Denial-of-Service Threat to Hadoop/YARN Clusters with Multi-Tenancy

Jingwei Huang, David M. Nicol, and Roy H. Campbell
Information Trust Institute, University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
{*jingwei,dmnicol,rhc*}*@illinois.edu*

***Abstract*—This paper studies the vulnerability of unconstrained computing resources in Hadoop and the threat of denial-of-service to a Hadoop cluster with multi-tenancy. We model the problem of how many nodes in a Hadoop cluster can be invaded by a malicious user with given allocated capacity as a k-ping-pong balls to n-boxes problem, and solve the problem by simulation. We construct a discrete event simulation model to estimate MapReduce job completion time in a Hadoop cluster under a DoS attack. Our study shows that even a small amount of compromised capacity may be used to launch a DoS attack and cause significant impacts on the performance of a Hadoop/YARN cluster.**

***Keywords*-Big Data; MapReduce; Hadoop; YARN; multi-tenancy; security; denial-of-service attacks.**

I. Introduction

Driven by the needs of handling big data, MapReduce [1], a scalable distributed parallel programming paradigm, has quickly become a popular tool for big data. Hadoop (hadoop.apache.org/), an open source implementation of MapReduce, now is a major cloud-based platform for big data, and has been widely deployed in the cloud. Given the important role of Hadoop as a primary infrastructure for big data, the trustworthiness of MapReduce job completion in Hadoop is a concern.

Hadoop has evolved into a new generation – Hadoop 2, in which the classic MapReduce module (used in Hadoop v.1.x) is upgraded into a new computing platform, called YARN (or MRv2) [2], [3]. YARN has many important improvements. It is more scalable, i.e. capable to facilitate larger Hadoop clusters with up to 6000 even $10,000$ nodes (machines). YARN is more flexible. YARN supports not only MapReduce but also many other parallel programming models such as Dryad, Spark, Giraph, Storm, Tez, and others [3]. YARN uses *containers* as a unified form of computing resource to replace the map slots and reduce slots used in classic MapReduce; a container can be used for a map task or a reduce task, and each container could be configured with different size. This is a fundamental improvement, which facilitates high flexible and efficient usage of computing resources in a Hadoop cluster. YARN uses *ResourceManager* to replace classic JobTracker, and uses *ApplicationMaster* to replace classic TaskTracker. *ResourceManager*, uses a scheduler to allocate *containers* to multiple users and their applications requesting computing resources in a Hadoop cluster. Previously, all Hadoop schedulers manage computing resources among multiple users based on memory only. Other computing resources, such CPU, GPU, HD r/w, and network bandwidth, are not considered in scheduling. Just most recently, the updated fair scheduler [4] for YARN permits to schedule based on both memory and CPU. To the best of our knowledge, there are few studies on the security and performance impacts of unconstrained shared resources in Hadoop. This paper aims to address the vulnerability of unconstrained computing resources in Hadoop and the threat of denial-of-service attacks to a Hadoop cluster with multi-tenancy.

The contents of this paper are organized as follows. Section II discusses the related work; section III analyzes the vulnerability of unconstrained resources, presents the threat of denial-of-service, and formalizes the threat as a k ping-pong ball to n boxes problem; section IV presents our simulation model to estimate MapReduce job completion time under DoS attacks, and discusses potential solutions; finally, we conclude this work in section V.

II. Related Work

MapReduce and its implementation, Hadoop, have received a great deal of attention from both academic and industrial communities. However, the research on MapReduce and Hadoop has mainly focused on the system performance aspect, and the security issues seemly have not received sufficient attention. Earlier designs of Hadoop assumed that Hadoop is used within an organization in a secured environment, so that no authentication and authorization mechanisms were considered. As Hadoop becomes more and more popular in the cloud, more security mechanisms have been introduced or studied for Hadoop [5], [6]. Examples include: Kerberos based authentication, mutual authentication applied to prevent from unauthorized access, an encrypted shuffle, and so on. To enhance Hadoop system security, authentication and authorization is definitely necessary. In addition to that, there also exist other security issues that should not be neglected. A study [7] on security incidents occurring in large computing facilities shows that the majority of incidents (55%) were attacks on authentication mechanisms; credential compromise is the number one incident type. This finding is particularly important to Hadoop systems with multi-tenancy; as we will see in this paper, a single

978-1-4799-5057-7/14 $31.00 © 2014 IEEE
DOI 10.1109/BigData.Congress.2014.17

compromised credential can be used to launch a Denial-of-Service attack and degrade the performance of a Hadoop cluster significantly.

