Research Inventy: International Journal of Engineering And Science
Vol.4, Issue 6 (June 2014), PP -63-67
Issn (e): 2278-4721, Issn (p):2319-6483, www.researchinventy.com

A Survey on Hadoop Distributed File System for Cloud Computing Using Rebalancing Algorithm

Gnv Vibhav Reddy[1], S Sampath[2], Raja Babu[3]
[1,2,3] *Computer Science Engineering Department, Sree Dattha Institute of Engineering & Science*

Abstract: *The main objective of the paper is to Enhance distributed load rebalancing algorithm to cope with the load imbalance factor, movement cost, and algorithmic overhead. The load rebalance algorithm is compared against a centralized approach in a production system and the performance of the proposal implemented in the Hadoop distributed file system for cloud computing applications. The cloud applications process large amount of data to provide the desired results. Data volumes to be processed by cloud applications are growing much faster than computing power. This growth demands on new strategies for processing and analyzing the information. The paper explores the use of Hadoop Map Reduce framework to execute scientific workflows in the cloud. Cloud computing provides massive clusters for efficient large computation and data analysis. In such file systems, a file is partitioned into a number of file chunks allocated in distinct nodes so that Map Reduce tasks can perform in parallel over the nodes to make resource utilization effective and to improve the response time of the job. In large failure prone cloud environments files and nodes are dynamically created, replaced and added in the system due to which some of the nodes are over loaded while some others are under loaded. It leads to load imbalance in distributed file system. To overcome this load imbalance problem, a fully distributed Load rebalancing algorithm has been implemented, which is dynamic in nature does not consider the previous state or behavior of the system (global knowledge) and it only depends on the present behavior of the system and estimation of load, comparison of load, stability of different system, performance of system, interaction between the nodes, nature of load to be transferred, selection of nodes and network traffic. The current Hadoop implementation assumes that computing nodes in a cluster are homogeneous in nature.*
Keywords: *Hadoop, Cloud computing, Dynamically, file system*

I. Introduction

Cloud computing refers to delivery of computer resources from a remote place based on user needs. Network connections are necessary to access information and utilize resources. According to Gartner [1], cloud computing can be defined as, a style of computing, where massively scalable IT- enabled capabilities are delivered „as a service" to external customers using Internet technologies. According to the Seccombe [2] and National Institute of Standards & Technology [3], guidelines for cloud computing, it has four different deployment models namely private, community, public and hybrid. Performance, security, data locality to both cloud architects and end users are the key features of public model. Increase in the challenges on how to transfer and where to store and compute data are the issues caused by large distributed file systems in cloud computing.

Cloud Computing Technologies such as Map Reduce paradigm [4], virtualization [5] and Distributed File Systems ([6], [7]) are used to achieve scalability and reliability in clouds. Hadoop File Systems (HDFS) and Google File Systems (GFS) are used to overcome the issues which arise in achieving those factors .HDFS cluster consist of single name node and a server manages the file system namespace and regulates access to files. Load balancing is the main issue in large scale distributed computing. It is the process of distributing tasks to all nodes involved in cloud computing. Efficient allocation of resources to every computing task helps to achieve resource utilization ratio and high user satisfaction. Minimizing resource consumption, avoiding bottlenecks, implementing fail-over, enabling scalability, reducing network inconsistencies and solving network traffic are the main goals of load computing. Whole cloud gets fail while analyzing existing system clouds performance bottleneck due to failure of central node. Functional difficulties and technical difficulties are caused because of those failures. Cloud computing allocate resources dynamically, which connects and add thousands of nodes together. The main goal is to allocate files to these nodes, for avoiding heavy nodes that files are uniformly distributed to these nodes. Load balancing provides maximization of network bandwidth, reduction of network traffic and network inconsistencies.

We can add, delete and update nodes dynamically for heterogeneity of the nodes. Heterogeneity of the nodes will increase the scalability and system performance. In Distributed File System the main functionalities of nodes is to serve computing and storage functions. Cloud computing is a relatively new way of referring to the use of shared computing resources, and it is an alternative to having local servers handle applications. Cloud computing groups together large numbers of computer servers and other resources and typically offers their combined capacity on an on-demand, pay-per-cycle basis without sophisticated deployment and management of

resources. The end users of a cloud computing network usually have no idea where the servers are physically located, they just spin up their application and start working. This flexibility is the key advantage to cloud computing, and what distinguishes it from other forms of grid or utility computing and software as a service (SaaS). The ability to launch new instances of an application with minimal labor and expense allows application providers to scale up and down rapidly, recover from a failure, bring up development or test instances, and roll out new versions to the customer base. Distributed file systems are key building blocks for cloud computing applications based on the Map Reduce J. Deanet all [1] programming paradigm. Map Reduce programs are designed to compute large volumes of data in a parallel fashion. This requires dividing the workload across a large number of machines.

