CADA: Collaborative Auditing for Distributed Aggregation

José Valério*, Pascal Felber*, Martin Rajman† and Etienne Rivière*
*Computer Science Department
University of Neuchâtel, Switzerland
Email: first.last@unine.ch
†EPFL, Lausanne, Switzerland
Email: martin.rajman@epfl.ch

Abstract—The aggregation of distributions, composed of the number of occurrences of each element in a set, is an operation that lies at the heart of several large-scale distributed applications. Examples include popularity tracking, recommendation systems, trust management, or popularity measurement mechanisms. These applications typically span multiple administrative domains that do not trust each other and are sensitive to biases in the distribution aggregation: the results can only be trusted if inserted values were not altered nor forged, and if nodes collecting the insertions do not arbitrarily modify the aggregation results. In order to increase the level of trust that can be granted to applications, there must be a disincentive for servers to bias the aggregation results. In this paper we present the CADA auditing mechanisms that let aggregation servers collaboratively and periodically audit one another based on probabilistic tests over server-local state. CADA differs from the existing work on accountability in that it leverages the nature of the operation being performed by the node rather than a general and application-oblivious model of the computation. The effectiveness of CADA is conveyed by an experimental evaluation that studies its ability to detect malevolent behaviors using lightweight auditing oracles.

Keywords—distributed systems; aggregation; accounting

I. INTRODUCTION

Several large-scale distributed applications rely on the aggregation of information as a central component of the application logic. Examples of such applications include monitoring, feedback aggregation [11], search mechanisms [4], [21], trust management [25], or popularity tracking and monitoring [5], [18]. More specifically, these systems rely on the aggregation of discrete distributions of data over a set of possible values. A distribution is composed of a set of <value,counter> pairs, where each possible value in the set is associated with a counter of its occurrences as aggregated from client insertions. Note that the number of options is not necessarily bounded a priori. Each distribution is associated with a unique key, itself associated to a node (or group of coordinated nodes) acting as an aggregation point. Distributed aggregation middleware helps locating this node by employing an indexing mechanism such as a multi-hop distributed hash table [22], [23] or a single-hop routing layer [9]. The node receives and aggregates insertions from clients, and answers to requests for part or all of the distribution values and counters.

Motivating examples. A simple example of the use of distributed aggregation is the tracking of relative popularities for different options in a poll. Consider a song of the year poll proposed in a media streaming application [16] implemented in a decentralized peer-to-peer manner. A similar mechanism could be used in the same streaming application to support billing and profit distribution based on the number of plays. Users of the streaming system can emit insertions to increment the counter associated with a given entry, in order to push their favorite option, or in the simplest case simply inform the system that they accessed one given element. Figure 1 presents the general mechanism associated with this aggregation.

Another example, which corresponds to the initial motivation of this work, is the support of collective search and recommendation mechanisms. The Buzzaar project\(^1\) proposes to exploit Web searches and access histories of users, collected from a simple browser extension, in order

\(^1\)http://www.buzzaar.net/wiki
to support automatic browsing suggestions. Once collected, histories are sent to a collaboratively-operated network of aggregating servers. Thereafter, the information about previous accesses by other users is used to generate navigation suggestions. These previous accesses take the form of a distribution of elements for a given query and/or previously accessed element, and are refined based on the user interests derived from her own previous searches and accesses.

Obviously, the nature of the aggregated information in these two examples is sensitive and the privacy of users must be protected. Furthermore, the risk of having the system polluted by malevolent users flooding fake insertions is high. Our previous work on the SPADS [12] system has been dealing with both aspects. Publisher anonymity is guaranteed by the use of anonymizing paths between the clients and the aggregation layer. The influence of malevolent users is reduced by means of rate limitation, enforced during the anonymizing routing operation. Figure 2 presents a high level picture of our base system: the aggregation layer encrypts the data and sends it to an aggregation server. The aggregation layer performs the aggregation and insertion operations. The actual action(s) to be taken by the system once a suspicious activity is detected is outside the scope of this paper.

1. Insertion bias. First, some servers in the distributed aggregation middleware may be under control of an attacker willing to influence the aggregation for some key $k$ managed by another server. In this case, the servers under the control of the attacker may generate fake insertions, e.g., to favor a given value over the others regardless of the inputs of the clients. As an example, in an aggregation system collecting feedback on website accesses and search queries [11], an aggregation server under the control of an attacker may wish to promote a given website regardless of the interest as perceived by users, making the recommendations obtained from user feedback useless.

2. Aggregation bias. Second, a server proceeding to the aggregation for a key $k$ may wish to return a counterfeit distribution that favors or disfavors one of the values in contradiction with the increment operations received. A typical example is for the aforementioned media popularity application, where the server responsible for the aggregation may be under the control of an institution with financial interests in one of the aggregated values.

Contributions. In this paper, we are interested in the detection of malevolent behaviors from servers participating to the aggregation layer. We propose and evaluate oracles to detect when one or several servers are attempting to bias the aggregation. These two oracles are lightweight and operate in a decentralized and autonomous manner. They are based on the nature of the operation performed, namely the aggregation of counters for a set of values, and consider the statistical deviation from an expected behavior.

The actual action(s) to be taken by the system once a server has been suspected are out of the scope of this paper, and will often be application-specific. The CADA oracles are intended to provide the input for an external mechanism such as a blacklist management service, a trust management layer, or the triggering of a more generic—but also considerably more costly—strict protocol auditing mechanism such as PeerReview [13].

