RunTest: Assuring Integrity of Dataflow Processing in Cloud Computing Infrastructures

Juan Du, Wei Wei, Xiaohui Gu, and Ting Yu
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
{jdu,wwei5}@ ncsu.edu, {gu,yu}@ csc.ncsu.edu

ABSTRACT

Cloud computing has emerged as a multi-tenant resource sharing platform, which allows different service providers to deliver software as services in an economical way. However, for many security sensitive applications such as critical data processing, we must provide necessary security protection for migrating those critical application services into shared open cloud infrastructures. In this paper, we present *RunTest*, a scalable runtime integrity attestation framework to assure the integrity of dataflow processing in cloud infrastructures. RunTest provides *light-weight application-level* attestation methods to dynamically verify the integrity of data processing results and pinpoint malicious service providers when inconsistent results are detected. We have implemented RunTest within IBM System S dataflow processing system and tested it on NCSU virtual computing lab. Our experimental results show that our scheme is effective and imposes low performance impact for dataflow processing in the cloud infrastructure.

Categories and Subject Descriptors

H.4.0 [**Information Systems Applications**]: General

General Terms

Security, Management, Verification

Keywords

Service Integrity Attestation, Cloud Computing, Secure Dataflow Processing

1. INTRODUCTION

Cloud computing [1, 3] has recently emerged as a promising hosting platform that allows multiple cloud users called tenants to share a common physical computing infrastructure. With rapid adoption of the concepts of Software as a Service (SaaS) [4] and Service Oriented Architecture (SOA) [9,18], the Internet has evolved into an important service delivery infrastructure instead of merely providing host connectivity. Thus, service providers can lease a set of resources from cloud infrastructures to provide their software as services in an economical way without maintaining their own physical computing infrastructures.

Data-intensive computing [2, 17, 21, 28] has recently received much research attention with many real world applications such as security surveillance, scientific study, and business intelligence. In particular, our work focuses on dataflow processing systems [6, 21, 23] that provide high-performance continuous processing over massive data streams. Previous work on distributed dataflow processing mainly focuses on resource and performance management issues [6, 23, 29, 34]. It usually assumes that all data processing components are trustworthy. This assumption generally holds for small-scale closed cluster systems. However, in multi-tenant cloud computing infrastructures consisting of different service providers, we can no longer assume that all processing components are trustworthy. For example, dataflow processing components may include security holes that can be exploited by attackers. Attackers can also pretend to be legitimate service providers to compromise dataflow processing. One of the top security concerns for cloud users is to verify the integrity of data processing results, especially for critical data processing applications such as fraud detection and business intelligence. Note that confidentiality and user data privacy are important orthogonal issues addressed by previous work [8,27,43,44].

Trust management for large-scale systems has been studied under different contexts such as observable Byzantine faults detection for distributed messaging systems [26] and wide-area systems [10], security guards for publish-subscribe systems [39], TVDc [13] for virtualized datacenters, and remote attestation to identify compromised service components using trusted hardware or secure kernel co-existed with the remote software platform [19, 20, 37, 38]. Different from previous work, our research focuses on providing light-weight application-level attestation methods to dynamically verify the integrity of dataflow processing services provisioned through multi-party cloud computing infrastructures. We aim at achieving a practical integrity attestation technique for large-scale cloud infrastructures without requiring application modifications or assuming a trusted entity at third-party service provider sites.

In this paper, we present *RunTest*, a light-weight application-level attestation scheme that can dynamically verify the integrity of data processing results in the cloud infrastructure and pinpoint malicious service providers when inconsistent results are detected. We validate service integrity by aggregating and analyzing *result consistency information* rather than comparing memory footprints of code execution as used by code attestation [38]. Thus, our approach does not require trusted hardware or secure kernel co-existed with third-party service providers in the cloud. The rationale behind our approach is that dataflow processing applications are mostly concerned about the accuracy of *final* data results instead of the in-

ASIACCS'10 April 13–16, 2010, Beijing, China.

tegrity of the code execution. One advantage of our approach is that we can easily provide *runtime continuous integrity attestation* that is highly desired by dataflow processing applications. Different from traditional consensus-based Byzantine fault detection schemes [32, 40], our approach does not rely on full time majority voting on all service nodes, which falls short for cloud infrastructures in terms of scalability. In contrast, our scheme leverages the unique features of dataflow processing to perform *randomized attestation* using a small subset of input data tuples over different subsets of cloud nodes. To the best of our knowledge, our work makes the first attempt to provide efficient runtime integrity attestation scheme for dataflow processing in the cloud infrastructure. Specifically, this paper makes the following contributions:

- We provide a new runtime service integrity attestation scheme that employs a novel *attestation graph model* to capture attestation results among different cloud nodes. We design a clique based attestation graph analysis algorithm to pinpoint malicious service providers and recognize colluding attack patterns. Our scheme can achieve runtime integrity attestation for cloud dataflow processing services using a small number of attestation data.
- We have implemented the RunTest system within IBM System S dataflow processing system [21] and tested it on NCSU virtual computing lab (VCL) [5], a production virtual cloud infrastructure. The prototype implementation indicates that our scheme can be easily integrated into cloud dataflow processing system.
- We have conducted both analytical study and experimental evaluation to quantify the performance of the RunTest system. Our analytical study and experiment results show that RunTest can ensure the integrity of dataflow processing while imposing low performance overhead.

The rest of the paper is organized as follows. Section 2 gives a brief background about cloud computing and dataflow processing service model followed by the integrity attack model considered by our work. Section 3 presents the design and algorithms of the RunTest system. Section 4 presents the analytical evaluation about the integrity assurance and data result quality achieved by our runtime attestation scheme. Section 5 presents the prototype implementation and experimental results. Section 6 compares our work with related work. Finally, the paper concludes in Section 7.

2. PRELIMINARY

In this section, we first give a brief background overview about the cloud computing infrastructure and data-intensive computing applications that can be delivered as services via the cloud infrastructure. We then describe the integrity attack scenarios that are addressed by our work.

2.1 Multi-Tenant Cloud Infrastructures

Cloud infrastructures [1, 3] support *multi-tenant* resource sharing, which allows different cloud users to safely share a common physical infrastructure, illustrated by Figure 1. We use Amazon Elastic Cloud Computing (EC2) [1] as an example to illustrate the basic idea of resource leasing in the cloud computing. In EC2, the cloud system leases several physical hosts running a set of virtual machines (VMs) to each user. The users can run any applications within the VM (e.g., data mining, web services) and will be charged by the exact cloud resources (CPU cycles, disk storage) they use. The benefit of cloud computing is that individual users do not need to maintain their own physical infrastructure that is often very costly.

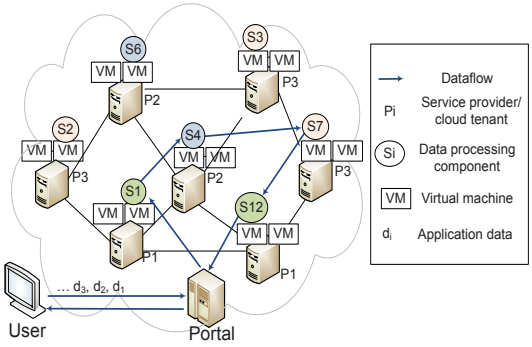

Figure 1: A cloud infrastructure shared by three tenants: P_1, P_2, and P_3.

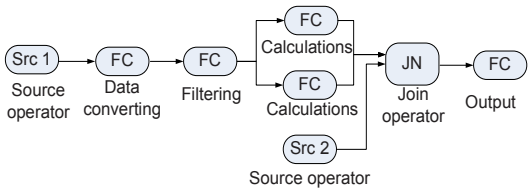

Figure 2: Dataflow processing application example.