Hadoop provides a systematic way to implement this programming paradigm. The computation takes a set of input key/value pairs and produces a set of output key/value pairs. The computation involves two basic operations: Map and Reduce. The Map operation, written by the user, takes an input pair and produces a set of intermediate key/value pairs. The MapReduce library groups together all intermediate values associated with the same intermediate Key #1 and passes them to the Reduce function. The Reduce function, also written by the user, accepts an intermediate Key #1 and a set of values for that key. It merges together these values to form a possibly smaller set of values. Typically just an output value of 0 or 1 is produced per Reduce invocation. The intermediate values are supplied to the user's Reduce function via an iterator (an object that allows a programmer to traverse through all the elements of a collection regardless of its specific implementation. The proposed fully distributed load rebalancing algorithm can be integrated with the Hadoop [3] Single-Node Cluster or Multi-Node Cluster to enhance the performance of the Name Node in balancing the loads of storage nodes present in the cluster. Figure 1 represents a typical Single-Node Hadoop Cluster.

The three major categories of machine roles in a Client machines, Masters Nodes, and Slave nodes. The Master nodes oversee the two key functional pieces that make up Hadoop: storing lots of data (HDFS), and running parallel computations on all that data (Map Reduce). The Name Node oversees and coordinates the data storage function (HDFS), while the Job Tracker oversees and coordinates the parallel processing of data using Map Reduce. Slave Nodes make up the vast majority of machines and do all the dirty work of storing the data and running the computations. Each slave runs both a Data Node and Task Tracker daemon that communicate with and receive instructions from their master nodes. The Task Tracker daemon is a slave to the Job Tracker, the Data Node daemon a slave to the Name Node. Client machines have Hadoop installed with all the cluster settings, but are neither a Master nor a Slave. Instead, the role of the Client machine is to load data into the cluster, submit Map Reduce jobs describing how that data should be processed and then retrieve or view the results of the job when it's finished. In smaller clusters (~40 nodes) you may have a single physical server playing multiple roles, such as both Job Tracker and Name Node.

II. Related work

Data Processing on Large Clusters

An associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model. The map and reduce primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a map operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a reduce operation to all the values that shared the same key, in order to combine the derived data appropriately.

The functional model with user specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance. Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. 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. The implementation of Map Reduce runs on a large cluster of commodity machines and is highly scalable: a typical Map Reduce computation processes many terabytes of data on thousands of machines.

Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day. The MapReduce programming model has been successfully used at Google for many different purposes. 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 MapReduce computations. For example, MapReduce is used for the generation of data for Google's production web search service, for sorting, for data mining, for machine learning, and many other systems. Third, we have developed an implementation of MapReduce that scales to large clusters of machines comprising thousands of machines. The implementation makes efficient use of these machine resources and therefore is suitable for use on many of the large computational problems encountered at Google. So MapReduce processes many terabytes of data on thousands of machines.

Easy to use scalable programming model for large-scale data processing on clusters. It achieves efficiency through disk-locality and also achieves fault-tolerance through replication. movement. Second, it can avoid transferring loads across high-latency wide area links, thereby enabling fast convergence on load balance and quick response to load imbalance. To use proximity information in load balancing the main contributions are: 1) Relying on a self-organized, fully distributed k-ary tree structure constructed on top of a DHT, load balance is achieved by aligning those two skews in load distribution and node capacity inherent in P2P systems. 2) Proximity information is used to guide virtual server reassignments such that virtual servers are reassigned and transferred between physically close heavily loaded nodes and lightly loaded nodes, thereby minimizing the load movement cost and allowing load balancing to perform efficiently.

