# Privacy-preserving collaborative filtering for the cloud

Anirban Basu
Graduate School of Engineering, Tokai University
2-3-23 Takanawa, Minato-ku, Tokyo 108-8619, Japan
abasu@cs.dm.u-tokai.ac.jp

Jaideep Vaidya
MSIS Department, Rutgers, The State University of New Jersey
1, Washington Park, Newark, New Jersey 07102-1897, USA
jsvaidya@business.rutgers.edu

Hiroaki Kikuchi
Graduate School of Engineering, Tokai University
1117, Kitakaname, Hiratsuka, Kanagawa 259-1292, Japan
kikn@tokai.ac.jp

Theo Dimitrakos
Security Futures Practice, Research & Technology, BT
Adastral Park, Martlesham Heath, IP5 3RE, UK
theo.dimitrakos@bt.com

***Abstract*—Rating-based collaborative filtering (CF) enables the prediction of the rating that a user will give to an item, based on the ratings of other items given by other users. However, doing this while preserving the privacy of rating data from individual users is a significant challenge. Several privacy preserving schemes have, so far been proposed in prior work. However, while these schemes are theoretically feasible, there are many practical implementation difficulties on real world public cloud computing platforms. In this paper, we approach the generalised problem of privacy preserving collaborative filtering from the cloud perspective and propose an efficient and secure approach that is built for the cloud. We present our implementation experiences and experimental results based on the Google App Engine for Java (GAE/J) cloud platform.**

***Index Terms*—privacy-preserving collaborative filtering; cloud computing; slope one; homomorphic encryption;**

## I. Introduction

Cloud computing is characterised by pooling and sharing of resources, broad network access, rapid elasticity, on-demand service provisioning (with a strong self-service element), offering measured service, supporting (although not necessitating) multi-tenancy. Cloud architecture embodiments offer a means of delivering ICT infrastructure, platform (i.e. application execution environment) and software as a service. They also support several deployment models including private, community, public and hybrid. The benefits promised by this paradigm include cost and performance optimisation, economy of scale, flexible utilisation and charging model, ease of connectivity and access to shared services, cost efficient introduction of redundancy and continuity of provision. Security, resilience and compliance are the main concerns that are challenging wider use of cloud computing and most likely to drive remaining innovation and market differentiation efforts in this area.

The key security challenges facing cloud computing include regulatory compliance, absence of security standards and certification, confidentiality and integrity of data at rest or in motion in the cloud, data and process isolation, multi-tenancy of shared security services. Such challenges come on top of the common security issues relating to the risk of externalising management processes commonly performed by privileged users, vulnerabilities introduced by inadequately protected use of virtualization technology to empower cloud services, and the security of its integration with corporate IT infrastructure, and lack of implementing protection in depth when de/re-perimeterising through the integration of cloud services into the corporate infrastructure.

Be it traditional web-based information providers, or those operating on cloud infrastructures, the rapid growth of services provided over the World Wide Web has resulted in individuals having to sift through overwhelming volumes of information – the problem of information overload [1]. *Recommendation systems* are an answer to solving this problem. Automated recommendation systems employ two techniques: *profile-based* and *collaborative filtering* (CF). The former uses information that relate to users' tastes in order to match the items to be recommended. Recommendation through CF makes use of the recorded preferences of the community. Profile-based recommendation for a user with rich profile information is thorough. However, CF is fairly accurate within reason, without the need for the users' profile information.

Grouped by filtering techniques, numeric rating based CF is broadly classified into: *memory-based* or *neighbourhood-based* and *model-based*. In *memory-based* approaches, recommendations are developed from user or item neighbourhoods, i.e. some sort of proximity (or deviation) measures between the ratings, e.g. cosine similarity, Euclidean distance and various statistical correlation coefficients. Memory-based CF can also be distinguished into: *user-based* and *item-based*. In the former, CF is performed using neighbourhood between users computed from the ratings provided by them. The latter is item-based where prediction is obtained using item neighbourhoods, i.e. proximity (or deviation) of ratings between various items.

In *model-based* approaches, the original user-item ratings dataset is used to *train* a compact model, which is then used for prediction. The model is developed by methods borrowed from artificial intelligence, such as Bayesian classification, latent classes and neural networks; or, from linear algebra, e.g. singular value decomposition (SVD), latent semantic indexing (LSI) and principal component analysis (PCA). Model-based algorithms are usually fast to query but relatively slow to update.

