

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

**CORSO DI LAUREA MAGISTRALE IN INFORMATICA**

A LOG BASED TOOL FOR MONITORING

SERVICE LEVEL AGREEMENTS VIOLATIONS IN CLOUD ENVIRONMENTS

|  |  |
| --- | --- |
| **Relatore:** | **Candidato:** |
| **Prof.ssa Filomena Ferrucci** | **s*abato Napolitano*** |
| **Correlatori:** | **mat. 0522500222** |
| **Prof. Tahar Kechadi** |  |
| **Dott. Pasquale Salza**  **Dott.ssa Lucia De Marco** |  |

Anno Accademico 2014/2015

Table of contents

[Abstract 0](#_Toc429905810)

[Introduction 1](#_Toc429905811)

[Background 3](#_Toc429905812)

[Cloud Computing and SLA 3](#_Toc429905813)

[Cloud Forensic and Forensic Readiness 4](#_Toc429905814)

[State of art 6](#_Toc429905815)

[Goals 9](#_Toc429905816)

[Cloud forensic tool using SLA and Logs information 9](#_Toc429905817)

[System Design 10](#_Toc429905818)

[Preliminary study: Cloud Threads and Attacks 10](#_Toc429905819)

[Scenario 11](#_Toc429905820)

[Narrative 11](#_Toc429905821)

[Context 11](#_Toc429905822)

[Amazon S3 SLA 12](#_Toc429905823)

[Amazon S3 SLA 13](#_Toc429905824)

[Effective Date: June 1, 2013 13](#_Toc429905825)

[Service Commitment 13](#_Toc429905826)

[Definitions 13](#_Toc429905827)

[Response time 14](#_Toc429905828)

[Number of Connections 14](#_Toc429905829)

[Service Credits 14](#_Toc429905830)

[Credit Request and Payment Procedures 15](#_Toc429905831)

[Amazon S3 SLA Exclusions 16](#_Toc429905832)

[Uptime percentage anomaly 17](#_Toc429905833)

[Cloud Forensic Readiness Tool (CFRT) for SLA violations recognition 17](#_Toc429905834)

[Working Environment 18](#_Toc429905835)

[*CFRT* 19](#_Toc429905836)

[*System Design* 19](#_Toc429905837)

[Architecture 19](#_Toc429905838)

[System implementation 22](#_Toc429905839)

[Importing of logs 22](#_Toc429905840)

[Mongo DB tables description 22](#_Toc429905841)

[Middleware 24](#_Toc429905842)

[AWS Dictionary and Formulas 25](#_Toc429905843)

[DBManager 26](#_Toc429905844)

[System Modules 27](#_Toc429905845)

[System Running 29](#_Toc429905846)

[Formula extraction and real value computation 29](#_Toc429905847)

[Graphic User Interface 38](#_Toc429905848)

[System Output 40](#_Toc429905849)

[System Testing 41](#_Toc429905850)

[Violations: 41](#_Toc429905851)

[Number of connections and response time violations 43](#_Toc429905852)

[Uptime and response time violations 44](#_Toc429905853)

[Number of connections violation 46](#_Toc429905854)

[Response time violation 47](#_Toc429905855)

[Empirical study 49](#_Toc429905856)

[Generation of S3 logs Data 51](#_Toc429905857)

[Attack simulation and dataset building 51](#_Toc429905858)

[Test environment 52](#_Toc429905859)

[Tools 53](#_Toc429905860)

[Methodology 53](#_Toc429905861)

[Technical specifications 55](#_Toc429905862)

[SLA used 56](#_Toc429905863)

[SimpleDoS 57](#_Toc429905864)

[S3 Log details 58](#_Toc429905865)

[Pearl translation script details 63](#_Toc429905866)

[MongoDB 64](#_Toc429905867)

[Attack Prediction idea 65](#_Toc429905868)

[Attacks/SLO Mapping 65](#_Toc429905869)

[SLA Violations and DoS Attacks evidence 68](#_Toc429905870)

[Results 69](#_Toc429905871)

[Linear Discriminant 70](#_Toc429905872)

[Naïve Bayes 70](#_Toc429905873)

[Decision Tree 70](#_Toc429905874)

[Attributes subset 71](#_Toc429905875)

[Conclusion 72](#_Toc429905876)

[References 75](#_Toc429905877)

[Figure 1 - LMS Layout with probes, proposed in [16] 7](#_Toc429172472)

[Figure 2 - Risk matrix used in [21] to classify cloud threats 10](#_Toc429172473)

[Figure 3: Amazon System Overview 12](#_Toc429172474)

[Figure 4 – Attack simulation architecture 54](#_Toc429172475)

[Figure 5 - SQL table of SLO used during the simulation. 56](#_Toc429172476)

[Figure 6 - First proposed architecture. 19](#_Toc429172477)

[Figure 7 - Class diagram that shows classes that access to DBManager utility class. 27](#_Toc429172478)

[Figure 8 – CFRT system modules. 28](#_Toc429172479)

[Figure 9- Flow chart for formula evaluation 30](#_Toc429172480)

[Figure 10 - Flow Chart for Average function evaluation. 32](#_Toc429172481)

[Figure 11 - Uptime evaluation and matching. 33](#_Toc429172482)

[Figure 12 - Average response time evaluation and matching. 34](#_Toc429172483)

[Figure 13 - Number of simultaneous client evaluation and matching. 36](#_Toc429172484)

[Figure 14 - Flow charts for attack checker. 37](#_Toc429172485)

[Figure 15 - Sequence diagram for DDoS attack check. 38](#_Toc429172486)

[Figure 16 - "Set parameters" window in FRS system 39](#_Toc429172487)

[Figure 17- File explorer window 40](#_Toc429172488)

[Figure 18 - SLO violation alert. 40](#_Toc429172489)

[Figure 19 - DDoS attack alert. 41](#_Toc429172490)

[Figure 20 - Results for linear discriminant analysis (LDA) classifier. 70](#_Toc429172491)

[Figure 21 - Results for naive Bayes (NB) classifier. 70](#_Toc429172492)

[Figure 22 - Results for decision tree classifier. 71](#_Toc429172493)

[Table 1- Hardware specification for machine used during the simulation process 55](#_Toc429172494)

[Table 2- Apache/S3 logs mapping 59](#_Toc429172495)

[Table 3- Generation methods for missing fields 60](#_Toc429172496)

[Table 4 - S3 access log fields description 62](#_Toc429172497)

[Table 5 - Mapping between SLO and cloud attack. 67](#_Toc429172498)

[Table 6 - Description of fields of Attack collection 67](#_Toc429172499)

[Table 7 - Description of fields of collection "data" 23](#_Toc429172500)

[Table 8 - Description of fields of collection "violations" 23](#_Toc429172501)

[Table 9 - Description of fields of collection "log" 24](#_Toc429172502)

[Table 10: Run parameters for FRS system 39](#_Toc429172503)

# Abstract

The traditional digital forensic investigation process has always had a post-event driven focus. In cloud environment, this process is harder and too long because of the cloud environment complexity. This work propose a tool to introduce the digital forensic readiness in cloud environment. The Cloud Forensic Readiness Tool (CFRT) can be used to quicken and update the traditional digital forensic investigation process to better suit cloud computing environments. Nowadays all big cloud providers offer a secure log utility for cloud monitoring. The author proposes an adaptable software based on cloud logging utilities to gather information about QoS and SLA violation in preparation of an investigation. This approach will quicken the acquisition of evidential data when an investigation is required. During this work, we tried to propose a new approach to the attack recognition process; CFRT tries to recognize an attack detecting SLO violations during logs analysis. In this way, it warns provider and customer, allowing the investigator to start the analysis and examination almost immediately.