Service oriented architecture [9] (SOA) allows service providers to deliver their software as services. Thus, the cloud infrastructure provides a natural platform for different service providers to deliver their software as services in an economical way. Each service provider p_i can lease a set of resources from the cloud to deliver its software services. We define a service component s_i as a self-contained software unit providing a certain service function f_i such as data filtering. Each service component can have one or more input ports for receiving input data tuples, denoted by d_i, and one or more output ports to emit output data tuples. For quality-of-service (QoS) management, we use a vector $Q^{s_i} = [q_1^{s_i}, ..., q_m^{s_i}]$ to characterize the dynamic QoS metrics of the component such as dataflow processing delay and loss rate. There also exist some portal service providers that can aggregate different service components into composite services based on the user's service function requirements [24, 25, 36].

2.2 Dataflow Processing in Clouds

Data processing systems (e.g., Google's MapReduce [17], Yahoo's Hadoop [2], IBM System S [21], Microsoft Dryad [28]) have become increasingly popular with applications in many domains such as business intelligence, security surveillance, and scientific computing. In particular, our work focuses on continuous dataflow processing applications [21, 23], which supports high performance in-memory data processing. For example, Figure 2 shows an example of dataflow processing application. The application reads data from NOAA (the National Oceanic and Atmospheric Administration) that reports weather conditions from weather stations throughout the world, and generates the most recent weather information for different locations where the fruit suppliers are located. Produced results will help in making decisions on whether to purchase fruit from a supplier. The raw data tuples are first converted into a set of structured streams of weather conditions information, and then are filtered to keep data only for some specific locations. Then

the next step is to perform some conversions and calculations on the weather data for different locations respectively. The weather data will then be joined with the supplier location data to generate the required output.

The user can request dataflow processing services from the cloud infrastructure through some portal nodes. A data processing application can be specified using a dataflow template denoting required dataflow processing functions and inter-function dependencies and desired QoS levels. The portal node accepts input data from users and delivers final results to the user. The portal sends *source tuples* received from the user to the first-hop component. Each component emits intermediate result tuples called *derived data*. Finally, the last-hop service component reports the final results to the portal node that forwards the final results to the user.

2.3 Attack Model

Malicious attackers can launch security attacks to a cloud infrastructure by pretending to be legitimate service providers or taking control of vulnerable service providers. Our work focuses on detecting integrity attacks to cloud dataflow processing where a malicious (or compromised) service node gives untruthful data processing results. Compared to user data privacy and confidentiality, integrity assurance is the most prevalent, which is needed no matter whether public or private data are processed by the cloud.

We assume that dataflow processing services are input-deterministic, that is, given the same input, a benign node always produces the same output. We also assume data processing services are stateless. Many dataflow processing functions fall into this category, such as selection, filtering, and mapping [2, 21].

Unlike standalone web servers or clusters, cloud infrastructures comprise a large number of distributed service provisioning nodes. We must consider colluding attack scenarios when multiple malicious attackers collude or multiple service sites are simultaneously compromised and controlled by a single malicious attacker. We assume that malicious nodes have no knowledge of other nodes except those they interact with for data receiving and forwarding or their colluding parties. However, attackers can communicate with their colluders in any arbitrary way.

Given a specific data processing function, suppose there are w malicious service components $A = \{m_1, m_2, ..., m_w\}$. For each attacker m_i, it has a set of colluders, which is a subset of A, denoted by L. Given an input tuple, if m_i ($1 \leq i \leq k$) is the first attacker among its colluders to receive this tuple, m_i has b_i ($0 < b_i \leq 1$) probability to misbehave on this tuple. If m_i is not the first one, it has c_i ($0 \leq c_i \leq 1$) probability to collude with its colluders by giving the same results with what their colluders have given, and $1 - c_i$ to behave independently. We define that two malicious components are *non-concluding* if they give *inconsistent* results on the same input. Further, if a malicious service component always gives correct data processing results (i.e., misbehaving probability $b_i = 0$), we say that the component's integrity attack fails since the accuracy of data processing results is not impacted. Thus, we can use parameters b_i and c_i to represent an attacker's behavior.

Malicious components have various strategies to choose from: they can always misbehave or probabilistically misbehave; they can collude with their colluders in different degrees and in different ways. We characterize all possible attack scenarios using different combinations of parameters (b_i, c_i) and classify those attacks into five attack patterns.

- **Non-Collusion Always Misbehave (NCAM)**. Malicious components always act independently and always give incorrect results. It corresponds to $b_i = 1$ and $c_i = 0$.
- **Non-Collusion Probabilistically Misbehave (NCPM)**. Malicious components always act independently and give incorrect results probabilistically with probability less than 1. Different malicious components may have different misbehaving probability b_i. It corresponds to $0 < b_i < 1$ and $c_i = 0$.
- **Full Time Full Collusion (FTFC)**. Malicious components always collude and always give the same incorrect results, corresponding to $b_i = 1$, and $c_i = 1$.
- **Partial Time Full Collusion (PTFC)**. Malicious components always collude and give the same incorrect results on selected tuples, corresponding to $0 < b_i < 1$ and $c_i = 1$. Malicious components cannot arbitrarily select tuples to misbehave, since they only have a local view and do not know if the selected tuple has already passed through a benign component or not. The composer is bound to detect the existence of malicious behavior if it receives different results for the same tuple. In order to reduce the risk of being detected, malicious components can select tuples to misbehave based on a pre-agreement. For example, colluding malicious components may choose to misbehave if the received tuple has an even number sequence number.
- **Partial Time Partial Collusion (PTPC)**. Malicious components sometimes collude and sometimes act independently. It corresponds to $0 < b_i < 1$ and $0 < c_i < 1$.

3. DESIGN AND ALGORITHMS

In this section, we first give an overview of our approach. Then, we describe the integrity attestation graph model that serves as the basis for our integrity attack analysis. We then describe a clique-based malicious node pinpointing algorithm followed by the attack pattern recognition algorithm for identifying attack colluding behavior in large-scale cloud infrastructures.

3.1 Approach Overview

The design objectives of the RunTest system are to identify untruthful dataflow processing results, pinpointing malicious data processing service providers, and detecting colluding attack behavior. To achieve these goals, we propose a new *integrity attestation graph* model to capture aggregated cross-node integrity attestation results, which includes the statistical output consistency/inconsistency information from different dataflow processing nodes. By analyzing the features of weighted attestation graphs, we can comprehensively verify the integrity of different dataflow processing results produced by the cloud infrastructure and pinpoint malicious service nodes in the cloud.

The RunTest system dynamically induces the weighted integrity attestation graph through *randomized data attestation*. When a tuple d first enters the cloud, the portal node first sends the data to a pre-defined dataflow path $p_1 \rightarrow p_2 ... \rightarrow p_l$ providing functions $f_1 \rightarrow f_2 ... \rightarrow f_l$. After the portal receives the processing result for d, the portal may decide to perform integrity attestation with a certain probability p_u. The portal performs integrity attestation by sending a duplicate of d to an alternative dataflow paths such as $p'_1 \rightarrow p'_2 ... \rightarrow p'_l$, where p'_i provides the same data processing function f_i as p_i. The portal may send multiple duplicates of d to perform concurrent attestation. For distributed dataflow processing applications, we verify both intermediate and final processing results to pinpoint malicious processing nodes. In order to achieve non-repudiation, each service provider is required to keep the se-

cure hash values of its input and output as evidence and sign the hash values with its private key.

