Privacy-preserving collaborative filtering for the cloud

Anirban Basu[1] Jaideep Vaidya[2] Hiroaki Kikuchi[1]
Theo Dimitrakos[3]

[1]Graduate School of Engineering, Tokai University, Japan

[2]MSIS Department, Rutgers The State University of New Jersey, USA

[3]Research & Technology, British Telecom, UK

IEEE Cloudcom 2011, Athens, Greece

Outline

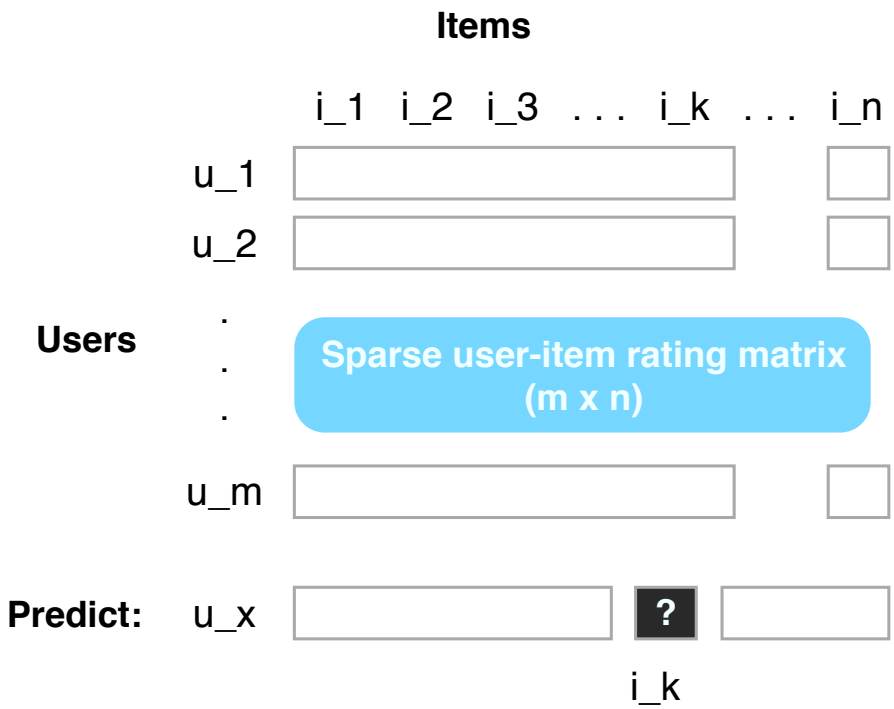
Recommendation and CF

- A recommendation example: Amazon's "people who buy **this** also buy **that**" (user profile analysis).
- Rating-based collaborative filtering (CF) – another mechanism for recommendation.

Recommendation and CF

- A recommendation example: Amazon's "people who buy **this** also buy **that**" (user profile analysis).
- Rating-based collaborative filtering (CF) – another mechanism for recommendation.

Recommendation and CF



- The task is to predict the rating user u_x will give to item i_k given the sparse user-item rating matrix.

CF: an illustrative example

An airlines example ("-" implies absence of ratings):

	Virgin Atlantic	Emirates	Singapore Airlines
Alice	3	?	5
Bob	3	4	5
Tracy	-	2	4
Steve	3	3	-

- Predict: how would Alice rate Emirates?

