MASCARA (ModulAr Semantic CAching fRAmework) towards FPGA acceleration for IoT Security monitoring

van Long Nguyen Huu, Julien Lallet, Emmanuel Casseau, Laurent d'Orazio

MASCARA (ModulAr Semantic CAching fRAmework) towards FPGA acceleration for IoT Security monitoring

Van Long Nguyen Huu [AB], Julien Lallet [A], Emmanuel Casseau [B], Laurent d'Orazio [B]

[A] Nokia Bell Labs, Lannion, France, {long.nguyen_huu, julien.lallet}@nokia.com
[B] Univ Rennes, CNRS, IRISA, Lannion, France, {emmanuel.casseau, laurent.dorazio}@irisa.fr

Abstract

With the explosive growth of the Internet Of Things (IOTs), emergency security monitoring becomes essential to efficiently manage an enormous amount of information from heterogeneous systems. In concern of increasing the performance for the sequence of online queries on long-term historical data, query caching with semantic organization, called Semantic Query Caching or Semantic Caching (SC), can play a vital role. SC is implemented mostly in software perspective without providing a generic description of modules or cache services in the given context. Hardware acceleration with FPGA opens new research directions to achieve better performance for SC. Hence, our work aims to propose a flexible, adaptable, and tunable ModulAr Semantic CAching fRAmework (MASCARA) towards FPGA acceleration for fast and accurate massive logs processing applications.

Type of Paper and Keywords

Vision paper: *hardware acceleration, semantic caching, security monitoring, IoT*

1 Introduction

Recently, the Internet of Things (IoT) is strongly growing from industries to society. A myriad of IoT devices are currently used in smart homes, smart campus, or even smart cities with numerous benefits and also the vulnerabilities or unprecedented security challenges. Thus, cybersecurity is essential to protect the various IoT ecosystems against increasingly cybercriminals. Within the cybersecurity activities, security monitoring as an instance of a Big Data context aims to observe and detect continuously the unusual events from an enormous amount of distributed system logs. Unfortunately, fine-grained re-usability is not addressed well in Big Data technologies. As an example, Figure 1 shows the data flow from IoT devices to a HTTP server where security monitoring is maintained by the activities of big data reasoning and visualization.

Query caching, in which the queries and/or the query results are stored in a cache system with query relevance management, is a crucial service to improve the performances of long term historical data processing or avoiding unnecessary query re-executions in security monitoring. The query-based monitoring tools reduce time-consuming of CPU to investigate the unusual events or harmful intrusions due to a well designed caching system [2]. A query cache uses semantic organization, called Semantic Query Cache (SQC) [12] or Semantic Caching (SC), is *flexible* and *can keep the relevance of information*. The most advantages of SC are allowing data processing and analysis effectively, reducing data transmission on servers, exploiting query relevance themselves [12], [19], [21], [7]. However, besides the expressiveness gain, the query processing of SC could induce non-negligible overhead [16] due to its complexity.

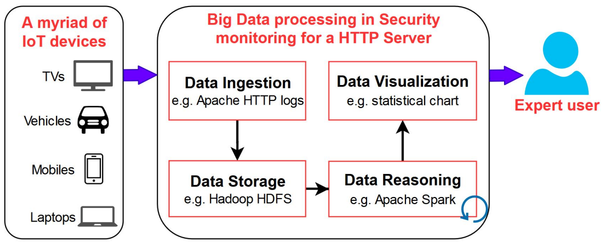


Figure 1: Overview of security monitoring for the IoT

Researches from [10] or [3] proposed the component interfaces, adaptable services or abstracting functions to facilitate developing a new cache solution from existing adaptable techniques. According to these approaches, the performance indicators related to hardware acceleration opportunities could be explored. However, they do not face directly with the SC framework implementation.

From the hardware perspective, data processing with hardware acceleration has been well presented in terms of Graphic Processing Unit (GPUs) implementation [31], [17] or more recently on Field Programmable Gate Arrays (FPGAs) [30], [26] thanks to its data throughput performance and low energy effort. Their contributions are limited with a dedicated hardware architecture and processing model that *depend heavily on the context characteristics* because of the lack of modular or service-oriented approach as a software perspective.

In our previous works, we took into account the SC framework towards FPGA acceleration [11], [22]. In [11], we proposed the vision of an abstract and tunable SC framework with multi-layer implementation. In [22], we continued with this vision and proposed a prototype where Apache Spark [1], Hadoop HDFS [23] are used to build a semantic cache prototype. Unfortunately, we have not yet presented any associated experiment of query processing within this prototype except for data communication between host and FPGA. Thus, the experiment in [22] is not complete to examine the performance indicators and point out the opportunities of SC framework acceleration with FPGA. Moreover, both works [11] and [22] did not present the mechanism of query processing as well as semantic management.

Therefore, our motivation is providing an appropriate solution like SC framework that not only maintains the *adaptable, flexible, and tunable* cache but also takes advantage of computing acceleration of hardware perspective. Our solution is called *MoudlAr Semantic CAching fRAmework (MASCARA)* towards FPGA acceleration directly aims to security monitoring application for IoT. Compared with these works in [11] and [22], MASCARA addresses an entire architecture of semantic caching which contains Query processing and Semantic management in detail. On one hand, MASCARA presents a modular architecture with different services to serve dedicated tasks in semantic query processing. On the other hand, it pointed out the opportunities of FPGA acceleration with high-level experiments for a given data store and execution environment.