CF based approaches perform better with the availability of more data. Cross domain recommendations are possible, if the corresponding data can be utilised (e.g. a person with a strong interest in horror movies may also rate certain Halloween products highly). However, sharing user-item preferential (i.e. rating) data for use in CF poses significant privacy and security challenges. Competing organisations, e.g. Netflix and Blockbuster may not wish to share specific user information, even though both may benefit from such sharing. Users themselves might not want detailed information about their ratings and buying habits known to any single organisation. To overcome this, there has been recent work in privacy-preserving collaborative filtering (PPCF) that can enable CF without leaking private information. Indeed, in CF, achieving accuracy and preserving privacy are orthogonal problems. The two main directions of research in privacy-preserving collaborative filtering are: *encryption-based* and *randomisation-based*. In encryption-based techniques, prior to sharing individual user-item ratings data are encrypted using cryptosystems that support homomorphic properties. In randomisation-based privacy preserving techniques, the ratings data is randomised either through random data swapping or data perturbation or anonymisation.

However, many theoretically feasible collaborative filtering schemes face practical implementation difficulties on real world public cloud computing platforms. These difficulties include computational complexity, dataset size and hence scalability, and dependence on trusted third party, amongst others. In this paper, we re-visit the generalised problem of privacy preserving collaborative filtering and demonstrate a novel approach and a realistic implementation of a privacy preserving weighted Slope One scheme on the Google App Engine for Java (GAE/J) – a specialised Platform-as-a-Service (PaaS) cloud offering that enables Software-as-a-Service (SaaS) construction.

### A. Motivating example

Consider the following motivating example: Alice has seen and rated a number of films from the Japanese animation Studio Ghibli on a film rating website built atop a cloud computing platform. She intends to see the 2011 film: *From Up on Poppy Hill* next and would like the film rating website to give her a rating prediction for this film based on her ratings of the other films that she has rated as well as such ratings from the community. She is completely unaware of (and does not care) who else has rated the various films in this way apart from obtaining a reasonable rating for *From Up on Poppy Hill*. Alice also is unwilling to send her entire rating vector for films she has rated to any third party but is happy to send some in such a way that they are de-linked from her identity through some anonymising mechanism. If Alice is to obtain a rating for *From Up on Poppy Hill*, 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 the ratings for any of her previously rated films.

### B. Objectives

Thus, we aim to build a privacy preserving collaborative filtering scheme on the cloud for any item such that: 1) a contributing user does not have to 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) the scheme is robust to insider threats to data privacy from the cloud infrastructure itself. We also assume honest but curious user participation, although we discuss in this paper what happens if we give up this assumption. The formal problem statement is presented in Section III-C.

To achieve this, in our approach, the user will use anonymising techniques (e.g. IPv4 network address translation and dynamic IPs, anonymiser network such as Tor, and pseudonymous identities; see: [2], [3], [4], [5], [6], [7], [8]) 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 anonymising mechanism, and are bounded by it.

### C. 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) While our solution is built for the case where each user independently contributes ratings, we also discuss the extension to the case where multiple cloud agents already own ratings for some of the items, and wish to collaboratively utilise item-based CF.

The rest of the paper is organised as follows: in §II, we list some of the related work in privacy preserving collaborative filtering. In §III, we introduce the building blocks for our proposed scheme, which is presented in §IV. We present the evaluation of our implementation in §VI preceded by practical implementation considerations in §V before concluding with future directions in §VII.

## II. Related work

In recent years, privacy has attracted a lot of attention. There are a number of existing works on privacy-preserving collaborative filtering (PPCF). One of the earliest such efforts is due to Canny [9] which uses a partial Singular Value Decomposition (SVD) model and homomorphic encryption to devise a multi-party PPCF scheme. Canny [10] also proposes a new solution based on a probabilistic factor analysis model that can handle missing data, and provides privacy through a peer-to-peer protocol.

Polat and Du have also investigated this problem from several perspectives: [11] proposes a randomized perturbation technique to protect individual privacy while still producing accurate recommendations results; [12] presents a privacy-preserving protocol for collaborative filtering over vertically partitioned data, while [13] presents a randomisation based SVD approach, while [14] enables recommendations via item-based algorithms using randomized response techniques.

Berkovsky et al. [15] present a decentralized distributed storage of user data combined with data modification techniques to mitigate privacy issues. Tada et al. presented a privacy-preserving item-based collaborative filtering scheme in [16].

