Adaptive Data-Driven Service Integrity Attestation for Multi-Tenant Cloud Systems

Juan Du
Department of Computer Science
North Carolina State University
jdu@ncsu.edu

Nidhi Shah
Cisco Systems
nidshah@cisco.com

Xiaohui Gu
Department of Computer Science
North Carolina State University
gu@csc.ncsu.edu

***Abstract*—Cloud systems provide a cost-effective service hosting infrastructure for application service providers (ASPs). However, cloud systems are often shared by multiple tenants from different security domains, which makes them vulnerable to various malicious attacks. Moreover, cloud systems often host long-running applications such as massive data processing, which provides more opportunities for attackers to exploit the system vulnerability and perform strategic attacks. In this paper, we present *AdapTest*, a novel adaptive data-driven runtime service integrity attestation framework for multi-tenant cloud systems. AdapTest can significantly reduce attestation overhead and shorten detection delay by adaptively selecting attested nodes based on dynamically derived trust scores. Our scheme treats attested services as black-boxes and does not impose any special hardware or software requirements on the cloud system or ASPs. We have implemented AdapTest on top of the IBM System S stream processing system and tested it within a virtualized computing cluster. Our experimental results show that AdapTest can reduce attestation overhead by up to 60% and shorten the detection delay by up to 40% compared to previous approaches.**

I. Introduction

Cloud systems [1] have recently emerged as popular resource leasing infrastructures. Application service providers (ASPs) can lease a set of resources from the cloud system to offer software as a service [3] without paying the expensive cost of owning and maintaining their own computing infrastructures. Could systems are particularly amenable for data processing services [2], [10], [15], [21], which are often extremely resource-intensive. In particular, our work focuses on dataflow processing systems [5], [15], [16] that have many real world applications such as security surveillance and business intelligence. As shown by Figure 1, users can feed data from various data sources into the cloud system to perform various data processing functions and receive final data processing results from the cloud.

However, cloud systems are often shared by multiple tenants that belong to different security domains, which makes them vulnerable to various malicious attacks. Moreover, data processing services are often long-running, which provides more opportunities for attackers to exploit the system vulnerability and perform strategic colluding attacks. Although virtualization ensures certain isolation between users, malicious attackers can still leverage the shared hardware to launch attacks [6], [26] from the VMs they own or by compromising VMs of benign users. One of the top security concerns[1] for cloud users is to verify the integrity of data processing results. For example, a malicious (or compromised) credit checking service may provide an incorrect credit score to lead to a wrong mortgage application decision. Note that the integrity attack is the most prevalent, which affects both public and private data processing.

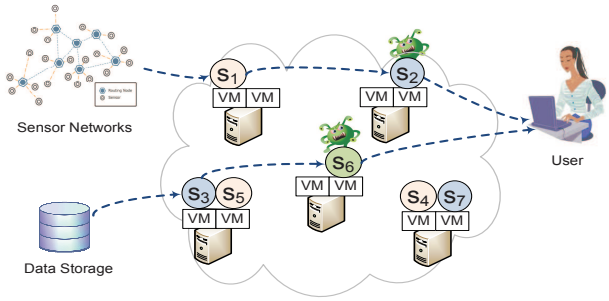

Fig. 1. **Integrity attack in cloud-based data processing.**

Although previous work has proposed various remote integrity attestation techniques [7], [30], [31], existing solutions often require trusted hardware or secure kernel to be co-existed with the remote computing platform, which are difficult to be deployed in cloud systems. Although traditional Byzantine Fault Tolerance (BFT) techniques (e.g., [9], [19]) can detect malicious behavior using replicated services, those techniques often incur high overhead and impose certain agreement protocol over all replicas. To this end, we explore a *data-driven* integrity attestation approach that only relies on result consistency to detect malicious attacks, which is completely *transparent* to the attested services without imposing any special software or hardware requirements.

In this paper, we present *AdapTest*, a novel adaptive runtime service integrity attestation framework for large-scale cloud systems. AdapTest builds on top of our previously developed system RunTest [13] that performs randomized probabilistic attestation and employs a clique-based algorithm to pinpoint malicious nodes. However, randomized attestation still imposes significant overhead for high-throughput multi-hop

978-1-4577-0103-0/11/$26.00 ©2011 IEEE.

[1]Note that confidentiality and user data privacy are important orthogonal issues addressed by previous work [20], [35].

data processing services. In contrast, AdapTest dynamically evaluates the trustiness of different services based on previous attestation results and adaptively selects attested services during attestation. Thus, AdapTest can significantly reduce the attestation overhead and shorten the detection delay. Specifically, this paper makes the following major contributions:

- We provide a novel adaptive multi-hop integrity attestation framework based on a new *weighted attestation graph* model. We derive both per-node trust scores and pair-wise trust scores to efficiently guide probabilistic attestation.
- We have implemented AdapTest on top of the IBM System S stream processing system [15] and tested it on the virtual computing lab (VCL) [4], a production virtualized computing cluster that operates in a similar way as Amazon EC2 [1]. Our experimental results show that AdapTest can significantly reduce attestation overhead for reaching the 100% detection rate by up to 60% and shorten detection time by up to 40% compared to previous randomized attestation approaches.

The rest of the paper is organized as follows. Section II provides a brief background about cloud service integrity attack and an overview of our approach together with our assumptions. Section III presents the design details. Section IV presents the prototype implementation and experimental results. Section V compares our work with related work. Finally, the paper concludes in Section VI.

II. Overview

In this section, we first describe our system model for cloud-based data processing services and service integrity attestation. Next, we present the service integrity attack model followed by an overview of our approach and our key assumptions.

