Google App Engineにおけるプライバシー保護協調フィルタリング

Anirban Basu† Jaideep Vaidya‡ Hiroaki Kikuchi† Theo Dimitrakos§

† 東海大学
108-8619 東京都港区高輪 2-3-23 Email: abasu@cs.dm.u-tokai.ac.jp
‡Rutgers, The State University of New Jersey
1, Washington Park, Newark, New Jersey 07102-1897, USA
Email: jsvaidya@business.rutgers.edu
§Security Futures Practice, Research & Technology, BT
Adastral Park, Martlesham Heath, IP5 3RE, UK
Email: theo.dimitrakos@bt.com

あらまし 本研究では，協調フィルタリングにおける個人評価データのプライバシー保護を目的とする．本稿では，協調フィルタリングのフィルタリングの問題を重み付 Slope One 予測方式を拡張することで検討し，提案方式を Platform-as-a-Service (PaaS) cloud によって実現されている Software-as-a-Service (SaaS) の代表例である，Google App Engine for Java (GAE/J) を用いて実装する．

Practical privacy preserving collaborative filtering on the Google App Engine

Anirban Basu† Jaideep Vaidya‡ Hiroaki Kikuchi† Theo Dimitrakos§

†Graduate School of Engineering, Tokai University
2-3-23 Takanawa, Minato-ku, Tokyo 108-8619, Japan
‡MSIS Department, Rutgers, The State University of New Jersey
1, Washington Park, Newark, New Jersey 07102-1897, USA
§Security Futures Practice, Research & Technology, BT
Adastral Park, Martlesham Heath, IP5 3RE, UK

Abstract With rating-based collaborative filtering (CF) one can predict the rating that a user will give to an item, derived from the ratings of other items given by other users. However, preserving privacy of rating data from individual users is a significant challenge. Many privacy preserving schemes have, so far, been proposed, such as our earlier work on extending the well known weighted Slope One predictor. However, many such theoretically feasible schemes face practical implementation difficulties on real world public cloud computing platforms. In this paper, we re-visit the generalised problem of privacy preserving collaborative filtering and demonstrate an approach and a realistic implementation on the specialised Software-as-a-Service (SaaS) construction Platform-as-a-Service (PaaS) cloud offering – the Google App Engine for Java (GAE/J).

1 Introduction

Consider a motivating example is as follows: Alice has been to London, København, Trøndheim, Napoli, Bangalore, Hong Kong, Tokyo and Kyoto. She intends to visit Melbourne next and would like a tourism information provider running on the cloud to give her a rating prediction for Melbourne based on her ratings of the cities she has visited as well as such ratings from the community. She is completely unaware of (and does not care) who else has rated various cities in this way apart from obtaining a reasonable rating for Melbourne. Alice also is unwilling to send the entire rating vector for her items (i.e. cities) to any third party but is happy to send some in such a way that they are delinked from her identity through some anonymising mechanism. If Alice is to obtain a rating for Melbourne, she would prefer confidentiality of the information and also does not want to reveal her identity in the prediction query. In future, Alice may also change her previous ratings on any city.

Thus, we aim to build a privacy preserving collaborative filtering scheme on the cloud for any item such that: 1. a contributing user need not reveal his/her entire rating vector to any other party, 2. any individual parts of information revealed by a user are insufficient to launch an inference based attack to reveal any additional information, 3. a trusted third party is not required for either model construction or for prediction, 4. assume honest but curious user participation, although we discuss in this paper what happens if we give up this assumption, and 5. assume insider threats to data privacy from the cloud infrastructure itself.

To achieve this, in our approach, the user will use anonymising techniques (e.g. anonymiser network such as Tor, pseudonyms) to de-identify himself/herself from his/her ratings sufficiently such that the complete rating vector for a user cannot be reconstructed. Thus our security guarantees are based upon the security guarantees provided by the underlying anonymiser/mix network, and are bounded by it.

Contributions

The contributions of this paper are summarised as follows.

