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

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

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

Theo Dimitrakos
British Telecom
Adastral Park
Martlesham Heath
Ipswitch IP5 3RE. UK
theo.dimitrakos@bt.com

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

ABSTRACT

The cloud is a utility computing infrastructure that has caused a paradigm shift in the way organisations requisition, allocate, and use IT resources. One big challenge is to preserve the confidentiality of information on the cloud. Most typical solutions use cryptographic techniques without considering how well suited they are to the cloud. This paper presents a performance case-study on implementing the building blocks of a privacy preserving collaborative filtering (PPCF) scheme in Java on the Google App Engine (GAE/J) cloud platform. The results show that the GAE/J in its current state exhibits serious performance bottlenecks for the chosen application scenario. This case study highlights the need for better performance from the GAE/J. It also informs the need for validating theoretical cloud security algorithms on real cloud computing platforms in which many performance expectations do not hold.

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

SAC'12 March 25-29, 2012, Riva del Garda, Italy.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

cloud computing evaluation, security

Keywords

cloud computing, performance, privacy

1. INTRODUCTION

The cloud landscape continues to evolve quickly, and, as a true ecosystem, has seen the introduction of several new technologies, delivery models and players into the market. Cloud computing is characterised by pooling and sharing of resources, broad network access, rapid elasticity, on-demand service provisioning (with a strong self-service element), offering measured service, supporting (although not necessitating) multi-tenancy. Cloud architecture embodiments offer a means of delivering ICT infrastructure, platform (i.e. application execution environment) and software as a service (SaaS). They also support several deployment models including private, community, public and hybrid. The benefits promised by this paradigm include cost and performance optimisation, economy of scale, flexible utilisation and charging model, ease of connectivity and access to shared services, cost efficient introduction of redundancy and continuity of provision. Security, resilience and compliance are the main concerns that are challenging wider use of cloud computing and most likely to drive remaining innovation and market differentiation efforts in this area. These benefits of the cloud model for IT service delivery are attractive, but the very nature of the model means that customers have less direct control over the infrastructure and the data that is being hosted or processed by the external cloud providers. While

this perceived lack of control[1] is not a new development, the relatively new model of the cloud brings this issue into sharp focus. The key security challenges facing cloud computing include regulatory compliance, absence of security standards and certification, confidentiality and integrity of data at rest or in motion in the cloud, data and process isolation, multi-tenancy of shared security services. Such challenges come on top of the common security issues relating to the risk of externalising management processes commonly performed by privileged users, vulnerabilities introduced by inadequately protected use of virtualisation technology to empower cloud services, and the security of its integration with corporate IT infrastructure, and lack of implementing protection in depth when de/re-perimeterising through the integration of cloud services into the corporate infrastructure. Different kinds of cloud computing services expose different entry points into the cloud provider and offer to the customer different types of service management operations, depending on the service model being offered, i.e. infrastructure, platform, and software. In turn, these create different attack surfaces, severity and effects of exploits, as well as different probabilities of a security breach[2].

1.1 The main goals of this paper

Given the privacy and security challenges in the cloud, consider a scenario where different web applications, hosted on a platform-as-a-service (PaaS) public cloud, together want to implement a privacy preserving rating based collaborative filtering. While there are a number of PaaS cloud offerings (e.g. Microsoft Azure, Amazon Elastic Beanstalk), we restrict ourselves to a specialised SaaS construction PaaS cloud: the Google App Engine for Java (GAE/J) in this paper. We present a case study on implementing the building blocks of a privacy preserving collaborative filtering (PPCF) scheme [2] on the GAE/J.

Many privacy preserving data mining schemes employ cryptographic techniques that theoretically assume their suitability in the cloud but are rarely tested in a pragmatic environment. While a collaborative filtering scheme is used as a suitable example, the results shown in this paper are applicable to any other applications that use similar primitives. Experimental results from this work-in-progress paper suggest that despite being a promising cloud computing platform with rapidly growing feature set, the GAE/J at its current state exhibits remarkable performance bottlenecks in the chosen application scenario. In addition, our work also informs theoretical researchers in privacy for cloud computing to validate their results on actual cloud computing platforms.

