0 reduces the load imbalance of Chord from O (log n) to a but three.6 while not increasing the quantity of links that a node has to maintain. Additionally, we have a tendency to study the result of heterogeneousness on each DHT, demonstrating considerably reduced average route length as node capacities become more and more heterogeneous. For a real-world distribution of node capacities, the route length in Y0 is asymptotically but [*fr1] the route length within the case of a homogeneous system.

P2P communication:

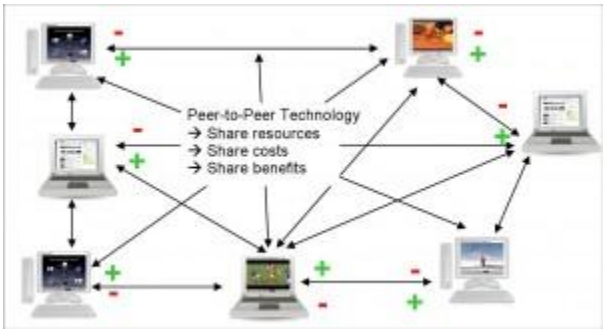


Fig:1- P2P Communication

Alternatively remarked as P2P, P-to-P and P2P communications, peer-to-peer communication refers to the transmission between 2 peer computers over a network. P2P became wide known by laptop users as they began sharing MP3s and alternative files over P2P networks. as an example, Napster is associate example of a P2P software package application. once downloading and putting in this program users were able to connect with alternative computers, look for songs, and transfer any of them freely.

***Node Departure*:** While within the network, every node manages information for a specific vary. once the node departs, the information is keep becomes unprocurable to the remainder of the peers. P2P networks reconcile this information loss in 2 ways: (a) Do nothing and let the "owners" of the information contend with its accessibility.

The house owners can oftentimes poll the information to discover its loss and re-insert the information into the network. Maintain replicas of every vary across multiple nodes. A Skip web DHT organizes peers and information objects per their composition addresses within the variety of a variant of a probabilistic skip list. It supports index time range-based lookups and guarantees path section. Mercury is a lot of general than Skip web since it supports range-based lookups on multiple-attributes. Our use of sampling to estimate question property constitutes a unique contribution towards implementing ascendable multi-dimensional vary queries. Load reconciliation is another necessary manner during which Mercury from Skip web. whereas Skip web incorporates a strained load-balancing mechanism, it's solely helpful once a part of an information name is hashed, during which case the half is inaccessible for performing arts a spread question. this means that Skip web supports load-balancing or vary queries not each.

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

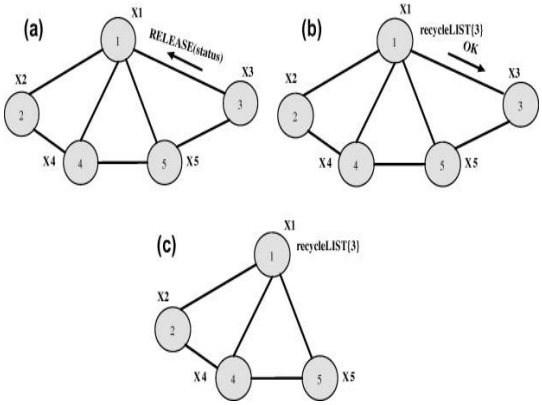


Fig: 2- Node Communication

III. MODULE IMPLEMENTATION

3.1 Chunk creation:

A file is partitioned into a number of chunks allocated in distinct nodes so that Map Reduce Tasks can be performed in parallel over the nodes. The load of a node is typically proportional to the number of file chunks the node possesses. Because the files in a cloud can be arbitrarily created, deleted, and appended, and nodes can be upgraded, replaced and added in the file system, the file chunks are not distributed as uniformly as possible among the nodes. Our objective is to allocate the chunks of files as uniformly as possible among the nodes such that no node manages an excessive number of chunks. Note also that only a few nodesare close enough to any vacated address to claim it (distantones will be shielded by some closer active node),and thus, as the address being vacated gets higher and higher in the order, it become less and less likely that any node that can take it will want it. We have shown how to balance the address space, but sometimes this is not enough. Some applications, such as those aiming to support range-searching operations, needto specify a particular, non-random mapping of items into the address space.