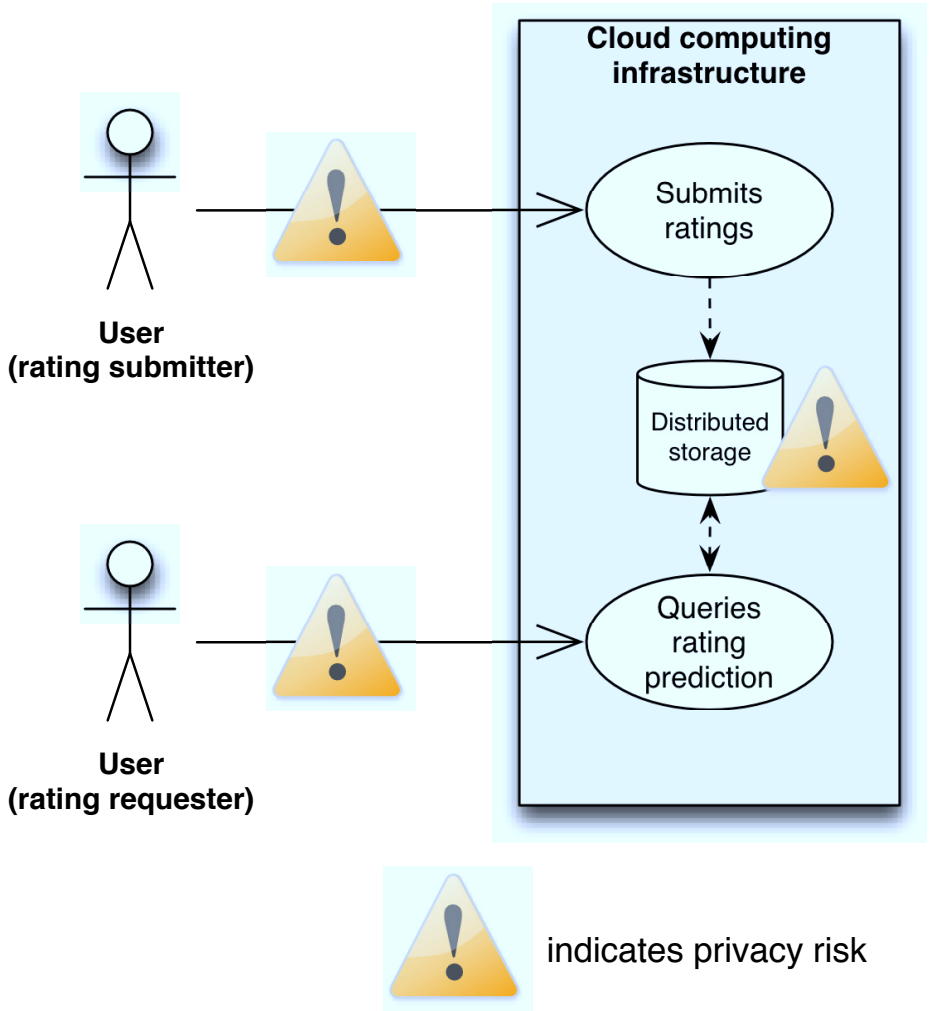
CF on the cloud – privacy risks

- Recommendation providers may run on cloud computing infrastructures.
- Your private rating data may not be safe on the cloud because of insider and outsider threats.

CF on the cloud – privacy risks

- Recommendation providers may run on cloud computing infrastructures.
- Your private rating data may not be safe on the cloud because of insider and outsider threats.

These are where privacy concerns are raised



Research problem: privacy preserving CF

Compute a privacy preserving rating prediction on a Software-as-a-Service (SaaS) construction Platform-as-a-Service (PaaS) cloud, such that we are:

- able to hide and/or delink user's private rating data without using any trusted third party,
- and it is robust to insider threats from the cloud,
- while assuming honest-but-curious user,
- and assuming identity concealing network infrastructures (e.g. anonymous networks, pseudonyms, IPv4 NAT).

Research problem: privacy preserving CF

Compute a privacy preserving rating prediction on a Software-as-a-Service (SaaS) construction Platform-as-a-Service (PaaS) cloud, such that we are:

- able to hide and/or delink user's private rating data without using any trusted third party,
- and it is robust to insider threats from the cloud,
- while assuming honest-but-curious user,
- and assuming identity concealing network infrastructures (e.g. anonymous networks, pseudonyms, IPv4 NAT).

Research problem: privacy preserving CF

Compute a privacy preserving rating prediction on a Software-as-a-Service (SaaS) construction Platform-as-a-Service (PaaS) cloud, such that we are:

- able to hide and/or delink user's private rating data without using any trusted third party,
- and it is robust to insider threats from the cloud,
- while assuming honest-but-curious user,
- and assuming identity concealing network infrastructures (e.g. anonymous networks, pseudonyms, IPv4 NAT).

Research problem: privacy preserving CF

Compute a privacy preserving rating prediction on a Software-as-a-Service (SaaS) construction Platform-as-a-Service (PaaS) cloud, such that we are:

- able to hide and/or delink user's private rating data without using any trusted third party,
- and it is robust to insider threats from the cloud,
- while assuming honest-but-curious user,
- and assuming identity concealing network infrastructures (e.g. anonymous networks, pseudonyms, IPv4 NAT).

Our contributions

- A privacy preserving CF solution for the Google App Engine for Java (GAE/J)[1] – a specialised SaaS construction PaaS cloud.
- Can be extended to vertical partitions[2].
- Feasible on a real world public PaaS cloud.

[1] http://code.google.com/appengine/

[2] See § IV.C in the paper. Left out of this presentation for simplicity.

Our contributions

- A privacy preserving CF solution for the Google App Engine for Java (GAE/J)[1] – a specialised SaaS construction PaaS cloud.
- Can be extended to vertical partitions[2].
- Feasible on a real world public PaaS cloud.

[1] http://code.google.com/appengine/

[2] See § IV.C in the paper. Left out of this presentation for simplicity.

Our contributions

- A privacy preserving CF solution for the Google App Engine for Java (GAE/J)[1] – a specialised SaaS construction PaaS cloud.
- Can be extended to vertical partitions[2].
- Feasible on a real world public PaaS cloud.

[1] http://code.google.com/appengine/

[2] See § IV.C in the paper. Left out of this presentation for simplicity.

Outline

Types of collaborative filtering

CF can be:

- either memory based using similarity or deviations between users (user-based) or items (item-based);
- or model based, such as utilising the singular value decomposition technique.

Types of collaborative filtering

CF can be:

- either memory based using similarity or deviations between users (user-based) or items (item-based);
- or model based, such as utilising the singular value decomposition technique.

