Sentinels and Twins: Effective Integrity Assessment for Distributed Computation

Sabrina De Capitani di Vimercati, *Senior Member, IEEE,* Sara Foresti, *Senior Member, IEEE,*
Sushil Jajodia, *Fellow, IEEE,* Stefano Paraboschi, *Member, IEEE,*
Pierangela Samarati, *Fellow, IEEE,* Roberto Sassi, *Senior Member, IEEE*

Abstract—Distributed computing supports large scale and data-intensive computations with the cooperation of a multitude of parties, each responsible for a portion of the workload. Such parties are often not fully reliable and may return incorrect results. In this paper, we address the problem of assessing the integrity of the computation results. We provide a comprehensive characterization of two techniques, *sentinels* and *twins*, evaluating their effectiveness and synergy. Sentinels are pre-computed tasks whose result is known apriori, and enable checking returned results against a ground truth. Twins are replicated tasks assigned to different workers, and enable cross-checking returned results for a same task.
The analysis considers many questions that arise in the design of a concrete integrity assessment strategy and identifies the parameters that have a critical impact on the overall protection. Our model enables to tune the integrity controls so to achieve best effectiveness. The model can be applied to a variety of scenarios and offers guidelines that can find extensive application.

Index Terms—Distributed data computation, probabilistic integrity guarantees, sentinels, twins

1 Introduction

Distributed computing has become the norm for the management of large computational problems, which can be decomposed in multiple sub-problems, each assigned to a different device. A demonstration of the importance of representing large-scale computations as a large number of independent tasks is the success of modern MapReduce architectures, like Apache Spark. A particularly important application of this paradigm occurs when machine learning is integrated with big data, which is arguably the topic in the IT domain that is currently receiving the greatest attention. A common feature of this integration is the need to extract knowledge from large collections of data, using an approach where initially a model is built in a training phase and then it is applied over an extremely large number of instances in the prediction phase, where each instance is classified. The computational requirements of the prediction phase can be extreme. For instance, in environmental monitoring it is today possible to use a large number of sensors (e.g., microphones and cameras) collecting large volumes of data, which are analyzed to identify specific subjects (e.g., elephants in the African jungle [1] or snow leopards [2]). Hospitals want to apply image analysis techniques to large collections of medical images. In all these cases, there is the need to process millions or even billions of jobs, requiring the use of large infrastructures. The motivation for outsourcing computations involving external parties (workers) can be both the need of high-performance computational capabilities and economic convenience.

A clear concern in such distributed outsourced scenarios is the lack of control over the jobs' computation and hence the uncertainty about the correctness of results returned by the different workers in the system. While one may assume an overall proper behavior, the open nature of the system is clearly vulnerable to possible misbehavior by workers, which can be either sloppy in their operation, or - even worse - intentionally misbehave (to get rewards without employing needed resources), and therefore opportunistic in their responses.

The problem is well known and recognized by the research and industrial communities, which have devoted attention to the development of techniques to assess integrity of the results of computations outsourced to external parties. A promising approach to assess integrity in contexts where computations are not fixed and predefined (and therefore authenticated data structures providing deterministic integrity guarantees are difficult to use) relies on probabilistic techniques. These can always be applied when each portion of the problem assigned to a worker can be structured as a collection of jobs, each producing a result. It is then possible to inject jobs against which the behavior of workers is controlled. Most common probabilistic techniques either (*a*) insert jobs whose result is known a-priori, alerting for violations whenever results are different from the known one, or (*b*) replicate jobs to multiple workers, alerting for violations whenever results from different workers in response to the same replicated job differ. While the two techniques are known and well recognized, the problem of their targeted generation and combination, so to provide best effectiveness for integrity guarantees, is still an open issue.

We address this problem and propose a model to reason on the combined use of pre-computed and replicated jobs,

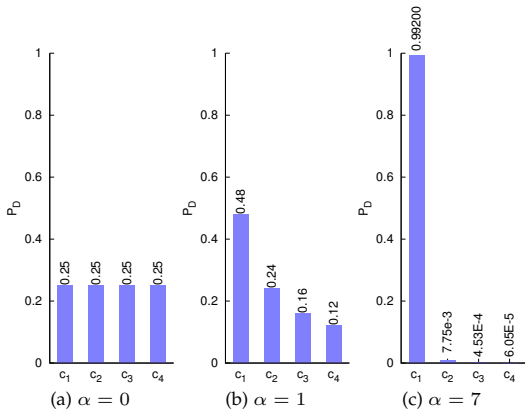
- *Sabrina De Capitani di Vimercati, Sara Foresti, Pierangela Samarati, and Roberto Sassi are with the Università degli Studi di Milano, Italy. E-mail:* firstname.lastname@*unimi.it*
- *Sushil Jajodia is with George Mason University, USA. E-mail: jajodia@gmu.edu*
- *Stefano Paraboschi is with the Università degli Studi di Bergamo, Italy. E-mail: parabosc@unibg.it*

which we call *sentinels* and *twins*, respectively, so to provide best effectiveness. We frame our work in the context of a data classification problem, which allows us to capture a variety of scenarios, and investigate different issues that naturally arise in the application of such controls. Our investigation produces an improved characterization of each technique and provides a response to many questions: What are the aspects that have an impact on the effectiveness of sentinels and twins? How should sentinels be generated and twins be distributed among workers so to provide best integrity guarantees? How many replicas should be used to get the best effectiveness? When are twins more effective than sentinels (or vice versa)? Given an application domain, what is the combination of sentinels and twins able to provide the best protection? The results of the investigation show the effectiveness of the two techniques when carefully combined based on our findings. Our work represents then a reference for the application of probabilistic techniques, supporting the realization of efficient integrity assessment solutions for many domains.

2 Scenario and Basic Concepts

Our problem is to enable a *client* to outsource data processing to possibly untrusted *workers* in a distributed system, while enjoying integrity guarantees on the returned results. There can be multiple reasons for a computation result to be incorrect: a defect, a temporary misconfiguration, or a malicious action by the worker (which may want to either sabotage the computation or get the reward for computing jobs while omitting to do so). From an integrity-assessment point of view, there is an integrity issue regardless of whether the incorrect result has been caused by failure, malfunctioning, sloppiness, or intentional opportunistic behavior since they all have the effect of the worker not correctly computing the jobs assigned to it. In our analysis, we consider the problem of detecting misbehavior of intelligent workers, which intentionally omit computation and behave opportunistically in their responses to avoid being detected in their omissions. The reason for considering intentional misbehavior is to set our work in the worst (more difficult to discover) scenario. Indeed, techniques that are able to withstand the action of an intelligent worker would also offer integrity guarantees when the violation is produced by accidental anomalies. In this way, we also cover the case of anomalies that exhibit a behavior that may otherwise be hard to detect (e.g., when a defect produces as a result the most common answer expected from the computation).

In our scenario, we assume workers to be computers executing a deterministic program. For concreteness, we consider a classification problem computing, for each data item in a collection, the class associated with it. The consideration of a generic classification problem allows us to capture different application scenarios characterized by processing tasks producing results in a finite domain of values. Common computational tasks can be considered as classification jobs with an extremely large number of classes. For instance, the prediction phase in machine learning can be seen as a classification problem, whose goal is to classify each instance according to the model defined in the training phase. Formally, we define a classification problem as follows.



(a) $\alpha = 0$ (b) $\alpha = 1$ (c) $\alpha = 7$

Fig. 1. An example of probability mass function P_D: Zipf's law with $\alpha = 0$ (a), $\alpha = 1$ (b), $\alpha = 7$ (c)

Definition 2.1 (Classification). Given a set $D = \{\mathtt{d}_1, \ldots, \mathtt{d}_n\}$ of data items and a set $C = \{\mathtt{c}_1, \ldots, \mathtt{c}_c\}$ of classes, a *classification* is a function $\gamma : D \to C$ that assigns each data item $\mathtt{d}_i \in D$ to a class $\mathtt{c}_k \in C$.

A classification function γ can be characterized by a probability mass function P_D that describes the probability of a data item to belong to each class in C. For concreteness, in the paper, we refer our examples to a classification over a set C=$\{\mathtt{c}_1,\mathtt{c}_2,\mathtt{c}_3,\mathtt{c}_4\}$ of four classes and three representative instances of probability mass function (Figure 1). The considered distributions follow a Zipf's law with: $\alpha = 0$, modeling uniform distribution; $\alpha = 1$, representing the distribution of classes known to be common in many domains; and $\alpha = 7$, representative of a very skewed distribution. We focus the initial analysis on classification jobs with a limited output domain, because in these cases it is easier to characterize the distribution of output values. This is not a limitation since, as our analysis will show, the number of classes has negligible impact on the effectiveness of the techniques, which depends instead on the probability of the most frequent class (regardless of the number of classes). The results of our analysis apply then to generic classification problems and data distributions, including those with only two or an extremely large number of classes (with this latter capturing generic computations with a wide variety of possible results).

We assume the classification problem to be *deterministic* and *complete*. Deterministic means that we assume a diagonal confusion matrix and an accuracy of 100% in the classification process (i.e., we assume workers to be machines running the same algorithm, in contrast to human beings performing a task). Then, an incorrect result is due to an incorrect (defective, sloppy, or malicious) computation. For instance, with reference to machine learning, the execution of a prediction job deterministically returns a class, as it returns the deterministic result produced by the application of the model obtained in the training phase. This holds also when the returned result may not enjoy perfect quality. As an example, for the problem of identification of snow leopards in camera images, the model may sometime miss the presence of the leopard, but this does not affect the

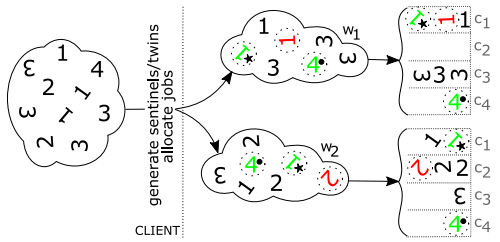


Fig. 2. Inclusion of sentinels and twins in the workers' workload. Circled items are sentinels and twins, marks identify twin sets

determinism of the model, and hence of the result of the job execution. Completeness implies that every data item is associated with a class in C. Note that a partial classification problem can be made complete by simply introducing a 'dummy' class to which data items not of interest for the classification will be mapped.

In the following, we refer to the computation of the classification of an individual data item as a *job* (short for *classification job*). Outsourcing of jobs to external, possibly untrusted, workers is then formalized as an allocation function defined as follows.

Definition 2.2 (Job allocation). Given a set $D = \{\mathtt{d}_1, \ldots, \mathtt{d}_n\}$ of data items on which a classification is to be computed, and a set $W = \{\mathtt{w}_1, \ldots, \mathtt{w}_w\}$ of workers, an *allocation* is a function $\omega : D \rightarrow W$ that assigns each data item $\mathtt{d}_i \in D$ to a worker $\mathtt{w}_k \in W$.

We use $\delta(\mathtt{w})$ to denote the set of data items assigned to worker $\mathtt{w}$. For simplicity, we assume an even distribution of jobs across workers, that is, the number of jobs assigned to any two workers differs of at most one. Formally, $\forall \mathtt{w} \in W : \lfloor g \rfloor \leq |\delta(\mathtt{w})| \leq \lceil g \rceil$, with $g = \frac{|D|}{|W|}$. Also, for simplicity in the formulation, we assume the number of data items in D to be a multiple of the number of workers, and therefore that all workers have exactly the same workload. This is not a limitation: a model with a heterogeneous assignment of jobs would make the analysis more complex, but it would produce identical results with respect to the efficiency and effectiveness of the integrity assessment techniques.