In this section, we consider a dynamic protocol that aims to balance load for *arbitrary* item distributions.To do so, we must sacrifice the previous protocol srestriction of each node to a small number of virtual node locations—instead, each node is free to migrate anywhere. This is unavoidable: if each node is limited to a bounded number of possible locations, then for any *n* nodes we can enumerate all the addresses they might possibly occupy, take two adjacent ones, and address all the items in between them: this assigns all the items toone unfortunate node.

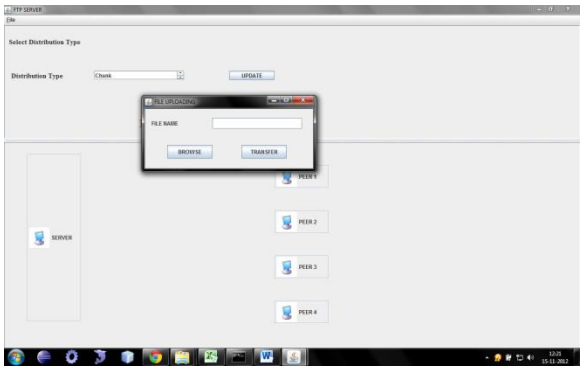


Fig:3-Chunk Creation

3.2 Distributed Hash Table formulation

The storage nodes are structured as a network based on *distributed hash tables* (*DHTs*), e.g., discovering a file chunk can simply refer to rapid key lookup in DHTs, given that a unique handle (or *identifier*) is assigned to each file chunk. DHTs enable nodes to self-organize and -
Repair while constantly offering lookup functionality in node dynamism, simplifying the system provision and management. The chunk servers in our proposal are organized as a DHT network. Typical DHTs guarantee that if a node leaves, then its locally hosted chunks are reliably migrated to its successor; if a node joins, then it allocates the chunks whose IDs immediately precede the joining node from its successor to manage.

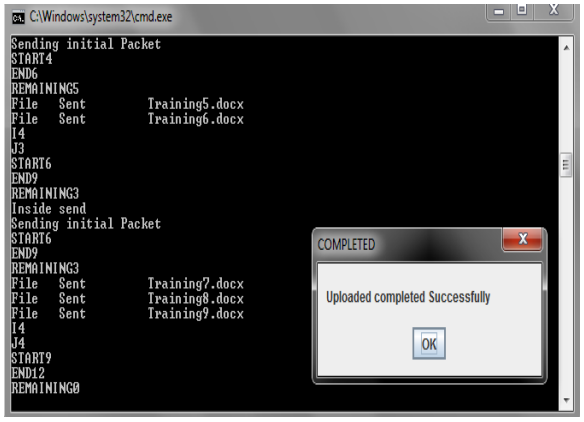


Fig: 4-DHT Formulation

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

3.3 Load balancing algorithm

In our proposed algorithm, each chunk server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. A node is *light* if the number of chunks it hosts is smaller than the threshold. Load statuses of a sample of randomly selected nodes. Specifically, each node contacts a number of randomly selected nodes in the system and builds a vector denoted by V. A vector consists of entries, and each entry contains the ID, network address and load status of a randomly selected node.

