



Article

SHIYF: A Secured and High-Integrity YARN Framework

Junyi Deng ¹ , Yanheng Liu ^{1,2}, Jian Wang ^{1,2,*}  and Shujing Li ¹

¹ College of Computer Science and Technology, Jilin University, Changchun 130012, China; dengjunyi@vip.sina.com (J.D.); yhliu@jlu.edu.cn (Y.L.); lsj202@jlu@163.com (S.L.)

² Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012, China

* Correspondence: wangjian591@jlu.edu.cn; Tel.: +86-431-8515-9419

Received: 17 April 2019; Accepted: 11 May 2019; Published: 15 May 2019



Abstract: Cloud computing is becoming a powerful parallel data processing method, and it can be adopted by many network service providers to build a service framework. Although cloud computing is able to efficiently process a large amount of data, it can be attacked easily due to its massively distributed cluster nodes. In this paper, we propose a secure and high-integrity YARN framework (SHIYF), which establishes a close relationship between speculative execution and the security of Yet Another Resource Negotiator (YARN, MapReduce 2.0). SHIYF computes and compares the MD5 hashes of the intermediate and final results in the MapReduce process by launching the speculative executions in a certain ratio, which is able to find actual and potentially malicious nodes in the Hadoop cluster. The prototype of SHIYF is implemented based on Hadoop 2.8.0. In this paper, theoretical derivations and experiments show that SHIYF not only guarantees the security and high integrity of the MapReduce process but also successfully locates the malicious nodes and the potential malicious ones in Hadoop, while increasing overhead slightly. Furthermore, the malicious node detection ratio is more than 87%.

Keywords: cloud computing; Hadoop; MapReduce; YARN; speculative execution; security; integrity

1. Introduction

With the rapid development of hardware, software, and high-speed networks, many cloud service providers (e.g., Google and Amazon) are establishing increasing cloud computing (CC) [1,2] realities around the world, as shown in Figure 1. However, many organizations and customers remain reluctant to accept CC because of security issues [3,4]. Therefore, solving relevant security problems has considerable significance for the long-term development of CC [5].

Some safety precautions are already eliciting attention [6]. For instance, Gartner et al. identified seven security issues of CC that must be solved [7]. Grobauer et al. discussed the security vulnerabilities of the cloud platform [8]. Jansen et al. proposed guidelines on privacy in public CC [9]. Furthermore, the security guidance of CC is published by the Cloud Security Alliance and IEEE [10].

Hadoop [11] is considered the most widely used CC platform [12], and it represents the state-of-the-art efficient framework for processing vast amounts of distributed data [13]. However, most researchers are still focusing on the performance and application of MapReduce rather than its security. For example, Dawei Jiang et al. identified five design factors that affect the performance of Hadoop [14]. Yanpei Chen et al. built the case for going beyond benchmarks for MapReduce performance evaluations [15]. Rares Vernica et al. studied how set similarity joins can be efficiently performed in parallel using the popular MapReduce framework [16]. A few studies have been conducted on the security of MapReduce, such as one study that focused on Airavat, which is

a MapReduce-based system that provides strong security and privacy guarantees for distributed computations on sensitive data [17]. Reference [18] introduces a new privacy-preserving encoding with “somewhat homomorphic” properties for MapReduce. In addition, some whitepapers about security designs of Hadoop and MapReduce have been published [19–21].

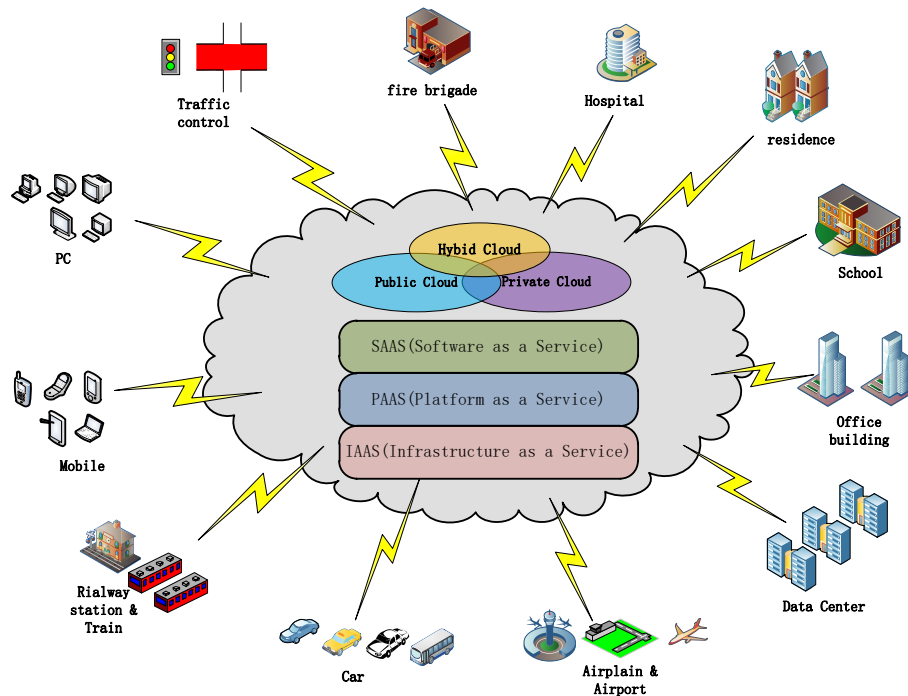


Figure 1. The deployment model of the cloud computing.

However, there are still several security breaches in Hadoop 2.0 as follows.

- Service identity forging. Since there is not the service certification, any malicious node can masquerade as a security node and join Hadoop cluster to get/calculate data as long as it knows the ResourceManager (RM) address.
- User identity forging. Because there is not the user authentication, any malicious client can fake the user identity to get Hadoop Distributed File System (HDFS) data or do job management.
- Lack of authorization mechanism. A client can do anything, such as a job submitted by user A can be killed by user B at will.
- Data communications are not encrypted. They are vulnerable to eavesdropping.

Yet Another Resource Negotiator (YARN, also known as MapReduce 2.0/MRv2) is one of the key features in the second-generation Hadoop and provides resource management and scheduling for large-scale MapReduce environments [22]. Research on the performance or security of YARN remains in its infancy. For example, Li Ping et al. proposed an energy-efficient service level agreement-aware scheduling scheme that allocates an appropriate amount of resources to MapReduce applications with YARN architecture [23]. Reference [24] presented a new methodology for determining desired hardware and software configuration parameters for MapReduce 2.0 applications; thus, the representative applications achieved up to 5× performance improvement. An energy-aware fair scheduling framework based on YARN (denoted as EFS) is proposed by Shao Yanling et al., which can effectively reduce energy consumption whilst meeting the required service level agreements (SLAs) [25]. Diarchy increases the reliability of YARN based on the sharing and backup of responsibilities between two masters working as peers [26]. A SECcapacity scheduler was proposed for the requirement of isolating the user’s job and data security [27]. Reference [28] proposed a novel partitioner for improving YARN performance

(NPIY) based on Hadoop 2.6.0, which adopts an innovative parallel sampling method to distribute intermediate data.

In Hadoop, speculative execution is equal to replication (also known as double-check), which sacrifices space for time. Replication-based techniques mainly rely on redundant computation resources to execute duplicated individual tasks for verifying the consistency of results [29]. W. Wei et al. proposed a service integrity assurance framework for MapReduce (SecureMR) based on Hadoop 1.0 [30]. SecureMR provides a decentralized replication-based integrity verification scheme for ensuring the integrity of MapReduce in open systems. Although MapReduce is a programming model for data processing on YARN, executing duplicated tasks (using speculative execution in Hadoop 2.0) is still an effective way to prevent service identity forging for identifying the malicious nodes in Hadoop cluster.

In this paper, we focus on improving the security of YARN. A secure and high-integrity YARN framework (SHIYF) is proposed by extending Hadoop 2.8.0. Sacrificing space for security is the key idea of SHIYF. Extensive theoretical derivations and experiments are performed to prove the framework’s validity, security, and malicious node detection efficiency. The main contributions are summarized as follows.

1. Speculative execution is used for Hadoop YARN security.
2. Some significant security improvements are made to Hadoop 2.0 in SHIYF, such as ensuring the correctness of MRv2 results and locating the malicious nodes and the potential ones in the Hadoop cluster.
3. A prototype of SHIYF is implemented based on Hadoop 2.8.0.
4. Results of theoretical derivations show that SHIYF adds 30% speculative tasks in the MRv2 job and achieves a malicious node detection ratio of more than 90%.
5. Experiment results show that SHIYF can ensure the security of MRv2 services while increasing overhead slightly. Moreover, the malicious node detection ratio is between 87% and 93.3%.
6. This finding is in line with the expectation of theoretical derivation.

The remainder of the paper is organized as follows. In the next section, we introduce the SHIYF design and implementation in detail. Section 3 provides the theoretical derivations. Section 4 reports the experimental results of SHIYF and compares them with the theoretical results. Section 5 contains the conclusions and prospects for future work.

2. SHIYF Design and Implementation

2.1. SHIYF Design

Given that SHIYF verified the validity of the intermediate and final results generated by Map and Reduce in a programming model, more TaskAttempts launched the speculative executions in a certain ratio, in contrast with YARN. These additional TaskAttempts executed the same tasks and computed the MD5 Message-Digest Algorithm (MD5) hashes of results. The programming model of SHIYF is shown in Figure 2.

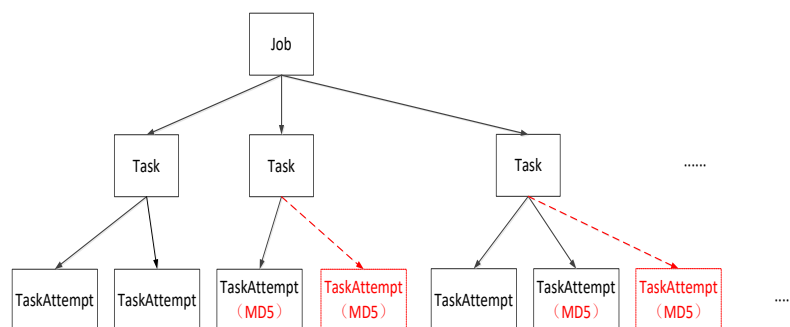


Figure 2. The programming model of a secure and high-integrity YARN framework (SHIYF).

MRAppMaster is the ApplicationMaster implementation of MapReduce, which allows MapReduce to be run directly on YARN. Its main function is to manage the life cycle of the job, including:

- Job creation, initialization, startup, and so on.
- Apply to RM for resources and reallocate resources.
- Container startup and release.
- Monitoring the operation status of the job.
- Job recovery.

In the runtime environment of SHIYF, MRAppMaster provided a set of security mechanisms, including the secured task duplication and assignment, intermediate result check, and final results verification. MRAppMaster could be applied to two containers to execute the same TaskAttempt for a task by the speculative execution. When MRAppMaster received two MD5 hashes from different containers, it would compare whether they were consistent or not. If they were the same, then MRAppMaster considered that the task had been completed and the result was correct. Otherwise, it applied for the third container to execute the same TaskAttempt again to verify the result. Finally, MRAppMaster considered two results with the same MD5 hashes as the right result. If not, then it would judge that this task failed. Therefore, SHIYF is shown in Figure 3. Simply and clearly, an MRAppMaster controls only one task so that three different jobs are used.

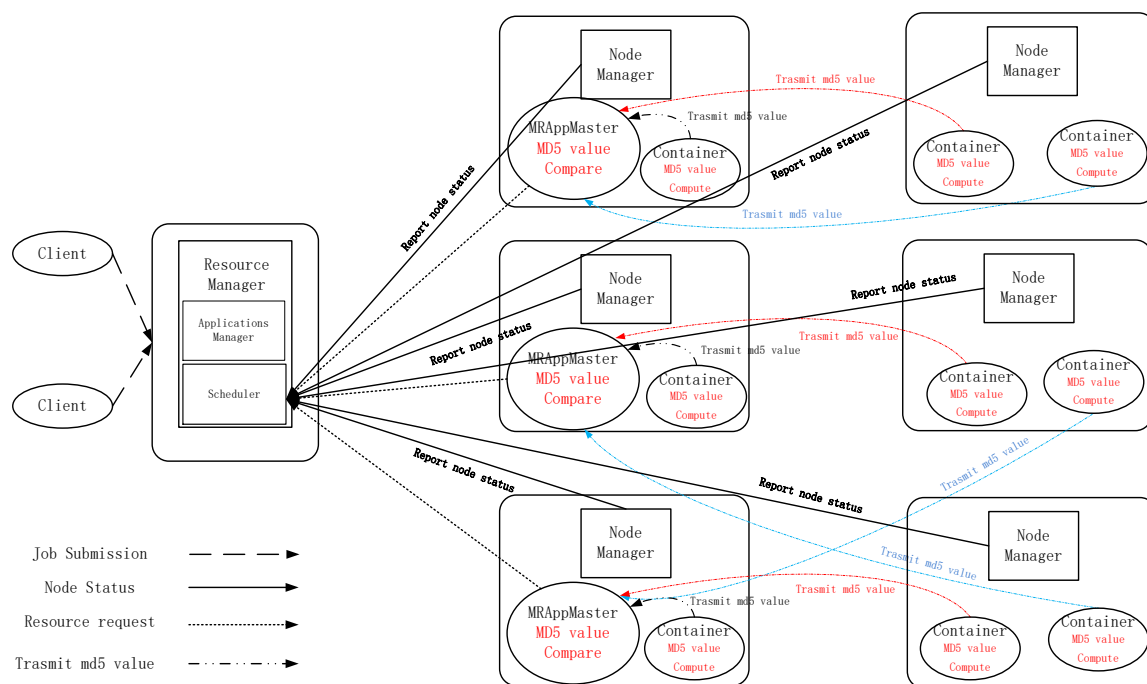


Figure 3. The secured and high-integrity Yet Another Resource Negotiator (YARN) framework.

