IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

Using HDFS Approach for Practical Performance Analysis

R.Vasavi[1], MD. Moona Tasleem[2], L. Bharath Reddy[3], B. Ramya[4], P. Rojashalini[5]

Assistant Professor, Department of Computer, VNR VJIET, Hyderabad, India[1]

Student, Department of Computer, VNR VJIET, Hyderabad, India[2, 3, 4, 5]

Abstract: Distributed file systems are the key component of any cloud-scale data processing middleware. Evaluating the performance of DFSs is accordingly very important. In this paper, we propose a systematic and practical performance analysis framework, driven by architecture and design models for defining the structure and behaviour of typical master/slave DFSs. Our approach is different from others because 1) most of existing works rely on performance measurements under a variety of workloads/strategies, comparing with other DFSs or running application programs, but our approach is based on architecture and design level models and systematically derived performance models; 2) our approach is able to both qualitatively and quantitatively evaluate the performance of DFSs; and 3) our approach not only can evaluate the overall performance of a DFS but also its components and individual steps. We demonstrate the effectiveness of our approach by evaluating Hadoop distributed file system (HDFS).

Keywords: DFS, HDFS, performance, Map reduce, PVFS.

1. INTRODUCTION

Data Intensive DFSs are any file system that allows multiple users to access to files distributed on multiple machines via a computer network, for the purpose of sharing files and storage resources. DFSs are emerging as a key component of large-scale cloud computing platforms. Applications on such computing paradigms come with increasing challenges on how to transfer and where to store and compute data reliably and efficiently. Specifically, these challenges include data transfer bottle-necks, performance unpredictability, scalable storage and so on. To deal with these challenges, various DFSs such as Hadoop distributed file system (HDFS), the Google file system (GFS), MooseFS, and ZFS, have been developed for large-scale distributed systems such as Facebook and Google. Performance analysis is an important concern in the distributed system research area. Researchers have made a lot effort to evaluate, model, and analyse distributed systems for computing intensive or data intensive applications. There exist well-known evaluation benchmarks (e.g., LINPACK, MPIBLAST) for computing paradigms. However, similar kinds of widely accepted benchmarks are rarely seen in DFSs. For example, there is no specific benchmark proposed for evaluating DFSs (e.g., HDFS, GFS) for web services. Some other related works evaluate performance of DFSs by comparing them with similar kinds of DFSs (e.g., NFS) via running in-house benchmarks or application programs. Some researchers measured the performance of DFSs under a variety of workloads and strategies. In the field of evaluating the performance of DFSs, a typical approach taken is through experiments by running DFSs; therefore, it is mainly based on the analysis of the experiment results, or draw conclusion by comparing with existing DFSs. Therefore, there rarely exist approaches that are capable of qualitatively and quantitatively analysing the overall performance of a DFS or its individual step or component, prior to the deployment of the DFS, without running benchmarks or particularly designed or selected applications. In this paper, we propose such an approach that is driven by architecture and design models of DFSs. System architects can use design-time performance to evaluate the resource utilization, throughput, and timing behaviour of a system prior to the deployment due to the following reasons: 1) analysing performance of the system is much files expensive than testing the performance of the system by running it, 2) it is simply infeasible to test all kinds of different configurations of the system by running it, and 3) performance analysis on models helps architects make configuration and deployment decisions to avoid costly redesign, reconfiguration or redeployment. Some model-driven performance analysis and prediction (MDPAP) approaches have been proposed in the literature and especially Balsamo et al. conducted a survey on MDPAP.

The survey results reveal that 1) most approaches make use of unified modelling language (UML) or UML-like formalisms to describe behavioural models, 2) few approaches provide direct correspondence between the software specification abstraction and the performance model evaluation results, and 3) the performance model should be easy to apply in practice. The key challenge of MDPAP approaches is finding the right architecture and performance abstraction of the system under study. Based on the above study, in this paper, we propose a practical performance analysis methodology particularly for DFSs. The approach is based on the UML specification of the DFS architecture and their key behaviours. Some elements of the architecture model are also characterized by some stereotypes from the MARTE profile, which is a UML profile for modelling and analysis of real-time and embedded systems.[6] We define the following characteristics of our methodology: 1.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