1. Our work is the first, to our knowledge, to attempt a novel practical implementation of a privacy preserving weighted Slope One predictor on a real world cloud computing platform.

2. Ours is a novel idea where encryption is used at the user level, allowing only the target user to decrypt the result of an encrypted prediction query, and thereby eliminating the requirement of trusted third parties, which were required in any privacy preserving scheme taking advantage of threshold decryption.

3. In our earlier work [1], we tackled the privacy preserving CF problem from a different angle. Our earlier scheme is applicable to pure horizontal and pure vertical dataset partitions. The scheme presented in this paper does not consider dataset partitioning in the cloud because user's rating data are not stored in the cloud at all. Even so, the general assumption is that each user knows only his or her own ratings, and does know all of them – similar to the case of horizontal partitioning of data outside the cloud. We do include a discussion on the case of vertical partitioning of data outside the cloud.

2 Background

2.1 Slope One collaborative filtering

The Slope One predictors due to Lemire and McLachlan [7] are item-based collaborative filtering schemes that predict the rating a user will give to an item from a pair-wise deviations of item ratings. The unweighted scheme estimates a missing rating using the average deviation of ratings between pairs of items with respect to their *cardinalities*. Slope One CF can be evaluated in two stages: pre-computation and prediction of ratings. The weighted Slope One predictor adds more weight to a pair-wise deviation if both items in the pair have been rated by many users.

In the pre-computation stage, the average deviations of ratings from item a to item b is given as:

$$\overline{\delta_{a,b}} = \frac{\Delta_{a,b}}{\phi_{a,b}} = \frac{\sum_i \delta_{i,a,b}}{\phi_{a,b}} = \frac{\sum_i (r_{i,a} - r_{i,b})}{\phi_{a,b}}. \quad (2.1)$$

where $\phi_{a,b}$ is the count of the users who have rated both items while $\delta_{i,a,b} = r_{i,a} - r_{i,b}$ is the deviation of the rating of item a from that of item b both given by user i.

In the prediction stage, the rating for user u and item x using the *weighted* Slope One is predicted as:

$$\begin{aligned} r_{u,x} &= \frac{\sum_{a|a\neq x}(\overline{\delta_{x,a}} + r_{u,a})\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}} && (2.2) \\ \implies r_{u,x} &= \frac{\sum_{a|a\neq x}(\Delta_{x,a} + r_{u,a}\phi_{x,a})}{\sum_{a|a\neq x}\phi_{x,a}}. && (2.3) \end{aligned}$$

The matrices Δ and ϕ are called deviation and cardinality matrices respectively. These matrices are sparse matrices. We need to store the upper triangulars only because the leading diagonal contains deviations and cardinalities of ratings between the same items, which is irrelevant. The lower triangular for the deviation matrix is the additive inverse of the upper triangular while the lower triangular of the cardinality matrix is the same as its upper triangular. These two matrices contain information about item pairs only, so these do not pose any privacy risk to user's rating data. Despite the plaintext deviation and cardinality storage, if the user sends his/her rating vector to the prediction function then it is a privacy threat. Therefore, we can use encrypted rating prediction.

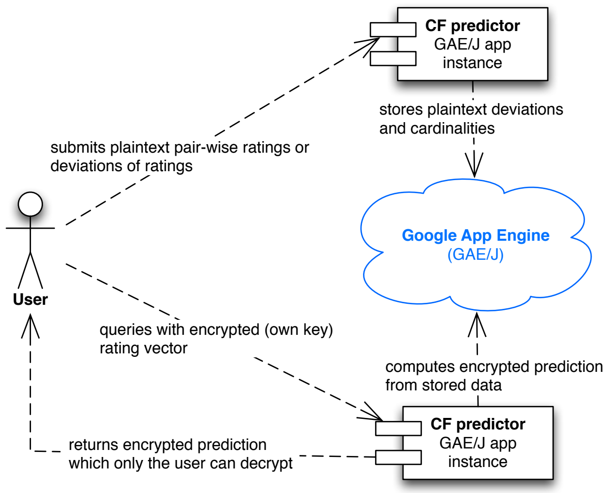
2.2 Problem statement

Definition *[Privacy-Preserving weighted Slope One Predictor] Given a set of m users $u_1, \ldots, u_m$ that* *may rate any number of n items $i_1, \ldots, i_n$, build the weighted Slope One predictor for each item satisfying the following two constraints:*