Outline. The rest of this paper is organized as follows. We introduce the system model in Section II. Section III presents a high-level overview of the oracles. The auditing mechanisms are described in details in Sections IV and V for insertion and aggregation biasing, respectively. The oracles are evaluated in Section VI. Finally, we survey related work in Section VII and conclude in Section VIII.
Table I: Notations for the base aggregation system without auditing support.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>Number of servers participating to the aggregation service.</td>
</tr>
<tr>
<td>( EP_n ) ( 1 \leq n \leq N )</td>
<td>Entry Point is one of the two roles of the servers, when they are used by clients for routing their increment requests towards the corresponding aggregation point.</td>
</tr>
<tr>
<td>( AP_n ) ( (1 \leq n \leq N) ) ( AP(k) ) ( D(k) )</td>
<td>Aggregation Point is the other role of servers, when they process and aggregate insertions propagated from the EPs on behalf of the client, for a distribution they maintain. ( AP(k) ) denotes the ( AP ) in charge of the aggregation for the distribution ( D(k) ) associated with key ( k ).</td>
</tr>
<tr>
<td>( a_k^v \equiv a ) ( a_k^v.c \equiv a.c ) ( a_k^v.i \equiv a.i )</td>
<td>An accumulator is a component of a distribution ( D(k) ) for a value ( v ): ( D(k) = {a_k^v, \ldots, a_k^n, \ldots} ). When considering a single accumulator for some oracle, we simply denote it as ( a ) and omit indices. ( a.c ) (for counter) denotes the value of the accumulator, and ( a.i ) (insertions) denotes the number of insertions received for computing ( a.c ) (i.e., ( \frac{2i}{n} ) gives the average insertion value).</td>
</tr>
</tbody>
</table>

II. SYSTEM MODEL

In this section, we present the model considered for aggregation middleware and the hypothesis that will later support collaborative auditing. Throughout the description, we will refer to the notations summarized in Table I.

We consider an aggregation middleware infrastructure formed by \( N \) servers. In the context of CADA, there are two important assumptions that support the design of the bias detection oracles: the entry points uniform distribution hypothesis, and the authenticated servers hypothesis.

The Entry Point (EP) is the contact point for clients wishing to send or receive information to the system. Any client can contact any of the servers that will act as an EP for this transaction. This contact server is not necessarily the server performing the aggregation but is in charge of propagating the request in the system to the appropriate aggregation server, and contacting back the client.

The entry points uniform distribution hypothesis is the assumption of a random and uniform distribution of the number of requests that are received by each EP and thus propagated to the appropriate Aggregation Point (AP). This can be enforced by the use of one or several proxy servers or redirect servers (e.g., DNS servers), placed as a layer between the clients and CADA, that distribute the load evenly among all the EPs.

The entry points uniform distribution hypothesis is also enforced when using the SPADS layer to support anonymization, as it exploits multi-hop anonymizing routes following the principle of Chaum mixes and onion routing [7], [10]: a chain of \( c + 1 \) servers ensures that no server can obtain the clear text of a request from a client along with her identity, as long as no more than \( c \) servers on the path collude to break anonymity. In this context, the last server of the chain acts as the EP to the aggregation layer. This last element of the path is selected uniformly at random like the other elements from the full set of servers in the system, hence fulfilling the hypothesis. CADA and SPADS are independent and the presence of the latter is not required to support the operation of the former as long as the entry point uniform distribution hypothesis holds.

Furthermore, our approach could be extended to any kind of stochastic distribution of the number of requests per EP as long as the distribution can be given as an input to the oracles or gathered during runtime. These extensions are planned as future work as described in our conclusive remarks (Section VIII).

The basis of our second hypothesis for the design of CADA oracles, the authenticated servers hypothesis, relies on the existence of a trusted Authentication Authority (AA), where all servers register and authenticate, and on the fact that all servers know the public key of every other server. This is needed to support the signing of insertions by the EP (See Algorithm 1).

A server acting as an EP signs the requests received by clients according to Algorithm 1. The EP calculates a hash of the concatenation of the destination key (to certify the routing destination), the request itself (to support an integrity check), and a sequence number (to ensure that a message cannot be taken into account twice at the AP level). The hash is then signed with the EP private key in order to support authentication, and sent along with the request.

The aggregation layer uses an indexing mechanism such as a DHT [22], [23] or a single-hop routing layer [9]. The server responsible for aggregating the distribution \( D(k) \) associated to key \( k \), and denoted as \( AP(k) \), is located by this indexing mechanism. When the request arrives, the AP first checks the validity of its signature and the authentication of the sender EP, and then processes the request. A request can be a query for part or the entire distribution or an insertion. Insertions increase the total associated with a value \( v \) in the distribution and stored in an accumulator \( a_k^v \). The two components \( a_k^v.c \) and \( a_k^v.i \) contain the total of the counter and the number of insertions so far, respectively. The counter entries \( a_k^v.c \) of the accumulators for all values \( v \) forms the distribution \( D(k) \), while the contributions allow computing the average increment \( a_k^v.c/a_k^v.i \).

Algorithm 1: Signature for authentication and uniqueness at the EP level.

<table>
<thead>
<tr>
<th>Notations</th>
</tr>
</thead>
<tbody>
<tr>
<td>([m]_{(pub</td>
</tr>
<tr>
<td>( EP_{id} )</td>
</tr>
<tr>
<td>( EP_{pub} )</td>
</tr>
<tr>
<td>( m + n )</td>
</tr>
<tr>
<td>( s )</td>
</tr>
</tbody>
</table>

Sign \((k, v)\):

1. \( h \leftarrow SHA1(k + s + v) \)
2. \( p \leftarrow [h]_{EP_{pub}} \)
3. \( s \leftarrow s + 1 \)
4. return \((k, v, EP_{id}, s, p)\)
III. CADA: AN OVERVIEW

In CADA, we are interested in detecting the two forms of bias introduced by servers participating to the aggregation layer (see Section I). In this section, we give a high-level overview of the oracles that we use for detecting attempts by the servers to introduce bias.

