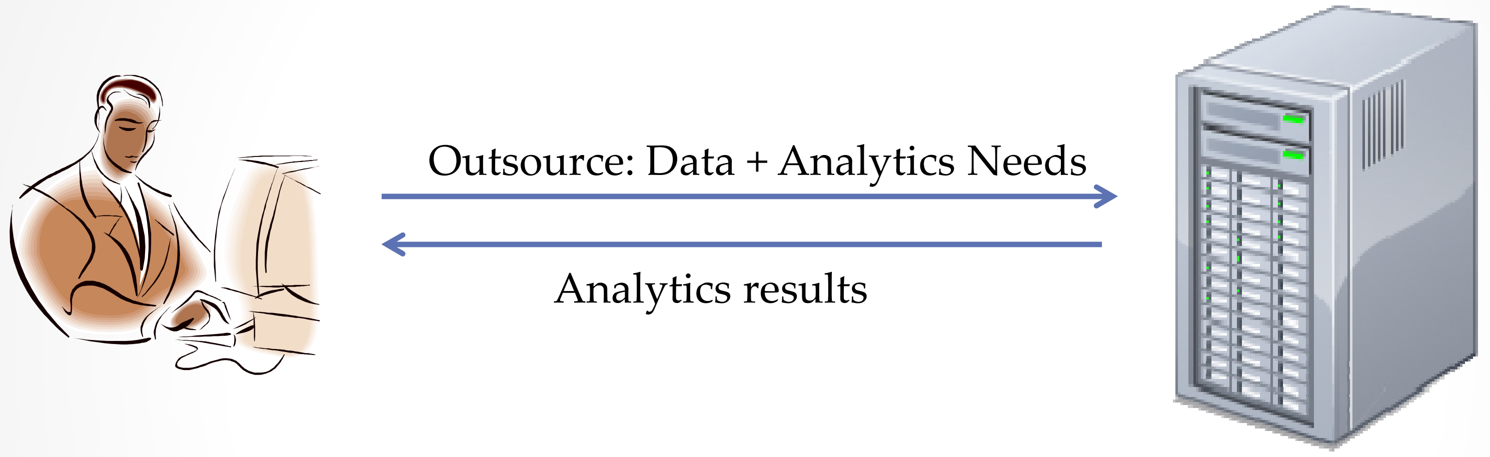
Integrity Verification of Cloud-hosted Data Analytics Computations (Position paper)

Hui (Wendy) Wang
Stevens Institute of Technology
New Jersey, USA

Data-Analytics-as-a-Service (DAaS)



Data Owner (Client)

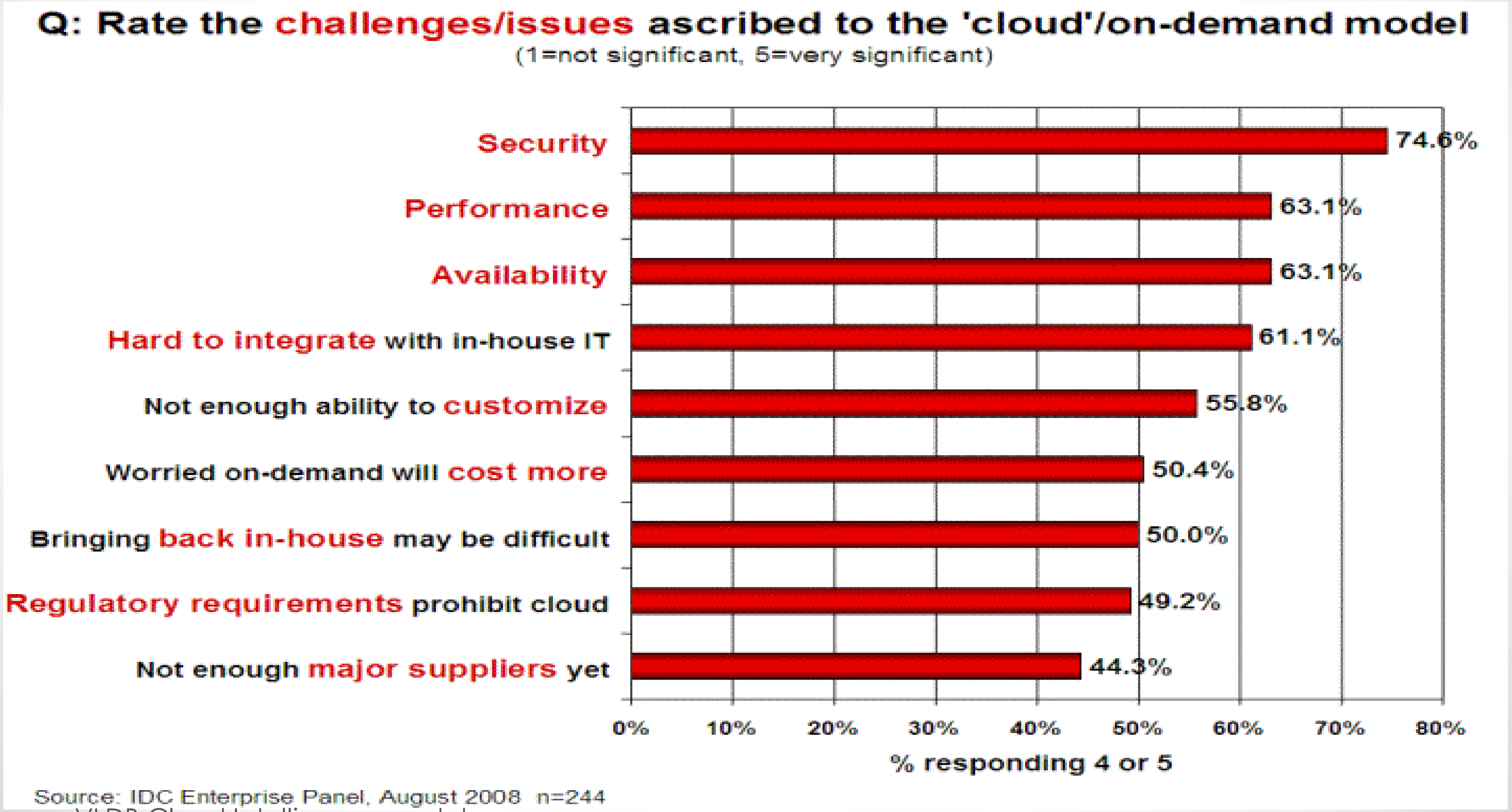
- Owns large volume of data
- Computationally weak