- *no submitted rating should be linked back to any user.*
- *any user should be able to obtain a prediction without leaking his/her private rating information.*

3 Proposed scheme

Akin to the original Slope One CF scheme, our proposed extension also contains a pre-computation phase and a prediction phase. Pre-computation is an on-going process as users add, update or delete pair-wise ratings or deviations of ratings. The overall user-interaction diagram of our proposed model is presented in figure 3.1 showing the addition of rating data only.



CF predictor
GAE/J app
instance
stores plaintext deviations and cardinalities
submits plaintext pair-wise ratings or deviations of ratings
Google App Engine
(GAE/J)
User
queries with encrypted (own key) rating vector
computes encrypted prediction from stored data
CF predictor
GAE/J app
instance
returns encrypted prediction which only the user can decrypt

Figure 3.1: User-interaction diagram of our scheme.

3.1 Pre-computation

In the pre-computation phase, the plaintext deviation matrix and the plaintext cardinality matrix are computed. In the absence of full rating vectors from users and consistent user identification, the combination of the deviation and the cardinality matrices pose no privacy threat to the users' private rating data. The collection of the rating data is done pair-wise and after the user identity is de-linked in the process through the use of known techniques, such as anonymising networks [8, 2, 5], mixed networks [3, 6, 4], pseudonymous group memberships [10], and so on. User submits a pair of ratings or the corresponding deviation to the cloud application at any point in time. Thus, if the user originally rated n items then $\frac{n(n-1)}{2}$ pair-wise ratings or deviations should be submitted. Since the user's identity (e.g. a pseudonym or an IP address) can (rather, must) change between consecutive submissions, the cloud cannot deterministically link the rating vector to a particular user.

3.1.1 Case of new ratings

In the pre-computation stage, the average deviations of ratings from item a to item b is given. The cloud application only maintains a list of items; their pairwise deviations and cardinalities but no other user data. The cloud only learns that the two ratings or their deviation by a particular user (provided the user identity changes in the consecutive submission), which is even insufficient to launch an offline knowledge based inference attack on the user's private rating vector. The process of rating addition is described in algorithm 3.1.

3.1.2 Updates and deletions

Updates or deletions of existing rating data are possible. For example, say the user has rated item a and b beforehand. When it comes to updating, he/she can notify the cloud of the difference between the new pair-wise rating deviation and the previous one and flag it to the cloud that it is an update. The process of rating update is described in algorithm 3.2.

Similarly, for the delete operation, the additive inverse of the previous deviation, i.e. $-\delta_{a,b}$ is sent by the user to the cloud signifying a deletion. The process of rating deletion is described in algorithm 3.3.

3.1.3 What if the user is dishonest?

If the user is dishonest, contrary to our assumption, then it is evident that automated bot-based addition, updates and deletions can disrupt the pre-computation stage. Although we leave this for future work, one possibility is to use CAPTCHA [9] to require human intervention, and hence slow down the number of additions, updates and deletions.

3.2 Prediction

In the prediction phase, the user queries the cloud with an encrypted and complete rating vector. The