Cissée and Albayrak [17] develop a method for privacy-preserving recommender systems based on a multi-agent technology which enables applications to generate recommendations via various filtering techniques while preserving the privacy of all participants. Ahmad and Khokar proposed [18] a privacy preserving collaborative filtering schemes for web portals using a trusted third party for threshold decryption. Kaleli and Polat propose [19] a naïve Bayesian classifier based CF over a P2P topology where the users protect the privacy of their data using masking, which is comparable to randomisation. Another homomorphic encryption based SVD scheme has been proposed by Han, et al. [20] but the authors also describe that their scheme does not scale well for realistic datasets. Gong et al. [21] present a new collaborative filtering technique based on randomised perturbation and secure multi-party computation. However, neither are these techniques has been built for the cloud, nor are they efficient enough for large-scale general purpose use.

Our work is the first PPCF scheme that includes a practical cloud-based implementation and results. In [22], we have proposed a privacy preserving (using encryption) CF scheme based on the well known weighted Slope One predictor [23]. Our prior 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.

## III. Background

### A. Slope One collaborative filtering

The Slope One predictors due to Lemire and McLachlan [23] are item-based collaborative filtering schemes that predict the rating of an item for a user from a pair-wise deviations of item ratings. Slope One CF can be evaluated in two stages: pre-computation and prediction of ratings.

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

$$\overline{\delta_{a,b}} = \frac{\Delta_{a,b}}{\phi_{a,b}} = \frac{\sum_i \delta_{i,a,b}}{\phi_{a,b}} = \frac{\sum_i (r_{i,a} - r_{i,b})}{\phi_{a,b}}. \tag{III.1}$$

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 given as:

$$\begin{aligned} r_{u,x} &= \frac{\sum_{a|a\neq x}(\overline{\delta_{x,a}} + r_{u,a})\phi_{x,a}}{\sum_{a|a\neq x}\phi_{x,a}} \\ &= \frac{\sum_{a|a\neq x}(\Delta_{x,a} + r_{u,a}\phi_{x,a})}{\sum_{a|a\neq x}\phi_{x,a}}. \end{aligned} \tag{III.2}$$

The matrices $\Delta$ and $\phi$ 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 are irrelevant. The lower triangular for the deviation matrix is the additive inverse of the upper triangular while that of the cardinality matrix is the same as its upper triangular. These two matrices contain information about item pairs only computed from ratings given by all users, so these do not pose any privacy risk to any particular user's rating data. Despite the plaintext deviation and cardinality storage, if the user sends his/her rating vector to the prediction function then there is a privacy threat. Therefore, we use encrypted rating prediction.

### B. Homomorphic cryptosystem

Paillier public-key cryptosystem [24] exhibits additively homomorphic properties. Denoting encryption and decryption functions as $\mathcal{E}()$ and $\mathcal{D}()$ respectively, the encryption of the sum of two plaintext messages $m_1$ and $m_2$ is the modular product of their individual ciphertexts:

$$\mathcal{E}(m_1 + m_2) = \mathcal{E}(m_1) \cdot \mathcal{E}(m_2) \tag{III.3}$$

and, the encryption of the product of one plaintext messages $m_1$ and a plaintext integer multiplicand $\pi$ is the modular exponentiation of the ciphertext of $m_1$ with $\pi$ as the exponent:

$$\mathcal{E}(m_1 \cdot \pi) = \mathcal{E}(m_1)^{\pi}. \tag{III.4}$$

The Paillier cryptosystem with optimisations are described in three steps: *key generation* (algorithm III.1), *encryption* (algorithm III.2) and *decryption* (algorithm III.3).

Our implementation of Paillier cryptosystem supports negative numbers for plaintext (see §V-C) because quite often a deviation of two ratings can be negative.

**Algorithm III.1** Paillier cryptosystem key generation.

```
1: Generate two large prime numbers $p$ and $q$, each with half the specified modulus bit length for the cryptosystem.
Ensure: $gcd(pq, (p-1)(q-1)) = 1$ and $p \neq q$.
2: Modulus $n = pq$ and pre-compute $n^2$.
3: Compute $\lambda = lcm(p-1, q-1) = \frac{(p-1)(q-1)}{gcd(p-1,q-1)}$.
4: $g \leftarrow (1+n)$. {Optimised here but originally: select random $g \in Z^*_{n^2}$ such that $n$ divides the order of $g$.}
Ensure: $gcd(L(g^\lambda \mod n^2), n) = 1$ where $L(u) = \frac{u-1}{n}$. {Optimisation: $g^\lambda \mod n^2 = (1+n\lambda) \mod n^2$.}
5: Pre-compute the modular multiplicative inverse $\mu = L(g^\lambda \mod n^2)^{-1} \mod n$.
6: return Public key: $(n, n^2, g)$ and private key: $(\lambda, \mu)$.
```

