IJDCST @Oct, Issue- V-2, I-7, SW-09
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)

Distributed Load Rebalancing by using Cloud Computing

[1]B.Trinadh ,[2]Ravi Mathey

[1]M,Tech Student- Vidya Jyothi Institute of Technology. Aziz nagar Hyderabad,

[2] Associate Professor and HOD of CSE Department, Vidya Jyothi Institute of Technology, Hyderabad,

Abstract: A novel load-balancing algorithm to deal with the load rebalancing problem in very large scale dynamic and distributed file systems in clouds. Distributed file systems are key building blocks for cloud computing applications based on the Map Reduce programming paradigm. In such file systems, nodes simultaneously serve computing and storage functions. Files can also be dynamically created, deleted, and appended. This results in load imbalance in a distributed file system; that is, the file chunks are not distributed as uniformly as possible among the nodes. Emerging distributed file systems in production systems strongly depend on a central node for chunk reallocation. This dependence is clearly inadequate in a large-scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size, and may thus become the performance bottleneck and the single point of failure. In this paper, a fully distributed load rebalancing algorithm is presented to cope with the load imbalance problem. 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 resulting in the heterogeneity of nodes. Exploiting capable nodes to improve the system performance is thus demanded.

Keyword--Load balance, Distributed file systems, Clouds, AES Algorithm

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 Map Reduce programming paradigm [1], distributed file systems (e.g., [3], [4]), 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 Map Reduce 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 so that Map Reduce tasks can be performed in parallel over the nodes. For example, consider a *word count* 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., chunk servers). Each storage node (or *node* for short) then calculates the frequency of each unique word by scanning and parsing its local file chunks. In this paper, the *load rebalancing problem* in distributed file systems specialized for large-scale, dynamic and data-intensive clouds. (The terms “rebalance” and “balance” is interchangeable 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 [3], [4], resulting in the heterogeneity of nodes.

OUR PROPOSAL

The chunk servers in our proposal are organized as a DHT network; that is, each chunk server implements a DHT protocol such as Chord [18] or Pastry [19]. 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" arerespectivelySHA1.

Each chunk server also has a unique ID. We represent the IDs of the chunk servers in V by 1n, 2n, 3n, · · · , nn; for short, denote the n chunk servers as 1, 2, 3, · · · , n. Unless otherwise clearly indicated, we denote the *successor* of chunk server i as chunk server i + 1 and the successor of chunk server n as chunk server 1. In a typical DHT, a chunk server i hosts the file chunks whose handles are within (i−1n , in], except for chunk server n, which manages the chunks whose handles are in (nn, 1n].To discover a file chunk, the DHT lookup operation is performed. In most DHTs, the average number of nodes visited for a lookup is O(log n) [18], [19] if each chunk server maintains log2 n *neighbors*, that is, nodes i + 2k mod n fork = 0, 1, 2, · · , log2 n − 1. Among the log2 n neighbors, the one i+20 is the successor of i. To look up a file with l chunks lookups are issued.

DHTs are used in our proposal for the following reasons:

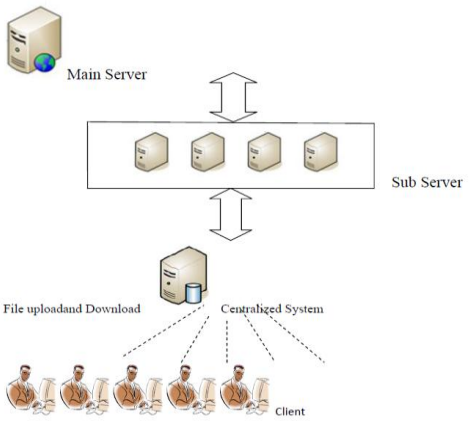
A. The chunk servers self-configure and self-heal in our proposal because of their arrivals, departures, and failures, simplifying the system provisioning and management.

B. if a node leaves, then its locally hosted chunks are reliably migrated to its successor;

C. if a node joins, then it allocates the chunks whose IDs immediately precede the joining node from its successor to manage.

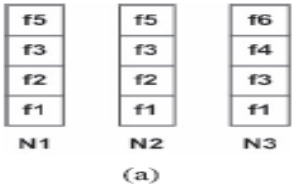
Our proposal heavily depends on the node arrival and departure operations to migrate file chunks among nodes.