We intentionally decouple the normal data processing phase from the attestation phase to prevent malicious attackers from detecting and escaping our attestation scheme. If we send attestation data together with the original data d, the malicious attacker can decide to cheat when it is sure that no attestation is triggered for d or when all attestation data are sent to its colluders. As a result, as long as the attestation is not employed for every tuple all the time, the malicious attackers can compromise the integrity of dataflow processing without being detected. In contrast, under our decoupled attestation scheme, the malicious attackers cannot avoid the risk of being detected when they produce false results on the original data d. Although the decoupled attestation scheme may cause delay in a single tuple processing, we can overlap the attestation and normal processing of consecutive tuples in the dataflow to hide the attestation delay from the cloud user.

After receiving the attestation results, the portal compares each intermediate result between pairs of functionally equivalent nodes p_i and p_i'. If p_i and p_i' receive the same input data but produce different output results, we say that p_i and p_i' are *inconsistent* with regard to function f_i. Otherwise, we say that p_i and p_i' are *consistent* with regard to function f_i. For each pair, the portal maintains counters of consistencies and inconsistencies in order to compute the weight of the corresponding edge in the attestation graph. The portal updates the counters and the weights each time when it receives attestation data. The attestation graph captures aggregated attestation results over a period of time.

The RunTest system then performs comprehensive analysis over the attestation graphs to pinpoint malicious service providers and identify untruthful results. Particularly, we look for attestation graph patterns (e.g., number of cliques, weights of non-clique edges) to identify malicious service providers and their collusion behavior. Furthermore, our scheme can also give accurate estimation about the quality of data (QoD) provided by the cloud dataflow processing services.

3.2 Integrity Attestation Graph

In order to detect service integrity attack, we employ data attestation on data processing service nodes. In contrast to code attestation schemes (e.g., [22, 37, 41, 42]), which often require special hardware/software to verify the code running on a system, data attestation verifies the integrity of service function by feeding same input data into redundant service nodes and comparing output data. The results of data attestation are in the form of consistency or inconsistency relationships, based on the comparison of output data on the same input data. We use the *integrity attestation graph* to aggregate individual data attestation results. Thus, we can achieve more efficient and scalable attack detection for dataflow processing in large-scale cloud infrastructures. We formally define the integrity attestation graph as follows,

Definition 1: An *Integrity Attestation Graph* is a weighted undirected graph, with all the attested service providers (or cloud nodes) as the vertex set and consistency/inconsistency relationships as the edges. Each edge in the attestation graph is associated with a weight. The *Weight* on an attestation graph edge, denoted as w ($0 \leq w \leq 1$), is the fraction of consistent output out of all output data between two service providers given the same input data.

Thus, if $w = 1$, it means that the two service providers always agree with each other. If $w = 0$, it means the two service providers never agree with each other. In particular, we define consistency pairs and inconsistency pairs as follows,

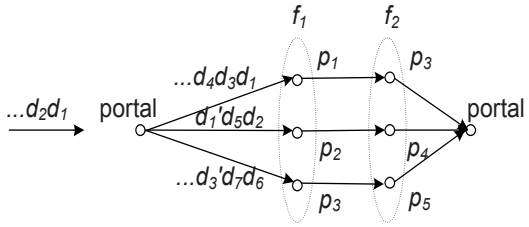

Figure 3: Dataflow processing application example.

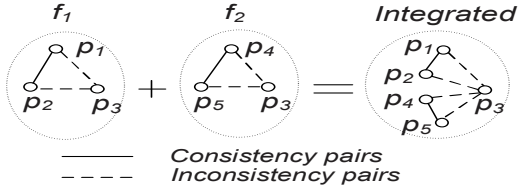

Figure 4: Integrity attestation graph example.

Definition 2: A *Consistency Pair* is a pair of service providers, which always give the same output on the same input data. An *Inconsistency Pair* is a pair of service providers, which give at least one inconsistent output on the same input data. Thus, the edge between a consistency pair has weight $w = 1$. The edge between an inconsistency pair has weight $w < 1$.

Figure 3 and Figure 4 show an example dataflow application and its corresponding attestation graph. Note that in Figure 3, only a small subset of input data will undergo the attestation process. Each attestation process only involves a subset of service providers. However, over a period of time, all service providers will be attested for at least once. Hence, our scheme can detect malicious attacks to dataflow processing in the cloud without losing scalability. For example, in Figure 3, d_1 is selected for attestation, and the attestation data d_1' is sent to p_2. Similarly, d_3 is selected for attestation and the attestation data d_3' is sent to p_3. Thus, p_1, p_2 and p_3 are all attested through different tuples. Figure 4 shows the generated attestation graphs. We use solid lines to represent consistency pairs, and dotted lines for inconsistency pairs in the attestation graph. For function f_1, the attestation graph contains one consistency pair (p_1, p_2) as well as two inconsistency pairs (p_1, p_3) and (p_2, p_3). For function f_2, the attestation graph contains one consistency pair (p_4, p_5) as well as two inconsistency pairs (p_3, p_4) and (p_3, p_5). Note that service provider p_3 provides both function f_1 and f_2. We can also generate an integrated attestation graph by combining all per-function attestation graphs, as shown by the rightmost graph. In the rest of the paper, we use attestation graph to refer to per-function attestation graph.

We propose to perform comprehensive analysis over the attestation graph to detect malicious service providers and identify untruthful data processing results. We would like to identify those representative attestation graph features, called *attestation graph motifs*, which can distinguish benign service providers from malicious attackers. Particularly, we observe that any two benign service providers always give the same results for the same input. In the attestation graph, this is expressed by an edge between them with weight one. We further define consistency clique as follows,

Definition 3: A *Consistency Clique* is a complete subgraph of an attestation graph such that 1) it has at least two nodes; 2) the weights of all edges are one (i.e., $w = 1$); and 3) it is maximal, that is, it is not contained in any other complete subgraph where

the weights of all edges are one.

If all pairs of service providers provisioning a particular data processing function have been attested, we can make the following proposition:

Proposition 1: All benign service providers always form a consistency clique in the integrity attestation graph.

Proof: we prove the above proposition by contradiction. Suppose there is a benign service node p_{i+1} that is not in the consistency clique formed by other benign service nodes $p_1, p_2, ..., p_i$. Then there must be at least one weight 1 edge missing from p_{i+1} to one of the benign nodes in the clique, say p_j ($1 \leq j \leq i$). Since pair (p_{i+1}, p_j) has been attested together and they both give correct results, they must have given the same results on all the common tuples. Thus, there should be a solid line edge of weight 1 between them. This contradicts with the previous assumption that there is no weight 1 edge between p_{i+1} and p_j.

3.3 Pinpointing Malicious Service Providers

Although our attestation scheme randomly attests a subset of all providers at a time, randomized attestation over a stream of data can cover all service providers over a period of time. In a large-scale cloud system, it is reasonable to assume that for any given data processing function, the number of benign providers is larger than that of malicious ones. Suppose there are k nodes in the attestation graph. Since all benign service providers always form a clique in the attestation graph (Proposition 1), we can claim that, at any time, a benign node must be in at least one clique with size larger than $\lfloor k/2 \rfloor$. Therefore, we can make the following proposition to pinpoint malicious nodes.

Proposition 2: Any node that is outside of all maximal cliques of size larger than $\lfloor k/2 \rfloor$ in a per-function attestation graph must be a malicious node.

Proof: we prove the above proposition by contradiction. Suppose there is a benign node p_{i+1} that is not in any of the maximal cliques of size larger than $\lfloor k/2 \rfloor$. According to Proposition 1, the benign node forms a clique with all other benign nodes. Since the number of benign nodes is larger than $\lfloor k/2 \rfloor$, node p_{i+1} is in a clique of size larger than $\lfloor k/2 \rfloor$. So, it is in a maximal clique of size larger than $\lfloor k/2 \rfloor$. This contradicts with the assumption.

