Security issues relating to inadequate authentication in MapReduce applications

DOI:

[10.1109/HPCSim.2013.6641427](https://doi.org/10.1109/HPCSim.2013.6641427)

[Link to publication record in Manchester Research Explorer](#)

Citation for published version (APA):

Dyer, J., Smari, W. W. (Ed.), Zeljkovic, V. (Ed.), & Zhang, N. (2013). Security issues relating to inadequate authentication in MapReduce applications. In W. W. Smari, & V. Zeljkovic (Eds.), *High Performance Computing and Simulation (HPCS), 2013 International Conference on* (pp. 281-288). IEEE.
<https://doi.org/10.1109/HPCSim.2013.6641427>

Published in:

High Performance Computing and Simulation (HPCS), 2013 International Conference on

Citing this paper

Please note that where the full-text provided on Manchester Research Explorer is the Author Accepted Manuscript or Proof version this may differ from the final Published version. If citing, it is advised that you check and use the publisher's definitive version.

General rights

Copyright and moral rights for the publications made accessible in the Research Explorer are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

Takedown policy

If you believe that this document breaches copyright please refer to the University of Manchester's Takedown Procedures [<http://man.ac.uk/04Y6Bo>] or contact uml.scholarlycommunications@manchester.ac.uk providing relevant details, so we can investigate your claim.

Security Issues Relating to Inadequate Authentication in MapReduce Applications

James Dyer, Ning Zhang
School of Computer Science
The University of Manchester
Manchester, M13 9PL, UK
{jdyer, nzhang}@cs.man.ac.uk

Abstract—In this paper, we discuss security issues relating to inadequate authentication in MapReduce applications. We examine MapReduce applications deployed in the cloud, as this environment significantly increases security risks to the applications. We outline a generic model of MapReduce computation and then perform a detailed threat analysis of this model. Then, based on this threat analysis, we produce a set of security requirements for the design of an authentication solution for MapReduce applications. We then review related work against these requirements, and conclude that most of the work surveyed does not address two of our requirements, which we believe to be of significant importance when deploying MapReduce in the cloud.

Keywords—Cloud computing, scalable computing, security, authentication, access control.

I. INTRODUCTION

Processing large volumes of data has become increasingly important to businesses and government. One of the popular tools used in processing such data is MapReduce [3].

Cloud computing has made MapReduce more accessible. Infrastructure-as-a-Service (IaaS) provided by cloud vendors allows for easy creation of MapReduce clusters by provisioning large numbers of compute instances. Furthermore, MapReduce Platform-as-a-Service (PaaS) is provided by cloud vendors such as Amazon [1] and Microsoft [5].

For many MapReduce applications, performance has been paramount and security has taken a back seat. In these applications, security is largely applied at the network or application gateway only. New MapReduce applications are developed to be deployed in untrusted environments, such as public clouds, or to be deployed across data centres or to process sensitive data. Network isolation alone is not sufficient to address the threats seen in these environments [8].

This paper presents a threat analysis and specifies a desirable set of requirements for an authentication system for MapReduce applications. This threat analysis concentrates on threats to a MapReduce application deployed in the cloud which result from inadequate authentication. It is performed using a generic model we have devised. In detail, section II outlines the generic model. Section III presents the threat analysis. Section IV specifies the requirements. In section V

we review related work on MapReduce authentication against the specified requirements. Finally, section VI concludes the paper and outlines our future work.

II. A GENERIC MODEL OF A MAPREDUCE COMPUTATION

A. Physical Architecture

The MapReduce application is physically deployed as virtual machines (VMs) in a public cloud (see Figure 1). One VM is a master node and the other VMs are worker nodes. These VMs are hosted on physical virtualisation hosts within the cloud infrastructure. As shown in the figure, a VM may share tenancy on a virtualisation host with other VMs. It may be part of the MapReduce application or related to another customer's application. The VMs for each customer application placed on the same virtualisation host are isolated from each other using the dedicated virtual switch for each application. A VM can communicate with another VM on the same virtualisation host via a virtual switch. The switch accesses the physical network via the virtualisation host's kernel, known as a *hypervisor*. The hypervisor puts data frames passed down from the switch on to its network interface.

