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.

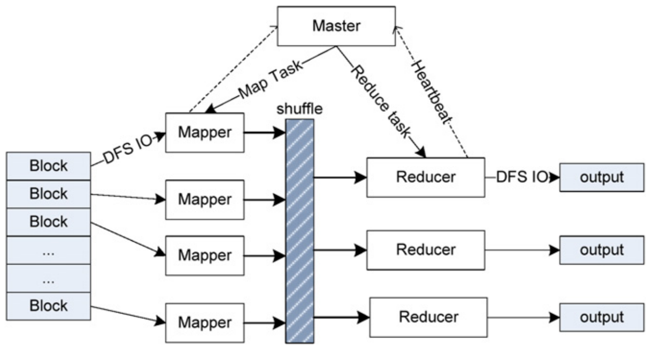

Fig. 1. MapReduce working flow.

3.2. *Fault tolerance*

MapReduce is designed to be fault tolerant because failures are a common phenomena in large scale distributed computing.

3.2.1. *Worker failure*

The master pings every mapper and reducer periodically. If no response is received for a certain amount of time, the machine is marked as failed. The ongoing task and any tasks completed by this mapper will be re-assigned to another mapper and executed from the very beginning. Completed reduce tasks do not need to be re-executed because their output is stored in the global file system.

3.2.2. *Master failure*

Since the master is a single machine, the probability of master failure is very small. MapReduce will re-start the entire job if the master fails.

3.2.3. *Byzantine fault tolerance*

A Byzantine fault [28] is an arbitrary fault that occurs during the execution of an algorithm by a distributed system. It encompasses both omission failures (e.g., crash failures, failing to receive a request, or failing to send a response) and commission failures (e.g., processing a request incorrectly, corrupting local state, and/or sending an incorrect or inconsistent response to a request).

The MapReduce framework can suffer both omission failures and commission failures. Omission failures can be properly solved by the MapReduce built-in fault tolerance mechanisms. However, commission failure is not considered in the original version.

4. Problem statement

4.1. *Attack model*

Fault-tolerance will address node failures, such as a worker not responding to the master or a worker machine totally crashing, etc. To address node failures, the master learns the task fail event and then takes further action (e.g., it re-executes the failed Map/Reduce task on another machine). However, fault-tolerance is unable to detect a malicious node intending to alter the Map/Reduce function and return inaccurate results. We illustrate this type of attack with an example.

Wordcount is a typical MapReduce application. Its job is to count the occurrences of each word in large input text data. If there are malicious working machines in the system, the output file, which contains word counts of every word, is inaccurate.

Consider the wordcount example in Fig. 2. Assume that the system is free of malicious nodes. There are three mappers, each of which maps one line of the file. After the mapping function, we have the map output as the intermediate result. Then, the intermediate results will be shuffled (sorted by key) and read by reducers (five, in this case), which reduce the intermediate results and generate the final output.

If all units faithfully execute their tasks, the final output will be accurate. Otherwise, we cannot trust the results because the malicious units may alter part of the results. For example, if a mapper is malicious, it has multiple ways to alter the output: (1) filter some keys, (2) create keys that do not exist in the input file, (3) modify the value intentionally, etc. A malicious reducer is able to cause similar errors.

To solve the problem, we propose Accountable MapReduce, which ensures that

(1) Malicious nodes intending to alter the processing result will be exposed; additionally, Accountable MapReduce is able to provide verifiable evidence to ensure that the detection is reputable.
(2) The failed jobs will be re-directed to another working node until it is verified as correct.

5. Accountable MapReduce

5.1. *Design principles*

A key function of Accountable MapReduce is detecting malicious nodes that generate inaccurate results. We now present the principles that guided our design:

(1) The accountability mechanism should be concealed so that malicious nodes are unaware of what is happening. We assume that machines may be fully controlled by attackers, and they may be smart enough to discover this; if a machine is aware of anything abnormal, it takes countermeasures to cover itself. It follows that we leave any machine alone when an *A*-test is ongoing.
(2) The overhead brought by the accountability mechanism should be minimized to reduce processing time.
(3) When a malicious node is caught, the system should be able to provide verifiable evidence to show that the node is indeed being malicious.

5.2. *Assumptions*

The design of Accountable MapReduce is based upon the following assumptions:

(1) The data set provided by cloud customers can be processed by MapReduce.
(2) The Auditor Group (AG) is a trustworthy domain, and this means that the machines of the AG are free of any malicious actions.

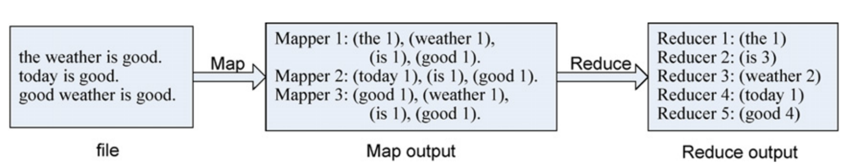

Fig. 2. A wordcount example of MapReduce.

(3) A worker cannot be reclaimed until the entire job is completed. When the customer confirms the job is done, all machines will be released back to the cloud.
(4) A malicious node randomly performs bad actions. This means that the faulty parts of the processing result also distribute randomly throughout the entire result. In addition, there may be multiple faulty parts in the processing results. Once a fault area is found, the test will stop because we already have evidence to expose the bad node.
(5) All input data, intermediate results, and output data will not be removed until the entire job is finished.
(6) Data from a cloud customer is correct.

5.3. Accountable MapReduce design

5.3.1. Correctness checking scheme

PeerReview [29] provides accountability for distributed systems. It assumes that every node in the system is a deterministic state machine (i.e., for some certain input, the output will be the same). Two critical technologies that are employed by PeerReview are tamper-evident logging and witnessing. A tamper-evident log is implemented by a hash chain, which guarantees that any modification to the log will be detected so that a node has to record its behavior faithfully. A witness, which is also a regular node, is able to check the correctness of other nodes that it is witnessing by replaying the log files kept in each node. As a result, malicious nodes will eventually be detected and exposed to all other correct nodes. PeerReview is applicable to most distributed applications. However, it is not applicable to MapReduce. The major concern is overhead. First, the input of a large task might be at the TB or even PB level (even though there are thousands of workers, the split task also has a large workload), and the output depends upon the input. This means that all input and output events will have to be logged so that the witness is capable of replaying log files and checking their correctness. Second, for witness checking, a node has to upload its log segment to multiple witnesses, which is extremely bandwidth-consuming.

The idea of correctness checking is simple. Assume that the auditor is a trustworthy node; both the worker and the auditor are regarded as deterministic state machines, and the protocol is running on them. If the input data is the same (adopting tamper-evidence logs to ensure it), the output should be the same as well. After comparing output (from the worker) and output' (from the auditor), the system is able to determine whether the worker is good or not. The evidence is the combination of input, output', and output; additionally, it is verifiable to any other auditors.

5.3.2. Auditor group

The Auditor Group (AG) carries out an Accountability Test (i.e., an *A*-test, which will be introduced next) to detect malicious nodes. Normally, as shown in Fig. 3, cloud resources will be divided into multiple slices, each of which is rented by a customer. A slice is a group of working machines assigned to a customer. We maintain an AG manager for the entire cloud and one AG for each slice that runs MapReduce. The reason for associating each slice with one AG is to conserve the privacy and independence of customers.

The AG Manager is a coordinator that conducts AG creation, management, and disposal. After the AG manager becomes aware of the customer's data size, timing, and other requirements, it will determine the AG size and then create an AG for the slice.

Each AG is internally structured as a cluster. The head node is the Group Head (GH), and the member node is the Group Member (GM). The GH randomly picks up workers as test targets. The master has a protocol with the GH to provide all the information needed for an *A*-test. The GH assigns *A*-test tasks to the available GMs, which are the actual machines that accomplish the tasks and report their status.

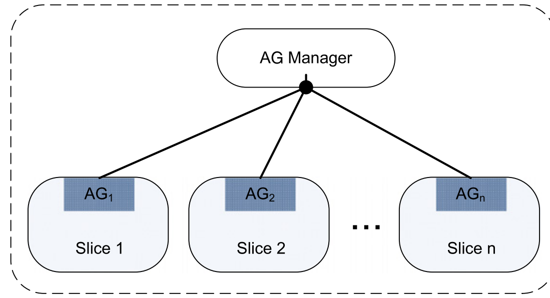

Fig. 3. Auditor group in cloud platform.

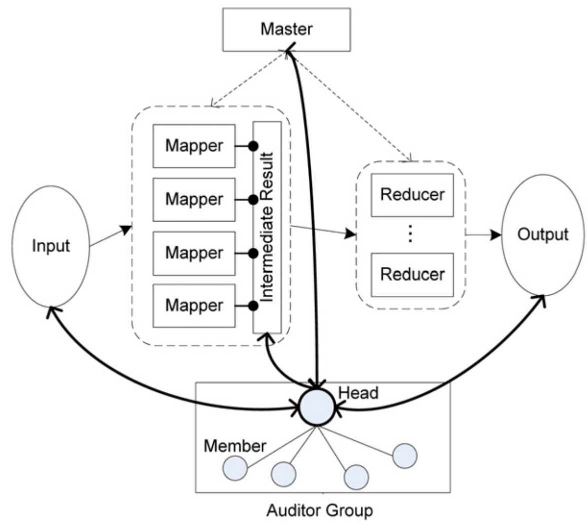

Fig. 4. Accountability test.