1.2 Contributions

The contributions of this paper are summarised as follows:

1. This is the first case-study, to our knowledge, to have looked into the current state-of-the-art of the Google App Engine for Java (GAE/J) platform with respect to the implementation of the building blocks of a privacy preserving collaborative filtering (PPCF) scheme.
2. The current limitations of the GAE/J are presented in this paper, which shows that the cloud does not always guarantee better performance than a non-cloud based implementations.
3. Through the exercise presented in this paper, the need for better performance from the GAE/J is highlighted. The paper also emphasizes the fact that many perfectly theoretically achievable performances may not be realistic in actual cloud deployments, and therefore any future cloud security research should ensure some form of validation on actual cloud environments.

The rest of this paper is organised as follows: §2 provides a brief overview of the related work and the background to the experiments. Experiments are described that constitute the building blocks for implementing the PPCF scheme in §3 followed by the computational models for batch processing in GAE/J in §4. §5 presents experimental results. A summary of the limitations of the GAE, in its current form, are presented in §6, followed by conclusions and future work in §7.

2. RELATED WORK AND BACKGROUND TO THE CASE STUDY

Armburst et. al. in [1] present an exhaustive overview, of the-then (in 2009) available cloud computing services. The paper compares the costs of running applications on public cloud platforms with running them on privately developed infrastructure. It also describes various issues with deploying applications on the cloud, one of which is privacy. Chow et. al. [4] characterize the problems and their impact on adoption of cloud computing. There has been much recent interest in preserving confidentiality on the cloud. Several ongoing and past efforts focus on helping organizations to understand and plan for security issues. By far the most widely acknowledged initiative has been the Security Guidance for Critical Areas of Focus in Cloud Computing[3], which aims to help organisations make informed decisions regarding if, when, and how to adopt cloud computing.The Trusted Cloud Initiative aims to help cloud providers develop industry-recommended, secure and interoperable identity, access and compliance management practices while the Cloud Controls Matrix[4] aims to provide fundamental security principles to guide cloud vendors and to assist prospective cloud customers in assessing the overall security risk of a cloud provider. A recent paper [9] presents a novel scheme of preventing cloud nodes from learning plaintext data stored in backend cloud databases. It allows users to perform content-level, fine-grained private search and decryption of the data stored in the cloud databases. Others [10, 3] have also recently proposed privacy preserving computations using MapReduce [7] style parallelism, which is suitable in a cloud computing infrastructure. MapReduce follows from a functional programming paradigm for parallel processing, where a user-specified function – *map* – takes a key-value pair to generate a set (may contain no elements) of intermediate key-value pairs. Another user-specified function – *reduce* – merges all intermediate values associated with the same intermediate key.

Google App Engine (GAE) is a SaaS construction PaaS cloud for running web applications developed in Python,

[1]The BT Viewpoint blog on this topic at `http://goo.gl/pHHvN` explores this in more detail.

[2]The blog-post at BT Secure Thinking site `http://goo.gl/x5fFv` explores this in greater detail.

[3]Report available at `http://goo.gl/xvI2A`.

[4]For more information see `http://goo.gl/yFOKC`.

Java and Go. A key advantage of GAE is its transparent scalability as presented in Google I/O 2011[5]. Google claims that GAE/J creates new application instances nearly instantly (and using some optimised re-allocation techniques if free instances are available) with incoming requests to the web front-end of an application. However, GAE/J penalises applications that have higher "application latency", which Google defines as the turnaround time for an application to process a user request. The higher this latency, the more conservative GAE/J is in terms of allocating new resources. GAE/J treats a value $\geq 1,000ms$ as a bad application latency, which is not unusual for a servlet-based application performing complex cryptographic operations.

GAE/J allows as little configurability as possible to the developer in terms of choosing what hardware or virtual machine an application in the cloud is run on. Automatic resource allocation in GAE makes it attractive in terms of transparent scalability. The downside of automatic resource allocation is that performance of individual units of computation is often unpredictable depending on resource availability and allocation overheads.

2.1 A privacy-preserving collaborative filtering scheme

This paper attempts to construct the building blocks in order to implement a privacy preserving collaborative filtering (PPCF) scheme proposed in [2], which extends from the well known weighted Slope One predictor due to Lemire and McLachlan [8]. It works for pure horizontal and pure vertical dataset partitions. It uses the additively homomorphic threshold variant of the Damgärd-Jurik [6] public key cryptosystem. Akin to the original Slope One CF, this PPCF scheme has a pre-computation stage and a prediction stage. The pre-computation stage involves a number of homomorphic cryptographic operations while the computational performance in the prediction stage is dominated by the time taken for the threshold decryption operation. The secure scalar product [5] in the pre-computation stage when the dataset is vertically partitioned.

