Feasibility of a privacy preserving collaborative filtering scheme on the Google App Engine – a performance case study

Anirban Basu Jaideep Vaidya Theo Dimitrakos
Hiroaki Kikuchi

Department of Electrical Engineering
Faculty of Engineering
Tokai University (Japan)

ACM SAC 2012
March 27, 2012

What are we doing and why?

- Cloud computing is attractive for many reasons: low total cost of ownership through virtualised resource sharing, rapid on-demand scaling, high speed network access, and so on.
- Recommender systems help users tackle vast amounts of information, but recommendation (e.g., using collaborative filtering) requires computing power.
- Cloud is a solution for building a recommendation system, but there is a problem. . .
- . . . *privacy* of users' preferential data, for which there is privacy preserving collaborative filtering (PPCF).

What are we doing and why?

- Cloud computing is attractive for many reasons: low total cost of ownership through virtualised resource sharing, rapid on-demand scaling, high speed network access, and so on.
- Recommender systems help users tackle vast amounts of information, but recommendation (e.g., using collaborative filtering) requires computing power.
- Cloud is a solution for building a recommendation system, but there is a problem. . .
- . . . ***privacy*** of users' preferential data, for which there is privacy preserving collaborative filtering (PPCF).

What are we doing and why?

- Cloud computing is attractive for many reasons: low total cost of ownership through virtualised resource sharing, rapid on-demand scaling, high speed network access, and so on.
- Recommender systems help users tackle vast amounts of information, but recommendation (e.g., using collaborative filtering) requires computing power.
- Cloud is a solution for building a recommendation system, but there is a problem. . .
- . . . ***privacy*** of users' preferential data, for which there is privacy preserving collaborative filtering (PPCF).

What are we doing and why?

- Cloud computing is attractive for many reasons: low total cost of ownership through virtualised resource sharing, rapid on-demand scaling, high speed network access, and so on.
- Recommender systems help users tackle vast amounts of information, but recommendation (e.g., using collaborative filtering) requires computing power.
- Cloud is a solution for building a recommendation system, but there is a problem. . .
- . . . ***privacy*** of users' preferential data, for which there is privacy preserving collaborative filtering (PPCF).

This talk:

- is about the feasibility (in terms of performance) of a privacy preserving collaborative filtering (PPCF) scheme on the Google App Engine for Java (GAE/J).
- refers to the PPCF scheme proposed in:

This talk:

- is about the feasibility (in terms of performance) of a privacy preserving collaborative filtering (PPCF) scheme on the Google App Engine for Java (GAE/J).
- refers to the PPCF scheme proposed in:

A. Basu, H. Kikuchi, and J. Vaidya. 2011. *Privacy Preserving weighted Slope One predictor for Item-based Collaborative Filtering*, In: International Workshop on Trust and Privacy in Distributed Information Processing (workshop at the IFIPTM), Copenhagen, Denmark.

This talk:

- is about the feasibility (in terms of performance) of a privacy preserving collaborative filtering (PPCF) scheme on the Google App Engine for Java (GAE/J).
- refers to the PPCF scheme proposed in:

A. Basu, H. Kikuchi, and J. Vaidya. 2011. *Privacy Preserving weighted Slope One predictor for Item-based Collaborative Filtering*, In: International Workshop on Trust and Privacy in Distributed Information Processing (workshop at the IFIPTM), Copenhagen, Denmark.

Collaborative filtering (CF), briefly!

We often have user-item rating data like this[1]:

	Canon 7D	Leica M9	Nikon D7000	...	Olympus OM-D
Alice	5	4	-	...	3
Bob	3	5	2	...	-
Carol	-	?	4	...	3
Dave	4	3	-	...	-

- The objective is to find the rating for Leica M9 for Carol.
- A well-known recommendation technique, based on the preferences of the community, is *collaborative filtering* (CF).

[1] "-" indicates the absence of a rating.

Collaborative filtering (CF), briefly!

We often have user-item rating data like this[1]:

	Canon 7D	Leica M9	Nikon D7000	...	Olympus OM-D
Alice	5	4	-	...	3
Bob	3	5	2	...	-
Carol	-	?	4	...	3
Dave	4	3	-	...	-

- The objective is to find the rating for Leica M9 for Carol.
- A well-known recommendation technique, based on the preferences of the community, is *collaborative filtering* (CF).