Privacy-preserving CF

Classified, as per mechanism:

- Encryption based – where privacy of data is preserved through homomorphic encryption [Canny2002a, Han2009].
- Randomisation based – where privacy of data is preserved through random perturbation of the data or by anonymising identities [Polat2003, 2005 and 2006].

Privacy-preserving CF

Classified, as per mechanism:

- Encryption based – where privacy of data is preserved through homomorphic encryption [Canny2002a, Han2009].
- Randomisation based – where privacy of data is preserved through random perturbation of the data or by anonymising identities [Polat2003, 2005 and 2006].

Privacy-preserving CF

Classified, as per mechanism:

- Encryption based – where privacy of data is preserved through homomorphic encryption [Canny2002a, Han2009].
- Randomisation based – where privacy of data is preserved through random perturbation of the data or by anonymising identities [Polat2003, 2005 and 2006].

Classified, as per infrastructure, PPCF can be:

- single machine or single cluster based [Tada2010, Basu2011], or
- large-scale distributed [Berkovsky2007, Canny2002b].

Privacy-preserving CF

Classified, as per mechanism:

- Encryption based – where privacy of data is preserved through homomorphic encryption [Canny2002a, Han2009].
- Randomisation based – where privacy of data is preserved through random perturbation of the data or by anonymising identities [Polat2003, 2005 and 2006].

Classified, as per infrastructure, PPCF can be:

- single machine or single cluster based [Tada2010, Basu2011], or
- large-scale distributed [Berkovsky2007, Canny2002b].

I will not bore you with bibliography slides at the end. . .

Please see the the paper for detailed references of the cited work.

Outline

What is Slope One?

The original paper on SlopeOne CF:

Lemire, D., Maclachlan, A. 2005. *Slope one predictors for online rating-based collaborative filtering*. In: Society for Industrial Mathematics.

What is Slope One?

- Collaborative filtering (CF) predictors of the form $f(x) = x + b$, hence "slope one".
- Weighted version is based on pre-computed average deviations between ratings of items, weighted by relative cardinalities of pairs of items.
- Accurate, fast and incrementally updatable.

What is Slope One?

- Collaborative filtering (CF) predictors of the form $f(x) = x + b$, hence "slope one".
- Weighted version is based on pre-computed average deviations between ratings of items, weighted by relative cardinalities of pairs of items.
- Accurate, fast and incrementally updatable.

What is Slope One?

- Collaborative filtering (CF) predictors of the form $f(x) = x + b$, hence "slope one".
- Weighted version is based on pre-computed average deviations between ratings of items, weighted by relative cardinalities of pairs of items.
- Accurate, fast and incrementally updatable.

Why Slope One?

The choice of the CF scheme has effect on performance and privacy on the cloud.

- Traditional user-based or item-based CF requires **storage of private rating data**; easy to update but **slow to query**.
- Low-rank matrix approximations (e.g. SVD) are **difficult to compute incrementally**; otherwise **slow to update** from **stored private rating data** but fast to query.
- Slope One uses an incrementally updatable item-item matrix model; fast to update and fast to query.

Why Slope One?

The choice of the CF scheme has effect on performance and privacy on the cloud.

- Traditional user-based or item-based CF requires **storage of private rating data**; easy to update but **slow to query**.
- Low-rank matrix approximations (e.g. SVD) are **difficult to compute incrementally**; otherwise **slow to update** from **stored private rating data** but fast to query.
- Slope One uses an incrementally updatable item-item matrix model; fast to update and fast to query.

Why Slope One?

The choice of the CF scheme has effect on performance and privacy on the cloud.

- Traditional user-based or item-based CF requires **storage of private rating data**; easy to update but **slow to query**.
- Low-rank matrix approximations (e.g. SVD) are **difficult to compute incrementally**; otherwise **slow to update** from **stored private rating data** but fast to query.
- Slope One uses an incrementally updatable item-item matrix model; fast to update and fast to query.

Outline

The weighted Slope One

- The average deviations of ratings from item a to item b is given as:

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

  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.

- Thus, the rating for user u and item x using the *weighted* Slope One is predicted as:

$$r_{u,x} = \frac{\sum_{a|a \neq x} (\overline{\delta_{x,a}} + r_{u,a}) \phi_{x,a}}{\sum_{a|a \neq x} \phi_{x,a}} = \frac{\sum_{a|a \neq x} (\Delta_{x,a} + r_{u,a} \phi_{x,a})}{\sum_{a|a \neq x} \phi_{x,a}} \quad (2)$$

The weighted Slope One

- The average deviations of ratings from item a to item b is given as:

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

  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.
- Thus, the rating for user u and item x using the *weighted* Slope One is predicted as:

$$r_{u,x} = \frac{\sum_{a|a \neq x} (\overline{\delta_{x,a}} + r_{u,a}) \phi_{x,a}}{\sum_{a|a \neq x} \phi_{x,a}} = \frac{\sum_{a|a \neq x} (\Delta_{x,a} + r_{u,a} \phi_{x,a})}{\sum_{a|a \neq x} \phi_{x,a}} \quad (2)$$

Pre-computed incrementally updatable matrices

Weighted Slope One predictor has the following two pre-computed, incrementally updatable matrices.