# Introduction

This work started during my Erasmus experience at the University College of Dublin (UCD). The work try to find out a new way to realize the concept of cloud forensic readiness in cloud computing platform. Starting from the idea that Forensic Readiness capability can be implemented in every computing platform, we focused on the new cloud platform.

The primary objectives of this research can be summarized as follows: Shortening the DFI process by quickening the acquisition of data with a software that could support forensic investigations. The secondary objectives include the identification of an attack as soon as possible.

Digital Forensic Science is defined by Palmer[26], as “the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations.”

Normally a DFI follows a search and seizure approach. This entails seizing suspected devices for acquisition and analysis of data. Acquisitions is normally done by making a bit-by-bit copy of the storage medium of a device. However in a cloud environment this is not a simple task to perform. The cloud holds some challenges for digital forensic investigators that is discussed by Barbara [24]. One such challenge discussed by Birk [25] includes gaining access to the physical hardware running the cloud instance as the physical location of devices is often unknown, making search and isolation of devices difficult if not impossible. The time taken to complete a Digital Forensic Investigation (DFI) has the potential to be one of the biggest challenges of the investigation due to time-line constraints and deadlines, and could determine the

success of the investigation. This poses the challenge to the investigators to attempt to complete the investigation or at least part of it as fast as possible.

The cloud service providers also has full control over the sources of evidence and also the company’s assets, making investigations by corporate security teams difficult if not impossible.

Barbara concludes that the challenge for digital forensic examiners, with regard to cloud computing environments, is to determine who, what, when, where, how, and why of cloud-based criminal activity [24]. This can be crucial information in a DFI, and these questions can be answered analyzing information stored in logs.

The traditional model for a DFI was not designed with the cloud in mind and therefore it is worth investigating whether this traditional model can be optimized or improved to better suit cloud computing environments.

Shortening the evidence acquisition phase already improves on the problem of time that is a major concern as Tan has shown [12]. Hence, the research questions that this work addresses is as follows: How can we use digital forensic readiness to help to shorten the digital forensic investigation phase in cloud computing environments? How can we use information disclosed by real cloud providers to help the DFI without access to cloud low-level infrastructure information?

The remainder of this document is structured as follows. The next two sections discuss the background and introduce topics of cloud computing and forensic analysis, it will be described the state of art leading up to this research being undertaken. In sections "System Design" and "CFRT", one discusses about design and implementation of a software tool for forensic readiness. The section "Empirical Study" describes a practical usage of CFRT. In the section "SLA Violations and DoS Attacks evidence" all information gathered from the empirical study are evaluated in order to prove that is possivle recognize and predict a DoS attack.

In section "Conclusions", there are described results and there are explained future developments of this work.

# Background

### Cloud Computing and SLA

Cloud computing paradigm extends concepts of big data centers and parallel computing. In the last decade, the cloud computing has seen tremendous growth, particularly for commercial web applications. The on-demand, pay-as-you-go model creates a flexible and cost-effective means to access compute resources. Thanks to the custom interfaces and automated processes, clouds platforms are easy to use and manage. For these reason, day by day, more companies and privates choose to migrate their systems on cloud platforms.

These services include infrastructure-as-a-service (IaaS), platform-as-a-service (PaaS), and software-as-a-service (SaaS) [10]. Each service is typically accompanied by a service level agreement (SLA) which defines the minimal guarantees that a provider offers to its customers. The lack of standardization in cloud-based services implies a corresponding lack of clarity in the service level agreements offered by different providers [9].

Because of the lack of standards, information from SLA are hard to acquire automatically. The standardization of SLA is nowadays an open challenge. The European Commission presented guidelines to define a standard for cloud SLAs documents.

In [19] is described the concept of Service Level objectives (SLO) as a quantitative measurements of a particular characteristic of service. Measurements should also be comparable since reduced comparability impedes adoption.

To be comparable, service level objectives need not be determined by identical means but sufficient information about the SLO needs to be provided by cloud service providers. Standardized terminology, metrics and templates can be helpful in documenting how a particular SLO is determined.

In this work, we use the monitoring of SLO as a resource to realize the Cloud Forensic readiness concept.

### Cloud Forensic and Forensic Readiness

As all new technologies, the cloud systems have vulnerabilities that can be exploited in order to make an attack and to breakdown a certain customer. Attacks on cloud aim to steal information, compromising the security of data, disrupting the service and producing economic problems.

A forensic investigation of digital evidence is commonly employed as a post-event response to a serious information security incident. In fact, there are many circumstances where an organization may benefit from an ability to gather and preserve digital evidence before an incident occurs. Forensic readiness is defined as the ability of an organization to maximize its potential to use digital evidence whilst minimizing the costs of an investigation [11].

In 2001, Tan [12] introduced the concept of forensic readiness to cover two objectives:

* Maximising an environment’s ability to collect credible digital evidence;
* Minimising the cost of forensics during an incident response.

The problem was approached from the need to reduce the time and costs of a forensic examination. Tan quotes the example of the HoneyNet project forensic challenge [20], where half an hour of attacker time required an average investigation time of 48 hours. Tan also discussed technical aspects such as time-stamping, system hardening and compromised kernels, and noted five factors that affect evidence preservation and investigation time:

* How logging is done;
* What is logged;
* Intrusion detection systems;
* Forensic acquisition;
* Evidence handling.

Tan’s definition is deeply bounded to classical systems. In [12], logging information are hardware dependent, and this fit better in non-virtualized architectures.   
Although, cloud systems run on normal hardware, they make use of a multi-tenant architecture, virtualizing the hardware. For this reason, the standard hardware-bounded logging is no more reliable. Furthermore, cloud forensics has different challenges enlisted in [13]. Hence, in this work we will try to migrate the concept of forensic readiness from classical systems to the newest cloud platforms.

# State of art

A branch of Digital Forensic is focusing on the Forensic Readiness capability; as said before, its purpose is rendering a computing platform ready for Digital Forensics by performing some activities that will be necessary for subsequent investigations with the side effect of saving time, money and energy. A Forensic Readiness capability can be implemented in every computing platform. Focusing on the new cloud platform, several proposals have been successfully published in the last years; also, the Cloud Forensic community is interested in such aspect.

According to [12], forensic readiness is based on logging activity. This activity result to be simple in standard systems, but when we talk about cloud environment, log production and management becomes a big challenge, especially working on SaaS and PaaS layers, as descripted in [15]. Some of these challenges about logging and log management are enlisted in [13]. In [13], analysis of syslogs and network logs is used find clues of attack after a DoS attack.

Some studies have given their own implementation of a monitoring system in cloud environment [14] and [16]. In [16], Van Staden and H.S.Venter try to describe an application of cloud forensic readiness in a LMS. Probes on every machine and router record information about performance of the system, storing all data in a big SQL database.



Figure 1 - LMS Layout with probes, proposed in [16]

Van Staden et al. do not give any automated processes to detect possible incidents in order to only collect data in case of possible incidents and then reduce the amount of data collected.

In order to use any cloud service, every customer need to agree the service level agreement (SLA) document. A service level agreement is an agreement between two or more parties, where one is the customer and the others are service providers. A common feature of an SLA is a contracted performance of services and it is called SLO. In [17] is introduced the idea of SLA monitoring for the cloud application layer. The concept makes use of SLO values as comparison for real system performance, and so the terminology “violation” referring to a mismatch between SLA’s values and real application performance. [17] proposes an application monitoring architecture named CASViD, which stands for Cloud Application SLA Violation Detection architecture. CASViD architecture monitors and detects SLA violations at the application layer, and includes tools for resource allocation, scheduling, and deployment.

