International Journal on Applications in Information and Communication Engineering
Volume 2: Issue 2: February 2016, pp 53-58. www.aetsjournal.com ISSN (Online) : 2394-6237

Energy Minimization of Big Data Processing for Distributed File Systems in Clouds

J.Litiya , M.Senthilkumar

Abstract— Resource Management is an important issue in cloud environment. The emerging cloud computing paradigm provides administrators and IT organizations with tremendous freedom to dynamically migrate virtualized computing services between physical servers in cloud data centers. Virtualization and VM migration capabilities enable the data center to consolidate their computing services and use minimal number of physical servers. VM migration offers great benefits such as load balancing, server consolidation, online maintenance and proactive fault tolerance. Cloud computing offers utility-oriented IT services to users worldwide. Based on a pay-as-you-go model, it enables hosting of pervasive applications from consumer, scientific, and business domains. However, data centers hosting Cloud applications consume huge amounts of electrical energy, contributing to high operational costs and carbon footprints to the environment. Therefore, to need Green Cloud computing solutions that can not only minimize operational costs but also reduce the environmental impact. So that to define an architectural framework and principles for energy-efficient cloud computing. Based on this architecture, to present the vision, open research challenges and resource provisioning and allocation algorithms for energy-efficient management of cloud computing environments. We Introduce MILP(Mixed Integer Linear Programming) for joint optimization of over all network processing cost.

Index Terms — Big Data, Load balance, distributed file systems, cloud computing

I. INTRODUCTION

CLOUD Computing (or cloud for short) is a compelling technology. In clouds, clients can dynamically allocate their resources on-demand without sophisticated deployment and management of resources. Key enabling technologies for clouds include the MapReduce programming paradigm [1], distributed file systems (e.g., [2], [3]), virtualization (e.g., [4], [5]), and so forth. These techniques emphasize scalability, so clouds (e.g., [6]) can be large in scale, and comprising entities can arbitrarily fail and join while maintaining system reliability.

Distributed file systems are key building blocks for cloud computing applications based on the MapReduce programming paradigm. In such file systems, nodes simultaneously serve computing and storage functions; a file is partitioned into a number of chunks allocated in distinct nodes.

J.Litiya M.E (Cse) Sree Sowdambika College Of Engineering, Aruppukottai, Tamilnadu, India.(Email : lidi421@gmail.com)
M.Senthilkumar Asst Prof, Dept Of Computer Science ,Sree Sowdambika College Of Engineering, Aruppukottai, Tamilnadu, India.
(Email : rmsenthik@gmail.com)

MapReduce tasks can be performed in parallel over the nodes. For example, consider a wordcount application that counts the number of distinct words and the frequency of each unique word in a large file. In such an application, a cloud partitions the file into a large number of disjointed and fixed size pieces (or file chunks) and assigns them to different cloud storage nodes (i.e., chunkservers). Each storage node (or node for short) then calculates the frequency of each unique word by scanning and parsing its local file chunks.

In such a distributed file system, the load of a node is typically proportional to the number of file chunks the node possesses [3]. 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 [7], the file chunks are not distributed as uniformly as possible among the nodes. Load balance among storage nodes is a critical function in clouds. In a load balanced cloud, the resources can be well utilized and provisioned, maximizing the performance of MapReduce based applications.

State of the art distributed file systems (e.g., Google GFS [2] and Hadoop HDFS [3]) in clouds rely on central nodes to manage the metadata information of the file systems and to balance the loads of storage nodes based on that metadata. The centralized approach simplifies the design and implementation of a distributed file system. However, recent experience (e.g., [8]) concludes that when the number of storage nodes, the number of files and the number of accesses to files increase linearly, the central nodes (e.g., the master in Google GFS) become a performance bottleneck, as they are unable to accommodate a large number of file accesses due to clients and MapReduce applications. Thus, depending on the central nodes to tackle the load imbalance problem exacerbate their heavy loads. Even with the latest development in distributed file systems the central nodes may still be overloaded. For example, HDFS federation [9] suggests an architecture with multiple namenodes (i.e., the nodes managing the metadata information). Its file system namespace is statically and manually partitioned to a number of namenodes. However, as the workload experienced by the namenodes may change over time and no adaptive workload consolidation and/or migration scheme is offered to balance the loads among the namenodes, any of the namenodes may become the performance bottleneck.