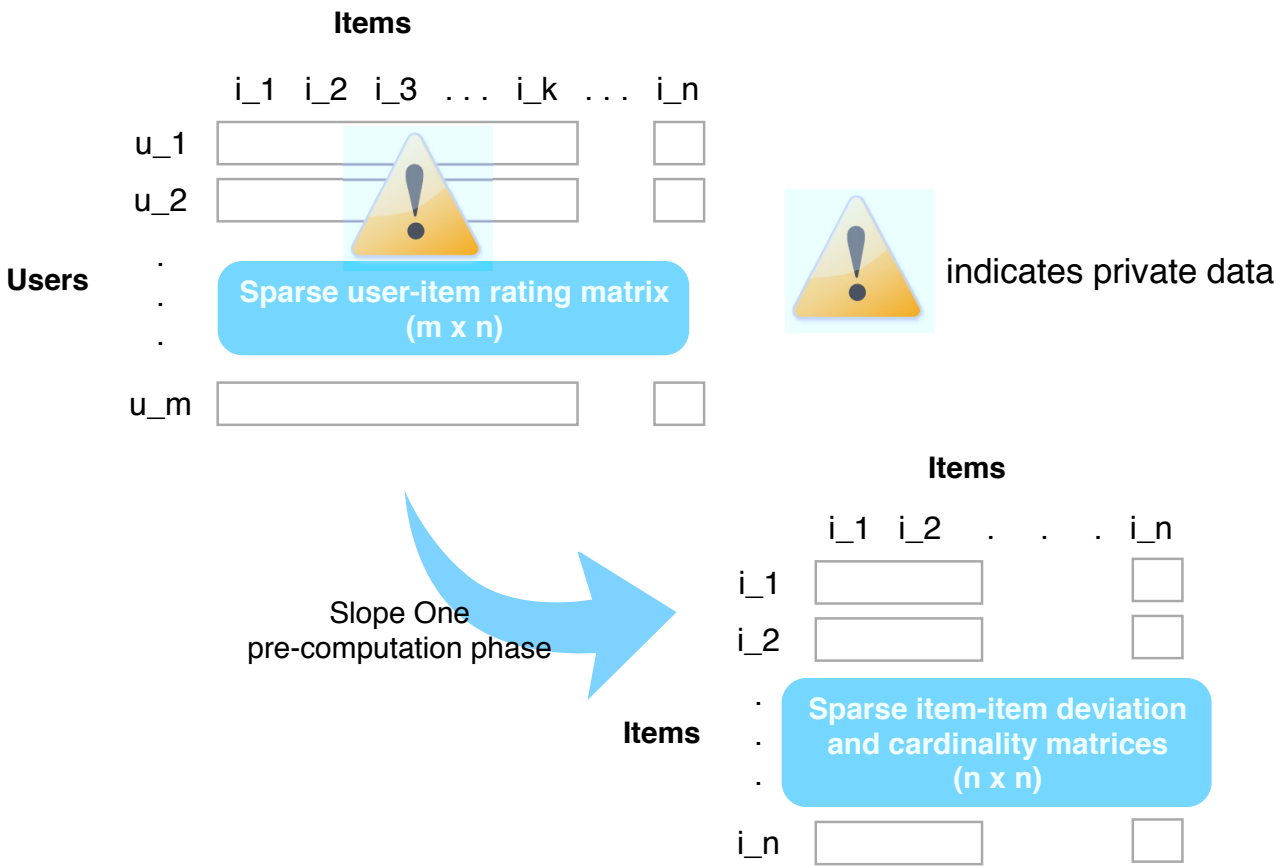
- ***Deviation matrix*** or Δ: each element is the total deviation of ratings between a pair of items, calculated over cases where both items have been rated by the same user. If the ratings matrix is of dimension *mxn* (i.e. *n* items) then Δ is of dimension *nxn*.
- ***Cardinality matrix*** or ϕ: each element is the count of the cases where items in a pair have been both rated by the same user. It is of the same dimension as Δ.

Pre-computed incrementally updatable matrices

Weighted Slope One predictor has the following two pre-computed, incrementally updatable matrices.

- ***Deviation matrix*** or Δ: each element is the total deviation of ratings between a pair of items, calculated over cases where both items have been rated by the same user. If the ratings matrix is of dimension *mxn* (i.e. *n* items) then Δ is of dimension *nxn*.
- ***Cardinality matrix*** or ϕ: each element is the count of the cases where items in a pair have been both rated by the same user. It is of the same dimension as Δ.

Pre-computed incrementally updatable matrices



Outline

Additively homomorphic Paillier cryptosystem

- homomorphic addition:

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

- homomorphic multiplication:

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

We denote encryption and decryption functions as $\mathcal{E}()$ and $\mathcal{D}()$ respectively with plaintext messages m_1, m_2 and integer multiplicand π.