A typical implementation scenario of this PPCF scheme, assuming "no insider threats", on the GAE/J is illustrated in figure 2.1, where k-CF sites (i.e. belonging to k organisations), each implemented as an independent GAE/J application, want to collaboratively build the privacy preserving weighted Slope One predictor. Note that the "no insider threats" assumption has been made for the sake of simplicity. It is an important to discard this assumption: something we will consider in future work. If we discard this assumption then key generation, encryption and decryption should be outsourced to a set of trusted third parties and the cloud should work with already encrypted ratings. However, doing so raises issues with range check on already encrypted rating data. For example, a malicious user can encrypt out of bounds rating values, which without validity checks, will generate unwanted results from homomorphic operations. A sufficiently large number of such unusable values can render the CF scheme unusable. To address this, either one would need to use zero knowledge proof techniques for range checking, or alternative encryption techniques [11] that support range checking (which may not have the homomorphic property though).

[5]Video reference: "Scaling App Engine Applications", May 2011. Available at: `http://goo.gl/WfhSV`.

Because of the simplicity of the PPCF scheme in [2], it is sufficient to test the performance of its building blocks, e.g. the cryptographic primitives.

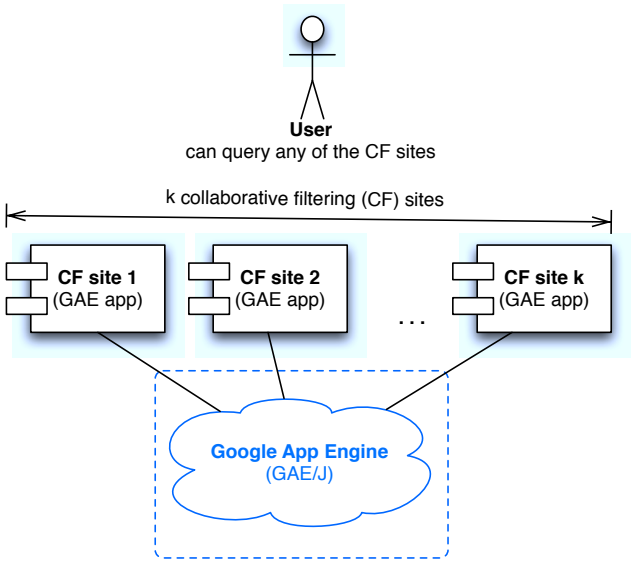

Figure 2.1: Implementation scenario: "no insider threats" assumption holds in the area within the blue dotted lines.

2.2 Environment for the experiments

At the time of writing this paper, the performance of the GAE/J was evaluated in comparison with the GAE/J SDK running on a development machine with 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. The reason for testing with the GAE/J SDK was to use an emulation platform comparable with the GAE/J. The tests were performed with the July 22, 2011 version 1.5.2 of the GAE/J SDK and the GAE/J production environment. The test results are correct as of July 22, 2011.

3. EVALUATION

3.1 Experiments

3.1.1 *Cryptographic primitives*

The tested cryptographic primitives of the Damgärd-Jurik cryptosystem are: *key generation*, *encryption*, *homomorphic multiplication*, *homomorphic addition*, and *threshold decryption*. In the absence of support for Java Cryptographic Extensions in the GAE/J, the open-source University of Texas (Dallas) Paillier Threshold Encryption toolbox[6] was extended. It contains an implementation of the Damgärd-Jurik cryptosystem.

3.1.2 *Secure scalar product*

Since the PPCF scheme uses scalar product in order to deal with dataset vertical partitions, experiments were run

[6]See: `http://goo.gl/5wnHg`.

with a non-parallelised implementation of the two-party version of a secure scalar product.

3.1.3 *Reading in, storing and deleting the MovieLens 100K dataset*

This test focussed on reading in, storing, and deleting the MovieLens 100K dataset in the distributed high replication datastore as well as in the Memcache, which is a distributed key-value based in-memory cache in GAE/J. The billing free quota imposes a restriction on BLOB storage, the absence of which may have impact on performance and on the way this experiment is run. Deleting the stored data was also performed over MapReduce to achieve improved performance[7].