[1] "-" indicates the absence of a rating.

CF using Slope One predictors

Based on:

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

CF using Slope One predictors

- A CF scheme with predictors of the form $f(x) = x + b$, hence "slope one".
- Simple yet efficient: compared with CF using cosine similarity and singular value decomposition, Slope One has the lowest mean absolute error (MAE) and root mean squared error (RMSE) on the Apache Mahout reference implementation for MovieLens 100K and 1M datasets.

CF using Slope One predictors

- A CF scheme with predictors of the form $f(x) = x + b$, hence "slope one".
- Simple yet efficient: compared with CF using cosine similarity and singular value decomposition, Slope One has the lowest mean absolute error (MAE) and root mean squared error (RMSE) on the Apache Mahout reference implementation for MovieLens 100K and 1M datasets.

CF using Slope One predictors

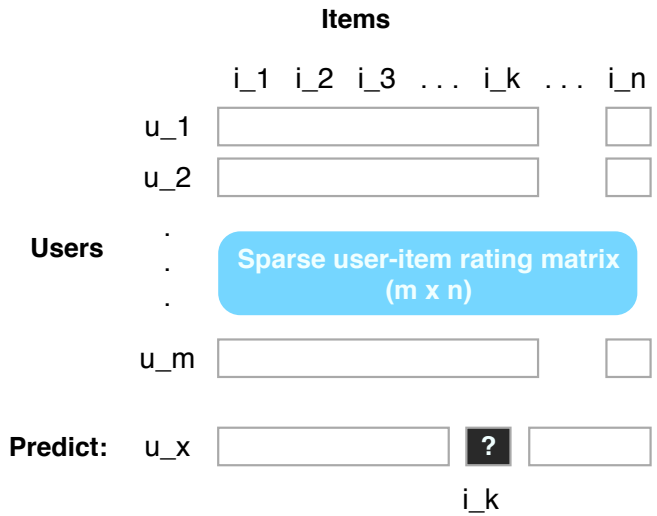

Figure: **The general CF problem.**

CF using Slope One predictors

To predict using Slope One, two things ought to be pre-computed:

- ***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 $m \times n$ (i.e., n items) then Δ is of dimension $n \times n$.
- ***Cardinality matrix*** or $\mathcal{C}$: 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.

CF using Slope One predictors

To predict using Slope One, two things ought to be pre-computed:

- ***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 $m \times n$ (i.e., n items) then Δ is of dimension $n \times n$.
- ***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.

