The Journal of Systems and Software 121 (2016) 16–27

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Adding data analytics capabilities to scaled-out object store

Cengiz Karakoyunlu[a,*], John A. Chandy[a], Alma Riska[b]

[a] Department of Electrical and Computer Engineering, University of Connecticut, Storrs, CT 06269, United States
[b] NetApp, Inc., United States

ARTICLE INFO

*Article history:*
Received 21 November 2015
Revised 10 May 2016
Accepted 21 July 2016
Available online 30 July 2016

*Keywords:*
In-situ data analytics
Object storage
Attribute-based storage
MapReduce

ABSTRACT

This work focuses on enabling effective data analytics on scaled-out object storage systems. Typically, applications perform MapReduce computations by first copying large amounts of data to a separate compute cluster (i.e. a Hadoop cluster). However; this approach is not very efficient considering that storage systems can host hundreds of petabytes of data. Network bandwidth can be easily saturated and the overall energy consumption would increase during large-scale data transfer. Instead of moving data between remote clusters; we propose the implementation of a data analytics layer on an object-based storage cluster to perform in-place MapReduce computation on existing data. The analytics layer is tied to the underlying object store, utilizing its data redundancy and distribution policies across the cluster. We implemented this approach with Ceph object storage system and Hadoop, and conducted evaluations with various benchmarks. Performance evaluations show that initial data copy performance is improved by up to 96% and the MapReduce performance is improved by up to 20% compared to the stock Hadoop implementation.

## 1. Introduction

High-performance computing on large-scale data has become an important use case in recent years. There are various storage system solutions for end users to perform high-performance computation on large-scale data, while also providing data protection and concurrency between different users Amazon elastic compute cloud.

Clusters and cloud storage applications that work on large-scale data typically employ separate compute and storage clusters, since the requirements of the compute and storage tiers are different from each other. However, a serious drawback of this architecture is the need to move large amounts of data from the storage nodes to the compute nodes in order to perform computation and then to move the results back to the storage cluster. Today, many storage systems store petabytes of data for various applications, such as climate modeling, astronomy, genomics analysis etc., and the amount of data stored in these systems is projected to reach exabyte scale in the near future (Gantz and Reinsel, 2012). Therefore, moving big amounts of data between storage and compute nodes is not an efficient way of performing computation on large-scale data anymore. Additionally, storing data both at the storage and compute sites increases storage overhead and with data replicated multiple times at both sites for resiliency, this overhead becomes even worse. Moving data between storage and compute nodes also increases the total energy consumption and the network load.

On the other hand, there have been many efforts that have gone into improving storage interfaces and abstractions in order to store and access data more efficiently. Object-based storage (Gibson et al., 1998; Mesnier et al., 2003) is an important effort in this respect and many scaled-out storage systems today  Lustre (Lustre; Maltzahn et al., 2010; Swift, 2015) are based on the object-based storage abstraction. Object-based storage is an alternative to the traditional block-based storage (i.e. SCSI, ATA). Data is stored in discrete containers, called *objects*, each of which is identified by a distinct numerical identifier. Each object stores data and data attributes that can be controlled by the user. Data attributes can be used to store metadata describing the data (i.e. size, name, replica locations etc.) and metadata management operations to query these attributes can be offloaded from dedicated servers to object storage for improved performance (Ali et al., 2008). As a result, object-based storage increases the interaction between the storage system and the end-user and simplifies the data management of a storage system.

Using object-based storage features, the computational applications in a cluster or cloud application can benefit from the intelligence of the underlying storage system and eliminate data movement while enabling in-place analytics capabilities. Consequently, the storage layer can be scaled while the computational layer remains lightweight. In this paper, we propose an example of

* Corresponding author.
*E-mail addresses:* cengiz.k@uconn.edu (C. Karakoyunlu), john.chandy@uconn.edu (J.A. Chandy).

http://dx.doi.org/10.1016/j.jss.2016.07.029
0164-1212/© 2016 Elsevier Inc. All rights reserved.

this approach by implementing a computational framework, Hadoop (Shvachko et al., 2010), on Ceph object-based storage system (Weil et al., 2006). We also conduct performance evaluations using *Grep* (Hadoop Grep, 2009), *Wordcount* (Hadoop WordCount, 2011), *TestDFSIO* HAD and *TeraSort* (Hadoop TeraSort, 2011) benchmarks with various redundancy and replication policies. The evaluation results indicate that initial data copy performance of Hadoop is improved by up to 96% and MapReduce performance is improved by up to 20%. It is important to note that, Hadoop and Ceph object storage system can still be used as stand-alone systems in this approach, meaning that their normal functionalities are not impacted.

