Integrity Verification of Cloud-hosted Data Analytics Computations

Hui (Wendy) Wang
Department of Computer Science
Stevens Institute of Technology
Hoboken, NJ, 07030
Hui.Wang@stevens.edu

Abstract

In this position paper, we present efficient and practical integrity verification techniques that check whether the untrusted cloud has returned correct result of outsourced data analytics computations. We consider the computation of summation form that is used in a large class of machine learning and data mining problems. We discuss our verification techniques for both non-collusive and collusive malicious workers in MapReduce.

Categories and Subject Descriptors

H.2.0 [**Database management**]: General—*Security, integrity, and protection*

General Terms

Security

Keywords

Cloud computing, cloud analytics as a service, machine learning, integrity verification

1. INTRODUCTION

With the advent of cloud computing, the opportunity to offer data analytics as an outsourced service is gaining increasing attention. Such cloud-based data-analytics-as-a-service ($DAaS$) paradigm has been witnessed by Google's Prediction APIs [1] and Microsoft's Daytona project [2]. For example, Google's Prediction APIs provide cloud-based machine learning tools for customer sentiment analysis, spam detection, and suspicious activity identification. The $DAaS$ paradigm provides cloud-based storage, hardware, and data analytics algorithms for clients who own large datasets but limited resources and expertise for data analysis.

There have been much work on adapting standard data analytics tasks to the cloud computing paradigm by using MapReduce framework [7]. The basic idea is to split data to different Mappers, and then collects the processed intermediate data from the Mappers. After the intermediate data is collected, the Reducers are invoked to process the intermediate data and return final results. Though simple, the MapReduce-based cloud computing framework suffers from integrity vulnerability. Due to the fact that the computations are distributed to multiple computing nodes, which may not be trusted in an open environment, merely one single incorrect intermediate result may lead to wrong final result. It is infeasible to always deploy Mappers and Reducers on honest nodes due to large numbers of them. Since sometimes the outsourced data analytics computations are so critical that it is imperative to rule out errors during the computation, it is vital that the client should be able to check the correctness of the data analytics result provided by the cloud without much computational effort.

Several existing techniques such as interactive proofs [10], probabilistically checkable proofs [4, 5], and non-interactive verifiable computing [8, 9, 12] have been proposed to address integrity issues for outsourced computations in general. The basic idea of these schemes is that the service provider returns some auxiliary information as a *proof* that the result is correct; the client should be able to quickly verify the result correctness by using the proof. In some of the protocols [3, 6, 8, 9], a client who delegates computation of a function is required to first run an expensive pre-processing phase to generate cryptographic keys needed for verification. This large initial cost is then amortized over multiple executions of the function with different inputs. This makes sense only if the client runs the same computation on many different inputs. However, it may not be applicable to the $DAaS$ paradigm as many data analytics tasks are one-time operations that involve only a single (and large) dataset.

In this paper, we present practical and efficient verification techniques that enable computationally weak users to verify whether the cloud returns correct result of outsourced data analytics computations, without having to make large initial commitment of resources for verification. We consider the computation of summation form that is used in a large class of machine learning and data mining problems (e.g., naive Bayes and neural networks). The complexity of verification is considerably less than required to actually perform the computation from scratch. The key idea of our verification techniques is to insert a set of *artificial* data values into the original data. The correctness of the computation result by the cloud will be validated by checking against the result on artificial data, which is a part of the returned answer. The correctness of the cloud's answer is determined in a non-deterministic manner (with certain probability).

Our paper is organized as following. Section 2 presents

Cloud-I'12 August 31 2012, Istanbul, Turkey

the preliminaries of the paper. Section 3 explains our verification approach. Section 4 discusses related work. Section 5 summarizes the paper.

2. PRELIMINARIES

2.1 Summation Form

The Statistical Query Model [13] allows the learning algorithm to access the learning problem only through a statistical query oracle. The statistical query oracle $STAT(f, D)$ takes as input a statistical query (SQ) of the form (X, τ) where X is a {+1\-1}-valued function on labeled examples and $\tau \in [0, 1]$ is the tolerance parameter. The statistical query oracle returns an estimate of the expectation of $f(X, D)$. The calculations of the expectation of $f(X, D)$ are expressible as a sum over data points. This form of the algorithm is called the *summation form*. An example of summation forms is the problem of finding ordinary least squares of linear regression. In this problem, each training data point is a n-dimensional vector $x_s = (x_{i1}, \ldots, x_{in})$, associated with a real valued target label y_s. Given m training data points, which defines a $m \times n$ dimensional matrix X of training instances and an $m \times 1$ matrix $Y = [y_1, \ldots, y_m]^m$ of target labels, the linear regression problem is to find the parameter vector θ^* such that $Y = \theta^T X$. The solution is to obtain $\theta^* = (X^T X)^{-1} X^T Y$. This computation can be reformulated to compute $A = X^T X$ and $B = X^T Y$, i.e., $A = \Sigma_{i=1}^m (x_i x_i^T)$ and $B = \Sigma_{i=1}^m (x_i y_i)$.

