Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Privacy preserving collaborative filtering for SaaS enabling PaaS clouds

Anirban Basu[1*], Jaideep Vaidya[2], Hiroaki Kikuchi[1], Theo Dimitrakos[3] and Srijith K Nair[3]

Abstract
Recommender systems use, amongst others, a mechanism called collaborative filtering (CF) to predict the rating that a user will give to an item given the ratings of other items provided by other users. While reasonably accurate CF can be achieved with various well-known techniques, preserving the privacy of rating data from individual users poses 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 present our implementation experience and experimental results on two public Software-as-a-Service (SaaS) enabling Platform-as-a-Service (PaaS) clouds: the Google App Engine for Java (GAE/J) and the Amazon Web Services Elastic Beanstalk (AWS EBS).[a]

Keywords: Collaborative filtering, Privacy, Cloud computing, Homomorphic cryptosystem, Slope one

Introduction

Pooling and sharing of resources, broad network access, rapid elasticity, on-demand service provisioning (with a strong self-service element), offering of measured service, supporting (although not necessitating) multi-tenancy, are some of the features that characterise cloud computing. Cloud architecture offers a means of delivering ICT infrastructure, platform (i.e. application execution environment) and software as a service. 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 some of 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 rapid growth of information 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* have been used as a rescue. Automated recommendation systems employ two techniques: *profile analysis* and *collaborative filtering* (CF). The former matches the items to be recommended using information that relate to users' tastes; while 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 even without the need for the users' profile information. CF has, thus, positioned itself as one of the predominant means of generating recommendations.

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. 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. In item-based schemes, 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

*Correspondence: abasu@cs.dm.u-tokai.ac.jp
[1]Graduate School of Engineering, Tokai University. 2-3-23 Takanawa, Minato-ku, Tokyo 108-8619, Japan
Full list of author information is available at the end of the article

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, the sharing of 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. 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 in real world computing infrastructures like 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: we present and extend our previously proposed novel approach [2] and realistic implementations of a privacy preserving weighted Slope One scheme on the Google App Engine for Java (GAE/J) and the Amazon Web Services Elastic Beanstalk (AWS EBS) – two specialised Platform-as-a-Service (PaaS) cloud services that enable Software-as-a-Service (SaaS) construction.

Motivating example

With data about individual's personal preferences being increasingly stored by applications built atop various SaaS and PaaS cloud platforms, there is a growing need to ensure that privacy of such data is preserved while enabling efficient statistical operations on the data, even over multiple cloud applications belonging to multiple cloud sites. Consider the following example: Alice has seen and rated a number of films from the Japanese animation Studio Ghibli on a film rating web application built on a PaaS cloud. 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 the future, Alice may also change the ratings for any of her previously rated films.

Objectives

In short, 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,
5. the scheme has acceptable efficiency (in terms of speed) as well as a high prediction accuracy.

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 "Problem Statement".

To achieve this, in our approach, the user will assume the presence of anonymising techniques (e.g. IPv4 network address translation and dynamic IPs, anonymiser networks such as Tor, and pseudonymous identities; see: [3-9]) 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.

Contributions

The main contribution of this paper is that we build on our previously proposed work [2] and show that it is feasible to implement on both Amazon and Google

Software-as-a-Service enabling Platform-as-a-Service clouds. We also include discussions on possible extensions of our idea. Note that other existing PPCF schemes do not have any cloud based implementations, hence our results are not comparable with implementation results of such PPCF schemes.

The rest of the paper is organised as follows: in §"Related Work", we list some of the related work in privacy preserving collaborative filtering. In §"Background", we introduce the building blocks for our proposed scheme, which is presented in §"Proposed Scheme". We present the comparative evaluation of our implementation in §"Evaluation" preceded by practical implementation considerations in §"Implementation Considerations" before concluding with future directions in §"Conclusion and Future Work".

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 [10] which uses a partial Singular Value Decomposition (SVD) model and homomorphic encryption to devise a multi-party PPCF scheme. Canny [11] 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: in [12], they proposed a randomized perturbation technique to protect individual privacy while still producing accurate recommendations results; in [13], the authors presented a privacy-preserving protocol for collaborative filtering over vertically partitioned data; in [14], a randomisation based SVD approach is presented, while in [15] the scheme enables recommendations via item-based algorithms using randomized response techniques.

Berkovsky et al. [16] 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 [17].

Cissée and Albayrak [18] 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 [19] a privacy preserving collaborative filtering schemes for web portals using a trusted third party for threshold decryption. Kaleli and Polat propose [20] 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. [21] but the authors also describe that their scheme does not scale well for realistic datasets. Gong et al. [22] present a new collaborative filtering technique based on randomised perturbation and secure multiparty computation. However, neither are these techniques 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 [23], we have proposed a privacy preserving (using encryption) CF scheme based on the well known weighted Slope One predictor [24]. 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. We have also experimented with an actual implementation on the cloud, but since none of the other existing work has tested their implementation on a cloud computing environment, we do not provide any explicit comparative evaluation.

Background

Slope one predictors for collaborative filtering

The Slope One predictors due to Lemire and McLachlan [24] 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:

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

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:

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

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 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.