Although FPGA and GPU acceleration for data processing has many issues to compare and analysis, we do not consider and express these issues as our research objective. We focus on FPGA acceleration because FPGAs have been noted by the database community as a data processing solution for their high parallelism, reconfigurability, and low power consumption, e.g. OpenCAPI [25], DoppioDB [31]. Thus, it can mitigate the computational pressure in the CPU to accelerate database analytic performance. In particular, with MASCARA, the circuit in FPGA allows us to customize only the semantic cache specific function, e.g. *query processing*, instead of designing as a generic task on CPU that do not use all the resources efficiently. Moreover, the modular approach allows us to extend, deploy MASCARA as a micro-service container for other related researches by using the orchestration platforms in the context of cloud computing services, such as Docker, Kubernetes, etc.

The remainder of this paper is organized as follows. In Section 2, we describe in detail the modules of MASCARA framework. Then, in section 3, we present our experimental results for MASCARA with different scenarios at high-level deploying on Massively Parallel Processing (MPP), i.e. Apache Spark and distributed file system based solution, i.e. Hadoop HDFS. Through the experiments at a high level, we prove that our MASCARA is the right approach for security monitoring application towards FPGA acceleration. Finally, in section 4, we conclude the proposed contribution and discuss several preliminary opportunities to accelerate the performance of MASCARA using FPGA.

2 MASCARA FRAMEWORK

The proposed MASCARA framework is based on a modular semantic caching approach which aims to exploit FPGA acceleration for large scale logs analysis with a given data store and execution environment. MASCARA presents two main contributions: *1) Abstract but comprehensive modular structure with different services to serve dedicated tasks in semantic query processing. These modules address how a user query is broken, trimmed, extracted and managed in terms of Query processing and Semantic management; 2) FPGA acceleration opportunities are pointed out with high-level experiments considering Select Project Join (SPJ) queries with aggregation.*

Figure 2 illustrates an overview of MASCARA architecture where *Query analyzing engine* and *Distribute file system* are described abstractly to be easily tuned to match the requirements of the considered application. In particular, HDFS could be used as a distributed storage solution. Meanwhile, log analysis can rely on Spark engine.

The general concept of SC could be divided into 2 parts: *Query processing* and *Semantic cache management* [27]. Due to this design principle, our

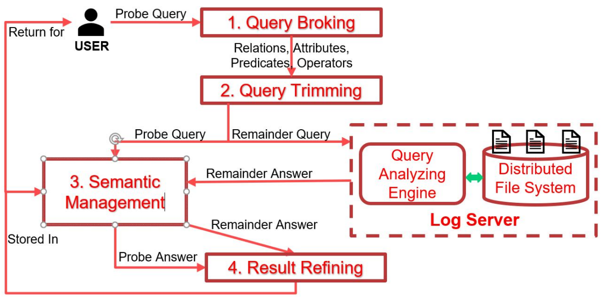
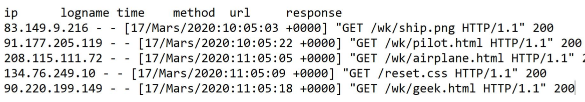


Figure 2: An overview of MASCARA architecture

MASCARA framework is composed of 4 main stages: *(1) Query Broking*, *(2) Query Trimming*, *(3) Semantic Management*, and *(4) Result refining*. The *Query Trimming*, or called Query Processing, is an attractive study in terms of sastifiablity [34], or automatically executed queries [33], etc. The modules in stages of MASCARA could be easily replaced, removed, composed, decomposed, or even delegated to low level such as hardware acceleration according to desirable context. Moreover, in each stage, we present the appropriate and flexible library services to serve different module activities.

Along with this proposal, we also state an example through HTTP big log analysis in which the monitoring security tool frequently supports Select Project Join (SPJ) queries with aggregation (illustrated in Figure 3). These kinds of activities are a part of Big Data issue where efficiently collecting, storing and analyzing the data from a huge amount of heterogeneous systems of the IoT is required.



```
ip      logname time    method  url     response
83.149.9.216 - - [17/Mars/2020:10:05:03 +0000] "GET /wk/ship.png HTTP/1.1" 200
91.177.205.119 - - [17/Mars/2020:10:05:22 +0000] "GET /wk/pilot.html HTTP/1.1" 200
208.115.111.72 - - [17/Mars/2020:11:05:05 +0000] "GET /wk/airplane.html HTTP/1.1" 200
134.76.249.10 - - [17/Mars/2020:11:05:09 +0000] "GET /reset.css HTTP/1.1" 200
90.220.199.149 - - [17/Mars/2020:11:05:18 +0000] "GET /wk/geek.html HTTP/1.1" 200
```

Figure 3: A log data pattern from Apache HTTP server

2.1 Query Broking

Query Broking is responsible to identify the user posed queries are in *Simple Format* or in *Complex Format* as illustrated in Figure 4.

