SEINA: A Stealthy and Effective Internal Attack in Hadoop Systems

Jiayin Wang*, Teng Wang*, Zhengyu Yang†, Ying Mao*, Ningfang Mi†, and Bo Sheng*
*Department of Computer Science, University of Massachusetts Boston, 100 Morrissey Boulevard, Boston, MA 02125
†Department of Electrical and Computer Engineering, Northeastern University, 360 Huntington Ave., Boston, MA 02115

Abstract—**Big data processing frameworks such as Hadoop [1] have been widely adopted in the past few years. However, the security issues in such large scale systems have not been well studied yet. While most of the prior work is focused on the data privacy and protection, this paper investigates a potential attack from a compromised internal node against the overall system performance. We explore the vulnerabilities of the existing Hadoop system, and develop an effective attack launched from the compromised node that can significantly degrade the data processing performance of the cluster without being detected and blacklisted for job execution. In addition, we present a mitigation scheme that protects a Hadoop system from such attack. We conduct experiments on real systems, and the results show that this attack greatly slows down the job executions in the native Hadoop system even with some basic defense mechanisms. Our mitigation scheme, while causing a minor overhead in normal circumstances, can keep the whole cluster running efficiently under this attack from the compromised internal node.**

I. Introduction

With the rise of cloud computing and big data analytics, more and more users are using big data processing frameworks to analyze various types of data. Users either own or rent a large-scale computing cluster to deploy a framework such as Hadoop [1], [2] and Spark [3], [4], and process these large volumes of data. Some companies also offer the data processing service, such as AWS [5].

Since big data analytics provides critical information to a wide set of applications, the security of the processing platform becomes a serious concern. While most of the prior work considers the data security and privacy, this paper considers a novel attack that aims to degrade the processing performance of the entire cluster. The problem is motived by two facts. First, when using a large scale cluster with tens or hundreds of machines, users may not be able to harden and protect each node perfectly. Security breach and node comprise are possible in practice. Second, once a node is compromised, the data loss is not the only damage. Job execution time is also crucial to such a cluster, especially when it is processing a large batch of jobs.

In this paper, we develop a stealthy and effective attack SEINA, that once launched can significantly prolong the job execution time. The main idea is to manipulate the task execution on the compromised node by pausing it and resuming it when needed. This attack is stealthy as it does not falsify any information or violate the data processing protocol. We also consider that the cluster may have deployed some defense mechanisms. Our attack will follow the security policy without triggering any alarm. On the other hand, this attack is very effective because it is designed based on the understanding of the data processing framework. Our main intuition is to explore the vulnerability of the current speculation scheme which starts a redundant task when the original task execution becomes slow. In this paper, we use Hadoop as the target platform, but the attack can be applied to any other frameworks with built-in speculation scheme. Our attack manipulates the execution timeline of the tasks assigned to the compromised node so that the whole cluster has to wait a long time for the finish of those tasks. With a careful management, this attack can yield an extremely long delay which is much more serious than just not participating the computation.

In addition, we present a mitigation scheme that helps protect the existing Hadoop system against this new attack. With a small overhead in regular circumstances, the mitigation scheme can greatly improve the performance under attack. We have implemented the attack and mitigation schemes in Hadoop YARN platform, and conducted extensive experiments in real system environments. The results show that this new attack is devastating in the native Hadoop system, and our mitigation scheme is an effective defense against it.

The rest of this paper is organized as follows: Section II reviews the related work and Section III introduces the background of Hadoop system and formulates our problem. The design of the new attack SEINA is presented in Section IV, and we discuss the mitigation scheme in Section V. Finally, we present the evaluation results in Section VI and conclude in Section VII.

II. Related Work

As one of the most popular open-source implementations of MapReduce [2], Hadoop [1] is wildly being adopted in Big Data processing. However, many comprehensive studies [6]–[8] also indicated that challenges and issues of security in cloud computing still remain in Hadoop, including the impacts of multi-tenancy, elasticity and third party control, upon the security requirements.

To solve these security related issues, a built-in secure mode module [9] is proposed in the native Hadoop YARN [10], which consists of three mechanisms: authentication (i.e., by using Kerberos or Delegation Tokens between core Hadoop services and clients), service level authorization (i.e., clients' permissions to access given Hadoop services) and data confidentiality (i.e., data encryption on Hadoop services/clients RPC and cross-data-node block data transferring).