Adversary model: First, servers may exceed their role as EP and influence distributions they are not in charge of, by forging and sending fake insertions to some APs, or modifying insertions received from the clients. A variant of this bias is when nodes acting as EP for clients selectively drop insertions. We treat these two variants of insertion biasing conjunctly. Second, servers may exceed their role as AP and bias one of the distribution they are in charge of, by returning fake counts for the requests they get from EPs on behalf of clients.

Detecting bias attempts: Our objective in CADA is to propose a set of oracles that trigger alarms, or suspicions, when detecting that a server is attempting to bias the distribution aggregation operation. This notion of suspicion is probabilistic, and is based on a statistical test over the observed behavior of the server(s) and their expected behavior. As such, it may yield false positives (unjustified suspicions) but a node that is effectively biasing will get a much larger number of suspicions than one that is not. Our oracles, being probabilistic, are associated to confidence levels that allow expressing a trade-off between sensitivity and number of false positives. The confidence level represents the degree of certainty that one has about a given assumption. For example, if a croupier takes an ace of spades out of a card deck (which has a probability of happening equal to 1/52th, or about 2%), we can say that he is cheating with confidence level of about 98%. That is because after doing a substantial number of tries, if the croupier is not cheating we should get such result only 2% of the time. Even high confidence levels like the one from the example can easily return a false positive after one try. This is why the output of CADA is not meant to be taken into account to directly ban or punish a server after one suspicion, but to serve as an input for trust or reputation systems that will accumulate suspicions to lower the reputation of a server. As a result, servers can sporadically cheat with a certain probability of not being detected, but a server that intends to cheat frequently, in order to alter the aggregation in a considerable way, will be detected by the accumulation of suspicions.

Insertion bias oracle: Our first oracle aims at detecting servers exceeding their roles as EP by sending illegitimate contributions that were not initially sent by users, or dropping or altering user contributions. This oracle is illustrated in Figure 4(a). It is run by each server as part of its AP role, in a periodic manner, or after a given number of contributions have been received since the last audit. This oracle monitors independently the counter for each value in the distribution. We denote the accumulator being audited as \( a \). Based on the entry points uniform distribution hypothesis, we can derive two important properties over the set of contributions received for an accumulator: with a large number of inputs to \( a \), the number of contributions received from each EP should be nearly equal, and the distribution of contributions to the total from each EP should also be similar, as the set of input received from each EP is simply a random uniform sample of all the client insertions. The authentication of the original EP is achieved thanks to the authenticated entry points hypothesis.

The oracle works by comparing the statistical properties of the set of insertions, and their number, received from

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a.I )</td>
<td>Vector of the number of insertions received from each of the EPs. For instance, ( a.I[n] ) denotes the number of insertions received from ( EP_i ); and ( \sum_{n=1}^{N} a.I[n] = a.i ).</td>
</tr>
<tr>
<td>( a.C )</td>
<td>Vector of the sum of the contributions to the accumulator counter received from each of the EPs: ( a.C[n] ) denotes the contribution to the total counter ( a.c ) received from ( EP_i ); ( \sum_{n=1}^{N} a.C[n] = a.c ).</td>
</tr>
<tr>
<td>( a.S )</td>
<td>Vector of the sum of the squares of each of the contributions to the accumulator counter received from each of the EPs. If ( a.inc_{1}^{n}, \ldots, a.inc_{m}^{n} ) are the individual contributions received for the accumulator ( a.c_{i}^{n} ) from ( EP_i ), then ( a.S[n] = \sum_{x=1}^{m} (a.inc_{x}^{n})^2 ).</td>
</tr>
</tbody>
</table>

Table II: Additions to the AP state for the insertion bias oracle.
each EP for an accumulator $a$. Note that we do not need to keep the full set of these insertions, but only aggregated values, of size $O(N)$. The additions to the state of the AP necessary to support the oracle are defined in Table II and illustrated in Figure 4(a). We replace the component $a.i$ (number of insertions) by a vector $a.I$ that keeps the number of insertions received from each EP independently. Similarly, the component $a.c$ (total of the accumulator) is replaced by $a.C$, a vector that distinguishes between each EP contribution to the counter $a.c$. Finally, to support the detection of statistically significant variations in the variance of elements sent by each EP, an additional vector $a.S$ collects the sum of the squares of all contributions received by each EP.

The insertion bias oracle operates in two steps. It first checks that the number of insertions for each EP follows a uniform multinomial distribution using a Pearson’s goodness-of-fit test. If that is not the case, it removes iteratively the most deviating EP from the test, and checks again the similarity with a multinomial distribution, until a tolerable level of confidence is reached. In the second step it performs again the Pearson test with the remaining nodes, but taking into account the contribution to the total for each EP. This step requires the use of $a.S$, as we detail in Section IV. All the EPs that are removed during the test are reported as suspicious. We describe formally in the next section how these elements are used for assessing statistically significant deviance, which results in a suspicion for the EP considered.