A client application is responsible for configuring, starting and monitoring the MapReduce job. In an IaaS MapReduce implementation, the client will manage the MapReduce architecture directly by accessing the master node VM. In a PaaS implementation, the client will connect to a management interface which uses cloud middleware to manage the MapReduce architecture. The interactions between the client and the cloud are performed via the Internet.

B. MapReduce Architecture

In the MapReduce architecture, we have several cluster nodes: a single master node and several worker nodes (see Figure 2). The master node executes a single master daemon. This daemon is responsible for coordinating the execution of the application on the worker nodes. Similarly, each worker node executes a single worker daemon. It is responsible for launching and managing mapper and reducer child daemons on the node. A mapper executes a single map task and a reducer executes a single reduce task. A daemon process is denoted as a *component* of the MapReduce application.

Figure 1. Physical architecture of a cloud MapReduce application

Figure 2. The architecture of a cloud MapReduce application

We designate the collection of components that form our MapReduce application to be a *framework*.

In addition to the MapReduce framework components, we have nodes that host an authentication server for entity authentication, vital networking services (not shown on Figure 2), such as DNS or DHCP, and a distributed file system (DFS) which is hosted on a separate DFS cluster. The DFS is a block oriented file system where each file is stored as a number of fixed size data blocks distributed across the DFS cluster. The DFS cluster has one catalogue server and multiple storage servers. The catalogue server is responsible for maintaining the file system directory tree along with the locations of the blocks associated with a file. Storage servers are responsible for hosting the individual file blocks.

C. MapReduce Execution Flow

The components involved in a MapReduce application are shown in Figure 3. The input files are divided into input splits, and there is at least one input split per mapper process. An input split is assigned to a mapper by the master. The mapper processes the input split and writes the output locally. The local output is divided into a number of data sets called partitions. The number of data partitions produced by each mapper is the same as the number of reducers specified by application configuration. Each reducer is assigned a partition index to process. Once all the mappers have completed their computations, the reducer will read the partition with its assigned index from each mapper. The partition data is combined and sorted, and then processed. The reducer's output is written to a file in a folder on the DFS. So, the output folder will have one file for each reducer. In detail, the application

Figure 4. MapReduce execution flow

is executed as follows. First, the framework components are launched. These include a master daemon launched on the master node and a worker daemon on each of the worker nodes. Before a worker daemon starts launching mappers and reducers it should first obtain admission to the MapReduce cluster (Figure 4(3)) and authenticates itself to the master.

The authentication process (Figure 5) has been deliberately modeled on the authentication solution for Hadoop outlined in

Figure 3. Simplified MapReduce execution flow

Figure 5. The MapReduce component authentication process

O'Malley et al. [6]. This authentication solution uses a combination of Kerberos and proprietary credentials. The modeled authentication process is identical for all the components in the framework so we will only describe it once. For simplicity, and without loss of generality, we assume that the authentication server is responsible for issuing both *master* and *service tickets*. A component makes an authentication request to the authentication server using its credential (Figure 5(1)). If the authentication outcome is positive it will obtain a *master ticket*. It then uses the master ticket to request service tickets to access remote resources. The service ticket request will contain the requested resource address in addition to the master ticket. Once the service ticket has been obtained, it can be used to access the specified resource (Figure 5(3)).

When a worker daemon has obtained a service ticket for the

master, it requests to be admitted to the MapReduce cluster (Figure 4(3)). The master verifies the request and sends a response indicating whether the worker has been admitted or not. If a worker node is not admitted, it will play no part in the computation.

