Protecting Organizational Data Confidentiality in the Cloud using a High-Performance Anonymization Engine

Vanessa Ayala-Rivera [1], Dawid Nowak [1], Patrick McDonagh [2]

[1] Lero@UCD, School of Computer Science and Informatics, University College Dublin.
vanessa.ayala-rivera@ucdconnect.ie; dawid.nowak@ucd.ie
[2] Lero@DCU, School of Electronic Engineering, Dublin City University.
patrick.mcdonagh@dcu.ie

Abstract

Data security remains a top concern for the adoption of cloud-based delivery models, especially in the case of the Software as a Service (SaaS). This concern is primarily caused due to the lack of transparency on how customer data is managed. Clients depend on the security measures implemented by the service providers to keep their information protected. However, not many practical solutions exist to protect data from malicious insiders working for the cloud providers, a factor that represents a high potential for data breaches.

This paper presents the High-Performance Anonymization Engine (HPAE), an approach to allow companies to protect their sensitive information from SaaS providers in a public cloud. This approach uses data anonymization to prevent the exposure of sensitive data in its original form, thus reducing the risk for misuses of customer information. This work involved the implementation of a prototype and an experimental validation phase, which assessed the performance of the HPAE in the context of a cloud-based log management service. The results showed that the architecture of the HPAE is a practical solution and can efficiently handle large volumes of data.

Keywords: Cloud Computing, SaaS, Data Confidentiality, Data Anonymization, Performance.

1 Introduction

In recent years, cloud computing [1] has become a popular business model and a promising technology paradigm for delivering IT services. This model offers multiple benefits to the diverse interests of customers and providers. For example, small and medium-sized enterprises (SMEs) find the cloud a potential area to expand their market and reach new clients. Consequently, enterprises from various domains are already using or planning to implement some type of cloud service in the near future [2, 3].

Despite these advantages, a large number of companies are still reluctant to use the cloud due to the associated risks of adoption [4]. According to the work presented in [5, 6, 7, 8], security continues to be the primary obstacle preventing the adoption of cloud services. In a time where data is a critical asset for many firms, data protection has become one of the top security concerns [9]. Therefore, companies are wary of losing control over their sensitive data by placing it on platforms that they do not manage.

Under these circumstances, customers depend on the cloud providers to have the appropriate security measures in place to protect the data. Nevertheless, the fact that customer data needs to be in a simple text format to be used, raises privacy concerns about potential attacks from malicious insiders working for providers. This threat is amplified due to the lack of transparency about providers' processes [10]. This situation is more concerning in the SaaS delivery model, because it offers the highest level of abstraction, hence offering the least visibility on how data is managed.

According to the 2009 Data Breach Study performed by the Ponemon Institute [11], over 44% of cases related to general data breaches in the industry were caused by third party contractors, partners and outsourced vendors. This and other similar worrying statistics [12] confirm companies' concerns about data security. These statistics should also encourage customers of cloud-based services to implement their own internal controls to compensate for the potential security deficiencies of providers.

Our work aims to facilitate customers in the implementation of their own methods to protect their critical information in-house, before uploading their data to the cloud. In this paper, we present the High-Performance

Anonymization Engine (HPAE), which is our proposed approach to protect the confidentiality of customers' data when using SaaS applications. As part of our work, we have implemented a prototype application and conducted a set of experiments to validate the feasibility of our approach.

The structure of this paper is as follows. Section 2 provides a review of the related work. Section 3 describes the proposed approach, including details of the implementation. Section 4 presents the results of the experimental validation. Finally, Section 5 presents the conclusions and future work.

2 Related Work

Data confidentiality has been an active research area in recent times as it remains a top concern for adoption of cloud computing model. As a result, many different approaches have been proposed to ensure data security in the cloud. One proposed solution is to simply avoid external clouds and build in-house private clouds. In this scheme, companies try to retain the advantages of the cloud model by employing private/hybrid cloud initiatives, hence avoiding the issues of public clouds [8]. However, this approach is expensive and not affordable for most companies.