2.2. SHIYF Implementation

In YARN, RMAp is a data structure that preserves an application life cycle in RM. Its realization class is RMApImpl. This class maintains an application state machine that records several application states and state-driven events. The finite-state machine (FSM) of RMApImpl is shown in Figure 4. When MRAppMaster is launched, the application will enter into the core state “RUNNING.” Every application may run several times. The transitions of states are determined by the return values of MRAppMaster. RMAp judges that an application has failed when all RMApAttempts failed. Therefore, MRAppMaster is the most important module in SHIYF.

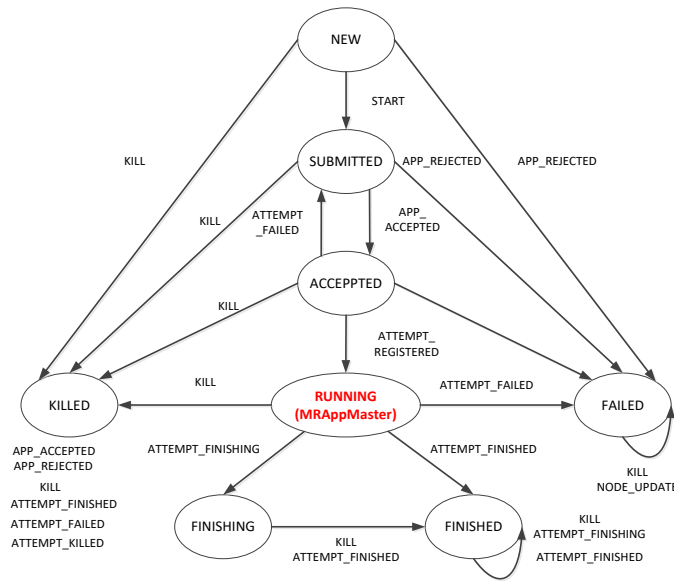


Figure 4. The finite-state machine of RMAppImpl.

In addition, because YARN uses the asynchronous programming model based on an event-driven mechanism, every component is an event handler. MRAppMaster establishes the relations with other components by the events and assigns all types of events to the corresponding schedulers. Figure 5 shows the components and the services of MRAppMaster. ContainerAllocator (CA), Speculator, Job, Task, and TaskAttempt must be redesigned to implement SHIYF.

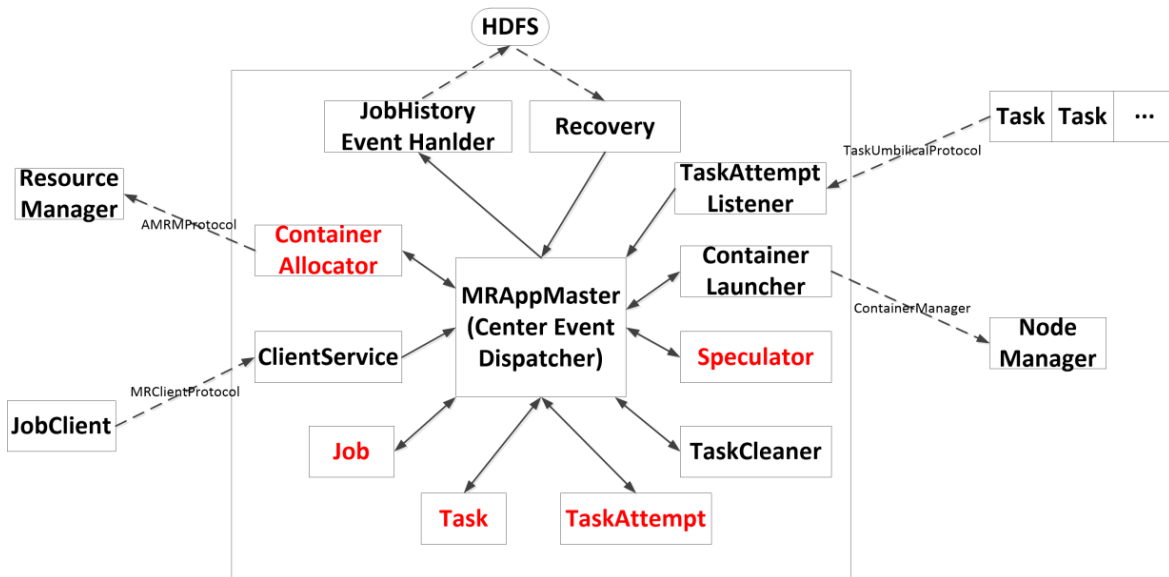


Figure 5. The components and services of MRAppMaster.

2.2.1. SHIYF ContainerAllocator

CA is a resource scheduler. It divides the application resource tasks into three categories, such as Failed Map, Reduce, and Map, from high to low priority. The workflow of SHIYF CA is as follows:

Step 1. To add the speculative tasks in a certain ratio, the chosen Maps/Reduces and their double resource applications would be sent to RM at the same time.

Step 2. If the scheduling conditions of Reduces were met, then CA would give priority to them.

Step 3. CA would be allowed double resource occupation simultaneously in SHIYF.

Step 4. CA would apply resources for a task again once it failed before.

Step 5. If a task ran too slowly, the CA would apply extra resources to start its speculative task.

Step 6. CA would withdraw all resource distributions to this node when it failed too many times. In SHIYF, if a node failed more than five times, then CA would withdraw all its resource applications and judged it as the malicious node.

The CA in SHIYF is shown in Figure 6.

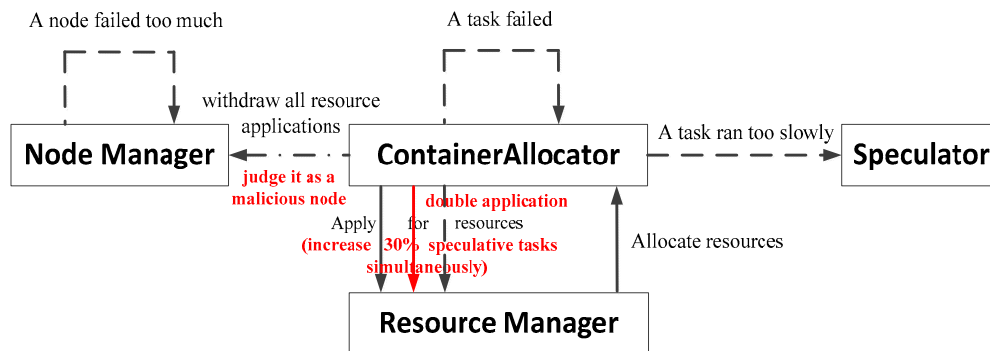


Figure 6. The ContainerAllocator in SHIYF.

2.2.2. SHIYF Speculator

In Hadoop, the speculative execution sacrifices space for time. However, sacrificing space for security is the key idea of SHIYF. We designed it to conduct speculative execution repeatedly for MD5 computation and hash comparison. The corresponding event handler of speculative execution is referred to as a speculator.

Prior to launching a new speculator in SHIYF, the redesigned MRAppMaster must check whether the current task conforms to the following three conditions:

1. Whether the current task had already a backup task. Every task could had two speculative tasks and a maximum of three.
2. The ratio of completed tasks was not less than `MINIMUM_COMPLETE_PROPORTION_TO_SPECULATE` (5%). Only then could the Speculator had sufficient historical task information to estimate `estimatedReplacementEndTime`.
3. `DefaultSpeculator` could launch speculative execution in a certain probability without calculating the `speculationValue`.

Because when the Speculative Execution Ratio reached 30%, SHIYF could achieve a desired malicious node detection ratio according to Section 3. We redesigned three parameters in SHIYF as follows.

- `MINIMUM_ALLOWED_SPECULATIVE_TASKS = 10`. It represents the minimum number of total speculative tasks that are allowed for a job.
- `PROPORTION_TOTAL_TASKS_SPECULATABLE = 0.35`. It denotes the highest percentage of speculative tasks to the total tasks is 35%.
- `PROPORTION_RUNNING_TASKS_SPECULATABLE = 0.3`. It indicates the highest percentage of speculative tasks to all running tasks is 30%.

Therefore, the number of speculative tasks that are allowed to perform in a job (`numberAllowedSpeculativeTasks`) is the maximum of the following three values.

- `MINIMUM_ALLOWED_SPECULATIVE_TASKS`
- `PROPORTION_TOTAL_TASKS_SPECULATABLE * totalTaskNumber`
- `PROPORTION_RUNNING_TASKS_SPECULATABLE * numberRunningTasks`

Meeting the requirements of SHIYF and limiting the number of speculative tasks in a job, which can effectively prevent the waste of resources caused by a large number of tasks launching speculative tasks at the same time.

2.2.3. SHIYF Security Control

To ensure the security of YARN, SHIYF should compute and compare the MD5 hashes of intermediate and final results. This process involves three services, namely, Job, Task, and TaskAttempt. Their communication processes in SHIYF are shown in Figure 7. Two types of tasks are used; one is the normal task, and the other is chosen to check the MD5 hashes of results.

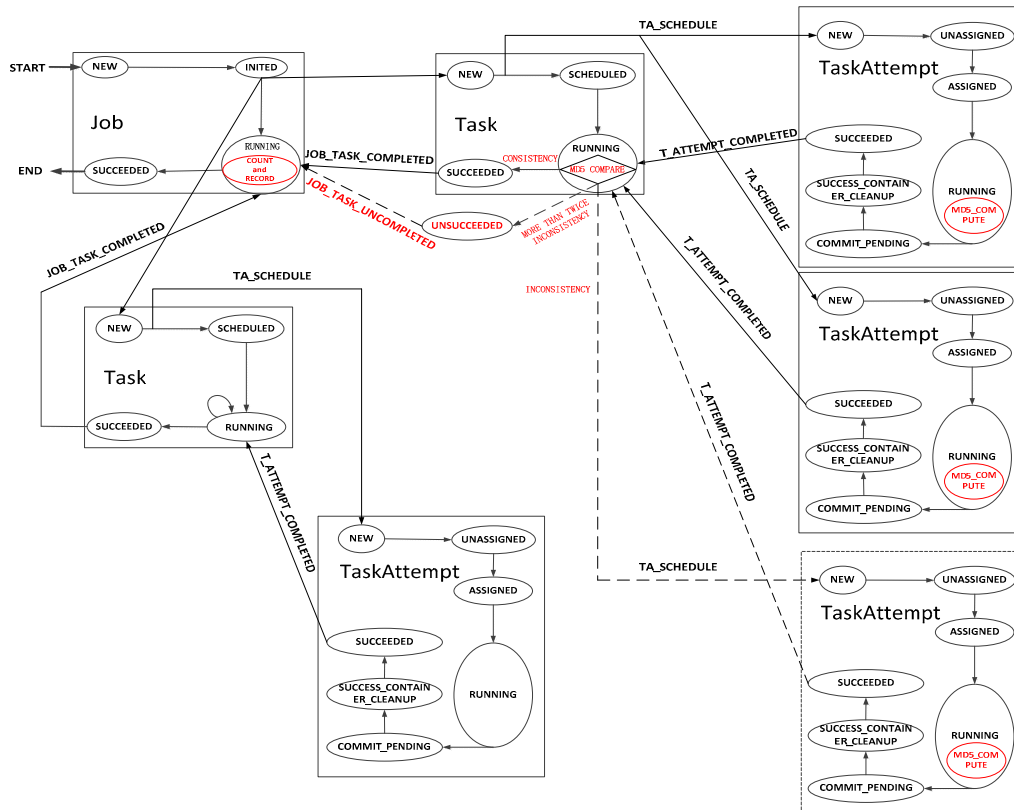


Figure 7. The communication of Job, Task, and TaskAttempt in SHIYF.