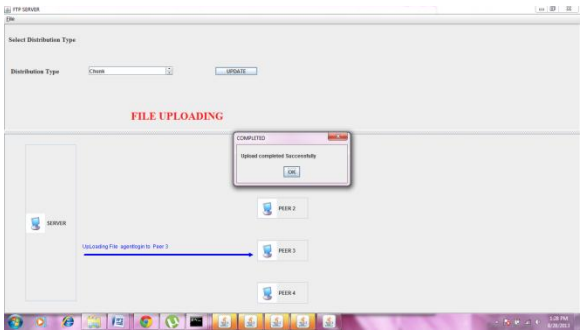


Fig:5 – Load Rebalancing

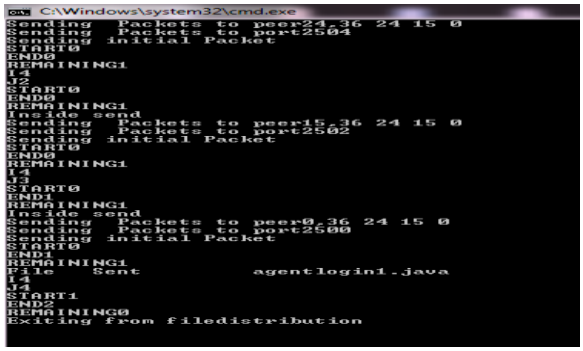


Fig:6 –File Upload in Cloud

3.4 Replica Management

In distributed file systems (e.g., Google GFS and Hadoop HDFS), a constant number of replicas for each file chunk are maintained in distinct nodes to improve file availability with respect to node failures and departures. Our current load balancing algorithm does not treat replicas distinctly. It is unlikely that two or more replicas are placed in an identical node because of the random nature of our load rebalancing algorithm. More specifically, each under loaded node samples a number of nodes, each selected with a probability of 1/n, to share their loads (where n is the total number of storage nodes).

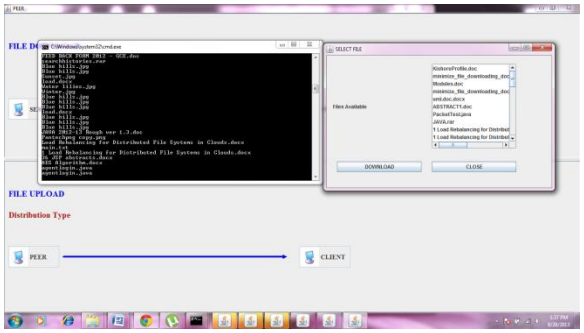


Fig:7 – File Download In cloud

Test cases:

A **test case**, in software engineering, is a set of conditions or variables under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do.

Test Case:

Test Case	Check Field	Objective	Expected Result
TC-001	Server(Sub Server1)	Start The Server1 and minimize it	Error means should create Executable jar files
TC-002	Server(Sub Server2)	Start The Server2 and minimize it	Error means should create Executable jar files and check server 1
TC-004	Server(Sub Server3)	Start The Server3 and minimize it	Error means should create Executable jar files and check server2
TC-003	Server(Sub Server4)	Start The Server4 and minimize it	Error means should create Executable jar files and check3
TC-005	Main Server	Not Enter his username and password correctly.	Error Message should appears that "Enter correct username and password"
TC-006	Main server	upload the files	Not uploaded means check lan connection
TC-007	Client	Download the upload files	ERROR"Should check all servers"

IV. Algorithm

Distributed load levelling scenario in that users assign resources during a non-cooperative and selfish fashion. The perceived performance of a resource fora user decreases with the amount of users that assign there source. In our dynamic, coincidental model, users might apportion resources during a round-based fashion. A user has zero utility when falling in need of an explicit minimum performance threshold and having positive utility otherwise. These protocols operate by activating users in parallel permitting them to boost their presently perceived performance.

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

Procedure 1 ADJUSTLOAD (Node Ni) fOn Tuple Insertg

1: Let L(Ni) = x 2 (Tm; Tm+1].
2: Let Nj be the lighter loaded of Ni□1 and Ni+1.
3: **if** L(Nj) _ Tm□1 **then** fDo NBRADJUSTg
4: Move tuples from Ni to Nj to equalize load.
5: ADJUSTLOAD(Nj)
6: ADJUSTLOAD(Ni)
7: **else**
8: Find the least-loaded node Nk.
9: **if** L(Nk) _ Tm□2 **then** fDo REORDERg
10: Transfer all data from Nk to N = Nk_1.
11: Transfer data from Ni to Nk, s.t. L(Ni) = dx=2e and
L(Nk) = bx=2c.
12: ADJUSTLOAD (N)
13: fRename nodes appropriately after REORDER.g
14: **end if**
15: **end if**