5.3.3. Accountability test

The *A*-test is built upon the correctness checking scheme that we adopt in this paper. The AG is the entity fulfilling the *A*-test. The AG consists of a set of trustworthy workers assigned by the AG manager; these are machines dedicated to performing the *A*-test as shown in Fig. 4. The working flow of the *A*-test is as follows:

1. The *A*-test is started when Map/Reduce starts.
2. A group of idle auditors will be chosen as the auditor group of a certain slice. The AG forms a cluster, and only the GH interacts with the master. The GH is thus able to request the job information from the master. Therefore, it knows (1) the input data and output (i.e., the intermediate data before it is shuffled and sorted) of each mapper; (2) the input (i.e., the intermediate data after it is shuffled and sorted) and output (i.e., the final result) of each reducer. This information is essential for the auditors to check the correctness of each worker.
3. After the job begins, the GM will receive test tasks from the master, which will be notified once a worker finishes its task. Based upon the processing sequence of Map/Reduce, the mappers will finish first, and then, the reduce process is started. Therefore, in the initial period, mappers will be tested, and then reducers will be tested after the mappers.
4. After the GH receives a test task of checking a worker, it finds an available GM to carry out the test. Each check is executed as follows:
   (a) The GM will find corresponding input and output based upon the task type (i.e., Map/Reduce).
   (b) The GM will process the input data again and compare its output with the original one to check for inconsistency. If there is an inconsistency, it indicates that the worker being tested is malicious.

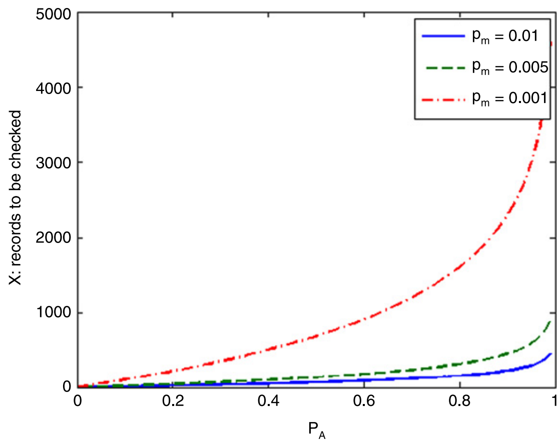

Fig. 5. P_A vs. x.

(c) The GM reports the test results to the GH, which will report back to the master.

5. If a worker is detected as malicious, the task it was assigned will be resubmitted to another worker so that the job continues.

5.3.4. *A-test with P-Accountability*

If the auditor is trustworthy and processes all the input of the worker, then the system can definitely determine whether the worker is malicious or not. This means that the task assigned to the worker is fully replicated. In the system view, the entire job will be executed twice, once by regular workers and once by the auditors. However, the high overhead of processing the job one time shows that it will take even longer and bring more overhead to process it twice. Therefore, instead of pursuing perfect accountability, the *A*-test provides *P*-Accountability [17], which gives the customers options. *P*-Accountability trades the degree of accountability for efficiency.

Definition of *P*-Accountability: we define *P*-Accountability as the probability that a malicious worker will be detected when it tampers with the processing result.

Let P_A denote *P*-Accountability, and let w denote the number of records in an input file, which can be either a raw data block for a map operation or a partition of intermediate results for a reduce operation. Assume that for any one record, a node has probability p_m of being malicious (i.e., tampering with the result); this will cause the corresponding output to be inaccurate. Variable x means that if we want to achieve P_A, we need to check at least x records. If $P_A = 1$, x is equal to w, meaning that the entire input file is checked, then

$$1 - (1 - p_m)^x \geq P_A. \tag{1}$$

We have

$$x \geq \left\lceil \log_{(1-p_m)}^{(1-P_A)} \right\rceil . \tag{2}$$

If $P_A = 0.9999$ and $p_m = 0.01$, we have $x = 917$, which means that only 917 records need to be checked. Under the assumption that a malicious node randomly (with probability p_m) tampers with the Map/Reduce result, we observe that x will not be affected by input data size, and only p_m and P_A will be related to x. Fig. 5 shows how x changes when P_A increases from 0 to 1. When P_A increases, the auditor needs to check more records to achieve a certain degree of P_A. We also observe that a smaller p_m indicates that there are more records that an auditor needs to check because the malicious node has less of a chance to tamper with the Map/Reduce operation.

Some features of the *A*-test are as follows:

(1) It is practical to implement the *A*-test, which makes the most of the existing properties of MapReduce. One important task for *A*-test is to acquire the input and output data of mappers/reducers, and the master has already kept this information.

(2) It is an online test, and this means that the malicious nodes will be detected as early as possible. The flow of the *A*-test ensures that a worker will be tested once it finishes.

(3) Workers do not know that they are being tested. Therefore, it is hard to take countermeasures to hide bad behavior.

(4) With *P*-Accountability, the *A*-test will be very efficient since a lower *P*-Accountability will significantly cut down the records that need to be checked.

Table 1
Input and output in MapReduce.

	Input	Worker	Output
Map	b_i	m_i	$\{h_{i,1}, h_{i,2}, \ldots, h_{i,k}\}$
Reduce	$\{h_{1,i}, h_{2,i}, \ldots, h_{n,i}\}$	r_i	o_i

One limitation is that false positives may occur if *P*-Accountability is less than 1. In real world MapReduce applications, we can adjust the parameters so that the probability of a false positive is close to zero.

6. Implementation of Accountable MapReduce

6.1. *Implementation of the master*

The master is the coordinator that holds all information necessary to conduct the *A*-test. The master has to maintain the following lists: mappers (we denote the mappers list as M), reducers (i.e., set R), input set (i.e., B), intermediate result set (i.e., H), and output set (i.e., O). Also, the master node is aware of every input/output relationship existing in the system. Therefore, a four-tuple set will be kept in order to respond to the requests from the AG head: $\{\langle\text{type, ID, input, output}\rangle\}$, where type is the worker type (i.e., mapper or reducer), ID is the worker's identity, and input and output depend on the worker type. Table 1 shows the input and output in MapReduce. Let the map output $\{h_{i,1}, h_{i,2}, \ldots, h_{i,k}\}$ be the intermediate result before it is shuffled and sorted; let the reduce input $\{h_{1,i}, h_{2,i}, \ldots, h_{n,i}\}$ be the intermediate result after it is shuffled and sorted. These sets are not complete in the beginning. Therefore, the master will maintain them while the job is running.

6.2. *Implementation of the auditor group*

Fig. 6 demonstrates the message flow during the *A*-test. Based upon the MapReduce primitives, the master will be notified whenever a worker is done with its job. To perform the *A*-test, the master also notifies the GH by sending Message 1. There are two cases of Message 1 based upon the worker type:

- *Case* 1: If the worker is a mapper m_i, then Message1 = (MAP, m_i, b_i, $\{h_{i,1}, h_{i,2}, \ldots, h_{i,k}\}$), which includes all information about the input and output of m_i. Message 2 is an assignment message of the *A*-test. The GH will randomly pick up a worker that has not yet been tested to generate a test assignment. Suppose that m_j is picked as the test object, then Message 2 = (MAP, m_j, b_j, $\{h_{j,1}, h_{j,2}, \ldots, h_{j,k}\}$). To accomplish the test, the GM reads input block b_j from DFS (i.e., action 3-a) intermediate result $\{h_{j,1}, h_{j,2}, \ldots, h_{j,k}\}$ from mapper m_j (i.e., action 3-b). The GM is then able to perform the *A*-test.
- *Case* 2: If the worker is a reducer r_i, then Message 1 = (REDUCE, r_i, $\{h_{1,i}, h_{2,i}, \ldots, h_{n,i}\}$, o_i). Suppose that r_j is also picked as the test object, then Message 2 = (REDUCE, r_j, $\{h_{1,j}, h_{2,j}, \ldots, h_{n,j}\}$, o_j); action 3-b: read $\{h_{1,j}, h_{2,j}, \ldots, h_{n,j}\}$ from the local disk of every mapper; action 3-c: read o_j from DFS. The GM is then ready to conduct the test.

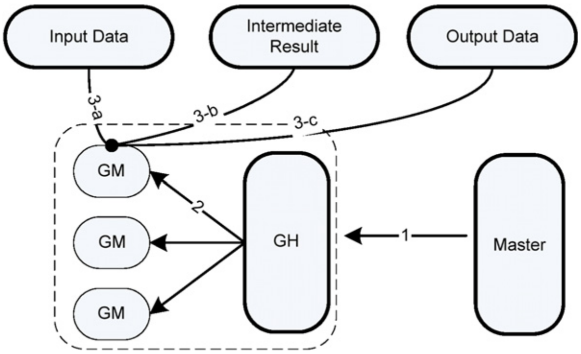

Fig. 6. Communication between the AG and other MapReduce entities. Numbers 1–3 represent messages.

6.2.1. Auditor Group Head (GH)

The GH maintains a list, L, of test tasks; L is a FIFO queue and will be updated in real time. When a GM is available, the GH will assign a new test task (i.e., the head of L) to it. When the GH is notified that worker w_i is done, the GH produces a test task for w_i immediately so that each worker will be tested at least once. The GH collects the test results from the GM and reports to the master if a bad node is detected.