A. System Model

Cloud systems are shared computing infrastructures consisting of a set of physical hosts interconnected via networks. Each host can run multiple virtual machines (VMs) that may belong to different owners. The application service provider (ASP) can lease a collection of VMs to host its software services. Each service instance, denoted by s_i, provides a specific data analysis function, denoted by f_i, such as sorting, filtering, correlation, or data mining utilities. Multiple service instances can be *functionally-equivalent*, providing the same service function for load balancing or fault tolerance purposes. Moreover, popular services naturally attract different service providers for profit. A multi-party service provisioning infrastructure usually employs some *portal* nodes [17], [28] to aggregate different service components into composite services based on the user's requirements. The user accesses cloud services by submitting input data to the portal node that will forward the user data to different service instances for processing and then deliver final results back to the user. Portal nodes can authenticate users to allow only authorized users to access the cloud services.

B. Attack Model

In a shared cloud infrastructure, malicious attackers can pretend to be legitimate service providers to provide fake service instances or compromise vulnerable benign service instances by exploiting their security roles. Our work focuses on detecting the service integrity attack where a malicious (or compromised) service instance gives untruthful data processing results.

To escape detection, malicious attackers may want to perform *selective cheating*. That is, they can misbehave on a selective subset of received data while pretending to be benign on other received data. Thus, the attack detection scheme must be able to capture misbehavior that are both unpredictable and occasional without losing scalability. Although we can perform integrity attestation on all service instances all the time, the overhead of integrity attestation would be very high, especially for high throughput data processing services in large-scale cloud systems. Thus, an effective attack detection scheme must perform *sneaky* attestation, which can prevent attackers from gaining knowledge about our attestation scheme (i.e., when and which set of data will be attested.). Otherwise, the attacker can compromise the integrity of selective data processing results without being detected at all.

Furthermore, cloud computing infrastructures often comprise a large number of hosts running many more VMs and application service instances. It creates new opportunities for colluding attacks where multiple malicious attackers launch coordinated attacks or multiple benign service instances are simultaneously compromised and controlled by a single malicious attacker. Colluders can communicate with each other in an arbitrary way and produce the same incorrect results on the same input. Attackers can also change their attacking and colluding strategies arbitrarily. However, we assume that attackers do not have knowledge of other benign service instances that they do not interact with.

C. Approach Overview & Assumptions

Our service integrity attestation scheme has two major design goals: 1) support runtime continuous attestation with low overhead; and 2) pinpoint malicious (or compromised) service instances among a large number of interacted service instances without assuming any prior knowledge about which service instances are trusted. AdapTest adopts a data-driven approach to achieve the above design goals without imposing any special hardware or software requirements over remote attested services, illustrated by Figure 2. AdapTest leverages the portal node to perform service integrity attestation. To achieve non-repudiation, each service instance is required to produce a receipt for each data it receives and sign the data it has processed [12].

AdapTest performs attack detection using replay-based consistency check [13]. The basic idea is to duplicate some original inputs and re-send them as *attestation data* to different functionally-equivalent service instances for consistency check. Note that attestation data and original data are made indistinguishable to service instances. Moreover, our attestation

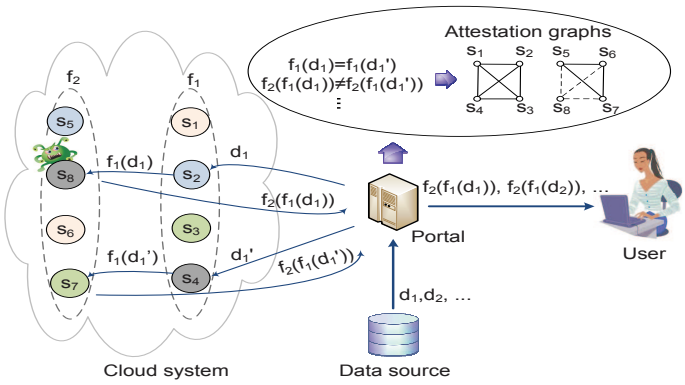

Fig. 2. **Data-driven service integrity attestation.**

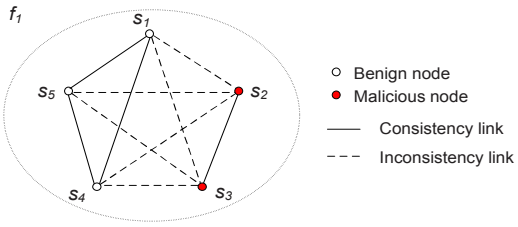

Fig. 3. **Clique-based malicious node pinpointing.**

scheme does not affect the original data processing. In other words, original data can be routed as before to different service instances for processing based on certain load balancing and quality-of-service (QoS) management objectives. The attestation data are replayed after the portal receives the original data processing results rather than being sent concurrently with the original data. Thus, we can prevent two colluding attackers from detecting attestation by comparing their received data and thus escaping detection. Although the replay scheme may cause delay in a single data item processing, we can overlap the attestation and normal processing of consecutive data items to hide the attestation delay from the user.

AdapTest leverages our previously developed clique-based algorithm [13] to pinpoint malicious nodes, illustrated by Figure 3. The portal node constructs an attestation graph where nodes are functionally equivalent service instances. If two nodes always give consistent output, we use a consistent link between these two nodes. Otherwise, if they give inconsistent results on at least one input data, we link them using an inconsistent link. Since all benign nodes will always give consistent correct results, they will form a *consistency clique* in the attestation graph. In contrast, the malicious nodes will be exposed with inconsistent links when their misbehavior is caught by our attestation scheme. Note that colluding malicious nodes may try to form a consistency clique by always giving the same wrong results. However, if we assume benign nodes are the majority, we can say a node is definitely malicious if the node is outside of all the cliques whose sizes are larger than half of the total nodes [13]. For example, in Figure 3, we can see the attestation graph includes two cliques $\{s_1, s_4, s_5\}$ and $\{s_2, s_3\}$. Since the size of the first clique is larger than half of the total nodes, s_2 and s_3 are successfully identified as malicious nodes even though they also try to form a clique through colluding.