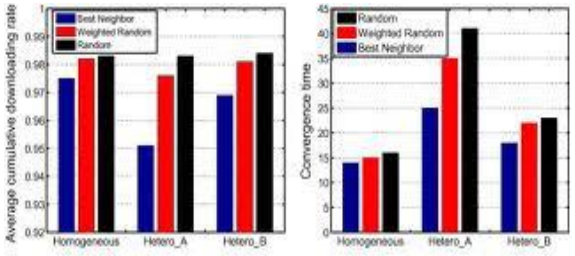
For example, a user presently assigned to a resource could sample another resource in keeping with a probability distribution and migrate to the new resource with a sure likelihood. Whereas being supported native info in theory, most of the protocols given within the literature additionally admit some quantity of worldwide info, e.g. the set of under loaded resources or the present performance of the sampled resource. In distinction, the user thresholds allow us to style algorithms, during which the actions performed by a user rely solely on info concerning the performance ofthe resource it's presently assigned to.

Fig:8- Load Balance

The load reconciliation formula (or load reconciliation method) defines the factors that the NetScaler uses to pick the server to that to send consumer requests. once the designed criteria area unit met for the chosen server, the NetScaler then selects a special server. Load reconciliation roughness refers to the factors that the NetScaler uses to determine the load reconciliation technique in a very given scenario. The NetScaler performs request-based, connection-based, or time-based load reconciliation, reckoning on the protocol of the service it's load reconciliation. at intervals every form of load reconciliation, there area unit varied load reconciliation ways. as an example, the smallest amount association technique selects the service with the smallest amount variety of active connections to confirm that the load of the active requests is balanced on the services.

Distribution Results:

We have given many incontrovertibly economical loads leveling for distributed file's protocols for distributed information storage in P2P systems. Additional details and analysis may be found in an exceedingly thesis. Our algorithms square measure straightforward and simple to implement in. distributed files therefore a lucid next analysis step ought to be a sensible analysis of those schemes. Additionally, many concrete open issues follow .From our work. First, it'd be potential to additional improve the consistent hashing theme as mentioned at the tip of our vary search system. Distributed doesn't easily generalize to quite one order. as an example (Fig.2) once storing music files, one may need to index them by each creator and song title, permitting lookups per 2 orderings. Since our protocol rearranges the things per the ordering, doing this for 2 orderings at a similar time looks tough. A simple, however unpolished, answer is to arrange not the things themselves, however simply store tips that could them on the nodes. this needs way less storage, and

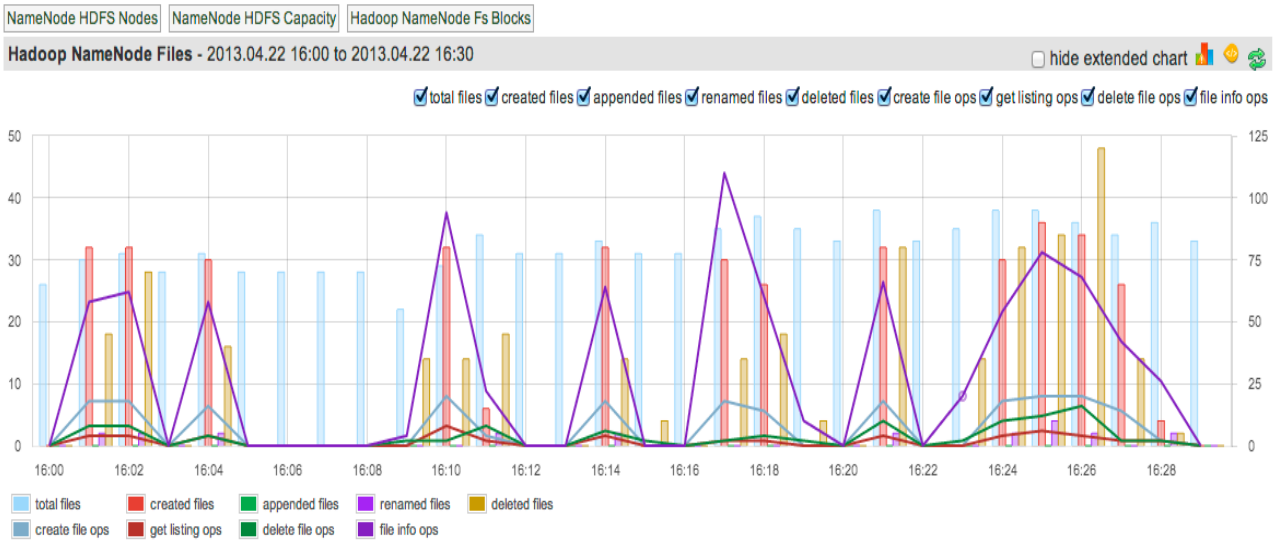


Network Setting Network Setting

The average downloading rate and Convergence time

ISSN-2320-7884 (ONLINE)
ISSN-2321-0257 (PRINT)