Algorithm 3.1 An algorithm for the addition of new ratings.

```
Require: An item pair identified by $a$ and $b$, ratings $r_a$ and $r_b$, or the deviation $\delta_{a,b} = r_a - r_b$ has been submitted. Calculate $\delta_{a,b}$ if it was not submitted.
1: Find the deviation $\Delta_{a,b}$ and cardinality $\phi_{a,v}$ in the in-memory cache; and in the datastore if not found in cache.
Ensure: While looking for deviations and cardinalities, also look for their inverses, i.e. $\Delta_{b,a}$ and $\phi_{b,a}$ because only the upper triangular is stored. {If the inverses are retrieved then deviation must be inverted before operating on it.}
2: if $\Delta_{a,b}$ and $\phi_{a,b}$ not found then
3:     $\Delta_{a,b} \leftarrow 0$ and $\phi_{a,b} \leftarrow 0$.
4: end if
5: Update $\Delta'_{a,b} \leftarrow \Delta_{a,b} + \delta_{a,b}$ and $\phi'_{a,b} \leftarrow \phi_{a,b} + 1$.
6: Store $\Delta'_{a,b}$ and $\phi'_{a,b}$ in the cache and also in the datastore.
Ensure: While writing to cache and to datastore, write to the inverses $\Delta'_{b,a}$ and $\phi'_{b,a}$ if these were initially retrieved. {If the inverses were retrieved then deviation must be inverted before storing it.}
7: Audit this add operation in the datastore, e.g. using user's IP address as the identity. {This is a typical insider threat in the cloud.}
```

encryption is carried out at the user's end with the user's public key. The prediction query, thus, also includes the user's public key, which is then used by the cloud to encrypt the necessary elements from the deviation matrix and to apply homomorphic multiplication according to the prediction equation defined in equation 3.1, where $\mathcal{D}()$ and $\mathcal{E}()$ are decryption and encryption operations, $\Delta_{x,a}$ is the deviation of ratings between item x and item a; $\phi_{x,a}$ is their relative cardinality and $\mathcal{E}(r_{u,a})$ is an encrypted rating on item a sent by user u, although the identity of the user is irrelevant in this process. Note that the final decryption is again performed at the user's end with the user's private key, thereby eliminating the need of any trusted third party for threshold decryption.

$$r_{u,x} = \frac{\mathcal{D}(\prod_{a|a\neq x}(\mathcal{E}(\Delta_{x,a})(\mathcal{E}(r_{u,a})^{\phi_{x,a}})))}{\sum_{a|a\neq x}\phi_{x,a}}. \quad (3.1)$$

which is optimised by reducing the number of encryptions as follows:

$$r_{u,x} = \frac{\mathcal{D}(\mathcal{E}(\sum_{a|a\neq x}\Delta_{x,a})\prod_{a|a\neq x}(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))}{\sum_{a|a\neq x}\phi_{x,a}}. \quad (3.2)$$

Algorithm 3.2 An algorithm for the updates of existing ratings.

```
Require: An item pair identified by $a$ and $b$, and $diff_{\delta_{a,b},\delta'_{a,b}}$.
1: Find the deviation $\Delta_{a,b}$ in the in-memory cache; and in the datastore if not found in cache.
Ensure: While looking for deviations, also look for its inverse, i.e. $\Delta_{b,a}$ because only the upper triangular is stored. {If the inverse is retrieved then deviation must be inverted before operating on it.}
2: if $\Delta_{a,b}$ not found then
3:     print error!
4: end if
5: Update $\Delta'_{a,b} \leftarrow \Delta_{a,b} + diff_{\delta_{a,b},\delta'_{a,b}}$.
6: Store $\Delta'_{a,b}$ in the cache and also in the datastore.
Ensure: While writing to cache and to datastore, write to the inverse $\Delta'_{a,b}$ if it was initially retrieved. {If the inverse was retrieved then deviation must be inverted before storing it.}
7: Audit this update operation in the datastore, e.g. using user's IP address as the identity. {This is a typical insider threat in the cloud.}
```