In most schedulers of Hadoop, only memory is considered in computing resources allocation among multiple users sharing a same Hadoop cluster; other resources remain unconstrained. Just recently, the fair scheduler [4] can be configured to consider both memory and CPU in resource allocation, using the concept of "Dominant Resource Fairness" (DRF) [8], which is a model of scheduling with multiple type of resources for multiple users. The basic idea of DRF is that the user having a smaller "dominant share" will have a higher priority for its new resource requests; the "dominant share" of a user is the maximal share of the resources allocated to that user. We believe, any shared resources left unconstrained in resource management will be a vulnerability. To the best of our knowledge, there are few studies on the security and performance impacts of this vulnerability to a Hadoop cluster with multi-tenancy.

MapReduce job completion time is critical to many applications. Highly relevant to our concerns, Verma et al constructed a MapReduce job makespan model [9], [10], which calculates the average and maximal completion time of a MapReduce job running on the classic MapReduce system, based on the estimated average and maximal tasks' completion time in four phrases: map, "first shuffle" (starting before map ends), "typical shuffle" (starting after map ends), and reduce. This work provides us useful clues. Different from the work, first, we will estimate MapReduce job completion time in a Hadoop/YARN cluster rather than in a classic Hadoop MapReduce system; secondly, we use the approach of stochastic discrete-event simulation, rather than static analytic approach as in Verma's model; as a result, our estimation is a cumulative distribution function of a MapReduce job completion rather than estimated average and maximal values; thirdly, we focus on the DoS impact on Map/Reduce task completion time.

III. Vulnerability and Threat

As discussed in §II, Hadoop employs a scheduler to assign computing resources in a cluster to each application launched by users, based on the computing capacity assigned to each user. Memory was the only computing resource considered in capacity allocation; just recently, CPU has been taken into account in the fair scheduler. Other resources such as HD r/w, GPU, and network bandwidth are still not constrained. We believe, the existence of unconstrained resources in a Hadoop cluster with multi-tenancy poses a vulnerability; this vulnerability not only can lead to unfair use of shared resources, but also can be exploited with denial-of-service (DoS) attacks. A malicious user, who may be an intruder using a compromised credential, an insider, or even a legitimate user with intent to attack a targeted user for business competition, can launch applications using minimal constrained resources but exhausting unconstrained resources in a Hadoop cluster shared with a targeted user. The overused resources become the bottleneck in each node running such DoS attack "tasks". This type of exploitation can potentially lead to a significant decrease of system performance, thus causing the loss to other cloud users. This paper attempts to study how bad this vulnerability could be.

A. Threat model

In our analysis for the above identified threat, we have the following assumption.

Assumption: *In order to maximize the impact of a DoS attack, attackers will take the strategy of using the capacity they gained to occupy unconstrained shared resources as many as possible, in particular, to hold as many containers as possible, and to distribute those containers in as many nodes as possible.*

First, we introduce the variables to be used as follows. n denotes the total number of living nodes that a Hadoop cluster currently has; c^i_{nd} denotes the total capacity of resource i on each node; here for simplicity, we assume that all of the nodes in a cluster are identical. c^i_{min} denotes the minimum capacity of resource i that a container is allowed to have; c^i_{max} denotes the maximum capacity of resource i that a container is allowed to have; C^i denotes the compromised capacity of resource i, represented as a percentage of the total capacity in a cluster; m denotes the number of nodes running DoS attack "tasks"; k denotes the number of containers (in a Hadoop cluster) that a malicious user can have within his allocated capacity cap.

We can characterize the scale of the addressed DoS attacks in two dimensions: (1) *attack broadness*, which is defined as $b = m/n$; (2) *attack strength*, denoted as s, which is the portion of resources occupied by the DoS attack in an infected node. For example, given attack broadness $b = 78.5\%$, and attack strength $s = 80\%$, a task will cost as $1/(1-s)$ (here 5) times long as usual to complete, with the probability of b (here 78.5%). A MapReduce job consisting of many map/reduce tasks may be further delayed, because some tasks under DoS attack may fail to complete within maximum time limits, so they were treated as failures and rescheduled. In the following, we mainly focus on attack broadness.

By the attack strategy stated earlier, k will be

$$k = \min_i \{n \cdot c^i_{nd} \cdot C^i / c^i_{min}\}. \quad (1)$$

At least, this number of containers will be allocated to the attacker and will be used for DoS attack "tasks". Those containers will be assigned to a set of nodes by a scheduler at run time. With respect to what nodes will be assigned to those containers with DoS attack "tasks", it will depend on the run time environment, such as data location in the

Number of nodes	Capacity per node	Minimal capacity per container	Compromised capacity	Number of nodes affected by DoS attack	Percentage of affected nodes
1000	16GB	0.5GB	2%	20 - 640	2% - 64%
1000	16GB	0.5GB	3%	30 - 960	3% - 96%
6000	64GB	0.5GB	0.5%	30 - 3840	0.5% - 64%
6000	64GB	1GB	1%	60 - 3840	1% - 64%
6000	64GB	0.5GB	1%	60 - 6000	1% - 100%

Figure 1. Scenarios: possible impacts of DoS attack on a Hadoop cluster

cluster, other jobs, scheduler type, scheduler configuration, and so on.