We started from CASViD idea, pursuing the purpose of the cloud forensic readiness. As seen, at the state of art there are no proposal to achieve attack prediction using SLA monitoring. Indeed, this two concepts coexist, each one for different purposes. In this work was developed CFRT to use SLA monitoring together with log analysis from cloud forensics , to implement the concept of cloud forensic readiness.

# Goals

## Cloud forensic tool using SLA and Logs information

Using SLA quality-of-service values and logs provided by cloud provider, or gathered using the approach of [15]. It became possible to realize a cloud forensic readiness tool for cloud systems that improve the approach of [16]. The forensic readiness is achieved filtering logs of service and giving extra information about time and quantity of SLA violations registered, only when these violations occur.

Since developing of a new method of log gathering was not the main purpose, we decided to use a well-developed and well-documented API provided by Amazon. In particular was used S3 as service and the bucket logging service. Amazon was chosen, mainly because it provide a way to generate and retrieve access logs easily, through a set of RESTful API

In the following pages will be explained the path and decisions taken that leaded to realization of a cloud forensic readiness tool (CFRT) starting from the idea of using SLAs and logs provided by the cloud services provider. In section about empirical study, it will be described a real case of use of CFRT, explaining all tools and environment used. In the last section, classification results will be show.

# System Design

## Preliminary study: Cloud Threads and Attacks

In order to recognize attacks on cloud, it’s made a preliminary study about cloud system security threads. There is a big literature about this topic, and, in the past years, the Cloud Security Alliance made a classification of top threats in cloud computing systems [21]. The report gives not only a description and an example of the threat, but also a classification based on the risk matrix.

Among all threats, it’s mostly paid attention on the “red-zone” threats. Indeed, in the top-right area of the matrix, both the perceived risk and the real risk is high.

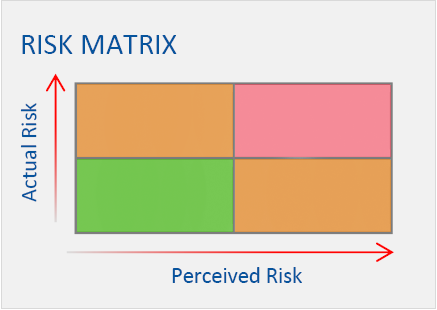


Figure 2 - Risk matrix used in [21] to classify cloud threats

One of the most quoted and common attack is the Denial of Service (DoS), and the improved variant: the Distributed Denial of Service (DDoS).

Both these attacks aim to exploit threats belonging to the “red-zone”, causing loss of information, money and, in some cases, the complete and permanent break down of the service. Since these attacks are not new to the forensic analysis, there are different strategies of DoS detection and prevention, most of them are based on network analysis as the one proposed in [22].

Although, DoS and DDoS attacks strategies keep changing with time. it is more difficult with time recognize and avoid them. Hence, in this work, it is proposed a new way to recognize those attacks. Indeed, the proposed CFRT tries to recognize DoS and DDoS attacks using violation recognitions.

## Scenario

In this paragraph is described a typical use case scenario of the system.

### Narrative

As administrator of a wiki website, Bob wants that his contents should be always available and the website has to be scalable, flexible and accessible from every country. For these reasons he chooses to host his system on a cloud platform. Bob is worried about data security and he wants to be advised if there is a possible attack in progress, so that he can start an analysis of the system and the cloud provider can start a forensic analysis to verify eventual attacks.

### Context

Bob’s system runs on Amazon cloud platform (**EC2**). This service provides a virtual machine called “instance” that runs on a multi-tenant hardware. (**5**) Bob uses his instances to run his web application. Data is stored through Amazon relational DB (**RDS**) (**7**). Some static contents, like HTML & CSS pages, images, video etc. are stored on Amazon **S3** (**3**) service.

The figure below shows the logical architecture of Bob’s system.

Bob has become an Amazon customer and has subscribed the AWS (Amazon Web Services) Service Level Agreement (**SLA**). This legal document, which is defined for each service included in the AWS pool, specifies some information about services provided. The most present is the uptime percentage for each service. For some kind of customers like big enterprises, provider and customer can accord to create a different SLA (private SLA). This SLA has more detailed information based on the negotiation between the two parties. Bob, being a simple customer, subscribes AWS public SLAs[5].