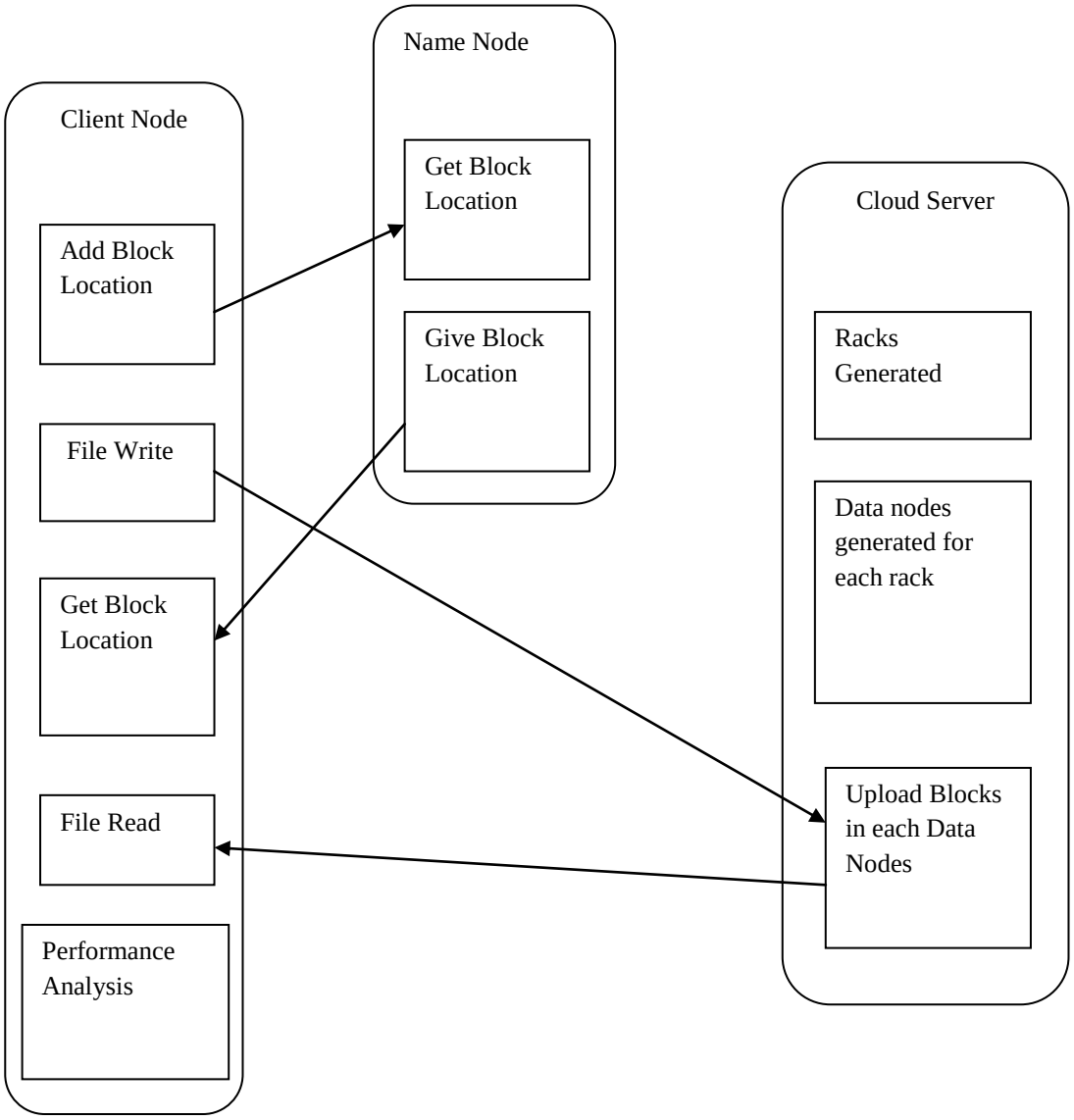


Fig. 1 Process Flow Diagram or Architecture Diagram

DFS architecture and design models provide a common understanding of DFS, which is considered crucial as DFS practices lack of such a common knowledge base. 2. Based on the models, one can systematically and automatically derive configurable parameters. Without them, this activity will heavily rely on expert implicit knowledge, inevitably file reading to the low-quality management of the process. 3. Configuring DFS systems is typically the first and most important step to set up experiments. Therefore, our methodology provides a way to design experiments whose results directly contribute to performance analysis. 4. Qualitative and quantitative performance analysis can be conducted, based on the models, system configurations and experiment results.

Analysis results can be also interpreted based on the architecture and design models, therefore making the architecture and design refinement much easier. 5. Based on the models, both the overall system performance and individual component or execution step performance can be analysed.

This is because the message interactions and relationships between components are clearly specified in the architecture and design models. As one popular master/slave structured DFS, HDFS is selected as the representative for evaluation. Three sets of real-world experiments were conducted to qualitatively assess the effectiveness of our performance analysis approach. We also conducted a set of experiments of HDFS on EC2 to quantitatively analyse the memory and CPU bottlenecks of the metadata server of HDFS and formulate the response time of the Read operation of the metadata server to client requests.