CF using Slope One predictors

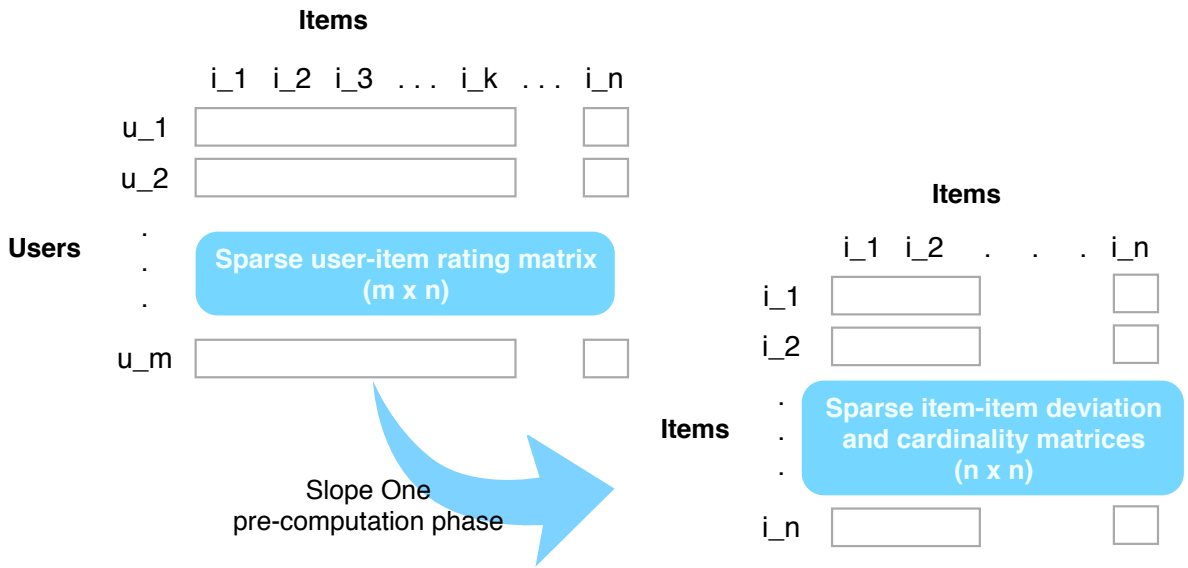

Figure: **Slope One pre-computation creates a 'model' which is used for prediction.**

The weighted Slope One predictor

- 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 number 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 predictor

- 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 number 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:

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

Preserving privacy with Slope One CF

- Uses an *additively homomorphic public key cryptosystem* – the Damgärd-Jurik cryptosystem.
- homomorphic addition (we denote encryption and decryption functions by $\mathcal{E}()$ and $\mathcal{D}()$ respectively):

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

and homomorphic multiplication:

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

Preserving privacy with Slope One CF

- Uses an *additively homomorphic public key cryptosystem* – the Damgärd-Jurik cryptosystem.
- homomorphic addition (we denote encryption and decryption functions by $\mathcal{E}()$ and $\mathcal{D}()$ respectively):

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

and homomorphic multiplication:

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

Preserving privacy with Slope One CF

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 thus, the final prediction is given as:

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

Preserving privacy with Slope One CF

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 thus, the final prediction is given as:

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

Preserving privacy with Slope One CF

- The $\mathcal{E}(\Delta)$ and ϕ matrices are pre-computed.
- An encrypted query is run and the result is decrypted using threshold decryption keys.
- Note that the encrypted query can be answered by any of the collaborating sites without leakage of private information.
- Supports horizontal and vertical partitioning, using secure scalar product (Vaidya and Clifton), of the rating dataset.

Preserving privacy with Slope One CF

- The $\mathcal{E}(\Delta)$ and ϕ matrices are pre-computed.
- An encrypted query is run and the result is decrypted using threshold decryption keys.
- Note that the encrypted query can be answered by any of the collaborating sites without leakage of private information.
- Supports horizontal and vertical partitioning, using secure scalar product (Vaidya and Clifton), of the rating dataset.

Preserving privacy with Slope One CF

- The $\mathcal{E}(\Delta)$ and ϕ matrices are pre-computed.
- An encrypted query is run and the result is decrypted using threshold decryption keys.
- Note that the encrypted query can be answered by any of the collaborating sites without leakage of private information.
- Supports horizontal and vertical partitioning, using secure scalar product (Vaidya and Clifton), of the rating dataset.

Preserving privacy with Slope One CF

- The $\mathcal{E}(\Delta)$ and ϕ matrices are pre-computed.
- An encrypted query is run and the result is decrypted using threshold decryption keys.
- Note that the encrypted query can be answered by any of the collaborating sites without leakage of private information.
- Supports horizontal and vertical partitioning, using secure scalar product (Vaidya and Clifton), of the rating dataset.

Google App Engine for Java (GAE/J)