Additively homomorphic Paillier cryptosystem

- homomorphic addition:

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

- homomorphic multiplication:

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

We denote encryption and decryption functions as $\mathcal{E}()$ and $\mathcal{D}()$ respectively with plaintext messages m_1, m_2 and integer multiplicand π.

Encrypted prediction query

Based on the previous equation for plaintext Slope One predictors, we can write:

$$\sum_{a|a\neq x}(\Delta_{x,a}+r_{u,a}\phi_{x,a})=\mathcal{D}(\prod_{a|a\neq x}(\mathcal{E}(\Delta_{x,a})(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))) \quad (3)$$

and reducing the number of encryptions, the final prediction is given as:

$$r_{u,x}=\frac{\mathcal{D}(\mathcal{E}(\sum_{a|a\neq x}\Delta_{x,a})\prod_{a|a\neq x}(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))}{\sum_{a|a\neq x}\phi_{x,a}} \quad (4)$$

Encrypted prediction query

Based on the previous equation for plaintext Slope One predictors, we can write:

$$\sum_{a|a\neq x}(\Delta_{x,a}+r_{u,a}\phi_{x,a})=\mathcal{D}(\prod_{a|a\neq x}(\mathcal{E}(\Delta_{x,a})(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))) \quad (3)$$

and reducing the number of encryptions, the final prediction is given as:

$$r_{u,x}=\frac{\mathcal{D}(\mathcal{E}(\sum_{a|a\neq x}\Delta_{x,a})\prod_{a|a\neq x}(\mathcal{E}(r_{u,a})^{\phi_{x,a}}))}{\sum_{a|a\neq x}\phi_{x,a}} \quad (4)$$

Privacy preserving Slope One

- Since Δ and ϕ are not private information with respect to user data, these are stored unencrypted in the cloud.
- These matrices are updated as ratings of items are added, updated or deleted in pairs.
- Proposed solution uses **user-encrypted** prediction query and response.

Privacy preserving Slope One

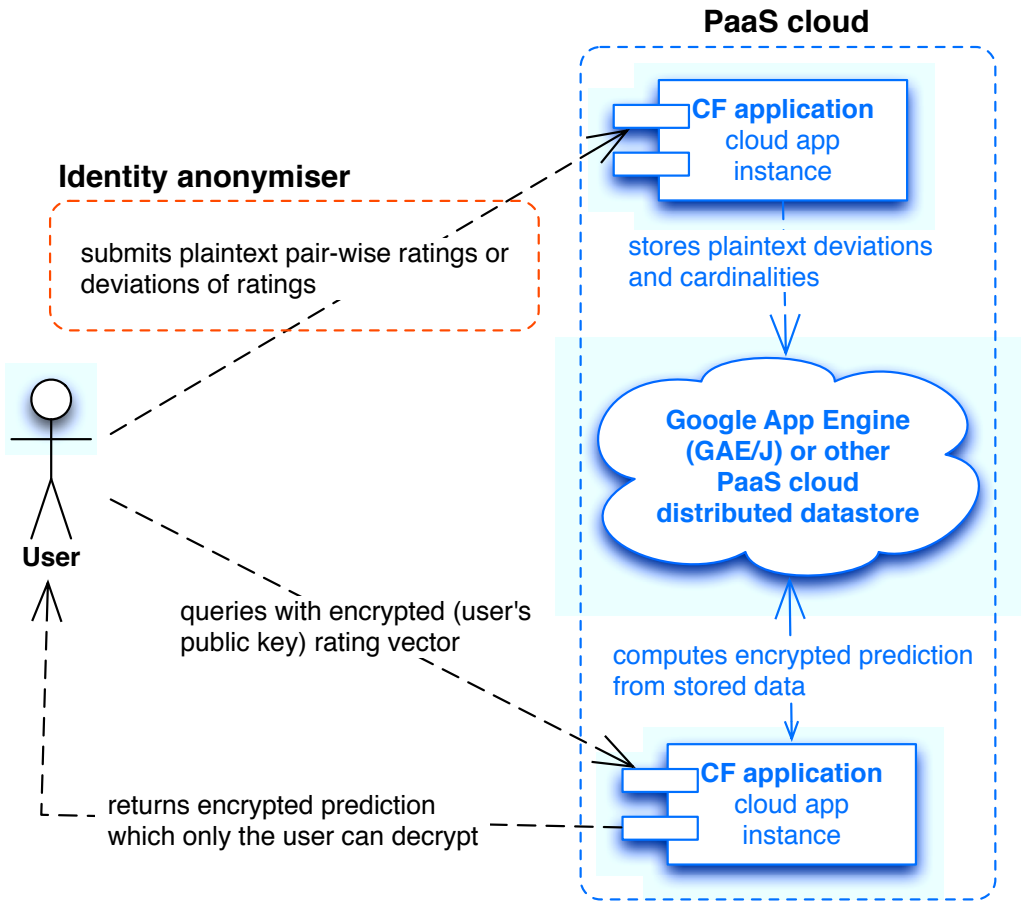
- Since Δ and ϕ are not private information with respect to user data, these are stored unencrypted in the cloud.
- These matrices are updated as ratings of items are added, updated or deleted in pairs.
- Proposed solution uses **user-encrypted** prediction query and response.