In this paper, we are interested in studying the load rebalancing problem in distributed file systems specialized for large-scale, dynamic and data-intensive clouds. (The terms “rebalance” and “balance” are inter changeable in this paper.) Such a large-scale cloud has hundreds or thousands of nodes (and may reach tens of thousands in the future). 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. Additionally, we aim to reduce network traffic (or movement cost) caused by rebalancing the loads of nodes as much as possible to maximize the network bandwidth available to normal applications. Moreover, as failure is the norm, nodes are newly added to sustain the overall system performance [2], [3], resulting in the heterogeneity of nodes. Exploiting capable nodes to improve the system performance is, thus, demanded.

Specifically, in this study, we suggest offloading the load rebalancing task to storage nodes by having the storage nodes balance their loads spontaneously. This eliminates the dependence on central nodes. The storage nodes are structured as a network based on distributed hash tables (DHTs), e.g., [10], [11], [12]; 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.

In summary, our contributions are threefold as follows:

- By leveraging DHTs, we present a load rebalancing algorithm for distributing file chunks as uniformly as possible and minimizing the movement cost as much as possible. Particularly, our proposed algo- rithm operates in a distributed manner in which nodes perform their load-balancing tasks indepen- dently without synchronization or global knowledge regarding the system.
- Load-balancing algorithms based on DHTs have been extensively studied (e.g., [13], [14], [15], [16], [17], [18], [19], [20], [21], [22]). However, most existing solutions are designed without considering both movement cost and node heterogeneity and may introduce significant maintenance network traffic to the DHTs. In contrast, our proposal not only takes advantage of physical network locality in the reallocation of file chunks to reduce the move- ment cost but also exploits capable nodes to improve the overall system performance. Additionally, our algorithm reduces algorithmic overhead introduced to the DHTs as much as possible.
- Our proposal is assessed through computer simula- tions. The simulation results indicate that although each node performs our load rebalancing algorithm independently without acquiring global knowledge, our proposal is comparable with the centralized approach in Hadoop HDFS [3] and remarkably outperforms the competing distributed algorithm in [14] in terms of load imbalance factor, movement cost, and algorithmic overhead. Additionally, our load-balancing algorithm exhibits a fast convergence rate. We derive analytical models to validate the efficiency and effectiveness of our design. Moreover, we have implemented our load-balancing algorithm in HDFS and investigated its performance in a cluster environment.

The remainder of the paper is organized as follows: the load rebalancing problem is formally specified in Section 2. Our load-balancing algorithm is presented in Section 3. We evaluate our proposal through computer simulations and discuss the simulation results in Section 4. In Section 5, the performance of our proposal is further investigated in a cluster environment. Our study is summarized in Section 6. Due to space limitation, we defer the extensive discussion of related works in the appendix, which can be found on the Computer Society Digital Library at http://doi. ieeecomputersociety.org/10.1109/TPDS.2012.196.

II. LOAD REBALANCING PROBLEM

We consider a large-scale distributed file system consisting of a set of chunkservers V in a cloud, where the cardinality of V is jV j ¼ n. Typically, n can be 1,000, 10,000, or more. In the system, a number of files are stored in the n chunkservers. First, let us denote the set of files as F . Each file f 2 F is partitioned into a number of disjointed, fixed- size chunks denoted by Cf For example, each chunk has the same size, 64 Mbytes, in Hadoop HDFS [3]. Second, the load of a chunkserver is proportional to the number of chunks hosted by the server [3]. Third, node failure is the norm in such a distributed system, and the chunkservers may be upgraded, replaced and added in the system. Finally, 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. Fig. 1 illustrates an example of the load rebalancing problem with the assumption that the chunkservers are homogeneous and have the same capacity.

