Asperathos: Running QoS-Aware Sensitive Batch Applications with Intel SGX

Lília Sampaio, Clenimar Souza, Gabriel Vinha, Andrey Brito

[1]Laboratório de Sistemas Distribuídos
Universidade Federal de Campina Grande (UFCG)
Campina Grande – PB – Brazil

{liliars, clenimar, gabrielvinha, andrey}@lsd.ufcg.edu.br

Abstract. *The massive amount of information being generated nowadays results in the need for efficient data processing frameworks. For sensitive information, concerns also emerge regarding data integrity and confidentiality. To address such concerns, we present Asperathos, a configurable framework to automate the execution of batch applications in cloud environments while complying with QoS goals and processing potentially sensitive data. Our demonstration leverages tools such as Kubernetes and Intel SGX in a smart grid scenario, computing the power consumption from a dataset containing detailed measurements of users. We illustrate Asperathos features through the integration with both a command line and a web-based interface.*

1. Introduction

The processing of large amounts of sensitive data, generated by millions of people, such as their locations, personal communications, and even power consumption, raises growing concern not only on the efficient processing all this data, but also on the confidentiality and integrity of these data. Many companies use the insights given by these computations to improve their services and generate a variety of user recommendations. As an example, the processing of data from distribution sensors and power meters from end users can enable optimization on the distribution system or early detection of power quality issues[1].

Although such applications are very useful, the data confidentiality required on its processing and storage infrastructure should be carefully considered. Another common demand on these types of applications is the need for agile data processing that meets predefined Quality of Service (QoS) agreements. In this context, QoS is defined as a measure to guarantee a certain level of performance to a data flow. Moreover, QoS is especially important for real-time streaming applications and in systems that need to optimize resource usage and, thus, cannot be safe by overprovisioning.

Nevertheless, controlling data processing applications may be far from trivial as each application may require different types of resources or have different goals. As an example, controlling one application may require monitoring the usage of infrastructure resources such as CPU and memory. In contrast, the control of other applications may

[1]SecureCloud documents. https://www.securecloudproject.eu/wp-content/uploads/D5.4.pdf (Last accessed: 14 mar. 2019)

require the monitoring of metrics of the application itself, as the time it takes to process a request.

In this paper we introduce Asperathos, an open-source tool for running QoS-aware applications. Asperathos provides a configurable framework to automate the execution of big data applications in cloud environments while complying with QoS goals. In Asperathos, QoS is supported through customized actuators that regulate applications in execution time. Besides that, in contrast to other orchestration tools, such as Kubernetes[2] and OpenStack Heat[3], Asperathos also consider specific application metrics. Finally, because its flexibility, Asperathos is easily integrated with other technologies and tools, enabling the execution of sensitive applications that make use of trusted execution environments to guarantee data integrity and confidentiality, such as Intel SGX[4].

Intel SGX is a relatively new hardware technology, available in processors since 2015 [McKeen et al. 2013, Barbosa et al. 2016] and provides guarantees regarding data integrity and confidentiality by processing them inside protected memory areas named enclaves. Some capabilities of Intel SGX make it suitable to support applications that deal with sensitive data in the cloud, for instance, being able to encrypt and integrity check data before storing it in main memory, and the possibility of attesting running software before granting it access to the data [Costan and Devadas 2016]. In our demonstration, we use SCONE, a Secure Container Environment that uses SGX to protect containerized processes while simplifying the porting of applications [Arnautov et al. 2016]. SCONE provides support both encrypted memory and the secure delivery of secrets only to applications that have proved to be running, and not tampered with, inside enclaves.

The rest of the paper is organized as follows. Section 2 describes our Asperathos framework, its features and existing plugins. Section 3 presents a relevant use case to motivate and demonstrate the usage of this tool when considering smart metering data analysis. Section 4 details the demonstration to be presented and pointers to documentation and code resources. Lastly, Section 6 includes some final considerations.

2. Asperathos

Time-critical applications rely on the premise that the processing of an input must end before a predefined time. Some concerns about these types of applications appear for example when combining applications with different timeliness constraints and/or different priorities. Therefore, allocating resources wisely enables more efficiency, be it on energy or costs. Specially with recurrent and resource demanding application or when the technique is applied by the cloud provider itself, the over usage of such resources may have a considerable financial impact for their users [Li et al. 2011].

In order to offer a solution for this issue, we developed Asperathos, a configurable framework to automate the execution of big data applications in cloud environments while supporting QoS goals. Asperathos is composed of a set of components that communicate through REST APIs and can be configured to actuate in a cloud environment and control their resources. To support different cloud infrastructures and application management tools, Asperathos uses a plugin approach, as will be detailed next.