When the capacity obtained by the attacker is fully used, the number of nodes running DoS attack "tasks", m, ranges in an interval $[m_{lb}, m_{ub}]$, where the lower bound, m_{lb}, is the least number of nodes to run the attacker's containers; the upper bound, m_{ub}, is the number of nodes when each container is on a different node. That is,

$$m_{lb} = \max_i \{\lceil (k \cdot c^i_{min}/c^i_{nd}) \rceil\}; \tag{2}$$

$$m_{ub} = \begin{cases} k & \text{if } k < n \\ n & \text{if } k \geq n \end{cases} \tag{3}$$

Let us consider some scenarios as given in figure 1. In these scenarios, only memory is considered as most of schedulers do. The first scenario tells that a Hadoop cluster has 1000 nodes, each of which has 16GB RAM; the minimal size for a container is 0.5GB RAM; a compromised client has capacity of 2%; then through this compromised client, a DoS attack, such as running CPU-intensive, or disk r/w intensive, or network bandwidth-intensive jobs, can impact 20 to 640 (or 2% to 64% of) nodes in the cluster. The other scenarios have even bigger possible impacts.

An attacker will try to make the infected nodes m as large as possible. We have discussed the possible range of impacted nodes m; now we discuss how to estimate the possible value of it by simulation.

By earlier assumption about a DoS attack, using the obtained portion of the capacity in a Hadoop cluster, the attacker will request as many containers as possible and put them in as many different nodes as possible. Therefore, the attacker's Application Master will request each of his containers in a new node that does not have his containers yet. In current schedulers, e.g. capacity scheduler [11], if the requested node is available, the request will be granted; otherwise, a container in a node located at the same rack as the requested node will be assigned, if available; in the case of that all nodes in that rack are not available, a node in another rack will be assigned. Let p_t be the probability that the targeted node is available; p_n be the probability that a requested container is assigned to a new node; m now represents the current number of nodes that have containers assigned to the attacker; m changes with the process of deploying attack tasks to the nodes in the cluster.

We model the possible maximal number of nodes an attacker can reach as the following problem of throwing ping-pong balls into a cluster of boxes.

k-Ping-Pong balls to n-boxes problem: *A player is given k ping-pong balls and n boxes, and he scores every time when he throws a ball into an empty box. Those n boxes are arranged one by one tightly full of a small pool; so that each ping-pong ball must fall into one of the boxes. The player can see whether or not a box is empty. So, the player will try to throw those balls into as many boxes as possible. Targeting at an empty box, the player may throw a ball into that targeted box accurately, with probability p_t; the ball may bounce over the boxes, but eventually will fall into one of the boxes; and the ball may still fall into another empty box, with probability $(n-m-1)/n$, where m is the current number of boxes with balls. The question is what is the most possible number of boxes with ping-pong balls when the game is over.*

Obviously, the probability of a ping-pong ball falling into an empty box is

$$p_n = p_t + (1 - p_t) \cdot \frac{n - m - 1}{n}. \tag{4}$$

If p_t at each throw is the same, the maximal number of boxes with ping-pong balls can be found analytically. Unfortunately, as discussed earlier, this probability p_t is not a constant for each request in our real problem regarding whether or not a request to assign a container in a specific machine is granted by the scheduler. Correspondingly, in the *k-ping-pong balls to n-boxes game*, there could be some uncertain factors such as unstable wind, so that the probability of a ping-pong ball entering into a targeted box at each different throw is also uncertain. We will find a solution of this problem by using discrete event simulation in the next section.

This abstraction of k-ping-pong balls to n-boxes problem is general, reflecting common features of different problems in the real world, e.g. airdropping humanitarian aid load to a disaster area.

B. Simulating k-Ping-Pong Balls to n-Boxes Problem

The k-ping-pong balls to n-boxes problem can be easily modeled and simulated by using Mobius [12]. Mobius is a stochastic discrete event system modeling and simulation software tool. Mobius supports several types of models. One of them is Stochastic Activity Network (SAN). SAN is a stochastic extension to Petri Net. SAN allows different types of places including *float* and complex data structures; SAN has timed activities; SAN has more flexible and expressible firing (Input_Gate) conditions and output functions (Output_Gate).