However, Hadoop is not ready for secure hybrid-cloud computing in large scale systems [11], [12], since it is originally designed to work on a single cloud and not aware of the presence of the data with different security levels. To overcome this challenge, a secure data-intensive computing system called Sedic [13] is proposed, which schedules individual map tasks over a carefully planned data placement, in a way that the tasks within the private cloud only work on sensitive data and those on the public cloud only processes public data. [14] further proposed a novel DDoS detection method based on Hadoop that implements a HTTP GET flooding detection algorithm in Hadoop on the distributed computing platform. Later, a security solution SAPSC [15] is proposed based on the HDFS layer, with master/slave architecture under the environment of a Private Storage Cloud extended with a Partner/Public Cloud. Meanwhile, a fullscale, data-centric, reputation-based trust management system for Hadoop clouds called Hatman [16] is developed to leverage the clouds distributed computing power to strengthen its security, which is achieved by formulating both consistency-checking and trust management as secure cloud computations.

III. Background and System Models

In this section, we take Hadoop as an instance to introduce the background of big data processing frameworks and the vulnerabilities to potential attacks from a compromised node. At the end, we formulate the problem by presenting the system model, security model, and adversaries' objective.

A. Architecture of big data processing frameworks

Big data processing frameworks, such as Hadoop, are usually deployed in a large scale cluster consisting of a master node and many slave nodes. Users submit their jobs to the master node, and each job is composed of many tasks which can be executed in parallel in the cluster. Centralized controlling modules are deployed at the master node in charge of monitoring slave nodes' status and dispatching tasks to slave nodes for execution.

Particularly in Hadoop YARN, ResourceManager in the master node manages the slave nodes and the resource allocation. And NodeManager is deployed on each slave node. Each job first launches a special task, ApplicationMaster, on a slave node. ApplicationMaster splits the job to multiple tasks, generates resource requests that are sent to ResourceManager, and negotiates resources with the scheduler in ResourceManager. In turn, ResourceManager responds to a specific resource request by granting a resource container which is an allocation of a specific amount of resources (CPU, memory etc.) on a specific NodeManager. Then ApplicationMaster works with NodeManagers to execute tasks in the containers. One task can execute in one container and the container will terminate once the task is finished. In addition, every NodeManager needs to report the available resources of its slave node periodically to ResourceManager.

B. How the cluster handle slow executions

In this paper, our design of the attack targets on the vulnerability of speculation scheme which is a common solution for handling slow task execution in big data processing platforms. In this subsection, we briefly introduce how the speculation works in Hadoop which helps understand the attacking technique presented in the next section.

In the Hadoop system, if a task is detected as running slowly, a redundant task (speculative task) will be created in the system as an alternative. Hadoop runs a background speculator service that maintains a statistics table to record the average execution time of a task for each job. The data in this table is updated upon the completion of each task. The speculator service will periodically check this table and the running tasks to find the candidate tasks for speculative execution. Specifically, it enumerates all the running tasks and estimate the finish time of each task t_i based on the elapsed time and the current progress as shown in Eq(1), where T_{now} is the current timestamp, $T_{start}(i)$ is the starting time of task t_i, T_{mean} is average execution time with the task type of t_i and $PG(i)$ indicates t_i's current progress. In addition, the speculator service estimates the finish time of the alternative speculative execution in Eq(2). The execution time of the speculative task is estimated as the mean value of the historic execution times of the same type of tasks maintained in the statistic table.

$$EstEnd = \frac{T_{now} - T_{start}(i)}{PG(i)} + T_{start}(i) \quad (1)$$

$$EstRepEnd = T_{mean} + T_{now} \quad (2)$$

The speculator service will create a new task attempt of the selected running task if $EstEnd > EstRepEnd$.

In addition, ApplicationMaster configures a threshold that every task cannot execute longer than (T_{term}). Above all, a redundant task will be created for a running task when: $T_{now} > \min(\frac{PG(i)}{1-PG(i)}T_{mean}, T_{term}) + T_{start}(i)$.

C. Problem Formulation

In our problem setting, we consider that a set of jobs are submitted to a Hadoop cluster consists of one master node and m slave nodes. We consider a homogeneous cluster and assume that for the same type of tasks in the same job, the execution time of each task is the same, generally denoted as t. Thus, we assume that the attacker is aware of the execution time t of each task she or he plans to attack. In practice, this execution time data can be leant from historic execution of the same type of tasks because each node including the compromised node before detected will be assigned with multiple tasks from the same type of processing.

In addition, we assume that the Hadoop system can apply a basic defense scheme that detects *abnormal* task executions and records the number of their occurrences on each node. Once the abnormal events happened on a node exceed a threshold τ, the node is considered suspicious and blacklisted. In practice, there are various ways to define an abnormal task execution. In this paper, we consider the following representative definitions referring to different defense schemes:

- **Defense I:** A task execution is considered abnormal if the original task is not successfully finished, i.e., the Hadoop system has to start a speculative task and it is finished before the original task.
- **Defense II:** A task execution is considered abnormal if the task is finished, but the execution time is abnormally long regardless if the original task is finished or not.

