Future Generation Computer Systems 30 (2014) 1–13

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Achieving Accountable MapReduce in cloud computing

Zhifeng Xiao[a], Yang Xiao[b,*]

[a] Behrend College, The Pennsylvania State University, Erie, PA 16563, USA

[b] The University of Alabama, Tuscaloosa, AL 35487-0290, USA

HIGHLIGHTS

- Propose Accountable MapReduce, which forces each machine to be held responsible for its behavior.
- To optimize the utilization resource, we formalize the Optimal Worker and Auditor Assignment (OWAA) problem.
- Our evaluation results show that the *A*-test can be practically and effectively applied to existing cloud platforms employing MapReduce.

ARTICLE INFO

Article history:
Received 19 October 2011
Received in revised form
13 June 2013
Accepted 17 July 2013
Available online 31 July 2013

Keywords:
Accountable
MapReduce
Cloud computing

ABSTRACT

MapReduce is a programming model that is capable of processing large data sets in distributed computing environments. The original MapReduce model was designed to be fault-tolerant in case of various network abnormalities. However, fault-tolerance does not guarantee that each working machine will be completely accountable; when nodes are malicious, they may intentionally misrepresent the processing result during mapping or reducing, and they may thus make the final results inaccurate and untrustworthy. In this paper, we propose Accountable MapReduce, which forces each machine to be held responsible for its behaviors. In our approach, we set up a group of auditors to perform an Accountability Test (*A*-test) that checks all of the working machines and detects malicious nodes in real time. The *A*-test can be implemented with different options depending upon how the auditors are assigned. To optimize the utilization resource, we also formalize the Optimal Worker and Auditor Assignment (OWAA) problem, which is aimed at finding the optimal number of workers and auditors in order to minimize the total processing time. Our evaluation results show that the *A*-test can be practically and effectively applied to existing cloud platforms employing MapReduce.

1. Introduction

MapReduce [1] has been widely used as a powerful data processing model. It has efficiently solved a wide range of large-scale computing problems, including distributed grep, distributed sort, web-link graph reversal, web-access log stats, document clustering, machine learning, etc. Cloud computing presents a unique opportunity for batch-processing and analyzing terabytes of data that would otherwise take hours to finish [2–5]. Most cloud providers (e.g., Google, Yahoo!, Facebook, etc.) adopt MapReduce to build multitenant computing environments. Usually, cloud customers have a large set of data to be processed under certain time constraints. They must provide a client with the MapReduce program and with data that is ready to be processed. Cloud providers maintain thousands of working machines to fulfill the data processing jobs submitted by their customers. As an example [6], The New York Times used 100 Amazon Elastic Compute Cloud (Amazon EC2) instances and a Hadoop [7] application to process 4 TB of raw image TIFF data (stored in Amazon Simple Storage Service (Amazon S3)) into 11 million finished PDFs in 24 h at a computation cost of about $240 (not including bandwidth).

In such a computing environment, the cloud customers outsource their data to the cloud, which performs the storing and computing operations required by the customers. Customers must, therefore, fully trust the cloud provider. However, a cloud provider cannot guarantee that its data center (which may have thousands of working machines) is 100% trustworthy. Some machines may become malicious if they are attacked and controlled by hackers; malicious machines will not faithfully carry out the tasks assigned to them. As a result, the processing result is no longer correct or trustworthy. In the New York Times example, malicious nodes may mess up the image conversion process so that the PDFs do not match the original TIFF images. It is even harder for the New York Times to check if these PDFs are correctly converted because of the tremendous data size of the PDFs. In this paper, we explore the use of accountability to address this problem.

* Correspondence to: Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35487-0290, USA. Tel.: +1 205 348 4038; fax: +1 205 348 0219.

E-mail addresses: zux2@psu.edu (Z. Xiao), yangxiao@ieee.org (Y. Xiao).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.07.001

Accountability has been a longstanding concern of trustworthy computer systems [8], and it has recently been elevated to a first class design principle for dependable network systems [9,10]. Accountability implies that an entity should be held responsible for its own actions or behaviors [11–16]. In the MapReduce scenario, accountability means that all working machines (e.g., mappers and reducers) will be responsible for the tasks that they have completed.

In this paper, we propose building an Accountable MapReduce to make the cloud computing platform trustworthy. We use an Accountability Test (*A*-test), which checks all working machines when a job is undertaken and detects malicious nodes in real time. The *A*-test is performed by a group of trusted machines, which are called the Auditor Group (AG). The AG takes advantage of the determination of the user's MapReduce program to replay the tasks executed by working machines. The MapReduce framework makes it possible for an auditor to acquire the input data block and processing results without knowledge of the working machine. Therefore, auditors are free to replay the tasks that have been finished. If the replay output does not match the original output, it means that the worker is returning bad results, and the evidence is the combination of the task, input, original output, and replay output.