Our objective in the current study is to design a load rebalancing algorithm to reallocate file chunks such that the chunks can be distributed to the system as uniformly as possible while reducing the movement cost as much as possible. Here, the movement cost is defined as the number of chunks migrated to balance the loads of the chunkser- vers. Let A be the ideal number of chunks that any chunkserver i 2 V is required to manage in a system-wide load-balanced state, that is where L_i denotes the load of node i (i.e., the number of file chunks hosted by i) and k k represents the absolute value function. Note that "chunkservers" and "nodes" are interchangeable in this paper.

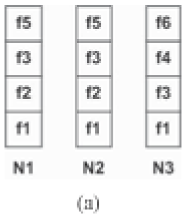
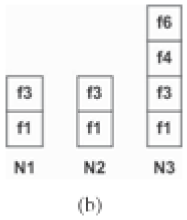
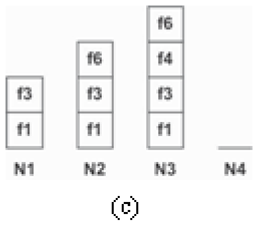


Fig. 1. An example illustrates the load rebalancing problem,where (a) an initial distribution of chunks of six files f_1, f_2, f_3, f_4, f_5, and f_6 in three nodes N1 , N2 , N3.



(b) files f_2 and f_5 are deleted,



(c) f_6 is appended, and

(d) node N_4 joins. The nodes in (b), (c), and (d) are in a load-imbalanced state.

Theorem 1. The load rebalancing problem is N P-hard.

Proof. By restriction, an instance of the decision version of the load rebalancing problem is the knapsack problem [23]. That is, consider any node i 2 V. i seeks to store a subset of the file chunks in F such that the number of chunks hosted by i is not more than A, and the "value" of the chunks hosted is at least , which is defined as the inverse of the sum of the movement cost caused by the migrated chunks. tu

To simplify the discussion, we first assume a homo- geneous environment, where migrating a file chunk between any two nodes takes a unit movement cost and each chunkserver has the identical storage capacity. How- ever, we will later deal with the practical considerations of node capacity heterogeneity and movement

successor of chunkserver n as chunkserver 1. In a typical DHT, a chunkserver i hosts the file chunks whose handles the chunks whose handles are in ðn ; n .

To discover a file chunk, the DHT lookup operation is performed. In most DHTs, the average number of nodes visited for a lookup is Oðlog nÞ [10], [11] if each chunkserver i maintains $\log_2$ n neighbors, that is, nodes i þ 2^k mod n for k ¼ 0; 1; 2; . . ., n Log 2 1 Log 2

Among the n neighbors the one i þ 2^0 is the successor of i. To look up a file with l cost based on chunk migration in physical network locality.

III. OUR PROPOSAL

Table 1 in Appendix B, which is available in the online supplemental material, summarizes the notations frequently used in the following discussions for ease of reference.

A. Architecture

The chunkservers in our proposal are organized as a DHT network; that is, each chunkserver implements a DHT protocol such as Chord [10] or Pastry [11]. A file in the system is partitioned into a number of fixed-size chunks, and "each" chunk has a unique chunk handle (or chunk identifier) named with a globally known hash function such as SHA1

[24]. The hash function returns a unique identifier for a given file's pathname string and a chunk index. For example, the identifiers of the first and third chunks of file "/user/tom/tmp/a.log" are, respectively, SHA1(/ user/tom/tmp/a.log, 0) and SHA1(/user/tom/ tmp/a.log, 2). Each chunkserver also has a unique ID. We represent the IDs of the chunkservers in V by chunks, l lookups are issued.

DHTs are used in our proposal for the following reasons:

- The chunkservers self-configure and self-heal in our proposal because of their arrivals, departures, and failures, simplifying the system provisioning and management. Specifically, 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. Our proposal heavily depends on the node arrival and departure operations to migrate file chunks among nodes. Interested readers are referred to [10], [11] for the details of the self-management technique in DHTs.
- While lookups take a modest delay by visiting Oðlog nÞ nodes in a typical DHT, the lookup latency can be reduced because discovering the l chunks of a file can be performed in parallel. On the other hand, our proposal is independent of the DHT protocols. To further reduce the lookup latency, we can adopt state-of-

the-art DHTs such as Amazon's Dynamo in [12] that offer one-hop lookup delay.