**Algorithm III.2** Paillier encryption algorithm.

```
Require: Plaintext $m \in Z_n$.
1: Choose random $r \in Z^*_n$.
2: return Ciphertext $c \leftarrow (1+mn)r^n \mod n^2$. {Optimised here but originally: $c \leftarrow g^m r^n \mod n^2$.}
```

**Algorithm III.3** Paillier decryption algorithm.

```
Require: Ciphertext $c \in Z^*_{n^2}$.
1: return Plaintext $m \leftarrow L(c^\lambda \mod n^2)\mu \mod n$.
```

### *C. Problem statement*

The problem can be formally defined in the following fashion:

**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 deterministically linked back to any user.*
- *any user should be able to obtain a prediction without leaking his/her private rating information.*

Note that this closely corresponds to the perfect horizontal partitioning case, wherein each site contains the data of one single user. However, this is not quite the same, since each user may not rate all of the items.

## IV. 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. The overall user-interaction diagram of our proposed model is presented in figure IV.1 showing the addition of rating data only.

### *A. 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, mixed networks, pseudonymous group memberships, 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.

*1) Case of new ratings:* In the pre-computation stage, the average deviations of ratings from item $a$ to item $b$ is given in equation III.1. The cloud application only maintains a list of items; their pairwise deviations and cardinalities but no other user data. The process of rating addition is described in algorithm IV.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 IV.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 also described in algorithm IV.2.

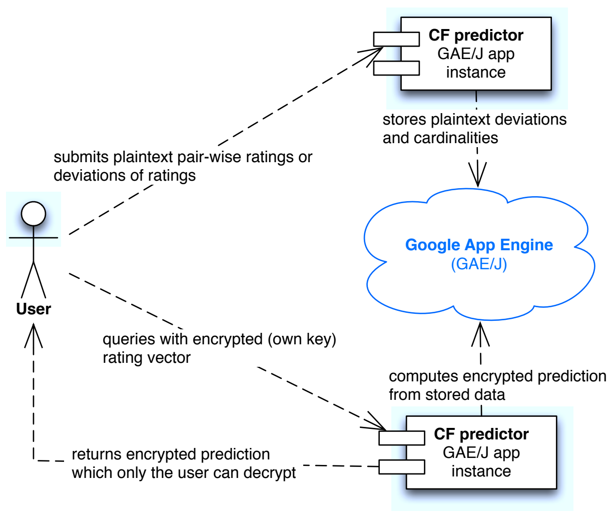

CF predictor
GAE/J app
instance
stores plaintext deviations and cardinalities
submits plaintext pair-wise ratings or deviations of ratings
Google App Engine
(GAE/J)
User
queries with encrypted (own key) rating vector
computes encrypted prediction from stored data
returns encrypted prediction which only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

Figure IV.1. User-interaction diagram of our scheme.

### *B. Prediction*

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

**Algorithm IV.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.
1: Find the deviation $\Delta_{a,b}$ and cardinality $\phi_{a,v}$.
2: {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.}
3: if $\Delta_{a,b}$ and $\phi_{a,b}$ not found then
4:     $\Delta_{a,b} \leftarrow 0$ and $\phi_{a,b} \leftarrow 0$.
5: end if
6: Update $\Delta'_{a,b} \leftarrow \Delta_{a,b} + \delta_{a,b}$ and $\phi'_{a,b} \leftarrow \phi_{a,b} + 1$.
7: Store $\Delta'_{a,b}$ and $\phi'_{a,b}$.
Ensure: While storing, 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.}
8: 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.}
```

**Algorithm IV.2** An algorithm for the updates or deletions of existing ratings.

```
Require: In case of update, an item pair identified by $a$ and $b$, and $diff_{\delta_{a,b},\delta'_{a,b}}$; or in case of deletion: an item pair identified by $a$ and $b$, and $-\delta_{a,b}$.
1: Find the deviation $\Delta_{a,b}$ and cardinality $\phi_{a,b}$.
2: {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.}
3: if $\Delta_{a,b}$ and $\phi_{a,b}$ not found then
4:     print error!
5: end if
6: In case of an update, $\Delta'_{a,b} \leftarrow \Delta_{a,b} + diff_{\delta_{a,b},\delta'_{a,b}}$; or in case of deletion: $\Delta'_{a,b} \leftarrow \Delta_{a,b} - \delta_{a,b}$ and $\phi'_{a,b} \leftarrow \phi_{x,y} - 1$.
7: Store $\Delta'_{a,b}$ and $\phi'_{a,b}$, if $\phi'_{a,b}$ was changed in case of deletion.
Ensure: While storing, 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.}
8: Audit this update or 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 deviation matrix and to apply homomorphic multiplication according to the prediction equation defined in equation IV.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.