AdapTest performs adaptive attestation to quickly expose malicious nodes. We make three key observations. First, we should attest suspicious nodes more often in order to capture selective cheating with minimum attestation data. Second, in order to quickly pinpoint malicious nodes, we need to expose as many inconsistency links as possible. Therefore, AdapTest dynamically derives a set of trust scores for each node based on previous attestation results and use those trust scores to guide future attestation. Specifically, AdapTest attests the nodes with lower trust scores with higher probability, and gives priority to those node pairs that have not been attested before or have been consistent before. Third, attesting multi-hop data processing services requires additional consideration since inconsistent intermediate processing results from upstream hops will invalidate attestation for all downstream hops. To address the problem, AdapTest intentionally picks good nodes based on previous attestation results for upstream hops in order to effectively attest downstream hops.

Note that AdapTest does not use the trust scores to directly pinpoint malicious nodes. Without assuming the trustiness of any nodes, the trust scores only represent the relative goodness of different nodes. The trust score of a specific node can dynamically change after the node is attested with different nodes. Even if the node trust scores are stabilized, it is very difficult, if not impossible, to pre-define a proper trust score threshold to separate the malicious and benign nodes. Such threshold depends on a set of unknown factors such as the percentage of malicious nodes and the misbehaving probability of those malicious nodes. Thus, AdapTest only uses trust scores to guide attestation but still uses the clique-based malicious node pinpointing algorithm to guarantee zero false positive [13].

Assumptions. First, we assume that data processing services are stateless and deterministic, that is, given the same input, a benign node always produces the same output. Many data processing functions such as projection, selection, filtering fall into this category [15]. We can also extend our scheme to support stateful data processing services [11], which however is beyond the scope of this paper. Second, we assume that benign nodes are the majority within each group of functionally-equivalent service instances. This assumption is the same as other common attack detection schemes [25]. Third, we assume that the portal node is trusted, which is solely managed by the portal service provider whose goal is to provide trust-worthy data processing services for its clients. The portal node plays a similar role as the dispatcher used by previous remote attestation schemes [30], which is also assumed to be trusted. Further, the portal node can employ authentication to easily protect itself from malicious clients or malicious application service providers.

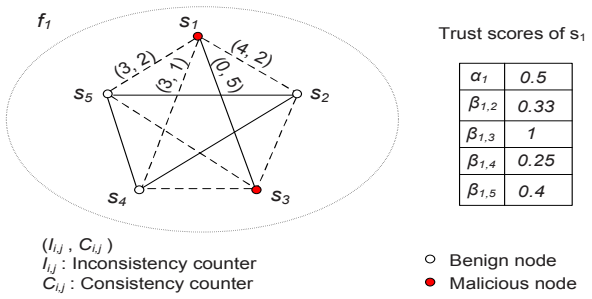

Fig. 4. **Weighted attestation graph.**

III. Design and Algorithms

In this section, we present the design and algorithm details of the AdapTest system. We first describe a weighted attestation graph model that serves as the basis of our approach. Next, we present the details of per-hop adaptive attestation algorithm followed by the multi-hop adaptive attestation scheme.

A. Weighted Attestation Graph

AdapTest strives to pinpoint malicious service instances without making any prior assumption about the trustiness of any service instance. Moreover, malicious attackers can perform selective cheating during long-running data processing services, which means the trust score of a service instance must be continuously monitored and updated. Thus, AdapTest employs a *weighted attestation graph* to aggregate previous attestation results and dynamically derives a set of trust scores for each service instance, illustrated by Figure 4. We formally define the weighted attestation graph as follows.

Definition 1: A *weighted attestation graph* is an undirected complete graph consisting of all functionally equivalent service instances as nodes. The weight of each edge consists of a pair of counters denoting the number of inconsistent results and the number of consistent results respectively.

For example, in Figure 4, s_1 produces three inconsistent results and two consistent results with s_5. Two nodes are connected by a consistent link only if they have zero inconsistent result. We can derive a *node trust score* and a set of *pairwise trust scores* for each node from the weighted attestation graph. The node trust score denotes how trustworthy a node is and the pair-wise trust score denotes how well two nodes trust each other. We formally define both the node trust score and the pairwise trust score as follows.

Definition 2: The *trust score* of the node s_i, denoted by α_i, is defined as the fraction of consistent results returned by the node s_i when attested with all the other nodes. Node trust scores range within [0,1], and are initialized to be 1.

Definition 3: The *pairwise trust score* between two service instances s_i and s_j, denoted by $\beta_{i,j}$, is calculated by the fraction of consistent results when s_i is attested against s_j. The pairwise trust score ranges within [0, 1], and are initialized to be -1, which means that s_i and s_j have not been attested with each other yet.

The trust score of a node takes the consistency relationships between this node with all the other nodes into consideration. For example, in Figure 4, s_1 has a node trust score of 0.5 since it has total 10 consistent results and 10 inconsistent results with $\{s_2, s_3, s_4, s_5\}$. Intuitively, malicious nodes should have higher probabilities than benign ones to be inconsistent with the other nodes given that benign nodes are the majority. Thus, we assign node trust scores according to how consistent a node is with the other nodes. Nodes that are more consistent with the others have higher trust scores and are considered to be more trustworthy. The node trust scores can be affected by two factors. The trust score of a node s_i decreases if i) the node s_i is inconsistent with *more* nodes; or ii) the node s_i is inconsistent with other nodes *more frequently*.