[2]Kubernetes. https://www.kubernetes.io (Last accessed: 12 mar. 2019)

[3]OpenStack. https://www.openstack.org/ (Last accessed: 12 mar. 2019)

[4]Intel SGX. https://software.intel.com/sgx (Last accessed: 25 mar. 2019)

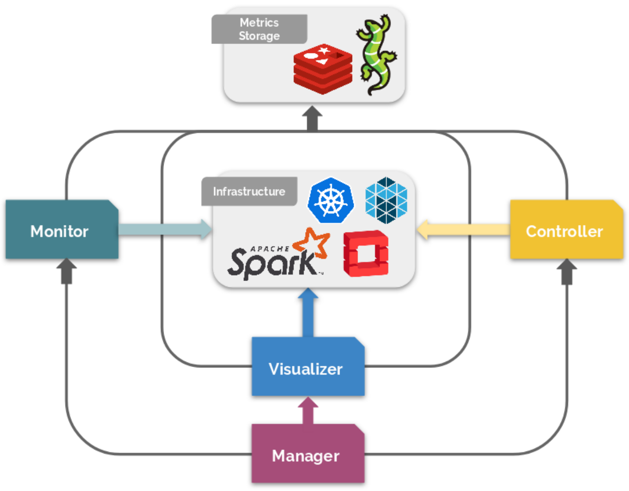


Figure 1. Asperathos component architecture.

2.1. Architecture

A diagram of the Asperathos architecture and how its components communicate is presented on Figure 1. The Manager is responsible to initiate new submissions and provide the necessary environment for an application to run. In our context, submissions are composed by the application itself and its respective execution parameters. In addition to that, the Manager is also responsible for starting the other components of the system.

The next component is the Monitor, which is responsible for collecting metrics from the environment by monitoring the units running the applications. Those metrics can then be used by the Controller, the component responsible to take actions on the environment based on the collected data and the QoS goals defined by the users. In this context, the metrics are pulled from some metric storage and are used, for instance, to decide whether to increase resource usage in order to finish a task on time or, alternatively, to decrease resource usage and save resources.

Lastly, the Visualizer component provides a visualization platform where the user can keep track of the application progress. The Visualizer service consumes the metrics collected by the Monitor and display graphics of these data in a dashboard. One visualization plugin will be detailed in Section 2.3 (Figure 2).

2.2. Existing plugins

Asperathos follows a plugin-based architecture. Plugins implement all available features, and allow the user to easily extend the framework functionality by creating new ones. Plugins currently available include **OpenStack**, which runs generic applications (defined in virtual machine images) in an OpenStack infrastructure; **Spark**, **Spark Mesos**, and **Spark Sahara**, which enable Apache Spark[5] data processing tasks to be run in different infrastructures (the basic Spark plugin considers a standalone cluster, Spark Mesos considers a cluster over Apache Mesos[6], and Spark Sahara considers a cluster provisioned through OpenStack Sahara[7]); the **Chronos** plugin runs periodic tasks with a strict time

[5]Spark Home Page: https://spark.apache.org/ (Last accessed: 14 mar. 2019)
[6]Mesos Home Page: http://mesos.apache.org (Last accessed: 14 mar. 2019)
[7]Sahara Documentation: https://docs.openstack.org/sahara/latest/ (Last accessed: 14 mar. 2019)

frame of execution; and the **KubeJobs** plugin runs containerized data processing applications in Kubernetes clusters. All the plugins aforementioned act on the infrastructure, increasing or decreasing the resources allocated to any given submission (be it containers, VMs, CPU and IOPS capacity, and so on) in order to adjust the application and meet QoS constraints. For this demonstration, we will focus on the KubeJobs plugin, as Kubernetes is currently the most popular tool for application orchestration, providing an application-centered, declarative API, and a rich ecosystem of tools and extensions.

2.3. Additional features

Besides the features already presented, Asperathos is also capable of using different clusters to execute tasks and register new ones to the platform. Clusters are selected as part of a new submission providing user control over this decision. Asperathos can manage credentials and keys to access such infrastructure.

Figure 2 depicts an example of execution that can be visualized using the Asperathos visualizer tool. The graphics show a `KubeJobs` plugin execution, the first one depicting the task progress relative to the time passed, the second showing the scaling of the replicas when necessary, and the last graph is the application error, which is the ratio of the task progress and the time spent. For this visualization, Grafana[8] was used, together with an Influx Database[9] as the metric source.