Number of Boxes (n)	Number of Balls (k)	Expected number of occupied boxes, E(m)	Confidence interval (in level 0.95)	Success Rate E(m)/k	Success Rate E(m)/n
1000	100	98.03	0.023	98.03%	
1000	500	440.64	0.084	88.13%	
1000	700	586.65	0.111	83.81%	
1000	900	719.93	0.135	79.99%	
1000	1000	781.6	0.1	78.16%	78.16%
1000	1100	840.31	0.152		84.03%
1000	1200	895.8	0.163		89.58%
1000	1300	948.71	0.173		94.87%
1000	1400	995.8	0.113		99.58%
1000	1460	999.996	0.003		100.00%
10000	14600	10000	0.000		100.00%
10000	10000	7824.13	0.455		78.24%
20000	20000	15648.07	0.646		78.24%

Figure 2. Simulated solutions of expected maximal number of occupied boxes in the k-ping-pong balls to n-boxes problem for different k

Because p_t (the probability of a ball falling into a targeted box) is uncertain in each throw, we treat it as a random variable over $[0, 1]$ with uniform distribution.

The simulation results for several typical scenarios are given in figure 2.

We define a success rate, λ, as the ratio of the expected number of occupied boxes to the targeted number of boxes,

$$\lambda = \begin{cases} E(m)/k & \text{if } k < n \\ E(m)/n & \text{if } k \geq n \end{cases}$$

As shown in figure 2, for a given number of boxes ($n = 1000$), the success rate (λ) changes with the growth of the number of ping-pong balls(k); when k is much smaller than n, λ is close to 100%; when k gets closer to n, it gradually becomes smaller; when k equals to n, λ reaches it minimum 78.5%; when k gets larger than n, λ becomes larger; when $k = 1.4n$, λ is 99.8%, approaching to 1.0. In the experiment with larger total number of boxes (n), the results are similar. For example, when $n = 1,000,000$, and k is equivalent, $\lambda = 78.43\%$, slightly smaller; when $k = 1.4n$, λ becomes 1.0.

The above results can be interpreted in the context of DoS attacks in Hadoop, by regarding k as the number of containers that an attacker has, regarding n as the number of nodes in a Hadoop cluster. For a cluster of $6,000$ nodes with 48GB RAM, the least container being allowed to have 0.5GB, a DoS attack, obtaining 1% of capacity, could have 5760 containers, and deploy attack tasks to 4560 nodes, covering 76% of the cluster. If the attacker gains 1.5% capacity of the cluster, he could deployed attack "tasks" to the whole cluster.

IV. Simulating MapReduce Job Completion Time Under DoS Attacks

In this section, first, we briefly introduce MapReduce; then we present how to simulate a MapReduce job.

A. MapReduce

MapReduce processes a large volume of data in parallel in two phrases: the map phrase and the reduce phrase. Each phrase is completed by a number of tasks, each of which handles a subset of data characterized by the types of key/value pairs. Basically, MapReduce uses two primitive functions[1]: a map function that processes a key/value pair to generate a list of intermediate key/value pairs; a reduce function that accepts an intermediate key and a list of intermediate values with the same intermediate key, to generate a list of new key/value pairs [13], that is,

$$\begin{aligned} map: &\quad (K_1, V_1) \rightarrow list(K_2, V_2); \\ reduce: &\quad (K_2, list(V_2)) \rightarrow list(K_3, V_3). \end{aligned} \tag{5}$$

The above functions and the types of keys/values needs to be defined by users; a MapReduce system will take care how to compute them. MapReduce is an elegant abstraction. It is simple, but captures the general and essential features of a large variety of parallel data processing problems in the real world; it allows users to easily map their problems into the MapReduce problem in an unified form, and to left the computing details on parallelization, optimization, and fault tolerance to computers.

In MapReduce, before the execution of a reduce function, the values with the same intermediate key, which are outputs from map tasks, need to be collected, sorted, and presented to the reduce function. This process is called "shuffle". Shuffle is executed by each reduce task as the first phrase of reduce [13].

B. Simulation Model

We have constructed a stochastic discrete event simulation model with Mobius to estimate the completion time of a MapReduce job (called an application in YARN) in a Hadoop/YARN cluster under a DoS attack.

1) Overall structure: The overall structure of our simulation model is illustrated by figure 3. Based on the workflow of MapReduce jobs running on YARN [2], [14], this model simulates a MapReduce job in four stages: initialization, map, reduce (including shuffle phrase and reduce phrase), and completion; each stage is simulated by a corresponding component. In addition, "resource management" component dynamically calculates the available resources to the simulated MapReduce job; "DoS attack" component simulates deploying DoS attack tasks to nodes in a Hadoop/YARN cluster, by using simulation model of k-ping-pong balls to n-boxes problem.

A MapReduce job consists of a set of map tasks and a set of reduce tasks. In our simulation model, each map or reduce task is simulated by an *atomic model* of Mobius; all of the map tasks (or reduce tasks) are combined together through a *Replicate* type of *composed model* in Mobius. Based on the YARN model, we simulate the execution process of a set of map tasks followed by a set of reduce tasks.

