Load Rebalancing for Distributed File Systems in Clouds

USIKAMALLA SRIKANTH
Department of Computer Science & Engineering, (M.Tech.)
Sindura College of Engineering and Technology
Ramagundam,Telangana

G.LAKSHMI
Asst. Professor,
Department of Computer Science & Engineering (M. Tech.)
Sindura College of Engineering and Technology
Ramagundam,Telangana

K.GEETA
Head of the Department of Computer Science & Engineering (M. Tech.)
Sindura College of Engineering and Technology
Ramagundam,Telangana

Abstract:
In distributed systems protecting the data is become more vulnerable and has to provide the secure to the digital applications. A novel load-balancing algorithm to deal with the load rebalancing problem in 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. Additionally.

Keywords: *Measurement, Security. Denial of service*

1. Introduction

96% of common people used to think that cloud is the best place to store and retrieve the values virtually, and 62% of business entrepreneurs used to think that cloud is the best place to store the content but the case about security from hackers. To make use of these resources we need search mechanisms that distill the information relevant to each user. Normally, such mechanisms require the user to provide a server with a query such as a textual keyword that the server will compare against the documents in some large data set. This model becomes problematic for applications in which the user would like to hide the search criteria. A user might want to protect the privacy of his search queries for a variety of reasons, including protection of commercial interests and personal privacy. Such privacy issues were brought into the spotlight in 2005 when the U.S. Department of Jus- tice subpoenaed records of search terms from popular web search engines. In the current era of digital world, different organizations produce a large amount of sensitive data including personal information, electronic health records, and financial data. The amount of digital data increases at a staggering rate; doubling almost every year and a half [1]. This data needs to be widely distributed and stored for a long time due to operational purposes and regulatory compliance. The local management of such huge amount of data is problematic and costly. While there is an observable drop in the cost of storage hardware, the management of storage has become more complex and represents approximately 75% of the total ownership cost [1]. SaaS offered by CSPs is an emerging solution to mitigate the burden of large local data storage and reduce the maintenance cost via the concept of outsourcing data storage

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 up- graded, 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.

2. Workflows

A workflow is a depiction of a sequence of operations, declared as work of a person, work of a simple or complex mechanism, work of a group of persons, work of an organization of staff, or machines. Workflow may be seen as any abstraction of real work, segregated in work share, work split or whatever types of ordering. For control purposes, workflow may be a view on real work under a chosen aspect, thus serving as a virtual representation of actual work. The flow being described often refers to a document that is being transferred from one step to another .

3. Related Works

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 algorithm operates in a distributed manner in which nodes perform their load-balancing tasks independently without synchronization or global knowledge regarding the system.

Many systems have provided restricted programming models and used the restrictions to parallelize the computation automatically. For example, an associative function can be computed over all of an N element array in log N time on N processors using parallel pre x computations [6, 9, 13]. Map Reduce can be considered a implication and distillation of some of these models based on our experience with large real-world computations. More signi cantly, we provide a fault-tolerant implementation that scales to thousands of processors. In contrast, most of the parallel processing systems have only been implemented on smaller scales and leave the details of handling machine failures to the programmer.

4. Proposed System

The chunk servers 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.

. 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 distributed 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 chunk server periodically piggybacks its locally hosted chunks' information to the master in a heartbeat message [2] so that the master can gather the locations of chunks in the system.

This eliminates the dependence on central nodes. The storage nodes are structured as a network based on distributed hash tables. 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 and a competing distributed solution presented in the literature. The simulation results indicate that although each node performs our load rebalancing algorithm independently without acquiring global knowledge.

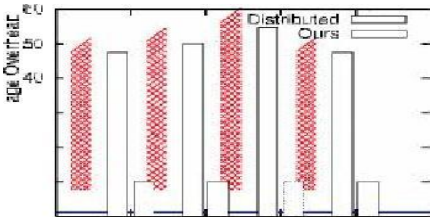
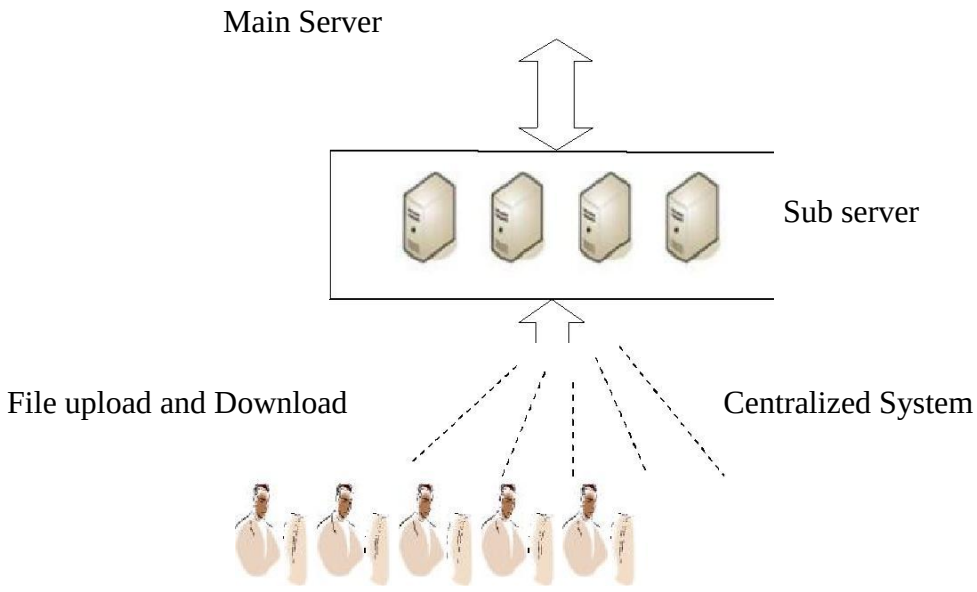


Fig. 1: The message overhead

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 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 be-cause nodes in centralized matching does not need to self-organize and self-heal for rejoining operations. Fig. 1 illustrates the simulation result

5. Architecture



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. 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 name node to mimic the situation as reported 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

6. Conclusions

The Map Reduce programming model has been success- fully used at Google for many different purposes. We attribute this success to several reasons. First, the model is easy to use, even for programmers without experience with parallel and distributed systems, since it hides the details of parallelization, fault-tolerance, locality optimization, and load balancing. Second, a large variety of problems are easily expressible as Map Reduce computations. For example, Map Reduce is used for the generation of data for Google's production web search ser- vice, for sorting, for data mining, for machine learning, and many other systems. Third, we have developed an implementation of Map Reduce that scales to large clusters of machines comprising thousands of machines. The implementation makes client use of these machine re- sources and therefore is suitable for use on many of the large computational problems encountered at Google.

References

1. 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.
2. J.W. Byers, J. Considine, and M. Mitzenmacher, "Simple Load Balancing Distributed Hash Tables," Proc. First Int'l Workshop Peer-to-Peer Systems (IPTPS '03), pp. 80-87, Feb2003
3. John R. Douceur, the sybil attack, international workshop on peer-to- peer systems, 2002.
4. K. McKusick and S. Quinlan, "GFS: Evolution on Fast-Forward," Comm. ACM, vol.53, no3, pp. 42-49, Jan. 2010.
5. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," Proc. Second Int'l Workshop Peer-to-Peer Systems (IPTPS '02), pp. 68-79, Feb.2003