LOAD REBALANCING FOR DISTRIBUTED FILE SYSTEMS IN CLOUD

[1]Rajkamal J
rajkamal.smit@gmail.com
Assistant Professor

[2]Malarvizhi V malarvalluvan5@gmail.com
[3] Rukzana k ruksanasana96@gmail.com
[4]Sandhiya R sandhiyasandy2135@gmail.com
[2][3][4]UG students
Department of Computer Science and Engineering
T.J.S. Engineering College

Abstract

Dynamic resource allocation is an up growing and challenging process in cloud environment key building blocks for cloud computing applications based on map reduce in distributed file system environment are programming paradigm .our algorithm is compared against a centralized approach in a production system and a competing distributed solution presented.

Index Terms – ***Distributed Hash Tables, Chunk creation, replica management.***

1. Introduction

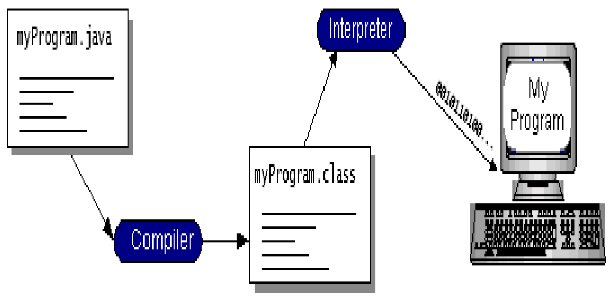
Cloud computing has the scope of moving OS to the web. It can bring about collaboration while solving platform interdependence problems. The important challenges for the transition to utility computing are performance and security. In the long run, performance of the system would only benefit the user with cheaper rates. The security of the system needs a serious check with good intrusion detection mechanisms.

It basically is a cycle where technology triggers infrastructure growth which improves operational efficiency. The efficiency concerns and demands can bring about tough competition which would lower the costs and bring about new advancement in technology. The growth of cloud computing could be something similar to the telecommunication gadgets around. It might even bring an anytime, anywhere computing possible because users don't want to invest in new hardware or software for doing their needs. They can rather use the services provided. This has been further enhanced with the aesthetic needs of consumers and the introduction of good graphics processing software with their hardware requirements.

In the implementation, it is proposed that an open source OS similar to EyeOS be used. EyeOS is an open source cloud computing web desktop. It provides a number of functionalities such as word processor, spreadsheets, text editor and also a process manager. Besides, it provides the user the opportunity to develop his own application.

Depending on whether the machine is hosted by the company or some other provider, clouds are classified as internal or hosted clouds. Clouds are broadly classified as public, private and hybrid clouds. Public clouds are those that are accessible to the general public and everyone can use the services. Private clouds are those that are accessible only to the particular organization. Hybrid clouds are those that are accessible to a group of organizations.

It is proposed that we use private clouds for the financial system. It is due to security and privacy concerns. User privacy is of utmost importance as any leaked information can contain valuable data which important to maintain good security in the system as any accidental modifications may need the entire operation of the system to be stopped and reviewed.



1.1 PROJECT PLATFORM SPECIFICATION STAND-ALONE:

JAVA:

Java is a programming language originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to byte code (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is general-purpose, concurrent, class-based, and object-oriented, and is specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere". Java is considered by many as one of the most influential programming languages of the 20th century, and widely used from application software to web application.

One characteristic of Java is portability, which means that computer programs written in the Java language must run similarly on any supported hardware/operating-system platform.

2.Java Technology

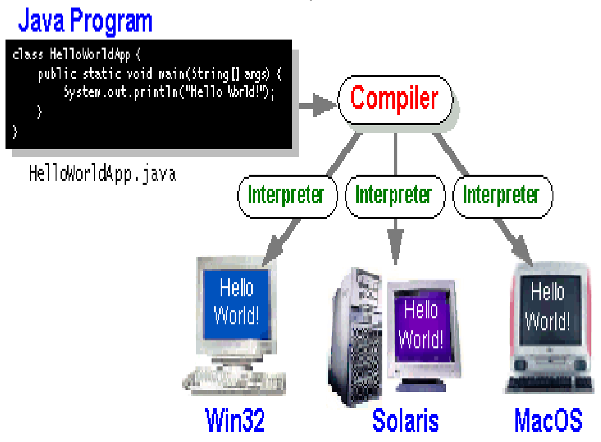
Java technology is both a programming language and a platform.