Cloud

- Computationally powerful
- Provide data analytics as a service

Examples of DAaS
Google Prediction APIs, Amazon EC2

Fear of Cloud: Security



Source: IDC Enterprise Panel, August 2008 n=244

Integrity Verification of the result in DAaS Paradigm

- The server cannot be fully trusted
- We focus on ***result integrity***
  - The client should be able to verify that the analytics result returned by the cloud is correct.
- Challenges:
  - Analytics result is unknown before mining.
  - The client is computationally weak to perform sophisticated analysis.

Related Work

- Integrity Assurance for Database-as-a-Service (DAS) Paradigm
  - Merkle hash trees [2,3], signatures on a chain of paired tuples [4], challenge tokens [5], and counterfeit records [9]
- Protect sensitive data and data mining results in the data-mining-as-a-service (DMAS)
  - Association rule mining [1, 6-7]
- Integrity Verification in DMAS Paradigm
  - Frequent itemset mining [8]

Outline

Outline

Summarization Form

- Training data: an $m \times n$ dimensional matrix **X**
- Target lables: an $m \times 1$ matrix **Y**
- Problem:
  - Find the parameter vector θ such that $Y = \theta^T X$.
- The solution: obtain $\theta^* = (X^T X)^{-1} X^T Y$.
- This computation can be reformulated to compute (1) $A = X^T X$, and
  (2) $B = X^T Y$.
- In other words, compute

$$A = \Sigma_{i=1}^{m}(x_i x_i^T) \text{ and } B = \Sigma_{i=1}^{m}(x_i y_i)$$

Importance of Summarization Form

- A large class of machine learning algorithms can be expressed in the summation form.
- Examples of summarization form based algorithms:
  - Locally weighted linear regression,
  - Naïve Bayes,
  - Neural network,
  - Principle component analysis,
  - …

Summarization Form in MapReduce

- Mappers:
  - Each Mapper is assigned a split $X_s \subseteq X$ and/or a split $Y_s \subseteq Y$
  - The Mappers compute the partial values

$$A_s = X_s^T X_s \qquad B_s = X_s^T Y_s$$

- Reducers:
  - The Reducers sum up the partial values As and Bs.

Attack Model

- Non-collusive workers
  - Return incorrect result independently, without consulting other malicious workers.
- Collusive workers
  - Communicate with each other before cheating.

Verification Goal

- Verify the correctness of $A_s = X_s^T X_s$ and $B_s = X_s^T Y_s$
- Formally:
  - M: the correct matrix
  - M^W: the matrix result returned by the Mappers
  - Precision of M^W: $r = \frac{|M \cap M^w|}{|M|}$
  - P: the probability to catch M^W of precision $r \le \beta$, where $\beta \in [0, 1]$ is a user-defined threshold.
  - We say a verification method provides **(α, β)-correctness** verification guarantee if $p \ge \alpha$, where $\alpha \in [0, 1]$ is a user-specified threshold.