The pairwise trust scores reflect how consistent two nodes are and therefore how trustworthy they think each other. The more frequently two nodes give inconsistent results, the less pairwise trust score between them. Note that we initialize pairwise trust scores with -1 to indicate that the two nodes have not been attested together before. For example, in Figure 4, if the pairwise trust score between two nodes equals to 1, we draw a solid line between them. Otherwise, if two nodes do not always agree with each other, we use a dashed line to represent the inconsistency relationship. The pairwise trust score between s_1 and s_2 is $2/(4+2) = 0.33$ since they produce 4 inconsistent results and 2 consistent results.

B. Per-Hop Adaptive Attestation

AdapTest leverages dynamically derived trust scores to intelligently guide probabilistic service attestation. The goal of our adaptive attestation scheme is to expose malicious nodes faster. We achieve the goal by capturing more inconsistency relationships for malicious nodes so that they can be pushed out from the maximum consistency clique.

AdapTest expedites the exposure of inconsistency relationships and therefore shorten detection time using two adaptive node selection schemes. First, AdapTest selects suspicious nodes that have low trust scores and attests those suspicious nodes more frequently. The rationale is that the nodes that have already delivered more inconsistent results have the potential to deliver even more inconsistent results in the future attestation. By intensively attesting suspicious service nodes, we may have higher probabilities to find inconsistency results. Second, AdapTest strives to attest suspicious nodes together with benign nodes since two colluding malicious nodes will try to avoid producing inconsistent results with each other. Attesting a suspicious node together with a benign one is more effective in producing inconsistent results.

For scalability, AdapTest performs probabilistic attestation by randomly selecting a subset of input data for consistency check. When an input data item is selected for attestation by the portal, AdapTest first identifies a pool of suspicious nodes based on node trust scores and randomly selects a

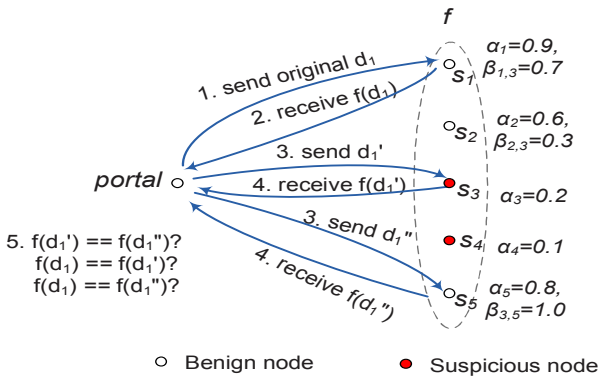

Fig. 5. **Per-hop adaptive attestation.**

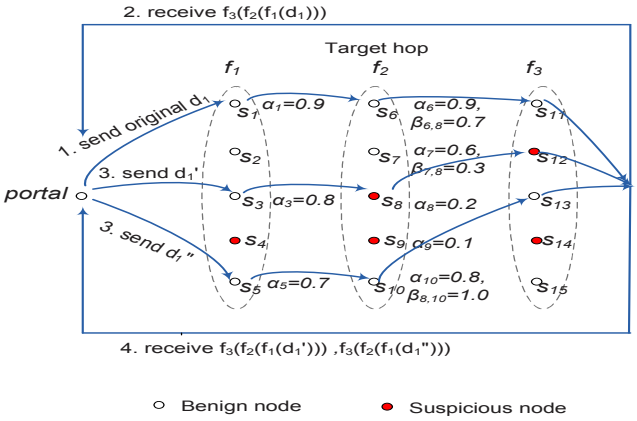

Fig. 6. **Multi-hop adaptive attestation.**

suspicious node from this pool to attest. Given the assumption that malicious nodes are no more than half of total nodes, we rank all nodes in an increasing order of trust scores, and mark the first $\lfloor N/2 \rfloor$ nodes as the suspicious node pool B and the rest of nodes as the benign node pool G, where N is the total number of nodes providing the function. We then randomly pick one node, s_i, from the suspicious node pool, excluding the node processing the original data, for attestation by comparing with the original node. Note that we do not want to always attest the node with the lowest trust score for maintaining attestation coverage and tolerating imprecise trust scores. Moreover, we want to avoid alerting the malicious node by continuously attesting it. At the beginning, since all nodes have the same initial trust scores, AdapTest will randomly pick nodes from the whole node pool to attest.

AdapTest may send multiple attestation data to attest different nodes concurrently. To maximize the chance of capturing inconsistent results, we want to attest a suspicious node with a benign node together. Thus, after picking a suspicious node s_i from B, AdapTest picks the other attested node from the benign node pool G using the following rules. First, if there are benign nodes that have not been attested with s_i before (i.e., pairwise trust score equals to -1 in the weighted attestation graph), we randomly pick one from them. Second, if all nodes in the benign set have been attested with s_i, we randomly pick one from G that have always been consistent with s_i. We avoid attesting two nodes that have already been inconsistent with each other, since further attestation will not result in new inconsistency links. Thus, if all nodes in G are inconsistent with s_i, instead of attesting s_i, we randomly select another node, s_j, from the suspicious node set to attest, and also select a node from the benign node set according to the above rules to pair with s_j for attestation. Note that if all nodes in the suspicious node set have inconsistency links with all the nodes in the benign node set, we randomly pick s_i, and then pick the one node from G that has the highest pair-wise trust score with s_i. Our scheme can achieve both good coverage and avoid wasting attestation traffic on those node pairs that have already presented inconsistency relationships.

Figure 5 shows an example of adaptive per-hop attestation. The number associated with each protocol step indicates the execution order. If two steps have the same number, it means that the two steps are executed concurrently. The portal first sends the original data d_1 to s_1 for processing. After the portal receives the result from s_1, it decides to perform attestation by replaying d_1 on two service instances. The portal first randomly picks one node from the suspicious node set $\{s_3, s_4\}$ to attest, say, s_3. Then the portal picks the node with the highest pairwise trust score with s_3 from the benign set $\{s_1, s_2, s_5\}$, say s_5.