– – – . The DHT network is transparent to the metadata management in our proposal. While the DHT net- work specifies the locations of chunks, our proposal can be integrated with existing large-scale distribu- ted file systems, e.g., Google GFS [2] and Hadoop HDFS [3], in which a centralized master node manages the namespace of the file system and the mapping of file chunks to storage nodes. Specifically, to incorporate our proposal with the master node in GFS, each chunkserver periodically piggybacks its locally hosted chunks' information to the master in a heartbeat message [2] so that the master can gather n ; n ; n ; · · ; n ; for short, denote the n chunkservers as the locations of chunks in the system.

B. Load Rebalancing Algorithm

1) Overview

A large-scale distributed file system is in a load-balanced state if each chunkserver hosts no more than A chunks. In our proposed algorithm, each chunkserver node i first. Low movement cost. As node i is the lightest node among all chunkservers, the number of chunks migrate.

2) MILP Algorithm.

An integer programming problem is a mathematical optimization or feasibility program in which some or all of the variables are restricted to be integers. In many settings the term refers to integer linear programming (ILP), in which the objective function and the constraints (other than the integer constraints) are linear.

A mixed-integer linear program is a problem with

- Linear objective function, $f^T x$, where *f* is a column vector of constants, and *x* is the column vector of unknowns
- Bounds and linear constraints, but no nonlinear constraints (for definitions,
- Restrictions on some components of *x* to have integer values

In mathematical terms, given vectors *f*, *lb*, and *ub*, matrices *A* and *Aeq*, corresponding vectors *b* and *beq*, and a set of indices intcon, find a vector *x* to solve

n mathematical terms, given vectors *f*, *lb*, and *ub*, matrices *A* and *Aeq*, con

$$\min_x f^T x \text{ subject to } \begin{cases} x(\text{intcon}) \text{ are integers} \\ A \cdot x \leq b \\ Aeq \cdot x = beq \\ lb \leq x \leq ub. \end{cases}$$

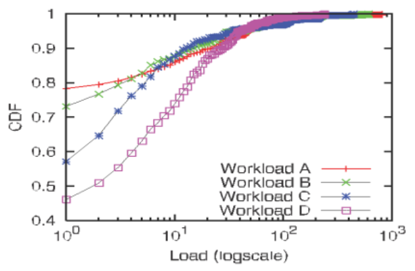


Fig.3. Effect of hetrogenity

Replica management in distributed systems has been extensively discussed in the literature. Given any file chunk, our proposal implements the directory-based scheme in [32] to trace the locations of k replicas for the file chunk. Precisely, the file chunk is associated with k 1 pointers that keep track of k 1 randomly selected nodes storing the replicas.

IV. SIMULATION

A. Simulation Setup and Workloads

The performance of our algorithm is evaluated through computer simulations. Our simulator is implemented with Pthreads. In the simulations, we carry out our proposal based on the Chord DHT protocol [10] and the gossip-based aggregation protocol in [26] and [27]. In the default setting, the number of nodes in the system is n ¼ 1;000, and the number of file chunks is m ¼ 10;000. To the best of our knowledge, there are no representative realistic workloads available. Thus, the number of file chunks initially hosted by a node in our simulations follows the geometric distribu- tion, enabling stress tests as suggested in [15] for various load rebalancing algorithms. Fig. 3 shows the cumulative distribution functions (CDF) of the file chunks in the simulations, where workloads A, B, C, and D represent results indicate that centralized matching introduces much less message overhead than distributed match- ing and our proposal, as each node in centralized matching simply informs the centralized load balancer of its load and capacity. On the contrary, in distributed matching and our proposal, each node probes a number of existing nodes in the system, and may then reallocate its load from/to the probed nodes, introducing more mes- sages. We also see that our proposal clearly produces less message overhead than distributed computing. Speci- fically, any node i in our proposal gathers partial system knowledge from its neighbors [26], [27], whereas node i in distributed matching takes Oðlog nÞ messages to probe a randomly selected node in the network.