In the design of SEINA, our objective is to develop an attacking strategy that maximizes the execution time of the jobs under the basic defense schemes.

Here is a reference table of the notations that will be used in our analysis. Their detailed definitions will be introduced in the next section.

m	number of nodes in the cluster
t_i/t	a particular task / execution time of a task
τ	abnormality threshold the Hadoop system sets
$\mathcal{D}(\theta,t)$	delay of a task(execution time t) if it's paused at progress θ
$D(t_i)$	delay of job execution when task t_i is attacked

TABLE I: Notations

IV. Design of SEINA

In this section, we present the details of the attack that stealthily and effectively prolongs the execution time of a batch of jobs in a processing cluster.

Once a node is compromised, the adversary gains the full control, and can launch a wide set of attacks. For example, the compromised node can refuse to execute the assigned tasks, report more resource availabilities to attract more tasks, and delete or revel the hosted data. However, these kinds of attacks can be easily detected and the Hadoop system will exclude it from the cluster by listing it in a blacklist. Then the worst damage is the loss of one computing node, and the data hosted there.

In this paper, we aim to develop a better attacking strategy that could cause much more serious performance degradation than just losing a node. Our main idea is to let the attacker intendedly delay the execution of victim tasks assigned to the compromised node which will eventually trigger speculative tasks to be launched on other nodes. In the rest of this section, we first present how to delay a task's execution, and analyze the additional overhead caused by the attack. Then we further analyze how a delayed task affects the execution time of a job and the makespan of multiple jobs. Finally, we present complete attacking algorithms that target on the three basic defenses mentioned in the previous section.

A. Delayed execution of a single task

In a Hadoop system, each task is executed in a resource container which refers to an individual process on the hosting node. Once a node is compromised, the attacker is able to view all the running processes, pause and resume their executions via external process management commands. Therefore, the attacker can pause the execution in any resource container, and resume it later if needed. Specifically, if the attacker plans to delay the execution of a particular task (given the task ID), she or he can check the Hadoop log messages and find the ID of the process (PID in Linux) that is executing the task. Then the attacker can execute the external commands to pause the specific process.

In the rest of this subsection, we analyze the impact of the delayed execution, i.e., if the system has to wait and start a speculative task to finish the task, how much more time it has to spend comparing to a regular execution. Assume that a task's regular execution time is t. The attacker starts to execute the task at time 0, and pauses the execution when the progress becomes θ. We use $\mathcal{D}(t,\theta)$ to indicate the delay caused by this setting. After the execution is paused, at a given time point $T \geq \theta \cdot t$, the expected finish time of the task is $\frac{T}{\theta}$, and the finish time of a speculative task is $T+t$. The cluster will decide to launch a speculative task when

$$\frac{T}{\theta} > T + t \quad \Rightarrow \quad T > \frac{t}{\frac{1}{\theta}-1}$$

Therefore, the longest delay we can achieve given t and θ is

$$\mathcal{D}(\theta,t) = \frac{t}{\frac{1}{\theta}-1} \tag{3}$$

The following Fig. 1 illustrates an example of delayed execution.

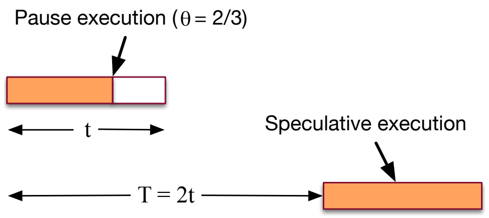

Fig. 1: Intuitions of the attack: In this example, the attacker pauses the execution of a task when it reaches the progress of $\frac{2}{3}$. The Hadoop system waits until time $T = 2 \cdot t$ to realize that it worthwhile to start a speculative task. At time T, the original task is expected to be finished at $\frac{T}{\theta} = \frac{3}{2} \cdot T = 3 \cdot t$, which is the same as starting a speculative task ($T + t = 3 \cdot t$).

Apparently, $\mathcal{D}(\theta,t)$ is an increasing function on θ. To cause the most serious damage, the attacker should pause the task execution when the progress is very close to 1. However, in practice, the progress report function in Hadoop is not fine-grained, and there are also some atomic processes that cannot be divided. Therefore, the reported progress value is discrete and bounded when inclining close to 1. We define $\alpha \in (0,1)$ as the largest progress an unfinished task can reach before the finish. Then the most delay the attacker can cause for a single task is

$$\mathcal{D}(\alpha,t) = \frac{t}{\frac{1}{\alpha}-1}.$$

For example, when $\alpha = 90\%$, the delay is $\mathcal{D}(\alpha,t) = 9 \cdot t$ which is quite significant.