We use γ^i to denote the classification performed by a worker $\mathtt{w}_i$, and γ^W to denote the classification performed by all workers (i.e., the union of the classifications computed by all the considered workers).

Our problem is then allowing the client to assess integrity of classification γ^W. We distinguish between *honest* and *lazy* workers. An honest worker correctly performs jobs assigned to it. A lazy worker omits some of the jobs assigned to it, returning for them a result that is freely chosen by the worker. Lazy workers can however possibly behave *opportunistically* on the omitted jobs to the aim of not being discovered (e.g., exploiting possible knowledge on the classification job). For simplicity, we assume lazy workers to be independent entities and to not communicate with each other. The impact of collusion would only be a reduction in the effectiveness of twins (see Section 7).

3 Integrity Checks

Our work is based on the inclusion, in the jobs to be outsourced, of additional jobs that will serve as checks for integrity assessment (Figure 2). For such additional jobs, we consider two complementary approaches: *i)* insertion of jobs whose result is known a priori; *ii)* replication of the same job to different workers. These two approaches well reflect state of the art techniques typically considered for integrity control. Controls of the first type correspond, for example, to sentinels and markers (e.g., [3], [4], [5], [6]), and watermarks (e.g., [7]). Controls of the second type correspond to classical replication (e.g., [8], [9]). In our work, we refer to jobs of the first kind as *sentinels* and to jobs of the second kind as *twins*, characterized as follows.

Sentinels. A sentinel is a data item generated ad-hoc by the client and whose classification (i.e., job's result) is known a priori. In the following, when clear from the context, we will use the term sentinel to refer interchangeably to the sentinel data item or its classification job. Receiving for a sentinel a result different from the known classification signals an integrity violation. Formally, a sentinel $\mathtt{d}$ signals an integrity violation whenever $\gamma^i(\mathtt{d}) \neq \gamma(\mathtt{d})$, with $\omega(\mathtt{d})=\mathtt{w}_i$.

Twins. A twin is a replica of a data item whose classification job is submitted to more than one worker. Like for sentinels, in the following, when clear from the context, we will use the term twin to refer interchangeably to a replicated data item or the corresponding classification job. We will use the term *twin set* to refer to a data item and its replicas. We also note that, while for simplicity we refer to a replicated data item, in practice there is no need to actually create a replica of the data item, but it is sufficient to allocate the corresponding job to multiple workers. Formally, allocation function ω is extended to possibly assign each data item to a set of workers (in contrast to a single one), that is, $\omega : D \rightarrow \mathcal{P}(W)$. A classification job can be replicated as many times as wished, and the identification of the optimal number of replicas to be used is one of the contributions of this paper. The name twin reveals the fact that, according to our analysis (Section 5), best effectiveness is achieved when a twin set has cardinality 2 (i.e., original data item plus one copy). Receiving for twin jobs inconsistent results signals an integrity violation. Formally, a twin set signals an integrity violation whenever $\exists \mathtt{w}_i, \mathtt{w}_k \in \omega(\mathtt{d})$ such that $\gamma^i(\mathtt{d}) \neq \gamma^k(\mathtt{d})$.

Clearly, to be effective, integrity checks should not be recognizable as such by the workers, which could otherwise go undetected in their possible misbehavior by simply performing well on the jobs on which they know to be controlled. For twins, allocating twin jobs to different workers, which we consider as a good design principle for such kind of control (as distribution of replicas to different workers provides a natural cross-check), already guarantees such a property. For sentinels, such a property has to be taken into account in the generation of sentinel jobs.

The contribution of this paper is to evaluate how sentinels and twins should be produced to be best effective and how integrity controls should be distributed among sentinels and twins so to be effective and provide the greatest integrity guarantees. We first analyze sentinels and twins independently, and then investigate their combination. Table 1 summarizes the notation used in this paper.

TABLE 1
Notation

c	number of classes in C
w	number of workers in W
g	number of genuine jobs assigned to a worker
j	number of genuine and control jobs assigned to a worker
s	number of sentinels assigned to a worker
s_{tot}	overall number of sentinels
t	number of twins assigned to a worker
T	overall number of twin sets
r	number of replicas in a twin set
l_T	number of lazy workers in a twin set
l	overall number of lazy workers
o	number of jobs omitted by a lazy worker
o_s	number of sentinels omitted by a lazy worker
P_D	probability mass function of genuine data in classes
P_S	probability mass function of sentinels in classes
P_O	probability mass function adopted by a worker for omitted jobs

TABLE 2
Probability p_guess_sent of correctly guessing the classification of a sentinel

	P_O		
P_S	**[1,0,0,0]**	**[0.25,0.25,0.25,0.25]**	**same as P_S**
$\alpha = 0$ **[0.25,0.25,0.25,0.25]**	0.2500	0.2500	0.2500
$\alpha = 1$ **[0.48,0.24,0.16,0.12]**	0.4800	0.2500	0.3280
$\alpha = 7$ **[0.992,7.75e-3,4.53e-4,6.053e-5]**	0.9920	0.2500	0.9841

4 SENTINEL ANALYSIS

We start by analyzing the effectiveness of integrity assessment through sentinels. We first focus on a single worker, which we assume to be lazy, because each sentinel controls the behavior of one worker (the one in charge of its evaluation) and its effectiveness is not influenced by the behavior of other workers. The result of the analysis applies in general to any of the workers. We then generalize the results to assess the effectiveness of sentinels in the system.

The goal of our analysis on sentinels is to determine how the number of sentinels (or, more precisely, their percentage with respect to the overall number of jobs assigned to the worker) and their distribution in the different classes affect integrity guarantees. Clearly, the amount of sentinels affects integrity guarantees: the higher the percentage of jobs on which a worker is checked, the lower the probability of the worker to go undetected when omitting jobs. As we will show in this section, also the distribution of sentinels in classes plays a role on the effectiveness of the control. In the following, we first evaluate the probability of a lazy worker to go undetected by sentinels control when omitting some of its jobs (Section 4.1). With such probability turning to be also dependent on the distribution of sentinels in classes, we then analyze possible sentinel distribution strategies and their effectiveness (Section 4.2). Finally, we extend this result to the whole collection of workers (Section 4.3).

4.1 Probability analysis

A lazy worker passes sentinels control if it performs correctly on all the sentinels, that is, if the classification returned for all sentinels coincides with the classification known to the client. This happens when the worker actually performs the job or when the worker does not perform the job but it returns a correct result for it. The probability of the worker to go undetected (i.e., to return a correct result for all sentinels) when omitting some of its jobs, depends then on: *1)* the probability of sentinels to fall in the omitted jobs (since only sentinels are controlled, omission of genuine jobs goes undetected) and *2)* the probability of the result returned for omitted sentinels to be correct. Let us then analyze each of these probabilities.

Probability of omitting sentinels. Consider a set of j jobs comprising s sentinels and assume the worker omits o of the jobs. The probability that o_s of the o omitted jobs are sentinels (i.e., the probability of the worker to omit o_s of the s sentinels) follows a hypergeometric distribution and is as follows:

$$p\_omit\_sent(o_s) = \frac{\binom{j-s}{o-o_s} \cdot \binom{s}{o_s}}{\binom{j}{o}}$$

The higher the number s of sentinels, the higher the probability of omitting o_s of them. For instance, the probability that the worker omits 5 sentinels is: 2.72% using 20 sentinels, 18.27% using 40 sentinels, and 18.51% using 60 sentinels.

Probability of correctly guessing a job. Even when omitting some sentinels, the worker could go undetected if the result it returns for them is correct. The probability that the worker correctly guesses the class of a sentinel when omitting its computation depends on the strategy adopted by the worker when selecting the classes to be returned for omitted jobs and on the distribution of sentinels into classes. The assignment, by the worker, of omitted jobs to classes can be modeled as a probability mass function P_O. Hence, $P_O(\mathtt{c}_i)$ is the probability that the worker assigns class $\mathtt{c}_i$ to an omitted job. If the worker randomly extracts the class of the omitted job from P_O, the probability that it guesses the correct class of an omitted sentinel is:

$$p\_guess\_sent = \sum_{i=1}^{c} (P_O(\mathtt{c}_i) \cdot P_S(\mathtt{c}_i))$$

where P_S describes the probability mass function of sentinels (i.e., their distribution) in classes. Indeed, the probability of correctly guessing class $\mathtt{c}_i$ to which a sentinel belongs is the product of the probability that the worker chooses $\mathtt{c}_i$ for the omitted sentinel job (i.e., $P_O(\mathtt{c}_i)$), multiplied by the probability that $\mathtt{c}_i$ is the correct class for it (i.e., $P_S(\mathtt{c}_i)$).

Table 2 illustrates the probability for a lazy worker to correctly guess the class of an omitted sentinel (p_guess_sent), considering different distributions for P_S (following the Zipf's law with α=0, α=1, and α=7) and different strategies P_O for the worker to classify omitted jobs. In particular, we consider three possible strategies that the worker can choose to classify omitted jobs: always in the most frequent

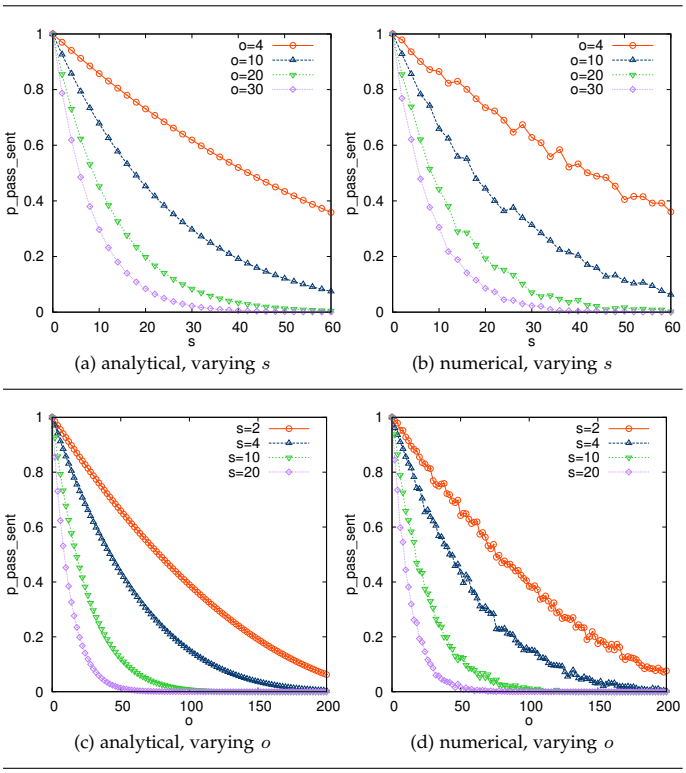


Fig. 3. Probability p_pass_sent varying the number s of sentinels (a,b) and the number o of omitted jobs (c,d) obtained analytically (a,c) and through Monte Carlo simulations (b,d)

class in P_S (i.e., P_O=[1,0,0,0]), according to a uniform distribution (i.e., P_O=[0.25,0.25,0.25,0.25]), according to P_S (i.e., P_O=P_S). For instance, when P_S is a Zipf's with $\alpha = 1$ and the worker classifies all the omitted jobs in the most frequent class, p_guess_sent is 48%. The table shows that, independently from P_S, the most convenient strategy for the worker to maximize p_guess_sent consists in classifying all the omitted jobs in one class (ideally, the most frequent in P_S, which the worker however does not know). In fact, distributing its guesses among different classes, the worker has greater probability of assigning a wrong class to an omitted sentinel. This is visible in Table 2, where the first column has always values greater than, or equal to, the ones in the other columns.