A large class of machine learning algorithms can be expressed in the summation form [7]. Examples of these algorithms include locally weighted linear regression, naive Bayes, neural network, and principle component analysis.

2.2 MapReduce Basics

A MapReduce program typically consists of a pair of user-defined *Map* and *Reduce* functions. A big number of Map tasks run in parallel and their results are combined back by the Reduce task. In the Map phase, the M Mappers run in parallel over different logical portions of an input file, called *splits*. A Mapper maps input key-value pairs (k_1, v_1) from its split to intermediate key-value pairs (k_2, v_2). When the Mapper ends, all (k_2, v_2) pairs are partitioned, and sorted w.r.t. k_2. Then each Reducer copies all (k_2, v_2) it is responsible for and sorts all received (k_2, v_2) by k_2 so all occurrences of key k_2 are grouped together. After data collection and sorting, a Reducer iterates over all its (k_2, v_2). For each distinct key k_2, the Reducer produces a final key-value pair (k_3, v_3) for every intermediate key k_2.

2.3 Summation Form in MapReduce

[7] shows that a large class of machine learning techniques that fit the summation form can be emigrated to MapReduce easily. To adapt the computation to MapReduce, one possibility is to split the input matrix X and Y into smaller pieces, and divide the computation of $A = X^T X$ and $B = X^T Y$ based on the split data. To be specific, given an $m \times n$ dimensional matrix X and an $m \times 1$ matrix Y, without losing generality, assume each Mapper is assigned a split of X, which is a $k \times p$ matrix $X_s \subseteq X$ ($k \leq m, p \leq n$), and/or a split of Y, which is a $k \times 1$-dimensional matrix $Y_s \subseteq Y$. The Mappers compute the partial values $A_s = X_s^T X_s$ and/or $B_s = X_s^T Y_s$, while the Reducers sum up the partial values A_s and B_s. The algorithm finally computes the solution $\theta = A^{-1} B$. We call the Mappers and Reducers involved in the computation the *workers*.

2.4 Attack Model

We consider the attackers as the malicious Mappers and Reducers that try to generate incorrect result in order to sabotage the output. Besides returning wrong answers, we assume that the malicious Mappers and Reducers may be aware of the verification procedure and try to escape from verification. We call these malicious Mappers and Reducers the malicious workers. We categorize the malicious workers into two types: *non-collusive workers* and *collusive workers*. The non-collusive workers return incorrect result independently, without consulting other malicious workers. The collusive workers communicate with each other before cheating. When a collusive worker is assigned a task, it consults other collusive partners that are assigned the same task, and return consistently incorrect result. It is more challenging to catch collusive workers than the non-collusive ones as they try their best to minimize the inconsistency of their returns.

2.5 Verification Goal

Our goal is to verify the correctness of $A_s = X_s^T X_s$ and $B_s = X_s^T Y_s$ computed by Mappers. As both A_s and B_s are matrices, we verify the correctness of A_s and B_s by checking the number of correct elements in the returned matrix. Without losing generality, given a $k \times p$ matrix M, let M^w be the matrix result returned by the worker. We define the *overlapping of M and M^w*, denoted as $M \cap M^w$, as the entries in M and M^w that match. In other words, $M \cap M^w = \{M[i, j] | M[i, j] = M^w[i, j]\}$. We define the *precision r* of a $k \times p$ matrix M^w as $r = \frac{c}{kp}$, where $c = |M \cap M^w|$. Our aim is achieve (α, β)-correctness. Formally, given the matrix M^w returned by the MapReduce workers, let p be the probability to catch M^w of precision $r \leq \beta$, where $\beta \in [0, 1]$ is a user-defined threshold. We say a verification method provides (α, β)-*correctness* verification guarantee if $p \geq \alpha$, where $\alpha \in [0, 1]$ is a user-specified threshold.