ARCHITECTURE:

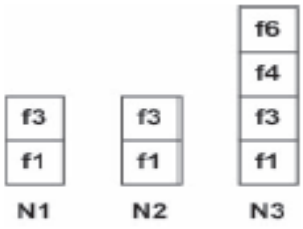


PHYSICAL NETWORK LOCALITY

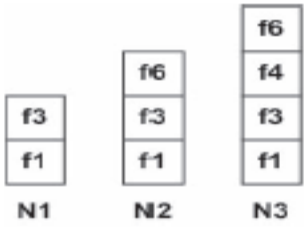
A DHT network is an overlay on the application level. The logical proximity abstraction derived from the DHT does not necessarily match the physical proximity information in reality. That means a message traveling between two neighbors in a DHT overlay may travel a long physical distance through several 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 traverse 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.



IJDCST @Oct, Issue- V-2, I-7, SW-09
ISSN-2320-7884 (Online)
ISSN-2321-0257 (Print)



(b)



(c)

Chunk creation

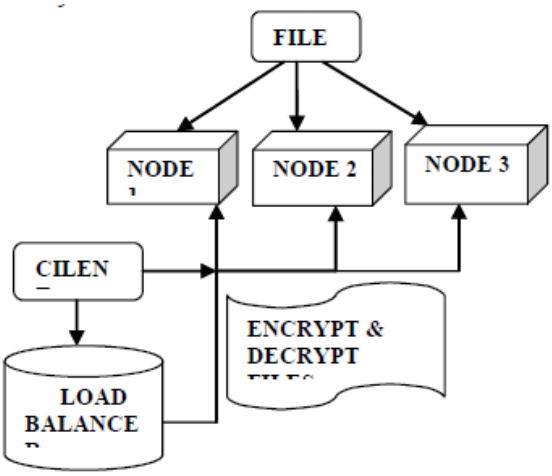
A file is partitioned into a number of chunks allocated in distinct nodes so that Map Reduce Tasks can be performed in parallel over the nodes. The load of a node is typically proportional to the number of file chunks the node possesses. Because the files in a cloud can be arbitrarily created, deleted, 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.

DHT 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.

PROPOSED SYSTEM

The proposed enhance load rebalancing algorithm first evaluates whether the loads are light (under loaded) or heavy (overloaded) in each sub servers without global knowledge. All heavy loads are changed in to light nodes. F are downloading or uploading with the aid of the centralized system. Load equalization technique used to distribute the F uniformly into sub servers.



The advantage of the technique is to reduce latency, isolated overload, and great utilization of resource provident outcome. DHTs enable nodes to self-organize and repair while constantly offering lookup functionality in node dynamism, simplifying the system provision and management. Our algorithm is compared against a centralized approach in a production system which uniformly distributes across sub servers.

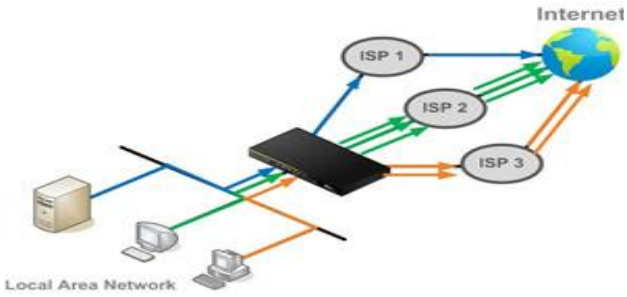
Load balancing Algorithm:

Load balancing algorithms help you easily fine-tune how traffic is distributed across connections. Each deployment has a unique setup, and Peplink's enterprise grade load balancing features can fulfil all of your special requirements. Create your own rule with the following algorithms and you can sit back and enjoy the high performance routing that Peplink brings to you. 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.

Weighted Balance:

Assign more traffic to a faster link or less traffic to a connection with a bandwidth cap. Set a weight on the scale for each connection and outgoing traffic will be proportionally distributed according to the specified ratio. (e.g. 1:3:2)



The time complexity of the above algorithm can be reduced if each light node can know which heavy node it needs to request chunks beforehand, and then all light nodes can balance their loads in parallel. Thus, we extend the algorithm by pairing the top-k1 under loaded nodes with the top-k2 overloaded nodes.