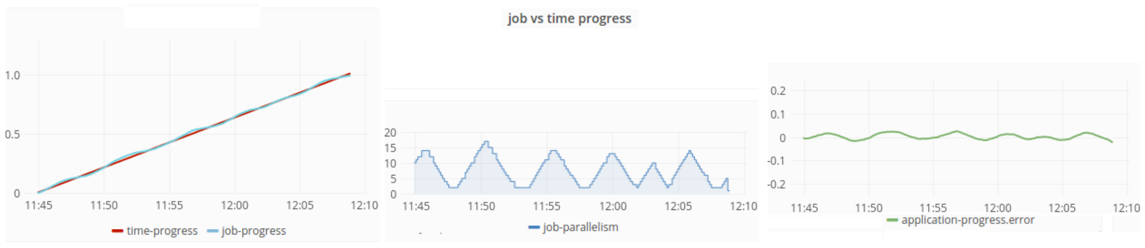


Figure 2. Task execution in the Grafana dashboard provided by Asperathos.

3. Use case: smart metering data analysis

Consider that the electricity supply network of an entire city is monitored with a rich network of sensors and smart meters, which collect measurements with high granularity. These smart meters and sensors continuously collect and send data to the cloud. Job routines process these data to detect anomalies, such as failures, frauds and quality issues, and also to simulate how the system responds to unusual scenarios (e.g., rapid demand escalations or failure of major components).

Such routines also need to consider data integrity and meet time constraints, as they are executed periodically. Asperathos acts to ensure those constraints are respected by allocating more resources to tasks that are behind the schedule or by freeing up resources if the tasks are ahead of their time, potentially reducing costs.

As Asperathos components expose REST APIs to submit and manage workloads and infrastructure, it is possible to build more components on top of the framework to bet-

[8]Grafana Home Page: https://grafana.com/ (Acesso em: 14 mar. 2019)
[9]Influx Data Home Page: https://www.influxdata.com/ (Acesso em: 14 mar. 2019)

ter suit the user needs, Subsections 3.2 and 3.3 present real world use cases to help illustrate such usages. For the examples here we are using the `KubeJobs` plugin, and assuming the Manager is running locally on the user's machine (`http://0.0.0.0:1500`), but potentially controlling a remote cluster (e.g., in a public cloud).

3.1. Implementation

The data analysis routine taken as an example consists of a Python program that downloads encrypted files from an object storage in the cloud. Each file contains smart meter data from a specific customer and is processed to calculate the average consumption and quality metrics, publishing the encrypted results back to the object storage. Encrypting the input and output files ensures that customer data is still protected even if the files leak (due to misconfiguration of the storage systems or due to invasions), as long as the encryption key is kept secret.

The Python routine relies on the SCONE framework to provide confidentiality and integrity guarantees. Confidentiality is ensured by running the entire application inside of Intel SGX enclaves, while integrity is provided by SCONE's remote attestation scheme. Attesting an enclave means that the application currently running was not modified from what the developer intended to run. Once attested, the application is able to securely get secrets (i.e. the encryption key) from a trusted secret management service.

Asperathos allows such routine to run in a distributed fashion, processing up to hundreds of customer records simultaneously, while supporting secure execution technologies (e.g. Intel SGX). Asperathos also controls resource allocation based on user-defined deadlines (QoS), and provides failure-recovery mechanisms, guaranteeing that each task is processed at least once.

3.2. Running from command line

The simplest way to submit workloads to Asperathos is by sending an HTTP request to its main entry point, the Asperathos Manager. The request body must contain a JSON document with all relevant parameters to the workload submission. Please refer to Asperathos Quick Start Guide[10] for more information.

```
$ curl -H "Content-Type: application/json" --data @submission.json \
  http://0.0.0.0:1500/submissions
```

Asperathos returns the submission ID (e.g. `kj-d59cf24`), which can be used to retrieve more information about the workload, such as its status and the address for the visualization platform (as shown in Figure 2):

```
$ curl http://0.0.0.0:1500/submissions/kj-d59cf24
{"kj-d59cf24": {"visualizer_url":"http//10.5.0.77:32044", "status": "
    ongoing", ...}}
```

The routine starts only after its code is attested. If any unintended or malicious change was made to the code, the routine will not start. After processing a customer