B. Impact of the delayed tasks on job executions

In this subsection, we further analyze the impact of the attack on the execution of a job or multiple jobs. Specifically, we need to answer the following two questions: (1) when the adversary attacks a task causing $\mathcal{D}(\alpha, t)$ delay, what is the impact on the execution time of the job the victim task belongs to? and (2) if multiple jobs are running, what is the impact on the makespan performance.

In a Hadoop MapReduce job, there are two kinds of tasks, i.e., map tasks and reduce tasks. We assume that the attacker will mainly focus on the map tasks, because there are much fewer reduce tasks and thus the compromised node may not get the chance to host them. Based on this assumption, we further present the analysis of the overhead of a single job caused by a delayed map task. Assume that a single job consists of M map tasks running in a cluster with m nodes. Each node can host r containers to run the map tasks and each map task's execution time is t. The map phase of the job will be finished in the time of $\lceil \frac{M}{m \cdot r} \rceil \cdot t$. In particular, the task executions roughly follow waves. There are $\lfloor \frac{M}{m \cdot r} \rfloor$ waves for every node, and another final wave for a subset of nodes.

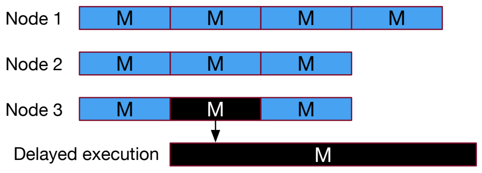

Fig. 2: An example of delay execution attack: the delayed task is the last finished map task in this case.

Assume the attacker delays the execution of a task in i-th wave, then in the following time period of $\mathcal{D}(\alpha, t)$, the compromised node can offer $r - 1$ containers to execute the map tasks. Thus, the finish time of the map phase under attack is

$$T_M = \begin{cases} T' & \lceil \frac{M'}{m \cdot r - 1} \rceil \cdot t < \mathcal{D}(\alpha, t) \\ T' + \lceil \frac{M' - \lceil \frac{\mathcal{D}(\alpha,t)}{t} \rceil \cdot (m \cdot r - 1)}{m \cdot r} \rceil \cdot t & \text{otherwise} \end{cases} \quad (4)$$

where $T' = i \cdot t + \mathcal{D}(\alpha, t)$ and $M' = M - i \cdot m \cdot r$.

Referring to Fig. 2, when the attack starts, there are M' tasks left to be assigned. Depending on the value of i and $\mathcal{D}(\alpha, t)$, there are two cases in the analysis. First, the delayed task is the last to finish in the map phase, which indicates the map phase will take $T_M = T'$ to finish. The condition for this case is that the delay $\mathcal{D}(\alpha, t)$ is long than the execution time of the remaining tasks with $m \cdot r - 1$ containers, i.e., $\lceil \frac{M'}{m \cdot r - 1} \rceil \cdot t$. In the second case, the attacked task is not the last one in the map phase. After it resumes to the normal state, the cluster will continue to execute other pending tasks $(M' - \lceil \frac{\mathcal{D}(\alpha,t)}{t} \rceil \cdot (m \cdot r - 1))$ with $m \cdot r$ containers.

For multiple job executions, the impact on the overall makespan cannot be expressed in a closed form. However, we can apply the similar analysis to estimate the delay. Due to the page limit, we omit the details in this paper. Overall, for a particular task t_i, we can derive the delay of the job execution caused by attacking it, denoted as $D(t_i)$.

C. Main algorithm

Finally, we present the complete attacking algorithms based the analysis of delayed execution attack. We develop different strategies against each of the two defenses we mentioned.

Against Defense I: The first defense only counts unfinished tasks and is not effective against an intelligent attack. With our delayed execution technique, the adversary can easily bypass abnormality detection and the threshold checking of τ, and attack every task assigned to the compromised node. Specifically, the attacker can first normal execute the task until the progress reaches α, then pauses the execution. The Hadoop system will eventually start a speculative task after $\mathcal{D}(\alpha, t)$ delay. With the knowledge of the task execution time and speculation policy, the attacker can estimate the time when the speculative task will be launched and finished. Then, the attacker can resume its own task's execution before the speculative task's progress reaches α, i.e., ensure that its task is finished before the speculative task. In this case, the attacker's behavior is not considered abnormal.