4. GAE COMPUTATION MODEL

A GAE/J application is usually a web application coded using Java servlets. The time limit for a servlet to respond to a user request is 30 seconds. Apart from the newly introduced support for MapReduce[8], batch computations can be run using: (1) the task queue, (2) backends, and (3) cron jobs.

4.1 Task queue

GAE/J applications are not allowed to start additional threads. Task queues guarantee eventual, hence not necessarily parallel, execution of tasks. Task queues retry (a certain number of times) executions of tasks in case of failures. Task queues can execute items e.g. a URL to another servlet, a `DeferredTask` which inherits Java's `Runnable` interface. A separate process is created for each task by the GAE/J. Such processes may run in parallel depending on the available computational resources. One limitation is that individual tasks in a task queue cannot run for longer than ten minutes unless they are run on a backend instance.

4.2 Backend

Backend instances in Google App Engine are transparent, described in XML. Applications and tasks can be made to run on a backend instance of a certain configuration without additional changes to the program. Backends remove the time limit for tasks in the task queue allowing them to run indefinitely (or so long as the free quota is available). Backend instances are of various types, e.g. the most powerful type provides 4.8 GHz of CPU and 1024 MB RAM, which is still eight-fold lower than the RAM in the machine used for the experiments. Without backends, the allocation of CPU and memory to application instances is dynamic so an application may be able to get more CPU on the default instance than on a specific backend.

4.3 Cron

Described in XML, cron jobs can continue to run for up to ten minutes unless they are run on a backend instance. Unlike tasks in a task queue, cron jobs cannot be accessed programmatically from inside a servlet. While unattended computations (e.g. web crawler, report generation) can run efficiently as cron jobs, it cannot run code that is expected to return values to its caller within a short period of time.

5. EXPERIMENTAL RESULTS

[7]Code adapted from: `http://goo.gl/1lIkM`.
[8]See: `http://goo.gl/ScBy`.

5.1 Cryptographic operations

The results of the experiments with the Damgärd-Jurik threshold cryptosystem are presented in figures 5.1 through 5.3. For threshold decryption, the parameter values used were: collaborating entities ($l = 8$) and threshold decryption entities ($w = 4$). Note that homomorphic multiplications are almost as slow as encryptions because -1 was used as the integer multiplicand, which is adjusted in the field of the modulus before exponentiation making it a sufficiently big positive value. Occasional improved performance from the GAE/J was observed but it was unpredictable depending on how it allocated resources to the application instances.

Few possibilities to obtain better performance from the GAE/J are: (1) more developer control over how resources are allocated, and (2) better concurrency support for parallelisable tasks.

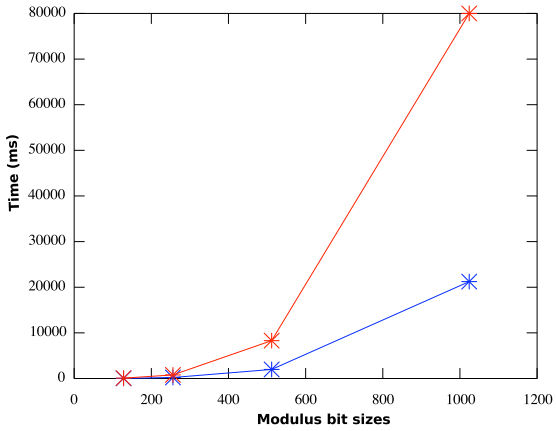

Figure 5.1: Graph showing performances (GAE/J in red and SDK in blue) of key generation. The timings for GAE/J at 1024 are linearly estimated because the test failed.

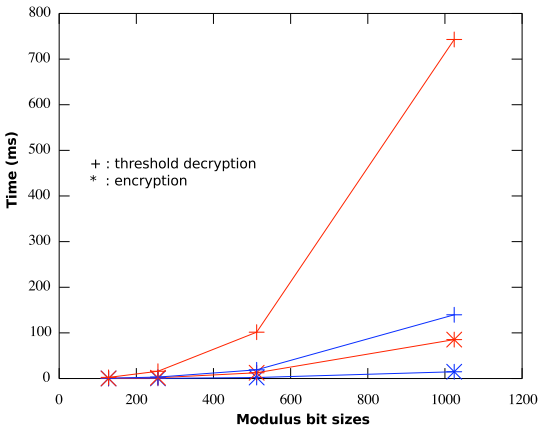

Figure 5.2: Graph showing performances (GAE/J in red and SDK in blue) of encryption and threshold decryption. The timings for GAE/J at 1024 are linearly estimated because the test failed.