Initially, all nodes are treated as benign nodes and stay in a single clique. As a malicious node keeps misbehaving, it will produce inconsistent results with that of benign nodes sooner or later through attestation, and thus gets excluded from the clique it stayed before. The malicious node either remains in a downsized clique or becomes an isolated node. When the malicious node is pushed away from any of the cliques with size larger than $\lfloor k/2 \rfloor$, it will be pinpointed as malicious. Ultimately, there will be only one clique with size larger than $\lfloor k/2 \rfloor$ in the per-function integrity attestation graph, which is formed by all benign nodes. This clique is the maximum clique in the attestation graph. All other cliques, if there is any, should have size less than $\lceil k/2 \rceil$.

Thus, pinpointing malicious nodes becomes the problem of finding consistency cliques in the attestation graph. We adapt the well-known Bron-Kerbosch (BK) clique finding algorithm [14, 15, 31] for finding consistency cliques in the attestation graph. We maintain three disjoint sets of nodes R, P, and X:

The set R stands for the currently growing clique, i.e. the set to be extended by a new node or shrunk by one node on traveling along a branch of the backtracking tree, and R is initialized to be $\emptyset$;

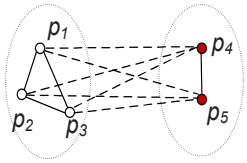

Figure 5: An example of finding maximal cliques.

The set P stands for prospective nodes which are connected to all nodes in R and using which R can be expanded, and P is initialized to contain all nodes;

The set X contains nodes already processed, i.e. the nodes that were previously in P and hence all maximal cliques containing them have already been reported, and X is initialized to be $\emptyset$. Maintaining the set X is because X being empty is one of the conditions to claim the current R contains a maximal clique. If X is not empty, R may be extended by considering nodes in X.

Note that all nodes that are connected to every node of R are either in P or X. The algorithm runs as traversing the recursion tree by moving nodes from P to R and updating the R, P, X sets recursively. A maximal clique is reported when both P and X are empty. The heuristic of the pivot selection is based on the identification and elimination of equal sub-trees appearing in different branches of the recursion tree which lead to the formation of non-maximal cliques.

We use an example to explain the algorithm shown by Figure 5. There are three benign nodes $\{p_1, p_2, p_3\}$ and two malicious nodes $\{p_4, p_5\}$ in the attestation graph. Based on our consistency clique definition, we only count consistency links as connections between any two nodes. Initially, the current clique node set $R = \emptyset$, and the candidate set $P = \{p_1, p_2, p_3, p_4, p_5\}$. We randomly take one of the nodes in P, say p_4 as the pivot node. Thus, we can move those nodes that are not the neighbors of p_4 from P to R. In this example, we can move the candidate nodes p_1, p_2, p_3 but not p_5 since p_5 is the neighbor of the pivot node p_4. Suppose we first move p_1 to R. Then, $R = \{p_1\}$, and we update the candidate set P to include the nodes that are connected to all node(s) in R, which should be $P = \{p_2, p_3\}$. X is the set containing already processed nodes with regard to the node currently under consideration. So the update of X is to shrink it so that it only contains nodes that has connections with p_1. In this case, X is $\emptyset$ and no need to be shrunk. Next, we use the new R, P, X in recursion to explore a maximal clique. We move another node p_2 from P to R to expand the clique, then updating R, P, and X into $R = \{p_1, p_2\}$, $P = \{p_3\}$, and $X = \emptyset$. Similarly, we move p_3 from P to R, and have $R = \{p_1, p_2, p_3\}$, $P = \emptyset$, and $X = \emptyset$. By now, we have identified a clique $R = \{p_1, p_2, p_3\}$ since both P and X become empty. After returning from the recursions, p_3, p_2, p_1 will all be added to X respectively, since they are processed nodes. Note that the usefulness of X would be clearer when there are other nodes connected to only a part of the currently forming clique, which is not presented in this simple example. Now, we start from a different pivot node, say p_1. We have $R = \emptyset$, $P = \{p_1, p_2, p_3, p_4, p_5\}$, $X = \{p_1, p_2, p_3\}$. We can move those nodes that are not the neighbors of p_1 from P to R, which include p_4 and p_5. Suppose we first move p_4 to R and then update R, P, and X into $R = \{p_4\}$, $P = \{p_5\}$, and $X = \emptyset$. Next, we can move p_5 from P to R since it is not the neighbor of the pivot p_1. Thus, we have $R = \{p_4, p_5\}$, $P = \emptyset$, and $X = \emptyset$, so that we identify another clique $\{p_4, p_5\}$.

Generally, a maximal clique is a complete subgraph that is not contained in any other complete subgraph. Among all cliques, the

AdaptiveBK(G)

1. Initialization 1: Mark any two nodes with $w < 1$ edge as unconnected, and with $w = 1$ edge as connected;
2. Initialization 2: Eliminate nodes that do not have any edge of $w = 1$
3. FindConsistencyClique($\emptyset, V(G), \emptyset$), where $V(G)$ is the node set of G

FindConsistencyClique(R, P, X)

1. if ($P == \emptyset$ and $X == \emptyset$ and size of $R > 1$)
2. Report R as a maximal clique
3. else
4. Let u_p be the pivot node
5. Assume $P = u_1, u_2, ..., u_k$
6. for i =1 to k do
7. if u_i is not a neighbor of u_p
8. $P = P - u_i$
9. $R_{new} = R \cup u_i$
10. $P_{new} = P \cap N[u_i]$, where $N[u_i]$ is neighbor set of u_i
11. $X_{new} = X \cap N[u_i]$
12. FindConsistencyClique($R_{new}, P_{new}, X_{new}$)
13. $X = X \cup u_i$

Figure 6: Consistency clique discovery algorithm.

largest one is the maximum clique. The maximum clique problem is one of the canonical NP-complete problems, while the problem of enumerating all cliques in a graph is NP-hard. The complexity of the BK algorithm increases with the number of cliques in the graph. However, in practice, the number of cliques in the attestation graph is very small. Furthermore, in this paper, we add two requirements for cliques: 1) The clique contains at least two nodes; and 2) weights of all edges are one. These two features can help us eliminate some nodes that do not satisfy the criteria in O($n + e$) time, with e being number of edges. Thus, we extend the BK algorithm by first reducing the attestation graph through eliminating a subset of nodes. Nodes without weight 1 edge cannot be in a clique. By going through each node in the graph at the beginning, such type of nodes can be eliminated, and the complexity of the BK algorithm is reduced. Figure 6 shows the pseudo-code of our consistency clique finding algorithm. The algorithm takes the attestation graph G as input, and returns all maximal cliques.

3.4 Identifying Attack Patterns

A large-scale cloud computing infrastructure often consists of many distributed service provisioning nodes. When multiple malicious attackers collude or multiple cloud nodes are compromised and controlled by a single malicious attacker, the cloud infrastructure will be exposed to collusive integrity attacks. Thus, in addition to pinpointing those malicious nodes, it is also important to discover colluding behavior among different nodes, which can assist in cloud security diagnosis. A nice feature of our integrity attestation graph is that it can not only expose malicious nodes but also reveal collusive attack patterns. We summarize the features of the corresponding attestation graphs for each attack pattern, and express them using attestation graph motifs. By the law of large numbers, when enough attestation outputs are observed, the attestation graph motifs of all possible integrity attack patterns can be described as follows, which are shown in Figures 7 - 10. Note that

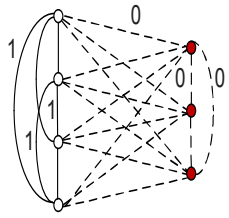

Figure 7: Attestation graph motif A.