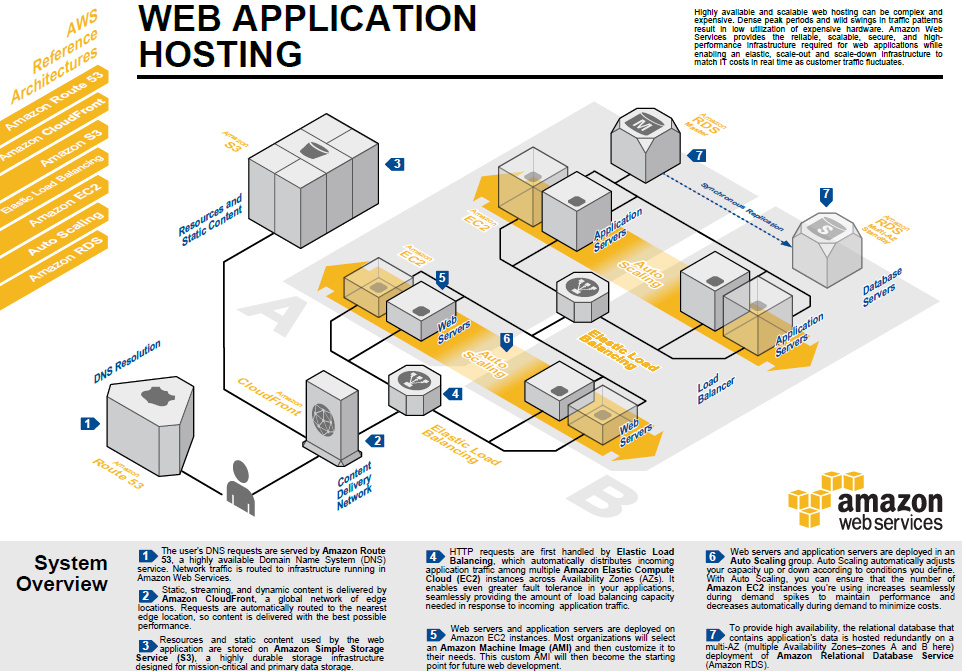


Figure 3: Amazon System Overview

### Amazon S3 SLA

This is the Amazon S3 SLA approved by Bob:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Amazon S3 SLAEffective Date: June 1, 2013 This Amazon S3 Service Level Agreement (“SLA”) is a policy governing the use of Amazon Simple Storage Service (“Amazon S3”) under the terms of the Amazon Web Services Customer Agreement (the “AWS Agreement”) between Amazon Web Services, Inc. (“AWS”, “us” or “we”) and users of AWS’ services (“you”). This SLA applies separately to each account using Amazon S3. Unless otherwise provided herein, this SLA is subject to the terms of the AWS Agreement and capitalized terms will have the meaning specified in the AWS Agreement. We reserve the right to change the terms of this SLA in accordance with the AWS Agreement. Service Commitment **AWS will use commercially reasonable efforts to make Amazon S3 available with a Monthly Uptime Percentage (defined below) of at least 99.9% during any monthly billing cycle (the “Service Commitment”).** In the event Amazon S3 does not meet the Service Commitment, you will be eligible to receive a Service Credit as described below. Definitions  * **“Error Rate” means: (i) the total number of internal server errors returned by Amazon S3 as error status “InternalError” or “ServiceUnavailable” divided by (ii) the total number of requests during that five minute period. We will calculate the Error Rate for each Amazon S3 account as a percentage for each five minute period in the monthly billing cycle. The calculation of the number of internal server errors will not include errors that arise directly or indirectly as a result of any of the Amazon S3 SLA Exclusions (as defined below).** * “**Monthly Uptime Percentage” is calculated by subtracting from 100% the average of the Error Rates from each five minute period in the monthly billing cycle.** * **“Average Response Time” is the average of all requests response times from each 5 minute in the billing month cycle.** * **“Number of Simultaneous Connections” is the number of separate cloud service customer users that can be using S3 service each 1 minute.** * A “Service Credit” is a dollar credit, calculated as set forth below, that we may credit back to an eligible Amazon S3 account.  Response time **The average response time is less than 500 ms every 5 minutes.** Number of Connections **The number of simultaneous clients is less than 500 each 1 minute.** Service Credits Service Credits are calculated as a percentage of the total charges paid by you for Amazon S3 for the billing cycle in which the error occurred in accordance with the schedule below.   |  |  | | --- | --- | | Monthly Uptime Percentage | Service Credit Percentage | | Equal to or greater than 99% but less than 99.9% | 10% | | Less than 99% | 25% |   We will apply any Service Credits only against future Amazon S3 payments otherwise due from you. At our discretion, we may issue the Service Credit to the credit card you used to pay for the billing cycle in which the error occurred. Service Credits will not entitle you to any refund or other payment from AWS. A Service Credit will be applicable and issued only if the credit amount for the applicable monthly billing cycle is greater than one dollar ($1 USD). Service Credits may not be transferred or applied to any other account. Unless otherwise provided in the AWS Agreement, your sole and exclusive remedy for any unavailability, non-performance, or other failure by us to provide Amazon S3 is the receipt of a Service Credit (if eligible) in accordance with the terms of this SLA. Credit Request and Payment Procedures To receive a Service Credit, you must submit a claim by [opening a case in the AWS Support Center](https://aws.amazon.com/support/createCase?type=account_billing). To be eligible, the credit request must be received by us by the end of the second billing cycle after which the incident occurred and must include:   1. the words “SLA Credit Request” in the subject line; 2. the dates and times of each incident of non-zero Error Rates that you are claiming; and 3. your request logs that document the errors and corroborate your claimed outage (any confidential or sensitive information in these logs should be removed or replaced with asterisks).   If the Monthly Uptime Percentage applicable to the month of such request is confirmed by us and is less than 99.9%, then we will issue the Service Credit to you within one billing cycle following the month in which your request is confirmed by us. Your failure to provide the request and other information as required above will disqualify you from receiving a Service Credit. Amazon S3 SLA Exclusions The Service Commitment does not apply to any unavailability, suspension or termination of Amazon S3, or any other Amazon S3 performance issues: (i) that result from a suspension described in Section 6.1 of the AWS Agreement; (ii) caused by factors outside of our reasonable control, including any force majeure event or Internet access or related problems beyond the demarcation point of Amazon S3; (iii) that result from any actions or inactions of you or any third party; (iv) that result from your equipment, software or other technology and/or third party equipment, software or other technology (other than third party equipment within our direct control); or (v) arising from our suspension and termination of your right to use Amazon S3 in accordance with the AWS Agreement (collectively, the “Amazon S3 SLA Exclusions”). If availability is impacted by factors other than those used in our calculation of the Error Rate, then we may issue a Service Credit considering such factors at our discretion. |

The interesting parts for the system are bold. We can recognize three SLOs: Monthly Uptime Percentage, Average Response Time and Number of simultaneous connections, with respective 99, 9%, 500 ms and 500 client connections values. The three SLOs are defined in the “Definitions” part of the document.

### Uptime percentage anomaly

With the web application and FRS regularly active, Bob performs his normal daily security controls. During these ones, FRS notifies him an anomaly on the SLO “Monthly Uptime Percentage" (MUP) of Amazon S3 service. At the same time the system warns the cloud provider of the anomaly because it could be caused by an attack.

In order to catch the anomaly, the system extract the minimum MUP value from SLA and how calculate it. According to the S3 SLA definition, MUP is the difference between 100% and the average of the Error Rates calculated every five minutes for a month. Error Rate, in turn, is the rate between requests with internal server errors (marked with the codes 500 and 503 in HTTP responses) on the total of request in a five minute period. The system recognized a suspected MUP difference equal to 0,1 % on the value contained in the SLA document. The SLA value is 99,9% and real value is 99,8%. According to the parameter definition, a decrease of monthly uptime percentage corresponds to an unusual increase of internal server errors, particularly in the last three days. This problem could be caused by an attack on Amazon S3 server (DDoS, DoS).

## Cloud Forensic Readiness Tool (CFRT) for SLA violations recognition

Therefore, the purpose of this project is create a forensic readiness system in order to recognize SLA violations in a cloud system. Then, new information were used as metric in order to evaluate if they are useful in attack prediction. The principal task of system is to compare the values contained in the SLA with the real time measurements calculated from log.

The system will have two different inputs:

1. Values extracted from SLA documents with Information Extraction (IE) techniques.
2. Logs delivered by cloud provider.

## Working Environment

Amazon was the starting point of our research, for the good documentation provided and the easy to use RESTful API. Furthermore, Amazon provide a free account (with some limitations) to use a real cloud service and APIs to programmatically interact with its services. Among all Amazons services, S3 was chosen to analyze. This choice was made for the good log services provided by Amazon S3 and the well-specified value for the uptime in the S3 SLA. All the detailed information on S3 logs are consultable at the online guide [4].

# CFRT

## System Design

### Architecture

Proposed system aims to aggregate information from provided logs, in order to monitor any violation of the SLA values. To do it, SLA are translated into structured objects containing a “formula” object.

This formula is used to aggregate data from logs and retrieve the real value of each SLO. The provided formula is evaluated using a mapping logic, in combination with JEval. Which is a Java library for valuation of mathematical formulas written as string value. This tool will be better explained further.

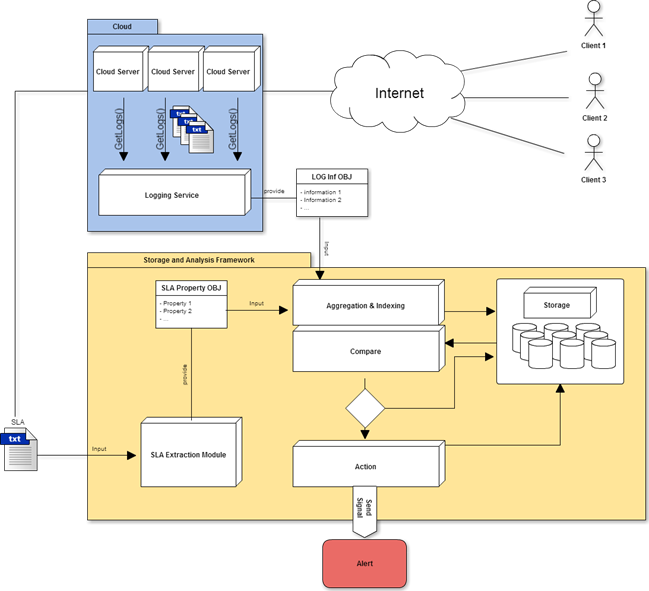


Figure 4 - First proposed architecture.

From the picture above, it’s possible to see different modules:

**SLA Extraction module**: this component of the system has the responsibility of translate the SLA document, written in natural language, into a structured language. This is achieved using an information-extracting tool called GATE. The output of this module is the SLA object that contains a collection of SLO, each one with its formula defined in the SLA, that will be used to calculate the value of the SLO.