Simple Format query is *selection* or even *projection* over single relation. An example of *SELECT - PROJECT* query is illustrated below with the reduced version of two log files $User < ip, time, date, idcam >$ and $Campus < idcam, name, stage >$.

```
Q1: SELECT ip, time FROM Log
WHERE ip >= '192.168.1.0' AND
ip <= '192.168.255.255'
```

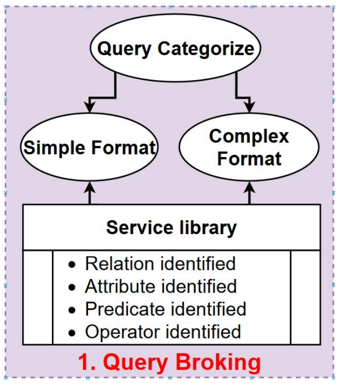


Figure 4: Modules and services of Query Broking

Complex format query may be *join* queries with no additional selection predicates that are not well suited for SC. In this case, the cache should manage more than one relation which effects directly in cache performance. Besides *join* query, *ordered* or *top-n* query could be considered as Complex format query [16]. An example of a join query with no selection on either relation is illustrated below:

```
Q2: SELECT *
FROM User JOIN Campus
ON User.idcam = Campus.idcam
```

The appropriate services would be called to materialize the query. In particular, the query would be decomposed into various types of elements, such as: *Relations*, *Attributes*, *Predicates* and *Operators*. These elements and their relationship would be examined and managed in the form of a hierarchy list or plan tree in the next stage (2) Query Trimming [20].

2.2 Query Trimming

The concept of *Query Trimming* that is firstly presented in [19] in which a query would be turned into two disjoint parts: the overlapped part and the non-overlapped part. Later, in [9], the two disjoint parts are further called: *probe query* and *remainder query*, respectively. Unlike Page or Tuple cache, thanks to probe and remainder query, besides HITS and MISSES, SC proposes PARTIAL HITS which is an unique and impressive advantage of SC [12]. It reduces waiting time to get the answer because the probe query can be processed in the cache while the remainder query has to be sent to the server. Probe and remainder query have been built by extracting explicitly or implicitly the semantic information from user query. This Query Trimming stage can be divided into two sub steps that have a strong relation: *Query Matching* and *Query Trimming*.

For instance, assume that the semantic cache stores a semantic entity *se*: $(date <' 17/03/2020 - 10 :$

$00 : 00')$. The incoming query *incq:* $(ip =' 208.115.111.72')$ would be divided into the probe query *prq:* $(date <' 17/03/2020 - 10 : 00 : 00' \wedge ip =' 208.115.111.72')$ and the remainder query query *req:* $(date \neq' 17/03/2020 - 10 : 00 : 00' \wedge ip =' 208.115.111.72')$

The *Sub Query engine* in stage (2) of MASCARA is designed with the concept of [9] to divide and manage a list of sub-queries with a given query equivalence management schema (illustrated in Figure 5). This schema, called *Query Matching*, is used to check the satisfiability or implied relationship between the queries and/or semantic fragment. Solving this problem in an appropriate time constraint is the key point for effectively *Query Trimming* stage. Hence, a lot of works has been done in this context, such as: *Relational algebra* [8], *Set Theory* [24], or *Satisfiability Modulo Theories - SMT* recently [34]. In our MASCARA, we use *Relational algebra* and *Set Theory* with several rules and constraints to express Query Matching in terms of a matching series: *Relation Matching*, *Attribute Matching* and *Predicate Matching*. The matching procedure is an iteration or even nested-iteration through list of candidate semantic information. Thus, it will take time to execute and get the matching conclusion on CPU, especially when the time complexity of query processing increases, i.e. complexity of semantic information needs to be decomposed in user query.

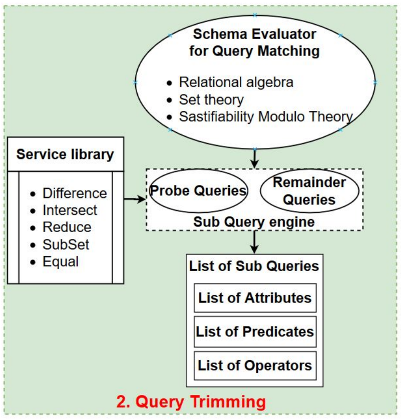


Figure 5: Modules and services of Query Trimming. Schema Evaluator is used for Query Matching

The information (attributes, relations, predicates...) of probe or remainder query would be turned in form of a semantic tuple $Q < Q_R, Q_A, Q_P, Q_C >$ as presented in [27] with A is Attributes, R is Relations, P is Predicates and C is number of result tuples. For instance, using the Simple Format query $Q1$ in Section 2.1, $Q1$ could be presented as $Q1 < Log, \{ip, time\}, (ip \geq' 192.168.1.0', ip \geq' 192.168.255.255'), \varnothing >$. Before Q1 gets answer, it contents is empty, then $Q_C = \varnothing$

Service library composes different computations (intersect, different...) to find out the query relevance. Currently, we are working on Select Project Join (SPJ) queries although MASCARA can be extended to handle other more complicated or aggregate queries.