Probability of passing sentinels control. Since the classifications of different jobs are independent events, the probability that a worker passes integrity control when omitting o_s sentinels is the product of the probability of the worker to omit o_s sentinels multiplied by the probability of correctly guessing their classification. Formally, such probability is $p\_omit\_sent(o_s) \cdot (p\_guess\_sent)^{o_s}$, with the second term reducing to 1 when o_s=0 (i.e., no sentinel is omitted). Therefore, the probability of the worker to pass sentinels control (i.e., the probability of a worker omitting o_s jobs to go undetected in its omissions), is the sum, over all possible values of o_s, of the probabilities of the worker to pass sentinels control when omitting o_s sentinels. Formally:

$$p\_pass\_sent = \sum_{o_s=0}^{\min(s,o)} (p\_omit\_sent(o_s) \cdot (p\_guess\_sent)^{o_s})$$

Note that the sum terminates at $min(s,o)$ since the worker cannot omit more sentinels than either the number of sentinels it has received or the number of jobs it omits.

Figure 3 compares the values of p_pass_sent, varying the number s of sentinels and the number o of omitted jobs, obtained analytically and through 1000 Monte Carlo simulations. In the simulations, we considered a worker in charge of the classification of $j = 200$ jobs that can be classified into 4 classes $C = \{c_1, c_2, c_3, c_4\}$. The probability mass function P_D is a Zipf's law with $\alpha = 1$, sentinels are distributed according to a uniform distribution (i.e., P_S is uniform), and the worker classifies all the omitted jobs in the most frequent class in P_D (i.e., P_O is [1,0,0,0]). As visible from the figure, the analytical and numerical values nicely match.

4.2 Sentinels distribution

The client can operate on two factors in injecting sentinels: the number s of sentinels, and their distribution P_S into classes. Indeed, the other parameters in the formula of p_pass_sent are not under the control of the client. Clearly, the higher the number of sentinels, the higher the probability of the worker to hit them (i.e., the higher o_s) when omitting jobs. The average value of o_s is $\frac{o \cdot s}{j}$, which is the average of the hypergeometric distribution regulating p_omit_sent. However, an omitted sentinel can be detected only if the response returned for it is different from the correct one, and here is where the distribution of sentinels into classes comes into play.

We identify three possible strategies that the client can use for distributing sentinels into classes.

- *Genuine data distribution* (P_S=P_D): sentinels are distributed into classes following the same distribution as the data. The rationale behind such a strategy is to follow cardinality of the genuine jobs in assigning sentinels, so that each class receives a number of sentinels according to its expected cardinality.
- *Normalized inverse distribution of genuine data* (P_S=${P_D}^{-1}$): sentinels are distributed into classes following the normalized inverse of P_D. Therefore, the most frequent class in P_D is the least frequent in P_S (and vice versa). The rationale behind such a strategy, working opposite to the one above, is to inject more control with a result that is less expected.
- *Uniform distribution* (P_S uniform): sentinels are distributed equally among classes, regardless of data distribution. The rationale behind such a strategy is to make any guess equally likely for correctness.

Note that a limited number of sentinels is sufficient to provide high integrity guarantees (see Section 6.3). Hence, the addition of sentinels is not expected to considerably modify the data distribution of genuine values.

The effectiveness of sentinels' control depends also on the strategy that the lazy worker can adopt at its side. As noted in the previous subsection, the lazy worker does not have any convenience in distributing omitted jobs in different classes (as its probability of correctly guessing the class would naturally decrease) and its best bet is instead to consistently assign omitted jobs to a single class. In fact,

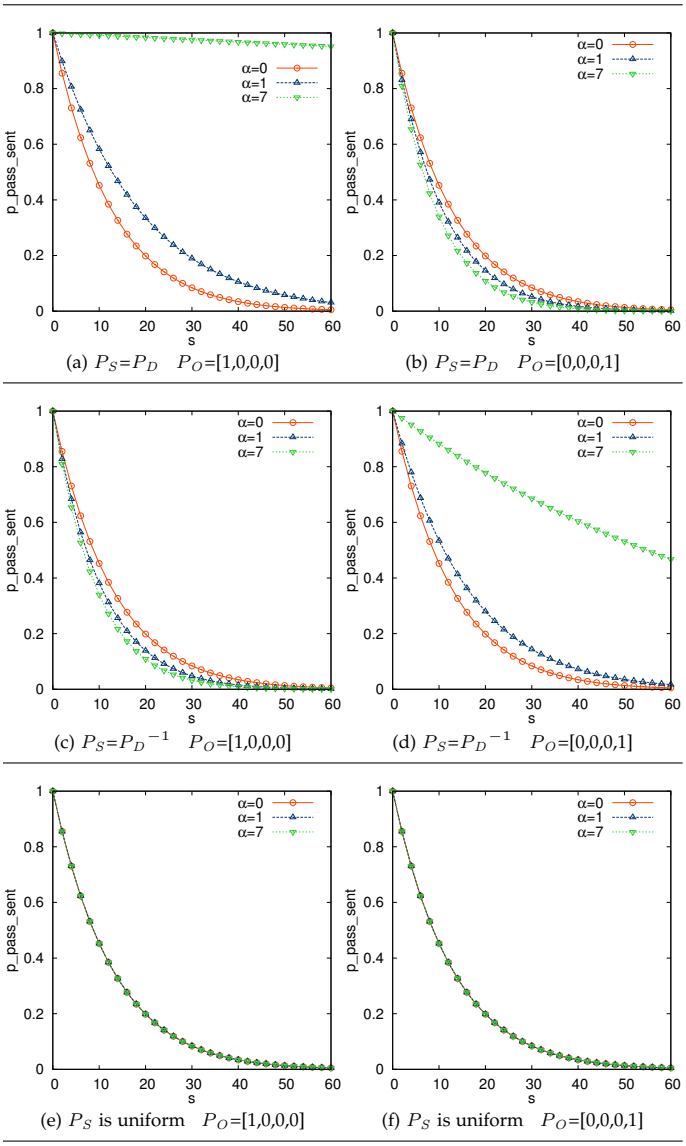


Fig. 4. Probability p_pass_sent varying the number s of sentinels and the distribution P_D of genuine data, considering different distribution strategies for the client and guess strategies for the lazy worker

such a class would be the one most probable according to data distribution but, since only sentinel jobs are actually checked, the worker should reason on what would be most probable as a correct guess for sentinels themselves (i.e., P_S in contrast to P_D). This observation implies two possible choices for the default class that a lazy worker could assign to omitted jobs:

- *the one most frequent in the data distribution*, assuming sentinels follow the same distribution as data;
- *the one least frequent in the data distribution*, assuming the hypothesis above to be too obvious and therefore cleverness at the client side in injecting control with results that are less expected.

Figure 4 illustrates the probability of the worker to pass integrity controls (i.e., value of p_pass_sent) in the different possible combinations of the client (P_S) and worker (P_O) strategies. It is immediate to see that a possible strategy for the client to follow a non-uniform distribution for sentinels, opting for either the same distribution as the data or the normalized inverse one, is exposed to the risk of the lazy worker to actually succeed in its opportunistic assignment of omitted jobs to the most frequent, or least frequent, class of data distribution. This is well visible in Figure 4, that shows the high probability of the worker to go undetected in omitting jobs when choosing the most frequent class (when the client distributed sentinels with same distribution as the data, Figure 4(a)) or the least frequent class (when the client distributed sentinels with a distribution inverse to the one of the data, Figure 4(d)). Uniform distribution for sentinels instead confirms to be resilient to possible opportunistic guesses (Figures 4(e) and 4(f)). Clearly, the weakness of sentinels' distribution different from the uniform one in Figure 4 is better visible when the data distribution (and therefore the sentinels' distribution) is skewed (i.e., $\alpha = 7$). When the data distribution is not skewed (i.e., $\alpha = 0$ and $\alpha = 1$), data will tend to be more uniformly distributed and therefore strategies P_S=P_D (Figures 4(a) and 4(b)) and P_S=$P_D{}^{-1}$ (Figures 4(c) and 4(a)) for sentinels distribution will resemble the uniform one. Intuitively, the strength of uniform distribution for sentinels is to not make any class (and hence correct guess for omitted jobs) more likely to be correct than another, and therefore dismount possible opportunistic behavior: the worker would not know on which class it is best to bet for its omitted jobs, and any possible strategy it could adopt would suffer from low probability of hitting a correct guess (as visible in the first row of Table 2, uniform distribution for sentinels being a Zipf with $\alpha = 0$).

4.3 Multiple workers

The analysis above has considered an individual worker, in line with the fact that each sentinel controls the behavior of a single worker. The effectiveness of sentinels control on the overall system is simply the combination of the controls over the different workers. The probability of possible lazy behavior in the overall system to go undetected (i.e., the probability that all workers pass sentinels control) is then:

$$p\_pass\_sentinels = \prod_{i=1}^{w} p\_pass\_sent_i$$

where $p\_pass\_sent_i$ is the value of p_pass_sent computed with the specific values of j jobs, s sentinels, and o omissions considered for worker $\mathtt{w}_i$. When the parameters are the same for all workers, $p\_pass\_sentinels$ reduces to $(p\_pass\_sent)^l$, with l the total number of lazy workers. In fact, p_pass_sent=1 for workers that are either honest (i.e., o=0) or that receive no sentinels (i.e., s=0).

Note that, since the client does not know nor suspect which workers are honest and which workers are lazy, sentinels are uniformly distributed among all the workers. The reason for workers to receive no sentinels might be that the total number of sentinels assumed to be injected overall is smaller than the number of workers. In fact, as we will see in Section 6, even a few sentinels covering just a few workers are sufficient and provide adequate effectiveness when combined with twins, with no need to distribute sentinels to every single worker. We also note that sentinels, operating independently on different workers are by definition not

TABLE 3
Probability p_guess_twin of correctly classifying a job by 4 or 10 workers that omitted it, and probability p_same_wrong of 10 workers returning the same (wrong) class for the omitted job

P_D	Prob. same classification	P_O [1,0,0,0]	P_O [0.25,0.25,0.25,0.25]	P_O as P_D
$\alpha = 0$	$p\_guess\_twin(4)$	0.2500	0.0039	0.0039
[0.25,0.25,0.25,0.25]	$p\_guess\_twin(10)$	0.2500	9.53e-7	3.81e-6
	$p\_same\_wrong(10)$	0.7500	2.86e-6	2.86e-6
$\alpha = 1$	$p\_guess\_twin(4)$	0.4800	0.0039	0.0264
[0.48,0.24,0.16,0.12]	$p\_guess\_twin(10)$	0.4800	9.53e-7	6.50e-4
	$p\_same\_wrong(10)$	0.5200	2.86e-6	3.38e-4
$\alpha = 7$	$p\_guess\_twin(4)$	0.9920	0.0039	0.9606
[0.992,7.75e-3,4.53e-4,6.053e-5]	$p\_guess\_twin(10)$	0.9920	9.53e-7	0.9228
	$p\_same\_wrong(10)$	0.0080	2.86e-6	0.0074

exposed to collusion. As a final remark, we note that the total number of sentinels that may be injected might be not sufficient to cover all the classes in the data distribution. In this case, maintaining the idea of distributing sentinels as uniformly as possible in the different classes, the best client strategy would be to randomly select classes to which assign sentinels.

