Poster: Cross Cloud MapReduce: an Uncheatable MapReduce

Yongzhi Wang *(student)*, Jinpeng Wei *(faculty)*
Florida International University
e-mail: {ywang032, weijp}@cis.fiu.edu

Mudhakar *Srivatsa (research scientist)*
IBM T. J. Watson Research Center
e-mail: msrivats@us.ibm.com

I. Introduction

MapReduce [1] is becoming a popular data processing application on Cloud Environment. However, security issues make many customers reluctant to move their critical computation tasks to cloud. For instance, [2] points out a real security vulnerability that the cloud service leader Amazon EC2 suffers from: some members of EC2 can create and share Amazon Machine Image (AMI) to the EC2 community so that other users can deploy their server by simply loading an AMI. A malicious AMI, if widely used, could flood the community with hundreds of infected virtual instances. On the other hand, in MapReduce where jobs are carried out via the collaboration of a number of computing nodes, merely one malicious node may render the overall results useless. In a traditional MapReduce setting where each node is deployed on the cloud, the integrity of a computation can be easily compromised and difficult to detect.

In this work, we propose a new MapReduce Framework, Cross Cloud MapReduce (CCMR), which can be deployed among a single private cloud and multiple public clouds. By using the replication, hold-and-test, verification and credit-based trust management approaches, CCMR can eliminate malicious compute nodes and guarantee high computation accuracy while incurring acceptable overhead.

II. Thread Model and System Design

A. Attacker Model and System Assumption

In our research, we model the attacker as a "powerful adversary" that controls malicious nodes in each public cloud environment. However, the number of malicious nodes is not big enough to overwhelm that of benign nodes in each cloud. The adversary can be an insider or external to any cloud, or it can be a distributed algorithm implemented and deployed among all the malicious workers. The adversary receives and shares information collected by the malicious nodes and instructs a select subset of malicious nodes to whether or not cheat the master in order to introduce as many errors as possible to the final result without detection. For example, if two malicious workers are assigned to execute the same task, the adversary can instruct them returning the same wrong result so that simply comparing the result will not detect the error.

Since we only focus on the computation, we assume that everything except for the malicious nodes is trusted: the private cloud which is deployed on the customer trusted organization is trusted. Hence, the master and the verifier deployed on the private cloud are also trusted. Also, we assume the distributed file system for data storage and the communication network inside and among the clouds are trusted and not compromised. Plus, since the worker only report the hash code of task result to the master, we resort to the commitment-based protocol [3] to assume the hash code is consistent with the actual result accepted by the master.

B. System Design

In CCMR, we redefine the architecture of MapReduce: the master and a small number of slave workers are deployed on the trusted private cloud within the customer's organization; and other slave workers are deployed on multiple public clouds. The workers deployed in the private cloud are called *verifiers*, since they are used to verify the results returned by the untrusted workers: they arbitrate the inconsistent results if needed and verify the consistent results in a non-deterministic manner. The master is responsible for assigning tasks and checking the consistency of the task results. The system architecture is shown in Figure 1.

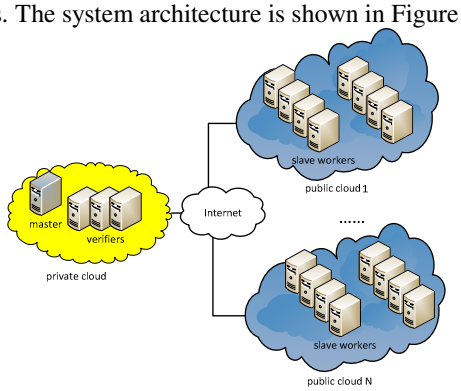

Figure 1: CCMR Architecture

In CCMR, we employ four approaches to assist the multi-cloud architecture to guarantee the integrity of computations: replication, hold-and-test, verification and credit based trust management. The overall process can be depicted in Figure 2. In Figure 2, cloud A and B are two clouds randomly picked from all the public clouds in the CCMR environment. W1 and W2 are two slave workers randomly selected from cloud A and B respectively. The "Verify task" step is completed by the verifier. The other components in the figure, including the Task Queue Hash code buffer for the worker W1, are all maintained in the master.