6.2.2. Auditor Group Member (GM)

The intermediate result of MapReduce is stored in the workers' local disks, which are controlled by the workers. If these disks are accessed by other machines, then a malicious worker may become suspicious and take some actions in response. Therefore, the first time a CM reads data from these local disks, it makes a copy of the data on the DFS so that in future accesses, all data can be obtained from the DFS. For convenience, we use the same symbol to denote the intermediate result.

Algorithm 1. *A*-test

Algorithm: A-test
Require: p_m, P_A, task l
$x \leftarrow \left\lceil \log_{(1-p_m)}^{(1-P_A)} \right\rceil$ *// number of records to be checked*
If l.type = MAP
  For record i in l.input and $i < x$
    tmp←map(l.input[i])
    If tmp is not equal to l.output[i]
      Report inconsistent MAP
If l.type = REDUCE
    while(l.input[x].key = l.input[++x].key);
    for key k in l.input
      tmp←reduce(k, list(v))
      if tmp is not equal to l.output(k)
        report inconsistent reduce

7. *A*-test: Plan *B*

Instead of setting up dedicated auditors, another option is to choose a set of random idle machines from the server firm to perform the *A*-test for all customer groups. The design benefits and drawbacks can be described as follows:

- *Benefits*:
  - Better resource utilization. In a large data center, at any specific moment, there are a number of machines swiping in/out. The idle machines can be utilized to perform the *A*-test, which will not take long if the *P*-value is less than 1.
  - Plan *B* does not occupy customers' computation resource. Plan *B* separates customers' computation and accountability mechanism, and it maintains the independence of customers' business computing.
- *Drawbacks*:
  - Less predictable. The size of the dedicated auditor group is fixed so that it is easier to evaluate the *A*-test performance. If the *A*-test workload is too much, the admin may add more auditors to share the workload. In contrast, Plan *B* presents high uncertainty. The performance of the *A*-test depends upon the available machines at a particular time whereas the number of available machines is dynamic all the time. Therefore, it is more difficult for Plan *B* to make adjustments for performance management.

7.1. The design of *A*-test Plan *B*

With Plan *B*, the master maintains a pool of auditors, which consists of the idle machines. Once a machine is swiped out and reclaimed, it reports to the master, which puts it into the pool. When the master receives a task (Map/Reduce) completion message from the workers, it randomly picks a number of machines from the pool as auditors to perform the *A*-test. The auditors will be returned to the pool when they complete their test missions and when they send the results to the master, which can analyze and conclude whether the worker being tested is malicious or not.

The structure of Plan *B* differs from Plan *A* in that the auditors are not dedicated machines to perform *A*-test. The auditor pool is composed of idle machines that were just released from their jobs. This means that the pool is highly dynamic since once an auditor accomplishes the *A*-test, it will quit the auditor pool and be ready to receive new Map/Reduce tasks.

Once an idle machine reports to the master, it becomes a candidate auditor. However, there is no guarantee of the correctness of a candidate auditor because any worker could be malicious. Therefore, to verify its correctness, the master will generate a puzzle, and allow a candidate auditor solve it. A puzzle is a random Map/Reduce task generated by a program. If a candidate auditor solves the puzzle, it becomes a former auditor, which is allowed to accept *A*-test tasks from the master. Auditors will be challenged constantly during the *A*-test period. Every challenge is a puzzle that needs to be solved.

7.1.1. The design of puzzle generation

Puzzle generation is a reverse procedure of a Map/Reduce task. It takes the output of a Map/Reduce task as input and generates one possible input of a Map/Reduce task as its output, which will be the main content of the puzzle. For example, if wordcount is considered to be the host application and the input text for Map function is "good weather is good", then the Map output is {(good, 1), (weather, 1), (is, 1), (good, 1)}, and the reduce output is {(good, 2), (is, 1), (weather, 1)}. For the puzzle generation, either a Map puzzle or a reduce puzzle will be generated. Given {(good, 1), (weather, 1), (is, 1), (good, 1)} as the input, the output plain text has multiple possibilities, and the program will pick a random one like "is good good weather" as the puzzle. The process can be applied to generate a reduce puzzle as well. A puzzle makes no difference with the regular *A*-test tasks. An auditor will normally be challenged multiple times within the entire *A*-test. If any one of them shows an anomaly (i.e., results do not match), the auditor will be isolated to receive further investigation.

The puzzle generation can be specified as follows:

R_Map(list(K2, intermediate_value))→(K1, V1)
R_Reduce(list(V3))→(K2, list(intermediate_value))

In this process, R_Map and R_Reduce are two primitives for the Map puzzle and reduce puzzle, respectively. Notice that both R_Map and R_Reduce are one-to-many mappings (i.e., given certain input, there are multiple output versions). Therefore, R_Map and R_Reduce will randomly choose one possible result as a puzzle. With wordcount as an example, these two functions can be specified in Algorithms 2–3 as follows:

Algorithm 2: R_Map for the wordcount example

```
R_Map(list⟨K2, V2⟩):
  // K2: a word
  // V2: an integer with value 1
  String result = null;
  For each⟨key, value⟩pair in list:
    result.append(key + "");
```

Algorithm 3: R_Reduce for the wordcount example

```
R_Reduce(list⟨K3, V3⟩)
  // K3: a word
  // V3: number of occurrences of K3
  List result = null;
  For each⟨key, value⟩pair in list:
    int i = 0;
    for (; i < value; ++i)
      result.add(⟨key, 1⟩);
```

7.2. Other considerations

There is a chance that the auditors are malicious, and if they are, incorrect test results will be generated. For example, if worker A finishes its map task, and the master allows three auditors, $t1$, $t2$, and $t3$, test A's task. However, if there are malicious auditors in the three, then the test results may be inconsistent or even confusing. There are multiple possibilities for how the auditors behave: (1) all of them are clean; (2) some/all of them are malicious and behave like a worker A; (3) some/all of them are malicious, and none of them behave identical to A; (4) situations 3 and 4 combined.

There is overhead. The cost of Plan B is that since more than one auditor is involved for the A-test, the computational cost of the A-test is multiple times more than that of Plan A.

The auditor pool is highly dynamic because every idle worker only stays for a short while as a temporary auditor and other auditors swipe in/out frequently.

8. Analysis of Accountable MapReduce

8.1. How many workers and auditors should be assigned?

Accountable MapReduce introduces auditors to the platform. There is no doubt that the existence of auditors will introduce extra overhead to the entire computation process. The remaining question is how many workers and auditors should be assigned before MapReduce in order to accomplish the job efficiently. Based upon the plans that we discussed in previous sections, there are two cases based upon whether auditors are part of the customer working group.

In this section, we formulate the Optimal Worker and Auditor Assignment (OWAA) problem, which is aimed at minimizing the total processing time with the given MapReduce parameter set. Notations of the OWAA problem are given in Table 2.

8.2. Formulation of Optimal Worker and Auditor Assignment (OWAA) problem

We have the following assumptions for the OWAA problem:

- The reduced workload for each Reducer can be equally partitioned.
- We assume that there is no hardware difference between workers and auditors.

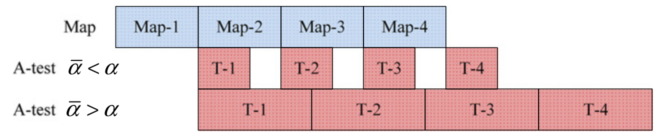

Fig. 7. Pipelining A-test and Map function.

- Workload is the only factor that determines the process time for Map/Reduce/A-test. This indicates that if two Mappers have a task workload with the same size, their processing time will be the same (the time can be regarded as average process time).

Fig. 7 shows the pipelining of the A-test and Map/Reduce. We can observe that normally each mapper will process multiple map tasks. According to the assumption, each map will take an equal amount of time, which is denoted by α. Each map task will be checked once it is finished. In this figure, T-1 represents the time slot to examine Map-1 through A-test. The average A-test time is denoted by $\bar{\alpha}$. If $\bar{\alpha} < \alpha$, T_m is mainly determined by ($\alpha \times$ the number of map tasks per mapper); otherwise T_m is mainly determined by ($\bar{\alpha} \times$ the number of A-test tasks per auditor). We can formulate the OWAA problem as follows:

Find three-tuple $\langle a, m, r \rangle$ to

$$\text{Minimize } T = T_m + T_s + T_r \tag{3}$$

in which $0 < m < n$, $0 < r < n$, $0 < a$, and m, r, a are integers.