[10]Asperathos user guide. https://github.com/ufcg-lsd/asperathos/wiki/User-Guide (Last accessed: 14 mar. 2019)

record file, the routine publishes the results back to the object storage. In this particular example, OpenStack Swift is our object storage of choice, but other services (e.g. S3 or Google Cloud Storage) can be used as well. Results are also encrypted when posted to the object storage. After downloading the output files from the object storage to a trusted computer, the user can then decrypt the items and see the actual results:

```
$ ./decrypt --key $KEY --file OUTPUT_consumer_data_13 >
    decrypted_output
$ cat decrypted_output
298
```

3.3. Running from web-based user interface

It is possible to build tools around Asperathos to extend its functionalities or to provide a better user experience. In this example we showcase a simple web-based user interface (UI) intended to run locally on the user's trusted machine. The UI simplifies the submission process by letting the workload parameters be defined in a user-friendly way. It is also possible to define resource allocation limits and QoS constraints, such as a completion deadline (Figure 3).

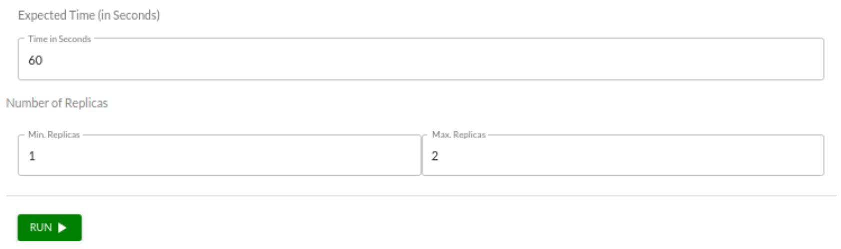
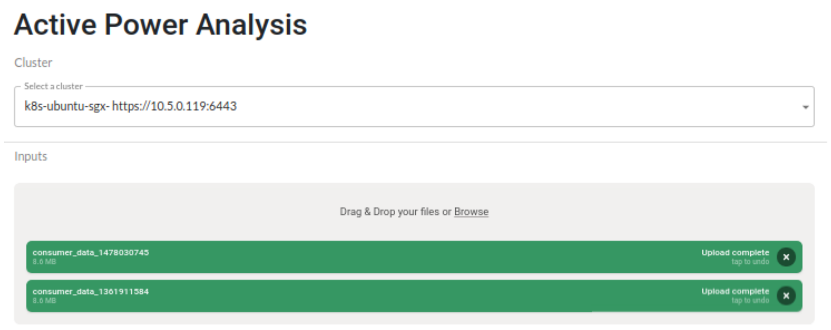

Expected Time (in Seconds)
Time in Seconds: 60
Number of Replicas
Min. Replicas: 1
Max. Replicas: 2

Figure 3. Submission form. The user can define QoS constraints for the submission (e.g. Expected Time), as well as other parameters for the execution (minimum and maximum amount of resources).

The user is able to choose the Kubernetes cluster where the workload must run, or to easily add new clusters, regardless of the provider on which it runs. This example requires that the clusters have Intel SGX support, as the workloads run inside of enclaves. The UI is also responsible for encrypting the input items (Figure 4), pushing them to an object storage of choice, triggering Asperathos and monitoring the submission progress and its visualization.

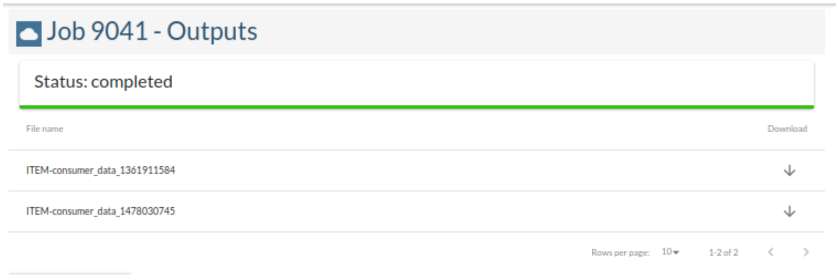

Active Power Analysis
Cluster
Select a cluster: k8s-ubuntu-sgx- https://10.5.0.119:6443
Inputs

consumer_data_1478030745 8.6 MB Upload complete tap to undo
consumer_data_1361911584 8.6 MB Upload complete tap to undo

Figure 4. Submission form. The user is able to select input files from their local storage. The UI will encrypt and push them to the object storage.

After the Python routine is attested, which means its exactly the same code (including external libraries) as the developer intended to run, the secret management service sends the encryption key to the enclaves, and the customer records downloaded from the object storage can be securely decrypted and processed. When outside of the user's trusted machine, all the sensitive data is either encrypted or inside Intel SGX enclaves, ensuring that it is protected even in compromised infrastructures.