Privacy threats

The two matrices (deviation and cardinality) 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. However, there is a privacy threat to individual user's rating data at the time of computing deviations and cardinality from plaintext ratings. This is why we ought to de-link the identities of users from the submitted ratings such that individual users and their ratings cannot be conclusively linked to each other. In addition to that, if the user sends his/her rating vector to the prediction function then there is a privacy threat. Therefore, we use encrypted rating prediction.

Anonymity and identifiability

The diagram in Figure 1 illustrates why a simple network address translator can provide a certain degree of anonymity by representing all IP addresses on its LAN side with just one IP address on its WAN side. A more complex and cryptographic pseudonymous method of anonymity was presented in [9].

Despite the importance of anonymity in this privacy preserving collaborative filtering technique, there is also a consensus on identifiability if there is a pressing need. In both the case of NAT and the case of pseudonyms, this identifiability can be obtained through information that the NAT administrator or pseudonymous group membership issuers provide. The process is not straightforward and requires offline intervention by administrators. Hence, the support for identifiability does not nullify the ability to anonymise the data.

Homomorphic cryptosystem

Paillier public-key cryptosystem [25] 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:

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

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

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

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

Algorithm 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.
```

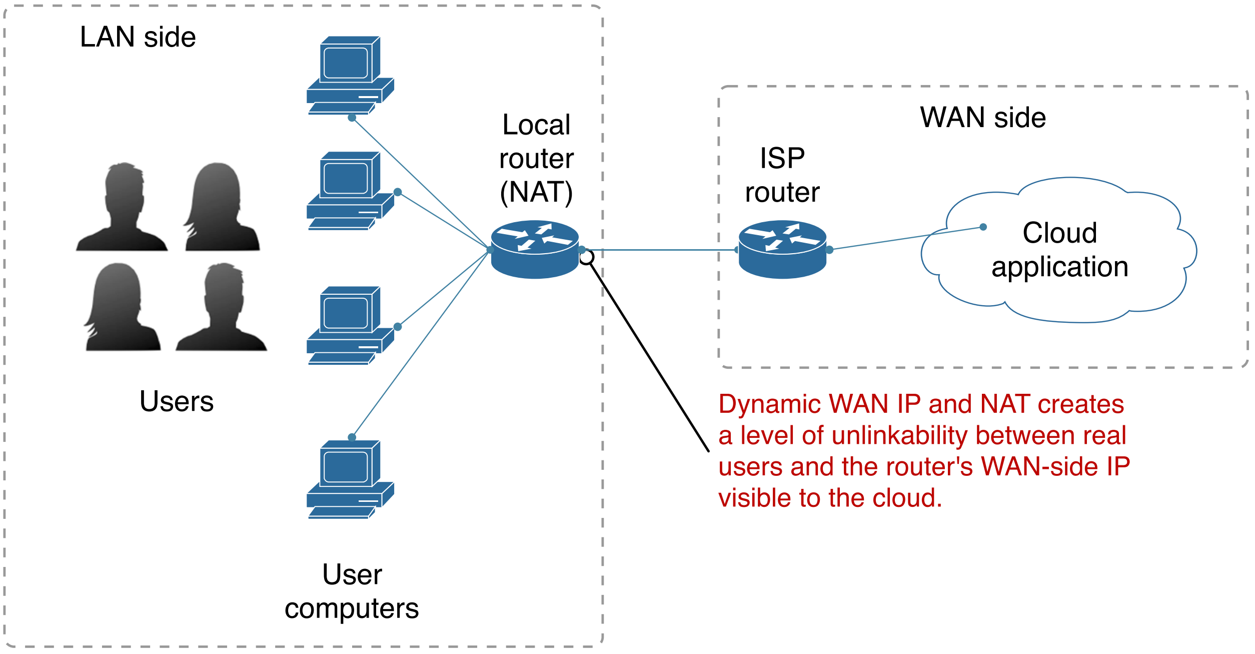

LAN side
Users
User computers
Local router (NAT)
WAN side
ISP router
Cloud application
Dynamic WAN IP and NAT creates a level of unlinkability between real users and the router's WAN-side IP visible to the cloud.

Figure 1 Typical anonymity that can be obtained through a network address translator.