$$T_m = \begin{cases} \left\lceil \dfrac{w_1}{a \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha & \bar{\alpha} > \alpha \\ \left\lceil \dfrac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \bar{\alpha} & a \geq m,\ \bar{\alpha} \leq \alpha \\ \left\lceil \dfrac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \left(\left\lceil \dfrac{w_1}{a \cdot b} \right\rceil - \left\lceil \dfrac{w_1}{m \cdot b} \right\rceil + 1 \right) \cdot \bar{\alpha} \\ \quad a < m,\ \bar{\alpha} \leq \alpha \end{cases} \tag{4}$$

$$T_s = f_s(m, w_2) \tag{5}$$

$$T_r = \begin{cases} \left\lceil \dfrac{w_2}{a \cdot b} \right\rceil \cdot \bar{\beta} + \beta & \bar{\beta} > \beta \\ \left\lceil \dfrac{w_2}{r \cdot b} \right\rceil \cdot \beta + \bar{\beta} & a \geq r,\ \bar{\beta} \leq \beta \\ \left\lceil \dfrac{w_2}{r \cdot b} \right\rceil \cdot \beta + \left(\left\lceil \dfrac{w_2}{a \cdot b} \right\rceil - \left\lceil \dfrac{w_2}{r \cdot b} \right\rceil + 1 \right) \cdot \bar{\beta} \\ \quad a < r,\ \bar{\beta} \leq \beta \end{cases} \tag{6}$$

$$h = \left\lceil \log_{(1-p_m)}^{(1-p_A)} \right\rceil \tag{7}$$

$$w_2 = f_w(w_1) \tag{8}$$

$$\alpha = f_\alpha(b) \tag{9}$$

$$\beta = f_\beta(b) \tag{10}$$

$$\bar{\alpha} = f_{\bar{\alpha}}(h) \tag{11}$$

$$\bar{\beta} = f_{\bar{\beta}}(h). \tag{12}$$

Eq. (3) gives the objective function T (i.e., total processing time of a job), which consists of the map phase time (i.e., T_m), the shuffle phase time (i.e., T_s), and the reduce phase time (i.e., T_r). Eq. (4) calculates T_m. Based on our analysis on Fig. 7, if $\bar{\alpha} > \alpha$, T_m is mainly determined by the A-test time, which is obtained from $\lceil w_1/(a \cdot b) \rceil \cdot \bar{\alpha}$. If $\bar{\alpha} < \alpha$, T_m is mainly determined by the map time, which is calculated from $\lceil w_1/(m \cdot b) \rceil \cdot \alpha$. In addition, the number of auditors affects the calculation of T_m. If the number of auditors are no less than the number of mappers (i.e., $a \geq m$), one $\bar{\alpha}$ is added into T_m (e.g., T-4 in Fig. 7); if $a < m$, each auditor will be assigned more A-test tasks, and the number of these extra A-test tasks can be calculated from $\lceil w_1/(a \cdot b) \rceil - \lceil w_1/(m \cdot b) \rceil + 1$. Similarly, we can obtain T_r. Eqs. (5), (8)–(12) are functions without

Table 2
Notations.

T	Total processing time of a job.
m	Number of mappers.
w_1	The initial job workload (i.e., data set volume before Map function).
w_2	The intermediate job workload (i.e., data set volume after Map function). Let w_2 be a function of w_1, so we have $w_2 = f_w(w_1)$.
T_m	Map phase time. It covers the entire Map phase and *A*-test for Map phase.
T_s	Shuffle time. Since $T_s \propto m$, and $T_s \propto w_2$, we have function $T_s = f_s(m, w_2)$.
T_r	Reduce phase time. It covers the entire reduce phase and *A*-test for reduce phase.
T_0	Total processing time that required by a customer. For example, the customer may need the job accomplished in 20 h, i.e., $T_0 = 20$ h.
h	Number of records to be checked during *A*-test. Based on Section 5, we have $h = \left\lceil \log_{(1-p_m)}^{(1-p_A)} \right\rceil$. The larger h is, the longer it needs for the *A*-test.
a	Number of auditors.
n	Number of workers in a customer's working group; n is constant.
r	Number of reducers.
b	Size of each data block; the default value is 64 MB.
α	Process time for one individual map task. α primarily depends on b, the host application, data set type, machine computation capability (e.g., CPU number, RAM size, etc.). In this case, we only keep factor b, i.e., $\alpha = f_\alpha(b)$.
β	Process time for one individual reduce task. Similar to α, we let $\beta = f_\beta(b)$.
$\bar{\alpha}$	Process time for one individual *A*-test for a map task. Since $\bar{\alpha} \propto w \propto h$, we let $\bar{\alpha}$ be a function of h, i.e., $\bar{\alpha} = f_{\bar{\alpha}}(h)$.
$\bar{\beta}$	Process time for one individual *A*-test for a reduce task. Since $\bar{\beta} \propto w \propto h$, we let $\bar{\beta}$ be a function of h, i.e., $\bar{\beta} = f_{\bar{\beta}}(h)$.

concrete forms. To simplify the problem, we further assume the following functions are linear. We have

$$w_2 = f_w(w_1) = k_w \cdot w_1 \tag{13}$$

$$T_s = f_s(m, w_2) = k_s \cdot \frac{w_2}{m} \tag{14}$$

$$\alpha = f_\alpha(b) = k_\alpha \cdot b \tag{15}$$

$$\beta = f_\beta(b) = k_\beta \cdot b \tag{16}$$

$$\bar{\alpha} = f_{\bar{\alpha}}(h) = k_{\bar{\alpha}} \cdot h \tag{17}$$

$$\bar{\beta} = f_{\bar{\beta}}(h) = k_{\bar{\beta}} \cdot h. \tag{18}$$

The coefficients of the above linear functions will be specified in evaluation.

8.3. Solve the OWAA problem

Accountability can be regarded as one type of quality of service that can be selected by customers with multiple service levels. Therefore, when the accountability degree increases, it needs a longer amount of time to accomplish the job. Based upon the plans we designed, there are two scenarios in which the relations among m, r, and a differ.

Scenario 1: the auditors are dedicated testing machines that are not included in the customer group working machines (i.e., $m + r = n$). In this case, the auditors are external to the customer working group. Therefore, with more auditors, the faster the *A*-test will perform. A bound a_0 is introduced to limit the number of auditors. We then have $a \le a_0$.

Scenario 2: the auditors are included in the customer group working machines (i.e., $m+r+a = n$). We consider the AG (Auditor Group) size to be the key factor that affects the processing time. The impact of the AG size on the processing time is twofold. First, since the AG is constantly used to conduct the *A*-test, it occupies some computing resources that are supposed to run MapReduce tasks. With a larger AG size, it will take longer to accomplish a certain amount of data set processing. On the other hand, the AG size determines the time of the *A*-test, which is a major part of the total processing time. Because of the larger AG, the test will go faster. Therefore, there is a tradeoff of the AG size.

Based on the formulation, we have four cases to obtain T:

8.3.1. Case A: if $\bar{\alpha} > \alpha$, $\bar{\beta} > \beta$

Combining Eqs. (3)–(6), we have

$$T = \left\lceil \frac{w_1}{a \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{a \cdot b} \right\rceil \cdot \bar{\beta} + \beta. \tag{19}$$

There are two sub-cases (i.e., A1 and A2), each representing a scenario:

A1: If $m + r = n$, then all terms but $k_s \cdot \frac{w_2}{m}$ are relevant to m or r; therefore, when $m = n - 1$, $r = 1$, and $a = a_0$, we have a minimum of T as follows:

$$T_{\min} = \left\lceil \frac{w_1}{a_0 \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha + k_s \cdot \frac{w_2}{n-1} + \left\lceil \frac{w_2}{a \cdot b} \right\rceil \cdot \bar{\beta} + \beta. \tag{20}$$

A2: If $m + r + a = n$, we have $r = 1$, and let $a = n - m - 1$. Then, Eq. (19) can be written as:

$$T = \left\lceil \frac{w_1}{(n-m-1) \cdot b} \right\rceil \cdot \bar{\alpha} + \alpha + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{(n-m-1) \cdot b} \right\rceil \cdot \bar{\beta} + \beta. \tag{21}$$

Since $1 \le m \le n - 2$, we can simplify (21) to the following:

$$T = \frac{c_1}{c_2 - m} + \frac{c_3}{m} + c_4, \tag{22}$$

in which $c_1 = \left(w_1 \cdot \bar{\alpha} + w_2 \cdot \bar{\beta}\right)/b$, $c_2 = n - 1$, $c_3 = k_s \cdot w_2$, $c_4 = \alpha + \beta$. Therefore, T is transformed to a function of a single variable. Based on calculus, we have

$$T' = \frac{c_1}{(c_2 - m)^2} - \frac{c_3}{m^2}.$$

Let $T' = 0$, since we have $c_1 > 0$, $c_2 - m > 0$, $c_3 > 0$, and $m > 0$. By solving $T' = 0$, we have $m = m_0 = \left(c_2 \cdot \sqrt{c_3}\right) / \left(\sqrt{c_1} + \sqrt{c_3}\right)$.

$T'' = \frac{2c_1}{(c_2-m)^3} + \frac{2c_3}{m^3}$, $T''|_{m_0} > 0$, therefore, T can achieve the minimum at this point. Since m, r, and a are integers. If $0 < m_0 < 1$, then $\langle m = 1,\ r = 1, a = n - 2\rangle$ is the optimal solution. If $m_0 > n - 2$, then $\langle m = n - 2, r = 1, a = 1\rangle$ is the optimal solution.

If $1 \le m_0 \le n-2$, then $\langle m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \ge T(\lfloor m_0 \rfloor), \end{cases}$ $r = 1, a = n - m - 1\rangle$ is the optimal solution.

8.3.2. Case B: if $\bar{\alpha} > \alpha$, $\bar{\beta} < \beta$

Based upon the *A*-test scheme, this case is impossible because once p_A is determined, it has the same effect on *A*-test time for both Map and Reduce. Therefore, it can either be $\bar{\alpha} > \alpha$, $\bar{\beta} > \beta$, or $\bar{\alpha} < \alpha$, $\bar{\beta} < \beta$. We can remove both case B and case C for this reason.

8.3.3. Case C: if $\bar{\alpha} < \alpha$, $\bar{\beta} > \beta$

Based on the analysis on case B, case C is impossible.

8.3.4. Case D: if $\bar{\alpha} < \alpha$, $\bar{\beta} < \beta$

There are four sub-cases, and each sub-case is discussed in two scenarios.