5 TWIN ANALYSIS

We now analyze the effectiveness of integrity assessment through twins. We recall that: the term *twin set* refers to the set of r's (virtual) replicas of a data item, T is the overall number of twin sets generated by the client, and t is the number of twins assigned to each worker. With uniform distribution of twins to workers, we have $\lfloor \frac{T \cdot r}{w} \rfloor \leq t \leq \lceil \frac{T \cdot r}{w} \rceil$. For simplicity of the formulas, without loss of generality, we assume $T \cdot r$ to be a multiple of the number w of workers and hence each worker to receive t twins. (We relaxed such an assumption in computing results for plotting curves of our analysis and in our simulations.)

We first evaluate the probability of workers to go undetected by twins control when omitting jobs (Section 5.1). With such probability turning out to be dependent on the replication factor and the number of twin sets, we then analyze possible strategies for twin generation and allocation (Section 5.2). We first focus on a single twin set because each twin set works independently for providing integrity guarantees. We then generalize the results to the consideration of multiple twin sets.

5.1 Probability analysis

Workers to which data items in a twin set are assigned pass integrity control by the twin set if they all return the same result for the items in the set (twin jobs). This happens in the following cases: *1)* all workers actually perform the twin job (and hence return the correct result for it), *2)* some of the workers omit the twin job but correctly guess its result, or *3)* all workers omit the twin job but they all return the same result for it. Let us then analyze the probability of lazy workers to be involved in the evaluation of a twin set and to omit the twin. We then evaluate the probability of workers involved in a twin set to return the same result, when some, or all, of them are lazy.

Probability of lazy workers in the twin set. The first parameter influencing twins effectiveness is the probability of lazy (vs honest) workers to be involved in the evaluation of a twin set. In fact, honest workers, by always returning a correct response, will increase exposure of possible misbehaviors by others. The probability of having lazy workers in the twin set depends on the number of lazy workers in the system and on the replication factor (i.e., the cardinality of the twin set). With a reasoning similar to the one followed for sentinel omissions, it is easy to see that the probability of having l_T lazy workers involved in a twin set assuming w workers in the system, l of which are lazy, and a replication factor of r, follows a hypergeometric distribution. Formally:

$$p\_lazy(l_T) = \frac{\binom{l}{l_T} \cdot \binom{w-l}{r-l_T}}{\binom{w}{r}}$$

Probability of omitting twins. Suppose that l_T out of the r workers in charge of evaluating a twin set are lazy. The probability that a lazy worker omits a twin job is $\frac{o}{j}$ (i.e., number of omissions divided by the number of jobs assigned to the worker). Since omissions by different workers are independent events, probability $\left(\frac{o}{j}\right)^i \cdot \left(1 - \frac{o}{j}\right)^{l_T - i}$ corresponds to the probability that i, with $0 \leq i \leq l_T$, workers omit the twin job and $l_T - i$ workers do not omit it. Given l_T lazy workers, there are $\binom{l_T}{i}$ possible combinations of i out of l_T lazy workers and therefore the probability that any subset of i lazy workers omits the twin job in the same twin set is:

$$p\_omit\_twin(i, l_T) = \binom{l_T}{i} \cdot \left(\frac{o}{j}\right)^i \cdot \left(1 - \frac{o}{j}\right)^{l_T - i}$$

Probability of same classification. In the evaluation of the probability of all workers to return the same classification, we distinguish the case where such classification is the correct one for the twin job from the case where it is a wrong one (but consistently returned by all workers). The probability of a worker to correctly guess the class of an omitted job is $\sum_{k=1}^{c} (P_O(\mathtt{c}_k) \cdot P_D(\mathtt{c}_k))$. The probability that i, with $0 \leq i \leq l_T$, workers guess the correct class of the omitted twin job is then:

$$p\_guess\_twin(i) = \sum_{k=1}^{c} \left((P_O(\mathtt{c}_k))^i \cdot P_D(\mathtt{c}_k) \right)$$

With a similar reasoning, the probability that i, with $0 \leq i \leq l_T$, workers omitting a job return the same (but wrong) result is:

$$p\_same\_wrong(i) = \sum_{k=1}^{c} \left((P_O(\mathtt{c}_k))^i \cdot (1 - P_D(\mathtt{c}_k)) \right)$$

Table 3 illustrates the probability of workers omitting the job to return the same class (correct for p_guess_twin and wrong for p_same_wrong) assuming different probability mass functions for genuine data P_D and for omitted jobs P_O. Note that, in the extreme case where a twin set is assigned to workers that are all lazy and all of them omit the job, if workers are deterministic in

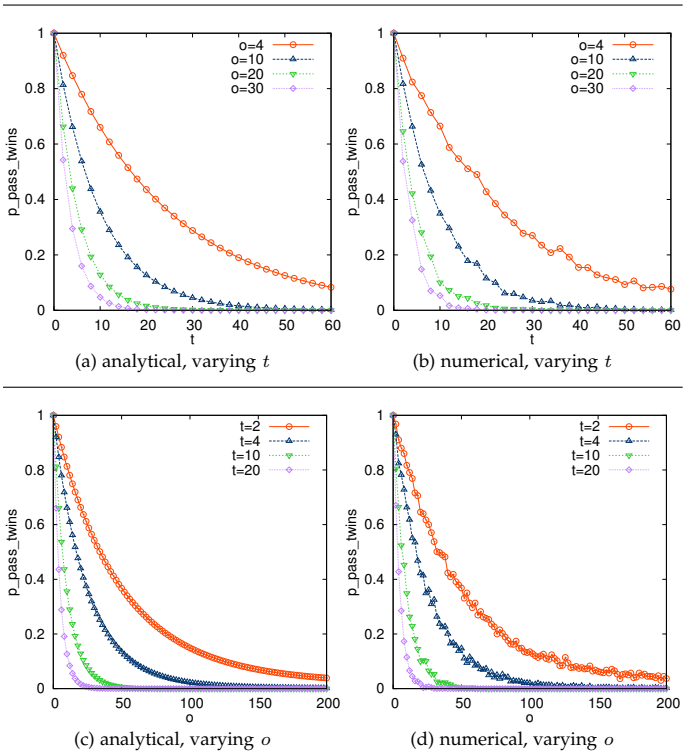


Fig. 5. Probability p_pass_twins varying the number t of twins (a,b) and the number o of omitted jobs (c,d) obtained analytically (a,c) and through Monte Carlo simulations (b,d)

their classification of omissions (e.g., classifying all items in the most frequent class in P_D), the misbehavior will not be discovered since all workers will be in agreement on their result (regardless of whether it is correct or wrong). This is visible in the table where, for P_O=[1,0,0,0], p_guess_twin(10)+p_same_wrong(10)=1, regardless of the distribution of P_D.

Table 3 is analogous to Table 2 for sentinels. Note, however, that rows in Table 2 report the distribution of sentinels while rows in Table 3 report the distribution of data themselves. In fact, since twins are randomly selected in the data population, unlike for sentinels, their distribution cannot be regulated. For the randomness of the process, distribution of twins can be considered to follow the distribution of the genuine data.

Probability of passing twins control. The probability of a twin set to show the same result from all workers and therefore not to signal any violation (even in the presence of lazy workers and omissions) can be obtained by applying the total probability theorem to the probabilities discussed above. More precisely, it is the sum of: *1)* the probability of having all workers involved in the twin set to return the correct result, even when some workers are lazy and omit the twin job, and *2)* the probability of all workers involved in the twin set to be lazy, to omit the twin job, and to return the same (wrong) result. The first probability is the sum, over all possible values of l_T, of the probability of having l_T lazy workers involved in the twin set multiplied by the probability of i of them omitting the job to guess the correct result (for all possible values of i). The second probability is the product of the probability of the different events concurring to it. Therefore, the probability of observing the same result for all jobs in a twin set is:

$$p\_consistent = \sum_{l_T=0}^{\min(l,r)} \Big(p\_lazy(l_T) \cdot \sum_{i=0}^{l_T} (p\_omit\_twin(i, l_T) \cdot p\_guess\_twin(i)) \Big) + p\_lazy(r) \cdot p\_omit\_twin(r, r) \cdot p\_same\_wrong(r)$$

Note that the sum terminates at $min(l,r)$ since a twin set cannot include more lazy workers than either the number of lazy workers assumed to be part of the system or the number of replicas in the twin set. When T twin sets are used, the probability, denoted p_pass_twins, of twins not to signal any violation is:

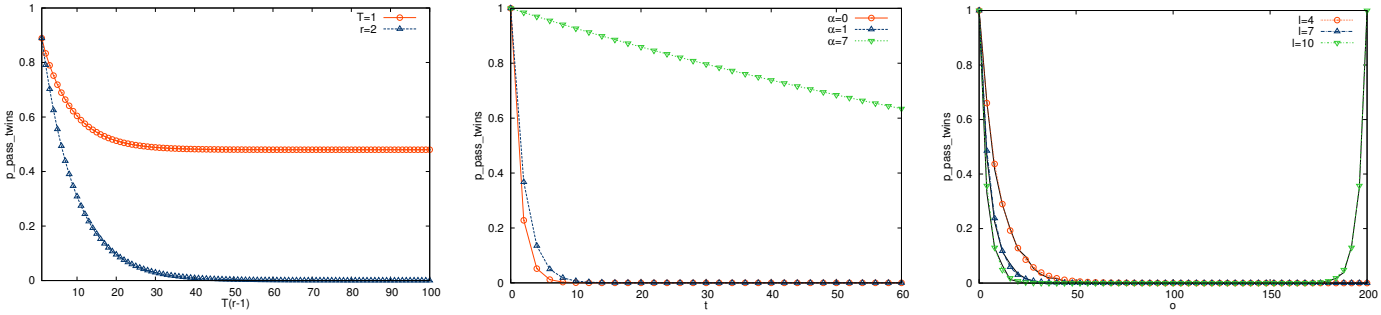
$$p\_pass\_twins = p\_consistent^{\,T}$$

Figure 5 compares the values obtained for p_pass_twins, varying the number t of twins per worker and the number o of omitted jobs, obtained analytically and through 1000 Monte Carlo simulations. The scenario assumes that each worker is in charge of the evaluation of j=200 jobs, some of which (2, 4, 10, or 20) are twins. Data are distributed according to Zipf's law with α=1 (i.e., P_D has α=1) in 4 classes $C = \{c_1, c_2, c_3, c_4\}$. Lazy workers classify the omitted jobs in the most frequent class in P_D (i.e., P_O is [1,0,0,0]).

5.2 Twin generation strategy

The client can operate on two factors when injecting twins control: the number of twin sets (T) and the replication factor (r), with the two being also related to the number of twin jobs assigned to each worker ($t \cdot w$=$T \cdot r$). Indeed, other parameters in the formula of p_pass_twins are not under the control of the client. Clearly, the more twins a worker receives, the more the control to which the worker is subject. But, assuming willingness of the client to pay a load of $T \cdot (r-1)$ additional jobs (twin jobs in addition to the original ones), is it better to have a larger T or a larger r? For instance, would it be better to have three twins for the same data item (T=1, r=3) or two twins of two data items (T=2, r=2)? Each of the two strategies would bring an equal additional load to the system (two additional jobs to be allocated), but which one is more effective?