2.3 Semantic Management

There are several ways to physically store semantic information of each query, called a semantic segment. To start with, we choose to store semantic segment as a basic tuple $S < S_R, S_A, S_P, S_C >$ and associates every segment with a semantic region in form of a list of the corresponding records [9] or linked pages [27]. This approach works fine for memory caching or even disk caching with a minimum of space overhead.

In stage (3), the *Semantic Management* is composed of two parts: *Semantic Description Table* and *Semantic Data Region* (illustrated in Figure 6). For allocation, if there is enough free memory to hold the semantic region, then store the semantic segment into Semantic description table and the relevant query results into Semantic data region at the same time. For de-allocation, remove the semantic segment and clean the relevant semantic data region.

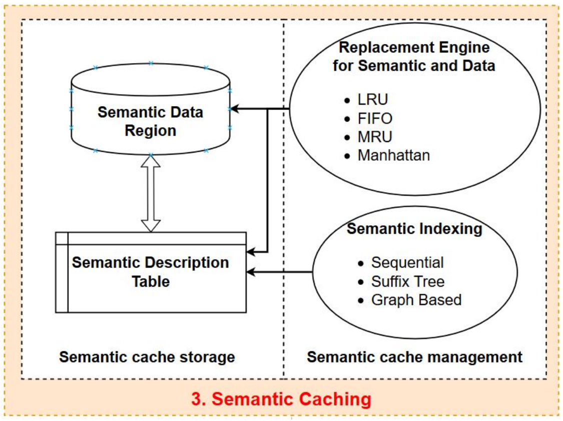
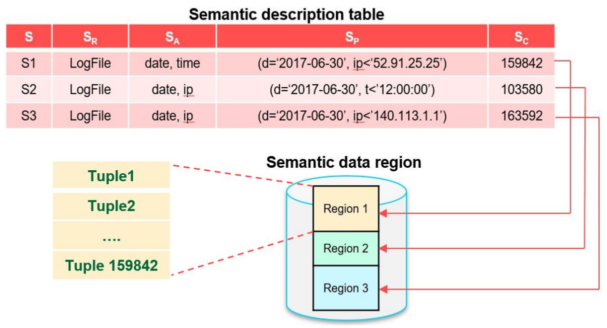


Figure 6: Modules and services of Semantic Management

There are two implemented engines at this stage (3): *Replacement Engine* and *Semantic Indexing Engine*. SC is usually allocated with a fixed amount of storage for holding items. When this storage space fills up (both for Semantic description table and Semantic data region), the *Replacement Engine* is responsible to determine one or more items to evict to make free space for newly referenced items based on various policies, such as: *Least Recently Used (LRU)*, *First In First Out (FIFO)*, *Most Recently Used (MRU)* or *Semantic locality* with


Semantic description table

S	S_R	S_A	S_P	S_C
S1	LogFile	date, time	(d='2017-06-30', ip<'52.91.25.25')	159842
S2	LogFile	date, ip	(d='2017-06-30', t<'12:00:00')	103580
S3	LogFile	date, ip	(d='2017-06-30', ip<'140.113.1.1')	163592

Figure 7: Association between Semantic Description Table and Semantic Data Region

Manhattan distance [9]. Time consumption of finding, removing, or replacing policies the semantic segment and data are strongly influenced by different strategies [28], [16], such as sequential, suffix tree or graph-based indexing are described in *Semantic Indexing Engine*.

Figure 7 illustrates the semantic segments $S1$, $S2$, $S3$ are stored in *Semantic Description Table* after continuously queries $Q1$, $Q2$ and $Q3$ as log query analysis. The last column S_C provides a number of query results in terms of the data region. Query results could be managed in terms of *tuple*, *chunk* or *dynamic data structure*. The implementation of the semantic description table could be easily extended with other parameters, such as time for replacement, dirty mark for data overwrite, etc. To avoid redundant information in semantic tuples, we do not allow saving and keeping existed semantic information through decomposition and coalescence with two following options: *Partial Coalescence in Query* and *Partial Coalescence in Segment*. MASCARA already implemented *Partial Coalescence in Query* where only the different part of semantic information from user query is saved in *Semantic Description Table*.

2.4 Result Refining

Result Refining receives the result of remainder queries from server and results of probe queries from SC (illustrated by Figure 8). While every individual segment may contain only a small part of a query result, multiple segments can be combined together to generate a much bigger part of, or even the whole result. Thus, *Result Refining* is the process in which an entire cache or a list of candidate segments is considered to contribute the final result. Therefore, to keep track with the relationship between every part of the query and the involved semantic segments, a *binary tree* is constructed *simultaneously* with the *Query Trimming* procedure. Finally, it can combine or merge the results into the final result as query answer. This final result could be sometimes updated back into the semantic data region if required.