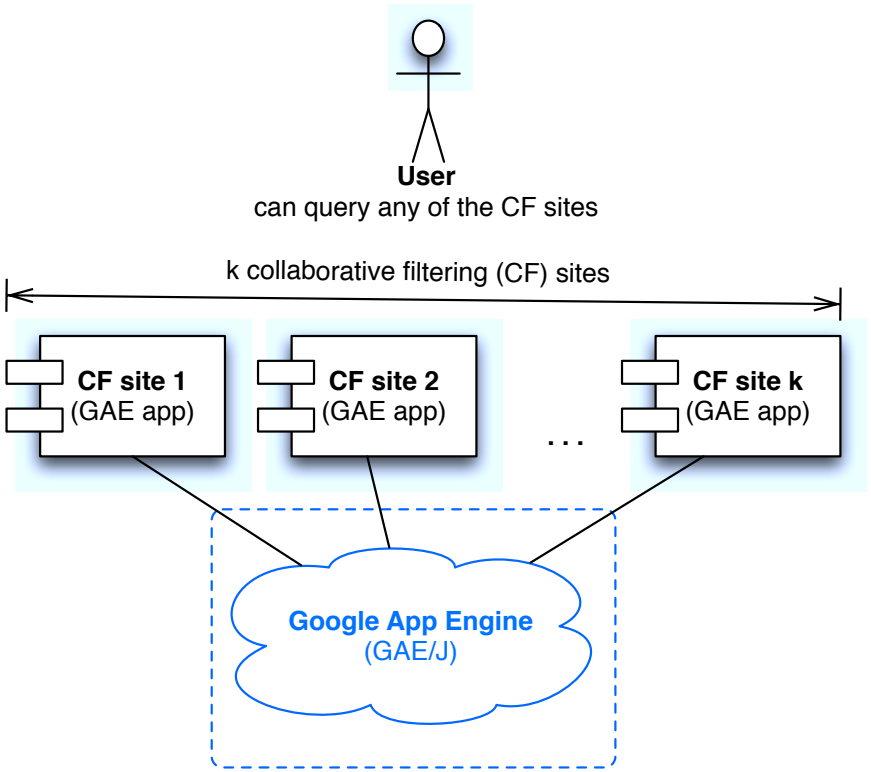

Figure: **An expected deployment on the Google App Engine.**

Google App Engine for Java (GAE/J)

- A Software-as-a-Service (SaaS) construction Platform-as-a-Service (PaaS) cloud.
- It offers on-demand transparent scalability with low costs, including a daily free quota.
- Java servlet based computation model but also allows for batch computations using task queues, computationally more powerful backend instances and cron.

Google App Engine for Java (GAE/J)

- A Software-as-a-Service (SaaS) construction Platform-as-a-Service (PaaS) cloud.
- It offers on-demand transparent scalability with low costs, including a daily free quota.
- Java servlet based computation model but also allows for batch computations using task queues, computationally more powerful backend instances and cron.

The experiments and our test platform

We run three experiments – building blocks for the PPCF scheme:

- Cryptographic primitives of the Damgärd-Jurik cryptosystem.
- Secure scalar product.
- Reading in and storing the MovieLens 100K dataset.

The experiments and our test platform

We run three experiments – building blocks for the PPCF scheme:

- Cryptographic primitives of the Damgärd-Jurik cryptosystem.
- Secure scalar product.
- Reading in and storing the MovieLens 100K dataset.

The experiments and our test platform

We run three experiments – building blocks for the PPCF scheme:

- Cryptographic primitives of the Damgärd-Jurik cryptosystem.
- Secure scalar product.
- Reading in and storing the MovieLens 100K dataset.

The experiments and our test platform

on:

- the Google App Engine for Java 1.5.2 production with the daily free quota.
- the GAE/J SDK 1.5.2 on a 2.53 GHz Intel Core 2 Duo 64-bit processor, 8 GB RAM running Mac OS X 10.6.8 and 64-bit Java 1.6.
- with, the tests correct as of July 22, 2011 noting that the GAE/J billing model has changed substantially after August 31, 2011.

The experiments and our test platform

on:

- the Google App Engine for Java 1.5.2 production with the daily free quota.
- the GAE/J SDK 1.5.2 on a 2.53 GHz Intel Core 2 Duo 64-bit processor, 8 GB RAM running Mac OS X 10.6.8 and 64-bit Java 1.6.
- with, the tests correct as of July 22, 2011 noting that the GAE/J billing model has changed substantially after August 31, 2011.

The experiments and our test platform

on:

- the Google App Engine for Java 1.5.2 production with the daily free quota.
- the GAE/J SDK 1.5.2 on a 2.53 GHz Intel Core 2 Duo 64-bit processor, 8 GB RAM running Mac OS X 10.6.8 and 64-bit Java 1.6.
- with, the tests correct as of July 22, 2011 noting that the GAE/J billing model has changed substantially after August 31, 2011.

