BALANCING BLOCKS FOR DISTRIBUTED FILE SYSTEMS IN CLOUD

Harika Pratibha Kovvuri[1], Chinabusi Koppula[2]

1. M.Tech Scholar, Department of CSE, Kaushik College of Engineering, Visakhapatnam, AP, India.
2. Assistant Professor, Department of CSE, Kaushik College of Engineering, Visakhapatnam, AP,India.

Abstract: Rebalancing for distributed systems such as cloud computing applications are quite common. Applications are approaching with high challenges on how to transfer and where to store and compute data. The most prevalent distributed file systems to deal with these challenges are the Hadoop. File System (HDFS) which is a variant of the Google File System (GFS). However HDFS has two potential problems. The first one is that it depends on a single name node to manage almost all operations of every data block in the file system. As a result it can be a bottleneck resource and a single point of failure. The second potential problem with HDFS is that it depends on TCP to transfer data. As has been cited in many studies TCP takes many rounds before it can send at the full capacity of the links in the cloud. This results in low link utilization and longer downloads times. To overcome these problems of HDFS we present a new distributed file system. Our scheme uses a light weight front end server to connect all requests with many name nodes. This helps distribute load of a single name node to many name nodes. Our second contribution is to use an efficient protocol to send and route data. Our protocol can achieve full link utilization and hence decreased download times. Based on simulation our protocol can outperform HDFS and hence GFS. Emerging distributed file systems in production systems strongly depend on a central node for chunk reallocation. Experimented algorithm is compared against a centralized approach in a production system and a competing distributed solution presented in this part.

Keywords: *Cloud architecture, resource allocation, distributed name node, Hadoop.*

I. Introduction

Cloud computing is the delivery of computing services over the Internet. Cloud services allow individuals and businesses to use software and hardware that are managed by third parties at remote locations. Examples of cloud services include online file storage, social networking sites, webmail, and online business applications. The cloud computing model allows access to information and computer resources from anywhere that a network connection is available. Cloud computing provides a shared pool of resources, including data storage space, networks, computer processing power, and specialized corporate and user applications. The following definition of cloud computing has been developed by the U.S. National Institute of Standards and Technology (NIST) Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

Deployment of cloud services

Cloud services are typically made available via a private cloud, community cloud, public cloud or hybrid cloud. Generally speaking, services provided by a public cloud are offered over the Internet and are owned and operated by a cloud provider. Some examples include services aimed at the general public, such as online photo storage services, e-mail services, or social networking sites. However, services for enterprises can also be offered in a public cloud. In a private cloud, the cloud infrastructure is operated solely for a specific organization, and is managed by the organization or a third party. In a community cloud, the service is shared by several organizations and made available only to those groups. The infrastructure may be owned and operated by the organizations or by a cloud service provider.

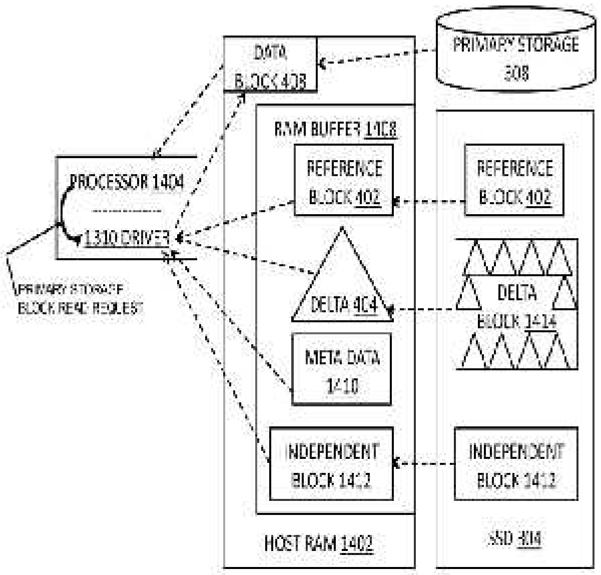
Cloud services are popular because they can reduce the cost and complexity of owning and operating computers and networks. Since cloud users do not have to invest in information technology infrastructure, purchase hardware, or buy software licenses, the benefits are low up-front costs, rapid return on investment, rapid deployment, customization, flexible use, and solutions that can make use of new innovations. In addition, cloud providers that have specialized in a particular area (such as e-mail) can bring advanced services that a single company might not be able to afford or develop. Some other benefits to users include scalability, reliability, and efficiency. Scalability means that cloud computing offers unlimited processing and storage capacity. The cloud is reliable in that it enables access to applications and documents anywhere in the world via the Internet. Cloud computing is often considered efficient because it allows organizations to free up resources to focus on innovation and product development.