Another alternative for data protection is to use traditional cryptography techniques to encrypt all cloud data. While this technique might be a good solution to protect data when it is transmitted or stored at the vendor side, it is not appropriate for data which is used for computation. The problem is that this technique highly restricts further data use, such as searching and indexing. Some state-of-the-art cryptography works have offered more versatile encryption schemes that allow operations upon and computation on the ciphertext [13, 14, 15]. However they are still too slow to be viable for real-world applications. Another encryption approach is Silverline [16], which identifies and encrypts all functionally encryptable data (any sensitive data that can be encrypted without limiting the functionality of the application in the cloud). However, the applicability of this approach is also limited because it assumes that web applications do not require access to raw data, which is rarely the case.

Our approach is closely related to the work described in [17], in the context of using data obfuscation to protect sensitive attributes. However, their solution requires cooperation from the service providers to implement logic on their side, situation which is not always feasible. Another approach related to our work is presented in [18], which also aimed to protect data from cloud service providers. Here, the authors describe three conditions to prevent that users' confidential information be collected by service providers. Firstly, separate software and infrastructure service providers. Secondly, hiding information about the owners of the data, and finally, the use of data obfuscation. Nevertheless, this flexibility is not always possible as it is often the case that the provider offering the software manages the infrastructure or platform service as well, so the user has no control over this.

3 Proposed Approach

The context of our approach is shown in Figure 1. In most companies, different data sources exist within the secure boundaries of the enterprise intranet. However, whenever an interaction occurs with a SaaS application, the company's data might leave the security of the intranet and be transferred to the SaaS provider. When this scenario arises, a preferable situation is that the information could be protected before being sent. For this purpose, companies can implement their own methods to protect their critical information in-house, before uploading their data to the cloud. This will keep the data safe from potential misuse by the service provider, while still retaining utility to be processed and analyzed. One technique that could be used for this purpose is data anonymization, which is the process of altering the original data in such way that it is difficult to infer anything private about the entities represented. This simultaneously limits the loss of information such that data is still meaningful permitting its analysis. Similarly, in some cases, users may need to access their original data. Once data is processed in the cloud, the output can then be returned to the enterprise secure intranet and a reversibility mechanism can be used to reveal the original values from the anonymized data. Our approach, the HPAE, aims to fulfil those responsibilities in Figure 1. The following sections describe in detail the components of our proposed architecture and implementation.

Finally, this work has the following goals:

- The development of an approach employing anonymization to protect confidential data on a public cloud. The objective is to protect customers' sensitive data, using the HPAE, from SaaS providers when data is processed in the cloud. Moreover, as the concept of confidentiality varies among users, our solution aims to be flexible enough to allow users to configure their privacy policies.
- The design of an architecture that efficiently handles large volumes of data offering high-throughput and fast processing. Performance is a determining factor in the adoption of a new approach, thus we aim to