**Aggregation bias oracle:** Our second oracle aims at detecting servers that exceed their roles as AP, that is, that do not aggregate properly the insertions they receive but rather favor or penalize some accumulator $a$ over the others in the distribution $D(k)$. The principle of the aggregation bias oracle is illustrated in Figure 4(b). The extension required to the aggregation layer is given in Table III. The main idea of the aggregation bias oracle is to maintain $R$ shadow aggregations of the distribution. These shadow aggregations are maintained by $R$ different shadow AP’s. A system wide parameter, the shadowing ratio $\varphi$, determines the probability for an EP to send an increment request not only to $AP(k)$ but also to the $R$ shadow $AP'(k)$. The operation done by the shadow $AP'(k)$ is exactly the same as the normal $AP(k)$, except that they operate on a sample of all increments received. The identity of the shadow $AP'(k)$ is determined by hashing $k$ with multiple hash functions $h_1(k), h_2(k), \ldots, h_R(k)$ defined system-wide.

As we describe in Section V, the oracle seeks to compare the distribution of values on the main distribution $D(k)$ and the shadow distributions. Periodic auditing is carried out in two ways. First, whenever receiving a query request for a distribution $k$, the associated entry point $EP$ performs an audit of $AP(k)$ with probability $\alpha$. Second, $EP$ periodically audit random $AP$s. A suspicion is triggered if there is a significant enough probabilistic difference between the main and the shadow aggregations.

**IV. Auditing mechanisms for insertion bias**

The insertion bias oracle is called by a server as part of its AP role. The oracle checks all the accumulators $a.i^k$ for all the distributions that this node aggregates. Each accumulator is checked individually. We denote the accumulator under test as $a$ for simplicity. The oracle performs in two steps:

1) First, it detects if insertions were made by some EPs that are suspected of being insertion biasing attempts by forgery or dropping insertions. The AP uses an iterative algorithm that checks if the vector of the number of insertions $a.I$ represents a uniform multinomial distribution;

2) In a second step, the AP detects EPs that are suspected of performing insertion biasing by modifying the content of the insertions they relay. We use a similar algorithm that operates on the contributions $a.C$.

The Pearson’s goodness-of-fit test: Based on the entry points uniform distribution hypothesis, the vector $a.I = (a.I[1], \ldots, a.I[n], \ldots, a.I[N])$ should be a random vector following a multinomial distribution with uniform parameters $p_1 = \cdots = p_n = \cdots = p_N = 1/N$ in the regular case with no bias. The deviation from this regular case can be tested using a standard statistical goodness-of-fit test. We use the Pearson’s goodness-of-fit test. It requires to compute the following statistic $T(a.I)$:

$$T(a.I) = \sum_{n=1}^{N} \left( a.I[n] - \frac{\sum_{n=1}^{N} a.I[n]}{N} \right)^2 \frac{\sum_{n=1}^{N} a.I[n]}{N},$$

where

$$\mu(a.I) = \frac{\sum_{n=1}^{N} a.I[n]}{N},$$

is the mean of the components of the vector $a.I$, and

$$\sigma^2(a.I) = \frac{\sum_{n=1}^{N} \left( a.I[n] - \mu(a.I) \right)^2}{N},$$

is the variance of its components. The no bias hypothesis can be rejected with a level of confidence $c$ ($0 \leq c \leq 1$) defined as $c = cdff(\chi^2_{N-1})(T(a.I))$, where $cdff(\chi^2_{n})(x)$ is the value at $x$ of the cumulative distribution function of the chi-square probability distribution with $F$ degrees of freedom $\chi^2_{F}$ [3]. It results:

$$c = cdff(\chi^2_{N-1}) \left( N \cdot \frac{\sigma^2(a.I)}{\mu(a.I)} \right)$$

Table III: Additions to the AP state for the aggregation bias oracle.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D'_j(k)$ ($1 \leq j \leq R$)</td>
<td>Shadow aggregations of the distribution $D(k)$, based on a sample of $\varphi%$ of the insertions sent to $AP(k)$ and used for aggregation bias detection for key $k$. Each $D'_j(k)$ is maintained by a different shadow aggregation point server $AP'_j(k)$.</td>
</tr>
</tbody>
</table>
Algorithm 2: Oracle for insertion bias.

Notations
- $c$: confidence level for step 1
- $c_2$: confidence level for step 2
- $c_t$: confidence threshold
- $a_{left}$: aggregation for the set of remaining, not-yet-detected EPs
- $\delta_i[n] = (a_{left}[n] - µ(a_{left}, I))^2$: deviation of the number of insertions of EP$_n$ from the mean of $Icft_1$
- $\delta_c[n] = (a_C[n] - µ(a_C, C))^2$: deviation of the number of contributions of EP$_n$ from the mean of left.$C$

\text{detectInsertionBias}(k, v)
\begin{align*}
\text{left} & \leftarrow \{\text{EP}_1, \ldots, \text{EP}_t, \ldots, \text{EP}_N\} \\
\text{Suspicious} & \leftarrow 0 \\
& \text{// Step 1:} \\
& \text{while sizeof(left)} \geq 2 \text{ do} \\
& \quad c \leftarrow \text{performPearson}(a_{left}) \quad // \text{See Equation 1} \\
& \quad \text{if } c \geq c_t \text{ then} \\
& \quad \quad \text{detected} \leftarrow \{\text{EP}_i\} : \delta_i[n] \leftarrow \max(\delta_i) \quad // \text{picks the most deviating EP} \\
& \quad \quad \text{Suspicious} \leftarrow \text{Suspicious} \cup \{\text{detected}\} \\
& \quad \quad \text{left} \leftarrow \text{left} \setminus \{\text{detected}\} \\
& \quad \text{else} \\
& \quad \quad \text{L break} \\
& \text{// Step 2:} \\
& \text{while sizeof(left)} \geq 2 \text{ do} \\
& \quad c_2 \leftarrow \text{performPearson2}(a_{left}) \quad // \text{See Equation 2} \\
& \quad \text{if } c_2 \geq c_t \text{ then} \\
& \quad \quad \text{detected} \leftarrow \{\text{EP}_i\} : \delta_i[n] \leftarrow \max(\delta_i) \quad // \text{picks the most deviating EP} \\
& \quad \quad \text{Suspicious} \leftarrow \text{Suspicious} \cup \{\text{detected}\} \\
& \quad \quad \text{left} \leftarrow \text{left} \setminus \{\text{detected}\} \\
& \quad \text{else} \\
& \quad \quad \text{L break} \\
& \text{return Suspicious}
\end{align*}

