International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-2 Issue-6, May 2013

Efficient Load Re Balancing Algorithm for Distributed File Systems

Revathy R, A.Illayarajaa

Abstract*—This system presents an innovative idea in cloud computing. In a giant cloud we are able to add thousands of nodes together. The main aim is to allot files to those nodes while not creating significant load to any of the nodes, for that files square measure partitioned off into completely different modules. Another objective is to cut back the network inconsistencies and network traffic attributable to the unbalancing of hundreds. The reduction of network inconsistency can result in maximization of network information measure in order that {so many|numerous|such a big amount of, such a giant amount of, such a lot of} large applications will run in it. Because of quantifiability property we are able to add, delete, update new nodes in order that it supports heterogeneousness of the system. To enhance the potential of nodes we tend to use Distributed file system in Cloud Computing Applications.

Index Terms*—Cloud Computing, distributed Hash tables, load rebalancing.

I. INTRODUCTION

Cloud Computing is a technology, which connects so many nodes together for allocating resources dynamically. Different types of technologies are used in clouds such as Map Reduce programming paradigm, distributed file systems, virtualization. These kinds of techniques are scalable which can add or delete new nodes or systems making it reliable. In a large cloud we can add thousands of nodes together. The main aim is to allocate files to these nodes without making heavy load to any of the nodes, for that files are partitioned into different modules. Another objective is to reduce the network inconsistencies and network traffic because of the unbalancing of loads. The reduction of network inconsistency will lead to maximization of network bandwidth so that so many large applications can run in it. Due to scalability property we can add, delete, update new nodes so that it supports heterogeneity of the system. To improve the capability of nodes we use Distributed file System in Cloud Computing Applications. In such file systems the main functionalities of nodes is to serve computing and storage functions. If we want to store a file into the system firstly we will divide the file into different modules and store it in different nodes. So we introduced new load rebalancing algorithm to avoid all these disadvantages.
When analyzing the existing system clouds rely on central nodes to balance the loads of storage nodes, there comes the performance bottleneck because the failure of central nodes leads to the failure of whole system and it will leads to many technical and functional difficulties**.**

Manuscript published on 30 May 2013.
***Correspondence Author(s)**
Ms. Revathy R, Department Of Computer Science and Engg, Annai Mathammal Sheela Engineering College, Namakkal Dist., Tamil Nadu,India.
Asst. Prof. A Ilayarajaa, Department Of Computer Science and Engg, Annai Mathammal Sheela Engineering College, Namakkal dist., Tamil Nadu, India.

II. LITERATURE SURVEY

In the paper, *Nonuniformity And Load Balance In Distributed Hash Tables*; Existing solutions to balance load in DHTs incur a high overhead either in terms of routing state or in terms of load movement generated by nodes incoming or outgoing the system. During this paper we have a tendency to propose a group of general techniques and use them to develop a protocol supported Chord, called Y0, that achieves load leveling with lowest overhead underneath the everyday assumption that the load is uniformly distributed within the symbol house. Specially, we have a tendency to prove that Y0 are able to do near-optimal load leveling, whereas moving very little load to take care of the balance and increasing the dimensions of the routing tables by at the most a continuing issue. Victimization in depth simulations supported real-world and artificial capability distributions, we have a tendency to show that Y0 reduces the load imbalance of Chord from O (log n) to a butthree.6 while not increasing the amount of links that a node must maintain. Additionally, we have a tendency to study the result of nonuniformity on each DHTs, demonstrating considerably reduced average route length as node capacities become progressively heterogeneous. For a real-world distribution of node capacities, the route length in Y0 is asymptotically but the route length within the case of a homogenous system.
In the paper, *Chord: A Scalable Peer-To-Peer Operation Protocol For Web Application*; An elementary drawback that confronts peer-to-peer applications is that the economical location of the node that stores a desired information item. This paper presents Chord, a distributed operation protocol that addresses this drawback. Chord provides support for only one operation: given a key, it maps the key onto a node. Information location is simply enforced on prime of Chord by associating a key with every information item, and storing the key/data combine at the node to that the key maps. Chord adapts expeditiously as nodes are part of and leave the system, and may answer queries even though the system is incessantly dynamical. Results from theoretical analysis and simulations show that Chord is scalable: communication price and therefore the state maintained by every node scale logarithmically with the amount of Chord nodes.

In the paper, *MapReduce: Simplified Processing On Massive Cluster*; MapReduce may be a programming model AND an associated implementation for processing and generating massive information sets. Users specify a map operate that processes a key/value combine to come up with a group of intermediate key/value pairs, and a cut back operate that merges all intermediate values related to constant intermediate key. Several world tasks are utterab le during this model, as shown within the paper. Programs written during this useful vogue are mechanically parallelized and dead on an oversized cluster of artifact machines. The run-time system takes care of the small print of partitioning the input file, planning the program's execution across a group of machines, handling machine failures, and managing the desired inter-machine communication. This enables programmers with none expertise with parallel and distributed systems to simply utilize the resources of an oversized distributed system. Our implementation of MapReduce runs on an oversized cluster of artifact machines and is very scalable a typical MapReduce computation processes several terabytes of knowledge on thousands of machines. Programmers notice the system straightforward to use. Many MapReduce programs are enforced and upwards of one thousand MapReduce jobs are dead on Google's clusters daily.