Looking at the formula of p_pass_twins, and the dependency between the variables, it is easy to see that p_pass_twins decreases exponentially with the increase of the number of twin sets T (which is larger for smaller values of r). It decreases instead very slowly with the increase of the number r of replicas, as visible in Figure 6(a). The figure shows how the probability that an omission of o=50 jobs, assuming P_D with $\alpha = 1$, w=100 and l=49 (i.e., just the slight majority of the workers is honest), goes undetected, varying the additional jobs inserted as twins (i.e., $T \cdot (r-1)$) in the two extreme cases: *1)* only one twin set is considered (i.e., T=1) while the number of replicas varies, *2)* only one additional replica per item is considered (i.e., r=2) and the number of twin sets varies. The figure confirms the observation above. In particular, the curve for T=1 shows



(a) o=50, w=100, l=49 (b) o=20, w=10, l=4 (c) w=10, t=10

Fig. 6. Probability p_pass_twins varying the number $T \cdot (r-1)$ of additional jobs inserted as twins (a), the number t of twins (b), and the number o of omitted jobs (c)

an asymptotic behavior, tending to P_{max}, which is the probability of the most frequent class in P_D (P_{max}=0.48 in the figure). In fact, with probability P_{max}, the jobs in the twin set belong to the most frequent class. In this case, the workers correctly guess the class of twin jobs. Such a correct guess then happens with probability P_{max}.

In summary, best effectiveness is achieved by keeping replication factor minimum (r=2) while increasing the number of twin sets (i.e., twinning different data items). Of course, care must be taken in the allocation of twins to workers, to guarantee that the cross-checks provided by twins do not partition workers dividing lazy from honest workers. Such condition can be easily ensured by assigning at least one twin to every worker, uniformly distributing twins between workers, and allocating twin sets to cover different combinations of workers. Good coverage of the different combinations provides also fine resilience against collusion. In this respect, we note that even if a fraction $\frac{1}{x}$ of the workers collude, only the twin sets fully covered by the colluding group become ineffective, roughly less than $\frac{1}{x^2}$ of the twin sets, hence leaving the lazy workers still exposed to the control on the other twin sets touching workers outside the clique (see Section 7).

The effectiveness of twins depends also on the probability mass function P_D and on the strategy adopted by the workers for classifying the omitted jobs (i.e., P_O). As already noted, since twins are selected randomly among the data items, the client cannot dictate twin distribution into classes as done for sentinels and, given the randomness of the process, twins can be expected to follow the same distribution P_D as the original data. Hence, the best strategy for lazy workers behaving opportunistically to the aim of going undetected in twin omissions is to classify omitted jobs in the most frequent class of the data distribution. This observation implies, unfortunately, determinism in opportunistic behavior of lazy workers and therefore their equal classification of omitted twin jobs that can make twin control ineffective when data distribution is highly skewed (as the most frequent class of data that lazy workers assign to twin jobs has high probability to be the correct one, hence matching the result returned by honest workers) or where all the workers are lazy and their omission extreme (as they will all omit twins, consistently classifying them in the same class). Figures 6 (b) and (c) illustrate the effectiveness of twins as well as their limitations in such situations.

Figure 6(b) shows p_pass_twins varying the number t of twins between 0 and 60 and assuming r=2, w=10 workers (4 of which are lazy), o=20 omitted jobs by each lazy worker, and j=200 jobs assigned to each worker. While quickly effective for Zipf's data distributions with α=0 and α=1, twins fail to be effective for α=7. Figure 6(c) considers a scenario with w=10 workers each in charge of 200 jobs, 10 of which are twins (with r=2 and T=50), and the distribution of data is expressed by a Zipf's law with α=1. It shows how probability p_pass_twins changes varying the number o of omitted jobs, assuming the number l of lazy workers to be 4, 7, or 10 (i.e., 40%, 70%, or 100%). Again, while quickly effective even when the majority of workers (70%) is lazy and their omissions extreme (i.e., approaching all 200 jobs), twins cannot help in detecting extreme omissions (i.e., approaching all the 200 jobs) when all workers are lazy. In this case, none of the workers would perform the work and all of them would consistently return the most probable class (i.e., the most frequent in the data distribution).

6 Combining Sentinels and Twins

In the previous sections, we have discussed sentinels and twins independently. When both techniques are used, the overall integrity guarantee is given by their combination. Hence, the probability of workers to pass integrity control becomes:

$$p\_undetected = p\_pass\_sentinels \cdot p\_pass\_twins$$

We now investigate how the two techniques should be used to provide best effectiveness. Our starting point is represented by the lessons learned from the analysis in the previous sections, which can be summarized with the following three observations. First, the best strategy for the client to maximize effectiveness of control is to distribute sentinels uniformly among the different classes, and to use a replication factor r of 2 for twins. Second, the best strategy for opportunistic lazy workers to maximize probability of going undetected when omitting jobs is to classify omissions in the most frequent class of the data distribution (i.e., the class with probability P_{max}). Third, twins are, in general, more effective than sentinels (roughly twice as effective, since with one additional job the behavior of two workers is controlled). However, twins fall short when the data distribution is highly skewed (i.e., P_{max} is high) given the

high probability of omitted jobs to be classified correctly. Also, twins lose effectiveness when omissions are extreme by all workers (since all lazy workers would return the same result for the twin jobs assigned to them which they omit to compute).

With the analysis in the previous sections enabling us to capture integrity guarantees (in terms of $p\_undetected$), our challenge is first to determine the value of P_{max} where sentinels take over in terms of effectiveness with respect to twins and second to determine how to counteract extreme omissions injecting some sentinels in scenarios where twins are more effective.

6.1 Threshold value of P_{max}

The value of P_{max} for which either twins or sentinels are expected to be more effective depends on two factors: the presence of lazy workers and the percentage of omissions. Identifying whether the P_{max} of a given scenario falls below or above such an optimal value would require to plot the curves of $p\_pass\_sentinels$ and p_pass_twins for the given scenario and see the value of P_{max} at which one becomes higher than the other. Considering the three observations above and some simplifying assumptions that enable analytical treatment, we obtain an analytical formula for the derivation of the threshold value of P_{max} indicating whether twins or sentinels are more effective.

As for sentinels, where $p\_pass\_sentinels = \prod_{i=1}^{w} p\_pass\_sent_i$ over the different workers, we simplify p_pass_sent assuming the omissions of sentinels as a sampling with replacement. Sampling with replacement means that the sample values are independent. In our scenario, this means that the omission of a job does not affect subsequent omissions. In other words, when we omit a job, that job is put back in the set of jobs; subsequent omissions operate considering the complete set of jobs. Hence, we consider for each worker the formula for one sentinel and elevate it to the number of sentinels assigned to the worker. The imprecision introduced by the adoption of this simplification is negligible and has the advantage that the formula exhibits a clear analytical structure. The overall product over the different workers equates then to elevating the p_pass_sent for a single sentinel to the total number of sentinels in the system, that is, $p\_pass\_sentinels = p\_pass\_sent^{s_{tot}}$. For a single sentinel ($s = 1$), the formula for p_pass_sent of Section 4.1 gives us:

$$
\begin{aligned}
&p\_pass\_sent = \sum_{o_s=0}^{1} (p\_omit\_sent(o_s) \cdot (p\_guess\_sent)^{o_s}) \\
&= p\_omit\_sent(0) + p\_omit\_sent(1) \cdot p\_guess\_sent \\
&= 1 - \frac{o}{j} + \frac{o}{j} \cdot \frac{1}{c} = 1 - \frac{o}{j} \cdot \left(1 - \frac{1}{c}\right)
\end{aligned}
$$

where $p\_guess\_sent = \frac{1}{c}$ derives from injecting in the formula of Section 4.1 the observations above about the client distributing sentinels uniformly and workers opportunistically returning the most frequent class for omitted jobs.

As for twins, where $p\_pass\_twins = p\_consistent^T$, we can rewrite the formula for $p\_consistent$ applying the observations above on the replication factor (i.e., $r = 2$) and considering opportunistic behavior by lazy workers for omitted jobs (i.e., they consistently classify the omitted jobs in the most frequent class of the data distribution). As a simplification, we consider the worst case scenario of both workers involved in the twin set to be lazy (i.e., l_T=2) and, for the single twin set, assume them to be the only workers in the system (i.e., w=2). The exponentiation to T will take care of the inclusion of the other workers. Basically, our simplification assumes twin sets to operate independently (covering disjoint sets of pairs of lazy workers). The formula for $p\_consistent$ of Section 5.1 then becomes:

$$
\begin{aligned}
&p\_consistent = \\
&p\_lazy(2) \cdot \sum_{i=0}^{2} (p\_omit\_twin(i,2) \cdot p\_guess\_twin(i)) + \\
&+ p\_lazy(2) \cdot p\_omit\_twin(2,2) \cdot p\_same\_wrong(2) = \\
&= \left(1 - \frac{o}{j}\right)^2 + \left[2 \cdot \frac{o}{j} \cdot \left(1 - \frac{o}{j}\right)\right] \cdot P_{max} + \left(\frac{o}{j}\right)^2 \cdot P_{max} + \\
&+ \left(\frac{o}{j}\right)^2 (1 - P_{max}) = 1 - 2 \cdot \left[\frac{o}{j} - \left(\frac{o}{j}\right)^2\right] \cdot (1 - P_{max}) \\
&\approx 1 - 2 \cdot \frac{o}{j} \cdot (1 - P_{max})
\end{aligned}
$$

where the last approximation removes the least significant factor since, assuming $o \ll j$, term $(\frac{o}{j})^2$ becomes negligible.

Let us then compare the formulas obtained above to evaluate when sentinels are more effective than twins, that is, when the probability of passing sentinel controls is lower than the one of passing twin controls ($p\_pass\_sentinels < p\_pass\_twins$). Of course, comparison is to be made assuming the same overhead for the client in terms of additional jobs to be inserted, hence considering an equal number of sentinels and twin sets, that is, $s_{tot} = T$ (as each of them requires an additional job to be injected in the system). With $s_{tot} = T$, the exponents at both sides of the equation can be discarded and comparison reduces to check when $p\_pass\_sent < p\_consistent$, that is, when:

$$
1 - \left(\frac{o}{j}\right) \cdot \left(1 - \frac{1}{c}\right) < 1 - 2 \cdot \left(\frac{o}{j}\right) \cdot (1 - P_{max})
$$

which gives:

$$
P_{max} > \frac{1}{2} \cdot \left(1 + \frac{1}{c}\right)
$$

In summary, our analysis tells us that, for each scenario, either sentinels or twins should be used as control jobs, depending on the value of P_{max} characterizing the scenario. When $c = 2$, which is the lowest value c can assume, twins (sentinels, resp.) should be used if P_{max} is lower (higher, resp.) than or equal to 0.75. As the number of classes grows the value of P_{max} at which sentinels are more effective than twins decreases, reaching 0.50 as the number of classes becomes very large (and hence $\frac{1}{c}$ negligible). The formula above also tells us that when P_{max} is not higher than 0.50, twins are always more effective than sentinels.