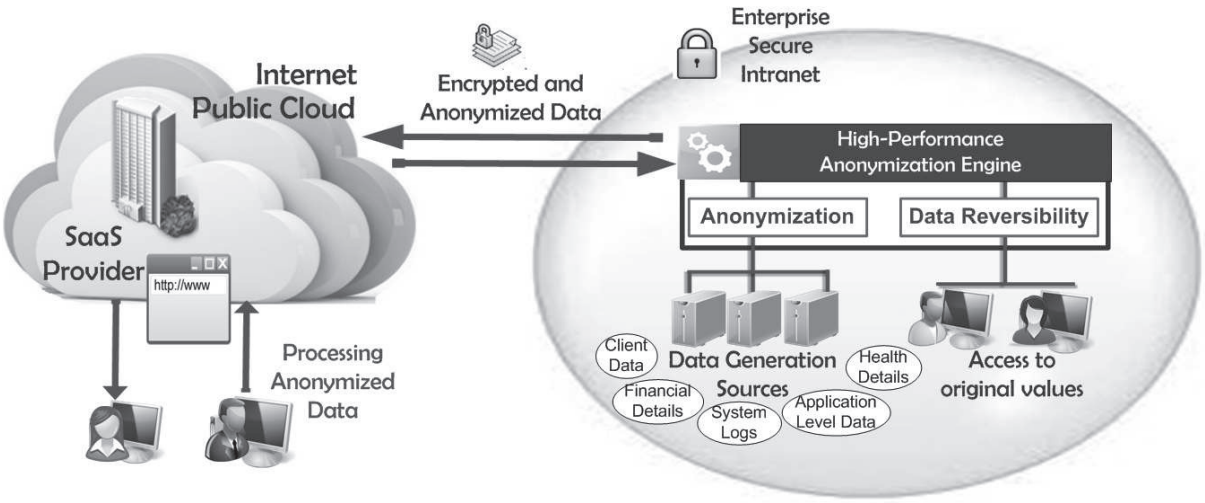


Figure 1: Contextual View of the HPAE

provide a solution that offers good performance (in terms of throughput) such that it is practical and allows users to anonymize data on-the-fly. Furthermore, the design should be modular and extensible to facilitate the accommodation of new components, such as new anonymization techniques, new input/output types, more efficient libraries/data structures etc.

- The development of a prototype tool in Java to demonstrate our approach and measure the performance of our architecture. To facilitate ease of integration of the HPAE with existing systems inside organizations, our Java implementation will provide support for various types of input data sources as well as various types of output destinations.

3.1 HPAE Architecture

This section describes the different components in the architecture of our prototype. The HPAE is the core component and it is composed by one-to-many *Data Processor* (DP) threads; although only a single DP is shown here, there may be as many DPs as input types as denoted in Figure 2. There are four stages in our approach: *Configuration, Reading, Anonymization* and *Writing*. The prototype tool for the HPAE is still currently being developed; more details about the implementation details are provided in Section 3.1.5.

3.1.1 Configuration

In this stage two files are configured: the *Engine Descriptor* and the *Rules Descriptor*.

The *Engine Descriptor* is an XML file containing the configuration parameters to initialize the HPAE and set values for the DPs. Among others, some of the parameters found in this file are: input sources/output destinations for DPs, the *Rules Descriptor* file for each DP and other technical properties like buffer size and the number of Event Handler threads to process the data.

The *Rules Descriptor* is an XML file where the rules to identify sensitive information are defined (i.e. IP addresses, emails, etc.). The rules are defined in the form of patterns that can be position based, expressions or occurrences of specific strings. One file can be configured for each input source.

Once the descriptor files have been configured, they are validated. The *Configuration Parser* is in charge of parsing the XML files and validating that parameter values are correct (i.e. positive numeric values, duplicate values, etc.). The output of this process is either; presenting a list of configuration errors that need to be fixed to the user, or passing the list of DPs to be run to the HPAE. The configuration of the descriptor files is currently performed manually as a *GUI* has yet not been implemented.

3.1.2 Reading

The components used in the reading process will correspond to the selected type of input. In order to minimize the effort required by organizations in making changes to their current systems (which can have different technologies implemented), the HPAE supports the most common input sources and output destinations. The HPAE reads and