D1: If $m \leq a, r \leq a$, then

$$T = \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \bar{\alpha} + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta + \bar{\beta}. \tag{23}$$

D11: If $m + r = n$, T is not related to a, meaning that a can be as small as possible. We have $a = \min\{a_0, \max\{m, r\}\}$.

T can be simplified as $T = \frac{p_1}{p_2 - m} + \frac{p_3}{m} + p_4$, where $p_1 = (w_2 \cdot \beta)/b$, $p_2 = n$, $p_3 = (w_1 \cdot \alpha)/b + k_s \cdot w_2$, and $p_4 = \bar{\alpha} + \bar{\beta}$. Similar to case A12, when $m = m_0 = \frac{p_2 \cdot \sqrt{p_3}}{\sqrt{p_1} + \sqrt{p_3}}$, T can achieve the minimum.

If $0 < m_0 < 1$, then $\langle m = 1, r = n - 1, a = \min\{a_0, n - 1\}\rangle$ is the optimal solution.

If $m_0 > n - 2$, then, $\langle m = n - 1, r = 1, a = \min\{a_0, n - 1\}\rangle$ is the optimal solution.

If $1 \leq m_0 \leq n - 2$, then $\langle m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \geq T(\lfloor m_0 \rfloor), \end{cases}$ $r = n - m$, $a = \min\{a_0, \max\{m, r\}\}\rangle$is the optimal solution.

D12: If $m + r + a = n$, T can be simplified as

$$T = \frac{p_1}{p_2 - a - m} + \frac{p_3}{m} + p_4. \tag{24}$$

Therefore, T is a function of two variables. To find the minimum of T, we have

$$\frac{\partial T}{\partial a} = \frac{p_1}{(p_2 - m - a)^2}, \qquad \frac{\partial T}{\partial m} = \frac{p_1}{(p_2 - m - a)^2} - \frac{p_3}{m^2},$$
$$\frac{\partial^2 T}{\partial a^2} = \frac{2p_1}{(p_2 - m - a)^3}, \qquad \frac{\partial^2 T}{\partial m^2} = \frac{2p_1}{(p_2 - m - a)^3} + \frac{2p_3}{m^3},$$
$$\frac{\partial T}{\partial a \partial m} = -\frac{2p_1}{(p_2 - m - a)^3}.$$

Let $\frac{\partial T}{\partial a} = 0$, and $\frac{\partial T}{\partial m} = 0$, we have $\begin{cases} \frac{p_1}{(p_2 - m - a)^2} = 0 \\ \frac{p_1}{(p_2 - m - a)^2} - \frac{p_3}{m^2} = 0. \end{cases}$

Since $p_1 > 0$, there is no solution of the above equation set. Therefore, there is no extreme point for T. By analyzing the trend of function T, we conclude that T is at its minimum when the following statements hold: (1) $m/r = \sqrt{p_3}/\sqrt{p_1}$, (2) $m + r + a = n$, (3) $m \leq a, r \leq a$, (4) a can be as small as possible. The optimal solution is $\langle m = \left\lfloor \min(\frac{n}{2 + \sqrt{p_1/p_3}}, \frac{n}{1 + 2\sqrt{p_1/p_3}}) \right\rfloor, r = \left\lfloor m\sqrt{p_1/p_3} \right\rfloor, a = n - m - r\rangle$.

D2: if $r > a \geq m$, then

$$T = \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \bar{\alpha} + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta + \left(\left\lceil \frac{w_2}{a \cdot b} \right\rceil - \left\lceil \frac{w_2}{r \cdot b} \right\rceil + 1 \right) \cdot \bar{\beta}.$$

D21: If $m + r = n$, T can be simplified as $T(a, m) = \frac{q_1}{n - m} + \frac{q_2}{m} + \frac{q_3}{a} + q_4$, where $q_1 = \frac{w_2}{b}(\beta - \bar{\beta})$, $q_2 = \frac{w_1 \cdot \alpha}{b} + k_s \cdot w_2$, $q_3 = \frac{w_2 \cdot \bar{\beta}}{b}$, $q_4 = \bar{\alpha} + \bar{\beta}$. We also have

$$\frac{\partial T}{\partial a} = -\frac{q_3}{a^2}, \qquad \frac{\partial T}{\partial m} = \frac{q_1}{(n - m)^2} - \frac{q_2}{m^2},$$
$$\frac{\partial^2 T}{\partial a^2} = \frac{q_3}{a^3}, \qquad \frac{\partial^2 T}{\partial m^2} = \frac{q_1}{(n - m)^3} + \frac{q_2}{m^3}, \qquad \frac{\partial T}{\partial a \partial m} = 0.$$

Let $\frac{\partial T}{\partial a} = 0$, and $\frac{\partial T}{\partial m} = 0$, we have no solution for a and m, meaning that there is no extreme point of T. By analyzing the trend of function T, we conclude that T is at its minimum when the following statements hold: (1) a is as large as possible but $a \leq a_0$, (2) $r > a \geq m$, (3) $\frac{\partial T}{\partial m} = 0$, from which we have $m = m_0 = (n\sqrt{q_2}) / (\sqrt{q_1} + \sqrt{q_2})$. We can then determine the optimal solution of T:

If $0 < m_0 < 1$, then $m = 1, r = n - 1, a = \min\{a_0, r - 1\}$ is optimal.

If $m_0 > n - 2$, then $\langle m = n - 2, r = 2, a = \min\{a_0, 1\}\rangle$ is optimal.

If $1 \leq m_0 \leq n - 2$, then $\langle m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \geq T(\lfloor m_0 \rfloor), \end{cases}$ $r = n - m$, $a = \min\{a_0, r - 1\}\rangle$is optimal.

D22: If $m + r + a = n$, we have

$T(a, m) = \frac{q_1}{n - m - a} + \frac{q_2}{m} + \frac{q_3}{a} + q_4$. Then

$$\frac{\partial T}{\partial a} = \frac{q_1}{(n - m - a)^2} - \frac{q_3}{a^2}, \qquad \frac{\partial T}{\partial m} = \frac{q_1}{(n - m - a)^2} - \frac{q_2}{m^2},$$
$$\frac{\partial^2 T}{\partial a^2} = \frac{2q_1}{(n - m - a)^3} - \frac{2q_3}{a^3},$$
$$\frac{\partial^2 T}{\partial m^2} = \frac{2q_1}{(n - m - a)^3} - \frac{2q_2}{m^3},$$

$\frac{\partial T}{\partial a \partial m} = \frac{2q_1}{(n - m - a)^3}$. Let $\frac{\partial T}{\partial a} = 0$, and $\frac{\partial T}{\partial m} = 0$, we have

$$\begin{cases} \frac{q_1}{(n - m - a)^2} - \frac{q_3}{a^2} = 0 \\ \frac{q_1}{(n - m - a)^2} - \frac{q_2}{m^2} = 0. \end{cases}$$

By solving the equation set, we have

$$\begin{cases} m = m_0 = n \cdot \sqrt{\frac{q_2}{q_1 + q_2 + q_3}} \\ a = a_0 = n \cdot \sqrt{\frac{q_3}{q_1 + q_2 + q_3}}. \end{cases}$$

Let $F(a, m) = \left(\frac{\partial T}{\partial a \partial m}\right)^2 - \left(\frac{\partial^2 T}{\partial a^2}\right) \cdot \left(\frac{\partial^2 T}{\partial m^2}\right)$, then we can compute $F(a_0, m_0) < 0$, and $\frac{\partial^2 T}{\partial a^2}|_{a_0} > 0$. Therefore, T can achieve its minimum. The optimal solution for this case is:

If $a_0 \geq m_0$, and $r = n - m_0 - a_0 > a_0$, then m will be $\lfloor m_0 \rfloor$ or $\lceil m_0 \rceil$, a will be $\lfloor a_0 \rfloor$ or $\lceil a_0 \rceil$, whichever makes the minimal T; and $r = n - m - a$.

Otherwise, there is no optimal solution.

D3: if $m > a \geq r$, then

$$T = \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \left(\left\lceil \frac{w_1}{a \cdot b} \right\rceil - \left\lceil \frac{w_1}{m \cdot b} \right\rceil + 1 \right) \cdot \bar{\alpha} + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta + \bar{\beta}.$$

D31: If $m + r = n$, T can be simplified as $T(a, m) = \frac{s_1}{n - m} + \frac{s_2}{m} + \frac{s_3}{a} + s_4$, which is similar to case D21. We can determine the optimal solution as follows:

Let $m = m_0 = (n\sqrt{s_2}) / (\sqrt{s_1} + \sqrt{s_2})$.

If $m_0 < n - m_0$, there is no optimal solution.

If $m_0 \geq n - m_0$, then $\langle m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \geq T(\lfloor m_0 \rfloor), \end{cases}$ $r = n - m$, $a = \min\{a_0, m - 1\}\rangle$is the optimal solution.

D32: If $m + r + a = n$, $T(a, m) = \frac{s_1}{n - m - a} + \frac{s_2}{m} + \frac{s_3}{a} + s_4$, where $s1 = \frac{w_2 \cdot \beta}{b}$, $s_2 = \frac{w_1}{b} \cdot (\alpha - \bar{\alpha}) + k_s \cdot w_2$, $s_3 = \frac{w_1 \cdot \bar{\alpha}}{b}$, $s_4 = \bar{\alpha} + \bar{\beta}$. This case is similar to D22.