A challenge of Accountable MapReduce is the reduction of the overhead introduced by the *A*-test. In theory, the *A*-test can guarantee the detection of any misbehavior by fully duplicating each task, and this causes the processing time to at least double. To make the *A*-test more efficient, we abandon pursuing 100% accountability, which guarantees exposure of every malicious node but has a high cost. We adopt *P*-Accountability [17], which quantifies the degree of accountability. We use *P*-Accountability for system efficiency. Based upon the batch-processing property of MapReduce, the performance of the *A*-test with *P*-Accountability can be greatly improved by decreasing the degree of accountability by less than 1%.

We summarize the contributions of this paper as follows:

1. We propose building an Accountable MapReduce to detect malicious nodes. Verifiable evidence will be generated to ensure that the malicious nodes cannot deny their behavior.
2. Instead of pursuing perfect accountability, *A*-test allows the system to achieve *P*-Accountability with less overhead and a higher performance.
3. We formalize the Optimal Worker and Auditor Assignment (OWAA) problem, which is aimed at finding the optimal numbers of workers and auditors in order to minimize the total processing time.
4. We also present another sentinel-based verification scheme for implementing the *A*-test. Our analysis shows the sentinel scheme is not as good as the original scheme.
5. We have implemented a prototype of Accountable MapReduce on Hadoop. The experiment's results show that our approach is both efficient and effective.

The rest of this paper is structured as follows: Related work will be reviewed in Section 2. Then, we will introduce MapReduce in Section 3. In Section 4, we address the accountability issue in MapReduce and define the problem that is our focus. Our solution, Accountable MapReduce, is discussed in Section 5. In Section 6, we give the implementation details. *A*-test is presented in Section 7. Analysis of Accountable MapReduce is given in Section 8. We present a sentinel-based verification scheme in Section 9. Evaluation is provided in Section 10. Finally, we conclude the paper in Section 11.

2. Related work

Its ability to process data intensive tasks has made MapReduce increasingly important in distributed computing areas [18]. Chu et al. [19] applied MapReduce to machine learning on multi-core platforms. He et al. [20] implemented Mars, a MapReduce framework, in graphics processors. Papadimitriou et al. [21] applied MapReduce to the area of data mining; they designed Disco, which is a practical approach for distributed data pre-processing. Ekanayake et al. [22] adopted the MapReduce technique for two scientific data analyses, which are high energy physics data analyses, and for *K*-means clustering. Existing work focuses on utilizing MapReduce to solve different problems in various domains. However, few have considered accountability issues in MapReduce. Accountable MapReduce is an attempt to address the issue of untrustworthy nodes and their behavior in MapReduce.

Security issues in MapReduce have been discussed in [23,24]. Wei et al. [25] present SecureMR, a practical service integrity assurance framework for MapReduce. SecureMR provides a decentralized, replication-based integrity verification scheme for ensuring the integrity of MapReduce in open systems. SecureMR is intended to achieve 100% integrity of MapReduce, which can affect its performance. We believe that for some applications, efficiency is more important than 100% integrity. Therefore, instead of pursuing 100% accountability, we allow the customers to choose the level of accountability that they need based upon their applications. It turns out that slightly decreasing the expectation of accountability leads to a significant improvement of system performance in terms of job processing time.

Accountability has been regarded as an important issue in cloud computing. Trustworthy relationships between the cloud provider and cloud customers have been addressed in [26,27]. The customer places his computation and data on machines that he cannot directly control; the provider agrees to run a service with details he/she does not know [26]. Therefore, accountability is employed to determine whether or not the Service Level Agreement (SLA) is fulfilled. If it is not, evidence should be provided in order to prove which unit is responsible. MapReduce is a popular computing framework in cloud platforms. In this paper, we build Accountable MapReduce, which solves the subset of problems addressed in [26]. Accountable MapReduce is able to detect malicious workers and provide verifiable evidence.

3. MapReduce background

3.1. Programming model

With the MapReduce programming model, programmers only need to specify two functions: *Map* and *Reduce*. The Map function receives a key/value pair as input and generates intermediate key/value pairs to be further processed. The Reduce function merges all the intermediate key/value pairs associated with the same (intermediate) key and then generates final output.

There are three main roles: the master, mappers, and reducers. The single master acts as the coordinator responsible for task scheduling, job management, etc. MapReduce is built upon a distributed file system (DFS), which provides distributed storage. Fig. 1 shows the execution process of MapReduce. The input data is split into a set of *M* blocks, which will be read by *M* mappers through DFS I/O. Each mapper will process the data by parsing through the key/value pair, and then, they will generate the intermediate results that are stored in its local file system. The intermediate result will be sorted by the keys so that all pairs with the same key will be grouped together (the shuffle phase). If the memory size is limited, an external sort might be used to deal with large amounts of data at one time. The locations of the intermediate results will be sent to the master who notifies the reducers to prepare to receive the intermediate results as their input. Reducers then use the Remote Procedure Call (RPC) to read data from mappers. The user defined reduce function is then applied to the sorted data; basically, key pairs with the same key will be reduced in some way depending upon the user defined reduce function. Finally, the output will be written to DFS.