$$r_{u,x} = \frac{\mathcal{D}(\prod_{a|a\neq x}(\mathcal{E}(\Delta_{x,a})(\mathcal{E}(r_{u,a})^{\phi_{x,a}})))}{\sum_{a|a\neq x}\phi_{x,a}}. \qquad \text{(IV.1)}$$

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

$$r_{u,x} = \frac{\mathcal{D}(\mathcal{E}(\sum_{a|a\neq x}\Delta_{x,a})\prod_{a|a\neq x}(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))}{\sum_{a|a\neq x}\phi_{x,a}}. \qquad \text{(IV.2)}$$

The steps for the prediction is shown in algorithm IV.3.

**Algorithm IV.3** 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 $\phi_{x,j}$.
4:     {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.}
5:     if $\Delta_{x,j}$ and $\phi_{x,j}$ found then
6:         $td \leftarrow td + \Delta_{x,j}$.
7:         $tc \leftarrow tc + \phi_{x,j}$.
8:         $tew \leftarrow \mathcal{E}(tew)(\mathcal{E}(r_j)^{\phi_{x,j}})$. {This step involves a homomorphic addition and a homomorphic multiplication.}
9:     end if
10: end for
11: $ted \leftarrow (\mathcal{E}(tew)\mathcal{E}(td))$. {This is a homomorphic addition.}
12: return $ted$ and $tc$. {User decrypts $ted$; the predicted result is $r_{u,x} = \frac{\mathcal{D}(ted)}{tc}$}
```

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.

### *C. Vertical partition across multiple organisations*

While the solution presented above works when each user knows their own ratings, it will not work if there are two or more cloud applications with disjoint item and user sets (e.g. Yahoo movies, Netflix and IMDB have their own cloud applications), the prior PPCF solution will work if we extend our scheme as follows. For example, assume the two applications know the values $r_a$ and $r_b$ respectively for two different items for the same user. To compute the deviation $r_a - r_b$ for that user without revealing it to either or to any third site, assume that the first application randomly splits the value $r_a$ into two shares (thus, randomly chooses $z_{a1} \in Z$ and compute $z_{a2} \in Z$ such that $r_a = z_{a1} + z_{a2}$). Similarly, the other application randomly chooses $z_{b1}$ and computes $z_{b2}$ such that $r_b = z_{b1} + z_{b2}$. The first application sends $z_{a2}$ to the other, and after receiving $-z_{b1}$ from it, computes $c_1 = z_{a1} - z_{b1}$, while the other computes $c_2 = z_{a2} - z_{b2}$. Finally, both can send $c_1$ and $c_2$ respectively in an unlinkable fashion to the cloud PPCF application, which obtains $c_1 + c_2 = z_{a1} - z_{b1} + z_{a2} - z_{b2} = r_a - r_b$. It is possible to do this with $k$ cloud applications as well, where each value is split into $k$ splits, to make the process more resistant to collusion. Note that one problem with this approach is that the same deviation is split over the $k$ splits – effectively, cardinality is increased by $k$ instead of by 1. One approximate fix for this is simply to increase the deviation $k$-fold. Thus, each application will send in $k * c_i$, instead of $c_i$. However, this will only approximate the effect since instead of adding the true deviation and 1, we end up adding $k$ times the true deviation and $k$. Alternatively, the cardinalities can be fixed by exposing a reduction capability, so that anyone can reduce the cardinalities appropriately in an unlinkable fashion. Assuming semi-honest participants, this works fine. If this is not suitable, the cardinalities can also be computed correctly by flagging to the PPCF application that submissions of all $c_k$ splits should be treated as one submission. However, in this case, the linkage between the splits will be established and now the security relies on the lack of collusion among all $k$ applications. We will explore this issue more in the future, and also leave the experimental results for it for future work.

## V. Implementation considerations

### A. Cache based datastore access

From the documentation of Google App Engine for Java (GAE/J) and from our experiments, we observe that the I/O operations with the datastore are CPU intensive. To speed up several concurrent lookup operations, we have used the GAE/J Memcache API which is a distributed in-memory cache. In the aforementioned algorithms, the deviation and cardinality matrices are read from the cache; and the datastore only if not found in cache. On the other hand, write operations to those matrices are done simultaneously to the cache and the datastore to ensure consistency. Although values in the cache are volatile, in the case of a cache-hit, the read operation is quicker than the datastore read operation. For writing, however, we have to flush the updates to the cache immediately in order to ensure consistency between the cache and the datastore. In our experimental implementation, we have a transactional *write-(to-datastore)-immediately* strategy. However, with multiple concurrent requests to the servlets, this may generate performance delays. In future, we plan to use a mechanism for optimised, consistent asynchronous writes to the datastore.