Cryptographic primitives

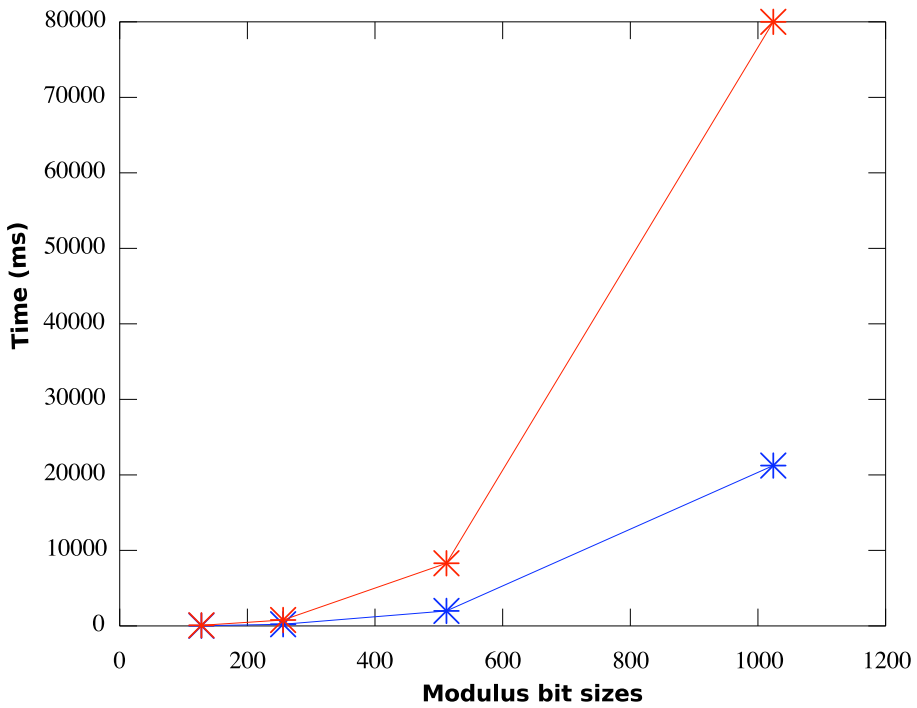

Figure: **Damgärd-Jurik threshold key generation. The timings for GAE/J at 1024 bits are linearly estimated because the test failed due to the 30s execution time limit.**