To ensure the validity of results and locate the malicious nodes, Job, Task, and TaskAttempt have the following additional functions:

- In Job, the hostnames of nodes that failed to execute tasks were recorded and written to HDFS logs. If failure occurred more than five times, then SHIYF would consider these nodes the malicious nodes.
- If two TaskAttempts disposed of the same data but returned the different hashes, then Task launched the other speculative TaskAttempt to verify the result again. The node returned the wrong hash once, it would be recorded as the potential malicious node. If the hash comparison failed twice, then Task returned “JOB_TASK_UNCOMPLETED” to Job and restarted. Moreover, the three nodes in this task would all be considered the potential malicious nodes.
- TaskAttempt with speculative execution should compute the MD5 hash of the result and transmit it to Task; however, the normal one does not do that. They are highlighted in red in Figure 7.

We could find the malicious nodes and the potential ones by reviewing the logs on HDFS.

2.2.4. SHIYF State Management

Many components and services were used in SHIYF. The FSMs of TaskAttempt, Task, and Job should be changed accordingly to achieve SHIYF security control.

First, the FSM of SHIYF TaskAttempt is shown in Figure 8. “TA_MD5_COMPUTE” was added to the state “RUNNING” to compute the MD5 hashes of the intermediate and final results. However, only tasks selected to check the result validity executed the speculative tasks and MD5 hash computations. The verification process of TaskAttempt in SHIYF is as follows:

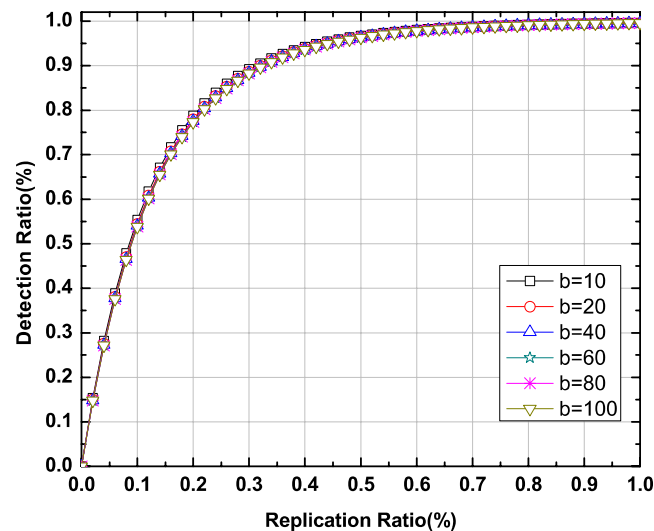


Figure 8. The effect of the number of blocks (b) on the Detection Ratio.

Step 1. TaskAttempt judged whether the task was checked based on the signature added by Job.

Step 2. TaskAttempt saved the related messages of container runtime such as LaunchTime, trackerName, httpPort, and MD5 hash of the result.

Step 3. TaskAttempt then renewed the counter messages and informs the history server and speculator service.

Step 4. TaskAttempt computed and transmitted the MD5 hashes to Task and informed it that this attempt was successful.

In addition, three relevant services, “TA_UPDATE,” “TA_UPDATE/StatusUpdater,” and “TA_CONTAINER_COMPLETED,” need to be changed accordingly to control and trigger the state transition, as emphasized in Figure A1.

Second, the FSM of SHIYF Task is shown in Figure A2. Three important improvements are as follows:

1. To check some task results, SHIYF need TaskAttempts and their speculative executions to run in parallel until they completed and returned MD5 hashes. Therefore, Task in SHIYF should be allowed two or three speculative Attempts retained at the same time, namely, Task will not kill other corresponding Attempts when it receives “T_ATTEMPT_COMMIT_PENDING” recording the Attempt running.
2. When Task received “T_ADD_SPEC_ATTEMPT,” it created a new speculative Attempt to run the same task. All the tasks were chosen for checking, and their speculative executions were added the sign “Extra_SETask” as the determined criteria of launching MD5 computation in TaskAttempt.
3. When a TaskAttempt runs successfully, Task in YARN will receive “T_ATTEMPT_SUCCEEDED” and kill other Attempts. However, SHIYF needed to compare the MD5 hashes of the two same TaskAttempts to ensure the validity of the results. Therefore, even if an Attempt has been completed and the MD5 hash has been returned, Task still should wait for the other speculative

TaskAttempts until the end. Thus, the other several relevant improvements had been occurred as follows.

- An event “T_ATTEMPT_MD5_COMPARE” was added in “RUNNING.” This event triggered MD5 hash comparison.
- If the first comparison failed, but the second or the third comparisons succeeded, Task would add a “SUCCEED_FALSE” to mark the Attempt being executed successfully but returning a wrong MD5 hash once. At the same time, Task recorded the hostnames of these TaskAttempt machines as evidences of the potential malicious nodes.
- “TA_ATTEMPT_SUCCEEDED,” “T_ADD_SPEC_ATTEMPT,” and “T_ATTEMPT_COMMIT_PENDING” in “SUCCEEDED” must be changed accordingly to control and trigger the state transition.

Finally, the FSM of the SHIYF Job is shown in Figure A3. When the job entered a “RUNNING” state, the entire event would turned into task until it returned the trigger events (e.g., JOB_TASK_ATTEMPT_COMPLETED, JOB_MAP_TASK_RESCHEDULED, JOB_TASK_ATTEMPT_FETCH_FAILURE, JOB_TASK_COMPLETED, and JOB_COMPLETED). The trigger events and the corresponding states marked with red in Figure A3 must be redesigned. For instance, when SHIYF Job received the “JOB_TASK_COMPLETED” trigger event, it not only calculated the numbers of completed tasks, failed tasks, and killed tasks, but also recorded the hostnames of the malicious nodes and the potential ones.

Therefore, the corresponding SHIYF ResourceManager and NodeManager implementations are shown in Figures A4 and A5.

3. Theoretical Derivation

3.1. Theoretical Arithmetic

Although the Map speculative task and the Reduce speculative task are slightly different, their principles are the same. Thus, we use the Map task replication as an example to show the theoretical arithmetic.

To easily compare differences and similarities without losing generality, we set every MRv2 job to dispose of the same size of data. Thus, the total blocks were fixed in every experiment; moreover, the data of every block were different. Every Map task that processed only one block implied that the number of the copied blocks was equal to the number of the replicated Map tasks. We assumed the number of blocks (Map tasks) was b . A container was the abstraction conception of a resource set in YARN. It would be allocated by RM and supervised by NodeManager (NM). Every task must be executed in a container; thus, the number of containers was also b .

Despite the security in SHIYF, replicating all the MRv2 tasks by speculative executions is not practical, because doing so consumes considerable resources and time. We introduced Execution Ratio (E_r) to indicate that $b \times E_r$ blocks would be duplicated. We let N be the number of the Map speculative tasks, then

$$N = b \times E_r \quad (1)$$

If an MRv2 job involves one MRAppMaster and n containers, then m containers might be malicious and $m < n$. The aims of SHIYF are to ensure the integrity of MRv2 results and find the malicious nodes. Theoretical arithmetic will show the relationship between Detection Ratio (D_{ratio}) and the above parameters as follows:

Step 1. P_{am} denotes the probability that a malicious Map task is present in b Map tasks. It is computed as

$$P_{am} = 1/b \quad (2)$$

P_{nm} is the probability that any Map task is not malicious, and it is obtained as

$$P_{nm} = 1 - 1/b = 1 - P_{am} \quad (3)$$

Step 2. If N duplicated blocks are in an MRv2 job, P_{Ns} denotes the probability that all N Map tasks are secure, then

$$P_{Ns} = P_{nm}^N = (1 - P_{am})^N \quad (4)$$

Step 3. In case a Map task is not executed in a secure container, P_{Nam} denotes the probability that a malicious Map speculative task occurs in N at least, then

$$P_{Nam} = 1 - P_{Ns} = 1 - (1 - P_{am})^N \quad (5)$$

Step 4. We suppose that malicious nodes execute the vicious actions in P probability. We let P_{mea} be the probability of the malicious containers (nodes) executing the vicious actions, and it is obtained as

$$P_{mea} = P_{Nam} \times P = (1 - P_{Ns}) \times P \quad (6)$$

Step 5. We can obtain the probability that the malicious nodes do not conduct the baleful behaviors. P_{mna} is computed as

$$P_{mna} = 1 - P_{mea} = 1 - P_{Nam} \times P \quad (7)$$

Step 6. The variable t represents the number of jobs executed by MRv2. If the malicious nodes perform the tasks correctly in t MRv2 jobs, then we can obtain this probability P_{mct} as

$$P_{mct} = P_{mna}^t = (1 - P_{mea})^t \quad (8)$$

Step 7. In case of the malicious nodes exposing themselves in t MRv2 jobs, the probability P_{mat} can be obtained by

$$P_{mat} = 1 - P_{mct} = 1 - P_{mna}^t \quad (9)$$

Step 8. The aim of SHIYF is to find all the malicious nodes in YARN. Therefore, detection ratio D_{ratio} is obtained by the above derivations, then

$$\begin{aligned} D_{ratio} &= P_{mat} = 1 - P_{mct} = 1 - P_{mna}^t \\ &= 1 - (1 - P_{Nam} \times P)^t \\ &= 1 - \left\{ 1 - [1 - (1 - P_{am})^N] \times P \right\}^t \\ &= 1 - \left\{ 1 - [1 - (1 - 1/b)^{b \times E_r}] \times P \right\}^t \end{aligned} \quad (10)$$

3.2. Theoretical Results

We experimented on the effects of b , P , t , and E_r to the detection ratio D_{ratio} based on the theoretical arithmetic presented in Section 3.1.

Figure 8 shows the change of D_{ratio} against the execution ratio E_r and the number of the blocks b , where $t = 40$ and $P = 0.2$. D_{ratio} increases along with the increase of E_r and decreases slightly with the increase of b .

The change of D_{ratio} against the E_r and the malicious behavior probability P , where $b = 20$ and $t = 10$, are shown in Figure 9. Evidently, D_{ratio} increases along with the increase of P . Given a certain E_r , the presence of more malicious behaviors corresponded to the increased effectiveness of the operation of SHIYF.

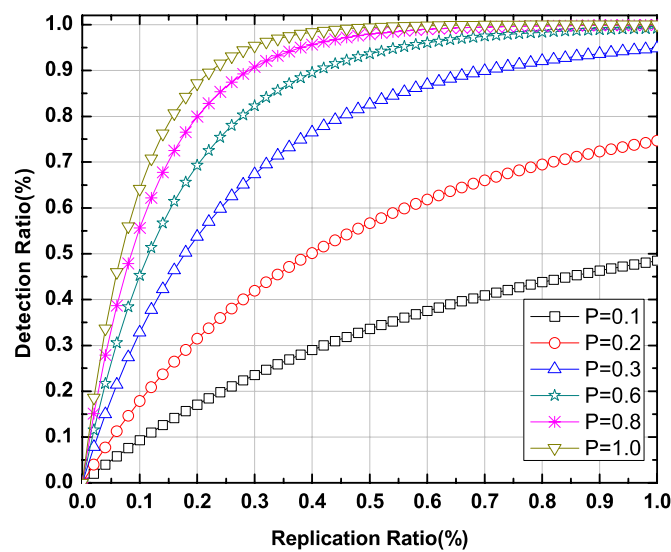


Figure 9. The effect of the the malicious action probability (P) on the Detection Ratio.

Figure 10 shows the change of D_{ratio} against the E_r and the number of jobs t , where $b = 20$ and $P = 0.2$. D_{ratio} increases along with the increase of E_r and t . If $t = 25$ and $E_r = 30\%$, then D_{ratio} is close to 90%.

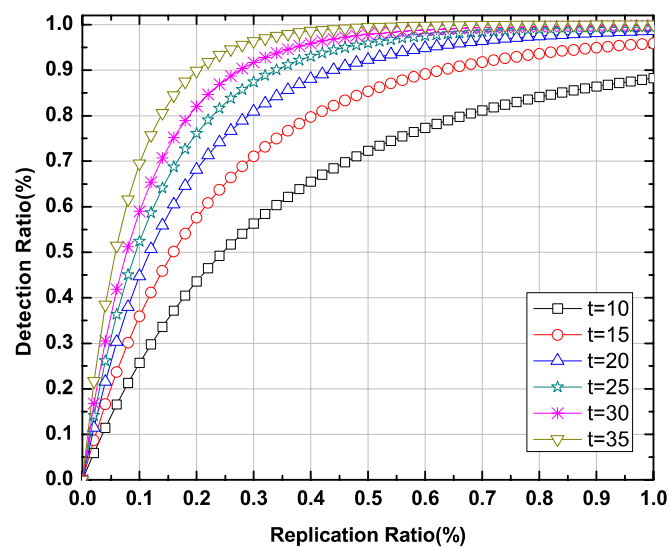


Figure 10. The effect of the number of jobs (t) on the Detection Ratio.