### B. Parallelism in pre-computation and in prediction

While parallelism is not necessary, it can be of help when the number of items is large, since effectively the number of deviations is in order of $O(n^2)$. Our scheme is intrinsically parallel in the pre-computation stage. In the case of a large number of concurrent user rating pair addition requests, a MapReduce [25] style parallelism in *pre-computation* is achieved at the user's end because the user submits an item pair instead of the entire rating vector and GAE/J handles the user requests concurrently. Each item pair ratings can be thought of as the output of a *map* function, where the key is composited by the identities of the two items and the value consists of the deviation of the ratings and the cardinality of unity. The *reduce* function in our implementation combines those individual item-item pairwise deviation-cardinality tuples to generate the combined value for each item-item pair.

In the *prediction* stage, we do not use concurrency yet. The speed of this process depends on the size of the cryptosystem modulus as well as the number of encrypted ratings provided because computing the encrypted prediction involves encryption and homomorphic multiplications, which are computationally expensive. In future, we plan to parallelise the prediction stage through asynchronous access to the datastore such that the prediction is progressively built and the user is kept informed at the client front-end of this progress, e.g. by using an asynchronous client notification framework such as the Android Cloud to Device Messaging (C2DM)[1]. From a performance standpoint, this will ensure that the servlet's response time will not be dependent on the size of the encrypted rating vector provided by the user, and therefore will not be subject to the GAE/J execution deadline of 30s.

### C. Handling negatives in non-threshold Paillier cryptosystem

We have used an optimised version of a non-threshold Paillier cryptosystem. Due to the absence of the threshold servers, decryption is faster than the threshold version. To cater for negative plaintext inputs, we divide the ring of $n$ in two parts and consider any plaintext $m \geq \frac{n}{2}$ as negative.

### D. Attack model: malicious cloud

The implementation of our model accepts, as input from a user, ratings for a pair of items, or the deviations of their ratings. The implementation quietly collects the user's IP addresses without the user's permission (constituting an example attack scenario). With the presence of IP address, a level of linkability between rating pairs can be attained. However, through the use of a network address translator (NAT), or an onion routing protocol (e.g. Tor) or dynamic IP addresses, the user's IP addresses act as pseudonyms with a many-to-many relation between a pseudonym and a real user, i.e. one IP address may be re-used by many users and one user may have many IP addresses over time. This makes predictable linkability hard and often incorrect. If the user uses different IP addresses for each separate pair of ratings then at any point

[1] Android C2DM: http://code.google.com/android/c2dm/.

in time, the server can at most link only one pair of ratings to a user. In reality, it will link more (and these are wrong) because of the IP addresses being re-used by other users.

## VI. Evaluation

*Demo URL:* http://gaejppcf.appspot.com/.

We evaluate the aforementioned scheme with an implementation on the Google App Engine for Java (GAE/J). For the sake of simplicity, we have not considered the case where there may be more than one applications on the GAE/J which interact with each other to make a prediction, i.e. the scenario described in §IV-C. Also, for test use, we have not implemented comprehensive error-checking on the web front-end. One will notice that the web application provides means to perform cryptographic operations, e.g. key generation, encryption and decryption on the GAE/J, which are provided for experimental purposes only, and they should be performed at the client's end in reality. For browser compatibility, we have used Google Web Toolkit[2] to code the web-based user interface.

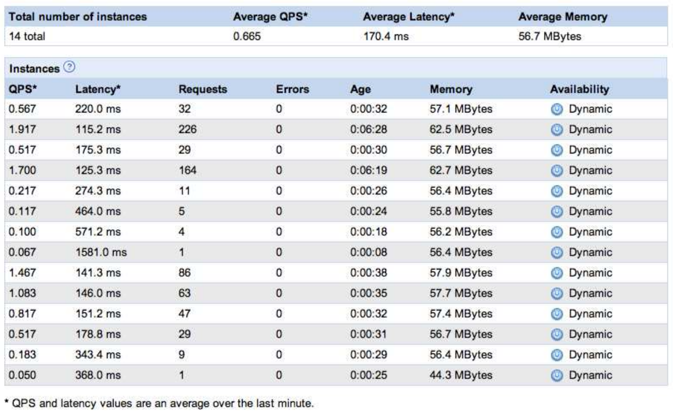