Privacy preserving Slope One

- Since Δ and ϕ are not private information with respect to user data, these are stored unencrypted in the cloud.
- These matrices are updated as ratings of items are added, updated or deleted in pairs.
- Proposed solution uses **user-encrypted** prediction query and response.

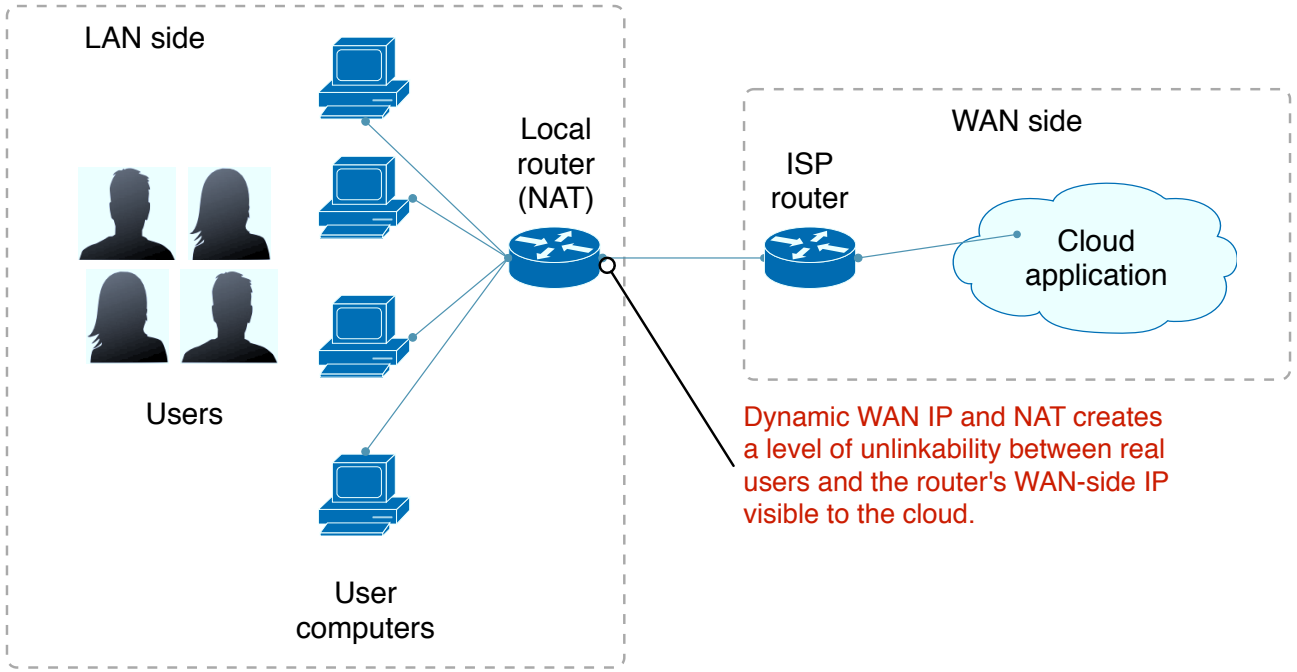
Outline

Overview of the proposed scheme



De-linking identities with IPv4 NAT

A simple IPv4 NAT can provide a naïve approach to make linkability between actual users and their WAN side IPs hard.



Addition, update, deletion and prediction of ratings[3]