On the basis of the theoretical derivation results, we can draw the following conclusions:

1. The detection ratio D_{ratio} increased with the increase in the execution ratio E_r , the number of jobs t , and the malicious action probability P .
2. The number of blocks b had a minimal impact on D_{ratio} .
3. As long as the number of jobs t was equal or greater than 25, we could set E_r at a low level ($\leq 30\%$) to achieve a desired D_{ratio} ($\geq 85\%$) when $P \geq 0.2$. Moreover, the more P was, the better D_{ratio} was.
4. Furthermore, if we combined map speculative tasks and reduce speculative tasks together, then we could reasonably believe D_{ratio} would be more than 90%.

In conclusion, theoretical derivation indicates that SHIYF is effective for finding malicious behaviors.

4. SHIYF Experiments

In this section, we evaluate the security, integrity, and performance of SHIYF by conducting three benchmark experiments: WordCount, TestDFSIO, and MRBench.

We deployed the entire SHIYF cluster with an RM node and six NM nodes. The RM machine was equipped with one quad-core 3.9 GHz Intel Xeon E3-1280 V6 CPU, 16 GB memory, one Intel DC S3710 800 GB SSD, and 1000M NIC. Six NM machines were equipped with one quad-core 3.0 GHz Intel Core i5-7400 CPU, 8 GB memory, one 500 GB SATA II disk, and 1000M NIC. All machines had the same software configurations, including Ubuntu Server 16.04 LTS (64-bit), JDK 1.8.0, and Hadoop 2.8.0.

Considering the efficiency of the Hadoop cluster, the local data, and the objective of SHIYF experiments, we configured and optimized the Hadoop cluster first as follows:

- The file replication number of HDFS (dfs.replication) was set at 2, because the experiments were executed in a local rack. The minimum size of each file chunk was set at 256 MB to facilitate the processing of large files. To avoid a large number of data copies from the remote machines, the size of the split was set to equal the size of the block. A task disposes of a split.
- Given that six NM machines were equipped with one quad-core CPU, the value of “mapred.tasktracker.tasks.maximum” was set to 4. The number of reductions equaled $1.75 \times$ (the numbers of NMs \times mapred.tasktracker.tasks.maximum), namely, 42. Then, the faster NMs that finished their first round of reduce tasks would launch the second round of reduces immediately, thereby indicating a much improved load balancing.

In addition, we defined three experiment scenarios as follows:

1. In Hadoop, the speculative execution was open by default.
2. In SHIYF, the 30% Map and Reduce tasks were selected randomly to check the validity of results; thus, they will execute the speculative tasks and MD5 hash computations.
3. In SHIYF, two NMs will execute the malicious behaviors and return the wrong MD5 hashes at 20% probability, which is equivalent to the 7%–33.3% malicious nodes in the Hadoop cluster.

4.1. WordCount Benchmark

4.1.1. Execution Results of SHIYF

In the WordCount benchmark, we chose various test files and compared the time cost in three different scenarios. To calculate big data, all the test files were greater than 256 MB and met the minimum file block setting. The results were the average values of 25 WordCount experiments based on Section 3. The corresponding histogram is shown in Figure 11.

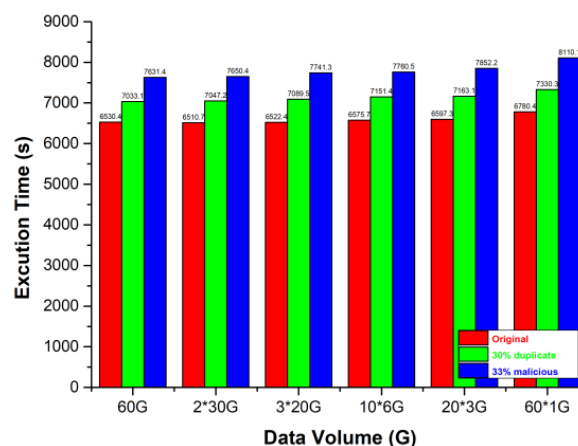


Figure 11. The execution time of WordCount.

We supposed that S_i was the size of one file and a was its numbers. We let N_b denote the number of blocks, then

$$N_b = \frac{\text{The total sizes of files}}{\text{mapred.min.split.size}} = \frac{\sum_{i=1}^n aS_i}{256M}, a = 1, 2, \dots, n. \tag{11}$$

Therefore, we could obtain three conclusions:

1. In the original YARN framework, although the input paths of “60 × 1 G” are 60 times that of “60 G,” the time cost increases slightly along with the increase of the total input paths when the numbers of blocks are equal to 240 according to Formula (11).
2. Without the malicious nodes, the time cost of WordCount increases by approximately 9% only in the SHIYF. A new speculative TaskAttempt is not equal to a new same task; therefore, the Job time does not increase by 30%. The extra time costs mainly come from the communication of the speculative TaskAttempts. By contrast, MD5 hash computing and comparing have little influence on SHIYF.
3. When two malicious NMs are given in SHIYF, the probability of Map/Reduce tasks assigned to them is close to 33.3% because of the load balancing of the Hadoop cluster. Furthermore, the probability of malicious behaviors is 20%. Therefore, the increasing time is mainly due to Task waiting for the returned values of extra speculative TaskAttempts. The increasing time cost of SHIYF is between 16% and 20% compared with that of the original condition.

4.1.2. Malicious Node Detection Ratio of SHIYF

In this section, we verify the malicious node detection ratio of SHIYF. In SHIYF, MRAppMaster will record time, Job_ID, the malicious node’s hostname, and right and wrong MD5 hashes in Job logs on HDFS.

All the test files were divided into 240 blocks, with the addition of 30% speculative executions; thus, every NM disposed 52 blocks in Map. Then, “hadoop2” and “hadoop5” were set to execute the malicious behaviors in Map and Reduce, in 20% probability amount, to approximately 22–30 times malicious actions. In 25 WordCount experiments, 20–30 malicious behavior records were found in the logs. The experiment results and the malicious node detection ratio computation are shown in Table 1.

Table 1. The detection ratio of SHIYF.

Reference Value	YARN (Original)	SHIYF (30% Duplicate)	SHIYF (33% Malicious)
Map (block numbers)	240	312	334–342
Reduce (block numbers)	240	314	337–345
Malicious behavior (times)	0	0	22–30
Malicious behavior records (times)	0	0	20–28
The malicious nodes (hostnames)	none	none	hadoop2, hadoop5
The potential malicious nodes (hostnames)	none	none	hadoop1, hadoop3
Detection ratio (%)	0	0	87%–93.3%

A failed task occurs because the three MD5 hashes returned by different TaskAttempts are inconsistent, and comparisons conducted twice are unsuccessful. Thus, Job launched the same Task again. Furthermore, all the hostnames, comparisons, and MD5 hashes in SHIYF are recorded, and we can obtain three conclusions as follows:

1. “Hadoop2” and “hadoop5” are the malicious nodes; “hadoop1” and “hadoop3” are the potential malicious ones.
2. The malicious node detection ratio of SHIYF is between 87% and 93.3%. This ratio is in line with the expected theoretical derivation shown in Figure 9 in Section 3. Therefore, “hadoop2”/“hadoop5”

are not the malicious NMs; they executed the malicious behaviors in their container tasks only once, and this instance was not chosen as among the verified malicious behaviors.

- On the basis of the conclusions of theoretical derivations in Section 3, the detection ratio increases with the increase of the execution ratio Er , the number of jobs t , and the malicious action probability P . Consequently, we believe that SHIYF will have the better malicious node detection ratio when it runs on a larger cluster and test data set.

4.1.3. Resource Utilization of SHIYF

In this section, we monitor the resource consumption of every machine on SHIYF, such as CPU utilization, memory utilization, disk throughput, and network throughput. With “10 × 6 G” taken as the example, Figures 12–14 reveal the resource consumptions of SHIYF on RM and NMs in three situations, respectively. NMs are divided into two types: one includes MRAppMaster and Containers, and the other includes Containers only.

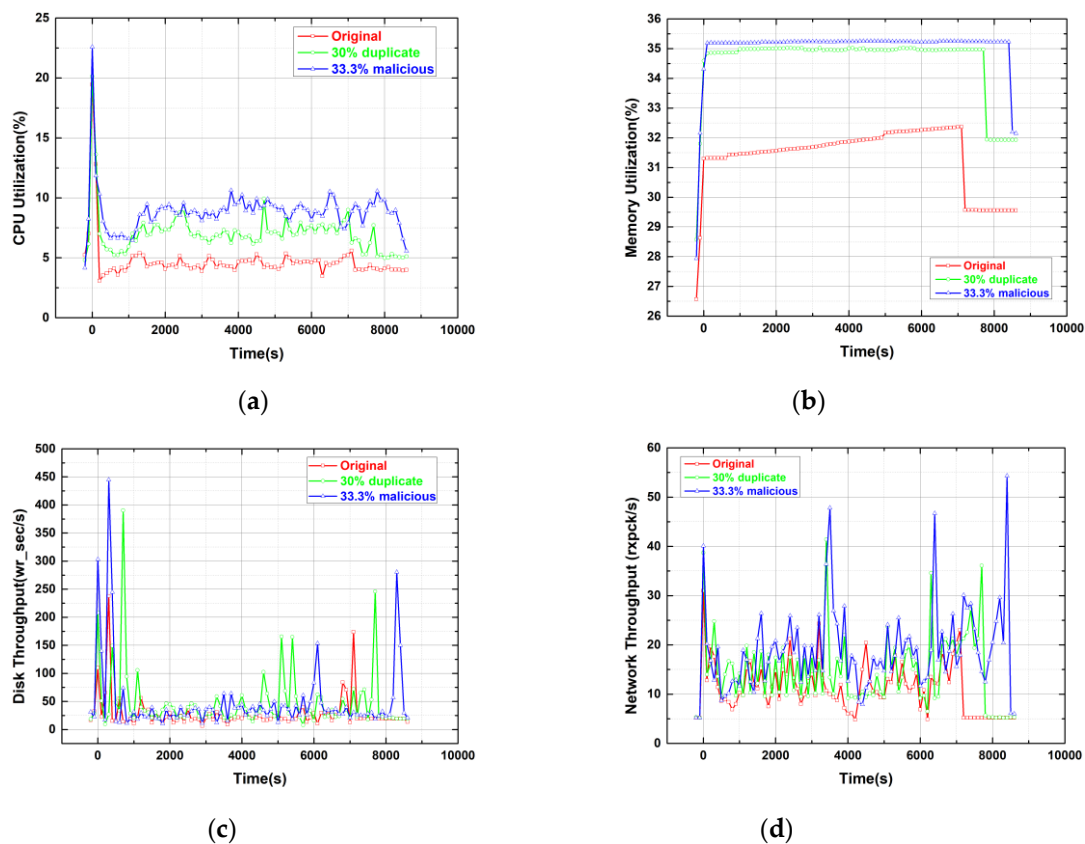


Figure 12. Resource utilization of ResourceManager: (a) Central Processing Unit (CPU) utilization; (b) memory utilization; (c) disk throughput; and (d) network throughput.

ResourceManager

In RM, ApplicationManager launches one MRAppMaster to control the Job and Scheduler that is responsible for the communication with the NMs.

We can obtain the following conclusions from the analysis of Figure 12.

- The addition of 30% extra speculative executions and executing MD5 hash computations and comparisons have a weak influence on RM. Figure 12a shows that the CPU utilization of RM in the WordCount experiment is relatively low except for the initial stage.
- Adding 30% speculative tasks and 33.3% malicious tasks merely increases a few status monitors to NMs and information communications between RM and NMs; memory utilization remains

lower than 36%. Moreover, the memory utilization of RM is markedly smooth, as shown in Figure 12b.

- Several reference variables are recorded to show the disk influence of SHIYF on RM, including the number of transfers per second “tps,” sectors read/written per second “rd_sec/wr_sec,” the average size (in sectors) of the requests that were issued to the device “avgrq-sz,” the average queue length of the requests that were issued to the device “avgqu-sz,” and so on. We take the most representative parameter “wr_sec/s” as an example. Figure 12c shows that adding 30% speculative tasks and MD5 comparisons has a weak influence on the disk throughput of RM. The primary influences are found in the initial and final phases because more statuses of NMs are transmitted to RM; thus, SHIYF evidently increases the hard disk writing of RM.
- Total number of packets received per second “rxpck/s,” total number of packets sent per second “txpck/s,” and data size received per second “rxkB/s,” among others, are recorded for monitoring the network throughput. Taking “rxpck/s” as an example, Figure 12d shows that adding 30% speculative tasks and 33.3% malicious nodes has a minimal influence on the network throughput of RM. Only repeated computing and comparison of MD5 hashes in SHIYF increase some resource applications and NM status reports.