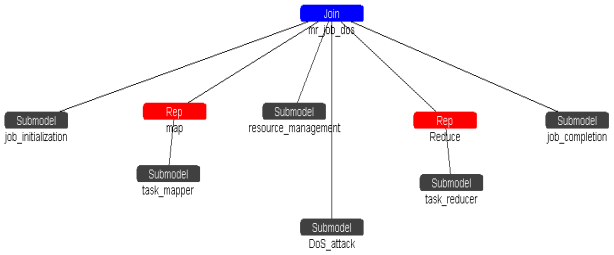

Figure 3. Overall structure of simulation

2) Simulation of a task: Now we discuss in more details the sub-model for simulating the completion time of a map (or reduce) task, in the running environment with DoS attacks. The model for a map task and the one for a reduce task have very similar components and structure. The only difference is that a reduce task has an extra shuffle phrase. The model structure for a reduce task is illustrated in figure 4; and the one for a map task is similar but without shuffle.

To execute a map (or reduce) task, first of all, the *ApplicationMaster* (*AM*) that manages the MapReduce job requests a container for this task from *ResourceManager* (*RM*); the *RM* will respond to assign a container in an available node; then that *AM* contacts the *NodeManager* (*NM*) of the assigned node to set up the container for this task. A container carrying out a map task, or a reduce task, is called a *mapper*, or a *reducer* correspondingly. The completion time of this container-assignment process is simulated as a random variable within an interval.

MapReduce is designed to assume the existence of operation failures in a Hadoop cluster. Our simulation considers this feature. After the container for a map (or reduce) task is set up, this task will be executed in that container within a specified maximum time limit; if this task fails to complete within the time limit, the task will be retried for a number of times, saying 3 times; if all tries are failed, this task will be treated as a failure, and that *AM* will contact *RM* to set up another container (possibly in another node) for this task, until this task is completed or the MapReduce job is terminated. We could also consider the failure of a node where the container resides. The failure could be node crash due to hardware failure or security attacks. For the purpose of this paper on the impact of DoS attacks, we do not consider node crash and set the probability as 0.

A mapper or a reducer possibly resides in a node with DoS attack "tasks". The chance for this to occur can be estimated as the ratio of the number of the nodes infected by DoS attacks, which is estimated by "DoS attacks" component, to the total number of living nodes in the cluster, i.e. m/n.

A mapper (or reducer) running on a node infected by DoS attack will receive degraded service, and may become very slow, because some computing resources necessary to the completion of the task but unconstrained by YARN scheduler are exhausted by DoS attack "tasks". The strength or effectiveness of a DoS attack is represented with a number between 0 and 1 (exclusive), representing the portion of computing resources occupied by the DoS attack, called "degraded degree"; 0 corresponds to no DoS attack; a number close to 1 corresponds to the most severe attack in which all computing resources are exhausted. We further assume an linear relation between the task completion time and the computing resource used for the task. With this assumption, the completion time of a task under a DoS attack is estimated as

$$t' = t/(1-d), \tag{6}$$

where t is the task completion time without DoS attacks, and d is the degree of degraded service; for example, $d = 80\%$ means that a DoS attack task consumes 80% of the computing resource of the resided node, such as CPU, GPU, HD I/O, and network bandwidth. In our simulation, if t', the completion time of a simulated task, falls within the maximum time limit T_{max}, the task is completed; otherwise, the task is treated as failure in this run; in the case of failure, as real MapReduce system does, the task will be retried for a number of times in the same container on the same node, and if the task is still failed, it will be rescheduled to run in a newly assigned container and possibly on a different node.

3) Completion time of an activity: We treat the completion time of an activity (execution of map function, or shuffle, or reduce function) as random variable following Gamma probability distribution.

$$f(x;\alpha,\beta) = \begin{cases} \frac{1}{\Gamma(\alpha)\beta}(\frac{x}{\beta})^{\alpha-1}e^{-\frac{x}{\beta}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where $\alpha > 0$ is called shape parameter, $\beta > 0$ is called scale parameter, and $\Gamma(\alpha)$ is Gamma function. When α is large (saying, $\alpha \geq 10$), the distribution is close to normal distribution. The expected value of the random variable will be $E(X) = \alpha\beta$; the mode, i.e. the peak in the probability density function of Gamma distribution, or the most likely value, will be $Mode(X) = (\alpha-1)\beta$. We set scale parameter (β) of Gamma distribution for this type of tasks as $\beta = T_{mode}/(\alpha-1)$, or $\beta = T_{mean}/\alpha$, where T_{mean} is the observed mean completion time of that type of activities, in normal Hadoop cluster operation environment (without DoS attacks); T_{mode} is the completion time which is most likely to appear for that type of activities.

The real completion time of an activity will depend on the run-time operation environment in the Hadoop cluster such as input data, the number of running jobs, or more exactly, the cluster capacity usage at run-time. The environment factors are uncertain. It is reasonable to assume that MapReduce task completion time follows Gamma distribution. The long tail of Gamma distribution reflects the lagging effects of some run-time environment factors.