A large - scale distributed file system is in a load - balanced state if each chunk server hosts no more than A chunks. 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 of (1−ΔL)A (where 0≤ Δ L < 1).

BASIC ALGORITHMS:

In the basic algorithm, each node implements the gossip-based aggregation protocol in to collect the 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. Using the gossip - based protocol, each node exchanges its locally maintained vector with its neighbors until its vector has s entries. It then calculates the average load of the s nodes denoted by A _ i and regards it as an estimation of A. The nodes perform our load rebalancing algorithm periodically, and they balance their loads and minimize the movement cost in a best - effort fashion

PHYSICAL NETWORK LOCALITY:



A DHT network is an overlay on the application level. The logical proximity abstraction derived from the DHT does not necessarily match the physical proximity information in reality. That means a message traveling between two neighbors in a DHT overlay may travel a long physical distance through several physical network links. In the load balancing algorithm, a light node i may rejoin as a successor of a remote heavy node j. Then, the requested chunks migrated from j to i need to raverse several physical network links, thus generating considerable network traffic and consuming significant network resources (i.e., the buffers in the switches on a communication path for transmitting a file chunk from a source node to a destination node). We improve our proposal by exploiting physical network locality. Basically, instead of collecting a single vector per algorithmic round, each light node i gathers NV vectors..

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 chunkserver. 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,

$$\mathcal{A} = \frac{\sum_{f \in F} |C_f|}{n}.$$

Then, our load rebalancing algorithm aims to minimize the load imbalance factor in each chunk server ***i*** as follows

$$\| L_i - \mathcal{A} \|,$$

where Li denotes the load of node i (i.e., the number of file chunks hosted by i) and k represents the absolute value function. Note that “chunkservers” and “nodes” are interchangeable in this paper

TAKING ADVANTAGE OF NODE HETEROGENEITY:

Nodes participating in the file system are possibly heterogeneous in terms of the numbers of file chunks that the nodes can accommodate. We assume that there is one bottleneck resource for optimization although a node’s capacity in practice should be a function of computational power, network bandwidth and storage space. In the distributed file system for Map Reduce - based applications, the load of a node is typically proportional to the number of file chunks the node possesses. Thus, the rationale of this design is to ensure that the number of file chunks managed by node i is proportional to its capacity

$$\widetilde{\mathcal{A}}_i = \gamma \beta_i,$$

where γ is the load per unit capacity a node should manage in the load balanced state and

$$\gamma = \frac{m}{\sum_{k=1}^{n} \beta_k},$$

where m is the number of file chunks stored in the system. As mentioned previously, in the distributed file system for Map Reduce-based applications, the load of a node is typically proportional to the number of file chunks the node possesses. Thus, the rationale of this design is to ensure that the number of file chunks managed by node i is proportional to its capacity. To estimate the aggregate, our proposal again relies on the gossip-based aggregation protocol in computing the value.

Algorithm 4 in Appendix C, which is available in the online supplemental material, presents the enhancement for Algorithm 1 to exploit node heterogeneity, which is similar to Algorithm 1 and is self-explanatory. If a node i estimates that it is light (i.e., Li < ð1 LÞAei), i then rejoins as a successor of a heavy node j. i seeks j based on its sampled node set V. i sorts the set in accordance with Lt , the load per capacity unit a node currently receives, for all t 2 V. When node i notices that it is the kth least-loaded node (Line 6 in Algorithm 4), it then identifies node j and rejoins as a successor of node j. Node j is the least-loaded node in the set of nodes P V having the minimum cardinality, where 1) the nodes in P are heavy, and 2) the total excess load of nodes in P is not less than Pk jth light node in V j¼1 Ae j (Line 7 in Algorithm 4). Here, Pk jth light node in Vj¼1Aej indicates the sum of loads that the top-k light nodes in V will manage in a load balanced system state.