Cryptographic primitives

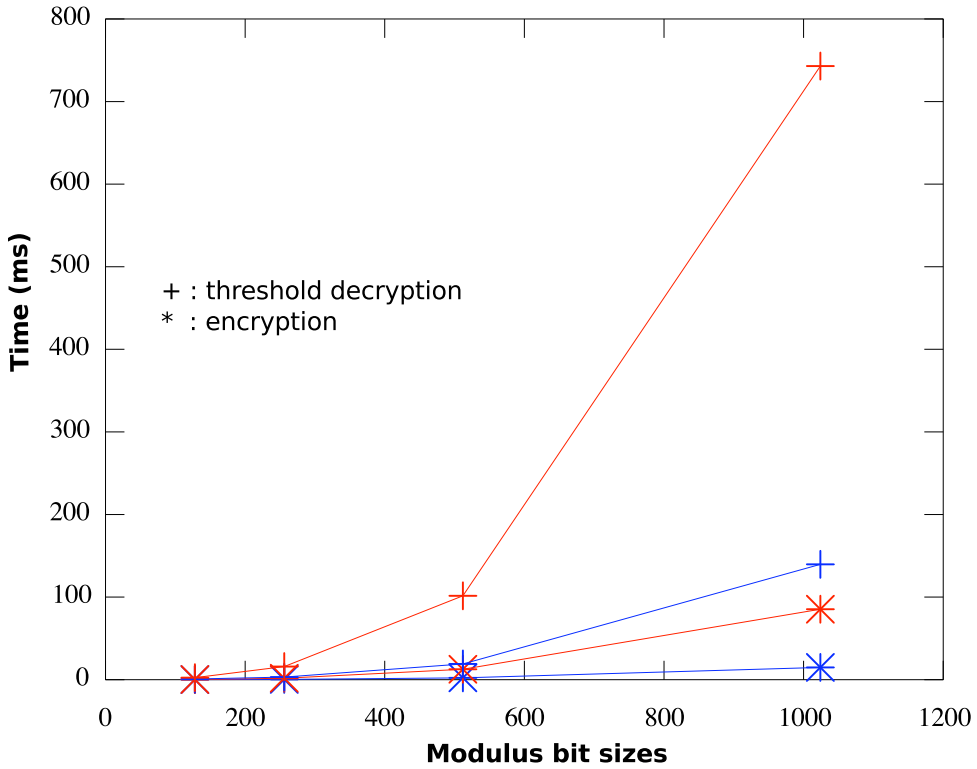

Figure: **Damgärd-Jurik encryption and threshold decryption.**

Cryptographic primitives

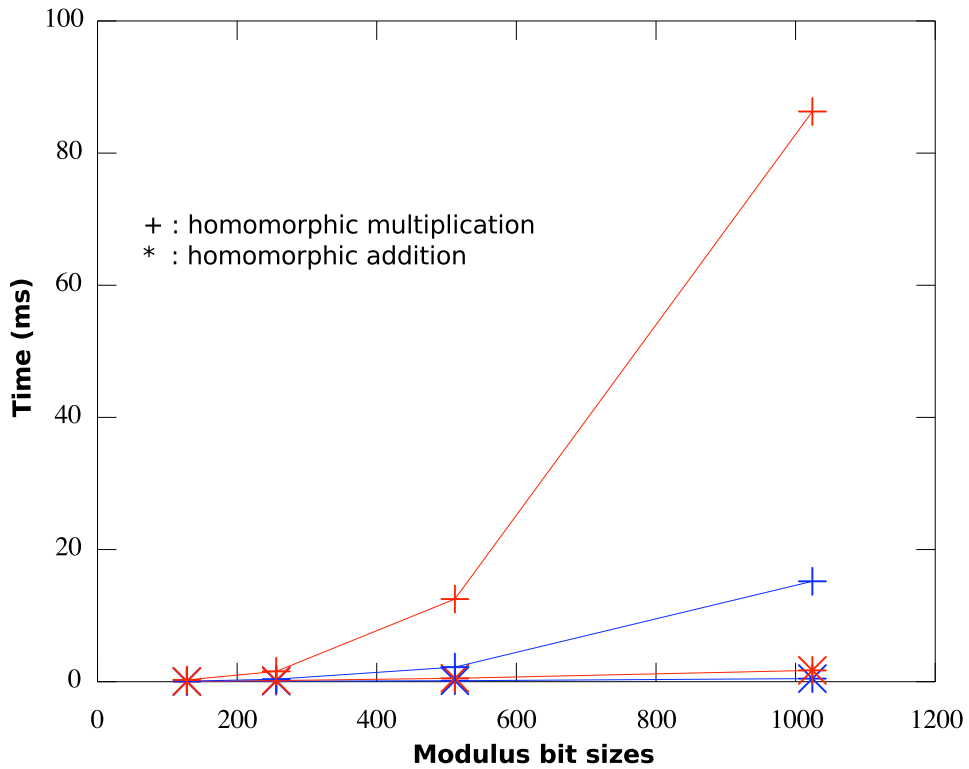

Figure: **Damgärd-Jurik homomorphic addition and multiplication.**

Secure scalar product

(run on the task queue)

Target	Bits	v-size[2]	enc[3]	ssp[4]	t-dec[5]
GAE/J	256	100	157.5ms	22.9ms	13.7ms
SDK	256	100	43.7ms	9.55ms	4.5ms
GAE/J	256	1000	1511ms	223ms	12.8ms
SDK	256	1000	362ms	88.5ms	3.4ms
GAE/J	256	10K	15649ms	2231ms	13.1ms
SDK	256	10K	3689ms	835ms	4.1ms
GAE/J	256	100K	154995ms	22674ms	12.6ms
SDK	256	100K	36056ms	7962ms	3.5ms

[2]v-size: Secure scalar product vector size

[3]enc: Total encryption

[4]ssp: Secure scalar product

[5]t-dec: Threshold decryption

Reading in and storing the MovieLens 100K dataset

- Multiple approaches had only partial successes on the GAE/J due to time quota restrictions. Blob storage not allowed in the free quota!
- Even using just the in-memory cache, on the SDK the task completed in 53,233ms while it failed after 598,055ms as a *DeferredTask* on the task queue on the GAE/J.
- Deletion of the dataset was fast using MapReduce, but the Google App Engine only supports Map function on the Java version so far.