where we denote $\rho = \frac{\sigma^2(a, I)}{\mu(a, I)}$. We observe (see Figure 5) that for $N$ sufficiently large, the level of confidence $c$ as a function of $\rho$, evolves according to a strongly S-shaped curve around the break-even value $\rho = 1$. This indicates that the bias detection will operate in an almost binary fashion, i.e., either $c \approx 0$ (when $\sigma^2(a, I) < \mu(a, I)$) or $c \approx 1$ (if $\sigma^2(a, I) > \mu(a, I)$).

Step 1—Bias by insertion forging or dropping: Both steps of the oracle are based on Algorithm 2. At the beginning of the first step, the server (as an AP) creates a full copy of the set of EPs, named left. The corresponding part of the aggregation $a$ that is related to all EPs in left is called $a_{left}$. It creates an empty set of suspicious EPs. Thereafter, the AP performs the Pearson’s test in an iterative manner: during each iteration, the AP removes the most deviating EP from left and places it into the set Suspicious. This process is repeated until the number of remaining EPs is less than 2, or the confidence level falls below a confidence threshold. We define the confidence threshold $c_t$ as the minimum confidence level that an iteration can yield in order to continue the auditing process. When the process yields less than the confidence threshold, it means that we are less sure than $c_t$ that the distribution is not uniform, and that there has been insertion bias, thus we stop.

Based on this detection process, we can calculate the probability for an EP that is not performing any insertion biasing, to be victim of a false positive as $P[e] = \frac{E(f)}{N}$, where $P[e]$ is the probability of an erroneous detection and $E(f)$ is the expected value of the number of false positives. The probability of having $r$ false positives is $(1 - c_t)^r c_t$ (the probability of making $r$ mistakes and correctly stopping at iteration $r + 1$). It follows:

\begin{align*}
P[e] & = \frac{1}{N} \left( (1 - c_t)c_t + 2(1 - c_t)^2c_t + \cdots + (N - 1)(1 - c_t)^{N-1}c_t + N(1 - c_t)^N \right) \\
& = \frac{1}{N} \left[ \sum_{n=1}^{N-1} (n(1 - c_t)^nc_t) + N(1 - c_t)^N \right] \\
& = \frac{c_t(1 - c_t)}{N} \sum_{n=1}^{N-1} (n(1 - c_t)^{n-1}) + (1 - c_t)^N
\end{align*}

And since we can apply:

\begin{align*}
\sum_{n=1}^{m} n.\delta^{n-1} = \frac{1 - \delta^{m+1}}{(1 - \delta)^2} - \frac{(m + 1)\delta^m}{1 - \delta}
\end{align*}

we obtain:

\begin{align*}
P[e] & = \frac{c_t(1 - c_t)}{N} \left[ 1 - (1 - c_t)^N -(N - 1)(1 - c_t)^{N-1} \right] + (1 - c_t)^N \\
& \approx \frac{c_t(1 - c_t)}{N} \cdot \frac{1 - c_t}{c_t N} = \frac{1 - c_t}{c_t N}
\end{align*}

We present the evolution of $P[e]$ in Figure 6, for varying confidence thresholds $c_t$ and several system size $N$. We can see in this figure that the values of $P[e]$ approach asymptotically to 0 as we increase $N$. For 50 and more nodes, we have less than 10% of probability of false positives when $c_t$ is more than 20%.

Step 2—Bias by insertion tampering: Our second step is based on a variant of the Pearson’s test. The AP performs this test iteratively until the confidence level is below the threshold $c_t$, as in step 1, and at each step it adds the most deviating EP to Suspicious and removes it from left. Here, the assumption on the distribution of the incoming contribution

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure5.png}
\caption{Confidence vs. variance/mean.}
\end{figure}
is not on a multinomial distribution anymore. Indeed, the $a.C$ vector is the result of the combination of the distribution $a.I$ (which is multinomial) and the probability distribution of the value of a contribution on each insertion. This latter distribution is not known \textit{a priori}. In our evaluation of the oracle, we used fixed, Gaussian and Zipf (power law where the $i$th most popular element out of 100 has a probability $P[i] \propto i^{-1}$) distributions. We present the results for the Zipf distribution as these constitute a worst-case scenario for the oracle, and is more representative of the target application context.

Let $a.C$ be a combination of the multinomial distribution $a.I$ with a given distribution $F(x)$. The resulting mean $\mu(a.C)$ and variance $\sigma^2(a.C)$ are the following:

$$\mu(a.C) = \mu(F) \cdot \mu(a.I),$$

$$\sigma^2(a.C) = \frac{a.i}{N} \left( \mu(F^2) + \frac{1}{N} (\mu(F))^2 \right)$$

For large $N$, $\sigma^2(a.I) \approx \frac{a.i}{N} \mu(F^2)$ and $\sigma^2(a.C) \approx \frac{a.i}{N} \mu(F^2)$,

where $\mu(F) = \frac{a.c}{a.i}$ is the mean of contributions per insertion and $\mu(F^2) = \frac{a.s}{a.i}$ is the mean of the squares of contributions per insertion.