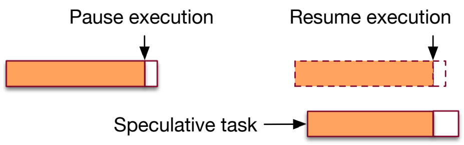

Fig. 3: Illustration of the attack against Defense I

Therefore, the best strategy against Defense I is to attack every task assigned to the compromised node by pausing each task at progress α, waiting for T_A, and then resuming the execution. The waiting T_A can be calculated as $T_A = \mathcal{D}(\alpha, t) + \alpha \cdot t$. In practice, the attacker can make a more conservative estimation to make sure its task will finish before the speculative task by setting

$$T_A = \beta \cdot (\mathcal{D}(\alpha, t) + \alpha \cdot t), \beta \in (0, 1)$$

In our evaluation, we set $\beta = 0.8$.

Against Defense II: The second defense is more effective as it counts the slow execution of tasks. Once the attacker launches the delayed execution process, it will be detected regardless if the task is finished or not. In this case, the attacker has to consider the abnormality threshold τ, and cannot delay every task assigned to it. The best strategy for the adversary is to select a set of victim tasks to attack without violating the abnormality policy. For example, if the system threshold τ is 5%, then the adversary can delay at most 5 tasks out of 100 tasks assigned to the compromised node.

Therefore, the problem of developing the best attacking strategy becomes how to select k victim tasks to attack in order to maximize the job execution time. The constraint parameter k is derived from the abnormality threshold τ. We find that the special case of this problem is similar to the traditional job scheduling problem, and can be reducible to the bin-packing problem which is NP-hard. The proof is omitted due to the page limit. In this paper, we present a greedy algorithm to solve the problem. The idea is to iteratively select the task that can cause the longest delay. The detailed algorithm is shown in Algorithm 1.

Algorithm 1 Attacker's Algorithm against Defense II

```
1: Initialize the result set of selected tasks $RET = \{\}$
2: For each job's map task, calculate $\mathcal{D}(\alpha, t)$
3: while $|RET| < k$ do
4:     for each job $j$'s map task do
5:         $d_{j1} = D(t_i)$, $t_i$ is in the last wave
6:         $d_{j2} = D(t_i)$, $t_i$ is in the second last wave
7:         $M' = M_j \% (m \cdot r)$
8:         $p_j = 1 - \binom{M'}{(m-1) \cdot r} / \binom{M'}{m \cdot r}$
9:     end for
10:    Sort all $d_{j1} \cdot p_j$ and $d_{j2}$ (totally $2 \cdot j$ values) in the descending order
11:    Add the first task in the list to the return set $RET$
12:    Rearrange the task executions considering the tasks in $RET$ being delayed
13: end while
```

In this algorithm, we use RET to represent the selected tasks for the attack, initializing it as empty (line 1). Then we consider each job's map tasks as the candidates, and calculate the delayed of each task if it is under attack (line 2). Then we use a while loop to select k tasks as the targets (lines 3–13). In each round, we enumerate all the jobs and calculate two delays for each job's map tasks. d_{j1} is the delay caused by attacking the task in the last wave while d_{j2} is the delay when we attack a task in the second last wave. Based on the analysis in the previous subsection and Eq. 4, we find that under the same circumstance, it is more effective to delay the tasks in the tailing waves than the tasks in the earlier waves. Thus the tasks in the last wave are the best candidate for the attack. However, not every node would be able to serve in the last wave. So we consider two candidates for each job, the task in the second last wave which for sure will be assigned to the compromised node, and the task in the last wave which causes the most damage, but may not be assigned to the compromised node. In our algorithm, we use p_j to represent the probability that the compromised node can execute a task in the last wave. In line 7, M_j is the total number of map task in job j, and M' is the remainder of M divided by $m \cdot r$ which is the number of map tasks in the last wave. The Hadoop resource manager will randomly pick containers across the cluster to serve these M' tasks. The equation in line 8 calculates the probability p_j. Eventually, the algorithm considers all the candidates, two values from each job. For the delay caused by the task in the last wave, we use the expected value of $p_j \cdot d_{j1}$ for comparison. The best choice for the attack is the task with the longest delay (lines 10–11). Once a candidate is selected for the attack, we will consider the new arrangement of the task executions assuming that tasks in RET will be delayed by $\mathcal{D}(\alpha, t)$. Then the algorithm repeats the process and selects the next candidate. It terminates when there are k selected tasks in the result set RET.