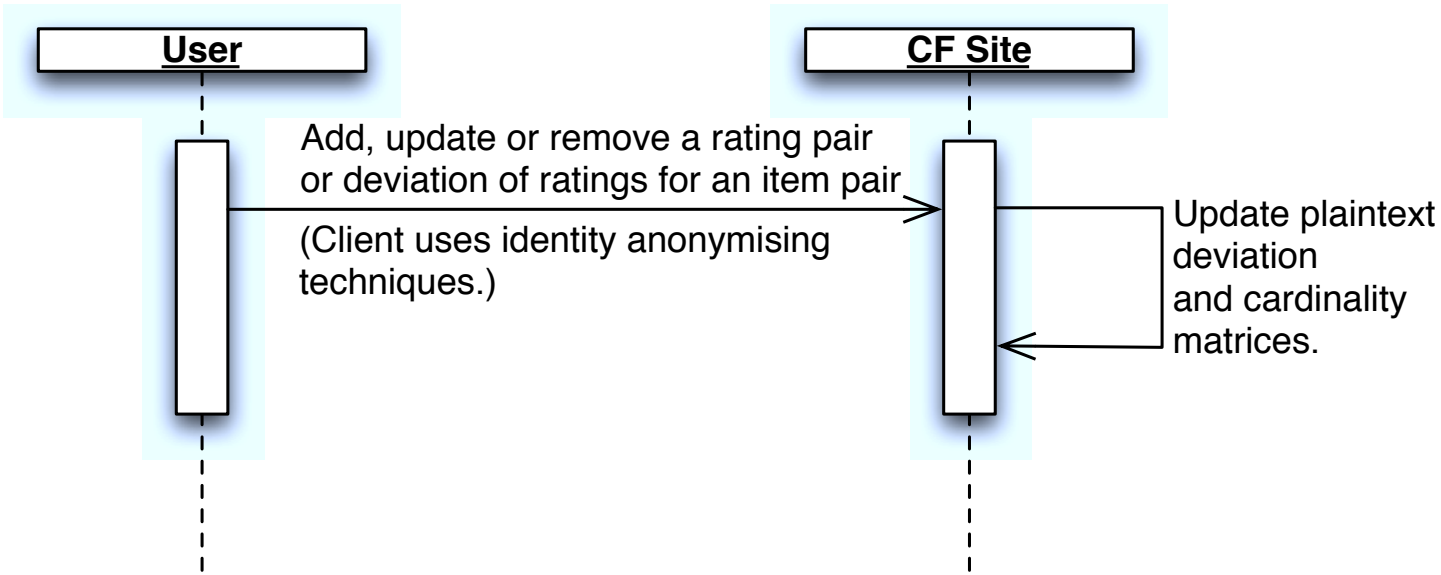


Figure: **UML sequence diagram for addition, update or deletion of data between any one user and the cloud-based CF site.**

[3] See algorithms IV.1-IV.3 in the paper.

Addition, update, deletion and prediction of ratings[3]

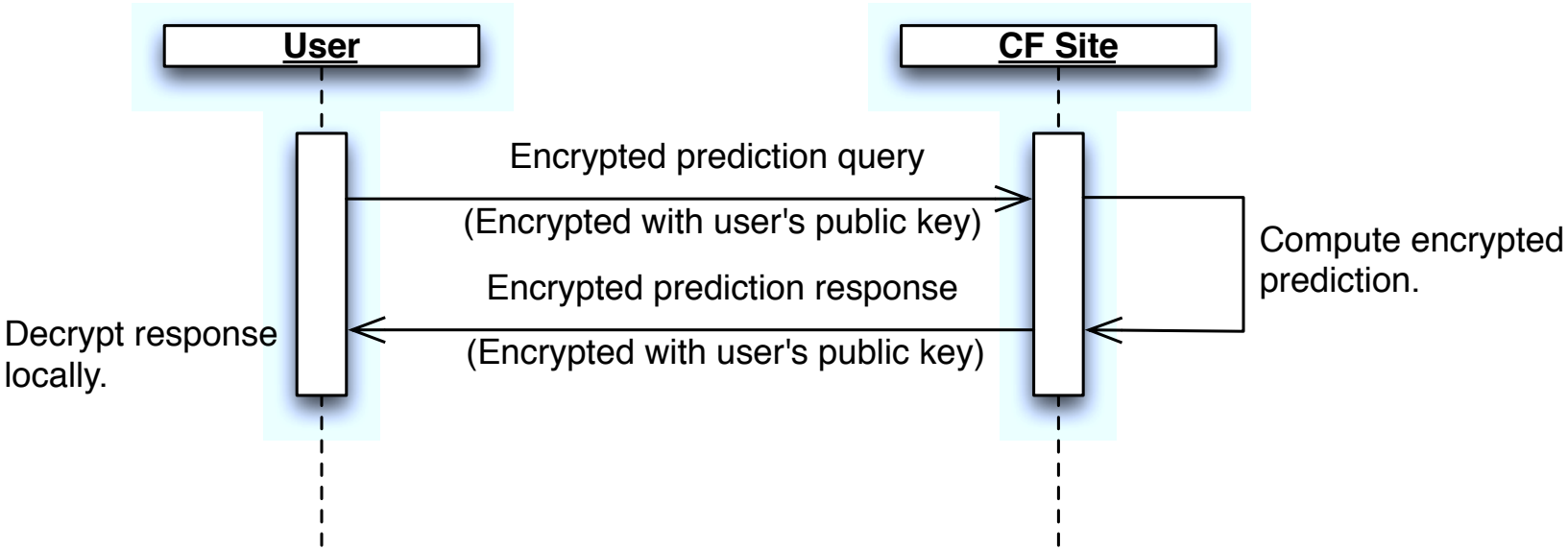


Figure: **UML sequence diagram for prediction of between any one user and the cloud-based CF site.**

[3] See algorithms IV.1-IV.3 in the paper.

Outline

Google App Engine for Java (GAE/J)

- Specialised SaaS construction PaaS cloud.
- SaaS application instances run on Java Virtual Machines with web front-ends.
- Automatically allocated scalable resources for growing user requests.
- Slow but high replication datastore access; and fast distributed in-memory cache.

Google App Engine for Java (GAE/J)

- Specialised SaaS construction PaaS cloud.
- SaaS application instances run on Java Virtual Machines with web front-ends.
- Automatically allocated scalable resources for growing user requests.
- Slow but high replication datastore access; and fast distributed in-memory cache.