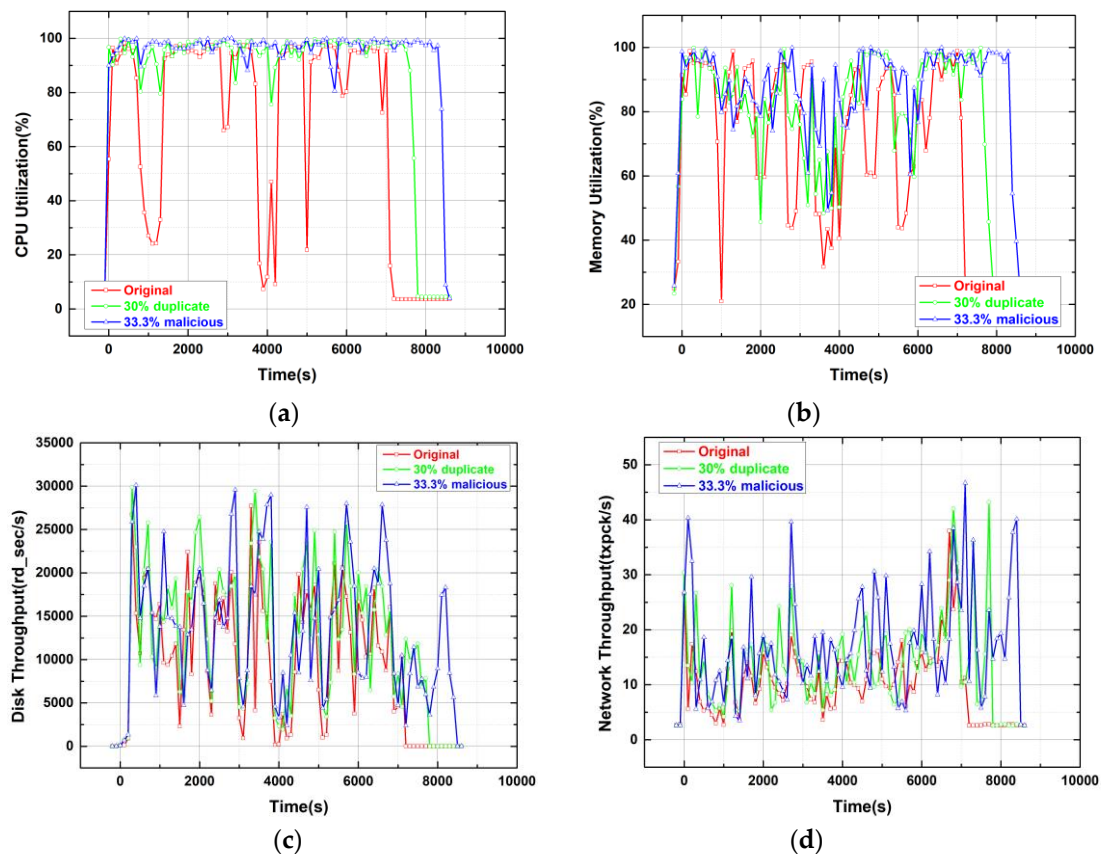


Figure 13. Resource utilization of NodeManager (MRAppMaster): (a) CPU utilization; (b) memory utilization; (c) disk throughput; and (d) network throughput.

NodeManager: NM(MRAppMaster)

NMs are divided into two categories; the first includes MRAppMaster and Containers, moreover, the second includes only Containers. Figure 13 reveals the resource utilization of NM (MRAppMaster). We can obtain some conclusions as follows from the analysis of Figure 13.

- The CPU utilization of NM (MRAppMaster) is shown in Figure 13a. In SHIYF, the lowest CPU occupancy is more than 80%; moreover, the time consumption of Job is longer than that in the

original YARN. However, their increases are under 20%, and a lower CPU utilization will occur if SHIYF is built on more powerful clusters.

2. In three conditions the memory utilization of NM (MRAppMaster) is only slightly different, as shown in Figure 13b.
3. Some reference parameters are recorded, such as “tps (The number of I/O per second from the physical disk),” “rd_sec/wr_sec (The number of sectors read/write from the device per second),” “avgqu-sz (Average IO request queue length waiting to be processed),” “util% (what percentage of a second is devoted to I/O operations)” etc. for manifesting the influence of SHIYF on the disk of NM (MRAppMaster). The number of sectors read from the device per second (rd_sec/s) is the most representative one. Nevertheless, the average disk reading speeds are close in three conditions, as shown in Figure 13c. The slight increase occurred because NM (MRAppMaster) launched the extra speculative tasks to compute and compare MD5 hashes.
4. The total number of packets transmitted per second (txpck/s) indicates the influence of SHIYF on the network throughput of NM (MRAppMaster), as shown in Figure 13d. SHIYF increases some network communications of NM (MRAppMaster) with RM and other NMs, while adding 30% speculative tasks and 33.3% malicious nodes, because NM (MRAppMaster) must report more node statuses to RM and communicate with more containers. However, the extra overhead is affordable.

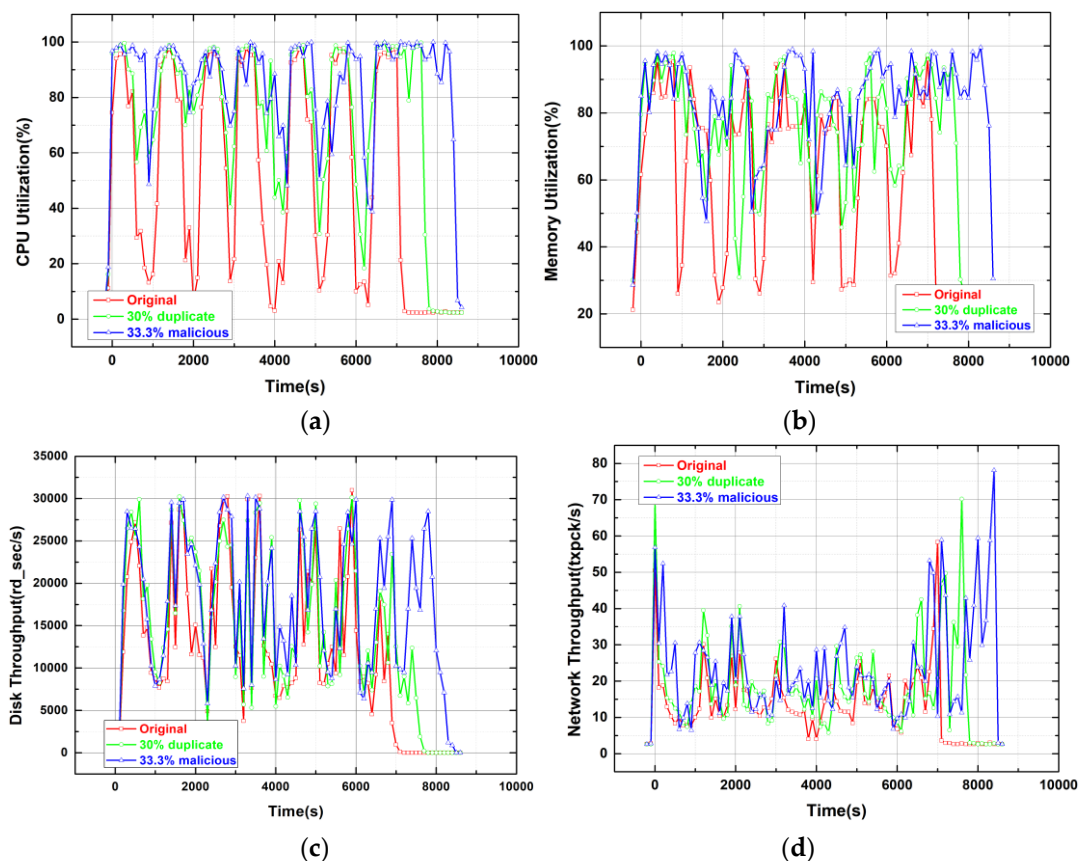


Figure 14. Resource utilization of NodeManager (Containers): (a) CPU utilization; (b) memory utilization; (c) disk throughput; and (d) network throughput.

NodeManager: NM (Containers)

Figure 14 shows the resource use of another class of NMs in SHIYF.

We can obtain the following conclusions from the analysis of Figure 14:

1. The CPU utilization of NM (Containers) is shown in Figure 14a. Compared with the CPU utilization shown in Figure 13a, it is lower than that of NM (MRAppMaster) in three conditions because NM (MRAppMaster) needs to manage NM, MRAppMaster, and all other containers in the job.
2. The memory utilization of NM (Containers) is also lower than that of NM (MRAppMaster), as shown in Figure 14). Both Figures 13b and 14b show that SHIYF has little effect on the memory utilization of NMs.
3. Figure 14c shows the number of sectors read from the device per second (rd_sec/s) in NM (Containers). Compared with Figure 13c, the disk throughput of NM (Containers) peaks earlier than that of NM (MRAppMaster), and the average throughput is higher. This situation shows that the machine on which MRAppMaster is run allocated fewer containers for dynamic load balancing in the Hadoop cluster. However, the effect of SHIYF is weak in three conditions.
4. A comparison of Figures 13d and 14d shows that NM (Containers) also needs to report more node statuses to RM and communicate more with NM (MRAppMaster) in SHIYF. By contrast, the resource consumption of NM (Containers) is lower than that of NM (MRAppMaster). Moreover, their overhead is affordable.

Finally, we can draw three conclusions from the WordCount benchmark.

1. SHIYF can locate the malicious nodes and the potential malicious ones. The malicious node detection ratio is between 87% and 93.3%. It is in line with the expected theoretical derivation.
2. The increasing time cost of SHIYF is between 16% and 20%. Moreover, it has little effect on increasing the resource overhead.
3. The limited computing ability of the experiment hardware may increase the time cost and resource consumption. We trust that SHIYF will perform better as it executes a much larger range of jobs in a more powerful Hadoop cluster. If so, SHIYF can use a lower speculative execution ratio to achieve high malicious node detection ratios.

4.2. TestDFSIO Benchmark

4.2.1. Execution Results of SHIYF

In this section, we use TestDFSIO to test the read-and-write file system performance of SHIYF. The intermediate results of TestDFSIO, including “tasks,” “size,” “time,” “rate,” and “sqrate,” will result in inconformity of MD5 hashes. Considering this particularity, we should configure SHIYF based on three different scenarios.

- In the original condition, we test the performance of the read-and-write file system of YARN without any modification.
- In the SHIYF (30% duplicate) condition, we abolish the MD5 comparison of SHIYF temporarily because MD5 is a simple and efficient digital digest algorithm, and no malicious nodes occur in this condition. Moreover, SHIYF has little impact on the total job execution time, as verified in Section 4.1.
- In the SHIYF (33.3% malicious) condition, every task chosen for checking needs to compute and compare the MD5 hashes of the intermediate or final results. Moreover, every result of TaskAttempt is different. Thus, we keep only the “tasks” and “size” as the Map/Reduce results to ensure that the MD5 hashes of the same TaskAttempts that ran in the secure NMs are equal.

In addition, TestDFSIO launches a MapReduce job to read or write files. The same amount of data is written into or read from HDFS, and four statistics are collected: throughput (mb/sec), average I/O rate (mb/sec), I/O rate std deviation, and test exec time (sec). We executed TestDFSIO 25 times in three conditions. The average values of the experiments are shown in Table 2.

Table 2. The results of TestDFSIO.