After the submission completes, the UI downloads the encrypted results from the object storage and decrypts them locally in the user's trusted machine (Figure 5). If the attestation process fails, indicating there was some unintended change to the code, the submission will show an *Attestation Error* status.



Job 9041 - Outputs

Status: completed

File name	Download
ITEM-consumer_data_1361911584	↓
ITEM-consumer_data_1478030745	↓

Rows per page: 10▾ 1-2 of 2

Figure 5. After the submission completes, the UI downloads the encrypted results and decrypts them locally.

4. Demonstration

Asperathos is released as an open source project that can be downloaded from Github[11]. For a quick start on Asperathos, documentation is available on the wiki page[12]. In addition to that, there is a video[13] presenting a motivation for Asperathos and two videos[14][15] showcasing the installation process and some of its features by navigating through our portal.

The use cases described in Section 3 allow for two different scenarios to be demonstrated. First a quick setup of the Asperathos components is demonstrated, followed by a simple application submission. For this scenario we assume the user have a configured Kubernetes cluster ready for use, with the proper Intel SGX software installed. All the steps for this scenario are command line based, using Asperathos REST API.

The second demonstration will present a use case that integrates Asperathos to a web-based user interface. Thus, we will demonstrate functionalities such as cluster creation, application submission, task progress, download of results, and an example of a Grafana visualization. Besides that, we are going to show an example of a possible attack, where an operator manages to replace the application code. The attack case will result in an error and the modified application will not produce results, since the SGX attestation process fails. This demonstrates the security aspect of this paper.

[11]Asperathos repository. https://github.com/ufcg-lsd/asperathos (Last accessed: 14 mar. 2019)
[12]Guide. https://github.com/ufcg-lsd/asperathos/wiki/Quick-Start (Last accessed: 14 mar. 2019)
[13]What is Asperathos. `https://youtu.be/UURyL_g9LT8` (Last accessed: 14 mar. 2019)
[14]Asperathos installation. `https://youtu.be/bMI02YJ5uH8` (Last accessed: 14 mar. 2019)
[15]Asperathos portal. `https://youtu.be/s8_hRV4SpCU` (Last accessed: 14 mar. 2019)

5. Related Work

Other systems have been built to provide QoS for time critical applications. [Evans et al. 2015] provides a system that achieves QoS requirements for workflow systems, however it is limited to these kind of applications and cannot be orchestrated with other cloud providers and technologies. As for security, [Schuster et al. 2015] considers using Hadoop in an SGX environment. In order to enable SGX processing of Hadoop applications, the application is split between a not trusted part and a trusted portion. As our model is simpler, the Asperathos framework can be configured to run in a variety of programming models and analytic infrastructures.

6. Conclusions

This work presents Asperathos, a framework for running QoS-aware sensitive batch applications on cloud environments, enabling customized plugins and actuators that adjust applications in runtime. We also demonstrate that such concepts can be applied in the execution of sensitive applications that need guarantees of data integrity and confidentiality, which in our case is obtained by using the Intel SGX technology together with Asperathos. The framework is demonstrated with a use case that considers smart grid analysis being securely processed by Asperathos and shows protection against attacks that tamper with the application code to leak sensitive information.

References

Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M. L., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., and Fetzer, C. (2016). SCONE: Secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 689–703. USENIX Association.

Barbosa, M., Portela, B., Scerri, G., and Warinschi, B. (2016). Foundations of hardware-based attested computation and application to sgx. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 245–260. IEEE.

Costan, V. and Devadas, S. (2016). Intel sgx explained. Cryptology ePrint Archive, Report 2016/086. http://eprint.iacr.org/2016/086.

Evans, K., Jones, A., Preece, A., Quevedo, F., Rogers, D., Spasić, I., Taylor, I., Stankovski, V., Taherizadeh, S., Trnkoczy, J., Suciu, G., Suciu, V., Martin, P., Wang, J., and Zhao, Z. (2015). Dynamically reconfigurable workflows for time-critical applications. In *Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science*, WORKS '15.

Li, H., Liu, J., and Tang, G. (2011). A pricing algorithm for cloud computing resources. In *2011 International Conference on Network Computing and Information Security*, pages 69–73.

McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C. V., Shafi, H., Shanbhogue, V., and Savagaonkar, U. R. (2013). Innovative instructions and software model for isolated execution. In *HASP@ ISCA*, page 10.

Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G., and Russinovich, M. (2015). Vc3: Trustworthy data analytics in the cloud using sgx. In *2015 IEEE Symposium on Security and Privacy*, pages 38–54.