The number of control jobs to be injected depends on the aimed integrity guarantees, expressed in terms of $p\_undetected$ and can be simply obtained from our formulas. Basically, given a client's established threshold ε, $p\_undetected < \varepsilon$ can be guaranteed by employing either $T > \frac{\log(\varepsilon)}{\log(p\_consistent)}$ twin sets or $s_{tot} > \frac{\log(\varepsilon)}{\log(p\_pass\_sent)}$ sentinels (Section 6.3 will elaborate more on this). A note

aside is to be made for scenarios where twins are more effective than sentinels (i.e., $P_{max} < \frac{1}{2} \cdot (1 + \frac{1}{c})$) with respect to extreme omissions, which make twins lose effectiveness. Luckily, the different nature of the two controls makes sentinels extremely effective in such scenarios, and adding a handful of sentinels when using twins suffices to detect extreme omissions, as we see next.

6.2 Extreme conditions

Extreme conditions refer to (quite unlikely) scenarios where omissions by all the workers in the system are considerable, approaching almost the total number of jobs.

The most extreme case where all jobs are omitted (i.e., all workers omit all the jobs) is easy to analyze. In such a case, p_pass_twins=1 (since all omissions will be classified in the most frequent class of the data distribution). Also, since all jobs are omitted (i.e., o=j and o_s=s), $p\_pass\_sentinels$ reduces to $(\frac{1}{c})^{s_{tot}}$ (i.e., p_pass_sent=$\frac{1}{c}$). In other words, a number of sentinels $s_{tot} > -\frac{\log(\varepsilon)}{\log(c)}$ is sufficient to maintain $p\_undetected < \varepsilon$, guaranteed by twins, also when all workers omit all their jobs.

The analysis of generic extreme conditions where workers omit most, but not all, of their jobs is much more complex. To tackle it, we consider the worst case scenario, modeling the whole set of workers as a single worker. We also assume workers to know the total number s_{tot} of sentinels and the number T of twin sets, and therefore to be able to estimate the number $\bar{E}$ of jobs to elaborate to maximize the probability that their extreme omissions go undetected. Providing protection against the worst case scenario clearly gives the strongest protection, also maintained in situations where such information is not known and workers operate independently in their opportunistic behaviors. When workers omit most of the jobs, the omissions go undetected if the workers process all the sentinels and do not process any of the twins. The probability, called $p\_extreme$, that, when processing only E out of the total number J of jobs in the system while omitting the other $O = J - E$ jobs, all the sentinels are processed and all the twins are omitted is:

$$p\_extreme(E) = \binom{E}{s_{tot}} \cdot \left(\frac{s_{tot}}{J}\right)^{s_{tot}} \cdot \left(1 - \frac{s_{tot} + 2T(1 - P_{max})}{J}\right)^{E - s_{tot}}$$

where $\left(\frac{s_{tot}}{J}\right)^{s_{tot}}$ is the probability of processing all the sentinels and $\left(1 - \frac{s_{tot} + 2T(1-P_{max})}{J}\right)^{E-s_{tot}}$ is the probability that, among the $E - s_{tot}$ processed jobs, twin jobs that do not fall in the most frequent class are all omitted. The rationale is that among the $2T$ twin jobs, only $2T(1 - P_{max})$ twin jobs that do not fall in the most frequent class are effective in signaling omissions (for twins that fall in the most frequent class, omissions cannot be distinguished from processing as they produce the same result).

To determine the maximum value of $p\_extreme$, denoting with $\bar{E}$ the value of E reaching it, we observe that to process as few E jobs as possible while minimizing the risk of being detected, opportunistic lazy workers need to aim for the lowest E such that $p\_extreme(E) > p\_extreme(E + 1)$. Considering the formula of $p\_extreme$ this translates to:

$$\binom{E}{s_{tot}} \cdot \left(\frac{s_{tot}}{J}\right)^{s_{tot}} \cdot \left(1 - \frac{s_{tot} + 2T(1 - P_{max})}{J}\right)^{E - s_{tot}} > \binom{E+1}{s_{tot}} \cdot \left(\frac{s_{tot}}{J}\right)^{s_{tot}} \cdot \left(1 - \frac{s_{tot} + 2T(1 - P_{max})}{J}\right)^{E+1-s_{tot}}$$

which corresponds to:

$$1 > \frac{E+1}{E+1-s_{tot}} \cdot \left(1 - \frac{s_{tot} + 2T(1 - P_{max})}{J}\right)$$

giving:

$$E > \frac{s_{tot} \cdot J}{s_{tot} + 2T(1 - P_{max})} - 1$$

Since the number s_{tot} of sentinels to be added is considerably smaller than the number T of twin sets and the number J of jobs is large (especially when compared with the total number of sentinels and twin sets), the value $\bar{E}$ that maximizes $p\_extreme(E)$ can be approximated to $\bar{E} = \frac{s_{tot} \cdot J}{2T(1-P_{max})}$. The probability of workers to be undetected when processing only $\bar{E}$ jobs is then:

$$p\_extreme(\bar{E}) = \binom{\frac{s_{tot} \cdot J}{2T(1-P_{max})}}{s_{tot}} \cdot \left(\frac{s_{tot}}{J}\right)^{s_{tot}} \cdot \left(1 - \frac{s_{tot} + 2T(1 - P_{max})}{J}\right)^{\frac{s_{tot} \cdot J}{2T(1-P_{max})} - s_{tot}}$$

which can be then simplified applying the following steps:

- $\binom{\frac{s_{tot} \cdot J}{2T(1-P_{max})}}{s_{tot}} \leq \frac{\left(\frac{s_{tot} J}{2T(1-P_{max})}\right)^{s_{tot}}}{s_{tot}!}$ which, using Stirling's approximation for the factorial at the denominator, can be approximated as $\frac{\left(\frac{s_{tot} J}{2T(1-P_{max})}\right)^{s_{tot}}}{\sqrt{2\pi s_{tot}}\left(\frac{s_{tot}}{e}\right)^{s_{tot}}}$
- since $s_{tot} \ll J$, $\left(1 - \frac{s_{tot} + 2T(1-P_{max})}{J}\right)^{\frac{s_{tot} \cdot J}{2T(1-P_{max})} - s_{tot}} \approx \left(1 - \frac{2T(1-P_{max})}{J}\right)^{\frac{s_{tot} \cdot J}{2T(1-P_{max})}}$ which, applying rule $(1 + \frac{y}{x})^x \approx e^y$, can be approximated as $e^{-s_{tot}}$.

The maximum value of $p\_extreme$ can then be approximated as:

$$p\_extreme(\bar{E}) = \frac{\left(\frac{s_{tot} J}{2T(1-P_{max})}\right)^{s_{tot}}}{\sqrt{2\pi s_{tot}}\left(\frac{s_{tot}}{e}\right)^{s_{tot}}} \cdot \left(\frac{s_{tot}}{J}\right)^{s_{tot}} \cdot e^{-s_{tot}} = \frac{\left(\frac{s_{tot}}{2T(1-P_{max})}\right)^{s_{tot}}}{\sqrt{2\pi s_{tot}}}$$

The validity of this formula has been verified using simulations in a variety of configurations.

For guaranteeing a probability of extreme omissions to go undetected lower than ε' (where ε' could be equal to or different from the ε required for $p\_undetected$), the number s_{tot} of sentinels to inject to maintain the protection guarantees provided by twins can be then derived instantiating the formula above with the P_{max} of the considered scenario and T computed as illustrated at the end of Section 6.1. Again, as we see next, a handful of sentinels overall suffices.

TABLE 4
Probability $p\_undetected$ for a scenario with 10^6 jobs for different P_{max}, and varying the percentage of omissions and of control jobs injected

$\frac{O}{J}$	P_{max}	Additional jobs 5%	10%	15%	20%	25%
1%	0	2.07e-439	4.18e-878	8.41e-1317	1.69e-1755	3.41e-2194
	0.25	9.80e-330	9.31e-659	8.85e-988	8.32e-1317	7.91e-1646
	≥0.50	4.63e-220	2.07e-439	9.31e-659	4.18e-878	1.85e-1097
0.1%	0	3.38e-44	1.14e-87	3.82e-131	1.29e-174	4.33e-218
	0.25	2.49e-33	6.19e-66	1.54e-98	3.82e-131	9.49e-164
	≥0.50	1.84e-22	3.38e-44	6.19e-66	1.14e-87	2.08e-109
0.01%	0	4.54e-05	2.06e-09	9.33e-14	4.23e-18	1.92e-22
	0.25	5.53e-04	3.06e-07	1.69e-10	9.33e-14	5.16e-17
	≥0.50	6.74e-03	4.54e-05	3.06e-07	2.06e-09	1.39e-11

TABLE 5
Probability $p\_undetected$ for a scenario with 10^8 jobs for different P_{max}, and varying the percentage of omissions and of control jobs injected

$\frac{O}{J}$	P_{max}	Additional jobs 5%	10%	15%	20%	25%
1%	0	2.44e-43870	5.83e-87740	1.39e-131609	3.33e-175479	7.96e-219349
	0.25	6.20e-32903	3.77e-65805	2.29e-98707	1.39e-131609	8.47e-164512
	≥0.50	1.58e-21935	2.44e-43870	3.77e-65805	5.83e-87740	9.02e-109675
0.1%	0	5.10e-4348	2.59e-8695	1.32e-13042	6.71e-17390	3.41e-21737
	0.25	3.39e-3261	1.15e-6521	3.89e-9782	1.32e-13042	4.47e-16303
	≥0.50	2.26e-2174	5.10e-4348	1.15e-6521	2.59e-8695	5.85e-10869
0.01%	0	4.59e-435	2.11e-869	9.69e-1304	4.45e-1738	2.04e-2172
	0.25	1.76e-326	3.11e-652	5.49e-978	9.69e-1304	1.71e-1629
	≥0.50	6.78e-218	4.59e-435	3.11e-652	2.11e-869	1.43e-1086

TABLE 6
Upper number of jobs that can be omitted by workers with probability of being detected below 10^{-10} for the scenarios of Table 5

P_{max}	Additional jobs 5%	10%	15%	20%	25%
0	231	116	77	58	47
0.25	308	154	103	77	62
≥0.50	461	231	154	116	93

6.3 Integrity guarantees

The analysis above enables us to evaluate effectiveness of integrity controls for realistic scenarios characterized by large workloads. Indeed, Sections 4 and 5 considered a relatively small workload ($J = 2000$ and $w = 10$), with the goal to understand the behavior of the techniques, rather than demonstrating their effectiveness. We have then evaluated $p\_undetected$ in configurations characterized by different numbers of jobs and varying: *i)* the percentage of control jobs (from 5% to 25%); *ii)* the percentage of omitted jobs (1%, 0.1%, and 0.01%); and *iii)* the probability P_{max} of the most frequent class (0, 0.25, and ≥0.50). The value of P_{max} determined also the kind (twin sets vs sentinels) of control jobs to be injected. As we assumed a large number of classes, $P_{max} = 0.50$ is the threshold where twins hand over to sentinels for effectiveness and therefore the values exhibited for the case $P_{max} \geq 0.50$ assumes use of sentinels, noting that the case $P_{max} = 0.50$ applies to twins as well. Also, since effectiveness of sentinels does not depend on P_{max} (given that sentinels are uniformly distributed), the exhibited values for the case $P_{max} \geq 0.50$ hold for all $P_{max} > 0.50$.