```
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: $gcd(L(g^\lambda \bmod n^2), n) = 1$ where
$L(u) = \frac{u-1}{n}$. {Optimisation:
$g^\lambda \bmod n^2 = (1 + n\lambda) \bmod n^2$.}
5: Pre-compute the modular multiplicative
   inverse $\mu = L(g^\lambda \bmod n^2)^{-1} \bmod n$.
6: return Public key: $(n, n^2, g)$ and private key:
   $(\lambda, \mu)$.
```

Algorithm 2
Paillier encryption algorithm.

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

Algorithm 3
Paillier decryption algorithm.

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

Our implementation of Paillier cryptosystem supports negative numbers for plaintext because quite often a deviation of two ratings can be negative. 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.

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.

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 2 showing the addition of rating data and an example CF prediction query. It illustrates users, Alice and Bob, each submitting plaintext (item) pair-wise deviations of ratings of items through an identity anonymiser to the Software-as-a-Service cloud application. Another user, Carol, then queries and obtains a prediction for an arbitrary item (item k, in the example) with her encrypted query vector. The SaaS cloud application computes the prediction using homomorphic encryption and responds to Carol with an encrypted answer, which only Carol herself can decrypt.

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.

Case of new ratings
In the pre-computation stage, the average deviations of ratings from item a to item b is given in equation 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 4.

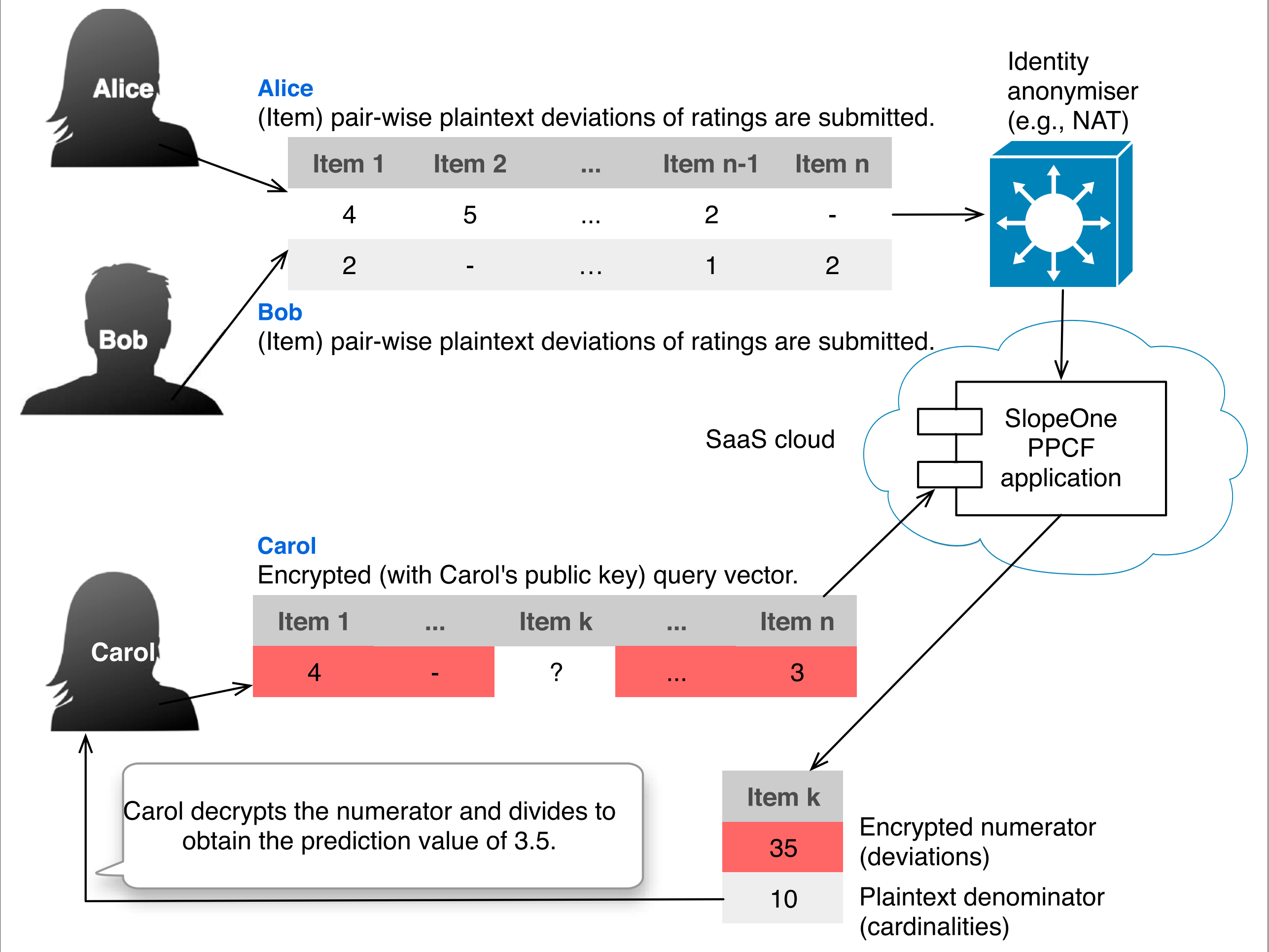

Alice
Alice
(Item) pair-wise plaintext deviations of ratings are submitted.
Item 1 | Item 2 | ... | Item n-1 | Item n
4 | 5 | ... | 2 | -
2 | - | ... | 1 | 2
Bob
Bob
(Item) pair-wise plaintext deviations of ratings are submitted.
Identity anonymiser (e.g., NAT)
SaaS cloud
SlopeOne PPCF application
Carol
Carol
Encrypted (with Carol's public key) query vector.
Item 1 | ... | Item k | ... | Item n
4 | - | ? | ... | 3
Carol decrypts the numerator and divides to obtain the prediction value of 3.5.
Item k
35
10
Encrypted numerator (deviations)
Plaintext denominator (cardinalities)

Figure 2 User-interaction diagram of our scheme.

Algorithm 4
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.}
```

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 5. 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 5.