V. Mitigation Scheme

In this section, we briefly discuss some mitigation schemes that can protect a Hadoop system from the delayed execution attack. First, as we have discussed, Defense II is a better strategy than Defense I. The Hadoop system needs to monitor the execution time of every task on each node. A abnormally slow task is a suspicious sign of attacks. However, in practice, it is difficult to set an appropriate value for the threshold τ because a benign node may occasionally yield a slow execution. On the other hand, the attack may behave normally in the earlier waves, and then attack the last wave or second last wave to significantly prolong the job execution.

In this paper, we present two techniques that can help reduce the negative impact when the system is under attack. Our main intuition is to protect the execution of the last wave which is the major target and the most important wave for the job execution. First, the Hadoop system should select the most reliable or fastest nodes to execute the tasks in the last wave. Various metrics can be used here to sort all the nodes, such as the average task execution time and the top/bottom task execution time. This is a stronger protection than the abnormality threshold checking. If the compromised node has launched delay execution in the earlier waves, but still within the threshold τ, it still has the equal probability to host the tasks in the last wave. In our mitigation scheme, however, if the system can find sufficient more reliable nodes (e.g., with no slow execution at all), the compromised node will be excluded from executing the last wave.

Second, we propose to change the policy for launching a speculative task in the last wave. In order to defend the system, we need a more aggressive policy. We may start a speculative task even if the estimated finish time of a speculative task is later than the original task. In our mitigation scheme, when the original task becomes slow, we use a parameter $\gamma \in (0, 1)$ to adjust the comparison. Let T_o and T_s be the estimated execution time of the original task and a speculative task. In stead of comparing T_o and T_s, our scheme checks whether T_s is short than $\gamma \cdot T_o$. If it is true, a speculative task will be launched. In our evaluation, we set $\gamma = 0.75$.

In fact, both techniques cause additional overhead in regular circumstances with no attacks. But they do save a lot of execution time when the system is under attack. In Section VI, we present evaluation results that show the effectiveness of our mitigation scheme.

VI. Implementation and Evaluation

In this section, we will first introduce the system implementation and present the performance evaluation results of both SEINA and our mitigation scheme.

A. System Implementation

First, to achieve the attacking of SEINA, we create a set of new modules in the node manager: the Container Monitor (**CM**), the Container Controller (**CC**), and the Container Operator (**CO**). **CM** is responsible for monitoring the execution of each container including the status of each container, the execution time of each completed container, the progress of each running container, and so on. **CC** is in charge to control a container to be paused or resumed according to the algorithms in SEINA and the statistics of **CM**. While receiving the commands from **CC**, **CO** triggers the process commands in Linux to pause or resume a container. Second, we implement our mitigation scheme on Hadoop YARN version 2.7.1. We modified the *RMContainerAllocator* of *ApplicationMaster* to assign tailing tasks of every job to the appropriate node managers, and *DefaultSpeculator* to change the policy for launching a speculative task in the last wave.

B. Testbed Setup and Workloads

All experiments are conducted on NSF CloudLab platform at the University of Utah [17]. Each server has 8 ARMv8 cores at 2.4GHz, 64 GB memory and 120 GB storage. We launched a Hadoop YARN cluster with 1 master node and 8 slave nodes. In each slave node, we configured 16 vcores and 64 GB memory.

Our workloads for evaluation consider general Hadoop benchmarks with large datasets as the input. In particular, we use two datasets in our experiments including 20 GB wiki category links data and 20 GB synthetic data. The wiki data includes wiki page categories information, and the synthetic data is generated by the tool TeraGen in Hadoop. We choose the following three Hadoop benchmarks from Hadoop examples library to evaluate the performance: (1) *Terasort*: Sort (key,value) tuples on the key with the synthetic data as input. (2) *Word Count*: Count the occurrences of each word with a list of Wikipedia documents as input. (3) *Wordmean*: Count the average length of the words with a list of Wikipedia documents as input.

C. Performance Evaluation

Our performance metric is the makespan of a batch of jobs. In the cluster, we assume only one node manager is attacked and others can work normally. We mainly compare the changing of the makespan in various situations. Two categories of tests are conducted with different workloads: *simple workloads* consist of the same type of jobs and *mixed workloads* represent a set of hybrid jobs. We generate 6 jobs of the same benchmarks in each test of simple workloads. And for testing mixed workloads, we mix all three benchmarks above and generate 2 jobs for each benchmark. For each job of both simple and mixed workloads, the input data is 20 GB. There are 80 map tasks and 10 reduce tasks created by each job. Each map task requires 1vcore + 2GB memory and each reduce task requires 1vcore + 3GB memory. In the rest of this subsection, we separately present the evaluation results of both the attacks and the mitigation scheme.