**Logging Service module**: This module is provided by the cloud service. In our case of study, information contained into the logs can be accessed with an https request, after the user authentication. Moreover, sometimes, cloud provider support some tool that automatically sends logs to a configured remote machine, using secure connection protocol.

The system stores information from logs in a simple text file, waiting for the others modules.

**Aggregation and indexing module**: This module has the responsibility to organize the input coming from the SLA extraction module and the Logging Service module to be stored into a database. For example, this module will parse all log entries adding the field that stores the date in milliseconds. In this way will be easier access programmatically to some entries basing on the time. After this pre-processing phase, data are stored using both relational and not relational databases.

**Compare:** also called “Evaluation module”, this module is the most important and the most challenging to realize. The evaluation module will retrieve the information from database, getting all SLAs and SLOs. Once this is done, it start to make queries on log database, according to formulas specified for each SLO object.

Data coming out from the evaluation module are aggregate data, in other words, these values are results of the SLO formulas evaluation. After the matching with the declared value into the SLA , these values are stored into the non-relational DB using the storage module for further analysis.

**Storage module**: The storage module consist of two databases. The system use one relational SQL database for storing SLA and SLO object. On the other hand, it use a non-relational DB, running on MongoDB platform, to store logs events, violations, attacks and aggregate data.

The non-relational database was chosen to store logs because the big amount of queries that the system will do. Our system have to deal with live data. A monitoring system needs good performance in order to discover an SLO violation or an attack on time.

The communication between the storage module and others modules is realized using Mongo Java Drivers. This library provides APIs to query the non-relational DB running locally (or if necessary remotely on a remote machine).

**Action module:** This simple module has the role to fire up alarms if the calculated value from logs is worse than the declared value for any SLO. For now, the warning system consist in a simple warning popup that notify the users if it revealed an attack or an SLO violation.

The proposed architecture has been kept during the implementation phase, but some names changed in order to improve the understanding of module names. The following paragraph explains implementation details.

## System implementation

### Importing of logs

In a theoretical use, CFRT should retrieve log file directly from the cloud service provider. In our case of study, generated logs were stored locally in log files and hence imported into the non-relational DB.

MongoDB was chosen as platform to host the non relational-database. For the SQL database a normal instance of MySQL database is used. To achieve the objective of have a proof of concept faster, we skipped the implementation of one part of module that take events from S3 logging bucket. Instead, a method is provided for the loading of the information directly from a local file.

### Mongo DB tables description

CFRT calculates different values for each SLO declared in the SLA document. All data produced during this process are stored into the MongoDB database. For a better understanding of query and data line of code reported below. Here is given the description of all NoSQL collections used to store data.

Data collection contains the evaluation of all SLOs. The structure of the entries of this collection is the following (words in green are keys):

{ "\_id" : ObjectId("553f9d77e4b0a4f6dd85020b"), "Type" : number\_simultaneous\_connections", "Value" : 11, "Time" : NumberLong("1423969733000") }

Where \_id is an auto assigned id.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Value example | Source |
| Type | SLO name. | String | “average\_response\_time” | SLA/CFRT |
| Value | Value of SLO calculated from logs. | Float | "8.307317073170731" | CFRT |
| Time | Time (in milliseconds) used to retrieve from log the last value used in evaluation process. | Long | "NumberLong("1440372901000")" | CFRT |

Table 1 - Description of fields of collection "data"

Instead, here is an example of violation entry:

{ "\_id" : ObjectId("553f9d77e4b0a4f6dd85020c"), "SLO" : "numberOfConnections", "Difference" : 1, "Unit" : "num", "Time" : NumberLong("1423969733000") }

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Value example | Source |
| SLO | SLO name. | String | "hourlyUptimePercentage" | SLA/CFRT |
| Difference | Numeric value of difference between value calculated from log and SLA declared value. | Float | "8.307317073170731" | CFRT |
| Unit | Is the measure unit of difference value. | String | “%” | SLA |
| Time | Time (in milliseconds) used to retrieve from log the last value used in evaluation process. | Long | "NumberLong("1440372901000")" | CFRT |

Table 2 - Description of fields of collection "violations"

The collection “*log*” stores all fields of S3 access logs (All fields were described in the table 4). It is described the only field created during the aggregation phase by the CFRT program and not explained before in this document.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Value | Source |
| Time | Time stamp converted in milliseconds as POSIX time. This field is used from CFRT to retrieve data from logs. | Long | "NumberLong("1440372901000")" | Log/CFRT |

Table 3 - Description of fields of collection "log"

Here is reported an entry of log collection from MongoDB:

{ "\_id" : ObjectId("55e81748e5a0786af2ddefd9"), "BucketOwner" : "5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b", "BucketName" : "mybucketwebsite", "Day" : "24", "Month" : "08", "Year" : "2015", "Hours" : "00", "Minutes" : "00", "Seconds" : "28", "Time" : NumberLong("1440367228000"), "GMT" : "+0000", "RemoteIP" : "192.168.60.28", "Requester" : "-", "RequestID" : "DJCEO5W0YENKV63E", "Operation" : "WEBSITE.PUT.OBJECT", "Key" : "littleS3-2.3.0/Bucket/file5", "Request" : "PUT", "URI" : "/littleS3-2.3.0/Bucket/file5?atk=0", "HTTPVer" : "HTTP/1.1", "HTTPStatus" : "200", "ErrorCode" : "-", "ByteSent" : "0", "ObjectSize" : "0", "TotalTime" : "36", "TurnAroundTime" : "36", "Referrer" : "-", "UserAgent" : "curl/7.38.0", "VersionID" : " " }

Keys are green highlighted. These keys have same name of S3 fields, as specified inside documentation page.

Once the database is initialized, our system provides an easy-to-use object for querying and managing database.

### Middleware

Referring to the figure 4, It’s defined as system middleware the set of modules: Aggregation and Indexing, Evaluation, Action and Alert. The first two modules form the Matching module. This module takes in input one or more SLA objects recovered from SLA database and logs events from logs database (which is described in the previous section), calculates real SLO values and compares them with values contained in the SLA object. In the next paragraphs it is described step-by-step this process.

AWS Dictionary and Formulas

In order to evaluate a real value from logs for a specific SLO, the system needs to evaluate a formula for each SLO. When we talk about a “formula”, the meaning is one or more mathematical operations applied to one or more variables and/or numerical constants. Variables need an extra evaluation before we can start the entire formula evaluation. Therefore, the system have to translate a variable in a query and then retrieve its numerical value querying logs information.

Before the translation process, we need to manipulate the formula until it has only atomic values, or values that cannot have further inner definitions.

In order to translate each atomic variables in query, it was implemented the class *AWSDictionary*. An inner class variable contains an ADT map that has the name of a formula variable as key and a parametrized MongoDB query as value. Using this class variable, we can easily translate a string into a query, and then query the database to retrieve a numerical value.

Parameters of queries are the upper and the lower bound of the time value. Time value is expressed in number of milliseconds as a Java Calendar Time. This parametrization is necessary to have the right result set based on time windows. In other words, every SLO has his own time window based on the evaluation period as specified into the SLA.  Moreover, because into the present database are stored all entries of one month that we need to filter query by query.

To use the AWSDictionary class and retrieve the right query, it’s necessary invoke methods setLimInf(Long time) e setLimSup(Long time) to update the upper and lower bound. Then, the method translate(String key) will return the DBObject that is the query matching the given atomic variable (key).