Figure 6. DFS read and write processes

As soon as all the worker nodes have been admitted, the client writes the input files and job program files into the DFS (Figure 4(1)). To do so, the client program submits a write request to the DFS catalogue server (Figure 6(1)). The DFS catalogue server verifies the request. If the verification outcome is positive it returns a response which includes the address of the DFS storage server that hosts the new block of data, the file system identifier of the new block, and a *block ticket*. This block ticket will authenticate the client to the storage server. The client can then request to write the data to the block on the specified DFS storage server

(Figure 6(2)). The request includes the block identifier and the block ticket. Upon verification, the storage server sends a response to indicate that the write request is accepted or denied. If the write request is accepted the client streams the data to the storage server which is written to the block (Figure 6(3)). When the write process is completed, the DFS storage server sends an acknowledgement to the client indicating if the operation is successful. If there is more data to write than a single block, the write process repeats.

Once all the input and program files have been written into the DFS, the client calculates the input splits and submits the job details to the master (Figure 4(2)). The request contains the names of the input and program files, the calculated input splits and a job configuration. The master verifies that the client is authorised to submit a new job and that the input and program files exist, before initialising the job.

During the job execution, each worker daemon periodically sends a heartbeat message to the master daemon (Figure 4(3)) to inform the master that the node is still operational. The heartbeat message may also be used as a data channel between the worker node and the master. For each such message received, the master sends a response to indicate how many mappers and reducers to launch on the node. If mappers and reducers are currently executing on the node, then the master's response may also indicate if any of the mappers or reducers should be killed.

Before launching mappers and/or reducers on a node, the worker daemon first needs to fetch the job program files from the DFS and cache them locally (Figure 4(4)). Once this is done, the worker daemon launches the mapper and/or reducer. The mapper then requests work from the master (Figure 4(5)). The master schedules the mapper and generates a response which includes the input split it has assigned to the mapper and the location of the input file to read and an *authorisation ticket* for the DFS catalogue server. The authorisation ticket is used in lieu of a service ticket when the mapper requests the input file.

The mapper fetches its assigned input split from the DFS (Figure 4(6)) and parses it into key-value pairs. The mapper invokes the user-defined map function on each key-value pair. Upon execution completion, the mapper separates the output into several sets and writes each set to a separate location on local storage (Figure 4(7)). The mapper now notifies the master of the outcome of the computation (Figure 4(8)).

The reducers periodically poll the master for work. This is done by a work request message (Figure 4(9)). When all mappers have notified the master that they have successfully completed their computations, the master chooses a partition for the reducer to process. The master responds to the polled request by sending the partition index, an authorisation ticket for the DFS catalogue server and an authorisation ticket for each mapper.

To read the key-value pairs from the specified partition on each mapper (Figure 4(10)), the reducer makes a request to the mapper. This request includes the partition index and the authorisation ticket supplied by the master. The mapper sends the requested key-value pairs to the reducer. The reducer sorts the key-value pairs received from all mappers by key. Where pairs have the same key, they are combined into a single pair. Each reducer writes its output to a file on the DFS (Figure 4(11)). After the output has been successfully written to the DFS file, the reducer now notifies the master that it has successfully completed the computation or, if the write process failed, that it has failed (Figure 4(12)).

Once all reducers have completed, the master sends a notification of job completion to the client (Figure 4(13)).

III. THREAT ANALYSIS

In this section we analyse and identify security threats or potential attacks on the MapReduce application outlined above. We focus on threats that result in unauthorised access to data or other resources and threats that delay or alter the execution of the MapReduce application.

A. Attacks on resources

The attacks in this category are intended to gain unauthorised access to cluster resources, such as files in the DFS, intermediate data on worker nodes or computational resources in the cluster.

1) *Impersonation Attacks*: Entity impersonation attacks occur when an attacker tries to impersonate an authorised identity. They may be leveraged by guessing, forging, replaying or using stolen credentials. Entity impersonation may also be conducted by replaying the tickets issued by the authentication service, master daemon, or DFS catalogue server. A service ticket may be captured and replayed to gain unauthorised access to a resource. A replayed service ticket for a worker daemon may be used to admit a malicious daemon to the MapReduce cluster. Similarly, a replayed service ticket for the DFS catalogue server would allow access to all the files associated with the original ticket holder. A request to read and write blocks on the DFS (Figure 6(1)) is authorised using a block ticket issued by the catalogue server. This ticket may be replayed to gain unauthorised access the same data block.