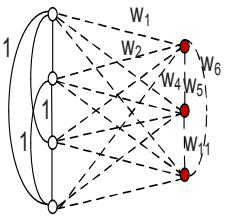

Figure 8: Attestation graph motif B.

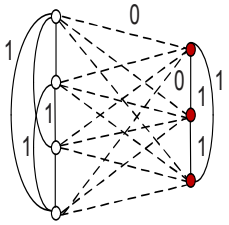

Figure 9: Attestation graph motif C.

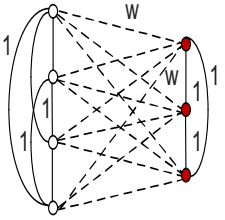

Figure 10: Attestation graph motif D.

we begin to analyze attack patterns after we find only one clique of size larger than $\lfloor k/2 \rfloor$.

Attestation graph motif A: If malicious nodes are not in collusion and always misbehave, the portal would find different results from different malicious nodes. In the attestation graph, malicious nodes would be isolated nodes in terms of consistency relationship. And in terms of inconsistency relationship, they are the nodes that have the maximal degrees. In other words, the weights on all the edges involving malicious nodes are all zero. Figure 7 shows an example of attestation graph motif A. This scenario corresponds to the Non-Collusion Always Misbehave (NCAM) attack pattern.

Attestation graph motif B: If malicious nodes misbehave probabilistically and independently, they would agree with benign nodes and even well-behaved malicious nodes on some attestation data. This scenario corresponds to the Non-Collusion Probabilistically Misbehave (NCPM) attack pattern. Alternatively, malicious nodes may collude sometimes and act independently for the rest of time, which corresponds to Partial Time Partial Collustion (PTPC) attack pattern. In both cases, the weights on the attestation graph edges involving malicious nodes appear to be randomized. Figure 8 shows an example of the attestation graph motif B.

Attestation graph motif C: If all malicious nodes collude together and always give the same incorrect results, malicious nodes also form a consistency clique among themselves in the attestation graph. However, the weights on the edges involving one malicious node and one benign node are all zero. Figure 9 shows an example the attestation graph motif C. This attestation graph motif can capture the Full Time Full Collusion (FTFC) attack pattern.

Attestation graph motif D: If all malicious nodes collude together but only on selected tuples based on a pre-agreement, malicious nodes also form a clique. Each malicious node connects to benign nodes with the same weight of w. Figure 10 shows an example attestation graph motif D. This scenario corresponds to the Partial Time Full Collusion (PTFC) attack pattern.

From the attestation graph motifs, it can be observed that different attack patterns can be distinguished according to two criteria: (a) number of cliques; and (b) the weight patterns of non-clique edges. Thus, we can identify attack patterns using the following rules: (1) Depending on number of cliques in an attestation graph,

IdentifyMaliciousNodes(G)

1. find all maximal cliques CL_i ($1 \leq i < k$) in G using adapted Bron-Kerbosch algorithm with pivot selection
2. in CL_i, find those maximal cliques with size larger than $\lfloor k/2 \rfloor$, CL_b ($b \leq i$), where k is the total number of nodes in G
3. check all nodes $Node_j$ in G against nodes in all CL_b
4. **if** ($Node_j$ is not in any of CL_b)
5. $Node_j$ is malicious
6. **if** (only one maximal clique in CL_b, i.e.,the maximum clique)
7. **if** (numCliques == 1)
8. nodes in the clique is identified as N_i
9. **if** (weights of edges from nodes in N_i to rest of nodes not in N_i are all 0s)
10. attack model NCAM
11. **else**
12. attack model NCPM or PTPC
13. **if** (numCliques $\geq$ 2)
14. nodes in the maximum clique are N_i, in the rest cliques are N_{1i}, N_{2i}, ...
15. check each clique other than the maximum clique
16. **if** (weights from N_i to any of N_{1i}, N_{2i}, ... are all 0s)
17. attack model FTFC
18. **else if** (all links between N_i and N_{ji} have same weight)
19. attack model PTFC

Figure 11: Cloud Dataflow Integrity attack detection algorithm.

we can divide the attack patterns into two groups. One group includes NCAM, NCPM, and PTPC, which contains only one clique. And the other group includes FTFC and PTFC. (2) Depending on the weights on non-clique edges, we can further identify individual attack patterns within each group. For example, in the first group, if weights are all zero, the attack can only be NCAM. However, in the second group, if weights are all zero, attack pattern is FTFC. Note that the algorithm cannot distinguish NCPM and PTPC, since their attestation graphs have the same features. This is acceptable because NCPM is a special case of PTPC, where the parameter controlling collusion goes from sometimes to never.

Figure 11 shows the pseudo code of our algorithm to identify attack patterns and malicious service nodes. The algorithm takes the attestation graph G as input, and outputs suspected malicious nodes. First, the algorithm finds all cliques in the attestation graph using the adapted BK algorithm. Second, it checks nodes against the identified cliques to pinpoint malicious service nodes. Finally, when there is only one clique of size larger than half of the total number of nodes, the algorithm identifies attack patterns. By checking the number of cliques in the attestation graph and the weights on non-clique edges one by one, the algorithm identifies specific attack patterns. Note that the algorithm needs to be started when all pairs of functionally equivalent nodes have been attested in order to assure that all benign nodes have showed up in the maximum clique.

4. SECURITY ANALYSIS

4.1 Security Properties

Our scheme of pinpointing malicious service providers is based on attestation graph analysis. We claim that the scheme preserves the following properties:

Property 1: No false positive: a benign service provider will not be pinpointed as malicious.

Proof: As proved for our Proposition 2, a benign node always stays in at least one clique of size larger than $\lfloor k/2 \rfloor$. Therefore, a benign node will not be treated as malicious since our algorithm only pinpoints a node when the node is outside of any maximal cliques with size larger than $\lfloor k/2 \rfloor$.

Property 2: Non-Repudiation: for any pinpointed malicious service provider, the trusted portal node can present evidence to prove it is malicious.

Proof: The data received by the portal contains the signed hash value of intermediate processing results provided by service providers. The portal can present proof that shows the pinpointed node is inconsistent with more than $\lfloor k/2 \rfloor$ of its functionally equivalent nodes. According to our algorithm, a pinpointed node is not in any of the maximal cliques with size larger than $\lfloor k/2 \rfloor$, which means it has inconsistent results with more than $\lfloor k/2 \rfloor$ nodes. Since the number of malicious nodes is less than or equal to $\lfloor k/2 \rfloor$, the pinpointed node must have been inconsistent with a benign node. Thus, it must be malicious.

Note that our scheme has false negative, since the randomized data attestation cannot capture the misbehavior of malicious service providers if they accidentally only misbehave on non-duplicated tuples or all our attestation are conducted on collusive service providers. However, since our scheme ensures that malicious nodes cannot predict when they will be attested, the probability of detecting misbehavior through observing inconsistent data results will increase accordingly as more attestations are performed. Thus, our runtime attestation scheme can eventually identify all misbehaving malicious nodes as data tuples continuously flow into the system.

4.2 Data Quality

We define data quality as the percentage of processed data with correct results. Our scheme can detect tampered data results probabilistically and report data quality close to actual data quality.