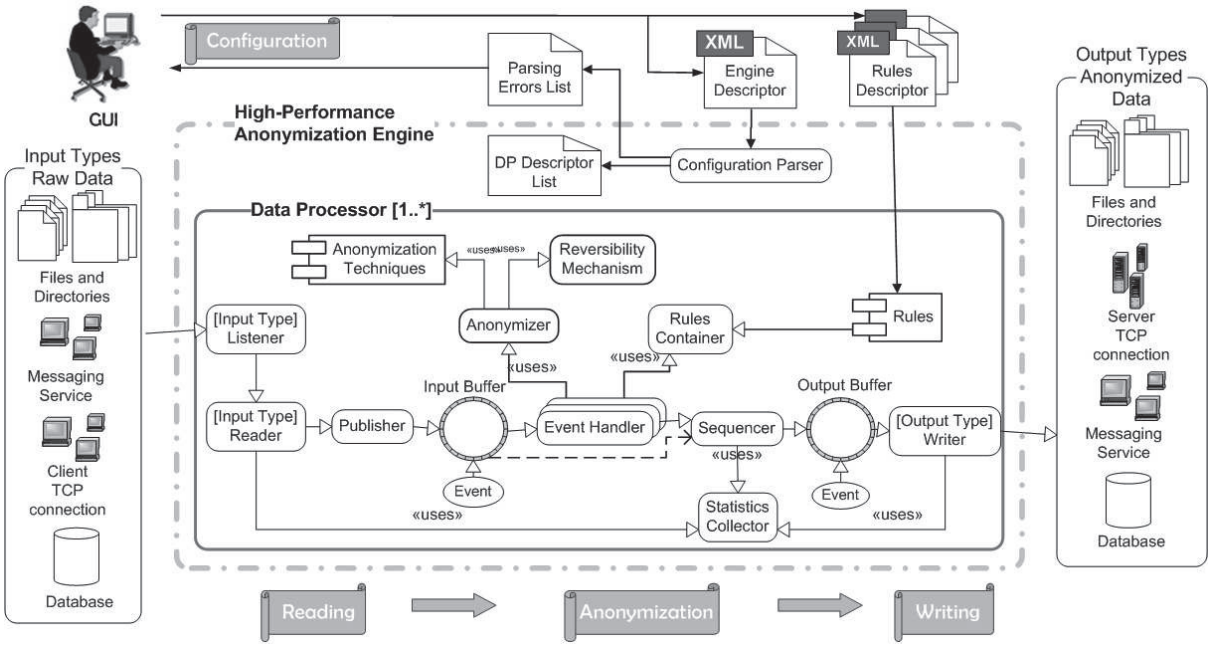


Figure 2: Technical View of the HPAE

writes data from/to databases, files, directories, TCP connections and messaging services like queues or topics. Support for the database type has yet not been implemented and remains as future work.

To start the data processing, the corresponding *Listener* waits for data to become available. For example, in the case of TCP connections, the *Listener* will wait until a connection is made to the specified port number. Once a client connection is accepted, the corresponding *Reader* will start receiving data from the socket. The received payload is encapsulated in an *Event* object, which can be a line of text in a file, a message from a queue/topic, a tuple from a database, etc. These *Events* are sent to the *Publisher*, which claims the next available slot in the *Input Buffer* and adds the entry. Buffers are circular data structures used to exchange the data between the different processing stages. These stages are asynchronously processed, meaning, one stage (i.e. Anonymization) does not need to wait until the previous one is fully finished (i.e. a file fully read) to start its processing.

3.1.3 Anonymization

The *Event Handlers* are the threads that fetch the *Events* from the *Input Buffer* and process them. The logic to publish entries to the buffer and retrieve them is based on the *Disruptor* pattern [19], explained in more detail in Section 3.1.5. Each *Event Handler* processes one *Event* from the buffer. To apply the anonymization, the *Event Handlers* use the *Rules Container* and the *Anonymizer*. The *Rules Container* has the patterns that are used to identify sensitive information. The *Anonymizer* contains the set of anonymization techniques to be applied. The *Event Handlers* perform the pattern matching; if a match occurs, the anonymization technique for that attribute is applied. Our prototype uses data substitution, extracting the sensitive attribute and replacing it with a new token in the *Event*.

As some anonymization techniques are not reversible, it is desired to have mechanisms that allow re-identification of the data regardless of the selected technique. In some scenarios, customers might require the original information, for example to perform root cause analysis of an incident investigation. The HPAE aims to provide various forms of *Reversibility Mechanism* to retrieve the original value such as keeping a translation table that tracks all the transformations done to the data or building dictionaries on-the-fly. This latter could also provide a consistent anonymization (using the same value mapping) across multiple data streams, which can be useful to correlate information from several sources. Currently, the *Reversibility Mechanism* remains part of our future work.

Since the HPAE can have multiple *Event Handlers* working in parallel, there is no guarantee on the order they will finish processing an *Event*. Thus, to ensure that the *Events* exit in the same order as they were received, the *Sequencer* is used. This component iterates the *Input Buffer* and retrieves the *Events* (once they have been anonymized) to publish them in the correct order to the *Output Buffer*.