3. VERIFICATION METHOD

3.1 Architecture

The MapReduce core consists of one master JobTracker task and many TaskTracker tasks. Typical configurations run the JobTracker task on the same machine, called the *master*, and run TaskTracker tasks on other machines, called *slaves*. The master is responsible for controlling the computation, such as job management, task scheduling, and load balance. Slaves are hosts that contribute computation resources to execute tasks assigned by the master. We assume the master node is trusted. Therefore, the master node will be responsible for verification.

3.2 Catch Non-collusive Malicious Workers

3.2.1 When honest workers take majority

When the honest workers take the majority, we use the *replication-based verification approach*. By this approach, the master node assigns the same task to multiple workers. Honest workers should return the same result of the task. Under the assumption that workers are non-collusive, the workers whose results are inconsistent with the majority of the workers that are assigned the same task are caught as malicious. If there is no winning answer by majority voting, the master node assigns more copies of the task to additional

workers, until there is a winning answer from the majority. The workers that return correct answers are assigned the correctness probability 1, while the workers that return incorrect answers are assigned the correctness probability 0.

3.2.2 When malicious workers take majority

When malicious workers take the majority, they cannot be caught by the replication-based approach. Therefore, we propose the *artificial data injection (ADI)* mechanism to catch non-collusive malicious workers. In particular, before assigning tasks, the master node injects a set of artificial data values X_a into the matrix X_s, and computes the required computations on X_a. The master node maintains the result on X_a as the *proof*. As the artificial data X_a is indistinguishable from the real data X_s, the workers compute A_s and B_s on X_s that contains both real and artificial values, and return the result back to the master node. If the workers' output does not contain the proof, the master node determines that the workers' output is incorrect with 100% certainty. Otherwise, the master nodes returns a probabilistic guarantee of catching the workers' incorrect result. Next, we explain the details of our verification mechanism.

Given the $k \times p$ matrix X_s and the $k \times 1$ matrix $\vec{Y_s}$, our goal is to verify the correctness of $A_s = X_s^T X_s$, and $B_s = X_s^T Y_s$. For verification purpose, the master node inserts an *artificial* $k \times \ell$ matrix X_a into X_s. After insertion X_s becomes a $k \times (p + \ell)$ matrix. The entries in X_a are picked randomly from the same domain of X_s. We require that X_a and X_s have the same distribution, so that the malicious workers cannot distinguish them easily. Therefore, the workers have equal probability to cheat on computation of X_a and X_s.

$$X_a = \begin{pmatrix} x_{1,p+1} & \dots & x_{1,p+\ell} \\ \dots & \vdots & \dots \\ x_{k,p+1} & \dots & x_{k,p+\ell} \end{pmatrix},$$

$$X'_s = X_s \circ X_a = \begin{pmatrix} x_{1,1} & \dots & x_{1,p} & x_{1,p+1} & \dots & x_{1,p+\ell} \\ \dots & \vdots & \vdots & \vdots & \vdots & \dots \\ x_{k,1} & \dots & x_{k,p} & x_{k,p+1} & \dots & x_{k,p+\ell} \end{pmatrix}.$$

Verification of $A_s = X_s^T X_s$. The master node precomputes $A_a = X_a^T X_a$. After the worker returns $A_s = X_s'^T X'_s$, the master node checks whether

$$\forall i, j \in [p+1, p+\ell], A_s[i,j] = A_a[i-p, j-p].$$

If there exists any mismatch, the master node concludes with 100% certainty that the worker returns incorrect answer. Otherwise, the master node determines the correctness of workers' answer in a non-deterministic manner (with certain probability). In particular, the probability that any entry in A_s is true is β, where β is the precision threshold given in (α, β)-correctness requirement. Therefore, the probability p of catching a dishonest worker with ℓ^2 entries in A_a is $p = 1 - \beta^{\ell^2}$. To satisfy (α, β)-correctness (i.e., $p \geq \alpha$), it must be true that $\ell \geq \sqrt{\lceil log_\beta(1-\alpha) \rceil}$.

Verification of $B_s = X_s^T Y_s$. The master node precomputes $B_a = X_a^T Y_s$. After the worker returns $B_s = X_s'^T Y_s$, the master node checks whether

$$\forall i \in [p+1, p+\ell], B_s[i] = B_a[i-p].$$

Any mismatch between B_s and B_a helps the master node to conclude that the worker returns incorrect answer with 100% certainty. Otherwise, the probability p of catching a dishonest worker with ℓ entries in B_a is $p = 1 - \beta^\ell$.