Within the paper, easy economical Load leveling Algorithms for Peer-to-Peer Systems; Load leveling may be a vital issue for the economical operation of peer-to-peer networks. We have a tendency to provide 2 new load-leveling protocols whose demonstrable performance guarantees are inside a continuing issue of best. Our protocols refine the consistent hashing arrangement that underlies the Chord (and Koorde) P2P networks. Each preserve Chord's power question time and near-optimal information migration price. Our 1st protocol balances the distribution of the key address house to nodes that yields a load-balanced system once the DHT maps things "randomly" into the address house. To our data, this yields the primary P2P theme at the same time achieving O(logn) degree, O(logn) look-up price, and constant-factor load balance (previous schemes settled for any 2 of the three).Our second protocol aims to directly balance the distribution of things among the nodes.

III. EXISTING SYSTEM

Node failure is the norm in such a distributed system, and the module servers may be upgraded, replaced and added in the system. Let, the set of files as F. Files in F may be arbitrarily created, deleted, and appended. Considering a large-scale distributed file system consisting of a set of module servers M in a cloud. Each file f is partitioned into a number of disjointed, fixed-size modules. Existing solutions to balance load in DHTs incur a high overhead either in terms of routing state or in terms of load movement generated by nodes arriving or departing the system. All DHTs make some effort to load balance, generally by (i) randomizing the DHT address associated with each item with a good enough hash function and (ii) making each DHT node responsible for a balanced portion of the DHT address space. First, the typical random partition of the address space among nodes is not completely balanced. Some nodes end up with a larger portion of the addresses and thus receive a larger portion of the randomly distributed items. An important issue in DHTs is load-balance the even distribution of items (or other load measures) to nodes in the DHT. This results the uneven distribution of module servers.

Limitations of Existing System

Emerging distributed file systems depends on a central node for module reallocation. This dependence is inefficient in an exceedingly large-scale, failure prone atmosphere as a result of the central load balancer is tested underneath sure employment that's linearly scaled with the system size, and will so become the performance bottleneck and therefore the single purpose of failure.

IV. PROPOSED SYSTEM

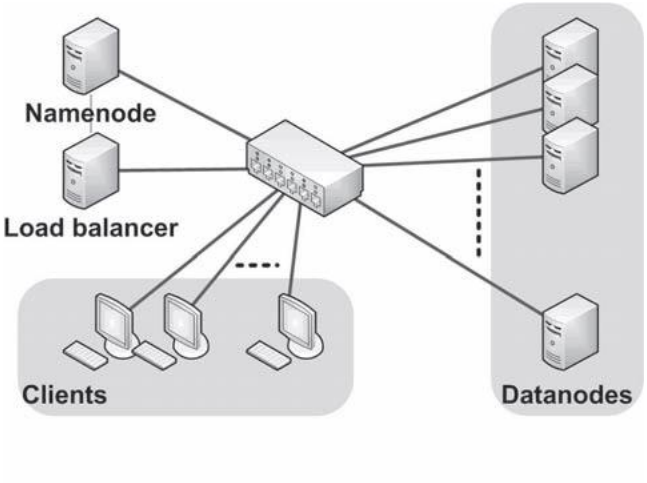

Fig.no.2 Implementation of data distributer sharing system

On the opposite hand, our proposal is freelance of the DHT protocols. The module servers in our proposal area unit organized as a DHTnetwork; that's, every module server implements a DHT protocol like Chord or Pastry. While lookups take a modest delay by visiting O(log n) nodes during a typical DHT, the search latency are often reduced as a result of discovering the l modules of a file are often performed in parallel. In DHTs, if nodes and file modules area unit selected with uniform IDs, the utmost load of a node is sure to be O (log n) times the common during chance of one O(one n), therefore the hundreds of nodes to an exact extent. DHTs area unit utilized in our proposal for the subsequent reasons: The self-configure and Prunella vulgaris in our proposal attributable to their arrivals, departures, and failures, simplifying the system provisioning and management. Specifically, to include our proposal with the master node in GFS, every module server sporadically piggybacks its domestically hosted modules info to the master during a heartbeat message in order that the master will gather the locations of modules within the system. The DHT network is clear to the data management in our proposal. While the DHT network specifies the locations of modules, our proposal are often integrated with existing large-scale distributed file systems, Interested readers area unit said for the small print of the self-management technique in DHTs. Google GFS and Hadoop HDFS, within which a centralized master node manages the namespace of the classification system and therefore the mapping of file modules to storage nodes.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

To any scale back the search latency, we will adopt progressive DHTs like Amazons generator in this provide one-hop search delay. Specifically, typical DHTs guarantee that if a node leaves, then its domestically hosted modules area unit dependably migrated to its successor; if a node joins, then it allocates the modules whose IDs forthwith precede the connexion node from its successor to manage. Our proposal heavily depends on the node arrival and departure operations to migrate file modules among nodes. A go into the system is partitioned off into variety of fixed-size modules, and every module contains a distinctive module handle (or module symbol) named with a globally far-famed hash perform like SHA .The hash perform returns a novel identifier for a given files pathname string and a module index.