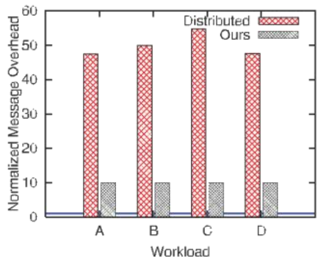


Fig.4.Message over head

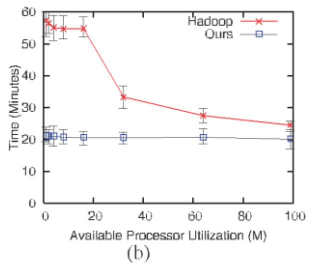


Fig..5.The Breakdown of WMC

physically closest node to pair with, leading a shorter physical distance for migrating a chunk. This operation effectively differentiates nodes in different network locations, and considerably reduces the WMC.

16 256 8

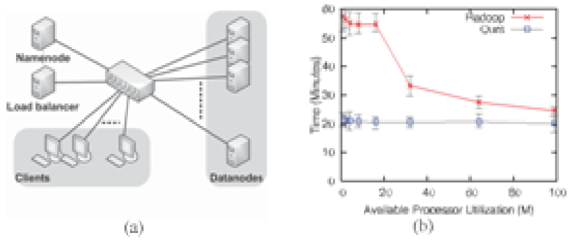


Fig 6. The experimental environment and performance results, where (a) shows the setup of the experimental environment, (b) indicates the time elapsed of performing the HDFS load balancer and our proposal, and (c) and (d) show the distributions of file chunks for the HDFS load balancer and our proposal, respectively.

256

V.IMPLEMENTATION AND MEASUREMENT

A. *Experimental Environment Setup*

We have implemented our proposal in Hadoop HDFS 0.21.0, and assessed our implementation against the load balancer in HDFS. Our implementation is demonstrated through a small-scale cluster environment (Fig. 11a) consisting of a single, dedicated namenode and 25 datanodes, each with Ubuntu 10.10 [34]. Specifically, the namenode is equipped with Intel Core 2 Duo E7400 processor and 3 Gbytes RAM. As the number of file chunks in our experimental environment is small, the RAM size of the namenode is sufficient to cache the entire namenode process and the metadata information, including the directories and the locations of file chunks.

In the experimental environment, a number of clients are established to issue requests to the namenode. The requests include commands to create directories with randomly designated names, to remove directories arbitrarily chosen, etc. Due to the scarce resources in our environment, we have deployed 4 clients to generate requests to the name- node. However, this cannot overload the namenode to mimic the situation as reported in [8]. To emulate the load of the namenode in a production system and investigate the effect of the namenode’s load on the performance of a load-balancing algorithm, we additionally limit the processor cycles available to the namenode by varying the maximum processor utilization, denoted by M, available to the namenode up to M ¼ 1%; 2%; 8%; 16%; 32%; 64%; 99%. The lower processor availability to the namenode represents the less CPU cycles that the namenode can allocate to handle the clients’ requests and to talk to the load balancer.

As data center networks proposed recently (e.g., [29]) can offer a fully bisection bandwidth, the total number of chunks scattered in the file system in our experiments is limited to 256 such that the network bandwidth in our environment (i.e., all nodes are chunk in the experiments is set to 16 Mbytes. Compared to each experimental run requiring

20-60 minutes, transferring these chunks takes no more than

100 328 seconds 5:5 minutes in case the network bandwidth is fully utilized. The initial placement of the

256 file chunks follows the geometric distribution as discussed in Section 4.

For each experimental run, we quantity the time elapsed to complete the load-balancing algorithms, including the HDFS load balancer and our proposal. We perform 20 runs for a Experimental Results

We demonstrate in Fig. 11 the experimental results. Fig. 11b shows the time required for performing the HDFS load balancer and our proposal. Our proposal clearly outper- forms the HDFS load balancer. When the namenode is heavily loaded (i.e., small M’s), our proposal remarkably performs better than the HDFS load balancer. For example, if M ¼ 1%, the HDFS load balancer takes approximately 60 minutes to balance the loads of datanodes. By contrast, our proposal takes nearly 20 minutes in the case of M ¼ 1%. Specifically, unlike the HDFS load balancer, our proposal is independent of the load of the namenode.In Figs. 11c and 11d, we further show the distributions of chunks after performing the HDFS load balancer and our proposal. As there are 256 file chunks and 25 datanodes, the ideal number of chunks that each datanode needs to host. Due to space limitation, we only offer the experimental results for M ¼ 1 and the results for M ¼ 1 conclude the similar. Figs. 11c and 11d indicate that our proposal is comparable to the HDFS load balancer, and balances the loads of datanodes, effectively.