Suppose the total number of unique data is n, and the number of tampered unique data is s. Thus, the actual data quality, denoted by Q_a, equals to $1 - s/n$. In contrast, our reported data quality, denoted by Q_r, can be computed as the percentage of data that we believe has correct results. For non-duplicated data, we count them as non-tampered since we are not sure whether it is tampered. For duplicated data, we also count it as non-tampered if we get consistent results. If we get inconsistent results, we count it as tampered. The expected number of unique data that gets duplicated for attestation is np_u. We suppose the number of data that gets duplicated but with inconsistent results is c. Thus, the reported data quality becomes $((n - np_u) + (np_u - c))/n$, which turns out to be $1 - c/n$. That is,

$$Q_r = 1 - \frac{c}{n} \tag{1}$$

where c can be observed and recorded by the portal through data attestation. It can be seen that $c < s$. The data quality reporting of our scheme has false negatives but without false positives.

We can introduce auto-correction mechanism to correct tampered data results. By checking visited components against the pinpointed malicious component set, we can either select data result from existing results processed by only benign components or process the data again through only benign components. After pinpointing a malicious component, we can also eliminate it from the system and thus improve data quality extensively.

5. EXPERIMENTAL EVALUATION

5.1 System Implementation

We have implemented our runtime integrity attestation scheme in C++ within IBM System S dataflow processing system [21]. The dataflow processing system is a high-performance continuous stream processing platform that can handle high-rate data streams and scale to hundreds of processing components. Our experiment is conducted on NCSU virtual computing lab [5] consisting of several hundreds of blade servers, which provides similar virtual cloud resources as Amazon EC2 [1]. We use about 10 blade servers in our experiments. Each host runs CentOS 5.2 64-bit with Xen 3.0.3 for managing virtual machines.

The dataflow processing application we use is extracted from the sample application, illustrated by Figure 2, provided by the IBM System S dataflow processing system. The application takes real data about weather information as input, performs conversions and calculations on the weather data, and generates the most recent weather information for different fruit suppliers. We perform attestation on three functions. Each function is provisioned by five different service providers. We have a trusted portal node that accepts a stream of application data tuples, launches randomized data attestation, and collects attestation results. For each function, the portal constructs an attestation graph for pinpointing malicious service providers and identifying attack patterns. The input data rate is 100 tuples per second.

For comparison, we have also implemented the common consensus-based integrity verification scheme *full-time majority voting* that employs all functionally redundant service providers all the time to cross validate each other. The scheme determines which service providers are malicious through majority voting. It can detect any service integrity attack behavior immediately when receiving inconsistent results with the assumption that the number of benign service providers is larger than that of malicious ones. Our scheme relies on the same assumption to pinpoint malicious service providers.

We evaluate our scheme using two important metrics: *detection rate* and *attestation overhead*. The detection rate is calculated by the number of pinpointed malicious service providers over the total number of malicious service providers that have misbehaved at least once during one experiment run. The attestation overhead is calculated by the number of duplicated data tuples that are redundantly processed for integrity attestation and the extra dataflow processing time incurred by the integrity attestation.

We evaluate the proposed scheme in three steps: 1) we evaluate the effectiveness of our service integrity verification scheme in terms of detecting malicious service providers under different attack strategies; 2) we investigate the sensitivity of our scheme to system parameter variations; 3) we compare our scheme with the full-time majority voting scheme. We show that our scheme can achieve similar attack detection performance as the full-time majority voting scheme while imposing much lower overhead.

5.2 Results and Analysis

We first evaluate the attack detection effectiveness of our scheme. We vary the percentage of malicious service providers from 20% to 40% but guarantee that for each function, the number of benign service providers is larger than that of malicious ones. After the portal receives the processing result of a new data tuple, it randomly decides whether to perform data attestation. Each tuple has 0.2 probability of getting attested (i.e., attestation probability $p_u = 0.2$), and only one attestation data is used (i.e., number of total data copies $r = 2$). Figure 12 through Figure 16 show the detection rate under

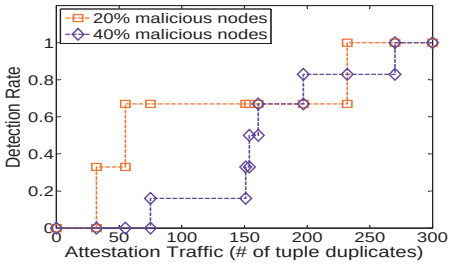

Figure 12: Service integrity attack detection rate under non-collusion scenario where $b_i = 1$ and $c_i = 0$.

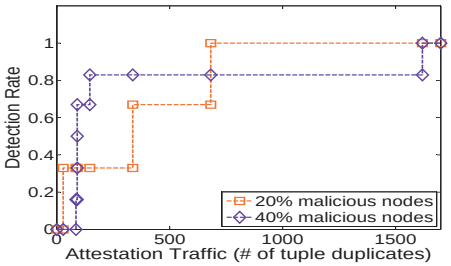

Figure 13: Service integrity attack detection rate under non-collusion scenario where $b_i = 0.2$ and $c_i = 0$.

different integrity attack scenarios. For each attack scenario, we plot the detection rates in the presence of different percentage of malicious service providers. The X axis shows the number of total attestation data used up to that point. We observe that our scheme can detect all malicious service providers using a small number of attestation data. Generally speaking, we need to use more attestation data to detect all malicious service providers under collusion attack patterns.

For non-collusion attacks, we test ($b_i = 1, c_i = 0$) and ($0 < b_i < 1, c_i = 0$) scenarios respectively. Our algorithm can correctly identify ($b_i = 1, c_i = 0$) as NCAM attack pattern, and identify ($0 < b_i < 1, c_i = 0$) as either NCPM or PTPC attack pattern. Note that our algorithm cannot distinguish NCPM and PTPC because they share the same attestation graph motif. Figure 12 is the NCAM scenario, where malicious service providers misbehave all the time and independently. As we can see, the detection rate reaches 100% earlier when there are less malicious service providers. However, the intermediate detection rate depends on the time when malicious service providers begin to misbehave as well as the time that misbehavior is captured through attestation. Figure 13 shows the NCPM scenario, where malicious service providers misbehave independently but probabilistically. They have 0.2 probability of misbehaving on a tuple (misbehaving probability b_i = 0.2). Compared with NCAM, the NCPM scenario needs longer time to reach 100% detection rate. This is because malicious service providers misbehave probabilistically, which makes it harder to catch their malicious behavior immediately.

For collusion attacks, we test ($b_i = 1, c_i = 1$), ($0 < b_i < 1, c_i = 1$), and ($0 < b_i < 1, 0 < c_i < 1$) scenarios respectively. Our algorithm correctly identifies ($b_i = 1, c_i = 1$) as FTFC, ($0 < b_i < 1, c_i = 1$) as PTFC, and ($0 < b_i < 1, 0 < c_i < 1$) as either PTPC or NCPM. Figure 14 shows the FTFC scenario, where all malicious service providers form a group and launch group colluding attacks all the time. They give the same output on a data tuple. Even though our scheme randomly selects service providers

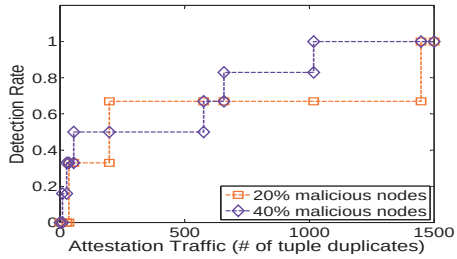

Figure 14: Service integrity attack detection rate under collusion scenario where $b_i = 1$ and $c_i = 1$.

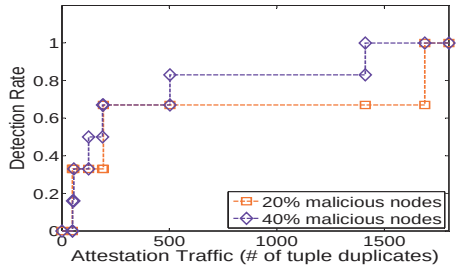

Figure 15: Service integrity attack detection rate under collusion scenario where $0 < b_i < 1$ and $c_i = 1$.