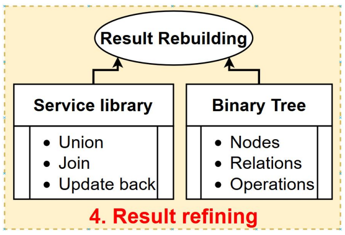


Figure 8: Modules and services of Result Refining

3 EXPERIMENTS

In this section, we first describe the environment for preliminary experiments, the system parameter settings. Then, based on the results, we present the performance of MASCARA compared to No Cache and Exact Matching Cache. Finally, through the experiment, we can analyze and point out the FPGA acceleration opportunities with the help of several performance indicators, e.g. response time of query processing.

3.1 Experiment setting

Spark [1] is a general-purpose cluster computing engine with libraries for streaming, graph processing, machine learning, and SQL. We use Spark SQL [32], a Spark module for structured data processing, which provides Data Frame API to perform relational operations on both external data sources and Spark's built-in distributed collections. The *Semantic Management* of MASCARA is based on Spark caching mechanism with multiple storage (memory only, disk only, or both).

At the same time, we use HDFS to store a massive number of logs because it is suitable for applications that have large data sets with high throughput access [23]. The implementation of MASCARA framework on Spark and HDFS is illustrated by Figure 9 with *Query Trimming* stage could be accelerated by offloading to FPGA.

The server used for the evaluation is based on an Intel® Core™ i7-6700 CPU to run at 3.40GHz with 32GB of RAM. The operating system is a Linux Ubuntu 14.04.6 Long Term Support distribution based on kernel 4.4.0-31-generic. We use Spark version 2.4.5 Pre-build for Apache Hadoop 2.7.

3.2 Preliminary performance study

To show the performance of MASCARA, we want to examine the Response Time with 6 tests in which

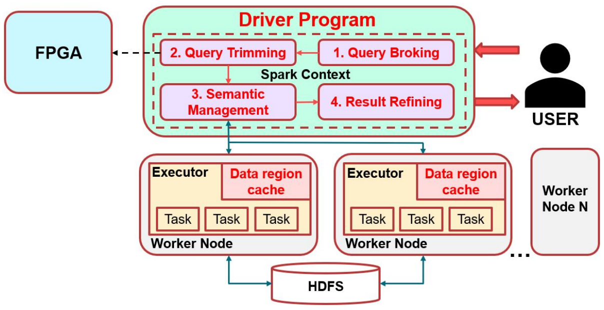


Figure 9: MASCARA modules implemented on Spark and HDFS towards FPGA acceleration

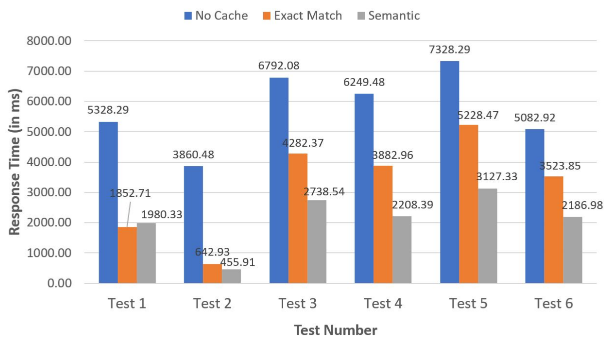


Figure 10: Response Time of 6 scenario tests

ratios of HITS (H), MISSES (M), or Partial HITS (PH) are changed (illustrated in *Table 1*). The Response Time consists of Query Processing Time (Query Broking, Query Trimming, Result Refining) and Semantic Management. In Semantic Management, the time of data transferring from storage to cache when MISSES or PARTIAL HITS is very expensive. We compare our MASCARA with NO Cache and Exact Match Cache. The Exact Match Cache is built based on Spark cache. When the contents of the two queries are the same, we have HITS. In contrast, with a minor difference in query content, we have MISSES. Thus, we call it the Exact Match Cache.

Table 1: Scenario tests to measure Response Time

	No cache	Exact Match	Semantic
Test 1	100% M	100% H	100% H
Test 2		70% H, 30% M	70% H, 20% PH, 10% M
Test 3		60% H 40% M	60% H, 30% PH, 10% M
Test 4		50% H, 50% M	50% H, 40% PH, 10% M
Test 5		40% H, 60% M	40% H, 50% PH, 10% M
Test 6		30% H, 70% M	30% H, 60% PH, 10% M

Each test contains 100 *SELECT-PROJECT* queries to apply with a HTTP log file 2.5 GBytes. Each query composes SELECT condition which is a disjunction (OR) of conjuntions (AND) of compare predicates. We use 10 warm-up queries, that are not measured in our experiment, to initiate data into cache. The cache replacement strategy, Least Recently Used (LRU), is applied for both caches and they can store a maximum of 1,000,000 records. We assume that the result of each query do not exceeded the cache capacity. Figure 10 presents the result on Response Time for SC, Exact Match cache, and No-cache. In most cases, using cache is far more better than No-cache. In the ideal case (Test 1 with 100% HITS), the Response Time of Exact Match cache is slightly better than SC because the Query Processing Time of SC is more complex. In particular, it needs to broke and trim the query into a list of sub-queries that compose a list of attributes, predicates, and operators. For detail, when the HITS ratio is decreased to 70% (while Partial HITS is 20%), the SC is 1.41 times better Exact Match. In conclusion, if we ignore the ideal case (100% HITS), we have found that the SC is faster than Exact Match 1.60 times and 3.69 times than No-cache for tests with Partial HITS and MISSES.