Tables 4 and 5 show the probability of omissions to go undetected for two scenarios characterized by 10^6 and 10^8 jobs, respectively, varying the different parameters. For instance, just injecting 5% of additional control jobs provides a $p\_undetected$ (i.e., the probability of omissions to go undetected) varying from 4.59e-435 to 6.78e-218 (depending on P_{max}) when omissions are up to 0.01% of the 10^8 jobs. Clearly, increasing the amount of controls or as omissions increase makes such a probability lower (with a value of 7.96e-219349, when 25% control jobs are added and omissions hit 1% of the jobs). Similarly, increasing the overall number of jobs leads, for the same percentage increase in integrity cost, to an improvement in the effectiveness of integrity verification (with 10^6 jobs, $p\_undetected$ varies, injecting 5% additional control jobs, from 4.54e-05 to 6.74e-03 when omissions are up to 0.01%). Table 6 shows the (impressively low) absolute maximum number of jobs that workers can afford to omit while maintaining the probability of being undetected higher than 10^{-10}. For instance, even when just 5% control jobs are injected, workers will be detected with probability greater than $1-10^{-10}$ if omissions are more than 231, 308, or 461 (depending on P_{max}).

For evaluating the number of sentinels to be injected to provide protection against extreme omissions when using twins (i.e., when $P_{max} \leq 0.50$), we imposed $p\_extreme < 10^{-10}$. The number of sentinels varied from a maximum of 5 (for a scenario with 10^5 jobs) to just 1 (for scenarios with 10^{12} jobs or above). The number of sentinels was either 2 or 3 for the scenario with 10^6 jobs and consistently 2 for the scenario with 10^8 jobs, for all the configurations in Tables 4 and 5.

7 Workers collusion

In the previous sections we have assumed workers not to collude. Collusion may occur if the workers appear to be independent, but are all executing under the control of a same party. As we will discuss in this section, the techniques are resilient to collusion. Indeed, it is sufficient to take into consideration the fact that in presence of collusion some of the control jobs (twins) can become ineffective. Sentinels, being unique, are clearly not affected by collusion, since no worker will have a sentinel in common with another worker. By contrast, twins are exposed to being recognized as such by colluding workers, which can agree on the same (not computed) response to be returned for the jobs so to go undetected in their omission. While noting that, if the characteristic of the computation allows it, the ability of colluding workers to recognize twins can be counteracted by making twins different one from the other (e.g., for image recognition altering the twinned copies in some ways without altering the result of the computation), we assume common twins to be completely recognizable by colluding workers. In the following, we first investigate how the effectiveness of twins changes and then discuss how colluding workers can be discovered.

Analysis. Assume that l_C of the l lazy workers (of the total of w workers) collude, forming a coalition. Considering a replication factor $r = 2$, a twin set can be recognized by the colluding workers only if both jobs are allocated to them (while twin sets for which at least one of the jobs is outside

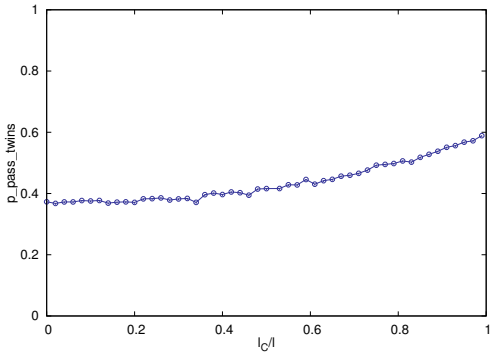


Fig. 7. Probability p_pass_twins varying the percentage of colluding workers (l_C) with respect to lazy workers (l) obtained through Monte Carlo simulations

the coalition cannot be recognized as such). The probability of a twin set to be recognized as such by a coalition is then:

$$p\_twin\_coalition = \frac{\binom{l_C}{2}}{\binom{w}{2}}$$

corresponding to a number of twin sets T_C under the control of the coalition equal to $T_C = T \cdot p\_twin\_coalition$. For instance, if a coalition controls half of the workers, around $\frac{1}{4}$-th of the twins will be controlled; this means that $\frac{3}{4}$-ths of the twins will still be effective.

Colluding workers can opportunistically behave on the discovered twins by omitting them and agreeing on the response to be returned for each twin set (while omitting the work). The formulas in Sections 4 and 5 continue then to hold removing, for lazy workers, the number of twinned jobs under the control of the coalition from both the number o of omitted jobs and the total number j of jobs. Assuming twins to be uniformly distributed among workers, on average each colluding worker is assigned $avg\_jobs = \frac{2T_C}{l}$ twins of the $2T_C$ twinned data items allocated to workers in the coalition. Hence, in the formalization presented in Sections 4 and 5, o becomes, for lazy workers, $\max(o - avg\_jobs, 0)$ and j becomes $j - avg\_jobs$.

Figure 7 illustrates the probability p_pass_twins obtained through 10000 Monte Carlo simulations varying the percentage of colluding workers with respect to lazy workers, assuming w=100 workers of which l=49 are lazy, j=200 jobs, T=20 twin sets, r=2, and $\alpha = 1$. As visible from the figure, p_pass_twins grows with the number of lazy workers in the coalition. However, the growth is smooth and less than linear.

Colluding workers identification. To identify colluding workers in presence of a single coalition that includes all the l lazy workers, we exploit the possible inconsistent results returned for twin sets that are not fully covered by colluding workers. Indeed, inconsistent results may only happen when a twin set is processed by a worker in the coalition (lazy worker) and by a worker outside the coalition (honest worker). The basic idea of the approach for discovering lazy workers consists in modeling the inconsistent results returned for twin sets as a graph G where there is a node for each worker $\mathtt{w}_i$ in the system, and an edge ($\mathtt{w}_i$,$\mathtt{w}_k$) iff $\exists \mathtt{d}$ such that $\mathtt{w}_i, \mathtt{w}_k \in \omega(\mathtt{d})$ and $\gamma^i(\mathtt{d}) \neq \gamma^k(\mathtt{d})$. Figure 8(a) illustrates

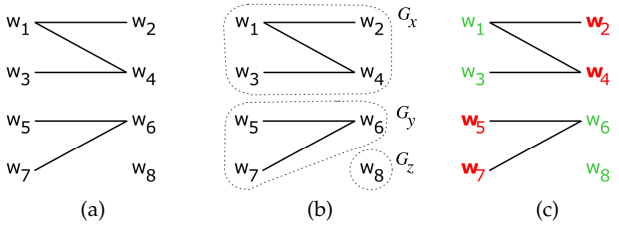


Fig. 8. An example of graph modeling twin mismatch (a), its connected (dotted) components (b), and its (color-coded) partitioning (c)

an example of graph modeling a system with eight workers where, for instance, $\mathtt{w}_1$ and $\mathtt{w}_2$ return a different class for the same data item. For each edge ($\mathtt{w}_i$,$\mathtt{w}_k$) in G, either $\mathtt{w}_i$ or $\mathtt{w}_j$ is lazy. Indeed honest workers compute the same (correct) class, while colluding workers return the same (non computed) class. For instance, considering the graph in Figure 8(a), either $\mathtt{w}_1$ or $\mathtt{w}_2$ is lazy and the other one is honest. Graph G is then *bipartite* and possibly composed of different connected components.

To identify lazy workers, given graph G, the client first computes the connected components of G. For each connected component with more than one node, the nodes in the component are partitioned in two subsets, V_1 and V_2, in such a way that, for each edge ($\mathtt{w}_i$,$\mathtt{w}_k$) in the component, $\mathtt{w}_i \in V_1$ and $\mathtt{w}_k \in V_2$. Finally, for each connected component the client verifies one of the twins corresponding to one edge in the component: the worker whose classification for the verified twin corresponds to the classification computed by the client is considered honest; the other one is lazy. As an example, consider the graph in Figure 8(a). This graph includes three connected components G_x, G_y, and G_z (Figure 8(b)). The single node in G_z is considered honest since it passes all integrity controls. Consider now connected component G_x. Here, the nodes are partitioned in two subsets: $\{\mathtt{w}_1, \mathtt{w}_3\}$ and $\{\mathtt{w}_2, \mathtt{w}_4\}$. The client could then verify the twin between $\mathtt{w}_1$ and $\mathtt{w}_2$: if $\mathtt{w}_1$ is honest also $\mathtt{w}_3$ is honest (with $\mathtt{w}_2$ and $\mathtt{w}_4$ lazy), and vice versa. Similarly, the nodes in the connected component G_y are partitioned in two subsets, namely $\{\mathtt{w}_5, \mathtt{w}_7\}$ and $\{\mathtt{w}_6\}$, and the client only needs to verify one of the two twins in G_y. Figure 8(c) illustrates a possible classification of the workers in the graph in Figure 8(a), where honest workers are green (gray in a b/w print-out) and lazy workers are red (bold in a b/w print-out). After having identified lazy workers, the client can discard the results of the jobs assigned to them, and do not pay them for their services. The jobs in the workload of lazy workers can be assigned to honest workers for their evaluation.

The cost, for the client, to identify lazy workers is linear in the number of twin sets that do not pass the integrity check and requires to recompute a limited number of jobs (one for each connected component in the graph). Considering our example above, the client needs to verify only two out of the five mismatching twins.

8 Experiments

To complete our analysis, we have performed experiments simulating a distributed data computation environment

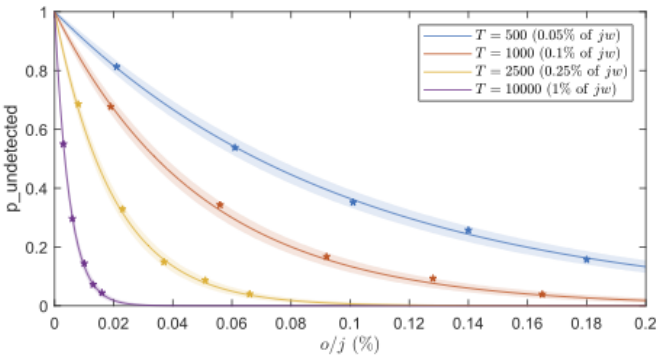


Fig. 9. Probability $p\_undetected$ varying the ratio o/j of omissions from 0.001% to 0.18%, considering different numbers of twin sets

comprising different (potentially lazy) workers. The distributed data computation was controlled by a client program invoking generic functions on an arbitrary number of workers randomly behaving as lazy, and introducing a parameterizable number of twins and sentinels. For the experiments, as an instance of a distributed data computation, we wrote a C program that identifies the lowest result of the application of a cryptographic hash function among a set of input values. (With respect to use in practice, we note that the computation is analogous to the computation that is executed in the Bitcoin network to produce the blocks in the blockchain.) More precisely, each job has as input 1000 values, and the worker has to apply a cryptographic hash function over them and return the value which had the smallest result. Workers behaving lazy were assumed to opportunistically return, for an omitted job, the lowest value in the job's input.