3.1.4 Writing

Based on the output type defined for the DP, the corresponding *Writer* is created. This component retrieves the anonymized *Events* from the *Output Buffer* and sends them to the specified destination. As an optional step, if the user indicated in the *Engine Descriptor* file to gather performance statistics for the DP such as throughput, these are calculated by the *Statistics Collector*. In the current implementation, statistics are collected and written to a CSV file for analysis.

3.1.5 Implementation Details

Our prototype was implemented in Java because of its object-oriented nature, built-in support for multi-threading, portability and the practical benefits it possesses such as vast amounts of libraries. Furthermore, Java is one of the most widely used languages in enterprise applications [20], which can facilitate the adoption of our solution. To achieve high performance, the architecture of the HPAE was designed to execute three main phases (depicted in Figure 2) as independent processes (in separate threads). Given these characteristics, we investigated the use of different high performance libraries that we could apply to our design. In the end, we decided to use the *Disruptor* framework created by LMAX Exchange [19]. This framework offers data structures and a pattern of use (composed of producers, a ring buffer and consumers) that deal efficiently with concurrent programming, improving time and memory allocation compared to regular Java queue implementations that tend to suffer from write contentions. For example, in a pipeline configuration, instead of using a queue to exchange information between stages, which introduces latency, *Disruptor* uses a single lock-free data structure that deals with concurrent access. This framework fits well with our design as it is intended for an asynchronous event processing architecture like ours. Furthermore, our design is flexible enough to accommodate other types of data structures if needed.

3.1.6 HPAE for Log Management Service in the Cloud

One of the areas that suits the cloud model, in terms of configurable resource provisioning, is log management (LM) [21]. This kind of service successfully overcomes the main challenges faced by the SMBs: balancing the limited amount of in-house resources (people and infrastructure) against the ever-increasing supply of log data. However, log data contains aspects that can compromise the safety of the enterprise, thus it is recommended to protect confidential data when these logs are transmitted to a third-party such as cloud service providers.

The inherently sensitive nature of logs and the steady growth of data make LM an ideal application scenario for our approach, where our prototype is aimed at anonymizing sensitive data in the logs. For experimental purposes, we defined the IP address attribute as the pattern of sensitive data in the *Rules Descriptor* file, as this is the most commonly anonymized field in security and network relevant logs. For our initial prototype, the selected anonymization technique was data substitution, which consisted of matching the desired pattern in the logs and replace it with a token, which in our case was a Base64 encoded version of the original value. This technique was selected because it is fast in terms of processing time. This characteristic allowed us to minimize the overhead caused by the anonymization process and assess exclusively the feasibility of our approach.

4 Experimental Evaluation

This section describes the experiments aimed to assess the performance of the HPAE. The experiments were performed on a machine running 64-bit Windows 7, with an Intel Core i7 processor (4 cores / 8 threads) at 2.0 GHz clock speed with 6 GB RAM using Java HotSpot 1.6.0_31. The first experiment validated the selection of the *Disruptor* framework for our implementation, while the second experiment validated the performance of the HPAE prototype.

4.1 Experiment 1: Disruptor Performance Testing

The objective here was to validate the efficiency of *Disruptor* against Java's *ArrayBlockingQueue* (identified by LMAX as the Java bounded queue with the best performance [22]) to determine if *Disruptor* could be a better structure for data transfer than the traditional queues. This test, conducted in two phases, involved the assessment of two different metrics: throughput and latency. All the tests used in this experiment were taken from *Disruptor* version 2.8 open source project available at [19].

Phase 1 - Throughput Performance Testing.