The following line of code is an example of the dictionary entry:

dictionary.put("error “ InternalError ” or “ ServiceUnavailable ”", "{ $and : [ {$or: [{ErrorCode : \"InternalError\"}, {ErrorCode : \"ServiceUnavailable\"}]}, {$and : [{Time: {$gte : "+limInf+"}}, {Time:{$lte : "+limSup+"}} ] } ] }");

In future developments new keys can be added to the map for AWS provider. For a new provider is recommended the implementation of a new dictionary.

Being a static class, the AWS dictionary needs a policy of access for each thread. We have realized this policy using a lock policy on the static class. Before use the dictionary every thread has to check the lock dictionary variable. In the future implementation, in order to solve the concurrency problem, the dictionary class will have multiple instances, one for each thread.

### DBManager

The DBManager class stands for the connection bridge between the Java logical architecture and the MongoDB platform that manages the read/write of data from/to the non-relational DB. All classes that needs to store or retrieve data from MongoDB uses the DBManager class.

For each collection, the system create a new static parameter. The references to these parameters are kept into a special class called ObjectHolder. As we can see in the following pictures.

The DBManager class belongs to the Aggregation and Indexing modules. Indeed, it has the responsibility of gathering data from logs files, that is a simple text file and then, manipulate this data, in order to load them into the right collection. The method that satisfy this responsibility is LoadDB(String path). This method, take as argument the path of the log file, and using a regular expression manipulate the data. This approach is similar to the one used for the generation of the logs file with the Perl script. During this method is generated the new Time field. Then every log event is written in the “data” collection using Java MongoDB Drivers.

At present status the LoadDB(String path) method can parse only logs written as the S3 access log.

For the future implementations (for a new log format) will be necessary to write a new regex to read data from the log text file.

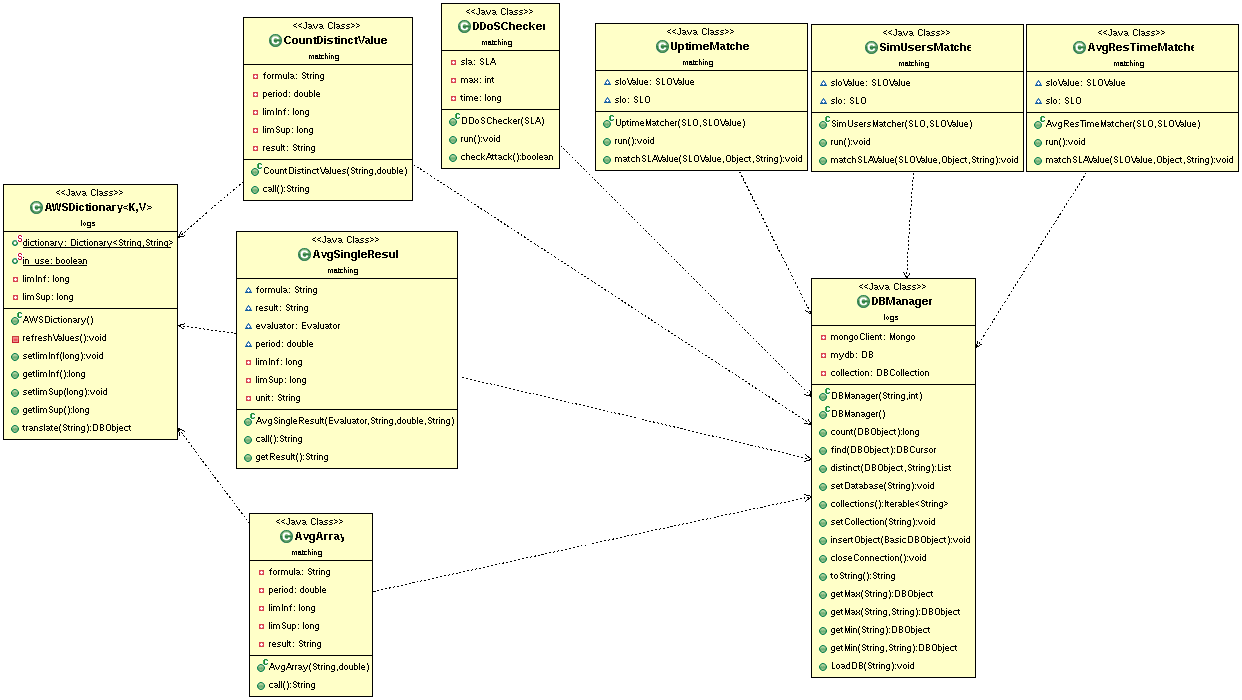


Figure 5 - Class diagram that shows classes that access to DBManager utility class.

### System Modules

This paragraph describes the actual organization of the system.

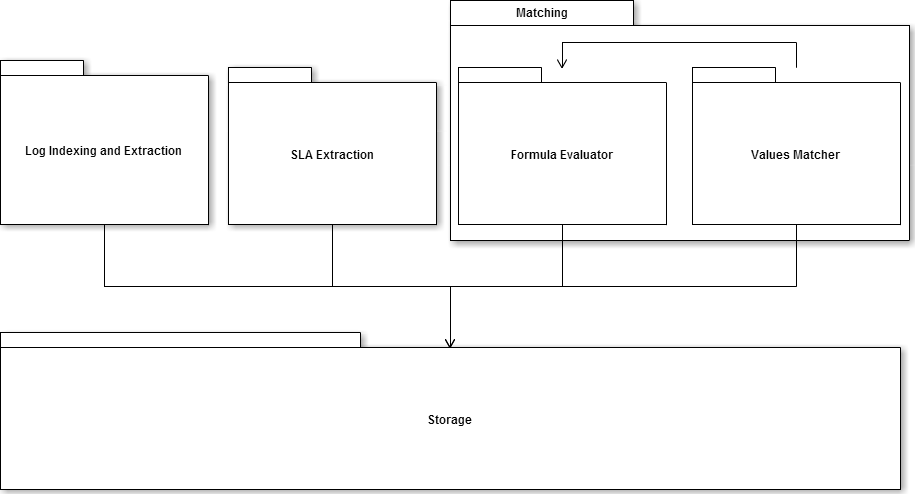


Figure 6 – CFRT system modules.

The system core was divided in four big packages. This organization is the direct evolution of the proposed logical architecture described in previous paragraph.

* SLA Extraction: in this module, using as input SLA document and using Information Extraction techniques SLO values and their definitions are extracted and stored into a relational database.
* Log Indexing and Extraction: In it, service logs are extracted from the provider server and stored in a non-relational database.
* Matching: it is the central and most important module. It has as input SLA information and logs (both extracted from DB). In it we can recognize two submodules:
  + Formula Evaluator: using the definition present in SLA document and information contained into logs database, it calculates the real SLO value real-time.
  + Values Matcher: using SLO value extracted from SLA database and the real value calculated by Formula Evaluator, it matches the two values and checks eventual SLO violations.
* Storage: in this module are present all the functions used to manage SLA and logs database, other than to store all information about violations and eventual attacks.

### System Running

At the first start, the system shows to the user a graphic interface to configure paths of different components: the GATE folder, the jape file, the SLA file and the log file. Once the user clicks on the OK button, the system runs GATE using the jape file, in order to retrieve information from the SLA. All information gathered by GATE are stored in the MySQL database. Then, the system loads the provided log file into MongoDB collection. Than the system invokes the matching module.