Impersonation attacks may also be performed by impersonating requests and responses. A request or response message may be fabricated, altered or replayed to gain unauthorised access. For example, such a message may be forged in one of two ways: (1) by including the identifier of an authorised node, job or task in the message; (2) by forging the source address of a trusted node.

An authorisation token for an input file on the DFS or partitioned data on a mapper is sent in response to a successful map or reduce work allocation request. A forged request may allow an attacker to obtain this token. Furthermore, a fake DFS

write response indicating that data should be streamed could be used to deceive the client into attempting to write the data into DFS. This would allow an attacker to intercept it as it is transmitted between the DFS client and the storage server.

In addition to impersonation, message modification may be used to gain unauthorised access to data or files in the DFS. Read and write requests sent to the DFS catalogue server (Figure 6(1)) may be altered to include an attacker's source address. The response from the server can then be sent to the attacker's host rather than the original requester. In this way, an attacker may receive block tickets which would allow access to the files in DFS. Indeed, the same outcome may be achieved by similarly modifying a request to a DFS storage server (Figure 6(1)).

A request replay may also be used to obtain unauthorised access to resources. For example, a request to the authentication service (Figure 5(1,2)) might be replayed to obtain the response. This could be used to capture a ticket contained in the response for later use or to invalidate a previously issued ticket. Similarly, a request to the DFS catalogue server (Figure 6(1)) may be replayed to gain a block ticket. A read request to a storage server may be replayed to gain unauthorised access to a data block. A write request might be replayed to overwrite the block with false data. The work requests from mappers and reducers (Figure 4(5,9)) may be replayed with the purpose of obtaining authorisation tickets for DFS or intermediate data.

2) *Privilege escalation attacks*: A MapReduce cluster may run multiple jobs concurrently. Also, a cloud MapReduce application may share the same physical hardware with other virtual applications. Where multiple MapReduce jobs are able to run in a cluster, it is possible for an attacker to create a job with the purpose of disrupting or controlling the tasks of another job. A MapReduce job allows for user-defined code to be executed as part of the mappers or reducers on worker nodes. If an attacker is able to launch a job of his own on the cluster, he may be able to execute code written to exploit possible vulnerabilities in the worker node Java Virtual Machine (JVM) or operating system. Such exploitation could lead to the compromise of the worker node. Once the node is compromised, the attacker may access the input data accessible by the mappers running on the node as well as the intermediate data generated by those mappers. Similarly, gaining control of the reducers on a worker node may allow the attacker to access intermediate data partitions on other worker nodes.

In the case of a cloud MapReduce application, the customer software running on the VMs (see Figure 1) is only isolated from other customer applications by the virtualisation host's hypervisor. If the hypervisor is compromised then all the VMs running on the host can be controlled by an attacker. A hypervisor presents an additional attack target for the virtualisation host. The hypervisor management interface also exposes it to network attacks [9]. Similarly, the virtual switches and simulated hardware drivers, which interact with the hypervisor, expose the host to side channel attacks [12]. Furthermore, two

popular virtualisation products, Citrix's Xen and Microsoft's Hyper-V, use a highly privileged VM to manage physical hardware access and control the hypervisor. Compromising this VM is equivalent to compromising the hypervisor itself. Xen is used in many cloud implementations, such as Amazon's Elastic Compute Cloud (EC2) [2], and Hyper-V is used as the basis for Microsoft's Azure platform.

In addition to the threats posed by the hypervisor, an IaaS provider allows customers to configure VMs as they wish, so an unpatched operating system may be installed on a VM. Also, the cloud provider usually makes a repository of pre-configured VMs available for customers to use. Customers may add VMs into the repository. An attacker may upload a trojanised VM to the repository which is then be used as part of a customer application [10]. Finally, cloud VMs have public IP addresses by default which make it possible for an attacker to "sniff" on ports to discover and exploit vulnerabilities in the VM guest operating system and other software hosted.