3.3 FPGA acceleration opportunity

MASCARA is a framework towards FPGA acceleration. Thus, to point out the opportunities to offload some parts or modules of to accelerate on FPGA, we check the average percentage of computing time of 4 stages in over the query response time (illustrated in Figure 11)

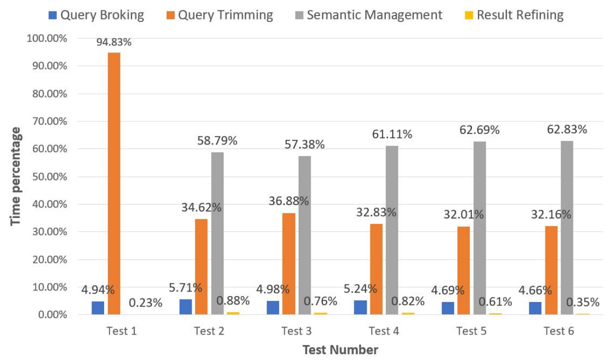


Figure 11: Average of percentage in time computing of MASCARA modules

Meanwhile the *Query Broking* and *Result Refining* do not consume too much time in processing a query, the *Query Trimming* (from 32% to 36%), and Semantic Management (from 57% to 62%) are noticeable. Skipping the first test which does not have much value for analysis, we do focus on the others. The results showed that if we could do the acceleration for *Query Trimming* and *Semantic Management*, then the performance of MASCARA would be improved. However, accelerating the *Semantic Management* stage seems not the most promising. *Semantic Management* consists of data transfers from disk or distributed storage that do not fit with FPGA acceleration. In contrast, *Query Trimming* is quite interesting and reasonable enough to accelerate with FPGA. In particular, finding the relevance between queries, checking query satisfiability (*Query Matching* sub-stage) and semantic information extracting (*Query Trimming* sub-stage), are very expensive in terms of computing time due to the time complexity of query processing (see Section 2.2). Even we can not offload all of the *Query Trimming* into FPGA, it is still valuable by accelerating some services or parts of this module, i.e. *Predicate Matching* over semantic segments. Moreover, the *Query Broking* stage (from 4.66% to 5.71%), where queries are materialized into various elements, is also considered to accelerate. By opportunistically offloading compute-heavy, compression tasks (*Query Trimming* stage) to FPGA, the CPU is freed to perform other tasks, resulting in an improved overall performance for MASCARA. Thus, consequently, in this preliminary analysis, we do point out what modules or services that we will focus on to accelerate with FPGA as the next step of our research.

4 Conclusion and Research Challenges

This paper presents the MASCARA framework towards FPGA acceleration with direct application in security monitoring for IoT. The proposal context works not only in security monitoring but also in relevant use cases for many research directions, such as: database management, parallel and distributed system, etc. Moreover, with this modular approach, MASCARA would be reasonable to deploy as a micro-service container by using the orchestration platforms in the context of cloud computing services.

This paper has first introduced the characteristics and operations of the framework through HTTP log analysis. Then, it has presented the experiment results to prove the performance of MASCARA preliminary. After that, it has pointed out a few limitations in query processing workflow and discussed at the same time about the opportunities in MASCARA to accelerate with FPGA. Soon, in our next developing step, we will extend MASCARA in both complicated queries processing with join and SC replacement strategy with semantic locality. It will result in having a comprehensive experiment to analyze deeply the performance indicators and show more convincing evidences to delegate works of MASCARA into FPGA.

Consequently, in this section, we intend to work along the following dimensions for the future research.

4.1 FPGA Accelerator Deployment

FPGAs have high intrinsic parallelism and internal bandwidth to speed up the query workloads. However, besides the high kernel throughput is achieved by FPGA [14], [18], the oblivious challenge of offloading modules or services to FPGA is the significant data communication overhead between high-level modules and low-level accelerators. Thus, the problem of using JNI (Java Native Interface) to support communication between JVM at high level and FPGA kernels at low level, needs to be considered carefully [6] [15].

4.2 Query Containment

The query containment, which includes query optimization, determining the independence of queries from updates, and rewriting queries using views, attracts a lot of attention for many years [4], [29], [5]. The latency of the workflow pipeline of the MASCARA framework can be improved by minimizing data movement, making stages faster, or even merging stages. Thus, with our early vision, the expensive query processing stage of MASCARA could benefit from answering conjunctive queries that are NP-complete [13].

Acknowledgements

We would like to thank members of SHAMAN team at IRISA and ENSA members at Nokia Bell Labs. We also thank the VLIoT reviewers for insightful comments and suggestions that improved the quality of this paper.