The aim of this test was to compare the throughput offered by *Disruptor* and *ArrayBlockingQueue* when given three different configurations. The topologies that were selected for this experiment were the ones found within

Table 1: Throughput Comparison between Disruptor and ArrayBlockingQueue

Configuration	Data Structure	Throughput (Operations per Second)		
		Best case	Average case	Worst case
Unicast (1 P - 1 C)	ArrayBlockingQueue	3,424,305	3,340,106	3,256,586
	Disruptor	51,921,079	47,324,781	45,641,259
Multicast (1 P - 3 C)	ArrayBlockingQueue	614,352	592,084	564,174
	Disruptor	64,432,989	33,724,278	23,036,166
Pipeline (1 P - 3 C in stages)	ArrayBlockingQueue	1,757,716	1,737,686	1,721,763
	Disruptor	40,617,384	30,688,199	20,777,062

Table 2: Latency Comparison between Disruptor and ArrayBlockingQueue

Data Structure	Latency (Nanoseconds)		
	Mean	Max	99% less than
ArrayBlockingQueue	6,750	6,142,182	8,192
Disruptor	25	397,500	2

our architecture: Unicast, which consists of one producer (P) to one consumer (C); Multicast, which consists of one producer to multiple consumers (3 in our test) and Pipeline, which consists of a chain of one producer to multiple consumers where each consumer depends on the output of the previous consumer.

Table 1 shows the throughput results, in terms of operations per second, for 5 runs processing 100 million messages. The results reported in this table are for the best, worst and average case scenarios. These results demonstrate how *Disruptor* has a greater throughput performance than *ArrayBlockingQueue* for all cases. For example in the average case, *Disruptor* is 14 times better in the Unicast configuration; similarly, *Disruptor* is 56 times better in the Multicast configuration and 17 times better in the Pipeline one.

Phase 2 - Latency Performance Testing.

The aim of this testing was to compare the latency introduced by *Disruptor* and *ArrayBlockingQueue* for a configuration of three stage pipeline. Entry events were generated and injected in intervals of 1 microsecond to prevent saturation, repeating this process 50 million times. Table 2 shows the latency performance results for *Disruptor* and *ArrayBlockingQueue*. We can observe that *Disruptor* outperforms the queue implementation in all comparisons. For example, in the 99% of observations the latency of *Disruptor* was minimal (only 2 nanoseconds), while *ArrayBlockingQueue* took more than 8,000 nanoseconds.

To summarize the results of Experiment #1, they demonstrated that *Disruptor* offers better throughput and lower latency than the best Java Queue class (*ArrayBlockingQueue*). Therefore, it was decided to use *Disruptor* as part of our prototype implementation.

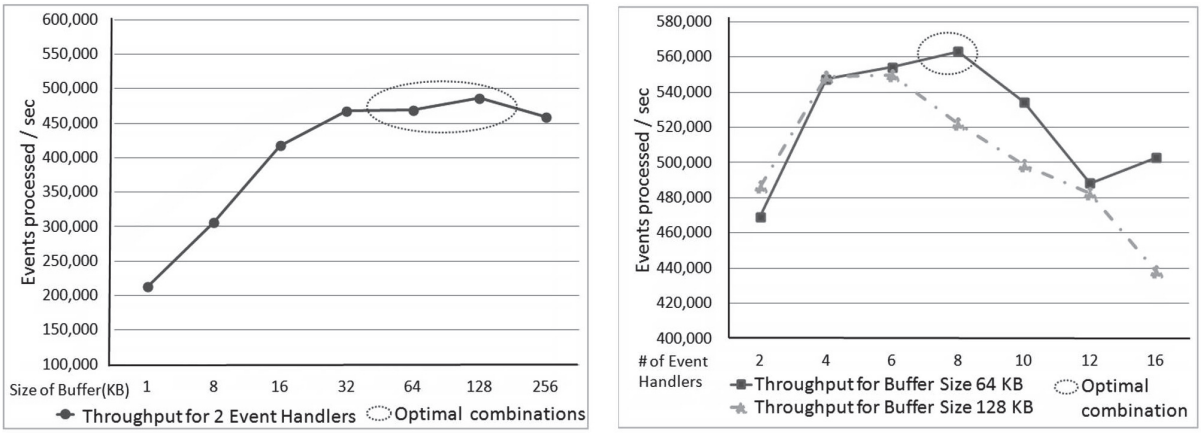
4.2 Experiment 2: HPAE Performance Testing

The objective of this experiment was to assess the efficiency of our architecture and implementation. We conducted a series of performance tests divided in two phases: The first phase focused on finding the optimal set of configuration parameters, while the second phase focused on the evaluation of the throughput using that configuration. For these tests, the HPAE was set up to run one socket DP and receive data from a client TCP connection in an isolated network. A second laptop was used for the client connection which has the same characteristics of the machine running the HPAE.