Managing Replicas 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 underloaded node samples a number of nodes, each selected with a probability of 1n, to share their loads (where n is the total number of storage nodes). Given k replicas for each file chunk (where k is typically a small constant, and k ¼ 3 in GFS), the probability that k0 replicas (k0 k) are placed in an identical node due to migration of our load-balancing algorithm is ð1 nÞk0 independent of their initial locations. For example, in a file system with n ¼ 1;000 storage nodes and k ¼ 3, then the probabilities are only 1106 and 1109 for two and three replicas stored in the same node, respectively. Consequently, the probability of more than one replica appearing in a node due to our proposal is approximately (as k << n)

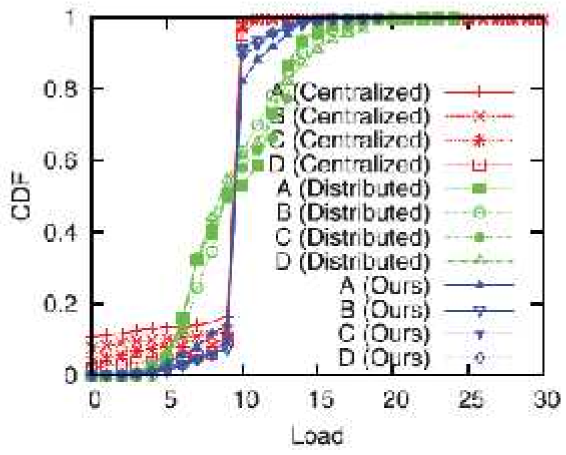
$$\sum_{i=2}^{k}\left(\frac{1}{n}\right)^{i}$$

We have investigated the percentage of nodes storing redundant replicas due to our proposal. In our experiments, the number of file chunks and the number of nodes in the system are m ¼ 10;000 and n ¼ 1;000, respectively. (Details of the experimental settings are discussed in Section 4.) Among the m ¼ 10,000 file chunks, we investigate in the experiment k ¼ 2; 4; 8 replicas for each file chunk; that is, there are 5,000, 2,500 and 1,250 unique chunks in the system, respectively. The experimental results indicate that the number of nodes managing more than one redundant chunk due to our proposal is very small. Specifically, each node maintains no redundant replicas for k ¼ 2; 4, and only 2 percent of nodes store ≥ 2 redundant replicas for k ¼ 8.

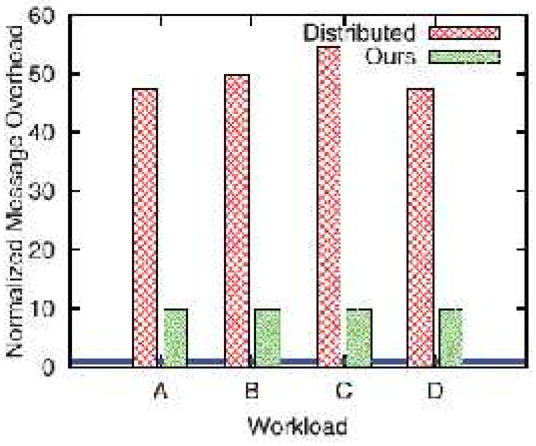
IMPLEMENTATION EXPERIMENTAL SETUP

I have implemented the proposal in Hadoop HDFS 0.21.0, and assessed our implementation against the load balancer in HDFS. The implementation is demonstrated through a small - scale cluster environment consisting of a single, dedicated name node and 25 data nodes, each with Ubuntu10.10]. Specifically, the name node 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 name node is sufficient to cache the entire name node 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 name node. There quests include commands to create directories with randomly designated names, to remove directories arbitrarily chosen, etc. Particularly, the size of a file chunk in the experiments is set to 16 Mbytes. Compared to each experimental run requiring 20 to 60 minutes, transferring these chunks takes no more than 328 seconds ≈ 5.5 minutes in case the network bandwidth is fully utilized. The initial placement of the 256 files chunks