To satisfy (α, β)-correctness (i.e. $p \geq \alpha$), it must be true that $\ell \geq \lceil log_\beta(1-\alpha) \rceil$. To be consistent with the requirement that $\ell \geq \sqrt{\lceil log_\beta(1-\alpha) \rceil}$ for the verification of $A_s = X_s^T X_s$, we require that $\ell \geq \lceil log_\beta(1-\alpha) \rceil$. We tried a few settings of α and β, and observed that it does not need large number of artificial data values to catch workers that change a small fraction of result with high correctness probability. For instance, when $\beta = 0.99$ (i.e., at least 1% of the computed matrix is incorrect) and $\alpha = 0.99$, we only need to add $\ell = 459$ artificial columns to the real matrix X_s. A nice property is that ℓ is independent of the size of the matrix X_a, i.e., our *ADI* mechanism is especially useful for verification of computation of large matrics.

Complexity. The complexity of preparation of X_a is $k\ell$. The complexity of computing $A_a = X_a^T X_a$ and $B_a = X_a^T Y_s$ is $O(k\ell^2)$ and $O(k\ell)$ respectively. Compared with the complexity $O(kp^2)$ of computing A_s and the complexity $O(kp)$ of computing B_s, the complexity of computing A_a and B_a is much smaller, since normally ℓ is much smaller than p. The complexity of correctness verification is $O(\ell^2)$.

3.3 Catch Collusive Malicious Workers

Attacks against verification. Apparently, the majority voting mechanism will not work if the collusive workers take the majority of the Mappers and Reducers. A seemly straightforward solution to catch collusive malicious workers is to modify the *ADI* mechanism slightly by always inserting a *unique* artificial data matrix X_a into X_s, even for the same matrix X_s assigned to different workers. This approach will only catch the malicious workers who do not have any knowledge of the verification procedure. For those malicious workers who have sophisticated knowledge of the *ADI* verification procedure, they may be able to distinguish X_a from X_s by comparing their matrices. Specifically, for those malicious workers who were assigned the same original matrix X_s, they may be aware that their matrices have a large portion of common values. As the workers are also aware of the details of the verification mechanism, they can conclude that the common values must be real values and thus distinguish between the real and artificial values.

Verification method. To catch the collusive malicious workers, we propose the *instance-hiding* verification technique. In particular, before assigning the matrix X_s to the workers, the master node applies transformation on X_s by computing $X'_s = T_s \times X_s$, where T_s is a $k \times k$ transformation matrix. The master node always uses a unique transformation matrix T_s for each input matrix X_s. This guarantees that each worker is assigned a unique matrix X'_s so that workers cannot distinguish real and artificial data by matrix comparison. Then the master node injects the artificial data X_a into the transformed matrix X'_s and apply the *ADI* verification procedure (Sec 3.2.2).

3.4 Post-processing

Inserting artificial data values into the matrix introduces noise to the final output. However, such noise can be easily eliminated. This is advantageous as our verification approach will not influence the computational result.

Non-collusive malicious workers. For the *ADI* verification approach, the master node returns

$$A'_s = \{A_s[i,j] | i, j \in [1, p]\}$$

as the real answer of $A_s = X_s^T X_s$, and

$$B'_s = \{B_s[i] | i \in [1, p]\}$$

as the real answer of $B_s = X_s^T Y$.

Collusive malicious workers. By the instance-hiding approach, the master node computes $A'_s = \{A_s[i, j] | i, j \in [1, p]\}$ and $B'_s = \{B_s[i] | i \in [1, p]\}$. After that, the master node computes $A''_s = (T_S^T T_S)^{-1} A'_s$, and $B'_s = (T_S)^{-1} B'_s$. The computation of A''_s and B''_s can be distributed to multiple Mappers and Reducers. The instance-hiding verification method can be applied again to check the correctness of the computation. The result of A''_s and B''_s will be returned as the final answer of A_s and B_s.

4. RELATED WORK

The problem of verifiable computation was tackled in many previous works in the theoretical computer science by using interactive proofs [10] and probabilistically checkable proofs (PCPs) [4, 5]. This body of theory is impractical [20], due to the complexity of the algorithms and difficulty to use general-purpose cryptographic techniques in practical data analytics problems.

In the last decade, intensive efforts have been put on the security issues of the database-as-a-service (DaS) paradigm [11, 14, 16, 17, 18, 19, 21, 24]. Only until recently some attention was paid to the security issues of the data-analytics-as-a-service ($DAaS$) paradigm [15, 22]. However, most of these work only focus on how to encrypt the data to protect data confidentiality and pattern privacy. There is surprisingly very little research [23] on correctness verification of outsourced data mining computations. [23] proposed verification methods for *frequent itemset mining* [23]. The basic idea is to insert some *fake items* that do not exist in the original dataset; these fake items will deliver a set of fake (in)frequent itemsets. Then by checking whether a fake frequent itemset is missing in the returned result, the client can verify whether the correctness of the mining answer by the server. We extend the view to the cloud-based data-analytics-as-a-service paradigm, with the focus on a large class of machine learning problems that fit the statistical query model.