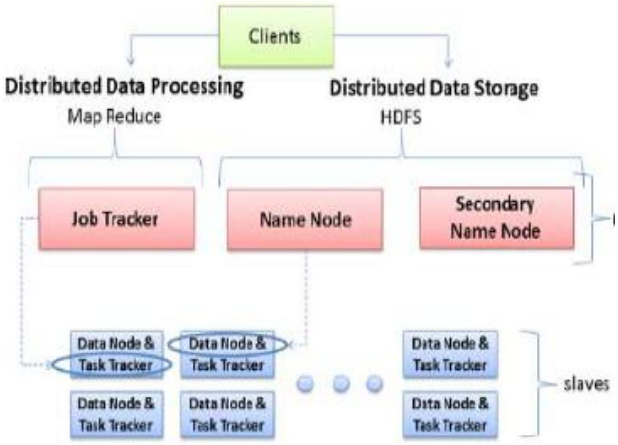

Fig: single node cluster

The load rebalancing problem in distributed file systems specialized for large scale, dynamic and data-intensive clouds. 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.

III. Load Balancing for Distributed Hash Table - Based P2P Systems

DHT based P2P systems offer a distributed hash table (DHT) abstraction for object storage and retrieval. Many solutions have been proposed to tackle the load balancing issue in DHT-based P2P systems. However, all these solutions either ignore the heterogeneity nature of the system, or reassign loads among nodes without considering proximity relationships, or both. To tackle this issue an efficient, proximity-aware load balancing scheme by using the concept of virtual servers. The goal is to ensure fair load distribution over nodes proportional to their capacities, but also to minimize the load-balancing cost (e.g., bandwidth consumption due to load movement) by transferring virtual servers between heavily loaded nodes and lightly loaded nodes in a proximity-aware fashion. To achieve the latter goal by using proximity information to guide virtual server reassignments. There are two main advantages of a proximity-aware load balancing scheme. First, from the system perspective, a load balancing scheme bearing network proximity in mind can reduce the bandwidth consumption (e.g., bisection backbone bandwidth) dedicated to load

IV. Peer-to-Peer Lookup Protocol for Internet Applications

A distributed peer-to-peer applications need to determine the node that stores a data item. The Chord protocol solves this challenging problem in decentralized manner. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data pair at the node to which the key maps. Chord simplifies the design of peer-to-peer systems and applications based on it by addressing these difficult problems: **Load balance:** Chord acts as a distributed hash function, spreading keys evenly over the nodes; this provides a degree of natural load balance.

Decentralization: Chord is fully distributed: no node is more important than any other. This improves robustness and makes Chord appropriate for loosely-organized peer to- peer applications.
Scalability: The cost of a Chord lookup grows as the log of the number of nodes, so even very large systems are feasible. No parameter tuning is required to achieve this scaling.
Availability: Chord automatically adjusts its internal tables to reflect newly joined nodes as well as node failures, ensuring that, barring major failures in the underlying network, the node responsible for a key can always be found. This is true even if the system is in a continuous state of change.
Flexible naming: Chord places no constraints on the structure of the keys it looks up: the Chord key-space is flat. This gives applications a large amount of flexibility in how they map their own names to Chord keys. Chord provides fast distributed computation of a hash function mapping keys to nodes responsible for them. Chord assigns keys to nodes with consistent hashing which has several desirable properties. With high probability the hash function balances load (all nodes receive roughly the same number of keys). Also with high probability, when an Nth node joins (or leaves) the network, only a O(1=N) fraction of the keys are moved to a different location—this is clearly the minimum necessary to maintain a balanced load. Chord improves the scalability of consistent hashing by avoiding the requirement that every node know about every other node. A Chord node needs only a small amount of "routing" information about other nodes. Because this information is distributed, a node resolves the hash function by communicating with other nodes. In an N-node network, each node maintains information about only O(logN) other nodes, and a lookup requires O(logN) messages.

The consistent hash function assigns each node and key an m bit identifier using SHA-1 [10] as a base hash function. A node's identifier is chosen by hashing the node's IP address, while a key identifier is produced by hashing the key. We will use the term "key" to refer to both the original key and its image under the hash function, as the meaning will be clear from context. Similarly, the term "node" will refer to both the node and its identifier under the hash function. The identifier length m must be large enough to make the probability of two nodes or keys hashing to the same identifier negligible. Consistent hashing assigns keys to nodes as follows. Identifiers are ordered on an identifier circle modulo 2m. Key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is called the successor node of key k, denoted by successor(k). If identifiers are represented as a circle of numbers from 0 to 2m- 1, then successor(k) is the first node clockwise from k. In this paper the Chord uses consistent hashing to assign keys to Chord nodes. Consistent hashing tends to balance load, since each node receives roughly the same number of keys, and requires relatively little movement of keys when nodes join and leave the system. Chord will be a valuable component for peer-to-peer, large-scale distributed applications and also adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Chord acts as a distributed hash function, spreading keys evenly over the nodes; this provides a degree of natural load balance.