Algorithm 5
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.}
```

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 the deviation matrix and to apply homomorphic multiplication according to the prediction equation defined in equation 5, 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.

$$r_{u,x} = \frac{\mathcal{D}(\prod_{a|a\neq x}(\mathcal{E}(\Delta_{x,a})(\mathcal{E}(r_{u,a})^{\phi_{x,a}})))}{\sum_{a|a\neq x}\phi_{x,a}}. \tag{5}$$

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

$$r_{u,x} = \frac{\mathcal{D}(\mathcal{E}(\sum_{a|a\neq x}\Delta_{x,a})\prod_{a|a\neq x}(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))}{\sum_{a|a\neq x}\phi_{x,a}}. \tag{6}$$

The steps for the prediction is shown in algorithm 6.

Algorithm 6
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. Note that the only privacy leakage in the prediction stage is the list of items in the query vector. The corresponding ratings can only be decrypted by the user's private key. Further to this, the user is free to use a different key pair for each prediction request such that a single public key cannot be used by an adversary to link the queries together.

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.

Implementation considerations

In this section, we present the various considerations and strategies that we have employed in the implementation of the aforementioned scheme on the Google App Engine for Java and the Amazon Elastic Beanstalk cloud platforms.

Database access and configurability

Cache based datastore access in the Google App Engine

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.

Direct database access in the Amazon Elastic Beanstalk

Having noticed that the Amazon Elastic Beanstalk allows significantly faster access to the database than Google App Engine, we have used direct access to the database instead of using Amazon's distributed cache service – ElastiCache, which will be considered in our future work.

Deployment configurability

From the perspective of configurability, our deployment can be made to run on either the Amazon Elastic Beanstalk or the Google App Engine without changing any code. We achieved this through modular design of the backend data access. We used generic data access techniques and made the data store configuration declarative, which can be changed without affecting the code. In future, we will extend this capability to facilitate access to the distributed cache available to Amazon Elastic Beanstalk.

Efficiency and parallelism

Parallelism in pre-computation

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 [26] 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.

Parallelism in prediction

In the *prediction* stage, we do not use concurrency yet in our current prototype. 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.

Inspecting the original Slope One prediction equation 2, one can see that both the numerator and the denominator on the right hand side of the equation are just sums on their own. Since the actual division is not done on the server, an encrypted numerator and the plaintext denominator is sent to the client as shown in equation 6. This means that the client is free to submit more than one queries with disjoint vectors of items in each query and combine the results to form the final prediction. That is a Map-Reduce style parallelism, which we plan to implement in future prototypes. 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 although Amazon Elastic Beanstalk presents no such execution deadline.

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 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.

Note that the server can also use browser-based cookies to more uniquely identify a particular browser, hence a user. However, we limit our discussion of the attack model to IP addresses only because even with browser cookies, it is possible for the same user to be identified separately, e.g. using more than one browsers; or different users identified by the same browser, e.g. a browser on a publicly shared user account. In addition browser cookies can be selectively blocked and the users can be warned of such cookies, thus making the user identification process less transparent than the one using IPs.

Potential extension to vertical partitions

While we have not explicitly experimented with an implementation of the algorithms for vertical partitions (given in Section "Vertical Partition Across Multiple Organisations"), we now discuss potential implementation and performance. It is important to note that the communication between the applications is independent of the PPCF application residing in the PaaS cloud. The different applications, holding vertical partitions, will need to follow some protocol to exchange randomised values between each other. From the perspective of the PPCF application, the performance should be largely unaffected by the process because the addition of the different randomised parts can happen in parallel. Once submitted, those parts are used to update the internal deviation matrix. The query stage is completely unaffected by this process, and so is the time performance of the query. The only scope of a performance slowdown depends on how fast the individual applications collaborate with each other to generate the partial randomised deviations.