1) Performance Impact by the Attack: For our first experiment, we conduct the test results of the makespan impact by various kinds of attack: the thorough attack, the attack against Defense I, and the attack against Defense II.

Thorough attack. First, we run the test with the thorough attack. In other words, all the task containers in the attacked node manager can be paused during the execution according to the algorithm in IV. To show the performance impact of the thorough attack, besides the makespan without any attack, we also test the results with one node manager blocked in the cluster. As illustrated in Fig. 4, *Hadoop:Non-attack* indicates the makespans of both simple and mixed workloads without any attack. *Node Blocked* shows the makespans with one node manager blocked, i.e., there are only 7 slave nodes working in the cluster. Compared to *Hadoop:Non-attack*, the average makespan with one slave node blocked only increases 14.3%. *Thorough Attack* presents a much better impact on reducing the performance of the makespan which is averagely 76.4% larger than *Hadoop:Non-attack*.

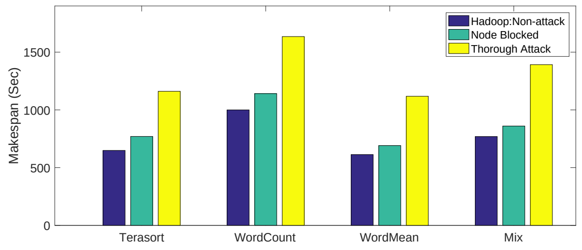

Fig. 4: Makespan without any attack: (1) Hadoop:Non-attack (Native Hadoop) (2) Node Blocked (Native Hadoop with one nodeManager blocked), and (3) with thorough attack.

Attack against Defense I. Second, we consider a policy in the ResourceManager to block the suspicious NodeManagers with several killed tasks. In Hadoop, a task may be killed because of two reasons: 1) its execution time is too long and beyond a threshold (600 seconds by default) in ApplicationMaster; 2) it runs too slow and its speculative task has finished earlier. If the ResourceManager detects any job with n percent of tasks killed in a NodeManager, such NodeManager will be blocked as an attacker. In this case, SEINA needs to control the pause time of every attacked container and guarantee each task can be finished before its speculative task and won't execute longer than the threshold configured in ApplicationMaster. Fig. 5 shows the test results of both simple and mixed workloads. During the experiments, we set a severe value of $n = 0.15\%$. For the job with 80 map task and 10 reduce tasks, only 1 killed task per job can be generated by SEINA. *SEINA:Time-control* indicates the attacking results of SEINA under Defense I. On average, the makespan under *SEINA:Time-control* prolongs 46.7% compared to the one without any attack. And the makespan under *SEINA:Time-control* is only 16% less than *Thorough Attack*.

Attack against Defense II. Furthermore, we consider another strict policy in the ResourceManager which monitors the execution time of every task on each NodeManager and block the one with several 'slow' containers. Specifically, for each job, if there are k tasks on a NodeManager running slower than the average task execution time, such NodeManager will be considered as an attacker and get blocked. In this case, instead of the thorough attack, the attacker needs to control the number of attacked task containers within the number of k per each job. Fig. 6 illustrates the test results with $k = 3$. During the

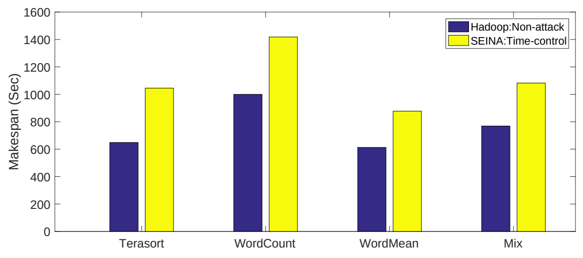

Fig. 5: Makespan under SEINA: controlling the pause time of every attacked container

test, SEINA only attacks 1 or 2 task containers for every job and still significantly affects the performance of the makespan of both simple and mixed workloads. *SEINA:1* / *SEINA:2* indicate the test results by attacking one/two task per job. For a total number of 80 tasks for each job, SEINA increases meanly 36.2% and 61% of the makespan by attacking 1 and 2 tasks per job. On average, the makespan of *SEINA:2* is only 8.7% less than the one of *Thorough Attack*. Overall, SEINA indicates both efficient and effective attacking results.

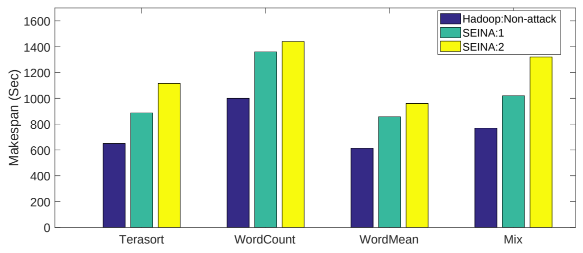

Fig. 6: Makespan under SEINA: (1) SEINA:1: attack one task per job, and (2) SEINA:2: attack two tasks per job.