The rest of this paper is organized as follows. Section 2 briefly introduces MapReduce and object-based storage, two main components of this work. Then, Section 3 discusses related studies in a number of categories: improving the performance of Hadoop as a stand-alone system, using a cluster file system as the backend storage of Hadoop and integrating the computation layer of Hadoop, MapReduce, with object storage systems for in-place computation. While presenting studies for the last category, their disadvantages against the method presented in this paper are discussed; namely, data is still transferred to HDFS, data management policies of the underlying storage system are overridden or data-compute locality is only provided through virtualization. Section 4 shows how to enable in-place analytics capabilities on large-scale data using Hadoop and Ceph object storage without transferring data from compute nodes to storage nodes and without changing how the underlying storage is managed. Section 5 gives the performance evaluation results of the proposed method from *Grep* (Hadoop Grep, 2009), *Wordcount* (Hadoop WordCount, 2011), *TestDFSIO* HAD and *TeraSort* (Hadoop TeraSort, 2011) benchmarks. Finally, Section 6 summarizes the findings of this work and discusses possible future research directions.

## 2. Background

This section gives a brief overview of the main components of the approach proposed in this work - MapReduce and object-based storage.

### 2.1. MapReduce

MapReduce is a parallel computational model developed originally by Google (Dean and Ghemawat, 2008) and it is widely used for distributed processing of large datasets over clusters. Data in MapReduce is represented with $<$ key, value $>$ pairs. The first step of an application using MapReduce is to partition its input data into blocks that are replicated across datanodes. This data is then processed in parallel with mappers that produce intermediate data from the input data. This intermediate data is then fed to reducers which process the intermediate data based on intermediate keys and combine intermediate values to form the final output data of the application.

Hadoop (Shvachko et al., 2010) is a commonly used open-source implementation of MapReduce and it consists of two layers - storage and computation. The MapReduce algorithm is implemented in the computational layer, whereas the storage layer is managed by the Hadoop Distributed File System (HDFS). HDFS provides redundancy by replicating data three times (by default) across the storage nodes while also trying to preserve the data locality of the system. One replica is stored locally, the second replica is located in another node in the same rack and the last replica is stored in another rack. Hadoop applications also follow a write-once-read-many workflow and as a result, they can benefit from the approach presented in this paper extensively, as data is not ingested from a remote storage cluster to the compute cluster.

### 2.2. Object-based storage

Object-based storage is a storage model that stores and accesses data in flexible-sized logical containers, called *objects*, instead of using the traditional fixed-sized, block-based containers. Objects store metadata either together with data or in dedicated object attributes. Metadata can be any type of data (i.e. size, access permissions, creation time etc.) describing the actual object data. Increasing interest in object-based storage led to the standardization of the T10 object-based storage interface OSD. There have been many examples of object-based storage systems in cluster file systems; such as PVFS (Carns et al., 2000) and Lustre as well as scaled out cloud storage systems; such as Ceph (Weil et al., 2006), OpenStack Swift (Swift, 2015), and Amazon S3 Amazon Simple Storage Service. These systems are typically designed as a software interface on top of an existing file system.

## 3. Related work

This section introduces related studies on improving the performance of Hadoop and its integration with object storage.

There have been several research efforts that analyzed and tried to improve the performance of Hadoop without integrating it with an underlying storage system. Shvachko et al. show the metadata scalability problem in Hadoop, by pointing out that a single namenode in HDFS is sufficient for read-intensive Hadoop workloads, while it will be saturated for write-intensive workloads (Shvachko, 2010). Some related studies improved the performance of Hadoop by modifying its internal data management methods. *Scarlett* replicates data based on popularity, rather then creating replicas uniformly and causing machines containing popular data to become bottlenecks in MapReduce applications (Ananthanarayanan et al., 2011). Porter analyzes the effects of decoupling storage and computation in Hadoop by using *SuperDataNodes*, servers that contain more disks than traditional Hadoop nodes, for the cases where the ratio of the computation to storage is not known in advance (Porter, 2010). *CoHadoop* modifies Hadoop by co-locating and copartitioning related data on the same set of nodes with the hints gathered from the applications (Eltabakh et al., 2011). *Maestro* identifies map task executions processing remote data as an important bottleneck in MapReduce applications and tries to overcome this problem with a scheduling algorithm for map tasks that improves locality (Ibrahim et al., 2012).

Hadoop is also integrated with cluster file systems in a number of studies, in order to analyze the outcomes of using cluster file systems for MapReduce applications. Tantisiriroj et al. integrate *PVFS* (Carns et al., 2000) with Hadoop and compare its performance to HDFS (Tantisiriroj et al., 2011). Ananthanarayanan et al. use *metablocks*, logical structures that support both large and small block interfaces, with *GPFS* to show that cluster file systems with metablocks can match the performance of Internet file systems for MapReduce applications (Ananthanarayanan et al., 2009). *Lustre* can also be used as the backend file system of Hadoop LUS.

More recent work integrates object storage with MapReduce for in-place data analytics. Rupprecht et al. integrates OpenStack Swift with MapReduce (Rupprecht et al., 2014); however, this work overrides the replication policy of OpenStack Swift and has performance loss due to the time reducers spend while renaming results. CAST (Cheng et al., 2015) performs cloud storage allocation and data placement for data analytics workloads by leveraging the heterogeneity in cloud storage resources and within jobs in an analytics workload. SupMR (Sevilla et al., 2014) creates MapReduce input splits from data chunks rather than entire data, meaning that data is still copied to the HDFS. Nakshatra (Kathpal and Yasa, 2014)