### ***Formula extraction and real value computation***

For calculate real values we need to know “how” calculate them: for this purpose we need to use SLO formulas. These formulas, in our system, are retrieved from the field "form" of Definition table (and JAVA object). Every Definition is connected to his SLO with a foreign key (named definition\_name) into SLO table. For matching we have to obtain a mathematical formula so that we can calculate SLO numeric value using a library, called JEval ([jeval.sourceforge.net](http://jeval.sourceforge.net/)). It uses an evaluator that, taken as input a mathematical formula, parses it and calculates correspondent value. Our formulas, even though manipulations made with GATE and JAVA, have one or more atomic elements written in natural language. There are information (for example “total number of requests”) derivable from data present into log database. We need a data structure that maps every atomic element with the MongoDB query that can calculate it. For this purpose we use a map that has atomic element string as a key and the correspondent MongoDB query as value. This map, for the difficulty to retrieve most of the information (often present in documentation or in other documents however different than SLA), could be filled manually and updated with the addition of new services supported in our system.

Most of the entries, however, will be valid for any provider (like the correspondences HTTP codename of the error). Obviously, every element added must be documented with the reason of value assignment.

Only three SLO are implemented in our system: MUP parameter (using the scenario described in chapter 3), Average Response Time and Number of Simultaneous Connections.

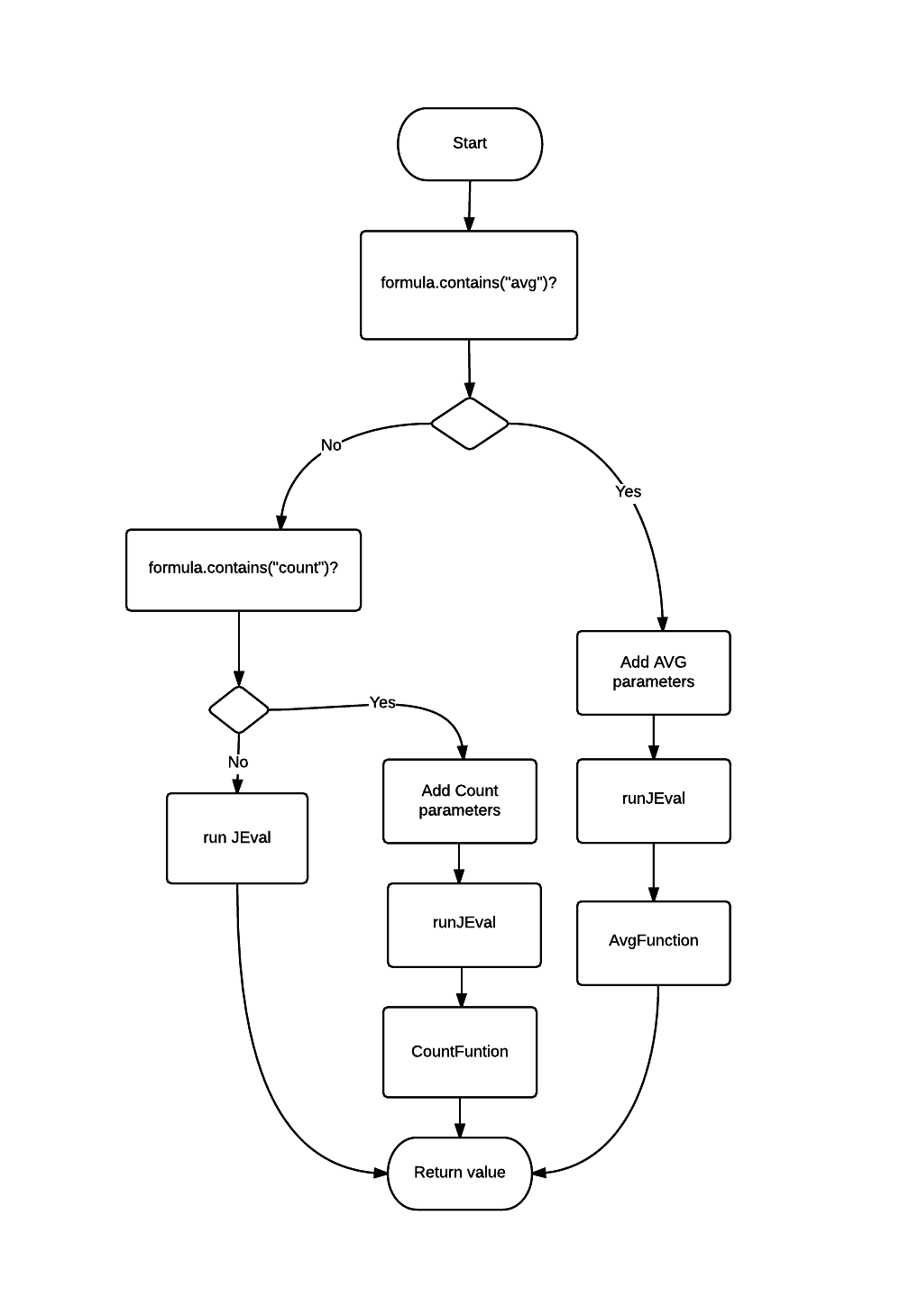
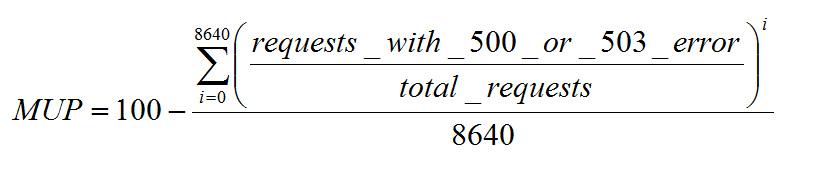


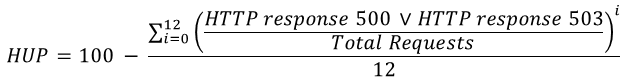
Figure 7- Flow chart for formula evaluation

**Uptime**: Using the scenario described in chapter 3, we implemented MUP parameter. According to its definition, MUP formula is:

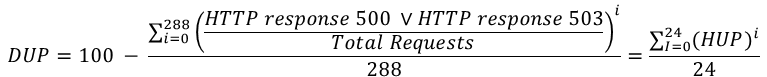


Equation 1 - Monthly uptime percentage (MUP) formula.

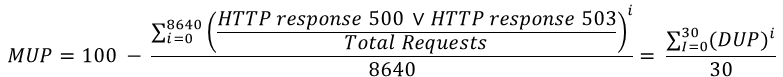
There are two atomic elements “Requests with\_500\_or\_503\_error” and “Total requests” while 8640 is the number of 5-minute intervals in a month. Therefore, every five minute we need to calculate the rate between these two values. However, for catching an attack of short period, a month could be a too long period of measure. For this motive, there are two measures: Hourly Uptime Percentage (HUP) and Daily Uptime Percentage (DAP). All three measures can be calculated starting from HUP.



Equation 2 - Hourly uptime percentage (HUP) formula



Equation 3 - Daily uptime percentage (DUP) formula



Equation 4 - MUP formula using DUP values.