The Java programming language is a high-level language that can be characterized by all of the following buzzwords:

With most programming languages, you either compile or interpret a program so that you can run it on your computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first you translate a program into an intermediate language called *Java byte codes* —the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

TECHNICAL HIGHLIGHT

JavaFX is based on the concept of a "Common profile" that is intended to span across all devices supported by JavaFX. This approach makes it possible for developers to use a common programming model while building an application targeted for both desktop and mobile devices and to share much of the code, graphics assets and content between desktop and mobile versions.



```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Drag-to-Install: From the point of view of the end user "Drag-to-Install" allows them to drag a JavaFX widget and drop it onto their desktop. The application will not lose its state or context even after the browser is closed. An application can also be re-launched by clicking on a shortcut that gets created automatically on the users desktop.

Integrating graphics created with third-party tools: JavaFX includes a set of plug-ins for Adobe Photoshop and Illustrator that enable advanced graphics to be integrated directly into JavaFX applications. The plug-ins generatesJavaFX Script code that preserves layers and structure of the graphics. Developers can then easily add animation or effects to the static graphics imported.

3.Literature Survey:

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. In this paper, we propose a set of general techniques and use them to develop a protocol based on Chord, called Y0, that achieves load balancing with minimal overhead under the typical assumption that the load is uniformly distributed in the identifier space.

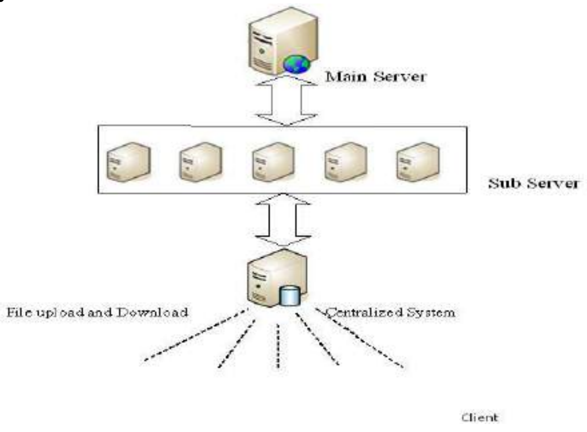
In particular, we prove that Y0 can achieve near-optimal load balancing, while moving little load to maintain thincreasing the size of the routing tables by at most a constant factor. Using extensive simulations based on real-world and synthetic capacity distributions, we show that Y0 reduces the load imbalance of Chord from O (log n) to a less than 3.6 without increasing the number of links that a node needs to maintain. In addition, we study the effect of heterogeneity on both DHTs, demonstrating significantly reduced average route length as node capacities become increasingly heterogeneous. For a real-world distribution of node capacities, the route length in Y0 is asymptotically less than half the the route length in the case of a homogeneous system.

4.EXISTING SYSTEM:

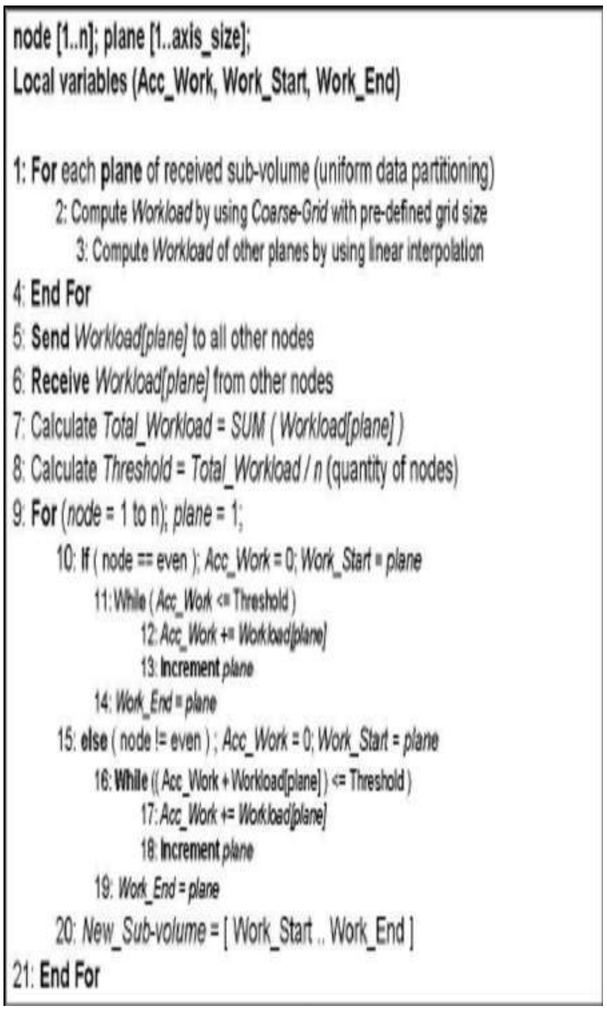
State-of-the-art distributed file systems (e.g., Google GFS and Hadoop HDFS) in clouds rely on central nodes tomanage the metadata information of the file systems and to balance the loads of storage nodes based on that metadata.The centralized approach simplifies the design and implementation of a distributed file system. However, recentexperience concludes that when the number of storage nodes, the number of files and the number of accesses to files increase linearly, the central nodes (e.g., themaster in Google GFS) become a performance bottleneck,as they are unable to accommodate a large number of fileaccesses due to clients and MapReduce applications. Mostexisting solutions are designed without consideringboth movement cost and node heterogeneity andmay introduce significant maintenance networktraffic to the DHTs.

5.PROPOSED SYSTEM:

Our objective is to allocate the chunks of files as uniformlyas possible among the nodes such that no node manages anexcessive number of chunks. we aim toreduce network traffic (or movement cost) caused byrebalancing the loads of nodes as much as possible tomaximize the network bandwidth available to normalapplications. Moreover, as failure is the norm, nodes are
newly added to sustain the overall system performance resulting in the heterogeneity of nodes. we present a load rebalancingalgorithm for distributing file chunks as uniformlyas possible and minimizing the movement cost asmuch as possible. Particularly, our proposed algorithmoperates in a distributed manner in whichnodes perform their load-balancing tasks independentlywithout synchronization or global knowledgeregarding the system.



System architecture