2. LITERATURE REVIEW

Data-intensive distributed file systems are emerging as a key component of large scale Internet services and cloud computing platforms. They are designed from the ground up and are tuned for specific application workloads. Google File System, Hadoop distributed file system (HDFS) [2] and Amazon S3 [1], are defining this new

DOI 10.17148/IJARCCE.2016.53207

purpose-built paradigm. In this paper we compare and contrast parallel file systems, developed for high performance computing, and data-intensive distributed file systems, developed for Internet services.

The goal of this paper is to compare Internet services file systems and parallel file systems, specifically can we use modern parallel file systems in the place of custom Internet services file systems. In this paper, we compare and contrast the Parallel Virtual File System (PVFS), a representative for parallel file systems, and the Hadoop Distributed File System (HDFS), a representative for Internet services file systems.[5]

We built a non-intrusive shim layer to plug PVFS in to the open-source Hadoop Internet services stack. This enables Hadoop applications to use PVFS for persistent storage without any changes to the PVFS source shim layer enables PVFS to over the same benefits that HDFS over to the Hadoop data-processing framework through three key features: Exposing data layout for function shipping HDFS is optimized for applications that process massive amount of data using the Hadoop / Mapreduce abstraction, A goal of this abstraction is to minimize the transfer of large amounts of input data by shipping computation to nodes that store the input data. The Hadoop framework achieves this collocation using file data layout exposed by HDFS. PVFS also maintains file layout information. Our shim layer extracts layout maps from PVFS to the Hadoop framework. Read a head buffering to avoid the overhead of synchronous small reads, HDFS clients pre-fetch large amount of data and pipeline the data transfer in smaller units. Because clients in PVFS are stateless and do not cache data, all requests for data are synchronously sent to the server, irrespective of the amount requested. Our shim layer implements a read a head buffer that enables PVFS to match the transfer efficiency of HDFS. Replication for fault tolerance HDFS provides high availability by storing three copies (by default) of a file. It uses a rack-aware replication policy to ensure data availability in face of machine and rack failures. PVFS relies on storage hardware to provide fault tolerance. Our shim layer emulates HDFS-like replication by writing all data, on behalf of the PVFS clients, to three different servers. We evaluate the performance of PVFS and HDFS by running micro benchmarks and macro benchmarks, comprised of a suite of four data-intensive applications, on the 4,000 core Yahoo! M45 cluster. Our experiments demonstrate that PVFS performs at file as good as HDFS for most workloads including data-intensive Hadoop applications that benefit from the data layout. The major exception to this is sort, which is a write-intensive workload. In such workloads, HDFS writes one copy un striped locally and two striped widely while our un modified PVFS writes all three remotely. With limited network bandwidth this can cause a 2:3 ratio in completion time. Moreover, PVFS outperforms HDFS for workloads doing concurrent writes to the same file because HDFS does not support concurrent writes. For instance, a “parallel" file copy operation using PVFS is more than four times faster than HDFS on 16 nodes.[5]

3. PRESENTATION OF THE MAIN CONTRIBUTION OF THIS PAPER

Model-driven performance analysis has been recognized as an important tool to analyze system performance. In the paper, we presented such an approach, particularly tailored for distributed file systems (DFSs). Our approach mainly has several components: the architecture and design models and explicitly captured performance-relevant, configurable parameters, and the systematically derived performance model. The related work in the field mainly evaluates the performance of DFSs and computing paradigms by for example, relying on running benchmarks or application programs, performance measurements under a variety of workloads/strategies, and comparing with other DFSs. Our approach, however, qualitatively and quantitatively analyses the DFS performance based on the models we constructed, such that early feedback on architectural design, configuration, and deployment of DFSs can be provided. Thereby one can avoid cost for architectural redesign and redeployment. We conducted a series of real-world experiments deployed on EC2, Tansuo, and Inspur to demonstrate how our approach should be applied and to evaluate how effective it is. Results show that our approach is practical and can achieve sufficient performance analysis.