Note that our scheme is robust to strategic attacks. For example, a malicious node s_3 may behave benignly at the beginning to join the benign node pool and then start to misbehave by colluding with its colluder s_4 in the suspicious node pool, with an intent to hide its misbehavior by sacrificing s_4. However, s_3 and s_4 are not just attested against each other, but also attested with the node that processes the original data, which is randomly selected. Even if s_3 and s_4 are consistent with each other, they will have inconsistency links with all benign nodes. Thus, s_3 and s_4 will eventually be pinpointed by our clique-based algorithm. If either s_3 or s_4 is selected as the original node, our replay-based attestation scheme prevents it from knowing whether it will be compared with benign nodes or its colluding party. Thus, if s_3 tries to pretend to be benign, it will have an inconsistency link with s_4 with high probability. In this case, AdapTest will rarely attest s_3 and s_4 since we try to avoid attesting two nodes that already have an inconsistency link between each other. Note that s_3 and s_4 can help each other to increase their trust scores and decrease other nodes' trust scores by providing consistent wrong results. However, as mentioned in Section II-C, AdapTest does not rely on the trust scores to pinpoint malicious nodes. Instead, AdapTest only considers the consistency/inconsistency links and uses clique-based algorithm to pinpoint malicious nodes. The trust scores are only used to expose more inconsistency links with less attestation data.

C. Multi-Hop Adaptive Attestation

Complicated data processing services often comprise multiple data processing functions called service hops. Malicious

attackers can attack any of the service hops to compromise the final data processing results. Suppose a data processing service consists of total n hops and an input data item is selected for attestation through two service paths, $s_1 \rightarrow ..., s_i \rightarrow ... \rightarrow s_n$ and $s'_1 \rightarrow ..., s'_i \rightarrow ... \rightarrow s'_n$, respectively. If the intermediate processing results begin to become inconsistent at the hop s_i, then the attestation for all service hops after s_i becomes invalid since all downstream node pairs, such as s_{i+1} and s'_{i+1}, would receive different input data. In this case, attestation data cannot be efficiently utilized. More attestation data are required to attest downstream service hops, which will result in extended detection time in multi-hop attestation.

AdapTest provides adaptive multi-hop attestation by intentionally picking benign upstream nodes based on the node trust scores in order to efficiently attest a downstream node, illustrated by Figure 6. Specifically, during a n-hop service attestation, we first randomly select a service hop, say the ith hop, as the target attestation hop. For each service hop before the ith hop, we intentionally select nodes that have high trust scores. For the ith to the nth service hops, we follows the same per-hop attestation node selection scheme described in the previous subsection. Thus, AdapTest maintains high probabilities for the nodes at the ith hop to receive consistent input to perform valid attestation.

Figure 6 shows an example of multi-hop adaptive attestation. We target to attest the second service hop. Thus, we select only benign nodes $\{s_1, s_3, s_5\}$ for the first service hop. For the second service hop, AdapTest randomly selects s_8 from the suspicious set, and selects s_{10} from the benign set for concurrent attestation. Similarly, for the third service hop, a suspicious node s_{12} and a benign node s_{13} are selected for attestation.

IV. Experimental Evaluation

In this section, we first describe our experiment setup, the schemes used for comparison as well as the evaluation metrics. We then present our experimental results in detail.

A. Experiment Setup

We have implemented AdapTest in c++ on top of the IBM System S stream processing system [15], a production system that can analyze massive continuous data streams in real-time. We have deployed and tested the AdapTest prototype on a subset nodes of the NCSU virtual computing lab (VCL) [4], which is a virtualized computing cluster similar to Amazon EC2. We used 10 blade servers in VCL, each of which runs CentOS 5.2 64-bit and a set of VMs with Xen 3.0.3 hypervisor.

The data processing application we use is extracted from the sample application provided by IBM System S stream processing system. The application takes real weather data from weather stations as input, performs conversions and calculations, and generates the most recent weather information for different locations where the fruit suppliers are located. The results help in making decisions on whether to purchase fruit from a supplier. We perform attestation on three service hops, with five service instances at each hop. The input data rate is 300 tuples per second. We have one trusted portal node that accepts input data streams and constructs the weighted attestation graphs for each service function based on adaptive attestation results.

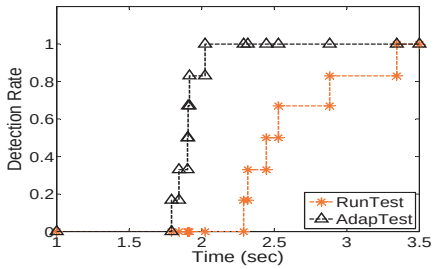

Fig. 7. **Detection time comparison under colluding attacks with node misbehaving probability = 0.2. (40% malicious nodes)**

We first compare AdapTest with RunTest, both of which perform attestation probabilistically and employ only a subset of nodes at a time for attestation. Note that RunTest randomly selects service instances at every hop independently for attestation. For both schemes, when the portal node receives the processing result of a new data item, it has p_u probability of duplicating the data to r copies to launch service attestation, where p_u is called attestation probability and r is called redundancy degree. In this experiment study, both schemes leverage the portal node to select service paths for attestation, which strives to avoid selecting the same node on different attestation paths for good attestation coverage[2].

We also compare AdapTest with the full time majority voting scheme used by traditional Byzantine Fault Tolerance techniques [25]. The full time majority voting scheme performs integrity attestation for all input data using all service instances all the time. When inconsistency happens, the scheme relies on majority voting to detect which instances are faulty.

We evaluate AdapTest using two major metrics: *detection time* and *attestation overhead*. The detection time is the time duration that is needed to detect all malicious nodes. Early detection is desired so that the system can make proper actions to prevent malicious nodes from compromising more data processing results. The attestation overhead is calculated as the number of attestation data that are needed to detect all malicious nodes.