B. Attacks on job executions

The attacks in this category are intended to interrupt, stop, delay or alter a MapReduce computation.

1) *Denial of Service (DoS) attacks*: We have two critical servers that are vulnerable to DoS attacks, the master daemon and the authentication service. The master daemon is responsible for scheduling and coordinating the execution of a MapReduce application on the worker nodes. It allocates work to mappers and reducers and also issues authorisation tickets to allow them to read and write data. A DoS attack on the master will halt the computation. The authentication service, another critical component, issues service tickets to workers to allow them to access the DFS. Without accessing the user defined program files located on the DFS, the mappers and reducers would not be able to process the data.

Furthermore, if a worker daemon's heartbeat message is dropped or inordinately delayed then the master will assume that the daemon or node has failed. It will not schedule further computational tasks on the worker. As a result, the available computational resources in the cluster is reduced.

Impersonated requests and responses may also be used to interrupt job executions in the cluster. For example, an attacker may use an authorised user's ID to submit a job request in order to specify arbitrary program code. Similarly, he may forge a heartbeat response (Figure 4(3)) to direct the worker daemon to cache malicious program code. Such attacks could disrupt the computation or gain control of the host. This may also be leveraged by replaying a valid service ticket to the master daemon as part of the request.

Alternatively, a job submission request (Figure 4(2)) may be modified in an attempt to change the job configuration. For example, an attacker may reduce the number of mappers and reducers in order to increase the execution time of the job giving the attacker more time to perform brute force attacks on

cluster nodes. Similarly, an impersonated heartbeat response may be used to notify a worker daemon to launch more or fewer mappers and reducers. Such an attack could be used to deprive the cluster of resources or to delay the execution of the attacked job.

An impersonated response from the master to a work request (Figure 4(5,9)) may allow an attacker to disrupt the job computation by specifying non-existent input files for a mapper to process or an invalid partition index for a reducer to process. Similarly, a crafted response from a mapper to a reducer (Figure 4(10)) may be used to misinform the reducer that accessing a partition has been denied. Without access to all the partition records from the mappers, the reducer is not able to successfully complete its computation.

Faked responses from the DFS catalogue server or storage server could be used to misinform the requester that file access has been denied or to direct the requester to a DFS storage location of the attacker's choice. This might allow an attacker to halt the job execution by denying it access to the user-defined program files or input files. In the former case, a worker daemon will not be able to launch mappers or reducer. In the latter case, a mapper will not be able to execute its task if it has no access to its input split. A forged response may also allow an attacker to specify files to be read by the requester. In the case of the worker daemon fetching the user-defined program files from the DFS (Figure 4(3)), such an attack could direct the worker daemon to fetch malicious program files instead.

On task completion, a mapper or reducer will send a notification message to the master (Figure 4(8,12)). A crafted notification message may be used to misinform the master that a task had failed while it was actually still being executed. The master may act on this notification and schedule the task to be re-executed, thus prolonging the execution time of the task. Alternatively, a forged notification may indicate a failed task had successfully been completed. If the task were a map task that had failed, then this attack might disrupt the computation as the mapper may have written no intermediate data, or a partial result set before failing.

2) *Attacks on the computation:* The result of a MapReduce computation is dependent on intermediate results being correctly computed by mappers and final results being correctly computed by reducers. If one mapper is compromised and produces false results this may result in an incorrect final result. An attacker wishing to disrupt the MapReduce computation need only compromise one mapper or reducer in order to invalidate the entire computation.

3) *Privilege escalation attacks:* The privilege escalation attacks described in section III-A2 may be also used to disrupt or attack legitimate job executions in the cluster. For example, an attacker may use malware to exhaust cluster resources assigned to other legitimate jobs, to delay or terminate the execution of authorised jobs. A map or reduce task of a rogue job may attempt to obtain more than its fair share of the CPU

or memory resources on a worker node. It may also create numerous files in an attempt to overload the file system index on a worker node, thus preventing the creation of new files.