Reading in and storing the MovieLens 100K dataset

- Multiple approaches had only partial successes on the GAE/J due to time quota restrictions. Blob storage not allowed in the free quota!
- Even using just the in-memory cache, on the SDK the task completed in 53,233ms while it failed after 598,055ms as a *DeferredTask* on the task queue on the GAE/J.
- Deletion of the dataset was fast using MapReduce, but the Google App Engine only supports Map function on the Java version so far.

Reading in and storing the MovieLens 100K dataset

- Multiple approaches had only partial successes on the GAE/J due to time quota restrictions. Blob storage not allowed in the free quota!
- Even using just the in-memory cache, on the SDK the task completed in 53,233ms while it failed after 598,055ms as a *DeferredTask* on the task queue on the GAE/J.
- Deletion of the dataset was fast using MapReduce, but the Google App Engine only supports Map function on the Java version so far.

The limitations of the GAE/J and alternatives

- Execution time limit.
- High replication but slow access to datastore.
- Lack of support for concurrency.
- Lack of control over resource allocation (it is a bit better now).

The limitations of the GAE/J and alternatives

- Execution time limit.
- High replication but slow access to datastore.
- Lack of support for concurrency.
- Lack of control over resource allocation (it is a bit better now).
- Alternative similar SaaS engine: Amazon Elastic Beanstalk.

The limitations of the GAE/J and alternatives

- Execution time limit.
- High replication but slow access to datastore.
- Lack of support for concurrency.
- Lack of control over resource allocation (it is a bit better now).
- Alternative similar SaaS engine: Amazon Elastic Beanstalk.

The limitations of the GAE/J and alternatives

- Execution time limit.
- High replication but slow access to datastore.
- Lack of support for concurrency.
- Lack of control over resource allocation (it is a bit better now).
- Alternative similar SaaS engine: Amazon Elastic Beanstalk.

The limitations of the GAE/J and alternatives

- Execution time limit.
- High replication but slow access to datastore.
- Lack of support for concurrency.
- Lack of control over resource allocation (it is a bit better now).
- Alternative similar SaaS engine: Amazon Elastic Beanstalk.

Conclusions

- The GAE/J in its current state (as of version 1.5.2) is unusable for our PPCF implementation scenario.
- In general:

Conclusions

- The GAE/J in its current state (as of version 1.5.2) is unusable for our PPCF implementation scenario.
- In general:
  - the cloud may not always provide better performance, especially with public key encryption.
  - GAE/J performance should be improved to match competitors.
  - theoretical algorithms that intend to be deployed on the cloud must be tested on real cloud platforms.

Conclusions

- The GAE/J in its current state (as of version 1.5.2) is unusable for our PPCF implementation scenario.
- In general:
  - the cloud may not always provide better performance, especially with public key encryption.
  - GAE/J performance should be improved to match competitors.
  - theoretical algorithms that intend to be deployed on the cloud must be tested on real cloud platforms.

Conclusions

- The GAE/J in its current state (as of version 1.5.2) is unusable for our PPCF implementation scenario.
- In general:
  - the cloud may not always provide better performance, especially with public key encryption.
  - GAE/J performance should be improved to match competitors.
  - theoretical algorithms that intend to be deployed on the cloud must be tested on real cloud platforms.

Conclusions

- The GAE/J in its current state (as of version 1.5.2) is unusable for our PPCF implementation scenario.
- In general:
  - the cloud may not always provide better performance, especially with public key encryption.
  - GAE/J performance should be improved to match competitors.
  - theoretical algorithms that intend to be deployed on the cloud must be tested on real cloud platforms.

Future work

- Run a more exhaustive set of experiments and compare the various SaaS engines.
- Investigate, design and implement: less computationally demanding and more suitable for the cloud privacy preserving recommender systems.

Future work

- Run a more exhaustive set of experiments and compare the various SaaS engines.
- Investigate, design and implement: less computationally demanding and more suitable for the cloud privacy preserving recommender systems.

Thank you for listening!

Any questions?