B. Results and Analysis

We first evaluate the detection time and start with the most challenging case. Figure 7 and Figure 8 compare the detection time in collusion scenarios, with 40% malicious nodes and node misbehaving probability of 0.2 and 0.4 respectively. Here, the detection rate is calculated as the number of pinpointed malicious nodes over the total number of malicious

[2]Our original RunTest system implementation makes the portal select attestation nodes for different attestation paths independently, which often involves the same node on different attestation paths. Thus, the performance of RunTest reported in this paper is already much improved compared to [13].

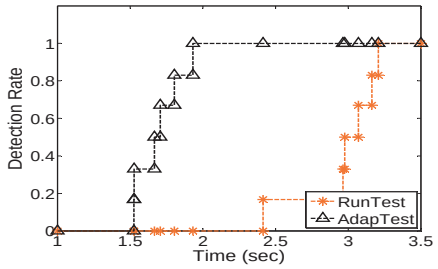

Fig. 8. **Detection time comparison under colluding attacks with node misbehaving probability = 0.4. (40% malicious nodes)**

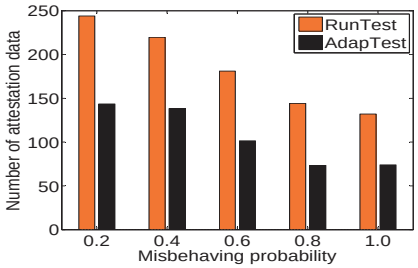

Fig. 9. **Attestation overhead comparison under colluding attacks. (40% malicious nodes)**

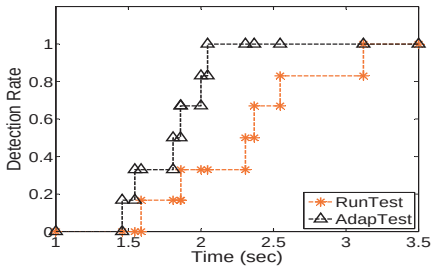

Fig. 10. **Detection time comparison under non-colluding attacks with node misbehaving probability = 0.2. (40% malicious nodes)**

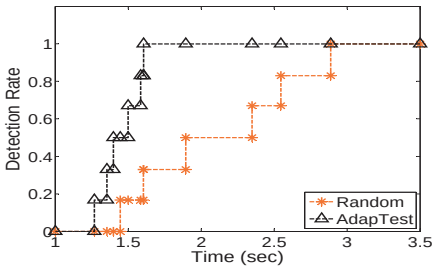

Fig. 11. **Detection time comparison under non-colluding attacks with node misbehaving probability = 0.4. (40% malicious nodes)**

nodes that have misbehaved at least once during the experiment. The detection rate starts at zero and keeps increasing as more malicious nodes are pinpointed. For both algorithms, the attestation probability p_u is 0.2. That is, the portal node randomly selects 20% of original input data for attestation. Each time, the portal node uses two duplicated attestation data items to perform two-way concurrent attestation. Our results show that our adaptive attestation scheme achieves much shorter detection time than the random attestation scheme.

We also evaluate the attestation overhead. Figure 9 compares AdapTest and RunTest under collusion scenarios, with 40% malicious nodes. The misbehaving probability of all malicious nodes varies from 0.2 to 1. When they misbehave, all malicious nodes give the same incorrect processing results. Our results show AdapTest can consistently achieve lower attestation overhead than RunTest.

We now evaluate our algorithms under non-colluding attack cases. Figure 10 and Figure 11 show the time to detect each of the malicious nodes with 40% malicious nodes under node misbehaving probability of 0.2 and 0.4, respectively. We again observe that in both scenarios AdapTest achieves 100% detection rate much earlier than RunTest. Figure 12 shows the attestation overhead comparison under non-colluding attack scenarios, with 40% malicious nodes. Each malicious node misbehaves independently with the misbehaving probability varying from 0.2 to 0.8. The results show that AdapTest consistently incurs less attestation overhead to achieve 100% detection rate, with up to 60% less attestation overhead than RunTest. Note that both schemes need less attestation traffic to detect all malicious nodes when malicious nodes misbehave more frequently. This is because the schemes have more opportunities to catch inconsistency results and derive inconsistency relationships between the nodes. We observe that AdapTest can achieve significant cost reduction under different node misbehaving probabilities.

Comparing Figure 9 and Figure 12, we can see that the attestation overhead for non-collusion scenarios is generally lower than that in collusion scenarios. This is because the inconsistency links of malicious nodes increase much faster in non-collusion scenarios, since malicious nodes produce inconsistent results with both benign nodes and other malicious nodes.

For sensitivity study, we also evaluate the attestation overhead for non-collusion scenarios with 20% malicious nodes, as shown in Figure 13. Again, AdapTest also incurs the lower overhead among all the probabilistic attestation schemes. We did not evaluate collusion scenarios with 20% malicious nodes because we would have only one malicious node per hop, which cannot form collusion. Comparing Figure 13 and Figure 12, we can see that the attestation overhead that is needed to detect all malicious nodes is lower under 20% malicious nodes than that under 40% malicious nodes. This is because we have less malicious nodes to detect.