IV. REQUIREMENT SPECIFICATION

This section specifies a set of requirements based on the threat analysis in section III. These will serve the design of an authentication service for MapReduce applications.

Entity Authentication

The master, worker, mapper, and reducer require access to cluster resources, such as files in the DFS. In order to determine whether these components have been granted authorisation for a resource, these components need to be correctly identified or classified. Therefore, we require that the worker, mapper and reducer be subject to mutual entity authentication with the master and the DFS. In addition, the reducer should perform mutual entity authentication with the mapper. Furthermore, whether the job client is authorised to submit a job or write files in to the DFS must also be verified. It should also be subject to mutual entity authentication with the master daemon and the DFS.

Message Authenticity

Many of the impersonation and DoS attacks outlined in section III leverage message modification, replay and forgery. To make the success of such attacks unlikely, the authentication solution should verify that messages between the MapReduce components originated with the stated source and that each message has not been altered after creation. Furthermore, the message authenticity process should detect replayed messages.

Server Availability

The authentication service should be continuously available to authenticate the MapReduce components and clients. It should also be able to resist DoS attacks. This means that the authentication service should be provided in a computationally cost-efficient manner. In addition, message authenticity verifications should take as little time as possible to reduce any risk of DoS attacks on daemons. Measures should be in place to detect any delayed or dropped messages to prevent unnecessary delays or live-lock; this is particularly important with regard to the heartbeat messages generated by daemons.

Performance

As MapReduce applications typically process terabytes or petabytes of data, timely completion of the computation is an important consideration. For short running applications, performance degradation can have a severe effect of the completion time. Indeed, one of the requirements of the Hadoop security design [6] was that performance only be degraded by 3% when running the GridMix benchmarking application. Authentication should not severely delay the completion of the computation.

Scalability

Scalability is the measure of how easy it is for an application to deal with increases in the number of users or volume of data processed. MapReduce scales by simply adding nodes to a cluster. Any authentication system introduced into MapReduce should scale similarly.

Dynamism

MapReduce is a dynamic distributed application. Where a cluster hosts multiple jobs there will be jobs starting and finishing at different times within the cluster. If a MapReduce cluster is hosted on VMs, such as in the cloud, then a cluster may be rapidly scaled up or down by adding new VMs, or removing existing VMs from the cluster. Authentication for MapReduce should be able to automatically adapt to these dynamic changes in the membership of a cluster.

Protection Against Compromised Nodes

Section III-A2 indicates that privilege escalation attacks may lead to a MapReduce cluster containing compromised nodes. This poses a problem, as compromised nodes can be very difficult to detect and isolate. MapReduce authentication should not assume that cluster nodes are trustworthy.

Result Authenticity

A subset of nodes in a cluster may not be trustworthy. They may attempt, either alone or in collusion, to alter intermediate data. This would lead to an incorrect computation result. There should be measures to verify the integrity of MapReduce computational results.

V. RELATED WORK

In this section, we examine related work in authentication for MapReduce. First, we discuss the current state of authentication in MapReduce frameworks. We then look at recent research in MapReduce authentication, particularly with regard to the set of requirements specified in section IV. The issues that have been addressed by each approach is summarised in table I.

The current state of authentication in MapReduce application frameworks is unclear. Many MapReduce application frameworks do not provide any architecture to support authentication or authorisation of MapReduce components. The focus of these projects is on the performance of the MapReduce application rather than its security. It is assumed that authentication is provided by means which are independent of the frameworks, such as host operating system-based authentication systems on worker nodes.