V. Global Load Balancing in Structured Peer-to-Peer Systems

A new framework, called Histogram-based Global Load Balancing (HiGLOB) to facilitate global load balancing in structured P2P systems. Each node P in HiGLOB has two key components. The first component is a histogram manager that maintains a histogram that reflects a global view of the distribution of the load in the system. The histogram stores statistical information that characterizes the average load of no overlapping groups of nodes in the P2P network. It is used to determine if a node is normally loaded, overloaded, or under loaded. The second component of the system is a load balancing manager that takes actions to redistribute the load whenever a node becomes overloaded or under loaded. The load-balancing manager may redistribute the load both statically when a new node joins the system and dynamically when an existing node in the system becomes overloaded or under loaded. the cost of constructing and maintaining them may be expensive especially in dynamic systems. As a result, we introduce two techniques that reduce the maintenance cost. .

Reduce the cost of constructing histogram. Constructing a histogram for a new node may be expensive since it requires histogram information from all neighbor nodes. Additionally, the histograms of the new node's neighbors also need to be updated since adding a new node to a group of nodes changes the average load of that group. Constructing and maintaining histograms may therefore be expensive if nodes join and leave the system frequently. In light of the fact that every new node in the P2P system must find and notify its neighbor nodes about its existence while these neighbor nodes need to send their information to the new node to setup connections after that, we suggest that histogram information can be piggybacked with messages used in this process. In this way, we can avoid sending separate histogram messages and totally eliminate the effect of node join on the histogram construction of the new node and histogram update of its neighbor nodes. The overhead cost of using histograms is now solely based on histogram update messages caused by changing of load at nodes in the system. Maintaining histograms can be expensive since any load change at a node causes an update to be propagated to all other nodes in the system. To avoid this propagation, we suggest that we do not need to keep exact histogram values. We only need to keep approximate values in the histogram. A node only needs to send load information to other nodes when there is a significant change in either its load or the average load of a non-overlapping group.

VI. Conclusion

In this paper the load rebalancing problem in large-scale, dynamic and distributed file systems in clouds has been presented. This is our first paper in which only the overview of load rebalancing algorithm have been done and we will provide a load balanced cloud, then only the resources can be well utilized and provisioned, maximizing the performance of MapReduce-based applications. The 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. The proposal strives to balance the loads of data nodes and task nodes efficiently. Then only can able to distribute the file chunks as uniformly as possible. The proposed algorithm operates in a distributed manner in which nodes perform their load-balancing tasks independently without synchronization or global knowledge regarding the system. In a loadbalanced cloud, the resources can be well utilized and provisioned, maximizing the performance of MapReduce-based applications. The algorithm also outperforms the competing distributed algorithm in terms of load imbalance factor, movement cost, and algorithmic overhead.

References

[1]. 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.
[2]. 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," in Proc. 21st ACM Symp.
[3]. Hadoop Distributed File System, "RebalancingBlocks," http://develop er.yahoo.com/hadoop/tutorial/module2.html#rebalancing.
[4]. HDFSFederation, http://h adoop.apach e.org/commo n/docs/r0 .23.0/hadoop-yarn/hadoop-yarn-site/Federation.html.
[5]. D. Karger and M. Ruhl, "Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems," in Proc. 16th ACM Symp. Parallel Algorithms and Architectures (SPAA'04), June 2004, pp. 36–43.
[6]. M. Raab and A. Steger, "Balls into Bins—A Simple and Tight Analysis,"LNCS 1518, pp. 159–170, Oct. 1998.
[7]. M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-Based Aggregationin Large Dynamic Networks," ACM Trans. Comput. Syst., vol. 23, no.3,pp. 219–252, Aug. 2005.
[8]. M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. V. Steen,"Gossip-Based Peer Sampling," ACM Trans. Comput. Syst., vol. 25, no. 3,Aug. 2007.
[9]. H. Sagan, Space-Filling Curves, 1st ed. Springer,1994.
[10]. C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, andS. Lu, "BCube: A High Performance, Server-Centric Network Architecturefor Modular Data Centers," in Proc. ACM SIGCOMM'09, Aug. 2009, pp.63–74.
[11]. H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly,"Symbiotic Routing in Future Data Centers," in Proc. ACM SIGCOMM'10,Aug. 2010, pp. 51–62.
[12]. S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, and I. Stoica,"Load Balancing in Dynamic Structured P2P Systems," PerformanceEvaluation,vol. 63, no. 6, pp. 217–240, Mar. 2006.