A. Load Rebalancing Algorithm

A node is light if the number of modules it hosts is smaller than the threshold as well as, a heavy node manages the number of modules greater than threshold. A large-scale distributed file system is in a load-balanced state if each module server hosts no more than **A** modules. In our proposed algorithm, each module server node I first estimate whether it is under loaded (light) or overloaded (heavy) without global knowledge. This process repeats until all the heavy nodes in the system become light nodes. In Proposed system, file downloading or uploading with the help of the centralized system.

Centralized system will be sharing the file (uploading and downloading). First of all we are going to notice the lightest node to require the set of modules from heaviest node. Thus we will do the method while not failure. Load equalization may be a technique to distribute employment across several computers or network to realize most utilization of resources economical output, reducing latency, and take away overload. The load equalization service is sometimes provided by dedicated code or hardware, like a multilayer switch or name server. During this project we have a tendency to use Load rebalancing formula. Then identical method is dead to unleash the additional load on following heaviest node within the system. Then we are going to once more notice the heaviest and lightest nodes, such a method repeats iteratively till there's not the heaviest.

Advantages of Proposed System

Using this we can use in large scale, failure-prone environment because the central load balancer is put under considerable workload that is linearly scaled with the system size. Another advantage of the proposed system is the security consistency provided by it. Various nodes with heavy loads have been proposed as alternatives to central node module so that so many drawbacks of the existing system can be avoided. The proposed system will help in keeping the system consistent so that we can avoid data loss.

V. IMPLEMENTATION ISSUES

There are four different modules in this system. They are depicted below.

- Module creation
- DHT formulation
- Load balancing algorithm
- Replica Management

Module Creation

Our objective is to allocate the modules of files as uniformly as possible among the nodes such that no node manages an excessive number of modules. A file is partitioned into a number of modules allocated in different nodes so that Map Reduce Tasks can be performed in parallel over the nodes. Because the files in a cloud can be dynamically created, deleted, and appended, and nodes can be upgraded, replaced and added in the file system, the file modules are distributed as uniformly as possible among the nodes.

DHT Formulation

The module servers in our proposal are organized as a DHT network. Typical DHTs guarantee that if a node leaves, then its locally hosted modules are reliably migrated to its successor; if a node joins, then it allocates the modules whose IDs immediately precede the joining node from its successor to manage. DHT's, given that a unique handle (or identifier) is assigned to each file module.

The storage nodes are structured as a network based on distributed hash tables (DHTs), DHTs enable nodes to self-organize and repair while constantly offering lookup functionality in node dynamism, simplifying the system provision and management.

Load Balancing Algorithm

In our proposed algorithm, each module 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 modules it hosts is smaller than the threshold. First of all we will find the lightest node to take the set of modules from heaviest node. So we can do the process without failure. Load balancing is a technique to distribute workload across many computers or network to achieve maximum resource utilization, maximize throughput, minimize response time, and avoid overload. The load equalization service is sometimes provided by dedicated software package or hardware, like a multilayer switch or name server.

Replica Management

Google GFS and Hadoop HDFS), a relentless variety of replicas for every file module square measure maintained in distinct nodes to enhance file handiness with relation to node failures and departures. Our current load equalization rule doesn't treat replicas clearly. It is unlikely that 2 or additional replicas square measure placed in a consistent node owing to the random nature of our load rebalancing rule.

More specifically, every underneath loaded node samples variety of nodes, every elect with a likelihood of 1/n, to share their hundreds (where n is that the total variety of storage nodes).

VI. CONCLUSION

We shouldn't standardize the algorithmic rule Inter-operability isn't a problem devices implementing completely different algorithms will inter-ope rate. Load reconciliation Algorithms ought to be deterministic; a minimum of for a specific flow Frame order should be preserved at intervals a flow. The synthesis workloads assay the load equalization algorithms by making a couple of storage node that square measure heavily

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

loaded. Intermixture sensible algorithms continually works, given a standard receive algorithm!

The potency and effectiveness of our style square measure any valid by analytical models and a true implementation with a small-scale cluster setting.

The computer simulation results square measure encouraging, indicating that our planned algorithmic rule performs fine. This proposal is like the centralized algorithmic rule within the Hadoop HDFS production system and dramatically outperforms the competitor distributed algorithmic rule in terms of load imbalance issue, movement price, and recursive overhead. Our proposal strives to balance the masses of nodes and scale back the demanded movement price the maximum amount as potential, whereas taking advantage of physical network vicinity and node nonuniformity. Leave space for vendors to boost and optimize a completely unique load equalization algorithmic rule to modify the load-rebalancing drawback in large-scale, dynamic, and distributed file systems in clouds has been conferred during this paper. Best algorithmic rule is commonly topology specific.

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, "Rebalancing Blocks," http://developer.yahoo.com/hadoop/tutorial/module2.html#rebalancing.

4 HDFSFederation, http://hadoop.apache.org/common/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.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)