For each task (e.g., task t2), it is first randomly assigned to a worker W1 from one cloud A. Then it is replicated and randomly assigned to another worker W2 from a different cloud B. The result hash code of both

the original and replicated task should be returned to the master. Instead of scheduling both tasks at the same time, The master holds the replicated task until the original task returns the result. When the master receives the result hash code of the original task R1 from the first worker (step 2), it stores R1 in W1's hash code buffer and assigns the replicated task to the second worker (step 3). When the master receives the result hash code of the replicated task R2, it compares the two values. If the two hash codes are different, the master will send the task to the verifier to arbitrate and determine the malicious node (step 5). We call this approach *hold-and-test*. Part III will show the efficacy of this approach in limiting the impact of collusive nodes. Still, there exists a small portion of collusions not detected by the hold-and-test approach. In order to defeat it, consistent results are randomly verified by the verifier (step 7). If the verifier generates a different result, the master determines that both workers are malicious. Each worker that executes one original task and passes the hold-and-test check will increment its credit (step 6). And the original task result is buffered in the worker's storage. When a worker has accumulated enough credit, which we call *credit threshold*, the master will accept all the results buffered in that worker and reset its credit (step 11). During the whole process, any worker that fails the "verify task" step will be added to a blacklist (step 8) and all the tasks whose results were obtained from that worker will be rescheduled (step 9).

III. System Analysis

CCMR has two lines of defense against the attacker. The multi-cloud architecture raises the bar for the attacker. Since each task is replicated and assigned to two (or more) workers from different cloud service providers, successfully breaking in two or multiple public cloud is already non-trivial challenge for the attacker. Figuring out the instances corresponding to a specific MapReduce job in each cloud and compromising them to construct collusion is even more difficult.

Even if the first line of defense is breached, the system design described in Part II.B can still protect the accuracy of computations. First, since each task is replicated, the naïve malicious worker that does not collude with others can be easily eliminated. Second, the hold-and-test approach can efficiently reduce the success rate of collusion. For each task, the first worker has to return the result before the replicate task is assigned to the second worker. Suppose the first worker is controlled by the adversary, since the assignment of task is random, the adversary cannot predict whether the second worker is also under his control. Since the benign worker takes the majority portion, asking the first worker to return a wrong result under this situation is very likely to reveal it. Even worse, when a malicious worker received a task, it cannot tell whether the task is the original or the replicated one if the adversary hasn't seen this before: it is possible the original task is assigned to a benign worker. So the best strategy to evade detection over a long run is to return correct result. For the malicious workers who try to return incorrect result in a

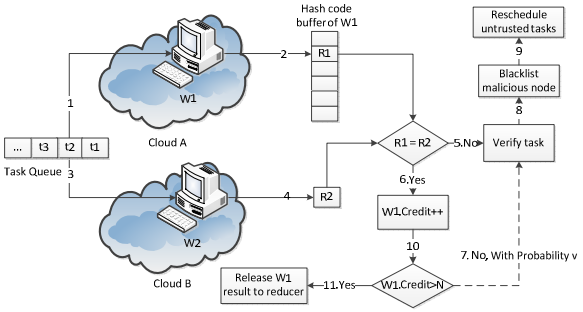

Figure 2: Data flow of CCMR

hope of not being discovered, the verification and credit based approach guarantee very low probability of introducing errors to the final result: malicious workers who return erroneous results frequently will be caught before his buffered results are accepted by the master; Malicious workers who return erroneous results rarely may introduce some error to the final results, however, in many applications such as data mining and statistics, minor errors would not severely undermine the fidelity of the application.

IV. Implementation and Experiments

Based on the system design, we implement the CCMR based on the Hadoop MapReduce framework and deploy it across a local private cloud, 6 extra small instances of Microsoft Azure and 6 small instances of Amazon EC2.

We successfully launch several Apache Mahout [4] application on CCMR: Bayes Classification, Canopy Clustering, *k*-means Clustering, Fuzzy *k*-means Clustering, and Dirlchlet Process Clustering.

We also measure the error detection efficacy and overhead under different scenarios and attacker models by running the Hadoop WordCount Application. In this experiment, we vary the fraction of malicious workers *n* from 0.15 to 0.5, and model different attacker strategies by changing the cheat ratio of adversary from 0.1 to 1.0. The experiment result shows that in each scenario, the probability that the master node accepts an erroneous result decreases very quickly with the increase of credit threshold *N*. When *N* is set to 9, each experiment configuration will accept 0 errors. Under different configurations, CCMR incurs from 100% to 160% execution overhead for workers or verifiers, and at most 45% execution overhead for the verifiers.

References

[1] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. InCommunications of the ACM, 51 (1): 107-113, 2008.

[2] "Cloud Security: Amazon's EC2 serves up 'certified pre-owned' server images" http://dvlabs.tippingpoint.com/blog/2011/04/11/cloud-security-amazons-ec2-serves-up-certified-pre-owned-server-images

[3] Wei Wei, Juan Du, Ting Yu, Xiaohui Gu, "SecureMR: A Service Integrity Assurance Framework for MapReduce", in Proceedings of the 2009 Annual Computer Applications Conference.

[4] "Apache Mahout", http://mahout.apache.org/