Apache Hadoop is a notable exception in that a security architecture has been provided for the Apache Hadoop MapReduce framework [6]. Prior to Hadoop 1.0.0, no security

measures were provided to protect the application from unauthorised access. This is still the default mode of operation. O'Malley et al. propose a solution which uses Kerberos authentication for requests between key participants, such as servers and users, but uses DIGEST-MD5 to authenticate access to resources by map and reduce tasks. Access to Hadoop services is performed using Hadoop's remote procedure calls (RPC). The security design adds Simple Authentication and Security Layer (SASL) support to the library. Two authentication mechanisms are provided using SASL: Kerberos (via Generic Security Service Application Programming Interface (GSSAPI)) and DIGEST-MD5. For Kerberos authentication, the client obtains a service ticket for the given service and mutual authentication is provided using SASL/GSSAPI. If the client and server share a secret, they can use SASL/DIGEST-MD5 to authenticate to each other. SASL/DIGEST-MD5 is the basis for service to service authentication using proprietary credentials. In addition, authentication by Kerberos over HTTP is provided by a Java SPNEGO library.

In the research domain there are approaches [11, 13–15] on verifying the authenticity of the result of a MapReduce computation. Wei et al.'s [11] propose a SecureMR framework which implements a service integrity assurance mechanism for MapReduce. They present an architecture and protocols to securely schedule and assign work to worker nodes. Map workers authenticate the intermediate data partitions they have generate using digital signatures. The master is notified of these partitions and verifies the digital signatures using the map worker's public key. The map worker's data is only designated as committed when the signature is verified. Similarly, reduce workers use a verification protocol to ensure that the data they have been allocated by the master is authentic. Protocol messages are also authenticated using digital signatures. Their solution demonstrates a performance penalty of 5% to 12%. Also, their performance tests are performed using Apache's Hadoop on a very small cluster of 14 hosts with approximately 100 concurrent programs running. They do not discuss how the performance might scale to a more realistic application size of thousands of concurrent programs.

Zhou et al. [15] approach this problem via an authenticated MapReduce application implemented on their DS2 platform. Each map worker provides an authenticator for the intermediate data it emits. A reduce worker will not accept intermediate data unless its authenticator can be verified. The authentication is provided by HMAC-SHA1 message authentication codes and RSA-1024 digital signatures. On a cluster of 16 quad-core machines, their results exhibit a job completion overhead of 17% and 78% using HMAC-SHA1 and RSA-1024, respectively. Furthermore, their work does not address performance in large scale MapReduce clusters.

In the approach proposed by Xiao and Xiao [13] workers are divided into two classes: normal and speculative. Multiple speculative workers are assigned the same task. Speculative workers generate an MD5 hash for the intermediate data they

	O'Malley et al.	Wei et al.	Zhou et al.	J. Xiao & Z. Xiao	Z. Xiao & Y. Xiao	Lin et al.
Entity Authentication	✓	✓	✓	✗	✗	✓
Message Authenticity	✗(default)	✓	✓	✗	✗	✓
Server Availability	✗	✓	✗	✗	✗	✗
Scalability	✗	?	?	?	?	?
Performance	✓	✓	✓(HMAC) ✗(RSA signing)	✗	?	✗
Dynamism	✗	✗	✗	✗	✗	✗
Protection Against Compromised Nodes	✗	✗	✗	✓	✓	✗
Result Authenticity	✗	✓	✓	✓	✓	✗

TABLE I
ISSUES ADDRESSED IN RELATED WORK

generate. In their paper, speculative workers are forced to enter a committing phase before completion. The master node waits for all running workers assigned to a task to enter the committing phase. It compares the MD5 hashes received from all the speculative workers for that task and chooses an MD5 hash for the correct result by majority decision. It then randomly chooses a speculative worker whose MD5 hash matches the one decided upon and kills all other workers. Although this is a lightweight approach to the problem it requires spawning multiple workers for the same task. This approach requires very large clusters where computation is replicated to guarantee output correctness. The performance analysis for speculating 10% and 30% of tasks shows an overhead of 41% and 72% respectively.

Xiao and Xiao [14] address the same problem using an accountability test for workers. Machines in a trusted domain form an auditor group which is responsible for checking the correctness of results produced by workers. Each map worker generates a tamper-proof log of task operations which may be replayed by an auditor to check correctness. Rather than duplicating all the work from untrusted workers by replaying logs in entirety, Xiao and Xiao propose a probabilistic measure of accountability where a sample of records are reprocessed. They do not discuss the performance overhead required for performing these accountability tests.