We also compare the probabilistic attestation schemes with the full time majority voting scheme. Figure 14 compares the detection time with 95% confidence for RunTest, AdapTest and the full time majority voting scheme under challenging collusion scenarios with 40% malicious nodes, where attestation probability is 0.2. Since the full time majority voting scheme employs all nodes all the time, it can detect malicious nodes in the shortest time. However, as Figure 15 shows, it has much higher overhead than the probabilistic attestation schemes. In contrast, RunTest and AdapTest tradeoff a short detection delay for a much lower attestation overhead. We

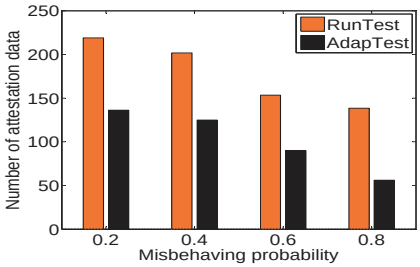

Fig. 12. **Attestation overhead comparison under non-colluding attacks. (40% malicious nodes)**

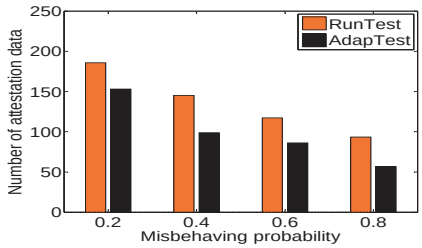

Fig. 13. **Attestation overhead comparison under non-colluding attacks. (20% malicious nodes)**

observe that our adaptive attestation scheme can significantly reduce the detection delay by up to 40% compared to the random attestation scheme. Note that AdapTest and RunTest have the same continuous attestation overhead because they generate the same amount of attestation data given the same attestation probability and the same number of duplicates per attestation.

We evaluate the overhead imposed on the data processing delay for AdapTest. We compare the average data processing delay with and without attestation under different data rates. Figure 16 shows AdapTest imposes little delay overhead on the stream processing.

V. Related Work

Trust management techniques have been studied in different contexts [7], [23], [32]. They are generally used in multi-party systems to protect the interests of honest parties and expose dishonest or malicious parties. For example, the EigenTrust [23] algorithm aims to reduce the number of fake files in peer-to-peer (P2P) networks. It assigns each peer a unique trust score based on the peer's history of uploading authentic files. It then identifies and isolates malicious peers by requiring peers to interact with each other based on the trust score. NetProbe [27] detects networks of fraudsters in online auction sites by analyzing user transactions and infers fraudsters by detecting suspicious patterns. In contrast, our scheme evaluates different service instances by *actively* attesting them rather than merely performing passive monitoring. We use trust scores to guide the active attestation rather than directly use trust scores for malicious node detection that can be highly inaccurate, especially under colluding attacks.

Remote attestation uses a challenge-response paradigm for detecting malicious behavior, which can be classified into

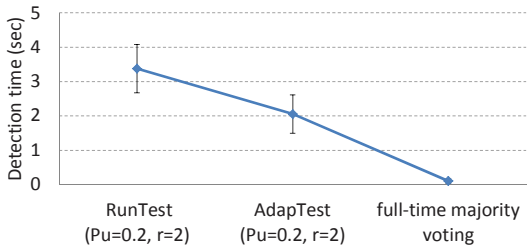

Fig. 14. **Detection time comparison between RunTest, AdapTest and the full time majority voting scheme.**

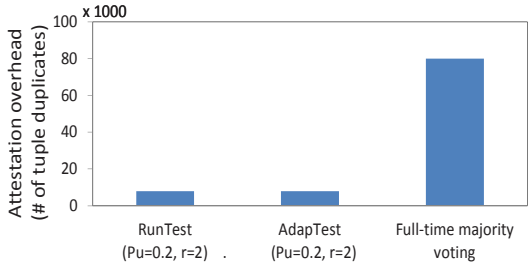

Fig. 15. **Attestation overhead comparison between probabilistic attestation schemes and the full time majority voting scheme.**

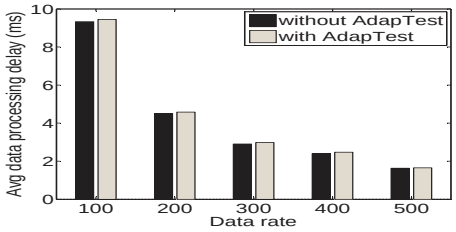

Fig. 16. **Data processing delay comparison.**

system-level and application-level attestation. System-level remote attestation techniques [7], [22], [30] ensure that a remote software platform is running code that is not compromised or altered by attackers. Most of the techniques require trusted hardware or trusted OS kernel to co-exist with the attested software platform. The integrity of the software platform is ensured by employing a remote trusted attester to challenge the trusted entity who provides an integrity evidence through some cryptographic means. Application-level attestation techniques perform integrity attestation using application data. They have been proposed in different application contexts, such as P2P messaging systems [18], publish-subscribe systems [32], cloud storage systems [8], and database systems [34]. In contrast, our work focuses on data processing in cloud systems. Our approach supports black box service integrity attestation, which are general to different application service functions.

Previous work [9], [24], [25] has extensively studied Byzantine Fault Tolerance (BFT) techniques for detecting arbitrary faults in replicated systems. BFT schemes rely on full time majority voting to resolve inconsistency and enforce a certain agreement protocol among all replicas to maintain consistency. Although BFT techniques provide powerful fault detection capabilities, they are impractical for large-scale cloud systems

due to scalability and deployment concerns. In contrast, our integrity attestation scheme is completely transparent to remote services and does not require any application modification. The service instances attested together are not necessarily replicas, which can have different internal implementations. By performing adaptive probabilistic attestation, our scheme can significantly reduce runtime integrity attestation overhead.

Cloud system security has recently received much attention [14], [29], [33]. For example, Erway et. al. presented a framework to protect the integrity of cloud storage system by employing cryptographic methods [14]. Ristenpart et. al. investigated the security holes of existing deployed cloud systems, and identified their vulnerability toward cross-VM side channel attacks [29]. In comparison, our work focuses on protecting data processing service integrity in cloud systems.