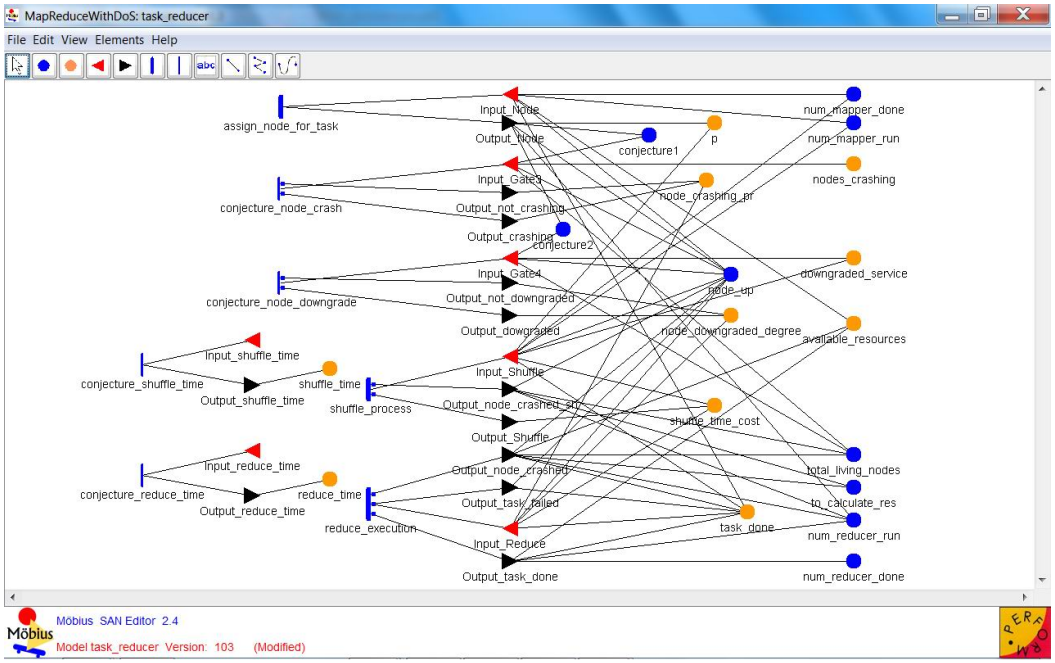

Figure 4. Modeling and simulation of a reduce task completion time

4) Simulation of shuffle: Now we discuss the execution of a shuffle phrase. Shuffle is performed by each reduce task as its first stage, to collect the outputs of some map tasks. Although reduce tasks can start to do shuffle before all of map tasks are completed, usually the shuffle of a reducer needs to collect outputs from a large number of mappers; thus being possibly unable to complete, before all of the mappers that the shuffle needs to read complete. A distinguishing improvement of YARN over classic MapReduce is that resources are no longer divided into map slots and reduce slot; containers can be used for both map and reduce tasks. Because of this feature, it is inefficient and a waste of allocated resources to start reducers, if there are mappers still waiting for execution, and some reducers' shuffle will have to wait for map completion.

For this reason, a better strategy for scheduling a MapReduce job is to do map tasks first, and schedule reduce tasks later when no more map tasks are in waiting. Although the strategy is not enforced in a scheduler, we assume that the *AM* of the simulated MapReduce job uses the following conditions to start a reducer: (1) no mappers are still waiting for execution, that is,

$$n_{map} = n_{map}^{done} - n_{map}^{run} \tag{7}$$

meaning that all mappers are either completed or already assigned with containers and in execution, even though some of them may be failed later and rescheduled to execute; (2) there are enough available resources for the requested container.

To estimate completion time of shuffle, at the time point when a reducer starts its shuffle phrase, we calculate the completion rate of map phrase as

$$p = n_{map}^{done}/n_{map}. \tag{8}$$

This number can be used to estimate how much work of a shuffle phrase left to finish at the time point that the map phrase completes. For example, at the time point that a reducer was launched to start its shuffle phrase, the map completion rate was 30%, which means there would be a large overlap between map and shuffle; later at the time point when all mappers complete, it is reasonable to think that the majority of work of shuffle (saying 70%) has been done, and roughly, only 30% left to execute. In another case, if the completion rate of map phrase is 100% when a shuffle starts, the shuffling work amount left is 100%.

C. Scenarios of DoS impact on job completion time

We simulated a number of scenarios with different levels of DoS attacks. In the experiments, we assume that a Hadoop/YARN cluster has 1000 nodes (n=1000); each of them has 16GB RAM; minimal capacity per container is 1 GB; the simulated MapReduce job is assigned with 5% of capacity of the cluster; the number of map tasks is 1024; the number of reduce tasks is 256; capacity scheduler is assumed to be used. The cumulative distribution functions of the completion time of this MapReduce job in different scenarios under different scales of DoS attacks are illustrated in figure 5.