4. METHODOLOGY

In this we use the approach of HDFS, the Hadoop Distributed File System, is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and provide high throughput access to this information. Files are stored in a redundant fashion across multiple machines to ensure their durability to failure and high availability to very parallel applications. HDFS has a master /slave architecture. An HDFS cluster consists of a single Name Node, a master server that manages the file system namespace and regulates access to files by clients. In addition, there are a number of Data Nodes, usually one per node in the cluster, which manages storage attached to the nodes that they run on. Internally, a file is split into one or more blocks and these blocks are stored in a set of Data Nodes. The Name Node executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to Data Nodes. The Data Nodes are responsible for serving read and write requests from the file system’s clients. The Data Nodes also perform block creation, deletion, and replication upon instruction from the Name Node.[11]

Map Reduce is also a data processing model. Its greatest advantage is the easy scaling of data processing over multiple computing nodes. Under the Map Reduce model, the data processing primitives are called mappers and reducers. In the mapping phase, Map Reduce takes the input data and feeds each data element to the mapper. In the reducing phase, the reducer processes all the outputs from the mapper and arrives at a final result. In simple terms, the mapper is meant to filter and transform the input into something that the reducer can aggregate over.[12]

DOI 10.17148/IJARCCE.2016.53207

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 5, Issue 3, March 2016

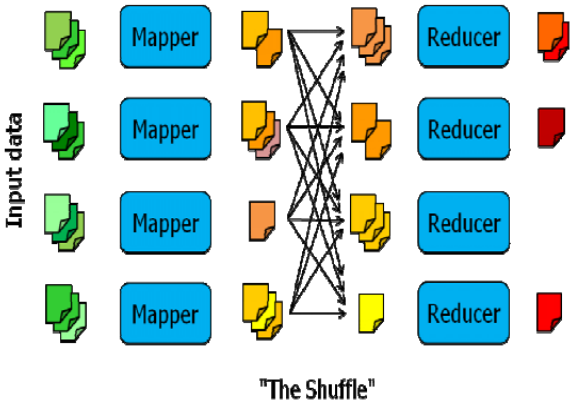
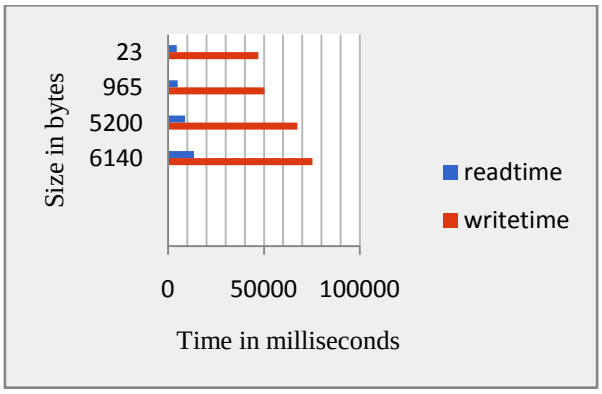


Fig. 2 Map Reduce

5. FINDINGS

In this paper, an approach is proposed which increases the performance of reading and writing the file. The file is divided into racks and each rack consists of one or more data nodes. Data node is used for file storage. Name node consists of the location of file. Depending on the read time and the write time performance is determined as shown in the table below.



6. CONCLUSION

The performance of DFSs and computing paradigms by, for example, relying on running benchmarks or application programs, performance measurements under a variety of workloads/strategies, and comparing with other DFSs. Our approach, however, qualitatively and quantitatively analyzes the DFS performance based on the models we constructed, such that early feedback on architectural design, configuration, and deployment of DFSs can be provided.

ACKNOWLEDGEMENT

Every work requires support from many people and areas. We would like to thank our guide **Prof. R. Vasavi** and H.O.D (Computer Dept.) for giving us the valuable guidance and encouragement and for providing all facilities for smooth progress of our project. We would also like to thank all the staff members of Computer Engineering Department for timely help and inspiration for completion of the project.

REFERENCES

[1] Amazon Simple Storage Service (Amazon S3), http://aws.amazon.com/S3/, 2013.
[2] Hadoop Distributed File System (HDFS), http://hadoop.apache.org/hdfs/, 2013.
[3] Moose FS, http://www.moosefs.org/, 2013
[4] OpenStack SWIFT, http://openstack.org/software/openstack storage/, 2013.
[5] Parallel Virtual File System, http://www.pvfs.org/, 2013.
[6] The UML MARTE Profile, http://www.omgmatre.org/, 2013.
[7] The Windows Azure, http://www.windowsazure.com/, 2013.
[8] M.G.Baker, J.H.Hartmen, M.D.Kupfer, K.W.Shirriff, and J.K.Ousterhout, "Measurements of a Distributed File System", Proc.ACM SIGOPS Operating Systems Rev., vol. 25, pp. 198-212, 1991.
[9] S.Gheamawat, H. Gobioff and S. T. LEUNG, "The Google File System", vol. 37pp. 29-43, 2003.
[10] OMG, "UML 2.2 Superstructure Specification (formal/2009-02-04)".
[11] HDFS Architecture Guide
[12] Hadoop in Action – Chuck Lam