Conforming to Google App Engine terminology, we will call the time taken by the application to respond to the user request as *application latency* or simply *latency*. This latency does not include network latencies encountered between our network and Google data centres.

### A. Pre-computation

In the pre-computation stage, there is no cryptographic operation on the cloud. 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 using the High-Replication datastore and about half that time while using the Master-Slave datastore. We also performed bulk addition of pair-wise deviations. Figure VI.1 is a screenshot of the Google App Engine control panel showing the typical load generated during the bulk data addition procedure. The bulk adding client generated 32 threads to process a part of the MovieLens 100K[3] dataset. The figure shows data of the 14 automatically allocated application instances[4]. Each such instance can handle multiple requests and are pooled in memory. The QPS (queries per second) column shows the average number of queries per second each instance handled at that point while the latency is the average time taken to complete such requests. This simulates the scenario if many users concurrently submit many requests to add rating or deviation data.

### B. 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 VI.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 5 seconds, while the prediction time improves by almost eight-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 availability on the Google's cloud computing clusters.



| Total number of instances | Average QPS* | Average Latency* | Average Memory |
|---|---|---|---|
| 14 total | 0.665 | 170.4 ms | 56.7 MBytes |

Instances

| QPS* | Latency* | Requests | Errors | Age | Memory | Availability |
|---|---|---|---|---|---|---|
| 0.567 | 220.0 ms | 32 | 0 | 0:00:32 | 57.1 MBytes | Dynamic |
| 1.917 | 115.2 ms | 226 | 0 | 0:06:28 | 62.5 MBytes | Dynamic |
| 0.517 | 175.3 ms | 29 | 0 | 0:00:30 | 56.7 MBytes | Dynamic |
| 1.700 | 125.3 ms | 164 | 0 | 0:06:19 | 62.7 MBytes | Dynamic |
| 0.217 | 274.3 ms | 11 | 0 | 0:00:26 | 56.4 MBytes | Dynamic |
| 0.117 | 464.0 ms | 5 | 0 | 0:00:24 | 55.8 MBytes | Dynamic |
| 0.100 | 571.2 ms | 4 | 0 | 0:00:18 | 56.2 MBytes | Dynamic |
| 0.067 | 1581.0 ms | 1 | 0 | 0:00:08 | 56.4 MBytes | Dynamic |
| 1.467 | 141.3 ms | 86 | 0 | 0:00:38 | 57.9 MBytes | Dynamic |
| 1.083 | 146.0 ms | 63 | 0 | 0:00:35 | 57.7 MBytes | Dynamic |
| 0.817 | 151.2 ms | 47 | 0 | 0:00:32 | 57.4 MBytes | Dynamic |
| 0.517 | 178.8 ms | 29 | 0 | 0:00:31 | 56.7 MBytes | Dynamic |
| 0.183 | 343.4 ms | 9 | 0 | 0:00:29 | 56.4 MBytes | Dynamic |
| 0.050 | 368.0 ms | 1 | 0 | 0:00:25 | 44.3 MBytes | Dynamic |

* QPS and latency values are an average over the last minute.

Figure VI.1. Typical load during pre-computation on the Google App Engine instances during partial bulk addition of the MovieLens 100K dataset.

Table VI.1
Comparison of typical prediction timings, based on the optimised equation IV.2.

| **Bit size**[a] | **Query vector size**[b] | **Typical prediction time** |
|---|---|---|
| 1024 | 5 | 500ms |
| 1024 | 10 | 650ms |
| 2048 | 5 | 3800ms |
| 2048 | 10 | 5000ms |

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

### C. Security

*1) Insider threat in the cloud:* Our implementation collects the IP address of the user in an attempt to link the ratings to a particular IP. We performed rating and deviation submission from a number of different networks. So long as the same user did not submit ratings and deviations from the same WAN IP, the cloud application could do little to conclusively link that specific user to his/her rating or deviation submissions. The cloud only learns that a pair of ratings or their deviation by a particular user (provided the user identity changes in the

[2]See: http://code.google.com/webtoolkit/.
[3]MovieLens datasets: http://www.grouplens.org/node/73.
[4]This is a snapshot of the GAE/J control panel. The number of instances change over time depending on the number of user requests.

consecutive submission), which is even insufficient to launch an offline knowledge based inference attack on the user's private rating vector.

*2) 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 to require human intervention, and hence slow down the number of additions, updates and deletions.