D4: if $m > a, r > a$, then

$$T = \left\lceil \frac{w_1}{m \cdot b} \right\rceil \cdot \alpha + \left(\left\lceil \frac{w_1}{a \cdot b} \right\rceil - \left\lceil \frac{w_1}{m \cdot b} \right\rceil + 1 \right) \cdot \bar{\alpha} + k_s \cdot \frac{w_2}{m} + \left\lceil \frac{w_2}{r \cdot b} \right\rceil \cdot \beta + \left(\left\lceil \frac{w_2}{a \cdot b} \right\rceil - \left\lceil \frac{w_2}{r \cdot b} \right\rceil + 1 \right) \cdot \bar{\beta}.$$

T can be simplified as $T(a, m) = \frac{g_1}{r} + \frac{g_2}{m} + \frac{g_3}{a} + g_4$, in which $g_1 = \frac{w_2}{b} \cdot (\beta - \bar{\beta})$, $g_2 = \frac{w_1}{b} \cdot (\alpha - \bar{\alpha}) + k_s \cdot w_2$,

$$g_3 = \frac{w_1 \cdot \bar{\alpha}}{a} + \frac{w_2 \cdot \bar{\beta}}{b}, \qquad g_4 = \bar{\alpha} + \bar{\beta}.$$

Table 3
Solution table.

Case #		Condition	
		$m+r=n$	$m+r+a=n$
A: $\bar{\alpha} > \alpha, \bar{\beta} > \beta$		$\surd$	$\surd$
B: $\bar{\alpha} > \alpha, \bar{\beta} < \beta$		N/A	N/A
C: $\bar{\alpha} < \alpha, \bar{\beta} > \beta$		N/A	N/A
D: $\bar{\alpha} < \alpha, \bar{\beta} < \beta$	$m \le a, r \le a$	$\surd$	$\surd$
	$r > a \ge m$	$\surd$	$\surd$
	$m > a \ge r$	$\surd$	$\surd$
	$m > a, r > a$	$\surd$	$\surd$

Table 4
Default parameter settings.

T_0	20 h	b	64 MB
w_1	2 TB	p_m	0.1
p_A	0.9	n	100
k_w	1.4	k_s	10^{-4}
k_α	0.16	k_β	0.23
$k_{\bar{\alpha}}$	10^{-4}	$k_{\bar{\beta}}$	10^{-4}

Table 5
Numeric results.

Para.		Cond.					
		$m+r=n$			$m+r+a=n$		
		Case	$\langle m, r, a\rangle$	$T_{\min}$	Case	$\langle m, r, a\rangle$	$T_{\min}$
Default		D11	$\langle 41, 59, 59\rangle$	5.2	D42	$\langle 57, 39, 4\rangle$	6.25
P_A	0.5	D11	$\langle 41, 59, 59\rangle$	5.2	D42	$\langle 57, 41, 2\rangle$	5.99
	0.999	D11	$\langle 41, 59, 59\rangle$	5.2	D42	$\langle 57, 36, 7\rangle$	6.7
	1	A1	$\langle 99, 1, 50\rangle$	5.89	A2	$\langle 1, 1, 98\rangle$	8.32
N	50	D11	$\langle 20, 30, 30\rangle$	10.4	D42	$\langle 28, 20, 2\rangle$	12.34
B	128	D11	$\langle 41, 59, 59\rangle$	5.2	D42	$\langle 57, 41, 2\rangle$	6.04
w_1	8	D11	$\langle 41, 59, 59\rangle$	20.2	D42	$\langle 57, 39, 4\rangle$	25.03

D41: If $m + r = n$, the case is similar to case D21. We can determine the optimal solution as follows:

Let $m = m_0 = \left(n\sqrt{g_2}\right) / \left(\sqrt{g_1} + \sqrt{g_2}\right)$.

If $0 < m_0 < 1$, there is no optimal solution.

If $m_0 > n - 2$, there is no optimal solution.

If $1 \le m_0 \le n-2$, then $\langle m = \begin{cases} \lceil m_0 \rceil & T(\lceil m_0 \rceil) < T(\lfloor m_0 \rfloor) \\ \lfloor m_0 \rfloor & T(\lceil m_0 \rceil) \ge T(\lfloor m_0 \rfloor), \end{cases} r = n - m,\ a = \min\{a_0, \min(m, r)\}\rangle$ is the optimal solution.

D42: If $m + r + a = n$, the case is similar to case D22.

The problem set and its solutions can be summarized in Table 3.

Taking wordcount as the host application, the parameters can be specified In Table 4:

For wordcount, each 'word' will be transformed to a key–value pair like 'word, 1', meaning that 2 letters (i.e., ',' and '1') are added. We assume that the average length of an English word is 5; the size of each word will be increased by 2/5, and we have $k_w = 7/5 = 1.4$. Based on Table 3, we inject the parameters into the OWAA problem, and obtain the numeric results in Table 5.

Figs. 8–10 describe how T changes in different cases when parameters are default values. Fig. 8 shows case A2 with default setting. In this case, we have $T = \frac{1060400}{99-m} + \frac{280}{m} + 24.96$. When m increases, T keeps increasing. Therefore, $\langle m = r = 1, a = n-m-r\rangle$ is the optimal solution. Fig. 9 shows case D11 with default setting. In this case, we have $T = \frac{644000}{100-m} + \frac{320280}{m} + 0.046$, T can achieve the minimum, which is 5.2 h. Fig. 10 shows case D42 with the default setting, we have $T(a, m) = \frac{642993}{100-m-a} + \frac{319561}{m} + \frac{1725}{a} + 0.046$, and $T_{\min} = 6.25$ h.

9. A sentinel-based verification scheme

Juels [30] et al. proposed a sentinel-based scheme for data possession. For A-test, sentinels can also be used for accuracy verification. In this section, we propose a sentinel-based verification scheme as another implementation for the A-test. Unlike the original A-test approach, which is generally a replay-and-match approach, we embed sentinels into the input data. If a malicious worker manipulates the data, there is a chance that the sentinels are corrupted as well. The output shows how well the sentinels are treated during processing, and it also provides evidence of misbehavior.

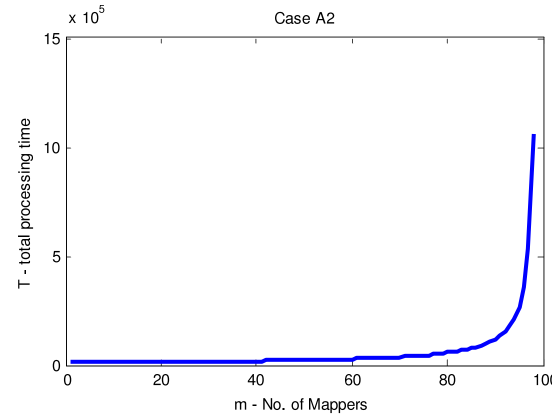

Fig. 8. Case A2 with default setting.

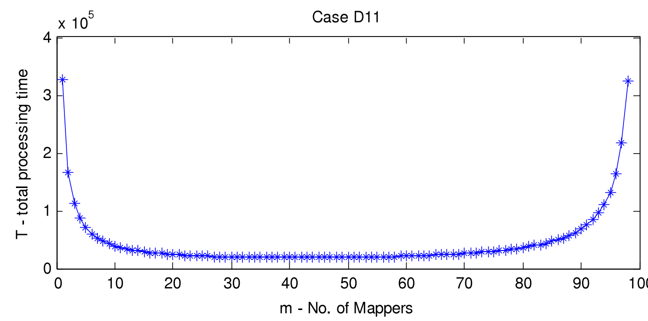

Fig. 9. Case D11 with default setting.

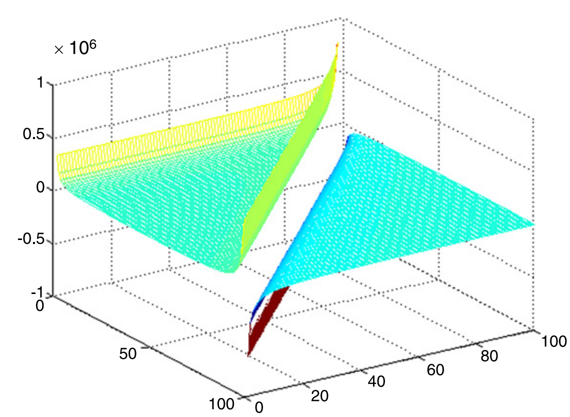

Fig. 10. Case D42 with default setting.

9.1. Motivation

The A-test is efficient and effective under the assumption that a malicious worker will randomly tamper with the Map/Reduce function (i.e., each key–value pair has an equal chance of being falsified). Under this assumption, the A-test can achieve good performance by only checking the initial part of the entire data. However, hackers/malicious users may not behave this way; they can choose their styles of manipulation for the data. Therefore, where and how a piece of data will be falsified is incomprehensible in reality. In this section, we change the assumption to "input data can be manipulated in anywhere with any manner". The *A*-test has limitations since it only scans the initial part of the input file and allows the rest to stay out of the law.