The performance of our algorithm is evaluated through computer simulations. Our simulator is implemented with P threads. In the simulations, we carry out our proposal based on the Chord DHT protocol and the gossip-based aggregation protocol. In the default setting, the number of nodes in the system is n ¼ 1000, 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 distribution, 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 four distinct geometric distributions. Specifically, these distributions indicate that a small number of nodes initially possess a large number of chunks. The four workloads exhibit different variations of the geometric distribution. We have compared our algorithm with the competing algorithms called centralized matching and distributed matching, respectively. In Hadoop HDFS, a standalone load-balancing server (i.e., balancer) is employed to rebalance the loads of storage nodes. The server acquires global information on the file chunks distributed in the system from the name node that manages the metadata of the entire file system. Based on this global knowledge, it partitions the node set into two subsets, where one (denoted by O) contains overloaded nodes, and the other (denoted by U) includes the underloaded nodes. Conceptually, the balancer randomly selects one heavy node i 2 O and one light node j 2 U to reallocate their loads. The reallocation terminates if the balancer cannot find a pair of heavy and light nodes to reallocate their loads. Notably, to exploit physical network locality and thus reduce network traffic, the balancer first pairs i and j if i and j appear in the same rack. If a node in a rack remains unbalanced, and if it cannot find any other node in the same rack to pair, then the node will be matched with another node, in a foreign rack. The balancer in HDFS does not differentiate different locations of foreign racks when performing the matches. In our simulations, each rack has 32 nodes in default. On the contrary, a storage node i in the decentralized random matching algorithm independently and randomly selects another node j to share its load if the ratio of i's load to j's is smaller (or larger) than a predefined threshold (or 1). As suggested by [14], is greater than 0 and not more than 14. In our simulations, ¼ 14. To be comparable, we also implement this algorithm on the Chord DHT. Thus, when node i attempts to share the load of node j, node i needs to leave and rejoin as node j's successor. In our algorithm, we set L ¼ U ¼ 0:2 in default. Each node maintains nV vectors, each consisting of s ¼ 100 random samples of nodes (entries in a vector may be duplicated), for estimating A. nV ¼ 1 in default. The cloud network topology interconnecting the storage nodes simulated is a 2D torus direct network, as suggested by the recent studies in [3] and [4]. (In Appendix E, which is available in the online supplemental material, we also investigate the performance effects on the hyper cube topology in [9].) Finally, unless otherwise noted, each node has an identical capacity in the simulations. Due to



space limitation, we report the major performance results in Section 4.2. Extensive performance results can be found in the appendix, which is available in the online supplemental material, including the effect of varying the number of file chunks (Appendix G, which is available in the online supplemental material), the effect of different numbers of samples (Appendix H, which



is available in the online supplemental material), the effect of different algorithmic rounds (Appendix I, which is available in the online supplemental material) and the effect of system dynamics (Appendix J, which is available in the online supplemental material). In the experimental environment, a number of clients are established to issue requests to the name node. The requests include commands to create directories with randomly designated names, to remove directories arbitrarily chosen, etc