As for the previous step, the ratio $\rho$ between variance and mean affects the confidence level. Let $\rho_2$ be the ratio between the variance and the mean, used in the calculation of $c$, the confidence factor, for this second step of the test. Noteworthy, $\rho_2$ can be computed from the $\rho$ used in the first step and observations on $F$. Since $\mu(a.C)$ and $\sigma^2(a.C)$ change by a factor of $\mu(F)$ and $\mu(F^2)$ respectively, we need to correct $T$ in order to position again the break-even point of $\rho$ at 1.

$$\rho_2 = \frac{\sigma^2(a.C)}{\mu(a.C)} = \frac{\mu(F^2) \cdot \frac{a.i}{N}}{\mu(F) \cdot \frac{a.i}{a.s}} = \mu(F^2) \cdot \rho$$

We thus need to use the following correction factor to compute the confidence level in the test of this second step [3]:

$$\frac{c}{\rho_2} = \frac{\mu(F)}{\mu(F^2)},$$

which yields the final confidence level $c_2$:

$$c_2 = \text{cdf}[\chi^2_{N-1}] \left( N \cdot \frac{\sigma^2(a.C)}{\mu(a.C)} \cdot \frac{\mu(F)}{\mu(F^2)} \right)$$

$$c_2 = \text{cdf}[\chi^2_{N-1}] \left( N \cdot \frac{\sigma^2(a.C)}{\mu(a.C)} \cdot \frac{a.c}{a.s} \right)$$

(2)

V. AUDITING MECHANISMS FOR AGGREGATION BIAS

The aggregation bias oracle also performs in two steps, and it is depicted in Algorithm 3. The first step checks the expected distribution with respect to the shadow distribution based on the insertions, while the second looks at the distribution of contributions.

First step: bias by insertion forging or dropping: We are interested in determining if the number of insertions reported by the main $AP$ and the shadow $AP'$ corresponds to a correct sampling. For any insertion received by the $AP$, there is a probability $\varphi$ that it was also received by $AP'$. The probability distribution of the value $a.i \cdot \varphi$ is thus a binomial, with its center at $a.i \cdot \varphi$. For $a.i$ sufficiently large, the binomial distribution $B(a.i, \varphi)$ can be approximated by a normal distribution with mean $a.i \cdot \varphi$ and variance $a.i \cdot \varphi \cdot (1 - \varphi)$. For $a.i > 5N$, the normal approximation is considered as valid if [3]:

$$\frac{\left| \sqrt{a.i} \cdot \varphi \right| - \sqrt{\frac{1 - \varphi}{a.i}}}{\sqrt{\varphi \cdot (1 - \varphi)}} \leq 0.3$$

and, if we normalize $a.i \cdot \varphi$ by subtracting the mean $a.i \cdot \varphi$ and dividing by the square root of the variance $\sqrt{a.i \cdot \varphi \cdot (1 - \varphi)}$, we obtain: $z = \frac{a.i \cdot \varphi}{\sqrt{a.i \cdot \varphi \cdot (1 - \varphi)}}$ that approximately follows a standard normal distribution $\mathcal{N}(0, 1)$. Thus, the value:

$$c = 2 \text{cdf}[\mathcal{N}_{0,1}] \left( \frac{\left| a.i \cdot \varphi \right| - 1}{\sqrt{a.i \cdot \varphi \cdot (1 - \varphi)}} \right)$$

represents the level of confidence that $a.i \cdot \varphi$ does not follow a normal distribution $\mathcal{N}(a.i \cdot \varphi, a.i \cdot \varphi \cdot (1 - \varphi))$ and thus $AP$ has not reported a trustworthy $a.i$ value.
Second step: bias by insertion tampering: For any population of values X with size \( a.i \), and any size \( a'.i \) random samples \( x \) drawn from \( X \) without replacement, we have:

\[
\sigma^2(\mu(x)) = \frac{\sigma^2(x)}{a'.i} \cdot \left( 1 - \frac{a'.i}{a.i} \right)
\]

where

\[
\sigma^2(x) = \left( \sum_{j=1}^{a'.i} \frac{x[j]^2}{a'.i} - \mu(x)^2 \right) \cdot \frac{a'.i}{a'.i - 1} \quad \text{and}
\]

\[
\mu(x) = \frac{\sum_{j=1}^{a'.i} x[j]}{a'.i}
\]

is an unbiased estimator of \( \sigma^2(\mu(x)) \). Therefore, for \( a'.i \) large enough, \( z = \frac{\mu(x) - \mu(X)}{\sqrt{\sigma^2(\mu(x))}} \) approximately follows a standard normal distribution \( N(0,1) \) [2]. With this additional result, a shadow \( AP' \) can compute the confidence level \( c \) that \( AP \) has not reported trustworthy \( a.i \) and \( a.c \) values:

\[
c_2 = 2 \text{cdf}[N_0,1] \left( \frac{| \mu(a'.C) - \mu(a.C) |}{\sqrt{\sigma^2(a'.C)} \cdot \left( 1 - \frac{a'.i}{a.i} \right)} \right) - 1 \quad (4)
\]

where \( \sigma^2(a'.C) = \left( \sigma^2(a'.C) \right) \cdot \frac{a'.i}{a'.i - 1} \).