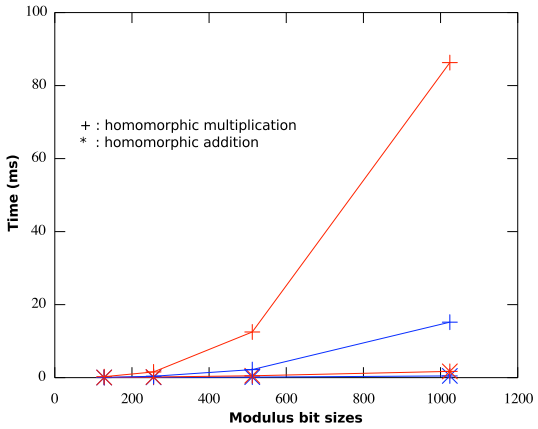

Figure 5.3: Graph showing performances (GAE/J in red and SDK in blue) of homomorphic addition and multiplication. The timings for GAE/J at 1024 are linearly estimated because the test failed.

5.2 Secure scalar product

Using the same parameters for the cryptosystem and fixing the modulus bit length to 256 bits, the results of the secure scalar product computation are shown in table 5.1. Note that the $|n| = 256$ for the cryptosystem is far from the industry standard of $|n| = 2048$ because the GAE/J servlet is unable to perform cryptographic operations if $|n| \geq 1024$. Despite the low bit size, note that the secure scalar product time cost linearly increases with the increase in size of the data vector. This consistency is expected.

Table 5.1: Comparison of secure scalar product performance between the GAE and the GAE SDK with the Damgärd-Jurik threshold cryptosystem. Rows with better performances are highlighted in green.

Target	Bits	v-size[a]	enc[b]	ssp[c]	t-dec[d]
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

[a] v-size: Secure scalar product vector size
[b] enc: Encryption
[c] ssp: Secure scalar product
[d] t-dec: Threshold decryption

5.3 Reading in, storing and deleting the MovieLens 100K dataset

With the free daily quota of the GAE/J, uploading the dataset as a Blob was not an option. Instead, the MovieLens 100K dataset file was fetched from an external HTTP URL. While fetching, a task in the task queue read each line from the HTTP socket stream because it could not be stored as a disk file on the GAE/J. In addition, the GAE/J imposes a total bandwidth restriction of 56MB/min on the billing free quota. Each line was parsed and a corresponding record was created in the datastore. Due to bandwidth and other resource constraints, the task queue met with the timeout of 10 minutes before the entire read could be completed. However, a substantial chunk of the data was read through the use of a backend (see §4.2) for over an hour before running out of the daily free backend quota of US $0.72. At that point, about 75,000 out of the 100,000 records were imported. A third approach of using servlets as tasks instead of `DeferredTask` achieved better parallelism but failed due to restrictions of the free quota. The performance was, however, below par compared to the MapReduce based deletion discussed below.

For deleting the records, first a naïve approach was taken through a fetch-all query followed by a delete-all query using `Query::deletePersistentAll()`. GAE/J documentation informs that a full scan of the datastore is done in such an operation making it computationally intensive. The deletion process failed because the fetch all query took long enough to be cancelled by GAE[9]. A more efficient deletion was employed using the App Engine implementation of MapReduce. The mapper broke down the deletion task into multiple shards to delete the rows in a parallel fashion. It accomplished deletion of 74,062 entities in 159 seconds, which is an impressive speed-up in comparison with the datastore write operation of over an hour.

In addition to the datastore, the volatile in-memory storage using Memcache was tested and compared the performance with the SDK. After increasing the default heap size for the SDK, 100K records were downloaded and stored into Memcache in 53,233 ms as a deferred task, while on the GAE/J, the task stopped responding after 598,055 ms.

6. LIMITATIONS OF THE GAE/J

The experimental results indicate that the Google App Engine PaaS, as of now, does not provide performance comparable with a reasonably priced hardware (single machine or cluster) in a research lab. Despite being made of cheap hardware instances, the GAE/J has restricted support for parallelism. Hence it is unsuitable for implementing applications that conduct computationally intense operations. The limitations of the GAE/J affecting the experiments are summarised under the following heads[10].