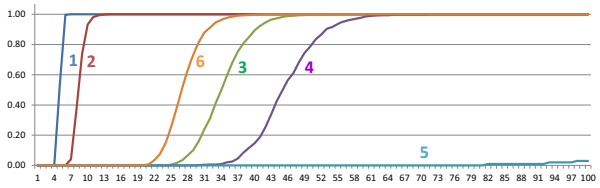

Figure 5. Cumulative distribution functions of the simulated MapReduce job completion time in different scenarios with DoS attacks (1).

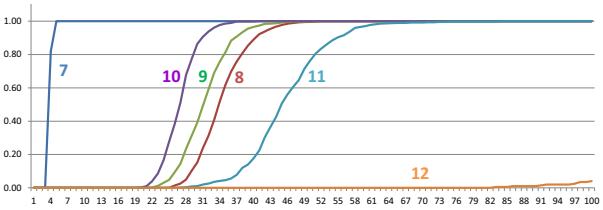

Figure 6. Cumulative distribution functions of the simulated MapReduce job completion time in different scenarios with DoS attacks (2).

- *Scenario 1 – no DoS attacks*: it is the normal case; at time point 5, the degree of belief (in the sense of probability) in job completion is 99.7%;
- *Scenario 2– "mild" DoS attack*: compromised capacity = 6.25%, correspondingly $k/n = 1$, and roughly attack broadness $m/n = 78.2\%$; attack strength = 50%; job completion time is roughly doubled;
- *Scenario 3 – stronger DoS attack*: compromised capacity = 6.25%, and attack strength = 80%; job completion time becomes about 8 times later;
- *Scenario 4 – broader and stronger DoS attack*: compromised capacity = 9.375%, corresponds to $k/n = 1.5$, and $m/n = 100\%$; and attack strength = 80%; in this scenario, all nodes are infected, and job completion time is further delayed, compared with scenario 3;
- *Scenario 5 – broader and strong DoS attack*: compromised capacity = 9.375%, and attack strength = 90%; in this scenario, the CDF of job completion time almost stays as 0 in the time window of observation;
- *Scenario 6 – longer maximum task time limit*: compromised capacity = 6.25%, attack strength = 80%, and the maximum time limits for tasks are longer; compared with scenario 3, job completion time is improved in this scenario. This is because the longer maximum time limit allows tasks delayed by DoS attack to finish.

The simulations for much larger Hadoop/YARN clusters with nodes up to 6000 or 10,000 have similar curves of cumulative probability functions in comparable scenarios. Figure 6 shows some simulations for a cluster of 10,000 nodes with 64 GB RAM. The job is the same as earlier one. In all scenarios except 9, the allocated capacity is 0.125%, thus having the same amount of capacity as scenarios in figure 5.

- *Scenario 7*: no DoS attack;
- *Scenario 8*: to have the same attack broadness and attack strength as earlier scenario 3, we set compromised capacity = 1.5625%, correspondingly $k/n = 1$, and attack strength = 80%; totally, this scenario is similar to earlier scenario 3;
- *Scenario 9*: compromised capacity = 1.5625%%, attack strength = 80%, and the allocated capacity for the job is sufficiently big so that all tasks of the job can be executed in parallel; compared with scenario 8, job completion time has a little improvement;
- *Scenario 10*: compromised capacity = 1.5625%%, attack strength = 80%, and the maximum time limits for tasks are longer; compared with scenario 8, job completion time is improved, as earlier scenario 6 does;
- *Scenario 11*: compromised capacity = 2.3%, which makes $k/n = 1.472$ and the attack broadness almost 100%; and attack strength = 80%; this scenario is similar to earlier scenario 4;
- *Scenario 12*: compromised capacity = 2.3%, and attack strength = 90%; similar to earlier scenario 5.

Although the completion time of a MapReduce job under DoS attack can be improved to a certain extent by allowing longer maximum execution time for tasks, this improvement is limited by the general DoS attack scale as discussed in §III-A; in other words, the job completion time will still cost at least as $1/(1-s)$ times long as normal, in probability m/n.

D. Further Discussion

The presented DoS threat can be addressed in several ways. First, since the root of this problem is that there exist computing resources unconstrained by Hadoop/YARN schedulers, the most straightforward recipe is to manage all types of computing resources in schedulers. Secondly, it would be interesting to have a mechanism to avoid providing to ApplicationMaster the real location of each container, thus shielding YARN system from the DoS threat. However, in the current YARN design, the ApplicationMaster needs to know which node an assigned container resides, and will contact with the NodeManager of that node to set up the container and to monitor the task in that container. Thirdly, the DoS attacks exists in multi-tenancy environment, so it is important for a user to learn the security and the resource usage patterns of other users sharing the cluster. However, it is unclear whether or not, or how much, a cloud provider should provide information about users sharing the same cluster, for security and privacy concerns. Finally, it is of interest to us to develop metrics to characterize the status of a node, a rack, and a cluster, the patterns of resource consumption by a job, and by a tenant. By using the metrics, a Hadoop system may easily identify DoS attacks and unfair resource consumption, and react correspondingly. Such metrics can be used as a new "transparency" mechanism

to make cloud services more trustworthy [15]. The metrics can also be used by the ApplicationMasters for each job, to optimize the execution of those jobs, at the same time, optimizing task distribution in the cluster.