Evaluation

We evaluated the aforementioned scheme with an implementation on the Google App Engine for Java (GAE/J) and another on the Amazon Web Services Elastic Beanstalk (AWS/EBS). The two implementations are very similar and actually their deployments on the two cloud platforms are simply controlled by configuration files that specify the datastore each should look at and whether or not to use the distributed in-memory cache. These two were the only factors that differentiated the two implementations.

The prototype implementations are available at the following locations: the *GAE/J demo at* http://gaejppcf.appspot.com/ and the *AWS/EBS demo at* http://gaejppcf.elasticbeanstalk.com/. We usually keep the backend database for the AWS/EBS demo switched off, so this demo will only work fully during certain conferences where we present this work. For the AWS/EBS demo, we have used t1.micro EC2 instances.

For the sake of simplicity, we have not considered the case where there may be more than one applications on the SaaS applications which interact with each other to make a prediction, i.e. the scenario described in § "Vertical Partition Across Multiple Organisations". Also, for test use, we have not implemented comprehensive error-checking on the web front-end. One will notice that while the web application provides means to perform cryptographic operations, e.g. key generation, encryption and decryption on the GAE/J and the AWS/EBS. These are provided only for experimental purposes, and in practice they should be performed at the client's end. For browser compatibility, we have used Google Web Toolkit[b] to code the web-based user interface.

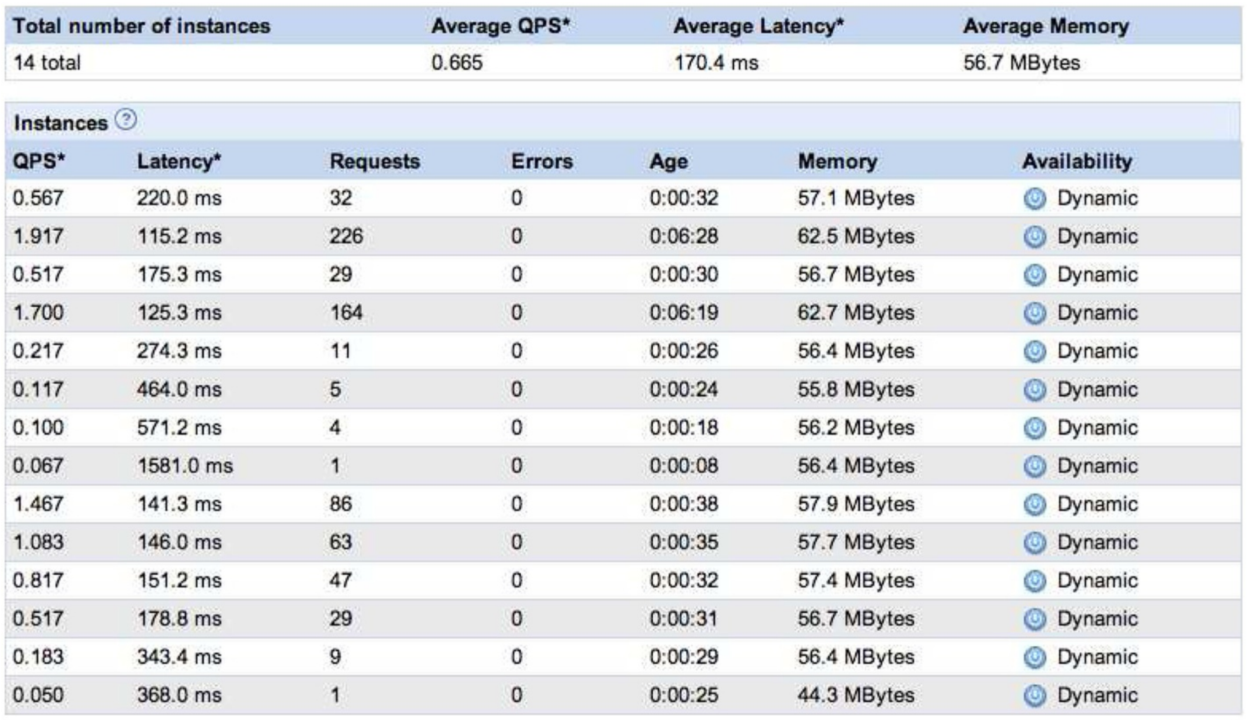
Conforming to Google App Engine terminology, we will call the time taken by the application to respond to the user request as *application latency* or simply *latency*. This latency does not include network latencies encountered between our network and Google and Amazon data centres.

Pre-computation

In the pre-computation stage, there is no cryptographic operation on the cloud.

In the case of Google App Engine, 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 3 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[c] dataset. The figure shows data of the 14 automatically allocated application instances[d]. 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 an instance received while the latency is the average time taken by that instance to complete one such request. This simulates the scenario if many users concurrently submit many requests to add rating or deviation data.

In the case of the Amazon Elastic Beanstalk, we adopted a very different approach to loading the complete MovieLens 100K dataset. We had direct access to port 3306 of the MySQL server EC2 instance running as part of Amazon Relational Database Service (RDS). Consequently, we pre-computed the deviation and cardinality matrices from



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

Figure 3 Typical load during pre-computation on the Google App Engine instances during partial bulk addition of the MovieLens 100K dataset. Note that even if the latency, for example, in the third row is only 175.3ms, it does not equate to a QPS value of $\frac{1}{175.3} = 0.0057$. The QPS signifies the number of queries the particular instance receives on an average while the latency signifies how long the instance takes on an average to process one such query. The fact that the latency is low does not imply that the QPS will be high because the QPS is the result of the queries sent to the front-end, the speed of which is affected by external factors, such as network latency, GAE/J queue processing speed and so on.

the 100K dataset user-item rating matrix and then used plain SQL to add that data. The process was faster and more reliable than adding data to Google App Engine's datastore.

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 Tables 1 and 2. Since the recent version of GAE/J allows front-end instance class configuration, we have also included prediction test results with the 1200MHz, 256MB RAM and the 2400MHz and 512MB RAM front-end instance classes in Tables 3 and 4 respectively. Note that the results presented in Table 1 were obtained using the previous version of the GAE/J which did not allow front-end instance class configuration. The front-end instance class from the previous version roughly compares to the 600MHz, 128MB RAM default instance class in terms of performance.

In terms of prediction performance, the less time it takes, the better. In a realistic example, the prediction of several items ought to be obtained with the user waiting to see some meaningful recommendations. While the prediction may be obtainable in parallel and reported asynchronously to the user (which falls in the remits of future work), users will still expect meaningful recommendations presented to them in reasonably quickly, which is often in order of just few seconds on a web browser or a mobile application.

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 on the Google App Engine 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, although in our Amazon implementation we did not take advantage of Amazon's in-memory cache – the ElastiCache. In addition, GAE/J instances may also perform better or worse depending on the shared resources availability on the Google's cloud computing clusters. In contrast with the Google App Engine, the results on the Amazon Elastic Beanstalk are significantly faster, with almost a 5.5 fold performance increase for a 2048-bit cryptosystem and query vector size 10.