## VII. Conclusion and future work

Many existing privacy preserving collaborative filtering schemes pose challenges with practical implementations on real world cloud computing platforms. In this paper, we extend the well-known weighted Slope One collaborative filtering predictor to propose a novel approach and a practical implementation on the Google App Engine for Java cloud computing platform. 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 results of our experiments show that the user can obtain predictions quickly despite encrypted rating queries.

The scheme proposed in this paper partly 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. We also plan to run more experiments with exhaustive user studies. We also plan to enhance the performance further by optimising our implementation.

## Acknowledgments

The work at Tokai University has been supported by the Japanese Ministry of Internal Affairs and Communications funded project "Research and Development of Security Technologies for Cloud Computing" involving Tokai University, Waseda University, NEC, Hitachi and KDDI. 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] J. B. Schafer, J. Konstan, and J. Riedi, "Recommender systems in e-commerce," in *Proceedings of the 1st ACM conference on Electronic Commerce*. New York, USA: ACM Press, 1999, pp. 158–166.

[2] M. Reed, P. Syverson, and D. Goldschlag, "Anonymous connections and onion routing," *Selected Areas in Communications*, vol. 16, no. 4, pp. 482–494, 1998.

[3] 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*. Springer, 2011, pp. 255–273.

[4] 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*. USENIX Association, 2004, pp. 21–21.

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

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

[7] G. Danezis, "Mix-networks with restricted routes," in *Privacy Enhancing Technologies*. Springer, 2003, pp. 1–17.

[8] 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*. ACM, 2007, pp. 7–12.

[9] J. Canny, "Collaborative filtering with privacy," *Proceedings 2002 IEEE Symposium on Security and Privacy*, pp. 45–57, 2002.

[10] ——, "Collaborative filtering with privacy via factor analysis," in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '02. New York, NY, USA: ACM, 2002, pp. 238–245. [Online]. Available: http://doi.acm.org/10.1145/564376.564419

[11] H. Polat and W. Du, "Privacy-preserving collaborative filtering using randomized perturbation techniques," in *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*. IEEE, 2003, pp. 625–628.

[12] ——, "Privacy-preserving collaborative filtering on vertically partitioned data," *Knowledge Discovery in Databases: PKDD 2005*, pp. 651–658, 2005.

[13] ——, "SVD-based collaborative filtering with privacy," *Proceedings of the 20th ACM Symposium on Applied Computing*, 2005.

[14] ——, "Achieving private recommendations using randomized response techniques," *Advances in Knowledge Discovery and Data Mining*, pp. 637–646, 2006.

[15] S. Berkovsky, Y. Eytani, T. Kuflik, and F. Ricci, "Enhancing privacy and preserving accuracy of a distributed collaborative filtering," in *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 2007, pp. 9–16.

[16] M. Tada, H. Kikuchi, and S. Puntheeranurak, "Privacy-preserving collaborative filtering protocol based on similarity between items," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. IEEE, 2010, pp. 573–578.

[17] R. Cissée and S. Albayrak, "An agent-based approach for privacy-preserving recommender systems," in *Proceedings of the 6th International Joint Conference on Autonomous agents and Multiagent Systems*. ACM, 2007, pp. 1–8.

[18] W. Ahmad and A. Khokhar, "An architecture for privacy preserving collaborative filtering on web portals," in *Proceedings of the Third International Symposium on Information Assurance and Security*. IEEE Computer Society, 2007, pp. 273–278.

[19] C. Kaleli and H. Polat, "P2P collaborative filtering with privacy," *Turkish Journal of Electric Electrical Engineering and Computer Sciences*, vol. 8, no. 1, pp. 101–116, 2010.

[20] S. Han, W. K. Ng, and P. S. Yu, "Privacy-Preserving Singular Value Decomposition," *IEEE 25th International Conference on Data Engineering*, pp. 1267–1270, 2009.

[21] S. Gong, "Privacy-preserving collaborative filtering based on randomized perturbation techniques and secure multiparty computation," *IJACT: International Journal of Advancements in Computing Technology*, vol. 3, no. 4, 2011.

[22] 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.

[23] D. Lemire and A. Maclachlan, "Slope one predictors for online rating-based collaborative filtering," *Society for Industrial Mathematics*, 2005.

[24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Advances in Cryptology EUROCRYPT*, vol. 1592. Springer, 1999, pp. 223–238.

[25] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.