VI. SUMMARY

A novel load-balancing algorithm to deal with the load rebalancing problem in large-scale, dynamic, and distrib- uted file systems in clouds has been presented in this paper. Our proposal strives to balance the loads of nodes and reduce the demanded movement cost as much as possible, while taking advantage of physical network locality and node heterogeneity. In the absence of representative real workloads investigated the performance of our proposal and com pared it against competing algorithms through synthesized probabilistic distributions of file chunks. The synthesis workloads stress test the load-balancing algorithms by creating a few storage nodes that are heavily loaded. The computer simulation results are encouraging, indicating that our proposed algorithm performs very well. Our proposal is comparable to the centralized algorithm in the Hadoop HDFS production system and dramatically out- performs the competing distributed algorithm in [14] in terms of load imbalance factor, movement cost, and algorithmic overhead. Particularly, our load-balancing algorithm exhibits a fast convergence rate. The efficiency and effectiveness of our design are further validated by analytical models and a real implementation with a small scale cluster environment.

ACKNOWLEDGMENTS

The authors are grateful to the anonymous reviewers who have provided us with valuable comments to improve their study. Hung-Chang Hsiao and Chung-Hsueh Yi were partially supported by Taiwan National Science Council under Grants 100-2221-E-006-193 and 101-2221-E-006-097, and by the Ministry of Education, Taiwan, under the NCKU Project of Promoting Academic Excellence & Developing World Class Research Centers. Haiying Shen was sup- ported in part by US National Science Foundation (NSF) grants CNS-1254006, CNS-1249603, OCI-1064230, CNS-1049947, CNS-1156875, CNS-0917056 and CNS-1057530, CNS-1025652, CNS-0938189, CSR-2008826, CSR-2008827, Microsoft Research Faculty Fellowship 8300751, and the US Department of Energy's Oak Ridge National Laboratory including the Extreme Scale Systems Center located at ORNL and DoD 4000111689.

REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Proces- sing on Large Clusters," Proc. Sixth Symp. Operating System Design and Implementation (OSDI '04), pp. 137-150, Dec. 2004.

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

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

[4] VMware, http://www.vmware.com/, 2012.

[5] Xen, http://www.xen.org/, 2012.

[6] Apache Hadoop, http://hadoop.apache.org/, 2012.

[7] Hadoop Distributed File System "Rebalancing Blocks," http:// developer.yahoo.com/hadoop/tutorial/module2.html#rebalan- cing, 2012.

[8] K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward," Comm. ACM, vol. 53, no. 3, pp. 42-49, Jan. 2010.

[9] HDFS Federation, http://hadoop.apache.org/common/docs/ r0.23.0/hadoop-yarn/hadoop-yarn-site/Federation.html, 2012.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] P. Ganesan, M. Bawa, and H. Garcia-Molina, "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Sys- tems," Proc. 13th Int'l Conf. Very Large Data Bases (VLDB '04), pp. 444-455, Sept. 2004.

[18] J.W. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing for Distributed Hash Tables," Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '03), pp. 80-87, Feb. 2003.

[19] G.S. Manku, "Balanced Binary Trees for ID Management and Load Balance in Distributed Hash Tables," Proc. 23rd ACM Symp. Principles Distributed Computing (PODC '04), pp. 197-205, July 2004.

[20] A. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting calable Multi-Attribute Range Queries," Proc. ACM SIGCOMM '04, pp. 353-366, Aug. 2004.

[21] Y. Zhu and Y. Hu, "Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems," IEEE Trans. Parallel and Distributed Systems, vol. 16, no. 4, pp. 349-361, Apr. 2005.

[22] 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.