TestDFSIO	Statistic	1×60 G	10×6 G	20×3 G	60×1 G
WRITE (ORIGINAL)	Throughput (mb/sec)	3.6532858730067091	4.1495804701173095	4.5426758165430426	5.0352236239847136
	Average I/O * rate(mb/sec)	3.8734657834132454	4.3471371178521935	4.7294176523151477	5.1312346348895038
	I/O rate std deviation	0.430984357634622	0.951427385613907	0.615915048946195	0.445020674520458
	Test exec time(sec)	2198.111	1561.544	1673.816	1890.473
READ (ORIGINAL)	Throughput (mb/sec)	12.970596932628902	18.046781056091405	17.784562337941026	15.858087512060852
	Average I/O rate(mb/sec)	13.012415248157041	18.178287573625446	17.792148534176382	15.885178120960451
	I/O rate std deviation	1.498127648219716	1.246210558435681	0.485214225977834	0.781471278556941
	Test exec time (sec)	660.835	474.956	481.959	540.508
WRITE(30% SPECULATIVE)	Throughput (mb/sec)	4.1869146712609413	4.8050487636647159	5.1450188934706481	5.5417910116501725
	Average I/O rate(mb/sec)	4.6151871263970814	4.7173163452271386	5.3504192374693026	5.9544018520531907
	I/O rate std deviation	1.657201551021365	1.711504547126103	0.935136452047035	0.753113208091526
	Test exec time (sec)	2371.649	1662.184	1764.051	1970.018
READ(30% SPECULATIVE)	Throughput (mb/sec)	12.457052964837827	17.893025372274712	17.039617160846092	14.901470746125078
	Average I/O rate(mb/sec)	12.568542145289061	17.932455253601074	17.809027862581273	15.076180259014069
	I/O rate std deviation	1.0519670225048039	0.5843634163160046	1.6418415710048093	0.8952104835410775
	Test exec time(sec)	688.078	479.037	506.295	578.209
WRITE(33.3% MALICIOUS)	Throughput (mb/sec)	5.272581215418207	5.6601450830043722	6.0414721722833548	6.4590725064507841
	Average I/O rate(mb/sec)	5.3641939105048191	5.9178047153028364	6.4681551987331872	7.0720323180910965
	I/O rate std deviation	0.505720201569015	1.450121028363904	1.256178142824583	2.045873016820649
	Test exec time (sec)	2549.196	1768.511	1836.149	2104.951
READ(33.3% MALICIOUS)	Throughput (mb/sec)	11.990184970451683	17.0845710814059107	16.193211574396384	14.006810615806931
	Average I/O rate(mb/sec)	12.005265249203364	17.83786672858012	16.350187485045837	14.75482619974933
	I/O rate std deviation	0.872031602505907	2.010804105194108	0.949113918481016	2.06249176210582
	Test exec time (sec)	716.870	503.706	530.322	610.947

*I/O: Input/Output.

We can draw three conclusions.

1. In the three conditions, the read speed is much faster than the write speed. In the beginning, with the increase in file size, the running time decreased, thereby indicating that HDFS was highly suitable for processing large-scale reading and writing data. However, a corresponding increase in average running time occurred along with the increase in file sizes because of the inevitable increase in the number of cluster nodes, the complicated hardware configurations, and other reasons.
2. Increasing speculative executions by 30% corresponds to an increased 30% TaskAttempts. However, the speculative executions are launched with the original TaskAttempts simultaneously; thus, the time consumption increase is under 8%, as shown in Table 3. This finding is mainly because of the inconsistency of the TaskAttempt completion times, regardless of the “WRITE” test or in the “READ” test.
3. In theory, adding 33.3% malicious nodes executed malicious behaviors at 20% probability is equivalent to an increase of approximately 6.66% extra speculative executions. The time increase occurred primarily because of the waiting time for the second speculative execution. Moreover, the increase in the total time cost does not exceed by 16%, unlike with the original YARN.

Table 3. Temporal growth rate (test exec time).

Scenarios	Temporal Growth Rate (Read)	Temporal Growth Rate (Write)
YARN (original)	0	0
SHIYF (30% duplicate)	6.981%	7.895%
SHIYF (33% malicious)	13.031%	15.094%

4.2.2. Influence of SHIYF to Network Throughput

Considering location optimization in HDFS, most data are read from the local disk rather than the network with limited bandwidth. Therefore, the read speed is faster than the write speed. The influence of SHIYF on network throughput was computed based on the write throughput (mb/sec) in TestDFSIO as follows:

Step 1. We assume N_t is the total node number in the SHIYF cluster. D_f is the total sizes of the test files (M). We let T_s and T_{hr} represent the test execution time (sec) and throughput (mb/sec), respectively. We derive T_{hr} as

$$T_{hr} = \frac{D_f}{T_s \times N_t} \quad (12)$$

Step 2. We suppose N_f is the number of files. Moreover, each concurrent process conducts one file in MRv2. We let the number of concurrent processes be N_p , then

$$N_p = N_f \quad (13)$$

Step 3. We let N_r denote the “dfs.replication,” then $N_r = 2$. Thus, $(N_r - 1) = 1$ network transmissions occur as one file is writing on HDFS. We assume that N_{wpm} is the total number of write processes in every NM. It is computed as

$$N_{wpm} = \frac{N_p}{N_r - 1} \quad (14)$$

Step 4. We can obtain the formula of network throughput NT_{hr}

$$\begin{aligned}
 NT_{hr} &= T_{hr} \times (N_r - 1) \times N_{wpm} \\
 &= \frac{D_f}{T_s \times N_t} \times (N_r - 1) \times \frac{N_p}{N_t - 1} \\
 &= 1 \times \frac{D_f}{T_s \times N_t} \times \frac{N_p}{N_t - 1}
 \end{aligned}
 \tag{15}$$

Combined with Table 3, we can calculate the network throughput in three conditions, as shown in Table 4.

Table 4. Influence of SHIYF to network throughput.

Network Throughput (mb/sec)	1 × 60 G	10 × 6 G	20 × 3 G	60 × 1 G
YARN (original)	0.609	6.916	30.285	50.352
SHIYF (30% duplicate)	0.698	8.008	34.300	55.417
SHIYF (33% malicious)	0.879	9.434	40.276	64.591

Two conclusions can be drawn as follows:

1. More files written on HDFS correspond to more copied files transmitted on the network. Therefore, the network throughput is higher.
2. The highest network throughput is 64.591 M/s, thereby indicating that the highest growth rate of network throughput is 28.28%. However, this value is far below the bandwidth of a gigabit network (128 M/s). Thus, no significant bandwidth load occurs. Instead, the process improves network bandwidth utilization.

4.2.3. Influence of SHIYF to HDFS

To examine the influence of SHIYF on HDFS, we chose “1 × 60 G” as the example due to its execution time being the longest in TestDFSIO benchmark. The number of sectors read from/written to NM (Containers) per second (rd_sec/s, wr_sec/s) can intuitively demonstrate the influence of SHIYF on HDFS, as shown in Figure 15.

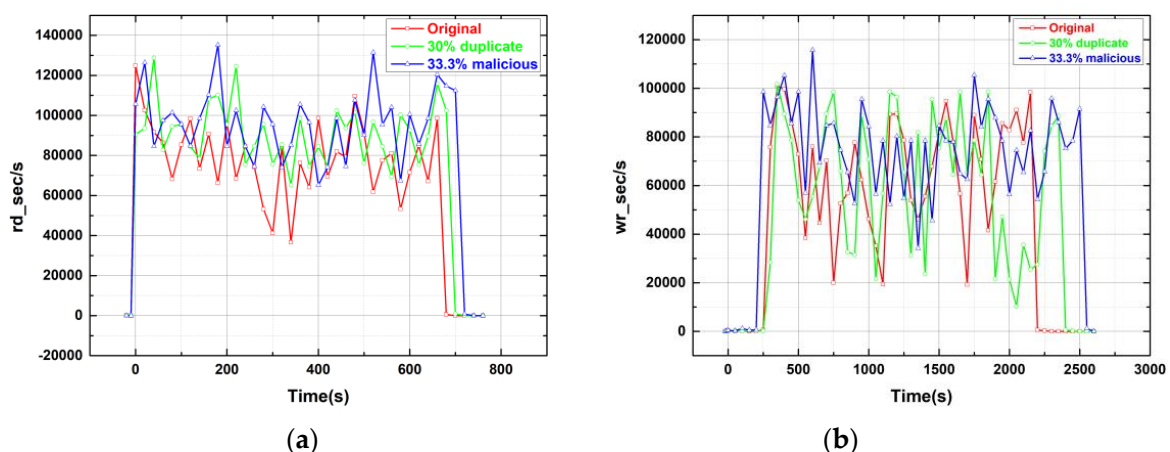


Figure 15. Influence of SHIYF to Hadoop Distributed File System (HDFS): (a) The number of sectors read from the device per second (rd_sec/s); (b) The number of sectors write from the device per second (wr_sec/s).

We can obtain three conclusions as follows.

1. More speculative executions correspond to more data read from or written to HDFS. However, the changes in the curves in the three conditions were minimal; moreover, they interlaced and partially overlapped.
2. Although SHIYF improves the use and efficiency of the disk, it does not increase the hard disk load. The minimal change follows the ideal states in three conditions.
3. SHIYF impacts the time of TestDFSIO. However, it has no effect on the read and write performance of HDFS.

In this section, we test SHIYF in the TestDFSIO benchmark and show the influence of SHIYF on network throughput and HDFS. Although SHIYF strengthens the security and integrity of YARN using the speculative executions and MD5 algorithm, it can also maintain the Input/Output (I/O) performance of HDFS. The slight growth of network throughput and time mainly results from the increasing speculative executions and the extra waiting time. Moreover, the overhead is affordable.

4.3. MRBench Benchmark

4.3.1. Execution Results of SHIYF

MRBench repeats a minor job many times, as specified by the user, to check whether the minor job running on a Hadoop cluster is repeatable and efficient. MRBench is used to test the performance to handle many minor jobs, and it has the security protection ability of SHIYF. Therefore, the times of job repetition t are set as 10, 15, 20, 25, 30, and 40, based on Section 3. Several parameters should be set as follows:

- inputLines = 1000. The number of every generated file is 1000 lines.
- maps = 200. 200 maps are used for each run.
- reduces = 100. The number of reduces for each run is 100.
- numRuns = 10, 15, 20, 25, 30, and 40.

The experiment results in three scenarios are shown in Figure 16. We obtain three conclusions.

1. In these three conditions, every experiment with the same configuration is executed with different repetition times. The execution time in Figure 16 is an average value of SHIYF that conducted the same job several times. More repetition times correspond to increased accuracy of the execution time.
2. Adding 30% speculative executions makes the MRBench time increase by approximately 9%, mainly due to the inconsistent completion time of TaskAttempts. Moreover, this process increases MD5 hash computations and comparisons.
3. In the 33.3% malicious nodes condition, execution time increases by approximately 16% because of the extra speculative TaskAttempts and the inconformity of two comparative MD5 hashes.

4.3.2. Malicious Node Location of SHIYF

MRBench is also used to test the security protection ability of SHIYF in locating malicious nodes. We chose the experiment “ $t = 25$ ” as the example following Section 4.3.1. The parameters “maps = 200” and “reduces = 100” decide that $200 + 100 = 300$ tasks are used in every MRBench benchmark. Hadoop2 and hadoop5 are two malicious nodes in the Hadoop cluster, and their probability of exhibiting malicious behaviors is 20%. Therefore, the upper limit of malicious tasks executed by hadoop2/hadoop5 is approximately $300/6 \times 20\% = 10$ times in MRBench. When MRBench is executed successfully and has achieved the goal of locating the malicious nodes, the upper limit of CA that withdraws all resource applications of the malicious nodes must be altered to 15(>10) rather than 5 in Section 2.2.1.

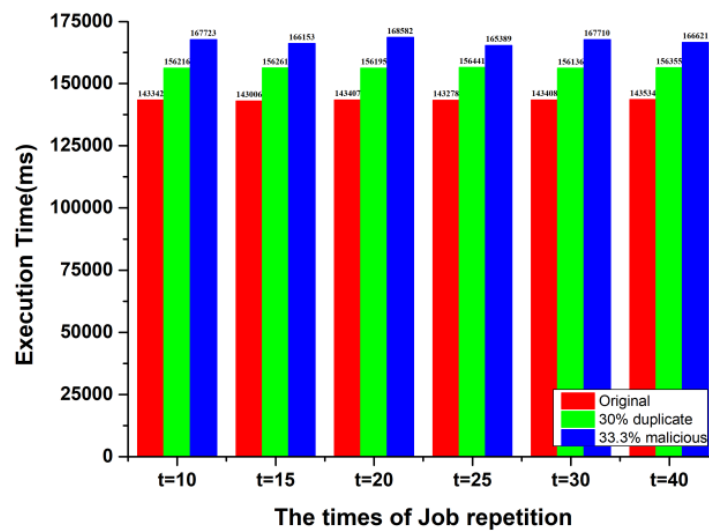


Figure 16. The results of MRbench Test.

After 25 MRBench experiments were performed, we check the logs of Job and compute the average values, as shown in Table 5. We obtain four conclusions.