In addition, we note that the t1.micro instance on the AWS/EBS still outperforms the GAE/J 2400MHz front-end instance class when using the 2048-bit cryptosystem. This is explicable from the description of Amazon's micro instances[e], which shows that not only do t1.micro instances have more memory (at 600MB) than the GAE/J but also have more powerful CPU available in short bursts. In fact, the micro instances are particularly suitable for applications requiring occasional short bursts of CPU, which is what our experiments do. Figure 4 presents a performance comparison of the various GAE/J instance classes as well as the t1.micro of AWS/EBS. The diagram shows each class has four different performance values. The leftmost is the one for 1024 bits with a query vector size of 5; followed by that with the query vector size of 10; and then followed by that with the query vector size of 5 for the 2048 bit cryptosystem and the rightmost using the 2048 bits cryptosystem with the query vector size 10.

Table 1 Comparison of typical prediction timings, based on the optimised equation 6 on the Google App Engine for Java

Bit size[1]	Query vector size[2]	Typical prediction time
1024	5	500ms
1024	10	650ms
2048	5	3800ms
2048	10	5000ms

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

Table 2 Comparison of typical prediction timings, based on the optimised equation 6 on the Amazon Web Services Elastic Beanstalk using the t1.micro instance

Bit size[1]	Query vector size[2]	Typical prediction time
1024	5	300ms
1024	10	550ms
2048	5	600ms
2048	10	900ms

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

Table 3 Comparison of typical prediction timings, based on the optimised equation 6 on the Google App Engine for Java with the 1200MHz front-end instance class

Bit size[1]	Query vector size[2]	Typical prediction time
1024	5	450ms
1024	10	600ms
2048	5	2500ms
2048	10	3100ms

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

Table 4 Comparison of typical prediction timings, based on the optimised equation 6 on the Google App Engine for Java with the 2400MHz front-end instance class

Bit size[1]	Query vector size[2]	Typical prediction time
1024	5	285ms
1024	10	400ms
2048	5	1200ms
2048	10	1750ms

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

Security

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

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 [27] to require human intervention, and hence slow down the number of additions, updates and deletions.

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 utilised the well-known weighted Slope One collaborative filtering predictor to demonstrate a novel approach for privacy preserving collaborative filtering and its practical applicability on two public Software-as-a-Service enabling Platform-as-a-Service clouds: the Google App Engine for Java and the Amazon Elastic Beanstalk. 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, with results on the Amazon Elastic Beanstalk outperforming those on the Google App Engine.

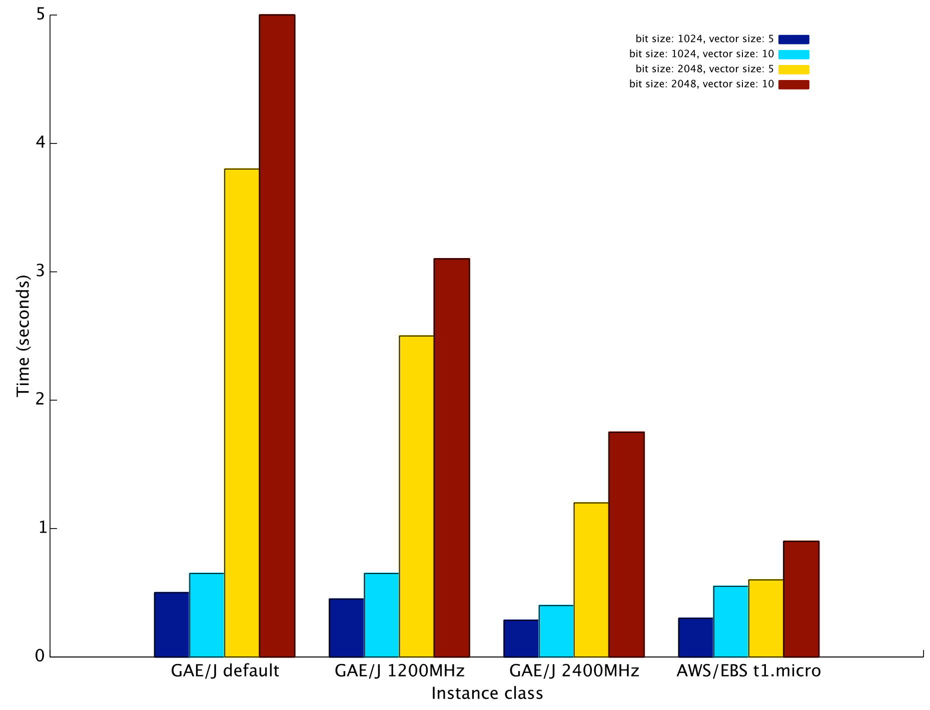


Figure 4 **Prediction performance comparison of the different instance classes of the GAE/J and the t1.micro class of AWS/EBS.**

The scheme proposed in this paper partly relies on the security guarantees of existing anonymising techniques, such as NAT, pseudonyms or an anonymiser/mix network. We discuss the relatively simple NAT-based anonymity but also mention that it is possible to obtain identifiability if need be. 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 rating query performance further by optimising our implementation.

Endnotes

[a] This is an extended journal version of our prior work "Privacy-preserving collaborative filtering for the cloud" published in the IEEE Cloudcom 2011.
[b] See: http://code.google.com/webtoolkit/.
[c] MovieLens datasets: http://www.grouplens.org/node/73.
[d] This is a snapshot of the GAE/J control panel. The number of instances change over time depending on the number of user requests.
[e] See http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/concepts_micro_instances.html.

Competing interests
The authors declare that they have no competing interests.

Authors' contributions
AB conceptualised the model and implemented it on the SaaS enabling PaaS cloud platforms. He also contributed to all of the other sections and took the lead on writing the article. JV primarily contributed to the conceptualisation of the paper, the analysis of privacy of the proposed scheme, and the related work. HK contributed to optimising the cryptographic operations in the prediction stage. TD and SN contributed to the introduction and to the overall writing of the paper. All authors read and approved the final manuscript.

Authors' information

Anirban Basu
Post-doctoral researcher (Department of Communication and Network Engineering), Tokai University