Makes it doable to take care of 2 or a lot of orderings right away. Lastly, allowing nodes to decide on capricious addresses in our item reconciliation protocol for distributed file's makes it easier for malicious nodes to disrupt the operation of the P2P network. It'd be attention-grabbing to seek out counter-measures for this downside.



V.CONCLUSION

In this paper our proposal strives to balance the masses of nodes and cut back the demanded movement value the maximum amount as attainable, whereas taking advantage of physical network section and node heterogeneousness. In the absence of representative real workloads (i.e., the distributions of file chunks in an exceedingly large-scale storage system) within the public domain, we've got investigated the performance of our proposal and compared it against competitory algorithms through synthesized probabilistic distributions of file chunks. The synthesis workloads check the load reconciliation algorithms by making a few storage nodes that area unit heavily loaded. the pc simulation results area unit encouraging, indicating that our planned algorithm performs o.k.. Our proposal is comparable to the centralized rule within the Hadoop HDFS production system and dramatically outperforms the competitory distributed algorithm in in terms of load imbalance issue, movement cost, and recursive overhead. Significantly, our load reconciliation algorithm exhibits a quick convergence rate. The potency and effectiveness of our style area unit more valid by analytical models and a true implementation with a small-scale cluster environment.

VI.REFERENCES

[1] H. Shen and C.-Z. Xu, "Locality-Aware and Churn-Resilient Load BalancingAlgorithms in Structured P2P Networks," *IEEE Trans. ParallelDistrib. Syst.*, vol. 18, no. 6, pp. 849–862, June 2007.

[2] Q. H. Vu, B. C. Ooi, M. Rinard, and K.-L. Tan, "Histogram-Based GlobalLoad Balancing in Structured Peer-to-Peer Systems," *IEEE Trans. Knowl.Data Eng.*, vol. 21, no. 4, pp. 595–608, Apr. 2009.

[3] H.-C. Hsiao, H. Liao, S.-S. Chen, and K.-C. Huang, "Load Balance withImperfect Information in Structured Peer-to-Peer Systems," *IEEE Trans.Parallel Distrib. Syst.*, vol. 22, no. 4, pp. 634–649, Apr. 2011.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guideto the Theory of NP-Completeness*. W.H. Freeman and Co., 1979.

[5] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," RFC3174, Sept. 2001.

[6] M. Raab and A. Steger, "Balls into Bins—A Simple and Tight Analysis,"*LNCS 1518*, pp. 159–170, Oct. 1998.

[7] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-Based Aggregationin Large Dynamic Networks," *ACM Trans. Comput. Syst.*, vol. 23, no. 3,pp. 219–252, Aug. 2005.

[8] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. V. Steen,"Gossip-Based Peer Sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3,Aug. 2007.

[9] H. Sagan, *Space-Filling Curves*, 1st ed. Springer, 1994.

[10] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, andS. Lu, "BCube: A High Performance, Server-Centric Network Architecturefor Modular Data Centers," in *Proc. ACM SIGCOMM'09*, Aug. 2009, pp.63–74.

[11] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly,"Symbiotic Routing in Future Data Centers," in *Proc. ACM SIGCOMM'10*, Aug. 2010, pp. 51–62.

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

[13] S. Iyer, A. Rowstron, and P. Druschel, "Squirrel: A Decentralized Peerto-Peer Web Cache," in *Proc. ACM PODC'02*, July 2002, pp. 213–222.

[14] I. Raicu, I. T. Foster, and P. Beckman, "Making a Case for DistributedFile systems at Exascale," in *Proc. 3rd Int'l Workshop Large-Scale Systemand Application Performance (LSAP'11)*, June 2011, pp. 11–18.