1. Any malicious action record about hadoop1 is found in Job logs. Thus, it is a secure NM.
2. The average value of hadoop3/hadoop6 records is between 0 and 1, mainly because two continuous failed MD5 hash verification records would be recorded in 25 experiments. Therefore, they might be the potential malicious NMs. Although they were the secure NMs, they were considered the potential malicious ones if they validated the result as the malicious NMs at the same time and the results were inconsistent.
3. The average value of hadoop2/hadoop5 malicious behaviors is 9. We can judge them as the malicious NMs in the Hadoop cluster. Therefore, the malicious node detection ratio of SHIYF is at least 90% in the MRBench benchmark.
4. Not only can SHIYF achieve a high malicious node detection ratio, but it can also locate the malicious nodes and the potential ones accurately.

Table 5. The log records in MRbench.

Reference Value	hadoop2	hadoop5	hadoop1	hadoop3	hadoop6
Malicious action times	≤10	≤10	0	0	0
Record times	9	9	0	0–1	0–1
Malicious node or Potential Malicious node	Malicious node	Malicious node	None	Potential Malicious node	Potential Malicious node

5. Conclusions and Future Work

SHIYF is proposed in this paper. Through theoretical derivation, we set the relevant parameters of SHIYF accurately and implemented the prototype framework SHIYF based on Hadoop 2.8.0. The framework advantage of speculative execution and MD5 hash verification is that they ensure the integrity and validity of MapReduce 2.0 results. Moreover, SHIYF is able to locate the malicious and potentially malicious nodes in the Hadoop cluster.

Three experiments on SHIYF adequately demonstrate its malicious node detection D_{ratio} , and resource consumption can achieve the expected goals. In particular, D_{ratio} is at least 87%, while the overhead is increased only slightly. Therefore, the proposed SHIYF will use the lower speculative execution ratio and consumes less resources to achieve a desirable D_{ratio} as long as it runs on a more powerful machine cluster and disposes of more jobs.

However, adding 30% speculative tasks in SHIYF is still a few wasted resources. We will work hard to reduce the ratio of speculative tasks and improve D_{ratio} . Using 15% speculative execution ratio to achieve more than 95% D_{ratio} is a much better tradeoff between resource usage and security. Meanwhile the efficiency of SHIYF will also be promoted. In addition, non-collusive malicious nodes are found in the experiment environment. If several collusive attackers are found in the Hadoop cluster, then they might return the same wrong MD5 hashes when they are incorrectly considered the secure nodes. Therefore, our future research will focus on improving the tradeoff between performance and security in SHIYF, moreover, preventing collusive malicious nodes.

Author Contributions: Conceptualization, J.D. and J.W.; methodology, J.D.; software, J.D.; validation, Y.L. and J.W.; formal analysis, J.D.; investigation, J.D. and S.L.; resources, Y.L.; data curation, J.D. and J.W.; writing—original draft preparation, J.D.; writing—review and editing, J.D. and S.L.; supervision, Y.L. and J.W.; project administration, Y.L.

Funding: This research was funded by the National Nature Science Foundation [61872158, 61572229, U1564211, 6171101066 and 61872158]; Jilin Provincial Science and Technology Development Foundation [20170204074GX, 20180201068GX]; Jilin Provincial International Cooperation Foundation [20180414015GH] and z under Grant [NGII20170413].

Acknowledgments: The authors would like to thank the editors and the reviewers for their valuable comments that helped to improve the quality of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The following abbreviations are used in this paper:

YARN	Yet Another Resource Negotiator
MRv2	MapReduce 2.0
HDFS	Hadoop Distributed File System
SHIYF	secure and high-integrity YARN framework
CC	cloud computing
RM	ResourceManager
NM	NodeManager
FSM	finite-state machine
CA	ContainerAllocator
CPU	Central Processing Unit
I/O	Input/Output

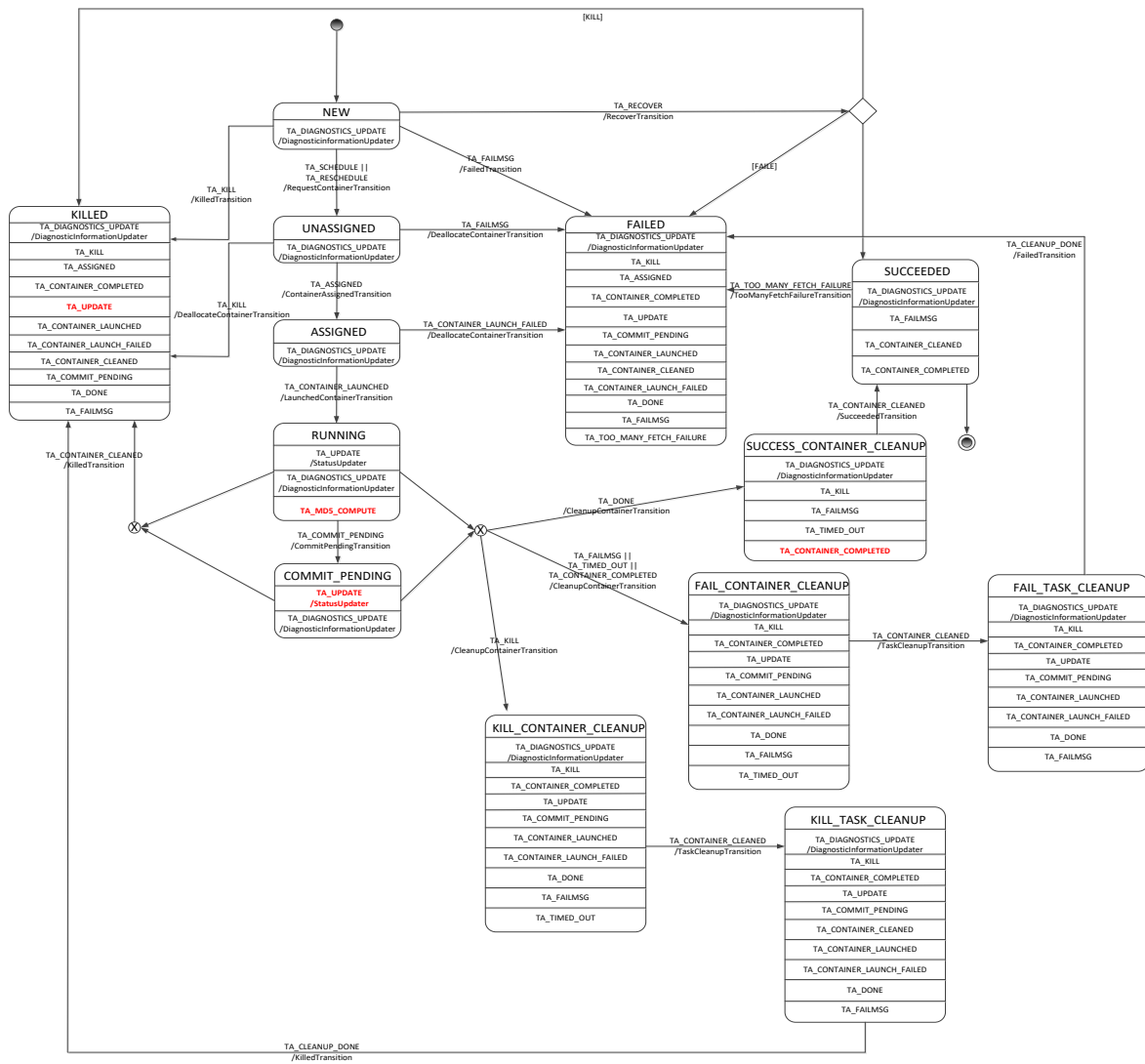


Figure A1. The finite-state machine of SHIYF TaskAttempt.

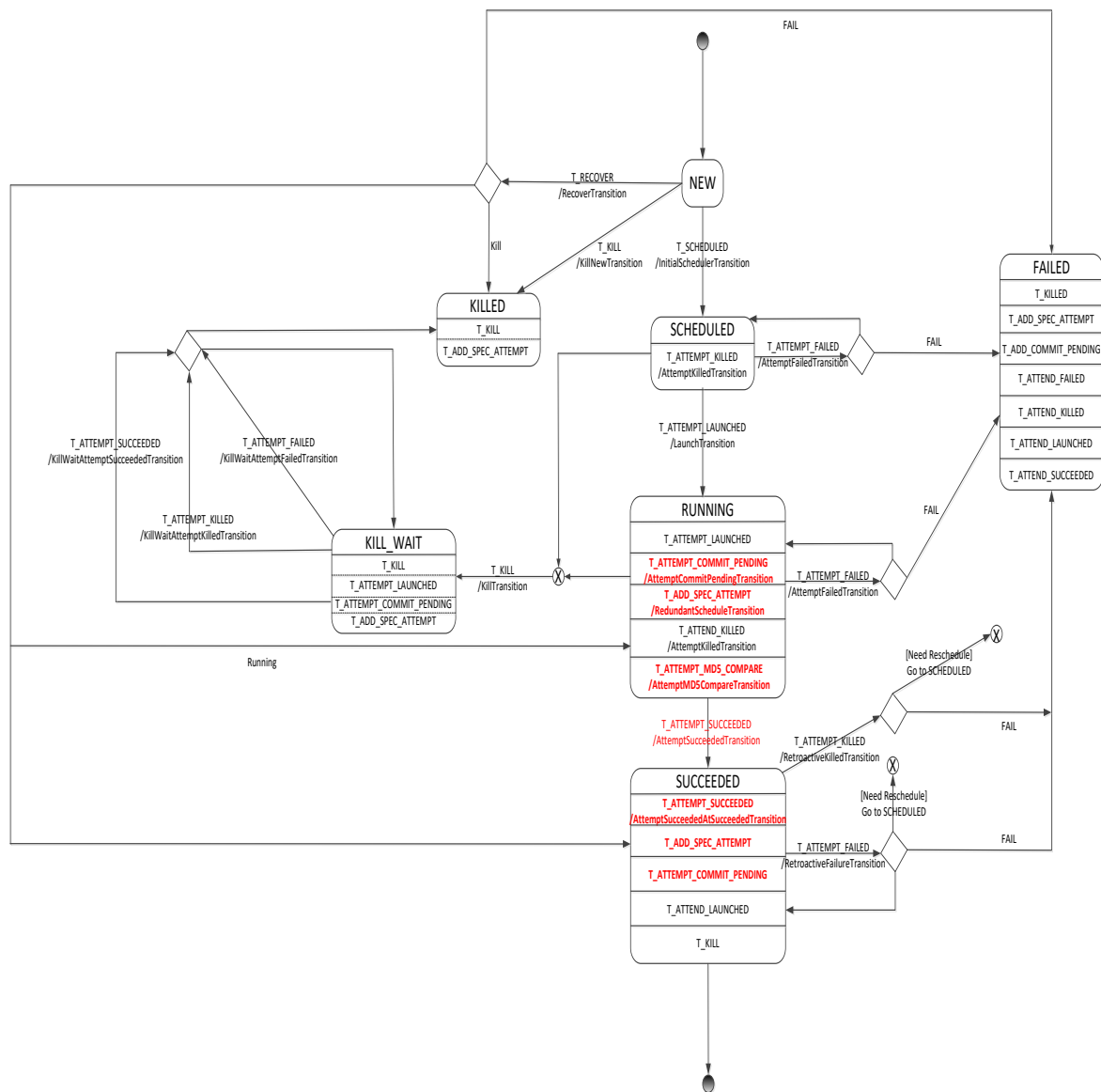


Figure A2. The finite-state machine of SHIYF Task.