9.2. Design of sentinel-based verification

There are a few requirements that a sentinel should satisfy: (1) a sentinel can be processed by regular Map/Reduce functions (i.e., it is essentially a key–value pair that is valid to Map/Reduce). (2) A sentinel should own a unique key that will not mix with other keys because a key's uniqueness facilitates the tracing and verification. Otherwise all duplicated keys will be reduced to one key through the reduce function. (3) A sentinel can be inserted into any place of the input file and removed after the process without much effort. (4) A sentinel cannot break the original content of an input file to prevent it from introducing new errors. For example, if a sentinel is inserted within a word in a text input file, then the word will be split. A word "good" may become "go /sentinel/ od", which will be treated as three distinct words by the Map/Reduce function. (5) A working machine has no idea where and how a sentinel is embedded.

Fig. 11 describes the sentinel-based verification scheme. Two new functions are defined: S_Insert (i.e., sentinel insert) and S_Verify (i.e., sentinel verification). The S_Insert and S_Verify functions are implemented as wrappers to the data set. They do not affect any other processes during Map/Reduce. The data set has already been embedded with sentinels before Map/Reduce. Therefore, requirement 5 can be satisfied.

9.2.1. S_Insert function

A parameter called insert frequency (F) is defined to determine how often a sentinel should be inserted. "$F = 100$" means one sentinel will be inserted for every 100 keys. The sentinel generation process can satisfy requirements 1 and 2. The way a sentinel is inserted is key-based, and this means that we can make sure no keys will be broken during insertion (i.e., requirement 4 is satisfied). S_insert is described in Algorithm 4.

Algorithm 4. S_insert

```
S_Insert(int F, Data block):
  // F: insert frequency
  get reader from block;
  int count = 0;
  Index start, end;
  Key key = null;
  Bool insert = true;
  Reference ref = new Reference(); // create a reference copy
  while ((key = reader.nextKey())! = null)
    ++count;
    if (insert) start = key.index();
    if (count % F == 0)
      // insert a sentinel among the last F keys
      end = key.index();
      Sentinel s = getSentinel ();
      Ref.add(s);
      int pos = (new Random()).nextInt(F);
      // insert s to a random position between start and end
      insert(block, start, end, pos, s);
```

During S_insert, every time a sentinel is inserted into the data set, it is also added into a data structure called the reference copy, which will be used for verification purposes. When the data set is being Mapped/Reduced, the sentinels in the reference copy are processed in the same way but not in the same machine where the data set is processed. Keys differ from application to application. Even plain text may choose various encoding standards (e.g., ASCII, Unicode, etc.). Some applications may limit keys within a small range (e.g., key space is 8 bit). Therefore, it is not possible to design a universal method for sentinel generation.

9.2.2. S_Verify function

The purpose of S_Verify is twofold: (1) to verify the correctness of Map/Reduce task and generate evidence if inconsistency is reported and (2) to remove sentinels from data set after verification. The S_Verify primitive is described in Algorithm 5.

Algorithm 5. S_verify

```
S_Verify(DataSet ds, ReferenceCopy rc):
  for Sentinel s in rc
    Sentinel tmp = findSentinel(ds, s.getKey());
    if (!s.match(tmp))
      EvidenceGen(s, tmp, ds);
      break;
    else // s matches tmp
      ds.remove(tmp)
```

To verify the accuracy of a task, we need to compare the sentinels in the reference copy and the sentinels embedded in the output data to check if they can match. If they do not match, which means that the Mapper/Reducer has manipulated the data, then a piece of evidence will be generated to prove the inconsistency.

9.2.3. Judgments

Pros—the good part is that there is no need to replay the Map/Reduce function.

Cons—It is not possible to guarantee complete accountability because this scheme does not check the accuracy of the original data. Therefore, false positives may exist.

9.2.4. Performance analysis

Since there is one sentinel for every F key–value pairs, the probability of any key–value pair being a sentinel is $1/F$. When a malicious worker misbehaves, a continuous piece of data is likely to be manipulated. Let l denote the length of a continuous piece of data (i.e., the number of key–value pairs is l) that has been falsified. The chance that there are no sentinels in the data piece is $(1-1/F)^l$. Therefore, the probability that a corrupted data piece with length l will be detected is $1-(1-1/F)^l$. The detection probability is plotted in Fig. 12.

10. Evaluation

We have implemented a prototype of Accountable MapReduce based upon Hadoop [7] and have tested it in both our local lab and the Utah Emulab testbed.

10.1. Experiment setup

We set up a VLAN with 20 PCs in the Emulab testbed to deploy the Accountable MapReduce. We simulated some malicious machines in the system to perform Map/Reduce mess-ups. The MapReduce application that we are using in our experiments is wordcount.

10.2. Experiment result

Fig. 13 depicts how the AG size affects the processing time when P-Accountability $= 1$. Because this experiment's purpose is to test how the *A*-test will impact processing time, we did not insert any malicious nodes. We built a 5-node cluster to run MapReduce, and

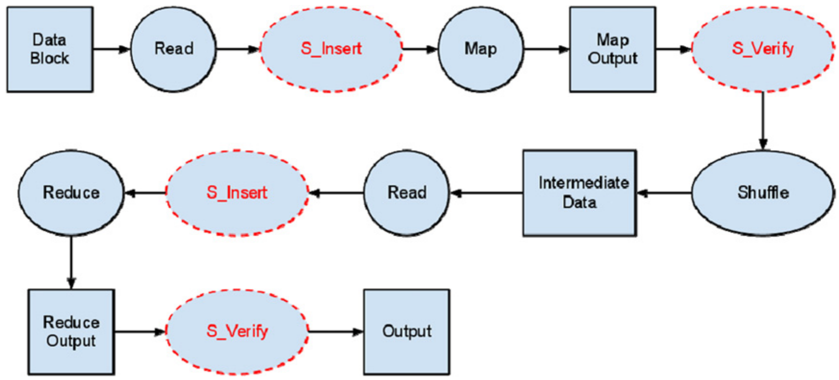

Fig. 11. Sentinel-based verification.

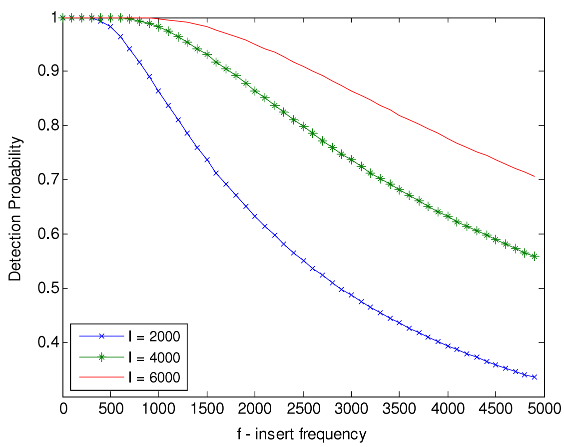

Fig. 12. Detection probability with sentinel-based verification.

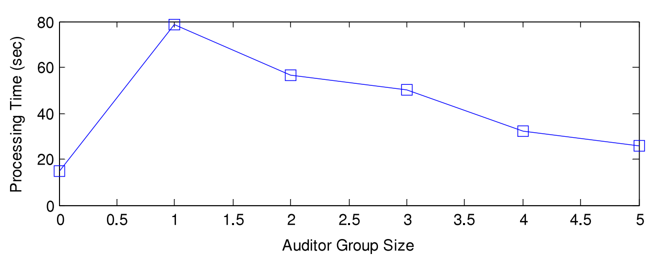

Fig. 13. AG size and processing time when $P = 1$.

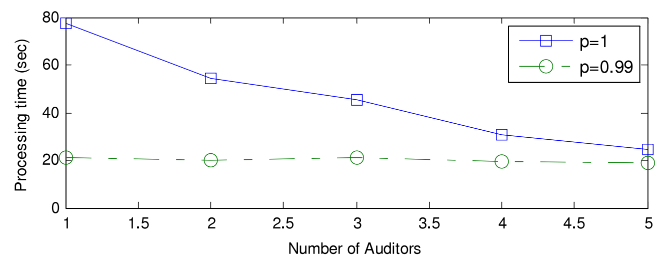

Fig. 14. *P*-Accountability and processing time.

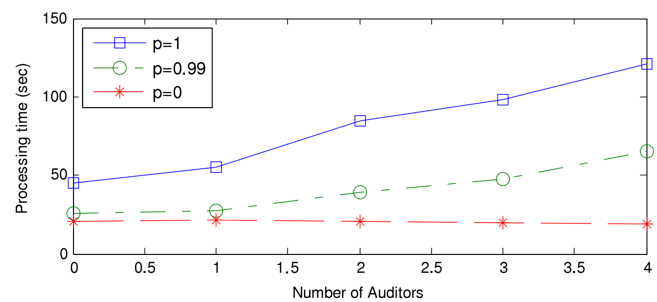

Fig. 15. Malicious nodes and processing time.

we varied the size of AG to observe how the processing time would change. This shows that when there are no auditors (AG# = 0), the MapReduce system is not accountable, but the processing time is minimal because there is no extra overhead added by the *A*-test. The increase of the AG size will bring extra overhead to the job (i.e., they conduct the *A*-test). Since $P = 1$ in this case, the job will be entirely duplicated. The more auditors we have, the quicker the *A*-test will finish. A straightforward observation is that when the AG size is equal to the number of workers, each worker will be tested by an individual auditor so that some waiting time will be saved. Also the processing time will be minimal.