2) Performance of the Mitigation Scheme: In the end, we run the experiments of our mitigation scheme. Fig. 7 illustrates the performance with SEINA attacking against Defense II, and the makespan of the mitigation scheme in a normal environment without any attack. *Ours:Attack* indicates the makespans with the mitigation scheme under SEINA and the average makespan only increase 11.1% compared to the one without any attack. *Ours:Non-attack* shows the makespans with the mitigation scheme without any attack. Without any attack, the mean makespan of *Ours:Non-attack* is just 5.6% larger than the one of the native Hadoop. From the test results, the mitigation scheme can effectively defense the attack of SEINA and causes a minor overhead in the environment without any attack.

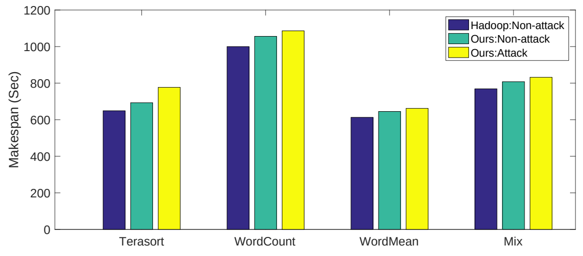

Fig. 7: Makespan under the mitigation scheme

VII. Conclusion

This paper studies a security problem in big data processing frameworks. Our goal is to degrade the overall system performance by launching attacks from a compromised internal node. We take Hadoop YARN as a representative platform and develop a new attack scheme SEINA which can effectively prolong the makespan of the applications against two basic defense schemes. Furthermore, we create a mitigation scheme to protect the Hadoop system from such attack. Our evaluation is based on experiments with various workloads and settings. The results show a significant performance reduction caused by SEINA on the native Hadoop system, and an effective protection from our mitigation scheme against SEINA.

References

[1] Apache Hadoop. http://hadoop.apache.org.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[3] Apache Spark. http://spark.apache.org.

[4] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, 2012.

[5] Amazon AWS EMR. https://aws.amazon.com/emr/.

[6] M Lockneed. Awareness, trust and security to shape government cloud adoption. *LM Cyber Security Alliance and Market Connection White Paper*, 2010.

[7] Kevin Hamlen, Murat Kantarcioglu, Latifur Khan, and Bhavani Thuraisingham. Security issues for cloud computing. *Optimizing Information Security and Advancing Privacy Assurance: New Technologies: New Technologies*, 150, 2012.

[8] Jiaqi Zhao, Lizhe Wang, Jie Tao, Jinjun Chen, Weiye Sun, Rajiv Ranjan, Joanna Kołodziej, Achim Streit, and Dimitrios Georgakopoulos. A security framework in g-hadoop for big data computing across distributed cloud data centres. *Journal of Computer and System Sciences*, 80(5):994–1007, 2014.

[9] Hadoop in secure mode. https://hadoop.apache.org/docs/r2.7.1/hadoop-project-dist/hadoop-common/SecureMode.html.

[10] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[11] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet computing*, 13(5):14–22, 2009.

[12] Jianzhe Tai, Deng Liu, Zhengyu Yang, Xiaoyun Zhu, Jack Lo, and Ningfang Mi. Improving flash resource utilization at minimal management cost in virtualized flash-based storage systems. *Cloud Computing, IEEE Transactions on*, PP:1–1, 2015.

[13] Kehuan Zhang, Xiaoyong Zhou, Yangyi Chen, XiaoFeng Wang, and Yaoping Ruan. Sedic: privacy-aware data intensive computing on hybrid clouds. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 515–526. ACM, 2011.

[14] Yeonhee Lee and Youngseok Lee. Detecting ddos attacks with hadoop. In *Proceedings of The ACM CoNEXT Student Workshop*, page 7. ACM, 2011.

[15] Qingni Shen, Yahui Yang, Zhonghai Wu, Xin Yang, Lizhe Zhang, Xi Yu, Zhenming Lao, Dandan Wang, and Min Long. Sapsc: security architecture of private storage cloud based on hdfs. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 1292–1297. IEEE, 2012.

[16] Safwan Mahmud Khan and Kevin W Hamlen. Hatman: Intra-cloud trust management for hadoop. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 494–501. IEEE, 2012.

[17] Robert Ricci, Eric Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX*, 39(6), December 2014.