Algorithm 3.3 An algorithm for the deletion of existing ratings.

```
Require: An item pair identified by $a$ and $b$, and $-\delta_{a,b}$.
1: Find the deviation $\Delta_{a,b}$ and cardinality $\phi_{a,b}$ in the in-memory cache; and in the datastore if not found in cache.
Ensure: While looking for deviations and cardinalities, also look for their inverses, i.e. $\Delta_{b,a}$ and $\phi_{b,a}$ because only the upper triangular is stored. {If the inverses are retrieved then deviation must be inverted before operating on it.}
2: if $\Delta_{a,b}$ and $\phi_{a,b}$ not found then
3:     print error!
4: end if
5: Update $\Delta'_{a,b} \leftarrow \Delta_{a,b} - \delta_{a,b}$ and $\phi'_{a,b} \leftarrow \phi_{x,y} - 1$.
6: Store $\Delta'_{a,b}$ and $\phi'_{a,b}$ in the cache and also in the datastore.
Ensure: While writing to cache and to datastore, write to the inverses $\Delta'_{b,a}$ and $\phi'_{b,a}$ if these were initially retrieved. {If the inverses were retrieved then deviation must be inverted before storing it.}
7: Audit this deletion operation in the datastore, e.g. using user's IP address as the identity. {This is a typical insider threat in the cloud.}
```

The steps for the prediction is shown in algorithm 3.4.

Algorithm 3.4 An algorithm for the prediction of an item.

```
Require: An item $x$ for which the prediction is to be made, a vector $\vec{RE} = \mathcal{E}(r_{a|a \neq x})$ of encrypted ratings for other items rated by the user (i.e. each item $a|a \neq x$) and the public key $pk_u$ of user $u$.
1:  total cardinality: $tc \leftarrow 0$; total deviation: $td \leftarrow 0$; total encrypted weight: $tew \leftarrow \mathcal{E}(0)$; total encrypted deviation: $ted \leftarrow \mathcal{E}(0)$.
2:  for $j = 1 \rightarrow length(\vec{RE})$ do
3:      Find the deviation $\Delta_{x,j}$ and cardinality $\phi_{x,j}$ in the in-memory cache; and in the datastore if not found in cache.
Ensure: While looking for deviations and cardinalities, also look for their inverses, i.e. $\Delta_{j,x}$ and $\phi_{j,x}$ because only the upper triangular is stored. {If the inverses are retrieved then deviation must be inverted before operating on it.}
4:      if $\Delta_{x,j}$ and $\phi_{x,j}$ found then
5:          $td \leftarrow td + \Delta_{x,j}$.
6:          $tc \leftarrow tc + \phi_{x,j}$.
7:          $tew \leftarrow \mathcal{E}(tew)(\mathcal{E}(r_j)^{\phi_{x,j}})$. {This step involves a homomorphic addition and a homomorphic multiplication.}
8:      end if
9:  end for
10: $ted \leftarrow \mathcal{E}(tew)(\mathcal{E}(td))$. {This is a homomorphic addition.}
11: return $ted$ and $tc$.
```

In the scheme described above, there is, in fact, one privacy leakage in the prediction phase: the number of items in the user's original rating vector. This can be addressed by computing the prediction at the user's end with the necessary elements from the deviation and cardinality matrices obtained from the cloud. The user can mask the actual rating vector by asking the cloud for an unnecessary number of extra items.

4 Evaluation

Implementation demo URL: http://evalgaej.appspot.com/.

Conforming to Google App Engine terminology, we will call the time taken by the application to respond to the user request as *application latency* or simply *latency*. This latency does not include network latencies encountered between our network and Google data centres.

4.1 Pre-computation

In the pre-computation stage, there is no cryptographic operation. The application latency is dominated by the time taken to complete a datastore write operation. Each such datastore write operation took between 80ms and 150ms. Google App Engine is designed to scale well. We did perform bulk addition of pair-wise deviations. The bulk adding client generated 32 threads to process the MovieLens 100K[1] dataset. The figure shows data of the 14 automatically allocated application instances. Each such instance can handle multiple requests and are pooled in memory. The QPS column shows how many queries per second each instance handled at that point while the latency is the average time taken to complete such requests.