Fig. 14 shows how *P*-Accountability affects the processing time. $P = 0$ means that the system is not accountable. We also reduce the *P*-value from 1 to 0.99 to observe how this change will affect the processing time. We still use a 5-node cluster to run wordcount (data size = 50 M). In order to rule out the interference of the re-submit/re-process time, this experiment is also free of malicious nodes. The result shows that the lower *P*-Accountability decreases significantly the workload of the *A*-test because fewer records will be checked. As a result, each mapper/reducer will get tested very quickly. Also, the AG size will not become an issue since equivalent performance can be achieved with fewer auditors.

When malicious nodes are taken into account, we need to add the re-submit/re-process/re-test time into the total processing time. The cluster still contains 5 workers. The AG size is 2, and the input data is 100 M. In Fig. 15, we compared the total processing times when $P = 0, 0.99$, and 1. In the chart, the gap between curve ($P = 1$) and curve ($P = 0.99$) means that the test time is greatly reduced. It is also obvious that with the more malicious nodes we have, the processing time will be longer because once malicious nodes are detected, they will be exposed and no longer take the Map/Reduce tasks. As a result, the remaining good workers will carry out the tasks that need to be reprocessed. The AG must also re-test the tasks.

False positives may exist when *P* is less than 1 because the Map/Reduce task will not be fully tested. However, based upon our assumption, if the malicious nodes randomly cause errors during Map/Reduce, then they can be detected with high probability (determined by *P*). Another point is that the major performance improvement has been saved even when *P* is close to 1 (e.g., $P = 0.999$). In our experiment (20 repetitions), we found that when $P = 0.99$ and when the probability of a bad node altering a record (denoted by p_m) is 0.01, the *A*-test runs very well without missing any malicious node. We also tested an extreme case in which $P = 0.99$ and $p_m = 0.0001$, and this means that the malicious node will alter one record out of every 10 000 records. In this case, we did observe false positives.

11. Conclusion

In this paper, we proposed Accountable MapReduce as an additional component for the current MapReduce model to support accountability. Accountable MapReduce employs an Auditor Group (AG) to conduct an A-test on every worker in the system without being noticed by the workers. If malicious behavior occurs, the AG is able to detect it and provide verifiable evidence. To improve the performance, we introduce P-Accountability in the A-test to trade the degree of accountability with efficiency. We formalize the Optimal Worker and Auditor Assignment (OWAA) problem, which has a target that is to find the optimal numbers of workers and auditors so that the total processing time can minimized. We implement a prototype of Accountable MapReduce in the Hadoop platform. Our evaluation results show that our scheme can be practically and efficiently utilized in realistic cloud systems.

Acknowledgment

This work was supported in part by the US National Science Foundation (NSF) under grants CNS-0737325, CNS-0716211, CCF-0829827, and CNS-1059265.

References

[1] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, in: OSDI'04: Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation, USENIX Association, Berkeley, CA, USA, 2004.
[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, et al., Above the Clouds: A Berkeley View of Cloud Computing, EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.
[3] Z. Xiao, Y. Xiao, Security and privacy in cloud computing, IEEE Communications Surveys & Tutorials 15 (2) (2013) 843–859. Second Quarter.
[4] M. Barua, X. Liang, R. Lu, X. Shen, ESPAC: enabling security and patient-centric access control for eHealth in cloud computing, International Journal of Security and Networks 6 (2/3) (2011) 67–76.
[5] D. Sun, G. Chang, C. Miao, X. Wang, Modelling and evaluating a high serviceability fault tolerance strategy in cloud computing environments, International Journal of Security and Networks 7 (4) (2012) 196–210.
[6] D. Gottfrid, Self-service, prorated super computing fun! The New York Times. http://open.blogs.nytimes.com/2007/11/01/self-service-prorated-super-computing-fun/?scp=1&sq=self%20service%20prorated&st=cse.
[7] http://hadoop.apache.org/.
[8] Department of Defense, Trusted computer system evaluation criteria, Technical Report 5200.28-STD, Department of Defense, 1985.
[9] A.R. Yumerefendi, J.S. Chase, The role of accountability in dependable distributed systems, in: Proc. of HotDep, 2005.
[10] A.R. Yumerefendi, J.S. Chase, Trust but verify: accountability for network services, in: Proc. of 11th Workshop on ACM SIGOPS 2004, p. 37.
[11] Y. Xiao, Flow-net methodology for accountability in wireless networks, IEEE Network 23 (5) (2009) 30–37.
[12] D. Takahashi, Y. Xiao, Retrieving knowledge from auditing log files for computer and network forensics and accountability, (Wiley Journal) Security and Communication Networks 1 (2) (2008) 147–160.
[13] Y. Xiao, Accountability for wireless LANs, ad hoc networks, and wireless mesh networks, IEEE Communications Magazine 46 (4) (2008) 116–126.
[14] J. Liu, Y. Xiao, Temporal accountability and anonymity in medical sensor networks, ACM/Springer Mobile Networks and Applications (MONET) 16 (6) (2011) 695–712. Special Issue: Wireless and Personal Communications.
[15] Y. Xiao, K. Meng, D. Takahashi, Accountability using flow-net: design, implementation, and performance evaluation, (Wiley Journal of) Security and Communication Networks 5 (1) (2012) 29–49. Special Issue on Security and Privacy in Emerging Information Technologies.
[16] Z. Xiao, Y. Xiao, PeerReview re-evaluation for accountability in distributed systems or networks, International Journal of Security and Networks 7 (1) (2012) 40–58.
[17] Z. Xiao, Y. Xiao, P-Accountable networked systems, in: INFOCOM IEEE Conference on Computer Communications Workshops, 2010, pp. 1–5.
[18] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM 51 (2008) 107–113.
[19] C.T. Chu, S.K. Kim, Y.A. Lin, Y.Y. Yu, G. Bradski, A.Y. Ng, K. Olukotun, Map-reduce for machine learning on multicore, in: Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference, 2007, pp. 281–288.
[20] B. He, W. Fang, Q. Luo, N.K. Govindaraju, T. Wang, Mars: a MapReduce framework on graphics processors, in: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, 2008, pp. 260–269.
[21] S. Papadimitriou, J. Sun, Disco: distributed co-clustering with Map-Reduce: a case study towards petabyte-scale end-to-end mining, in: Eighth IEEE International Conference on Data Mining, 2008, ICDM'08, 2008, pp. 512–521.
[22] J. Ekanayake, S. Pallickara, G. Fox, MapReduce for data intensive scientific analyses, in: IEEE Fourth International Conference on eScience, 2008, eScience'08, 2008, pp. 277–284.
[23] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, E. Witchel, Airavat: security and privacy for MapReduce, in: Proc. USENIX Symposium on Networked Systems Design and Implementation, 2010, pp. 297–312.
[24] J. Schlesinger, Cloud security in Map/Reduce, 2009. http://www.defcon.org/images/defcon-17/dc-17-presentations/defcon-17-jason_schlesinger-cloud_security.pdf.
[25] W. Wei, J. Du, T. Yu, X. Gu, SecureMR: a service integrity assurance framework for MapReduce, in: Proceedings of the 2009 Annual Computer Security Applications Conference, 2009, pp. 73–82.
[26] A. Haeberlen, A case for the accountable cloud, ACM SIGOPS Operating Systems Review 44 (2010) 52–57.
[27] C. Wang, Y. Zhou, A collaborative monitoring mechanism for making a multitenant platform accountable, in: Hotcloud 2010.
[28] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Transactions on Programming Languages and Systems 4 (3) (1982) 382–401.
[29] A. Haeberlen, P. Kouznetsov, P. Druschel, PeerReview: practical accountability for distributed systems, in: Proc. of ACM SIGOPS 2007.
[30] A. Juels, B.S. Kaliski, PORs: proofs of retrievability for large files, in: ACM CCS, 2007, pp. 584–597.

Zhifeng Xiao is an Assistant Professor in the Department of Computer Science at the Penn State Erie, the Behrend College. He obtained the Ph.D. degree in Computer Science at the University of Alabama in 2013. He received the bachelor degree in Computer Science at Shandong University, China, in 2008. His research interests are in design and analysis of secure distributed and Internet systems.

Yang Xiao Yang Xiao worked in industry as a MAC (Medium Access Control) architect involving the IEEE 802.11 standard enhancement work before he joined Academia. Dr. Xiao currently is a professor of Department of Computer Science at The University of Alabama. He was a voting member of IEEE 802.11 Working Group from 2001 to 2004. He is an IEEE Senior Member. He serves as a panelist for the US National Science Foundation (NSF), Canada Foundation for Innovation (CFI)'s Telecommunications expert committee, and the American Institute of Biological Sciences (AIBS), as well as a referee/reviewer for many national and international funding agencies. His research areas are security and communications/networks. He has published more than 200 refereed journal papers (including 50 IEEE/ACM transactions papers) and over 200 refereed conference papers and book chapters related to these research areas. Dr. Xiao's research has been supported by the US National Science Foundation (NSF), U.S. Army Research, The Global Environment for Network Innovations (GENI), Fleet Industrial Supply Center-San Diego (FISCSD), FIATECH, and The University of Alabama's Research Grants Committee. He currently serves as Editor-in-Chief for International Journal of Security and Networks (IJSN) and International Journal of Sensor Networks (IJSNet) (SCI-index). He was the founding Editor-in-Chief for International Journal of Telemedicine and Applications (IJTA) (2007-2009). Dr. Xiao had directed 8 doctoral dissertations in the past and is currently supervising 7 Ph.D. students/candidates in computer security and networking areas. Dr. Xiao also supervised 19 M.S. graduates in the past.