We run experiments on a variety of machines (a server with an Intel Xeon W-2135 3.7 GHz CPU with 6 physical cores; a server with 2 Intel Xeon E5-2620 2.1 GHz CPUs and 16 physical cores; a server with 2 Intel Xeon Gold 5118 2.3 GHz CPUs and 20 physical cores). We have considered 10^6 jobs, each requiring 1000 computations. Jobs were distributed to w=10 workers, with each worker receiving j=10^5 jobs. To note here that, as our analysis has shown, effectiveness of the control depends on the number of overall jobs (i.e., one million for us) rather than the number of workers and individual workload. We considered the use of one sentinel and four different configurations for twins, assuming 500, 1000, 2500, and 10000 twin sets, respectively. In terms of percentage, the different configurations injected from 0.05% to 1% twin jobs, evenly distributed across workers. We then assessed the probability of workers to be undetected when omitting jobs with an omitted ratio (o/j) from 0.001% to 0.18% (which corresponded to theoretical values of $p\_undetected$ distant from 0 and 1).

Figure 9 illustrates the result of our experiments. In the figure, the x axis reports the percentage of jobs omitted by workers and the y axis the value of $p\_undetected$. Observations for each twin set configuration are color-coded, with the color-coding giving the correspondence with the legend (for b/w printout we note that the legend and the curves are in the same order from top to bottom) For each twin set configuration, the continuous line corresponds to the prediction of the model, with the color band describing for each setup the 95% confidence interval provided by the model. The starred points are the observations (ratios of undetected omissions) from the experiments. Each of the 20 points in the figure has been obtained through 1000 runs of the different configurations above, with an overall execution time of more than 20 CPU-days per worker. As visible from the figure, the experiments fully confirm the accuracy of the model (all stars are aligned with the lines resulting from the theoretical analysis), and the ability to offer high integrity guarantees with a small additional cost.

9 Related work

Existing integrity verification techniques are usually classified in *deterministic* (e.g., [10], [11], [12], [13], [14], [15], [16], [17]), and *probabilistic*, (e.g., [3], [8], [9], [11], [18]). Deterministic techniques provide integrity guarantees with full confidence and are based on authenticated data structures (e.g., Merkle hash tree) stored at the provider's site, but can be used to verify the integrity of computations performed only on data used to construct the authenticated structures.

Probabilistic techniques are more flexible than deterministic techniques and are based on the use of *integrity checks* added to the original data. Several existing probabilistic techniques are based on replication (e.g., [8], [9], [19]), on the injection of fake data into the original data collection (e.g., [4], [5], [6], [20]), or on watermarks (e.g., [7]). Some works have analyzed the trade-off between the integrity guarantees offered by probabilistic techniques and their overhead, by proposing a game-theoretic framework and assuming that the same task can be assigned to more workers (e.g., [21], [22]). The goal of these proposals is to design a solution that provides an incentive for the workers to compute the correct result. Other proposals (e.g., [18], [19]) focus on MapReduce computations and propose the adoption of a voting mechanism on the result of replicated tasks. These works analyze how much replication is necessary to provide a given probability of detecting integrity violations. In the database context, some works (e.g., [8]) consider the combination of different probabilistic techniques to verify the integrity of join results. They, however, focus more on the efficient design of such integrity techniques but do not provide an analysis of how to effectively tune their use to achieve high-level guarantees with limited cost. Integrity verification techniques have been also developed in the data mining context (e.g., [20]) to support different types of computations. Our work is complementary to the efforts above and provides a novel perspective of investigation.

Our work presents similarities with, and nicely complements, solutions that permit to verify whether a worker correctly computed a function over an input (i.e., the correctness of a single job) at limited cost (e.g., [23]).

Another line of work aims at minimizing the estimation error when the same job is assigned (replicated) to multiple users (e.g., [24], [25]), or at determining whether a user correctly evaluates jobs to optimize job assignment (e.g., [26], [27]). These proposals are complementary to ours since they aim to determine a strategy for maximizing the probability that the results computed by users are correct.

Also, they do not consider the possibility of workers to behave opportunistically in returning their results to the aim of being undetected in their omissions.

10 Conclusions

Probabilistic integrity techniques allow for assessing integrity of distributed computations performed by possibly untrustworthy workers. If not carefully used, however, such techniques can suffer from limited effectiveness. In this paper, we have focused on two probabilistic techniques, namely sentinels and twins, and provided a model capturing their characteristics and enabling their controlled generation and injection so to provide best effective in achieving integrity guarantees. Our model can then represent a reference for effective integrity assessment in different application scenarios. Our findings can also enable clients to best frame their distributed processing problem so to maximize effectiveness of the control. For instance, with twins being twice as effective as sentinels, an image recognition problem with a *yes/no* answer where *no* is the most common answer, and hence sentinels should be applied, can be transformed into a finer-grained problem requesting workers to return, instead of the simple *no*, the result of a numerical computation on the input (hence distributing the probability of the *no*), thus enjoying a low P_{max} and enabling the use of twins.

Acknowledgment

This work was supported in part by the Office of Naval Research under grant N00014-20-1-2407, by the Army Research Office under grant W911NF-13-1-0421, by the National Science Foundation under grant CNS-1822094, by the EC under grants 101017171 (MARSAL) and 101070141 (GLACIATION), and by the Italian MIUR under PNRR project SERICS and PRIN project HOPE.

References

[1] J. Bjorck, B. Rappazzo, D. Chen, R. Bernstein, P. Wrege, and C. Gomes, "Automatic detection and compression for passive acoustic monitoring of the african forest elephant," in *Proc. of AAAI*, Honolulu, HI, USA, Jan. – Feb. 2019, pp. 476–484.

[2] M. Hamilton *et al.*, "Flexible and scalable deep learning with MMLSpark," in *Proc. of PAPIs*, Boston, MA, USA, October 2017, pp. 11–22.

[3] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, G. Livraga, S. Paraboschi, and P. Samarati, "Integrity for distributed queries," in *Proc. of CNS*, San Francisco, CA, USA, Oct. 2014, pp. 364–372.

[4] P. Ghazizadeh, R. Mukkamala, and S. Olariu, "Data integrity evaluation in cloud database-as-a-service," in *Proc. of IEEE SERVICES*, Santa Clara, CA, June 2013, pp. 280–285.

[5] C. Wang, S. Chow, Q. Wang, K. Ren, and W. Lou, "Privacy-preserving public auditing for secure cloud storage," *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 362–375, Feb. 2013.

[6] M. Xie, H. Wang, J. Yin, and X. Meng, "Integrity auditing of outsourced data," in *Proc. of VLDB*, Vienna, Austria, Sep. 2007, pp. 782–793.

[7] P. Derbeko, S. Dolev, E. Gudes, and S. Sharma, "Security and privacy aspects in MapReduce on clouds: A survey," *Computer Science Review*, vol. 20, pp. 1–28, May 2016.

[8] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati, "Efficient integrity checks for join queries in the cloud," *JCS*, vol. 24, no. 3, pp. 347–378, 2016.

[9] H. Wand, J. Yin, C.-S. Perng, and P. Yu, "Dual encryption for query integrity assurance," in *Proc of ACM CIKM*, Napa Valley, CA, USA, October 2008, pp. 863–872.

[10] A. Arasu *et al.*, "Concerto: A high concurrency key-value store with integrity," in *Proc. of ACM SIGMOD*, Chicago, IL, USA, May 2017, pp. 251–266.

[11] B. Dong, R. Liu, and H. W. Wang, "Trust-but-verify: Verifying result correctness of outsourced frequent itemset mining in data-mining-as-a-service paradigm," *IEEE TSC*, vol. 9, no. 1, pp. 18–32, 2016.

[12] E. Esiner and A. Datta, "On query result integrity over encrypted data," *Information Processing Letters*, vol. 122, pp. 34–39, June 2017.

[13] H. Hacigümüş, B. Iyer, and S. Mehrotra, "Ensuring the integrity of encrypted databases in the database-as-a-service model," in *Proc. of DBSec*, Estes Park, CO, USA, Aug. 2003, pp. 61–74.

[14] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Proc. of ACM SIGMOD*, Chicago, IL, USA, June 2006, pp. 121–132.

[15] Y. Wang, Y. Shen, H. Wang, J. Cao, and X. Jiang, "MtMR: Ensuring MapReduce computation integrity with Merkle tree-based verifications," *IEEE TBD*, vol. 4, no. 3, pp. 418–431, Sep. 2018.

[16] M. Xie, H. Wang, J. Yin, and X. Meng, "Providing freshness guarantees for outsourced databases," in *Proc. of EDBT*, Nantes, France, March 2008, pp. 323–332.

[17] B. Zhang, B. Dong, and H. Wang, "CorrectMR: Authentication of distributed SQL execution on MapReduce," *IEEE TKDE*, vol. 33, no. 3, pp. 897–908, 2021.

[18] W. Wei, J. Du, T. Yu, and X. Gu, "SecureMR: A service integrity assurance framework for MapReduce," in *Proc. of ACSAC*, Honolulu, HI, USA, Dec. 2009, pp. 73–82.

[19] H. Ulusoy, M. Kantarcioglu, and E. Pattuk, "TrustMR: Computation integrity assurance system for MapReduce," in *Proc. of BigData*, Santa Clara, CA, USA, Oct.-Nov. 2015, pp. 441–450.

[20] W. Wong, D. Cheung, B. Kao, E. Hung, and N. Mamoulis, "An audit environment for outsourcing of frequent itemset mining," *PVLDB*, vol. 2, pp. 1162–1172, 2009.

[21] A. Anta, C. Georgiou, M. Mosteiro, and D. Pareja, "Algorithmic mechanisms for reliable crowdsourcing computation under collusion," *PLoS ONE*, vol. 10, no. 3, pp. 1–22, March 2015.

[22] J. Vaidya, I. Yakut, and A.Basu, "Efficient integrity verification for outsourced collaborative filtering," in *Proc. of ICDM*, Shenzhen, China, Dec. 2014, pp. 560–569.

[23] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. of the IEEE S&P*, Berkeley, CA, USA, May 2013, pp. 238–252.

[24] A. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom, "CrowdScreen: Algorithms for filtering data with humans," in *Proc. of ACM SIGMOD*, Scottsdale, AZ, USA, May 2012, pp. 361–372.

[25] L. Tran-Thanh, M. Venanzi, A. Rogers, and N. Jennings, "Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks," in *Proc. of AAMAS*, St. Paul, MN, USA, May 2013, pp. 901–908.

[26] J. Wang, P. G. Ipeirotis, and F. Provost, "Cost-effective quality assurance in crowd labeling," *Information Systems Research*, vol. 28, no. 1, pp. 137–158, 2017.

[27] P. Welinder and P. Perona, "Online crowdsourcing: Rating annotators and obtaining cost-effective labels," in *Proc. of ACVHL*, San Francisco, CA, USA, June 2010, pp. 25–32.

Sabrina De Capitani di Vimercati — Professor at the CS Dept., Università degli Studi di Milano, Italy. http://www.di.unimi.it/decapita

Sara Foresti — Professor at the CS Dept., Università degli Studi di Milano, Italy. http://www.di.unimi.it/foresti

Sushil Jajodia — Professor at Volgenau School of Engineering, George Mason University, Fairfax, VA, USA. http://csis.gmu.edu/jajodia

Stefano Paraboschi — Professor at DIGIP Dept., Università degli Studi di Bergamo, Italy. https://cs.unibg.it/parabosc

Pierangela Samarati — Professor at the CS Dept., Università degli Studi di Milano, Italy. http://www.di.unimi.it/samarati

Roberto Sassi — Professor at the CS Dept., Università degli Studi di Milano, Italy. http://www.di.unimi.it/sassi