V. Concluding Remarks

In this paper, we studied the vulnerability of unconstrained computing resources in Hadoop and the threat of denial-of-service attacks to a Hadoop/YARN cluster with multi-tenancy. We formalized the problem about how many nodes in a Hadoop cluster can be invaded by a malicious user with a given amount of allocated capacity as a k-ping-pong balls to n-boxes problem, and solved the problem by simulation. We developed a discrete event simulation model to estimate MapReduce job completion time in a Hadoop cluster under DoS attack. Our study shows that even a small amount of compromised capacity may be used to launch a DoS attack and can cause significant impact on the performance of a Hadoop/YARN cluster. Although our simulation assumes the use of YARN capacity scheduler, the result applies to Hadoop systems using other schedulers having unconstrained resources shared among multiple users.

Regarding future research, we will move forward to developing metrics to measure the resilience of MapReduce systems, and extending our trust calculus [16] for estimating and optimizing the trustworthiness of cloud workflow for handling big data.

Acknowledgment

This material is based on research sponsored by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI 2004*, 2004, pp. 137–150. [Online]. Available: http://www.usenix.org/events/osdi04/tech/dean.html

[2] A. C. Murthy, C. Douglas, M. Konar, O. O'Malley, S. Radia, S. Agarwal, and K. V. Vinod, "Architecture of next generation apache hadoop mapreduce framework," Tech. Rep., 2011. [Online]. Available: https://issues.apache.org/jira/secure/attachment/12486023/MapReduce_NextGen_Architecture.pdf

[3] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:16. [Online]. Available: http://doi.acm.org/10.1145/2523616.2523633

[4] Apache. Hadoop mapreduce next generation - fair scheduler. [Online]. Available: http://hadoop.apache.org/docs/current2/hadoop-yarn/hadoop-yarn-site/FairScheduler.html

[5] O. O'Malley, K. Zhang, S. Radia, R. Marti, and C. Harrell, "Hadoop security design," Yahoo!, Tech. Rep., 2009.

[6] Securosis, "Securing big data: Security recommendations for hadoop and nosql environments," Oct 2012. [Online]. Available: https://securosis.com/assets/library/reports/SecuringBigData_FINAL.pdf

[7] A. Sharma, Z. Kalbarczyk, J. Barlow, and R. Iyer, "Analysis of security data from a large computing organization," in *Dependable Systems Networks (DSN), 2011 IEEE/IFIP 41st International Conference on*, 2011, pp. 506–517.

[8] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 24–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972490

[9] A. Verma, L. Cherkasova, and R. H. Campbell, "Two sides of a coin: Optimizing the schedule of mapreduce jobs to minimize their makespan and improve cluster performance," in *Proceedings of the 2012 IEEE 20th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, ser. MASCOTS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 11–18. [Online]. Available: http://dx.doi.org/10.1109/MASCOTS.2012.12

[10] A. Verma, *Performance Modeling Framework for Service Level Objective-driven MapReduce En vironments*. PhD Disertation, University of Illinois at Urbana-Champaign, 2012.

[11] Apache, "Hadoop mapreduce next generation - capacity scheduler," Tech. Rep., 2013. [Online]. Available: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html

[12] UIUC Mobius Team, *Mobius Manual, Version 2.4, Rev. 1*, University of Illinois at Urbana-Champaign, Dec. 2012. [Online]. Available: https://www.mobius.illinois.edu/docs/MobiusManual.pdf

[13] T. White, *Hadoop: the definitive Guide, 3rd Edition*. O'Reilly, 2012.

[14] A. C. Murthy, "Next generation of apache hadoop mapreduce the scheduler," March 2011. [Online]. Available: http://developer.yahoo.com/blogs/hadoop/posts/2011/03/mapreduce-nextgen-scheduler/

[15] J. Huang and D. M. Nicol, "Trust mechanisms for cloud computing," *Journal of Cloud Computing*, vol. 2, no. 1, 2013. [Online]. Available: http://www.journalofcloudcomputing.com/content/2/1/9

[16] J. Huang and D. Nicol, "A Calculus of Trust and Its Application to PKI and Identity Management," in *Proceedings of IDTrust'09, ACM Digital Library, http://portal.acm.org/citation.cfm?id=1527017.1527021*, 2009.