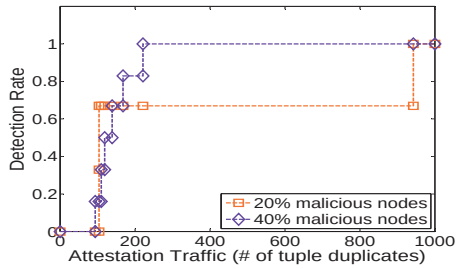

Figure 16: Service integrity attack detection rate under collusion scenario where $0 < b_i < 1$ and $0 < c_i < 1$.

for attestation, it can still achieve 100% detection rate under such colluding attacks. Figure 15 shows a more intelligent attack scenario of PTFC, where all malicious service providers are in collusion and they have pre-agreement specifying on which tuples to misbehave. They only misbehave on tuples with even-numbered sequence number, so $b_i = 0.5$. Figure 16 is the PTPC scenario, where attackers may form collusion sometimes and act independently otherwise. In the experiments, each attacker relies on a random number generator to decide whether to collude. And if they collude, they give the same results on the same input data. Our scheme can still capture such attacks and pinpoint malicious service components under such scenario.

Secondly, we investigate the impact of system parameters, such as attestation probability and malicious service providers misbehaving probability, on the effectiveness of our algorithm. We fix the percentage of malicious service providers at 20%. Figure 17 shows the detection rate under different attestation probability in $(b_i = 1, c_i = 1)$ scenario. By increasing attestation probability, attestation traffic has more opportunities to cover malicious service providers and capture their misbehavior. Thus, our scheme can reach higher detection rate earlier. However, the system overhead, in terms of attestation traffic would increase accordingly since we

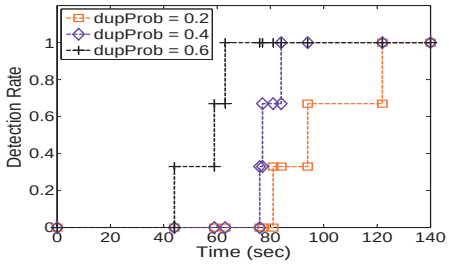

Figure 17: Detection rate under different attestation probability in $(b_i = 1, c_i = 1)$ scenario.

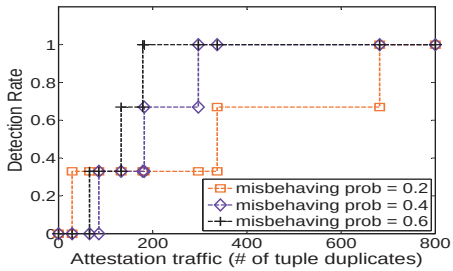

Figure 18: Detection rate under different misbehaving probability in $(0 < b_i < 1, c_i = 0)$ scenario. (attestation probability = 0.2)

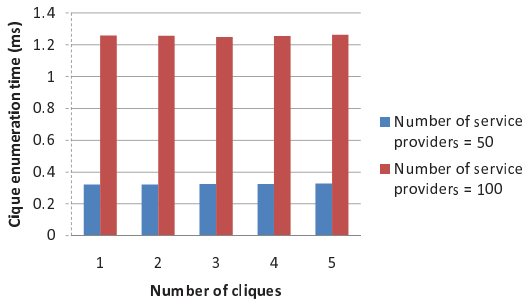

Figure 19: Computing time of clique discovery algorithm.

duplicate more tuples for attestation. Figure 18 shows the detection rate under different misbehaving probability of malicious service providers in $(0 < b_i < 1, c_i = 0)$ scenario. As we can see, the more frequently malicious service providers misbehave, the less attestation traffic and less time we use to detect them. In other words, our scheme forces attackers to slow down and reduce their attacks. Note that even with low misbehaving probability, e.g. 0.2, our scheme can pinpoint those attackers with limited attestation traffic. As long as attackers keep misbehaving, no matter how infrequently, our scheme can detect them eventually.

We also measure the computation time of the clique discovery algorithm. Figure 19 shows the total clique enumeration time of attestation graphs with 1, 2, till 5 cliques. We test it with different number of service providers in the system. It shows that given a small number of cliques, the number of cliques in the graph does not have much impact on clique enumeration time. Even with 100 service providers serving the same function, clique enumeration time measurements are within two milliseconds.

We compare actual data quality with reported data quality to evaluate whether our scheme can accurately capture data quality.

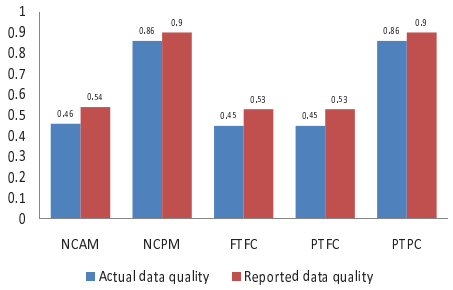

Figure 20: Comparison of actual data quality and reported data quality.

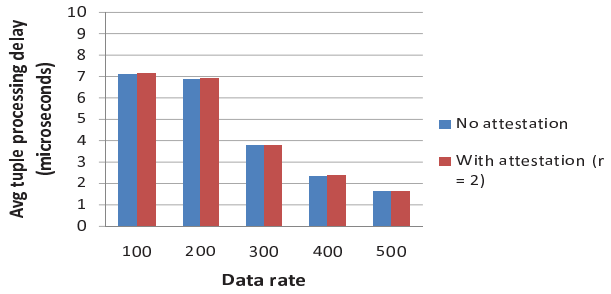

Figure 21: Dataflow processing delay comparison.

Actual data quality is defined as the percentage of processed data with correct results. The reported data quality is calculated using equation 1 by the portal. Figure 20 shows the data quality comparison under different attack patterns. It shows that for each scenario, our scheme can give a very close estimate of actual data quality.

We compare no attestation and with attestation ($r = 2$) schemes in terms of the average dataflow processing delay. The delay is measured by the average per-tuple turnaround time (i.e., the duration between the time when the first dataflow tuple enters the system and the time when the last dataflow tuple leaves the system over the total number of tuples processed). Figure 21 shows the comparison in cases of different data rate. We can see that our scheme had little delay overhead compared with the no attestation scheme.

Thirdly, we compare our scheme with a common existing scheme, full-time majority voting. Figure 22 compares the detection time of our scheme with the full-time majority voting scheme in the ($b_i = 1, c_i = 0$) attack scenario (identified as NCAM). Our scheme was run with different attestation probabilities. Each time, two functionally equivalent service providers are selected, while in the full-time majority voting scheme, it sends duplicated tuples to all the five service components. The full-time majority voting scheme has the shortest detection time. Our scheme can achieve 100% detection with a short delay, which can be shortened by increasing the duplication probability. Note that such a short delay is acceptable by dataflow processing applications, where we can mark a set of result data as tentative and commit the final correct results after the attestation [12]. Figure 23 compares the attestation overhead in terms of number of redundant tuple duplicates processed. Our scheme outperforms the naive full-time majority voting scheme with the limited overhead at the expense of only a little delay toward detecting all malicious service providers.

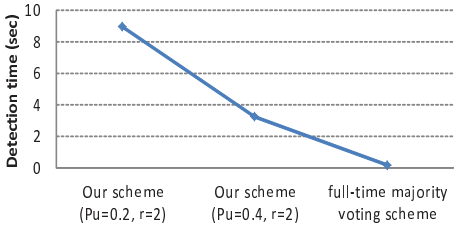

Figure 22: Detection time comparison between our scheme and the full-time majority voting scheme.