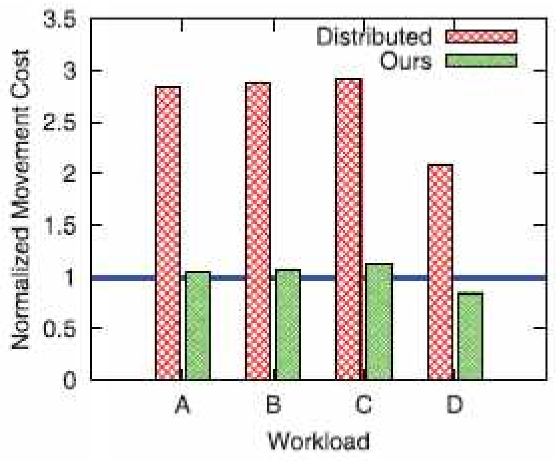
Execution Results

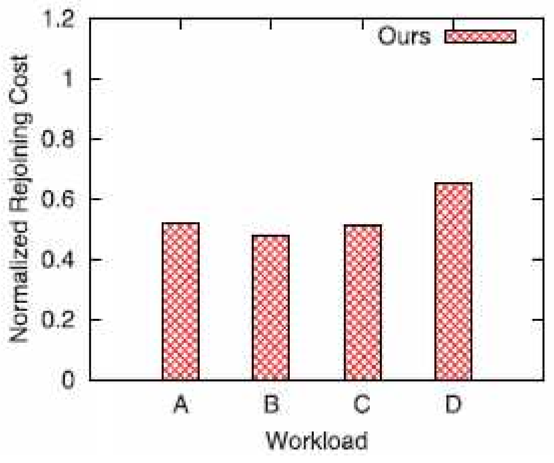
Presents the simulation results of the load distribution after performing the investigated load-balancing algorithms. Here, the nodes simulated have identical capacity. The simulation results show that centralized matching performs very well as the load balancer gathers the global information from the name node managing the entire file system. Since A ¼ 10 is the ideal number of file chunks a node should manage in a load-balanced state, in centralized matching, most nodes have 10 chunks. In contrast, distributed matching performs worse than centralized matching and our proposal. This is because each node randomly probes other nodes without global knowledge about the system. Although our proposal is distributed and need not require each node to obtain global system knowledge, it is comparable with centralized matching and remarkably outperforms distributed matching in terms of load imbalance factor. Fig shows the movement costs of centralized matching, distributed matching, and our algorithm, where the movement costs have been normalized to that of centralized matching (indicated by the horizontal line in the figure). Clearly, the movement cost of our proposal is only 0.37 times the cost of distributed matching. Our algorithm matches the top least-loaded light nodes with the top most-loaded heavy nodes, leading to a fewer number of file chunks migrated. In contrast, in distributed matching, a heavy node i may be requested to relieve another node j with a relatively heavier load, resulting in the migration of a large number of chunks originally hosted by i to i’s successor. We also observe that our proposal may incur slightly more movement cost than that of centralized matching. This is because in our proposal, a light node needs to shed its load to its successor. The total number of messages generated by a load rebalancing algorithm, where the message overheads in distributed matching and our proposal are normalized to that of centralized matching. The simulation results indicate that centralized matching introduces much less message overhead than distributed matching 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 messages. We also see that our proposal clearly produces less message overhead than distributed computing. Specifically, 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. Both distributed matching [14] and our proposal depend on the Chord DHT network in the simulations. However, nodes may leave and rejoin the DHT network for load rebalancing, thus increasing the overhead required to maintain the DHT structure. Thus, we further investigate the number of rejoining operations. Note that centralized matching introduces no rejoining overhead because a node in centralized matching does not need to self-organize and self-heal for rejoining operations. Illustrates the simulation results, where the number of rejoining operations caused by our algorithm is normalized to that of distributed matching (indicated by the horizontal line). We see that the number of rejoining operations in distributed matching can be up to two times greater than that of our algorithm. This is because a heavy node in distributed matching may leave and rejoin the

$$\sum_{\text{chunk } i \in \mathcal{M}} \text{size}_i \times \text{link}_i$$

network to reduce the load of another heavy node. On the contrary, in our proposal, only light nodes rejoin the system as successors of heavy nodes. Our algorithm attempts to pair light and heavy nodes precisely, thus reducing the number of rejoining operations. In Section 3.2.3, we improve our basic load rebalancing algorithm by exploiting physical network locality. The network traffic introduced by centralized matching.

Distributed matching, and our proposal is thus investigated. Specifically, we define the weighted movement cost (WMC) as follows.





where M denotes the set of file chunks selected for reallocation by a load rebalancing algorithm, size i is the size of file chunk i, and link i represents the number of physical links chunk i traverses. In the simulations, the size of each file chunk is identical. We assume that size i ¼ 1 for all i2Mwithout loss of generality. Hence, based on (7), the greater the WMC, the more physical network links used for load reallocation

IMPLEMENTATION AND MEASUREMENT

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 name node and 25 data nodes, each with U buntu 10.10. Specifically, the name node 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 namenode. 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 namenodes load on the performance of a loadbalancing 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 M¼1%; 2%; 8%; 16%; 32%; 64%; 99%. The lower processor availability to the name node represents the less CPU cycles that the name node can allocate to handle the clients' requests and to talk to the load balancer.

As data center networks proposed recently (e.g., [9]) 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 connected with a 100 Mbps fast Ethernet switch) is not the performance bottleneck. Particularly, the size of a file 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 16 256 8100 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 given M and average the time required for executing a loadbalancing algorithm. Additionally, the 5- and 95-percentiles are reported. For our proposal, we let U ¼ L ¼ 0:2. Each datanode performs 10 random samples. Note that 1) in the experimental results discussed later, we favor HDFS by dedicating a standalone node to perform the HDFS load-balancing function. By contrast, our proposal excludes the extra, standalone node. 2) The datanodes in our cluster environment are homogeneous, each with Intel Celeron 430 and 3 Gbytes RAM. We, thus, do not study the effect of the node heterogeneity on our proposal. 3) We also do not investigate the effect of network locality on our proposal as the nodes in our environment are only linked with a single switch. Our proposal clearly outperforms the HDFS load balancer. When the name node is heavily loaded (i.e., small M's), our proposal remarkably performs better than the HDFS load balancer. For example, if M = 1%, the HDFS load balancer takes approximately 60 minutes to balance the loads of data nodes. By contrast, our proposal takes nearly 20 minutes in the case of M = 1%. Specifically, unlike the HDFS load balancer, our proposal is independent of the load of the name node

Summary:

A novel load-balancing algorithm to deal with the load rebalancing problem in large-scale, dynamic, and distributed 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 (i.e., the distributions of file chunks in a large scale storage system) in the public domain, we 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 investigated the performance of our proposal and compared 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 outperforms the competing distributed algorithm in [14] interms 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.