Lin et al. [4] discuss the use of a Shamir secret sharing scheme [7] to allow a cluster to recover from the failure of a PKI server safely. Each map worker is assigned a piece of the PKI private key, a *shadow*, by the PKI server. Interestingly, the map workers use these shadows as a private key to sign their intermediate data. When a signature needs to be verified, the public key associated with a shadow is requested from the PKI server. In the event that the PKI server fails then the secret key can be reconstructed by a new master by assembling t shadows from map workers. Provided that revocation lists are published in an accessible location, the reconstruction of the server private key allows the system to return to its state before failure. Lin et al.'s scheme has a severe performance overhead. With 40 tasks, MapReduce job takes twice as long to complete using their mechanism compared to without security.

VI. CONCLUSION

This paper has presented a threat analysis, focusing on authentication, for MapReduce applications. Furthermore, we have proposed a set of requirements for the design of an authentication method for MapReduce. We have then discussed current approaches to MapReduce authentication with regard to those requirements. We note that while the surveyed works variously address entity authentication, message authenticity, message integrity, availability and result authenticity, most do not address the issues of scalability, dynamism or handling nodes that are not trustworthy. We believe that these latter two requirements are especially important for MapReduce applications deployed in the cloud.

We argue that crypto-based authentication methods alone, such as digital signing or message authentication codes, may not be able to effectively address the two requirements. An approach that relies on cryptographic methods alone is vulnerable to node compromise as the cryptographic keys that are used to authenticate may also be compromised. In such a situation, it is impossible to distinguish between a trustworthy and a compromised node. We require additional authentication evidence that can be used to determine whether the node is still trustworthy.

Our future work intends to investigate the use of such additional evidence in an authentication solution for MapReduce. Such an approach would make impersonation and modification attacks significantly harder.

REFERENCES

- [1] Amazon Web Services. (2012). Amazon Elastic MapReduce (Amazon EMR), [Online]. Available: <http://aws.amazon.com/elasticmapreduce/>.
- [2] S. Bleikertz. (Feb. 1, 2010). On Amazon EC2's underlying architecture, [Online]. Available: http://openfoo.org/blog/amazon_ec2_underlying_architecture.html.
- [3] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," in *Proc. OSDI 2004*, pp. 137–150.
- [4] H. Lin, C.-Y. Yang, and M.-Y. Hsieh, "Secure MapReduce data transmission mechanism in cloud computing using threshold secret sharing scheme," in *Proc. ACSAC 2009*.
- [5] Microsoft Corporation. (2012). Apache Hadoop-based services for Windows Azure, Microsoft, [Online]. Available: <http://www.hadooponazure.com/>.
- [6] O. O'Malley et al., "Hadoop security design," Yahoo! Tech. Rep., 2009.
- [7] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

- [8] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *JNCA*, vol. 34, no. 1, pp. 1–11, 2011.
- [9] A. Van Cleeff, W. Pieters, and R. Wieringa, "Security implications of virtualization: a literature study," in *Proc. CSE 2009*, pp. 353–358.
- [10] J. Wei *et al.*, "Managing security of virtual machine images in a cloud environment," in *Proc. CCSW 2009*, pp. 91–96.
- [11] W. Wei *et al.*, "SecureMR: a service integrity assurance framework for MapReduce," in *Proc. ACSAC 2009*, pp. 73–82.
- [12] R. Wojtczuk, "Subverting the Xen hypervisor," in *Proc. of Black Hat USA 2008*.
- [13] J. Xiao and Z. Xiao, "High-integrity MapReduce computation in cloud with speculative execution," in *Proc. ICTMF 2011*, pp. 397–404.
- [14] Z. Xiao and Y. Xiao., "Accountable MapReduce in cloud computing," in *Proc. SCNS 2011*, pp. 1082–1087.
- [15] W. Zhou *et al.*, "Towards a data-centric view of cloud security," in *Proc. CloudDB 2010*, pp. 25–32.