References

[1] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1383–1394. [Online]. Available: https://doi.org/10.1145/2723372.2742797

[2] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri, "The impact of caching on search engines," in *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development*

in Information Retrieval, ser. SIGIR '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 183–190. [Online]. Available: https://doi.org/10.1145/1277741.1277775

[3] C. Bobineau, C. Collet, and T.-T. Vu, "A strategy to develop adaptive and interactive query brokers," in *Proceedings of the 2008 International Symposium on Database Engineering & Applications*, ser. IDEAS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 237–247. [Online]. Available: https://doi.org/10.1145/1451940.1451973

[4] A. K. Chandra and P. M. Merlin, "Optimal implementation of conjunctive queries in relational data bases," in *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, ser. STOC '77. New York, NY, USA: Association for Computing Machinery, 1977, p. 77–90. [Online]. Available: https://doi.org/10.1145/800105.803397

[5] C. Chekuri and A. Rajaraman, "Conjunctive query containment revisited," *Theor. Comput. Sci.*, vol. 239, no. 2, p. 211–229, May 2000. [Online]. Available: https://doi.org/10.1016/S0304-3975(99)00220-0

[6] Y.-T. Chen, J. Cong, Z. Fang, J. Lei, and P. Wei, "When apache spark meets fpgas: A case study for next-generation dna sequencing acceleration," in *Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'16. USA: USENIX Association, 2016, p. 64–70.

[7] B. Chidlovskii and U. Borghoff, "Semantic caching of web queries." *The VLDB Journal*, vol. 9, pp. 2–17, 04 2000.

[8] S. Chu, A. Cheung, and D. Suciu, "Axiomatic foundations and algorithms for deciding semantic equivalences of SQL queries," *CoRR*, vol. abs/1802.02229, 2018. [Online]. Available: http://arxiv.org/abs/1802.02229

[9] S. Dar, M. J. Franklin, B. T. Jónsson, D. Srivastava, and M. Tan, "Semantic data caching and replacement," in *Proceedings of the 22th International Conference on Very Large Data Bases*, ser. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, p. 330–341.

[10] L. d'Orazio, F. Jouanot, C. Labbé, and C. Roncancio, "Building adaptable cache services," *ACM International Conference Proceeding Series*, vol. 117, pp. 1–6, 01 2005.

[11] L. d'Orazio and J. Lallet, "Semantic caching framework: An fpga-based application for iot security monitoring," *OJIOT*, vol. 4, pp. 150–157, 2018.

[12] P. Godfrey and J. Gryz, "Answering queries by semantic caches," in *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, ser. DEXA '99. Berlin, Heidelberg: Springer-Verlag, 1999, p. 485–498.

[13] A. Y. Halevy, "Answering queries using views: A survey," *The VLDB Journal*, vol. 10, no. 4, p. 270–294, Dec. 2001. [Online]. Available: https://doi.org/10.1007/s007780100054

[14] R. J. Halstead, I. Absalyamov, W. A. Najjar, and V. J. Tsotras, "Fpga-based multithreading for in-memory hash joins," in *CIDR*, 2015.

[15] M. Huang, D. Wu, C. H. Yu, Z. Fang, M. Interlandi, T. Condie, and J. Cong, "Programming and runtime support to blaze fpga accelerator deployment at datacenter scale," in *Proceedings of the Seventh ACM Symposium on Cloud Computing*, ser. SoCC '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 456–469. [Online]. Available: https://doi.org/10.1145/2987550.2987569

[16] B. T. Jónsson, M. Arinbjarnar, B. Þórsson, M. J. Franklin, and D. Srivastava, "Performance and overhead of semantic cache management," *ACM Trans. Internet Technol.*, vol. 6, no. 3, p. 302–331, Aug. 2006. [Online]. Available: https://doi.org/10.1145/1151087.1151091

[17] T. Kaldewey, G. Lohman, R. Mueller, and P. Volk, "Gpu join processing revisited," in *Proceedings of the Eighth International Workshop on Data Management on New Hardware*, ser. DaMoN '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 55–62. [Online]. Available: https://doi.org/10.1145/2236584.2236592

[18] K. Kara, J. Giceva, and G. Alonso, "Fpga-based data partitioning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 433–445. [Online]. Available: https://doi.org/10.1145/3035918.3035946

[19] A. M. Keller and J. Basu, "A predicate-based caching scheme for client-server database architectures," in *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, 1994, pp. 229–238.

[20] P.-r. Larson and H. Z. Yang, "Computing queries from derived relations," in *Proceedings of the 11th International Conference on Very Large Data*

Bases - Volume 11, ser. VLDB '85. VLDB Endowment, 1985, p. 259–269.

[21] K. C. K. Lee, H. V. Leong, and A. Si, "Semantic query caching in a mobile environment," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 2, p. 28–36, Apr. 1999. [Online]. Available: https://doi.org/10.1145/584027.584029

[22] M. Maghzaoui, L. d'Orazio, and J. Lallet, "Toward fpga-based semantic caching for accelerating data analysis with spark and HDFS," in *Information Search, Integration, and Personalization - 12th International Workshop, ISIP 2018, Fukuoka, Japan, May 14-15, 2018, Revised Selected Papers*, ser. Communications in Computer and Information Science, D. Kotzinos, D. Laurent, N. Spyratos, Y. Tanaka, and R. Taniguchi, Eds., vol. 1040. Springer, 2018, pp. 104–115. [Online]. Available: https://doi.org/10.1007/978-3-030-30284-9_7

[23] J. Nandimath, E. Banerjee, A. Patil, P. Kakade, S. Vaidya, and D. Chaturvedi, "Big data analysis using apache hadoop," in *2013 IEEE 14th International Conference on Information Reuse Integration (IRI)*, 2013, pp. 700–703.