Phase 1 - Find the optimal parameters for HPAE: Size of Buffers and Number of Event Handlers.

The goal of this phase was to find the optimal combination of parameters for the HPAE, which maximized the throughput, measured in events processed per second (eps).

Figure 3 shows the results for the throughput performance tests for 5 runs processing 70 million log events for different combinations of parameters. In Figure 3 (a) we searched for the optimal size for the buffers, thus this was the variant parameter and the number of *Event Handlers* was set to 2. Given a uniform workload, we can see that when the size of the buffer is incremented, the throughput also increases until it reaches the maximum point at 128 KB. After this, the HPAE reaches a steady state (experiencing queuing delay in a saturated buffer) where the throughput slightly decreases. As a result of this experiment, we took sizes of 64 and 128 KB as the optimal values. Using these two values as constants, we then investigated the optimal number of *Event Handlers*, thus this was the variant factor in a second experiment. In Figure 3 (b) we can observe that using the buffer size of 64 KB, the throughput rate increases as the number of *Event Handlers* is augmented until reaching the maximum point



(a) (b)

Figure 3: Throughput Performance Testing: (a) Finding the optimal buffer size; (b) Finding the optimal number of Event Handlers.

Table 3: Results for the HPAE performance testing

Measure	Average	Max
Throughput (eps)	429,514	563,066
CPU usage (%)	33.44	34.5
Memory usage (MB)	85	102

(around 560,000 eps) at 8 *Event Handlers*. For the buffer size of 128 KB, the throughput reaches its peak at 6 *Event Handlers* with a maximum throughput of around 550,000 eps. The results of this phase demonstrated that the optimal configuration parameters for a DP are 64 KB for buffer size and 8 threads of *Event Handlers*. Given the characteristics of the machine used for the experiments, having beyond 6-8 *Event Handlers* causes performance degradation and scheduling overhead, moreover, not all threads would be actively running.

Phase 2 - Throughput Performance Testing using Optimal Parameter Combination.

The goal of this phase was to carry out a first round of performance testing of our architecture using the optimal combination obtained in Phase 1. The data load generated by the client was 70 million event logs containing network details. The IP address was the data pattern defined as sensitive information in the *Rules Descriptor* file and the anonymization technique applied was Base64 encoding. In order to reduce the cost of the output operation, the anonymized output stream was not stored. Table 3 shows the results for the measured metrics: memory usage, CPU usage and throughput.

The results of this phase showed that the HPAE can achieve a high throughput (more than 400,000 eps, reaching a peak of more than 560,000 eps) with low resources (less than 35% CPU usage and a maximum of 102 MB of memory) using a relatively modest test machine.

5 Conclusions and Future Work

Security remains to be the primary obstacle preventing the adoption of cloud services, inside which data protection is one of the top concerns.

In this paper, we presented the HPAE, a practical approach that will enable organizations to implement their own controls to protect sensitive information from service providers in a public cloud environment. We demonstrated that the approach is practical by implementing a prototype and then validated its feasibility to efficiently handle large volumes of data. In our experiments, we achieved a throughput of more than 560,000 eps, using less than 35% of CPU and only 102 MB of memory.

Future work will focus on the integration of the remaining elements of our approach into our prototype, as well as extending the number of available anonymization techniques with a special emphasis on assessing the performance overhead they might introduce to the system. Furthermore, high performance will continue to be a key objective, thus investigating other possible mechanisms to increase throughput and reduce response time will also be part of future work. Other interesting aspects would be to automate the identification of sensitive information. This could be achieved by using self-learning algorithms that discover new data patterns on-the-fly, hence facilitating the automatic configuration of privacy policies.

Acknowledgements

This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero - the Irish Software Engineering Research Centre (www.lero.ie). This work has also received support from Science Foundation Ireland (SFI) via grant 08/SRC/I1403 FAME SRC (Federated, Autonomic Management of End-to-End Communications Services - Scientific Research Cluster).