Dr. Anirban Basu is a Post-doctoral Researcher at Kikuchi lab at Tokai University working on a Japanese Ministry of Internal Affairs and Communications funded project in collaboration with Waseda University, Hitachi, NEC and KDDI. He holds a Ph.D. in Computer Science and a Bachelor of Engineering (Hons.) in Computer Systems Engineering from the University of Sussex. His research interests are in computational trust management, privacy and security and peer-to-peer networks. He has several years of experience with academic research at the University of Sussex where he was involved with two EPSRC funded and one EU IST FP5 funded research projects alongside his doctoral research.

Jaideep Vaidya
Associate Professor (Management Science and Information Systems Department), Rutgers The State University of New Jersey
Dr. Jaideep Vaidya is an Associate Professor of Computer Information Systems at Rutgers University. He received his Masters and Ph.D. in Computer Science from Purdue University and his Bachelors degree in Computer Engineering from the University of Mumbai. His research interests are in Privacy, Security, Data Mining, and Data Management. He has published over 60 papers in international conferences and archival journals, and has received three best paper awards from the premier conferences in data mining, databases, and digital government research. He is also the recipient of a NSF Career Award and a Rutgers Board of Trustees Research Fellowship for Scholarly Excellence.

Hiroaki Kikuchi
Professor (Department of Communication and Network Engineering), Tokai University
Dr. Hiroaki Kikuchi received B.E., M.E. and Ph.D. degrees from Meiji University. He is currently a Professor in the Department of Communication and Network Engineering, School of Information and Telecommunication Engineering, Tokai University. He was a visiting researcher of the School of Computer Science, Carnegie Mellon University in 1997. His main research interests are fuzzy logic, cryptographic protocols, and network security. He is a member of the Institute of Electronics, Information and Communication Engineers of Japan (IEICE), the Information Processing Society of Japan (IPSJ), the Japan Society for Fuzzy Theory and Systems (SOFT), IEEE and ACM. He is an IPSJ Fellow.

Theo Dimitrakos
Head of Security Architectures Research, BT Research and Technology
Dr. Theo Dimitrakos the leading the Security Architectures Research area at the Security Futures Practice of BT Research & Technology (part of BT Innovate & Design). He has over fifteen years of experience in Information Security, Identity and Entitlement Management, SOA Web Services, Grid and Cloud Computing and, more recently, Cyber Security including Intrusion Prevention Systems, Anti-Evasion Analysis and Malware Detection / Containment for Virtual Data Centres and large scale Open Systems. Theo has been the technical director of several European research initiatives including large scale innovation delivery projects such as BEinGRID and TrustCoM, and more recently OPTIMIS as well as research community building initiatives such as iTrust. He has contributed to or authored several expert advisory reports on Protection against Cyber-crime, on Virtualisation and Cloud Security Innovations, on Data Protection and Privacy in the Cloud, on Cloud Computing Risks and on Security and Resilience of Cloud Computing Infrastructure and Services for Government use by BT, ENISA and other European and UK agencies or expert groups. Some of these have been discussed at the WEF in Davos and the European Parliament. Theo has also been offering consultancy steering the product development roadmaps of innovative start-ups in Europe and North America. Theo holds a Ph.D. in Computing from Imperial College, London; and a B.Sc. in Mathematics from Panepistimio Kritis.

Srijith Nair
Senior Security Researcher, BT Research and Technology
Dr. Srijith Nair is a Senior Security Researcher at BT Innovate and Design and is currently looking at security issues related to virtualisation and the cloud computing delivery model and other security issues involving SOA/SOI themes. He has been part of multiple expert groups advising European Network and Information Security Agency (ENISA) on the impact of cloud computing on the resilience of eGov services as well as on cloud computing security assessment and was also a part of the working group of the Cloud Security Alliance (CSA) delivering the security guidance report. He has a Ph.D. in computer science from Vrije Universiteit, Amsterdam where he worked with Prof. Andrew S. Tanenbaum and Associate Prof. Bruno Crispo on the problem of intrasystem information flow control within the scope of data-centric policy enforcement. He also worked on other aspects of security, including applied cryptography, DRM, and system security in general. Srijith received his M.Sc. (Computer Science) degree from National University of Singapore, Singapore in 2002 and a B.Tech. (EEE) degree (First class honors) from Nanyang Technological University, Singapore in 2000, under the SIA/NOL Undergraduate Scholarship Program.

Acknowledgements
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 and Srijith Nair 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.

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

Received: 6 February 2012 Accepted: 5 June 2012
Published: 9 July 2012

References

1. Schafer JB, Konstan J, Riedi J (1999) Recommender systems in e-commerce. In Proceedings of the 1st ACM conference on Electronic Commerce 158–166. New York: ACM Press