Conclusion

Cloud computing offers benefits for organizations and individuals. There are also privacy and security concerns. If you are considering a cloud service, you should think about how your personal information, and that of your customers, can best be protected. Carefully review the terms of service or contracts, and challenge the provider to meet your needs. A novel load balancing algorithm to deal with the load rebalancing problem in large - scale, dynamic and distributed 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 in the public domain, we have investigated the performance of our proposal and compared 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 outperforms the competing distributed algorithm interms 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.

References

1. 1. Hsiao,Tainan Chung, Hsueh-Yi ; Shen, Haiying ,Chao, Yu-Chang," Load Rebalancing for Distributed File Systems in Clouds ", Parallel And Distributed Systems, Vol –24 issue-5, may 2013.

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

3. I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek,F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-PeerLookup Protocol for Internet Applications," IEEE/ACM Trans. Networking, vol. 11, no. 1, pp. 17-21, Feb. 2003.

4. M. Raab and A. Steger, "Balls into Bins-A Simple and Tight Analysis," Proc. Second Int'l Workshop Randomization and Approximation Techniques in Computer Science, pp. 159-170, Oct. 1998.

5. 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, June2007.

6. 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.

7. 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.

8. H.-C. Hsiao, H. Liao, S.-S. Chen, and K.-C. Huang, "Load Balance with Imperfect Information in Structured Peer-to-Peer Systems,"IEEE Trans. Parallel Distributed Systems, vol. 22, no. 4, pp. 634-649,Apr. 2011.

9. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M.V. Steen, "Gossip-Based Peer Sampling," ACM Trans. Computer Systems, vol. 25, no. 3, Aug. 2007.

10. 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

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

12. 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.

13. Q.H. Vu, B.C. Ooi, M. Rinard, and K.-L. Tan, "Histogram-Based Global Load Balancing in Structured Peer-to-Peer Systems," IEEE Trans. Knowledge Data Eng., vol. 21, no. 4, pp. 595-608, Apr. 2009.

14. 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.

15. M. Jelasity, A. Montresor, and O. Babaoglu, “Gossip-Based Aggregation in Large Dynamic Networks,” ACM Trans. Computer Systems, vol. 23, no. 3, pp. 219-252, Aug. 2005.

16. P. Ganesan, M. Bawa, and H. Garcia-Molina, “Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems,”

17. M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., 1979.

18. D. Karger and M. Ruhl, “Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems,” Proc. 16th ACM Symp.Parallel Algorithms and Architectures (SPAA ’04), pp. 36-43, June2004.

19. D. Eastlake and P. Jones, “US Secure Hash Algorithm 1 (SHA1),” RFC 3174, Sept. 2001.

20. 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.

21. S. Iyer, A. Rowstron, and P. Druschel, “Squirrel: A Decentralized Peer-to-Peer Web Cache,” Proc. 21st Ann. Symp. Principles of Distributed Computing (PODC ’02), pp. 213-222, July 2002.

BIOGRAPHIES

Harika Pratibha Kovvuri pursuing her M.Tech in Computer Science and Engineering at **Kaushik College of Engineering**, Visakhapatnam. Her research interest includes Cloud Computing and distributed computing.

Chinabusi Koppula He is currently working as Assistant professor in the Department of C.S.E, Kaushik college of Engineering, Visakhapatnam. Andhra Pradesh, India. His pursued his B.Tech from JNTUK and M.Tech from G.I.E.T. Rajahmundry His research areas of interest include Cloud Computing, Distributed computing, compiler design & Network security. He is associated with assisting of various academic projects among various fields with 5 years of teaching experience and 2 years of industrial experience. To his credit 7 international publications, 2 national publications and 4 workshops .Now he is guiding 5 U.G level and 2 P.G level projects