[24] M. Negri, G. Pelagatti, and L. Sbattella, "Formal semantics of sql queries," *ACM Trans. Database Syst.*, vol. 16, no. 3, p. 513–534, Sep. 1991. [Online]. Available: https://doi.org/10.1145/111197.111212

[25] OpenCAPI, "A new standard for high performance memory, acceleration and networks," https://opencapi.github.io/oc-accel-doc/, Last accessed on 2020-20-06.

[26] M. Owaida, D. Sidler, K. Kara, and G. Alonso, "Centaur: A framework for hybrid cpu-fpga databases," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 211–218.

[27] Qun Ren, M. H. Dunham, and V. Kumar, "Semantic caching and query processing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 1, pp. 192–210, 2003.

[28] Q. Ren and M. H. Dunham, "Using semantic caching to manage location dependent data in mobile computing," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 210–221. [Online]. Available: https://doi.org/10.1145/345910.345948

[29] Y. Sagiv and M. Yannakakis, "Equivalences among relational expressions with the union and difference operators," *J. ACM*, vol. 27, no. 4, p. 633–655, Oct. 1980. [Online]. Available: https://doi.org/10.1145/322217.322221

[30] D. Sidler, Z. István, M. Owaida, and G. Alonso, "Accelerating pattern matching queries in hybrid cpu-fpga architectures," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 403–415. [Online]. Available: https://doi.org/10.1145/3035918.3035954

[31] D. Sidler, Z. Istvan, M. Owaida, K. Kara, and G. Alonso, "Doppiodb: A hardware accelerated database," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 1659–1662. [Online]. Available: https://doi.org/10.1145/3035918.3058746

[32] Spark SQL Guide, "Spark sql, dataframes and datasets guide," https://spark.apache.org/docs/latest/sql-programming-guide.html, Last accessed on 2020-20-06.

[33] S. Werner, D. Heinrich, S. Groppe, C. Blochwitz, and T. Pionteck, "Runtime adaptive hybrid query engine based on fpgas," *Open Journal of Databases (OJDB)*, vol. 3, no. 1, pp. 21–41, 2016. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-201705194645

[34] Q. Zhou, J. Arulraj, S. Navathe, W. Harris, and D. Xu, "Automated verification of query equivalence using satisfiability modulo theories," *Proc. VLDB Endow.*, vol. 12, no. 11, p. 1276–1288, Jul. 2019. [Online]. Available: https://doi.org/10.14778/3342263.3342267

AUTHOR BIOGRAPHIES

Laurent d'Orazio has been a Professor at Unv Rennes, CNRS, IRISA since 2016. He received his PhD degree in computer science from Grenoble National Polytechnic Institute in 2007. He was a Associate Professor at Blaise Pascal University and LIMOS NCRS, Clermont-Ferrand from 2008 to 2016. His research interests include (big) data algorithms and architectures, distributed and parallel databases. He has published papers in Information Systems, Sigmod Record, Concurrency and Computation Pratice and Experience. He served in Program Committees in BPM, workshops affilicated to VLDB, EDBT, etc. and Reviewing Commitees in Transactions on Parallel and Distributed Systems, Concurrency and Computation Pratice and Experience. He is or has been involved (sometimes as a coordinator) in research projects such as the NSF MOCCAD project (since 2013), the ANR SYSEO project (797 000 euroes funding, 2010-2015) and the STIC ASIA GOD project (30 000 euros funding, 2013-2015).

Emmanuel Casseau is a Professor at the University of Rennes, Lannion, France. He is a member of the IRISA Lab. / INRIA (French Institute for Research in Computer Science and Automation), France. He received the Ph.D Degree in Electronic Engineering from UBO University, Brest, France, in 1994 and the MS Degree in Electronic Engineering in 1990. His research interests are in the broader area of computer architecture, where he investigates the design of high-performance and cost-effective embedded systems and reconfigurable-based architecture design. Within this context, he has performed research in high-level synthesis, mapping/scheduling techniques, custom and application-specific hardware architectures targeting multimedia and signal processing applications, reconfigurable architectures and FPGA-based accelerators.

Julien Lallet joined the company Alcatel-Lucent in 2011 and is currently a research engineer at Nokia Bell Labs since 2016. He received his PhD degree in electrical engineering from the University of Rennes in 2008. He was a Post-doctoral fellow at the University of Bielefeld in Germany from 2009 to 2010. His research interests include efficient processing in the context of cloud computing and hardware acceleration on FPGA. He has published papers in computing architectures and FPGA systems.

Van Long Nguyen Huu is a lecturer of CanTho university, Vietnam since 2010. He received the MSc degree in Embedded system from USTH, Vietnam and from University of Limoges, France in 2015. He has started his PhD in Computer Science at University of Rennes from 2019. His research is financed by CIFRE scholarship and supported by IRISA, CNRS and Nokia Bell Labs. His research interests are semantic caching, query processing and hardware acceleration in Big Data.