4.2 Prediction

The prediction stage involves one homomorphic encryption as well as several homomorphic multiplications. Therefore, increasing the size of the encrypted rating vector typically linearly increased the time taken to predict. It is not dependent on the size of the deviation and cardinality matrices. This is shown in table 4.1. Note that given a 2048-bit Paillier cryptosystem, the total prediction time with 10 encrypted ratings as the input vector is reasonably fast: about 3.5 seconds, while the prediction time improves by about four-fold if we use a 1024-bit cryptosystem. Sometimes even if the input vector is large, pair-wise ratings between the queried for item and the items in the input vector may not exist, which will reduce the prediction time. Another factor impacting on performance is the availability of the deviation and cardinality matrix data on the distributed in-memory cache versus the datastore. In addition, GAE/J instances may also perform better or worse depending on the shared resources available on the Google's cloud computing clusters.

5 Conclusion and future work

Many existing privacy preserving collaborative filtering schemes pose challenges with practical implementations on the cloud. In this paper, we extend the well-known weighted Slope One collaborative filtering predictor to propose a novel approach and a practical implementation on a real world

[1]MovieLens datasets: `http://www.grouplens.org/node/73`.

Table 4.1: Comparison of typical prediction timings, based on the optimised equation 3.2.

Bit size[a]	**vector size**[b]	**prediction time**
1024	5	410ms
1024	10	825ms
2048	5	1900ms
2048	10	3500ms

[a]Paillier cryptosystem modulus bit size, i.e. $|n|$.
[b]Size of the encrypted rating vector.

SaaS construction PaaS cloud computing platform – the Google App Engine for Java. In our scheme, user's rating data is not stored in the cloud. Our scheme does not rely on any trusted third party for threshold decryption by allowing the users to encrypt and decrypt a prediction query and its results respectively.

The scheme proposed in this paper relies on the security guarantees of existing anonymising techniques, such as an anonymiser/mix network. We also assume that the user is honest. In future work, we plan to extend our proposed scheme by discarding those assumptions.

Acknowledgments

The work has been supported by the Japanese Ministry of Internal Affairs and Communications funded project "Research and Development of Security Technologies for Cloud Computing". Jaideep Vaidya's work is supported in part by the United States National Science Foundation under Grant No. CNS-0746943 and by the Trustees Research Fellowship Program at Rutgers, The State University of New Jersey. Contributions by Theo Dimitrakos relate to research in British Telecommunications under the IST Framework Programme 7 integrated project OPTIMIS that is partly funded by the European Commission under contract number 257115.

References

[1] A. Basu, H. Kikuchi, and J. Vaidya. Privacy-preserving weighted Slope One predictor for Item-based Collaborative Filtering. In *Proceedings of the international workshop on Trust and Privacy in Distributed Information Processing (workshop at the IFIPTM 2011), Copenhagen, Denmark*, 2011.

[2] D. Catalano, M. Di Raimondo, D. Fiore, R. Gennaro, and O. Puglisi. Fully non-interactive onion routing with forward-secrecy. In *Applied Cryptography and Network Security*, pages 255–273. Springer, 2011.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pages 46–66. Springer, 2001.

[4] G. Danezis. Mix-networks with restricted routes. In *Privacy Enhancing Technologies*, pages 1–17. Springer, 2003.

[5] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13*, pages 21–21. USENIX Association, 2004.

[6] J. Furukawa and K. Sako. Mix-net system, January 8 2010. US Patent Application. 20,100/115,285.

[7] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. *Society for Industrial Mathematics*, 2005.

[8] M.G. Reed, P.F. Syverson, and D.M. Goldschlag. Anonymous connections and onion routing. *Selected Areas in Communications*, 16(4):482–494, 1998.

[9] L. Von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. *Advances in Cryptology EUROCRYPT*, pages 646–646, 2003.

[10] I. Wakeman, D. Chalmers, and M. Fry. Reconciling privacy and security in pervasive computing: the case for pseudonymous group membership. In *Proceedings of the 5th International workshop on Middleware for pervasive and ad-hoc computing: held at the ACM/IFIP/USENIX 8th International Middleware Conference*, pages 7–12. ACM, 2007.