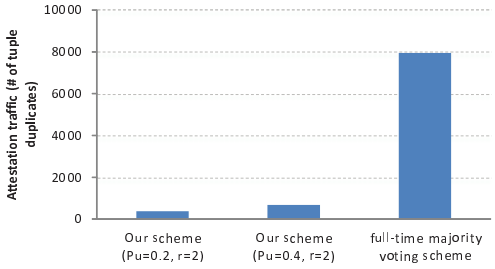

Figure 23: Attestation overhead comparison between our scheme and the full-time majority voting scheme.

6. RELATED WORK

Trust management and reputation systems for distributed systems have been studied under different contexts. In addition to those mentioned in the introduction, the EigenTrust algorithm applies a global feedback reputation system for P2P networks, and attempts to address the problem of peer collusion by assuming pretrusted peers in the network [30]. NetProbe models auction users and transactions as a Markov Random Field tuned to detect suspicious patterns that fraudsters create, and employs a belief propagation mechanism to detect likely fraudsters in online auction networks [33]. BIND provides a fine-grained code attestation scheme for distributed systems [38]. Alam et al. proposed a set of specification, hardware-based trust measurement, and verification schemes for attesting the behavior of business processes [7]. Different from previous work, our research focuses on developing application-level attestation schemes for pinpointing malicious nodes in large-scale multi-tenant cloud infrastructures.

The problem of pinpointing malicious components share some similarities with fault diagnosis in complex systems. The pioneering work is the PMC model proposed in [35]. Efficient diagnosing algorithms to identify faulty nodes are also proposed [11, 16]. They either assume permanent faults, which means a faulty node always fails a test, or assume the number of incorrect outcomes is bounded. However, for dataflow processing in cloud infrastructures, malicious components can behave arbitrarily on any input data, and collude with collaborators at will. The methods in fault diagnosis systems cannot be readily applied to cloud dataflow processing systems. Previous work on Byzantine fault detection (BFD) generally focuses on the designing of communication protocols or message/detector structures, so that a proper voting mechanism can lead to the exposure of Byzantine generals [32, 40]. However, BFD generally relies on full time majority voting scheme to detect faults, which is hardly applicable to large-scale cloud computing infrastructure due to issues related to scalability and deployment difficulty. In contrast, our scheme provides comprehensive randomized attestation to achieve both scalability and efficiency.

7. CONCLUSION

In this paper, we have presented the design and implementation of RunTest, a new service integrity attestation system for verifying the integrity of dataflow processing in multi-tenant cloud infrastructures. RunTest employs application-level randomized data attestation for pinpointing malicious dataflow processing service providers in large-scale cloud infrastructures. We propose a new integrity attestation graph model to capture aggregated data processing integrity attestation results. By analyzing the integrity attestation graph, RunTest can i) pinpoint malicious service providers, ii) identify untruthful data processing results, and iii) discovering colluding attack patterns in the large-scale cloud infrastructure. We have implemented our service integrity attestation scheme within IBM System S dataflow processing system and tested it on NCSU virtual computing lab. Our initial experimental results show that the proposed scheme is effective and imposes low performance impact for dataflow processing in cloud infrastructures.

8. ACKNOWLEDGMENT

This work was sponsored in part by U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI), NSF CNS-0915567, and NSF IIS-0430166. Any opinions expressed in this paper are those of the authors and do not necessarily reflect the views of the ARO, NSF or U.S. Government.

9. REFERENCES

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

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

[3] Microsoft Azure Services Platform. http://www.microsoft.com/azure/default.mspx.

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

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

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

[7] M. Alam, M. Nauman, X. Zhang, T. Ali, and P. C.K. Hung. Behavioral attestation for business processes. In *IEEE International Conference on Web Services*, 2009.

[8] L. Alchaal, V. Roca, and M. Habert. Managing and securing web services with vpns. In *IEEE International Conference on Web Services*, pages 236–243, San Diego, CA, June 2004.

[9] G. Alonso amd F. Casati, H. Kuno, and V. Machiraju. Web Services Concepts, Architectures and Applications Series: Data-Centric Systems and Applications. *Addison-Wesley Professional*, 2002.

[10] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling byzantine fault-tolerant systems to wide area networks. In *In The International Conference on Dependable Systems and Networks (DSN)*, 2006.

[11] T. Araki and Y. Shibata. (t,k)-diagnosable system: A generalization of the pmc models. *IEEE Trans. on Computers*, 52(7), 2003.

[12] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-Tolerance in the Borealis Distributed Stream Processing System. In *ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, 2005.

[13] 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.

[14] C. Bron and J. Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[15] F. Cazals and C. Karande. A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, 407(1-3), 2008.

[16] A. Dahbura, K. Sabnani, and L. King. The comparison approach to multiprocessor fault diagnosis. *IEEE Trans. on Computers*, C-36(3):373–378, 1987.

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

[18] T. Erl. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. *Prentice Hall*, 2005.

[19] J. Garay and L. Huelsbergen. Software integrity protection using timed executable agents. In *Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, Taiwan, March 2006.

[20] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, October 2003.

[21] B. Gedik, H. Andrade, K.-L. Wu, P. S. Yu, and M. Doo. SPADE: the system s declarative stream processing engine. *Proc. of SIGMOD*, April 2008.

[22] J. L. Griffin, T. Jaeger, R. Perez, and R. Sailer. Trusted virtual domains: Toward secure distributed services. In *Proceedings of First Workshop on Hot Topics in System Dependability*, June 2005.

[23] The STREAM Group. STREAM: The Stanford Stream Data Manager. *IEEE Data Engineering Bulletin, 26(1):19-26*, March 2003.

[24] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. *Proc. of IEEE 23nd International Conference on Distributed Computing Systems (ICDCS 2003)*, 2003.

[25] X. Gu, K. Nahrstedt, and B. Yu. SpiderNet: An Integrated Peer-to-Peer Service Composition Framework. *Proc. of IEEE International Symposium on High-Performance Distributed Computing (HPDC-13), Honolulu, Hawaii*, June 2004.

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

[27] 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.

[28] 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.

[29] N. Jain and et al. Design, Implementation, and Evaluation of the Linear Road Benchmark on the Stream Processing Core. *Proc. of SIGMOD*, 2006.

[30] 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.

[31] I. Koch. Fundamental study: Enumerating all connected maximal common subgraphs in two graphs. *Theoretical Computer Science*, 250(1-2):1–30, 2001.

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

[33] 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.

[34] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-Aware Operator Placement for Stream-Processing Systems. *Proc. of ICDE'06*, April 2006.

[35] F.P. Preparata, G. Metze, and R. T. Chien. On the connection assignment problem of diagosable systems. *IEEE Trans. on Electronic Computers*, 16(6):848–854, 1967.

[36] B. Raman and et. al. The SAHARA Model for Service Composition Across Multiple Providers. *International Conference on Pervasive Computing (Pervasive 2002)*, August 2002.

[37] 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)*, October 2005.

[38] E. Shi, A. Perrig, and L. V. Doorn. Bind: A fine-grained attestation service for secure distributed systems. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2005.

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

[40] R. Koetter T. Ho, B. Leong and et. al. Byzantine modification detection in multicast networks using randomized network coding. In *In IEEE International Symposium on Information Theory (ISIT)*, 2004.

[41] TPM Specifications Version 1.2. https://www.trustedcomputinggroup.org/downloads/specifications/tpm/tpm.

[42] Trusted computing group. https://www.trustedcomputinggroup.org/home.

[43] 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, September 2006.

[44] H. Zhang, M. Savoie, S. Campbell, S. Figuerola, G. von Bochmann, and B. S. Arnaud. Service-oriented virtual private networks for grid applications. In *IEEE International Conference on Web Services*, pages 944–951, Salt Lake City, UT, July 2007.