References

[1] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," 2011. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf

[2] Deloitte, "CIO Survey Report 2012 Leading the transformation to a digital future," 2012. [Online]. Available: http://www.deloitte.com/assets/Dcom-Ireland/LocalAssets/Documents/Consulting/IE_Deloitte_CIO_Survey_2012.pdf

[3] J. G. Harris and A. E. Alter, "Cloud Rise: Rewards and Risks at the Dawn of Cloud Computing," 2010. [Online]. Available: http://www.accenture.com/SiteCollectionDocuments/PDF/Accenture_Cloudrise_Rewards_and_Risks_at_the_Dawn_of_Cloud_Computing.pdf

[4] K. Kessinger and M. Gellman, "2010 ISACA IT Risk/Reward BarometerUS Edition," 2010. [Online]. Available: http://www.isaca.org/About-ISACA/Press-room/News-Releases/2010/Documents/2010_ISACA_Risk_Reward_Barometer_Results_US.pdf

[5] F. Gens, "New IDC IT Cloud Services Survey: Top Benefits and Challenges," 2009. [Online]. Available: http://blogs.idc.com/ie/?p=730

[6] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing," *Journal of Network and Computer Applications*, vol. 34, pp. 1–11, Jan. 2011.

[7] M. Zhou, R. Zhang, W. Xie, W. Qian, and A. Zhou, "Security and Privacy in Cloud Computing: A Survey," *6th International Conf. on Semantics, Knowledge and Grids*, pp. 105–112, Nov. 2010.

[8] R. Chow, P. Golle, and M. Jakobsson, "Controlling data in the cloud: outsourcing computation without outsourcing control," in *ACM Workshop on Cloud Computing Security*, Chicago, IL, 2009.

[9] iSMG, "Overcoming the Apprehension of Cloud Computing," 2012. [Online]. Available: http://docs.ismgcorp.com/files/handbooks/Cloud-Survey-2012/Cloud_Survey_Report_2012.pdf

[10] Cloud Security Alliance, "Top Threats to Cloud Computing," 2010. [Online]. Available: https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

[11] Ponemon Institute, "2009 Annual Study: Global Cost of a Data Breach," 2009. [Online]. Available: http://www.securityprivacyandthelaw.com/uploads/file/Ponemon_COB_2009_GL.pdf

[12] S. et al., "Data Breach Trends & Stats," 2012. [Online]. Available: http://www.indefenseofdata.com/data-breach-trends-stats/

[13] D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *IEEE Symposium on Security and Privacy*. IEEE, 2000, pp. 44–55.

[14] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," *Advances in Cryptology Eurocrypt*, no. 3027, pp. 506–522, 2004.

[15] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. thesis, Stanford University, 2009.

[16] K. Puttaswamy, C. Kruegel, and B. Zhao, "Silverline: toward data confidentiality in storage-intensive cloud applications," 2011.

[17] M. Mowbray, S. Pearson, and Y. Shen, "Enhancing privacy in cloud computing via policy-based obfuscation," *The Journal of Supercomputing*, pp. 267–291, Mar. 2010.

[18] S. S. Yau and H. G. An, "Protection of users' data confidentiality in cloud computing," *Proceedings of the Second Asia-Pacific Symposium on Internetware*, pp. 1–6, 2010.

[19] LMAX-Exchange, "LMAX Disruptor High Performance Inter-Thread Messaging Library." [Online]. Available: http://lmax-exchange.github.com/disruptor/

[20] L. Dignan, "Software development budgets on the rise, study finds." [Online]. Available: http://www.zdnet.com/software-development-budgets-on-the-rise-study-finds-3040092512/

[21] K. Kent and M. Souppaya, "Guide to Computer Security Log Management," 2006. [Online]. Available: http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf

[22] M. Thompson, D. Farley, M. Barker, P. Gee, and A. Stewart, "Disruptor: High performance alternative to bounded queues for exchanging data between concurrent threads," 2011. [Online]. Available: http://disruptor.googlecode.com/files/Disruptor-1.0.pdf