Security

Cloud computing is an emerging technology that is still unclear to many security problems. Ensuring the security of stored data in cloud servers is one of the most challenging issues in such environments. The main aim of this project is to use the cryptography concepts in cloud computing communications and to increase the security of encrypted data in cloud servers with the least consumption of time and cost at the both of encryption and decryption Processes. To make sure the security of data, our proposed a method of providing security by implementing AES algorithm, the encrypted data that will be stored in the sub servers. The key send to user can access original data through this key. Otherwise user can get only cipher text without key.

AES Algorithm:

AES is based on a design principle known as a Substitution permutation network. It is fast in both software and hardware. Unlike its predecessor, DES, AES does not use a Feistel network.AES has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits, whereas Rijndael can be specified with block and key sizes in any multiple of 32 bits, with a minimum of 128 bits. The block size has a maximum of 256 bits, but the key size has no theoretical maximum.AES operates on a 4×4 column-major order matrix of bytes, termed the state (versions of Rijndael with a larger block size have additional columns in the state). Most AES calculations are done in a special finite field. The AES cipher is specified as a number of repetitions of transformation rounds that convert the input plaintext into the final output of cipher text. Each round consists of several processing steps, including one that depends on the encryption key. A set of reverse rounds are applied to transform cipher text back into the original plaintext using the same encryption key.

CONCLUSION

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 have investigated the performance of our proposal and compared it against competing algorithms through synthesized Probabilistic distributions of file chunks. Emerging distributed file systems in production systems strongly depend on a central node for chunk reallocation. This dependence is clearly inadequate in a large-scale, failure-prone

environment because the central load balancer is put under considerable workload that is linearly scaled with the system size, and may thus become the performance bottleneck and the single point of failure. Our algorithm is compared against a centralized approach in a production system and a competing distributed solution presented in the literature. The simulation results indicate that our proposal is comparable with the existing centralized approach and considerably outperforms the prior distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhea a fully distributed load rebalancing algorithm is presented to cope with the load imbalance problem.

FUTURE WORK

In future we have increase efficiency and effectiveness of our design is further validated by analytical models and a real implementation with a small-scale cluster environment. Highly desirable to improve the network efficiency by reducing each user's download time. In contrast to the commonly-held practice focusing on the notion of average capacity, we have shown that both the spatial heterogeneity and the temporal correlation sin the service capacity can significantly increase the average download time of the users in the network, even when the average capacity of the network remains the same.

REFERENCES:

1. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processingon Large Clusters," in *Proc. 6th Symp. Operating System Design and Implementation (OSDI'04)*, Dec. 2004, pp. 137–150.

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

3.Heiser J. What you need to know about cloud computing security and compliance, Gartner, Research, ID Number: G00168345, 2009.

4. Seccombe A.., Hutton A, Meisel A, Windel A, Mohammed A, Licciardi A, (2009). Security guidance for critical areas of focus in cloud computing, v2.1. Cloud Security Alliance, 25 p.

5.Mell P, Grance T (2011) The NIST definition of Cloud Computing. NIST, Special Publication 800–145, Gaithersburg, MD.

6. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in Proc. 6th Symp,Operating System Design and Implementation (OSDI"04), Dec. 2004, pp. 137–150

7. J. Byers, J. Considine, and M. Mitzenmacher, Simple load balancing for distributed hash tables, in Proceedings of IPTPS, Berkeley, CA, Feb. 2003.

8.Ram Prasad Padhy (107CS046), PGoutam Prasad Rao (107CS039)."Load balancing in cloud computing system" Department of Computer Science and Engineering National Institute of Technology, Rourkela Rourkela-769 008, Orissa, India May, 2011.

9.M. Randles, D. Lamb, and A. Taleb-Bendiab, ―A Comparative Study into Distributed Load Balancing Algorithms for Cloud Computing‖, Proceedings of 24th IEEE International Conference on Advanced Information Networking and Applications Workshops, Perth, Australia, April 2010, pages 551-556.

10. S.Penmatsa and T.Chronopoulos, Game-theoretic static load balancing for distributed systems, Journal of Parallel and Distributed Computing, vol.71, no.4, pp.537-555, Apr. 2011.