```
node [1..n]; plane [1..axis_size];
Local variables (Acc_Work, Work_Start, Work_End)

1: For each plane of received sub-volume (uniform data partitioning)
    2: Compute Workload by using Coarse-Grid with pre-defined grid size
      3: Compute Workload of other planes by using linear interpolation
4: End For
5: Send Workload[plane] to all other nodes
6: Receive Workload[plane] from other nodes
7: Calculate Total_Workload = SUM ( Workload[plane] )
8: Calculate Threshold = Total_Workload / n (quantity of nodes)
9: For (node = 1 to n); plane = 1;
    10: If ( node == even ); Acc_Work = 0; Work_Start = plane
        11: While ( Acc_Work <= Threshold )
            12: Acc_Work += Workload[plane]
            13: Increment plane
        14: Work_End = plane
    15: else ( node != even ) ; Acc_Work = 0; Work_Start = plane
        16: While (( Acc_Work + Workload[plane] ) <= Threshold )
            17: Acc_Work += Workload[plane]
            18: Increment plane
        19: Work_End = plane
    20: New_Sub-volume = [ Work_Start .. Work_End ]
21: End For
```

AES ALGORITHM:

AES is an iterative rather than Feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES Key.

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −

Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row.

Mix columns

Shift row and mixed columns provide Diffusion to the cipher.Each column is processed separately.Each byte is replaced by value dependent on all 4bytes in the columns.

6.Conclusion:

A novel load-balancing algorithm to deal with the load rebalancing problem in large-scale, dynamic, and distributedfile 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.

7.REFERENCES

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," Proc. Sixth Symp. Operating System Design and Implementation (OSDI '04), pp. 137-150, Dec. 2004.

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

[3] Hadoop Distributed File System, http://hadoop.apache.org/ hdfs/, 2012.

[4] VMware, http://www.vmware.com/, 2012.

[5] Xen, http://www.xen.org/, 2012.

[6]ApacheHadoop, http://hadoop.apache.org/, 2012.

[7] Hadoop Distributed File System "RebalancingBlocks,"http://developer.yahoo.com/hadoop/tutorial/module2.html#rebalaning,2012.