Google App Engine for Java (GAE/J)

- Specialised SaaS construction PaaS cloud.
- SaaS application instances run on Java Virtual Machines with web front-ends.
- Automatically allocated scalable resources for growing user requests.
- Slow but high replication datastore access; and fast distributed in-memory cache.

Google App Engine for Java (GAE/J)

- Specialised SaaS construction PaaS cloud.
- SaaS application instances run on Java Virtual Machines with web front-ends.
- Automatically allocated scalable resources for growing user requests.
- Slow but high replication datastore access; and fast distributed in-memory cache.

Google App Engine for Java (GAE/J)

- Low CPU performance per application instance: affects cryptographic operations.
- Various performance limitations on the free quota.
- As of July 22, 2011, we measured some limitations of the GAE/J.

Google App Engine for Java (GAE/J)

- Low CPU performance per application instance: affects cryptographic operations.
- Various performance limitations on the free quota.
- As of July 22, 2011, we measured some limitations of the GAE/J.

Google App Engine for Java (GAE/J)

- Low CPU performance per application instance: affects cryptographic operations.
- Various performance limitations on the free quota.
- As of July 22, 2011, we measured some limitations of the GAE/J.

Feasibility of a PPCF scheme on the GAE/J

A. Basu, J. Vaidya, T. Dimitrakos, H. Kikuchi, *Feasibility of a privacy preserving collaborative filtering scheme on the Google App Engine – a performance case study*, Proceedings of the 27th ACM Symposium on Applied Computing (SAC) Cloud Computing track, Trento, Italy, 2012.

Google App Engine for Java (GAE/J)

- Low CPU performance per application instance: affects cryptographic operations.
- Various performance limitations on the free quota.
- As of July 22, 2011, we measured some limitations of the GAE/J.

Feasibility of a PPCF scheme on the GAE/J

A. Basu, J. Vaidya, T. Dimitrakos, H. Kikuchi, *Feasibility of a privacy preserving collaborative filtering scheme on the Google App Engine – a performance case study*, Proceedings of the 27th ACM Symposium on Applied Computing (SAC) Cloud Computing track, Trento, Italy, 2012.

Performance results on the GAE/J

Bit size[a]	Vector size[b]	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 query vector.

Performance results on the GAE/J

Time taken to predict grows linearly . . .

. . . with the size of the query vector. With 100 given ratings in the query vector, the prediction time will be about 50 seconds – an awfully long wait on a web interface!

Bit size[a]	Vector size[b]	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 query vector.

Demo

- Google App Engine for Java implementation: `http://gaejppcf.appspot.com/`.
- Attack simulation on private data: in both cases, the cloud application tracks user's IPv4 address – a typical attack scenario to attempt to link ratings to users.

Demo

- Google App Engine for Java implementation: `http://gaejppcf.appspot.com/`.
- Attack simulation on private data: in both cases, the cloud application tracks user's IPv4 address – a typical attack scenario to attempt to link ratings to users.

Outline

Conclusions

Our proposed scheme:

- uses user-encrypted predicted query and does not store users' rating data;
- makes rating-to-user linkability hard; and
- scales well on real world cloud platforms.

Conclusions

Our proposed scheme:

- uses user-encrypted predicted query and does not store users' rating data;
- makes rating-to-user linkability hard; and
- scales well on real world cloud platforms.

Conclusions

Our proposed scheme:

- uses user-encrypted predicted query and does not store users' rating data;
- makes rating-to-user linkability hard; and
- scales well on real world cloud platforms.

Future work

- Implement the proposal on vertical partition.
- Introduce parallelism in prediction queries with large query vectors.
- Conduct comparative performance analyses with other privacy preserving CF implementations on different SaaS construction PaaS clouds, e.g. the Amazon Elastic Beanstalk.
- Improve our scheme by discarding some assumptions (e.g. honest user) and dependencies (e.g. anonymiser networks).

Future work

- Implement the proposal on vertical partition.
- Introduce parallelism in prediction queries with large query vectors.
- Conduct comparative performance analyses with other privacy preserving CF implementations on different SaaS construction PaaS clouds, e.g. the Amazon Elastic Beanstalk.
- Improve our scheme by discarding some assumptions (e.g. honest user) and dependencies (e.g. anonymiser networks).

Future work

- Implement the proposal on vertical partition.
- Introduce parallelism in prediction queries with large query vectors.
- Conduct comparative performance analyses with other privacy preserving CF implementations on different SaaS construction PaaS clouds, e.g. the Amazon Elastic Beanstalk.
- Improve our scheme by discarding some assumptions (e.g. honest user) and dependencies (e.g. anonymiser networks).

Future work

- Implement the proposal on vertical partition.
- Introduce parallelism in prediction queries with large query vectors.
- Conduct comparative performance analyses with other privacy preserving CF implementations on different SaaS construction PaaS clouds, e.g. the Amazon Elastic Beanstalk.
- Improve our scheme by discarding some assumptions (e.g. honest user) and dependencies (e.g. anonymiser networks).

Thank you for listening!

Any questions?