2. Basu A, Vaidya J, Kikuchi H, Dimitrakos T (2011) Privacy-preserving collaborative filtering for the cloud. In Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (Cloudcom), Athens, Greece
3. Reed M, Syverson P, Goldschlag D (1998) Anonymous connections and onion routing. Sel Areas Commun 16(4): 482–494
4. Catalano D, Di Raimondo M, Fiore D, Gennaro R, Puglisi O (2011) Fully non-interactive onion routing with forward-secrecy. In Proceedings of the 9th International conference on Applied cryptography and network security. ACNS'11. Nerja, Spain 255–273. Berlin, Heidelberg: Springer-Verlag
5. Dingledine R, Mathewson N, Syverson P (2004) Tor: The second-generation onion router. In Proceedings of the 13th conference on USENIX Security Symposium-Volume 13. SSYM'04, San Diego, CA 21–21. Berkeley, CA, USA: USENIX Association
6. Clarke I, Sandberg O, Wiley B, Hong T (2001) Freenet: A distributed anonymous information storage and retrieval system. In International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability 46–66. New York, NY, USA: Springer-Verlag New York, Inc.
7. Furukawa J, Sako K (2010) Mix-net system. [US Patent Application. 20,100/115,285]
8. Danezis G (2003) Mix-networks with restricted routes. In Proceedings of Privacy Enhancing Technologies workshop (PET 2003) 1–17. Dresden, Germany: Springer-Verlag
9. Wakeman I, Chalmers D, Fry M (2007) 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. MPAC '07. Newport Beach, California 7–12. New York, NY, USA: ACM
10. Canny J (2002) Collaborative filtering with privacy. In Proceedings of the 2002 IEEE Symposium on, Security and Privacy. SP '02. Oakland, CA, USA 45–57. Washington, DC, USA: IEEE Computer Society
11. Canny J (2002) Collaborative filtering with privacy via factor analysis. In Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, *SIGIR '02* 238–245. New York: ACM. [http://doi.acm.org/10.1145/564376.564419]
12. Polat H, Du W (2003) Privacy-preserving collaborative filtering using randomized perturbation techniques. In Data Mining, 2003. ICDM 2003. Third IEEE International, Conference on 625–628. IEEE
13. Polat H, Du W (2005) Privacy-preserving collaborative filtering on vertically partitioned data. Knowledge Discovery in Databases: PKDD 2005. pp. 651–658
14. Polat H, Du W (2005) SVD-based collaborative filtering with privacy. In Proceedings of the 20th ACM Symposium on Applied Computin
15. Polat H, Du W (2006) Achieving private recommendations using randomized response techniques. In Proceedings of the 10th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining. PAKDD'06. Singapore 637–646. Berlin, Heidelberg: Springer-Verlag
16. Berkovsky S, Eytani Y, Kuflik T, Ricci F (2007) Enhancing privacy and preserving accuracy of a distributed collaborative filtering. In Proceedings of the 2007 ACM conference on, Recommender systems. RecSys '07. Minneapolis, MN, USA 9–16. New York, NY, USA: ACM
17. Tada M, Kikuchi H, Puntheeranurak S (2010) Privacy-Preserving Collaborative Filtering Protocol Based on Similarity between Items. In Proceedings of the 2010 24th IEEE International, Conference on Advanced Information Networking and Applications. AINA '10. Perth, Australia 573–578. Washington, DC, USA: IEEE Computer Society
18. Cissée R, Albayrak S (2007) An agent-based approach for privacy-preserving recommender systems. In Proceedings of the 6th international joint conference on autonomous agents and multiagent systems. AAMAS '07. Honolulu, Hawaii 1–8. New York, NY, USA: ACM
19. Ahmad W, Khokhar A (2007) An Architecture for Privacy Preserving Collaborative Filtering on Web Portals. In Proceedings of the Third International Symposium on Information Assurance and Security. IAS '07. Manchester, UK 273–278. Washington, DC, USA: IEEE Computer Society
20. Kaleli C, Polat H (2010) P2P collaborative filtering with privacy. Turkish J Electr Electrical Eng Comput Sci 8: 101–116
21. Han S, Ng WK, Yu PS (2009) Privacy-Preserving Singular Value Decomposition. In Proceedings of the 2009 IEEE International Conference on, Data Engineering. ICDE '09. Shanghai, China 1267–1270. Washington, DC, USA: IEEE Computer Society
22. Gong S (2011) Privacy-preserving collaborative filtering based on randomized perturbation techniques and secure multiparty computation. IJACT: Int J Advancements Comput Technol 3(4): 89–99
23. Basu A, Kikuchi H, Vaidya J (2011) 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
24. Lemire D, Maclachlan A (2005) Slope one predictors for online rating-based collaborative filtering. Soc Ind Mathematics 5: 471–480
25. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In Proceedings of the 17th international conference on Theory and application of cryptographic techniques, Volume 1592, EUROCRYPT'99 223–238. Berlin, Heidelberg: Springer-Verlag
26. Dean J, Ghemawat S (2008) MapReduce: Simplified data processing on large clusters. Commun ACM 51: 107–113
27. Von Ahn L, Blum M, Hopper N, Langford J (2003) CAPTCHA: Using hard AI problems for security. In Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques. EUROCRYPT'03 294–311. Berlin, Heidelberg: Springer-Verlag

doi:10.1186/2192-113X-1-8
Cite this article as: Basu *et al.*: Privacy preserving collaborative filtering for SaaS enabling PaaS clouds. *Journal of Cloud Computing: Advances, Systems and Applications* 2012 **1**:8.