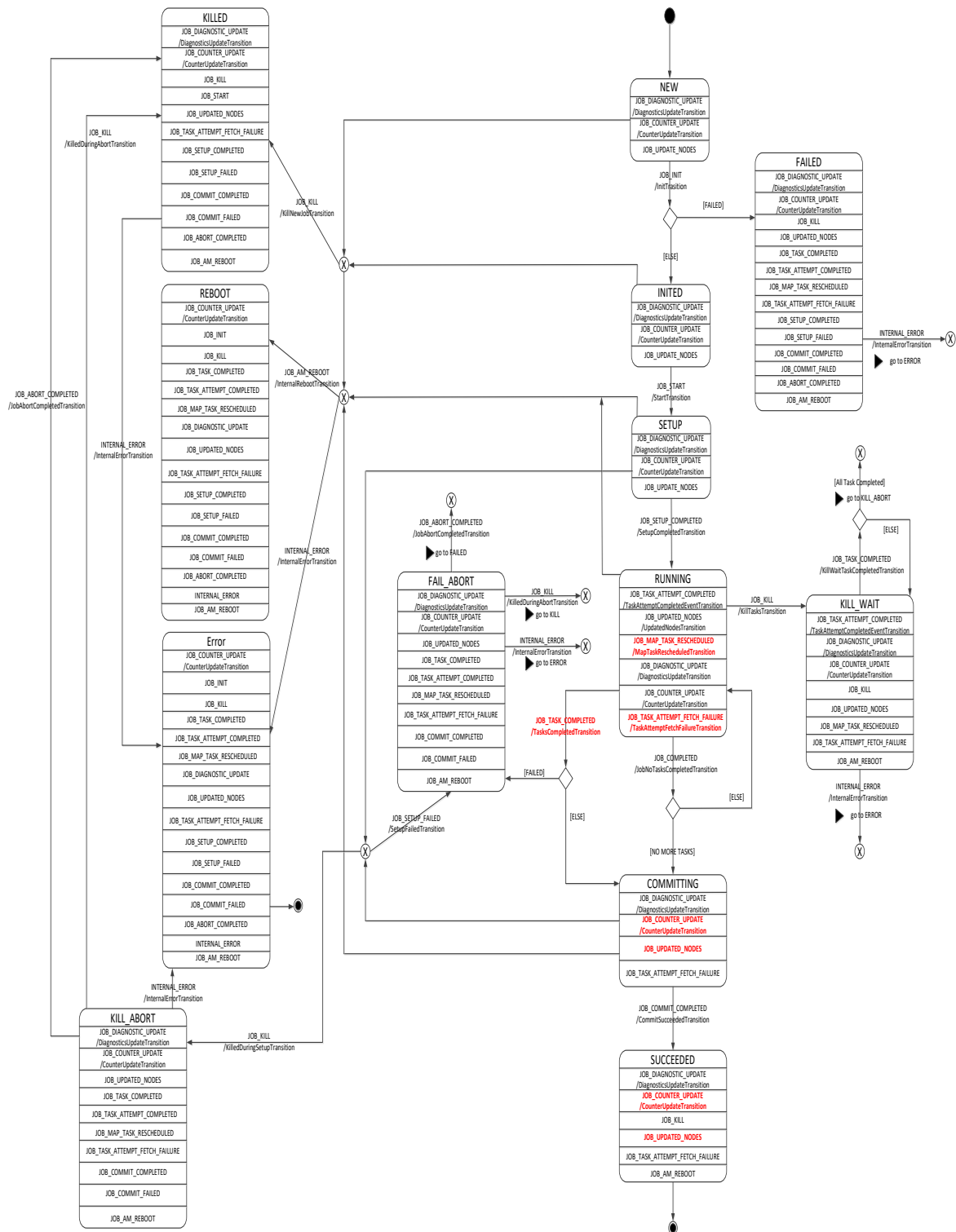


Figure A3. The finite-state machine of SHIYF Job.

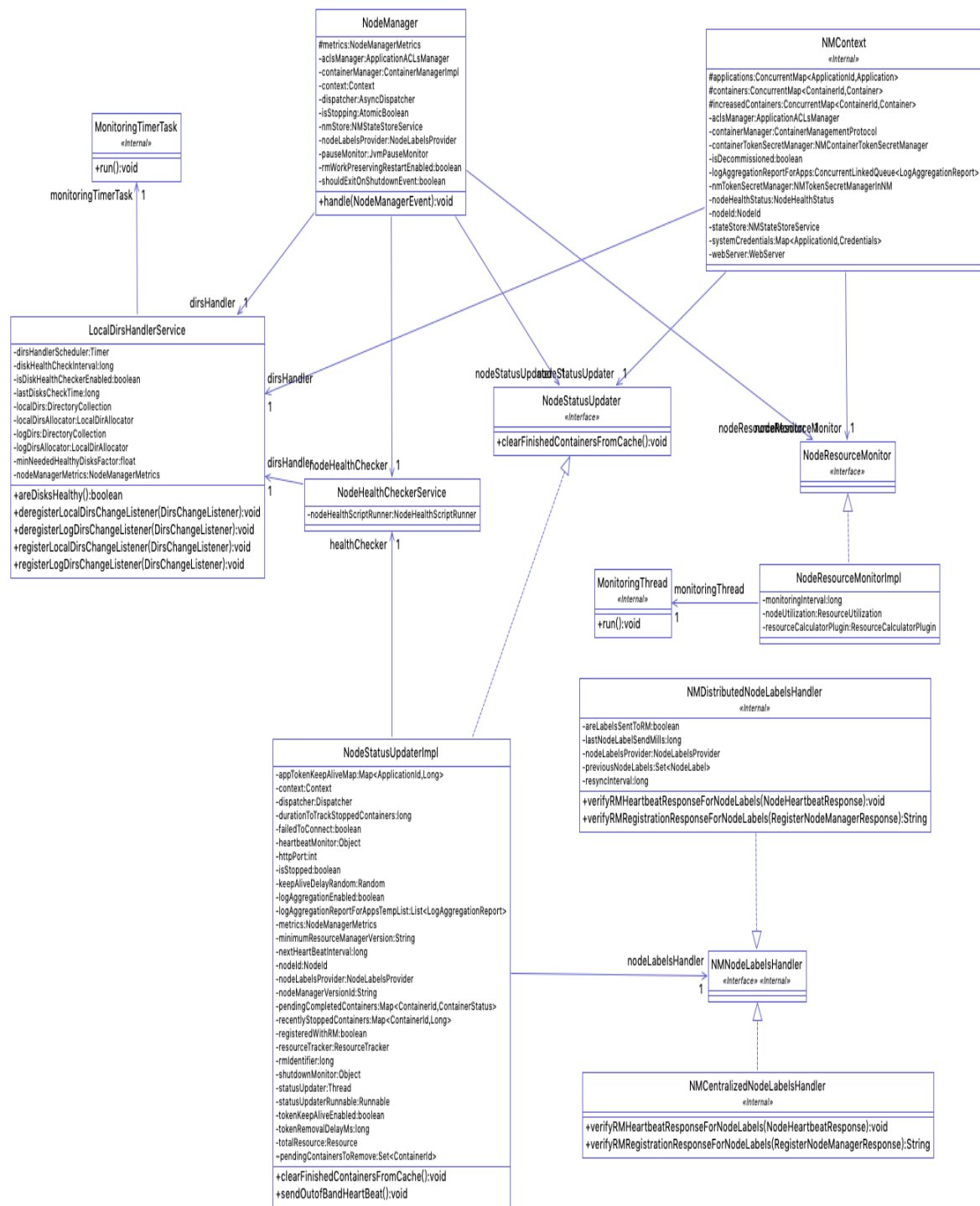


Figure A4. SHIYF NodeManager implementation.



Figure A5. SHIYF ResourceManager implementation.

References

- Hayes, B. Cloud computing. *Commun. ACM* **2008**, *51*, 9–11. [CrossRef]
- Dempsey, D.; Kelliher, F. Cloud Computing. 2018. Available online: https://link.springer.com/chapter/10.1007/978-3-319-63994-9_2 (accessed on 17 April 2019).
- Hashizume, K.; Rosado, D.G.; Fernández-Medina, E.; Fernandez, E.B. An analysis of security issues for cloud computing. *J. Internet Serv. Appl.* **2013**, *4*, 5. [CrossRef]
- Duncan, A.; Creese, S.; Goldsmith, M. An overview of insider attacks in cloud computing. *Concurr. Comput. Pract. Exp.* **2015**, *27*, 2964–2981. [CrossRef]
- Waqar, A.; Raza, A.; Abbas, H.; Khan, M.K. A framework for preservation of cloud users’ data privacy using dynamic reconstruction of metadata. *J. Netw. Comput. Appl.* **2013**, *36*, 235–248. [CrossRef]
- Lombardi, F.; Pietro, R.D. Secure virtualization for cloud computing. *J. Netw. Comput. Appl.* **2011**, *34*, 1113–1122. [CrossRef]
- Brodkin, J. Gartner: Seven cloud-computing security risks. In Proceedings of the Infoworld, Framingham, MA, USA, 2 July 2008.
- Grobauer, B.; Walloschek, T.; Stocker, E. Understanding cloud computing vulnerabilities. *IEEE Secur. Priv.* **2011**, *9*, 50–57. [CrossRef]

9. Jansen, W.; Grance, T. *Guidelines on Security and Privacy in Public Cloud Computing*; NIST Special Publication; U.S. Department of Commerce: Washington, DC, USA, 2011; Volume 800, p. 144.
10. Alliance, C.S. Security Guidance for Critical Areas of Cloud Computing Version 3.0. Available online: <https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf> (accessed on 17 April 2019).
11. White, T. *Hadoop: The Definitive Guide*[M], 3rd ed.; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2011.
12. Lam, C. *Hadoop in Action*; Manning Publications Co.: Newton, MA, USA, 2011.
13. Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. In Proceedings of the Sixth Symposium on Operating System Design and Implementation (OSD2004), USENIX Association, San Francisco, CA, USA, 6–8 December 2004.
14. Jiang, D.; Ooi, B.C.; Shi, L.; Wu, S. The performance of mapreduce: An indepth study. *Proc. Vldb Endow.* **2010**, *3*, 472–483. [CrossRef]
15. Chen, Y.; Ganapathi, A.; Griffith, R.; Katz, R.H. The Case for Evaluating MapReduce Performance Using Workload Suites. *Mascots* **2011**, 390–399. [CrossRef]
16. Vernica, R.; Carey, M.J.; Li, C. Efficient parallel set-similarity joins using MapReduce. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, Indianapolis, IN, USA, 6–10 June 2010.
17. Roy, I.; Setty ST, V.; Kilzer, A.; Shmatikov, V.; Witchel, E. Airavat: Security and Privacy for MapReduce. In Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, San Jose, CA, USA, 28–30 April 2010.
18. Dang Vo-Huu, T.; Erik-Oliver, B.; Guevara, N. EPiC: Efficient privacy-preserving counting for MapReduce. *Computing* **2018**. [CrossRef]
19. O'Malley, O.; Zhang, K.; Radia, S.; Marti, R.; Harrell, C. Hadoop Security Design, 2009. Available online: <http://carfield.com.hk:8080/document/distributed/hadoop-security-design.pdf> (accessed on 17 April 2019).
20. Foundation, T.A.S. Service Level Authorization Guide. 2013. Available online: https://hadoop.apache.org/docs/r1.2.1/service_level_auth.html (accessed on 9 July 2018).
21. Das, D.; Malley, O.; Radia, S.; Zhang, K. *Adding Security to Apache Hadoop*; Hortonworks Technical Report.
22. Vavilapalli, V.K.; Murthy, A.C.; Douglas, C.; Agarwal, S.; Konar, M.; Evans, R.; Graves, T.; Lowe, J.; Shah, H.; Seth, S.; et al. Apache Hadoop YARN: Yet another resource negotiator. In Proceedings of the 4th Annual Symposium on Cloud Computing, Santa Clara, CA, USA, 1–3 October 2013.
23. Li, P.; Ju, L.; Jia, Z.; Sun, Z. SLA-Aware Energy-Efficient Scheduling Scheme for Hadoop YARN. In Proceedings of the IEEE International Conference on High Performance Computing & Communications, New York, NY, USA, 24–26 August 2015.
24. Gencer, A.E.; Bindel, D.; Siner Emin, G.; Renesse, R.V. Configuring Distributed Computations Using Response Surfaces. In Proceedings of the Middleware Conference, Vancouver, BC, Canada, 7–11 December 2015.
25. Shao, Y.; Li, C.; Gu, J.; Zhang, J.; Luo, Y. Efficient Jobs Scheduling Approach for Big Data Applications. *Comput. Ind. Eng.* **2018**, *117*, 249–261. [CrossRef]
26. Memishi, B.; Perez Maria, S.; Antoniu, G. Diarchy: An Optimized Management Approach for MapReduce Masters. *Procedia Comput. Sci.* **2015**, *51*, 9–18. [CrossRef]
27. Dong, C.; Shen, Q.; Cheng, L.; Yang, Y.; Wu, Z. SECapacity: A Secure Capacity Scheduler in YARN. In Proceedings of the International Conference on Information and Communications Security (ICICS), Singapore, 29 November–2 December 2016; Lecture Notes in Computer Science. Volume 9977.
28. Lu, W.; Chen, L.; Wang, L.; Yuan, H.; Xing, W.; Yang, Y. NPIY: A Novel Partitioner for Improving MapReduce Performance. *J. Vis. Lang. Comput.* **2018**, *46*, 1–11. [CrossRef]
29. Lin, J.; Lee, M. Performance evaluation of job schedulers on Hadoop YARN. *Concurr. Comput. Pract. Exp.* **2016**, *28*, 2711–2728. [CrossRef]
30. Wei, W.; Du, J.; Yu, T.; Gu, X. SecureMR: A Service Integrity Assurance Framework for MapReduce. In Proceedings of the Annual Computer Security Applications Conference, Honolulu, HI, USA, 7–11 December 2009; pp. 73–82.