Of course, if \( \sigma^2(x) \) is equal to 0, this estimator does not work. In this case, since the variance is 0, it means that all insertions of the sample contain an equal number of contributions. We can use the same model as in the first step, thus:

\[
c = 2 \text{cdf}[N_0,1] \left( \frac{| a'.c - a.c \cdot \varphi |}{\sqrt{a.s \cdot \varphi (1 - \varphi)}} \right) - 1 \quad (5)
\]

If several shadow \( AP' \)s are used, the suspicion will be based on a majority voting amongst their respective auditing results, with the leverage of a quorum-based technique. The use of several shadow \( AP' \)s avoids that a single \( AP' \) can trigger fake suspicions to incriminate an \( AP \).

VI. EVALUATION

We present in this section the evaluation of CADA oracles. These results were obtained by simulating 200 servers performing the role of \( EP \), one server \( AP(k) \) (henceforth called simply \( AP \)) responsible for the accumulator \( a_k \), and one shadow \( AP'(k) \) (henceforth called \( AP' \)) for the aggregation bias oracle. We use only one shadow \( AP' \) in our experiments to focus on the auditing process and leave the complexity of the agreement protocol to quorum-based techniques. For the experiments that evaluate the performance of the insertion bias oracle, a subset of 40 of the 200 nodes attempt to bias the aggregation, all in the same manner.

As the communication cost is not important in these experiments, processes were run in our local cluster. Yet, the implementation is full-featured and can readily be deployed in large networks. It was developed using the SPLAY [17] framework using a combination of C and Lua code.

Insertion bias oracle: In our first experiment, we observe how the confidence threshold affects the accuracy of the Pearson’s goodness-of-fit test. Figure 7 presents averaged results from 5,000 experiments with a network of 200 nodes. We consider a scenario where no server attempts to bias the insertions. Before doing the tests, we aggregate 5,000 contributions per \( EP \) (for a total of 1,000,000 contributions). One curve shows the result of performing only one iteration of the first step of the insertion bias oracle (i.e., one Pearson test), while the other curve shows the result of performing one Pearson test from the second step of the same oracle. Contributions on each of the insertions follow a 1-100 Zipf distribution (in the 1 to 100 range, and value 1 is 100 times more popular than 100). On the vertical axis we present the percentage of cases where the chi-square test yields a false positive. On the horizontal axis we measure the confidence threshold (on percentage levels) required to trigger a suspicion. As the chi-square test establishes, the expected behavior of both curves is to be linearly dependent on the confidence threshold: if the oracle requires 80% of confidence that the distribution is not multinomial and uniform, it is expected that the test will yield false positives 20% of the times in practice. As expected, the curves evolve in a linear way, starting from almost 100% at 0% confidence threshold and decreasing to 0% at 100% confidence threshold.

Figures 8(a) and 8(b) show the probability for a non-biasing \( EP \) to be victim of a false positive and the probability for a biasing \( EP \) to not be detected (false negative), depending on the confidence threshold. In our network of 200 nodes, 40 (20% of the population) introduce a bias in insertions while the remaining 160 (80%) have no bias. In order to test the first step of the oracle, the bias in Figure 8(a) is performed only through duplication of insertions. For test-

![Figure 7: Insertion bias oracle without EP introducing bias. The graph shows the evaluation of the two steps.](image-url)
from experiments where EP contributions for each insertion follow a 1-100 Zipf distribution. Through duplication of contributions on the insertions. Concluding the second step, the bias in Figure 8(b) is performed only of the bias on the aggregation. Percentages on the legend represent the global impact bias. Plots show false positive and negative rates vs. confidence threshold. AP presents the percentage of cases where the test yields a false positive as a function of the confidence threshold (in terms of percentage) required to trigger a suspicion. Since we can never tell with 100% of certainty that a server is biasing the aggregations (because of the probabilistic nature of the distributions), what we can provide is a confidence level: a level of certainty that such assumption is correct. Figures 7 and 9(a) confirm in fact that the theoretical confidence levels are correctly reflected in the number of false positives.

For instance, with 80% confidence that the totals in AP do not correspond to the sample in the shadow AP', the test yields false positives approximately 20% of the times in practice.

In Figure 9(b) we show the impact of the number of insertions aggregated before performing the tests, for the second step of the aggregation bias oracle. In this scenario the servers do not introduce any bias. We measure the probability of a false positive when increasing the number of insertions. Since both the expected value and the variance of the evaluated aggregation reach a stable value when the number of samples tends to infinity, we expect the results of the test to stabilize after a given number of aggregated insertions. The graph clearly shows that, after 10,000 inser-
tions, the probability reaches a lower bound whose value depends on the confidence threshold (see Figure 9(a)). The incidence of false positives in the cases where the confidence threshold is set to 80% and 95% respectively stabilizes at 20% and 5%. The graph also shows the impact of changing the shadowing ratio $\varphi$. As expected, the figure confirms that larger shadowing ratios produce more accurate tests. Indeed, the probability of false positives stabilizes faster for a shadowing ratio of 40% than for 20%, and convergence is even faster for 80%.

In Figures 10(a) and 10(b), we show the probability that the oracle detects a biasing $AP$. The probability of detecting the $AP$ is expected to grow and approach 100% as $AP$ introduces a higher bias. In these experiments, we use two models of bias: the $AP$ can either manipulate insertions or manipulate their contributions, which allows us to test the first and second steps, respectively. When the $AP$ drops an insertion or sets its contribution to 0, it produces a negative impact on the component $a.c$ (and also on $a.i$ in the case of dropping the insertion). This effect can be observed in the left side of the figures. When the $AP$ duplicates an insertion or doubles its contribution, it introduces a positive bias by increasing $a.c$. This appears on the right side. The graphs present results from experiments on 5,000 insertions aggregated on the $AP$. Each value is an average over 1,000 runs. A sample equivalent to 20% of the insertions is sent the shadow $AP'$, and the $AP$ introduces biases from -20% to 20%.