Time limit User requests that cannot complete within specific times in a servlet (30 seconds) or in the task queue (10 minutes) are automatically halted. This limitation is, however, not applicable to tasks and cron jobs running on a backend instance.

High replication datastore While providing resilient storage, the high replication datastore provides highly con-

[9] A separate Datastore API deadline restricts the query completion time to 30s.
[10] As of August 31, 2011 Google notified (see `http://goo.gl/18s59`) App Engine users of the changes in their billing structure as well as the free quota. The results in this paper are valid as of July 22, 2011. The limitations presented here are observed at this point in time and are likely to change in future.

current data access but at the cost of performance when compared with an in-memory access to a data structure or in-memory lightweight databases, such as the Oracle Berkeley DB[11]. In addition, the query language for the datastore, i.e. GQL is severely limited in comparison with competitor data query languages.

Concurrency The GAE/J lacks extensive support for concurrent computation. For example, it only supports limited Map operations of MapReduce. Applications could greatly benefit from parallelism despite the individual cheap hardware instances used in the GAE.

Lack of control GAE/J provides hardly any control to the developer in terms of how computing resources are allocated. Hence, performances from individual computing instances are unpredictable and below par compared with cheap personal hardware.

6.1 An alternative to the GAE/J

Another similar cloud computing platform gaining attention is the Amazon Web Services (AWS) Elastic Beanstalk (EB)[12]. Similar to GAE/J, AWS EB also allows automatic scaling of Java based web applications running within the standard Apache Tomcat[13] application servers. Beanstalk is part of the bigger AWS set of cloud computing offerings, and thus developers can use other services, e.g. Amazon's storage services, compute clusters, MapReduce, and so on. While the GAE/J severely limits the flexibility in developing applications, AWS EB provides comparable transparent scalability but not at the cost of flexibility and control.

7. CONCLUSION AND FUTURE WORK

This paper evaluated the Google App Engine for Java (GAE/J) PaaS cloud platform in a case study for implementing cryptographic protocols as the building blocks of a privacy preserving collaborative filtering scheme. The experimental results show that GAE/J in its current state is unsuitable for such applications, although with the planned support for MapReduce in the GAE/J better parallelism could be expected in future.

This case study also explicitly highlighted the problem that many theoretically feasible algorithms may not be practically feasible on exemplary cloud computing platforms. While translating theory to practice is not easy in any domain, highlighting this challenge in the context of cloud computing is especially important since the nature of the cloud lends itself to the misconception that anything is computable – the case study shows that one cannot blindly assume this.

Conducting further case studies of various application scenarios on other cloud computing platforms, and addressing implementation concerns of many theoretical privacy preserving models on real cloud computing infrastructure are avenues of future work.

8. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and Others. Above the clouds: A berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*, 2009.

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

[3] E.-O. Blass, R. D. Pietro, R. Molva, and M. Onen. Prism – privacy-preserving searches in mapreduce. Cryptology ePrint Archive, Report 2011/244, 2011.

[4] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina. Controlling data in the cloud: outsourcing computation without outsourcing control. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 85–90. ACM, 2009.

[5] C. Clifton, M. Kantarcioglu, X. Lin, J. Vaidya, and M. Zhu. Tools for Privacy Preserving Distributed Data Mining. *SIGKDD Explorations*, 4(2):28–34, Jan. 2003.

[6] I. Damgärd and M. Jurik. A Generalisation, a Simplification and Some Applications of Paillier's Probabilistic Public-Key System. In *Public Key Cryptography*, pages 119–136. Springer, 2001.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

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

[9] Y. Lu and G. Tsudik. Enhancing data privacy in the cloud. In *Proceedings of the 5th IFIP Trust Management conference (IFIPTM 2011), Copenhagen, Denmark*. Springer, 2011.

[10] I. Roy, S. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 20–20. USENIX Association, 2010.

[11] E. Shi, J. Bethencourt, T. Chan, D. Song, and A. Perrig. Multi-dimensional range query over encrypted data. 2007.

[11] The Python version of the GAE, however, supports SQLite. See: `http://goo.gl/nfu8e`.

[12] See: `http://goo.gl/Esg9A`.

[13] See: `http://goo.gl/FiSm`.