Outline

- Introduction
- Related Work
- Preliminaries
- Our Verification Approach
- Future Work and Conclusion

Architecture

- 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*.
- We assume the master node is trusted, and is responsible for verification.

Overview of Our Verification Approaches

- Catch non-collusive malicious workers
  - When honest workers take majority: the *replication-based* verification approach
  - When malicious workers take majority: the *artificial data injection* (ADI) approach
- Catch collusive malicious workers
  - The *instance-hiding* verification approach

Catch Non-collusive Malicious Workers

- When honest workers take majority
  - We propose the *replication-based* verification approach
    - The master node assigns the same task to multiple workers.
    - The majority of workers will return correct answer
    - 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

Catch Non-collusive Malicious Workers

- When malicious workers take majority
  - We propose the artificial data injection (ADI) verification approach
    - The master nodes inserts an artificial k × ℓ matrix Xa into Xs

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

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

ADI Approach (Cont.)

- Verification
  - The master node pre-com $A_a = X_a^T X_a$.
  - 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 result correctness with a probability $p = 1 - \beta^{\ell^2}$, where β is the precision threshold given in (α, β)-correctness requirement.

ADT Discussion

- To satisfy (α, β)-correctness, it must satisfy that

$$\ell \geq \sqrt{\lceil log_{\beta}(1-\alpha) \rceil}$$

Where ℓ is the number of columns in the artificial matrix X_a (the number of rows of X_a is the same as that of X_s).

- It only needs a small ℓ to catch workers that change a small fraction of result with high correctness probability.
- ℓ is independent of the size of the input matrix. Thus our ADI mechanism is especially useful for verification of computation of large matrics.

ADT Complexity

- The complexity of verification preparation: $O(k\ell)$.
- The complexity of verification: $O(k\ell^2)$
  - K: number of columns of the input matrix X_s
  - L: number of columns of the artificial matrix X_a

Catch Collusive Malicious Workers

- ADT approach cannot resist the cheating of collusive malicious workers
- We propose the *instance-hiding* verification approach.
- Before assigning 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×k transformation matrix.

  - Ts is unique for each input matrix Xs.
  - Then the master node injects the artificial data Xa into the transformed matrix X's and apply the ADI verification procedure.

Post-processing

- The post-processing procedure eliminates the noise due to the insertion of artificial matrix
- 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 $A_s = X_s^T X_s$
- Collusive malicious workers
  - For the instance-hiding approach, the master node computes $A'_s = \{A_s[i,j] | i,j \in [1,p]\}$
  - After that, the master node computes
    $$A''_s = (T_S^T T_S)^{-1} A'_s$$

Conclusion

- Result integrity verification of the result in cloud-based data-analytics-as-a-service paradigm is very important
- We consider summarization form, in which a large class of machine learning algorithms can be expressed.
- We propose verification approaches for both non-collusive and collusive malicious Mappers

Open Questions

- 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 a 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 to DAaS paradigm?

References

1. Giannotti, F., Lakshmanan, L.V., Monreale, A., Pedreschi, D., Wang, H.: *Privacy-preserving mining of association rules from outsourced transaction databases*. In: SPCC (2010)
2. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: *Dynamic authenticated index structures for outsourced databases*. In: SIGMOD (2006)
3. Mykletun, E., Narasimha, M., Tsudik, G.: *Authentication and integrity in outsourced databases*. Trans. Storage 2 (May 2006)
4. Pang, H., Jain, A., Ramamritham, K., Tan, K.-L.: *Verifying completeness of relational query results in data publishing*. In: SIGMOD (2005)
5. Sion, R.: *Query execution assurance for outsourced databases*. In: VLDB (2005)
6. Tai, C.-H., Yu, P.S., Chen, M.-S.: *k-support anonymity based on pseudo taxonomy for outsourcing of frequent itemset mining*. In: SIGKDD (2010)
7. Wong, W.K., Cheung, D.W., Hung, E., Kao, B., Mamoulis, N.: *Security in outsourcing of association rule mining*. In: VLDB (2007)
8. Wong, W.K., Cheung, D.W., Kao, B., Hung, E., Mamoulis, N.: *An audit environment for outsourcing of frequent itemset mining*. PVLDB 2 (2009)
9. Xie, M., Wang, H., Yin, J., Meng, X.: *Integrity auditing of outsourced data. In*: VLDB (2007)

Q & A

- Thanks !