VI. Conclusion

In this paper, we have presented AdapTest, a novel adaptive runtime service integrity attestation framework for large-scale multi-tenant cloud systems. AdapTest adopts a data-driven integrity attestation approach to achieve both practicality and scalability. Particularly, AdapTest dynamically derives a set of trust scores to achieve differentiated probabilistic attestation. We have implemented AdapTest on top of the IBM System S stream processing system and tested it on the NCSU virtual computing lab. Our prototype implementation indicates that AdapTest is feasible and efficient for real cloud systems. The experimental results show that AdapTest can reduce attestation overhead by up to 60% and shorten the malicious node pinpointing delay by up to 40% compared to previous approaches.

VII. Acknowledgment

This work was sponsored in part by NSF CNS0915567 grant, NSF CNS0915861 grant, U.S. Army Research Office (ARO) under grant W911NF-10-1-0273, and IBM Faculty Award. Any opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the NSF, ARO, or U.S. Government. The authors would like to thank the anonymous reviewers for their insightful comments.

References

[1] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/.

[2] Apache Hadoop System. http://hadoop.apache.org/core/.

[3] Software as a Service. *http://en.wikipedia.org/wiki/Software as a Service*.

[4] Virtual Computing Lab. http://vcl.ncsu.edu/.

[5] D. J. Abadi and et al. The Design of the Borealis Stream Processing Engine. *Proc. of CIDR*, 2005.

[6] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. Hypersentry: Enabling stealthy in-context measurement of hypervisor integrity. In *Proc. of CCS*, Oct. 2010.

[7] S. Berger, R. Caceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan. Tvdc: Managing security n the trusted virtual datacenter. *ACM SIGOPS Operating Systems Review*, 42(1):40–47, 2008.

[8] K. Bowers, A. Juels, and A. Oprea. Hail: A high-availability and integrity layer for cloud storage. In *Proc. of CCS*, 2009.

[9] M. Castro and B. Liskov. Practcal byzantine fault tolerance. In *Proc. of OSDI*, New Orleans, LA, 1999.

[10] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Proc. of USENIX Symposium on Operating System Design and Implementation*, 2004.

[11] J. Du, X. Gu, and T. Yu. On verifying stateful dataflow processing services in large-scale cloud systems. In *Proc. of CCS (poster session)*, Oct. 2010.

[12] J. Du, W. Wei, X. Gu, and T. Yu. Toward secure dataflow processing in open distributed systems. In *Proc. of ACM Scalable Trusted Computing Workshop (STC)*, Nov. 2009.

[13] J. Du, W. Wei, X. Gu, and T. Yu. Runtest: Assuring integrity of dataflow processing in cloud computing infrastructures. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2010.

[14] C. Erway, C. Papamanthou, A. Kupcu, and R. Tamassia. Dynamic provable data possession. In *Proc. of CCS*, 2009.

[15] B. Gedik, H. Andrade, and et. al. SPADE: the System S Declarative Stream Processing Engine. *Proc. of SIGMOD*, Apr. 2008.

[16] T. S. Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin, 26(1):19-26*, Mar. 2003.

[17] X. Gu, K. Nahrstedt, and et. al. QoS-Assured Service Composition in Managed Service Overlay Networks. *Proc. of ICDCS, 194-202*, 2003.

[18] A. Haeberlen, P. Kuznetsov, and P. Druschel. Peerreview: Practical accountability for distributed systems. In *ACM Symposium on Operating Systems Principles*, 2007.

[19] T. Ho, B. Leong, R. Koetter, and et. al. Byzantine modification detection in multicast networks using randomized network coding. In *IEEE ISIT*, 2004.

[20] P. C. K. Hung, E. Ferrari, and B. Carminati. Towards standardized web services privacy technologies. In *IEEE International Conference on Web Services*, pages 174–183, San Diego, CA, June 2004.

[21] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: Distributed data-parallel programs from sequential building blocks. *Proc. of European Conference on Computer Systems (EuroSys), Lisbon, Portugal*, 2007.

[22] E. Kaiser, W. Feng, and T. Schluessler. Fides: Remote anomaly-based cheat detection using client emulation. In *Proc. of CCS*, 2009.

[23] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th International World Wide Web Conference*, 2003.

[24] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proc. of SOSP*, 2007.

[25] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems, 4(3)*, 1982.

[26] L. Litty, H. A. L.-Cavilla, and D. Lie. Computer meteorology: Monitoring compute clouds. In *Proc. of HotOS*, May 2009.

[27] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos. NetProbe: A Fast and Scalable System for Fraud Detection in Online Auction Networks. In *Proceedings of the 16th international conference on World Wide Web (WWW)*, 2007.

[28] B. Raman, S. Agarwal, and et. al. The SAHARA Model for Service Composition Across Multiple Providers. *Proceedings of the First International Conference on Pervasive Computing*, August 2002.

[29] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off my cloud! exploring information leakage in third- party compute clouds. In *Proc. of CCS*, 2009.

[30] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying code integrity and enforcing untampered code execution on legacy systems. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2005.

[31] E. Shi, A. Perrig, and L. V. Doorn. Bind: A fine-grained attestation service for secure distributed systems. In *Proc. of SSP*, 2005.

[32] M. Srivatsa and L. Liu. Securing publish-subscribe overlay services with eventguard. *Proc. of ACM Computer and Communication Security (CCS)*, 2005.

[33] P. Williams, R. Sion, and D. Shasha. The blind stone tablet: Outsourcing durability. In *Proc. of NDSS*, 2009.

[34] M. Xie, H. Wang, J. Yin, and X. Meng. Integrity auditing of outsourced data. In *International Conference on Very Large Data Base (VLDB)*, 2007.

[35] W. Xu, V. N. Venkatakrishnan, R. Sekar, and I. V. Ramakrishnan. A framework for building privacy-conscious composite web services. In *IEEE International Conference on Web Services*, pages 655–662, Chicago, IL, Sept. 2006.