5. DISCUSSION AND CONCLUSION

In this position paper, we present our preliminary study of correctness verification techniques for cloud-based data analytics problems that can fit the summation form. There are many open questions, each requiring further study. For example, what will be the cost that our verification techniques will bring to computations in real-world cloud, e.g., Amazon EC2? Can we define a budget-driven model to allow the client to specify her verification needs in terms of budget (possibly in monetary format) besides α and β? How can we identify the collusive and non-collusive workers, as well as whether collusive workers take the majority, in a cloud in practice? Can we achieve a deterministic verification guarantee by adapting the existing cryptographic techniques (e.g., [6]) to the data-analytics-as-as-service paradigm which allows a one-time computationally expensive initialization phase? It also will be interesting to compare the deterministic approach and our probabilistic approach in terms of verification budget and computational overhead.

6. REFERENCES

[1] http://research.microsoft.com/en-us/projects/azure/daytona.aspx.

[2] http://code.google.com/apis/predict/.

[3] Applebaum, B., Ishai, Y., and Kushilevitz, E. From secrecy to soundness: efficient verification via secure computation. In *Proceedings of the 37th international colloquium conference on Automata, languages and programming* (2010).

[4] Arora, S., Lund, C., Motwani, R., Sudan, M., and Szegedy, M. Proof verification and the hardness of approximation problems. *Journal of ACM 45* (May 1998).

[5] Babai, L., Fortnow, L., Levin, L. A., and Szegedy, M. Checking computations in polylogarithmic time. In *STOC* (1991).

[6] Benabbas, S., Gennaro, R., and Vahlis, Y. Verifiable delegation of computation over large datasets. In *CRYTO* (2011).

[7] Chu, C.-t., Kim, S. K., Lin, Y.-a., and Ng, A. Y. Map-reduce for machine learning on multicore. *Architecture 19*, 23 (2007), 281.

[8] Chung, K.-M., Kalai, Y., and Vadhan, S. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO* (2010).

[9] Gennaro, R., Gentry, C., and Parno, B. Non-interactive verifiable computing: outsourcing computation to untrusted workers. In *CRYPTO* (2010).

[10] Goldwasser, S., Micali, S., and Rackoff, C. The knowledge complexity of interactive proof systems. *SIAM Journal of Computing 18* (February 1989), 186–208.

[11] Hacigümüş, H., Iyer, B., Li, C., and Mehrotra, S. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD* (2002).

[12] Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., and Sahai, A. Efficient non-interactive secure computation. In *EUROCRYPT* (2011).

[13] Kearns, M. Efficient noise-tolerant learning from statistical queries. In *Journal of the ACM* (1993), pp. 392–401.

[14] Li, F., Hadjieleftheriou, M., Kollios, G., and Reyzin, L. Dynamic authenticated index structures for outsourced databases. In *SIGMOD* (2006).

[15] Molloy, I., Li, N., and Li, T. On the (in)security and (im)practicality of outsourcing precise association rule mining. In *ICDM* (2009).

[16] Narasimha, M., and Tsudik, G. Dsac: Integrity of outsourced databases with signature aggregation and chaining. In *CIKM* (2005).

[17] Pang, H., Jain, A., Ramamritham, K., and Tan, K.-L. Verifying completeness of relational query results in data publishing. In *SIGMOD* (2005).

[18] Pang, H., Zhang, J., and Mouratidis, K. Scalable verification for outsourced dynamic databases. *Proceedings of VLDB Endowment 2* (August 2009), 802–813.

[19] Papadopoulos, S., Papadias, D., Cheng, W., and Tan, K.-L. Separating authentication from query execution in outsourced databases. In *ICDE* (2009).

[20] Setty, S., Blumberg, A. J., and Walfish, M. Toward practical and unconditional verification of remote computations. In *Proceedings of the 13th Workshop on Hot Topics in Operating Systems (HotOS)* (2011).

[21] Sion, R. Query execution assurance for outsourced databases. In *VLDB* (2005).

[22] Tai, C.-H., Yu, P. S., and Chen, M.-S. k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining. In *SIGKDD* (2010).

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

[24] Xie, M., Wang, H., Yin, J., and Meng, X. Integrity auditing of outsourced data. In *VLDB* (2007).