Both graphs show a higher likelihood for a server to be detected when increasing the bias level, on either the negative or the positive side. The curve is symmetric, which results from the detection method that effectively measures the absolute deviation from the expected amount of insertions. The graphs shows three curves for confidence thresholds of 95%, 80%, and 60%. One can observe that, in practice, there is a bias threshold from which a biasing $AP$ server will be detected with probability of almost 100%, which meets our objective. It is important to notice that the minimum values of the curves coincide with the detection probability when the $AP$ does not perform any bias, as shown in Figure 9.

VII. RELATED WORK

CADA provides a application-specific auditing framework that ensures a form of low-overhead fraud detection. The suspicions of misbehavior can be leveraged to ensure some confidence on the aggregated data. It operates by means of statistical tests between the expected behavior of servers and
the observed one. Other aggregation layers and protocols for large-scale systems include probabilistic techniques to deal with the loss of data elements [14], an orthogonal problem to the accountability of the aggregation servers, or top-k aggregation mechanisms [19].

CADA can be seen as a form of probabilistic failure detector [6] that is specialized for the aggregation operation. Byzantine failure detectors [15] in the context of agreement protocols also produce suspicions of detectable misbehavior but under a more general model of Byzantine behavior. Their generality does however come at a much higher price than the application-specific detection of CADA. In [1], the authors propose to detect Byzantine server failures in the context of a replicated database service. The system gathers statistics about the number of faulty servers and their impact on client requests, which can be seen as a common feature to CADA’s oracles mechanisms.

CADA, when coupled with trust or blacklist management, also constitutes an incentive mechanism for preventing servers from deviating from the protocol specification. Such incentive mechanisms are typically used to prevent free-riding [8], [20]. Similarly to CADA, they are application-specific and tailored to a particular kind of Byzantine behavior. Incentive mechanisms however typically only consider the lack of service from free-riders, whereas CADA also considers the case where nodes wish to serve compromised data.

CADA finally relates to accountability mechanisms for distributed systems. The PeerReview [13] accountability framework checks the conformance of a distributed protocol to its specification as a deterministic state machine, by logging the message sent and received by each node and having them replayed by an external (witness) node against the state machine. PeerReview targets general accountability and, as opposed to CADA, does not take into account the specificities of the operation implemented by the protocol. This leads to a high overhead both in terms of memory and computation, and puts strong requirements on the protocol. The focus of CADA is different in that it tolerates some limited deviant behavior from a server in the aggregation layer before issuing a suspicion. Note also that, unlike PeerReview, CADA does not log the network activity for each node and as such cannot support verifiable evidence of misbehavior.

CATS [24] is another accountability mechanism that, similarly to CADA is application specific (it targets a storage service). Similarly to the AP auditing mechanism of CADA, CATS checks the legitimacy of insertions and modifications to the stored elements, but does so in a comprehensive manner, checking all incoming messages as in PeerReview. This results in a larger overhead than with the node-local operations performed by CADA oracles.

Since CADA is a loose auditing system (it does not check every message sent or received like PeerReview or CATS), it is lightweight and highly scalable. In fact, increasing the rate of insertions or contributions, improves the accuracy of the system, without any compromise on computing load, because the statistics that the system use are performed on the aggregation, not on individual insertions.

VIII. Conclusion

The aggregation of distributions is a fundamental component of many large-scale distributed applications. However, in order to enforce the liability of a service based on aggregation, such as a monitoring infrastructure, a feedback aggregation and recommendation layer, or a trust management system, some guarantees on the good behavior of the servers collaboratively providing the service must be enforced. In particular, it is necessary to discourage biasing behaviors where a node in the aggregation layer may want to compromise either the distributions it maintains or the distribution maintained by others. In this paper, we presented the CADA oracles, which consider two misbehaviors that servers may implement: insertion and aggregation biasing. The oracles assess the probability of correct behavior of the servers according to statistical tests over the distributions of information at aggregation peers and as received from clients. Based on these tests, CADA issues suspicions for servers that deviate from the expected behavior observed in the rest of the system.

Our work on CADA oracles opens many interesting perspectives. First, we are considering the use of the oracles to automatically purge from aggregated distributions the information originating from misbehaving servers (as identified by their number of suspicions). Since the distribution of elements coming from servers that are deemed safe follows the same trend, the distribution with no bias can be built by interpolating the distributions from safe servers (or from shadow servers in the case of aggregation biasing). This results in an additional disincentive for servers to bias the aggregation: their contributions may simply be ignored silently.

Second, we plan on relaxing the entry points uniform distribution hypothesis, by supporting any distribution of EPs origin in the insertion bias oracle. This approach is feasible only if the distribution of EPs contributions is known in advance: we plan to collectively build this distribution based on observations made by other servers in the aggregation layer, and provide it as an input to the insertion bias oracle.

Finally, we will consider the pairing of the oracles with a reputation management mechanism, in which the signed suspicions reports by the servers will be used to build the reputation of servers and enforce trust metrics.

Acknowledgment

This work was supported in part by the Swiss National Foundation under agreement number 200021-127271/1. We would like to thank the anonymous reviewers for their
constructive comments, as well as Bastien Marmet for the valuable discussions during the writing of this paper.

REFERENCES