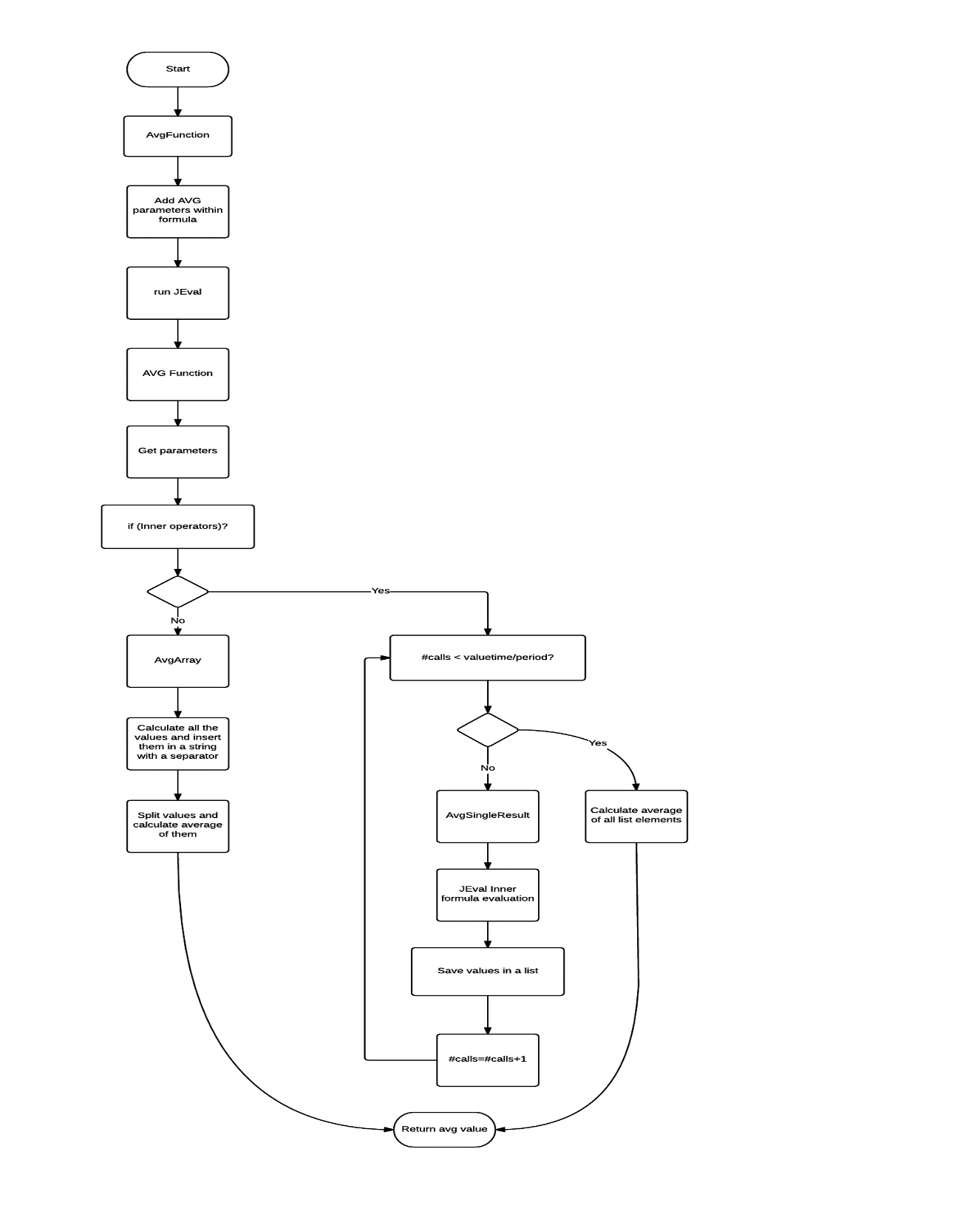


Figure 8 - Flow Chart for Average function evaluation.

Our system must calculate HUP every hour and, after 24 hours, compute the average of all the previous values. After 30 days, we can compute the average of all the DUP to calculate MUP. For this purpose, we use two circular arrays. Every uptime value obtained is stored in a MongoDB collection. The decision was to use this method rather than three threads that calculates respectively HUP, DUP and MUP to avoid repeated calculations.

The task that find HUP is a thread Calculator launched every hour that evaluates the formula using JEval. For mapping implementation, we write a custom JEval function Avg for average. This function every period (in our case 5 minutes) launches a Callable thread that searches atomic elements into formula and, using the map, performs the correspondent MongoDB queries. The system replaces calculated values and JEval evaluates the formula that is variables free. Its value is returned to Avg function that performs the average of all values computing the average Error Rate.

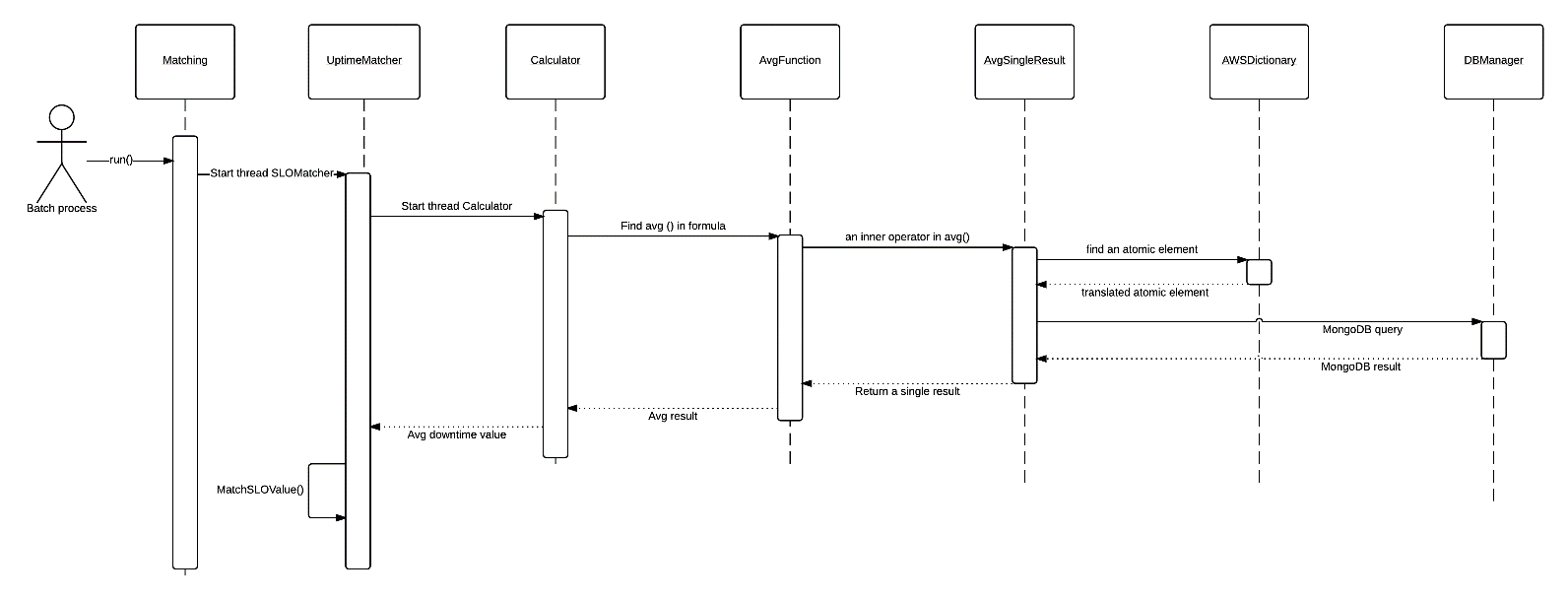


Figure 9 - Uptime evaluation and matching.

**Average Response Time:** Unlike the previous parameter, Average Response Time isn’t present in the public Amazon S3 SLA. We define it using the European guidelines definition and, for know its value, we use a benchmark, published by Nasumi[3], on which is documented an average

response time of 500 ms every 5 minutes. According to the EU guidelines, Average Response Time (ART) is the average of all requests response times from each period in the billing month cycle. Response time is only the Turn-Around Time without transmission time, because it is related to the file size. ART formula is simple and is the following:

The SLO implementation is similar to the previous one. A thread Calculator is launched every 5 minutes evaluating the formula using JEval. Avg function calculates the average of all response times (field Turn-Around time) using a Callable Task that extracts from logs database all the requests in the five minute period. The average result is returned to Calculator for the matching between real value and SLO value.

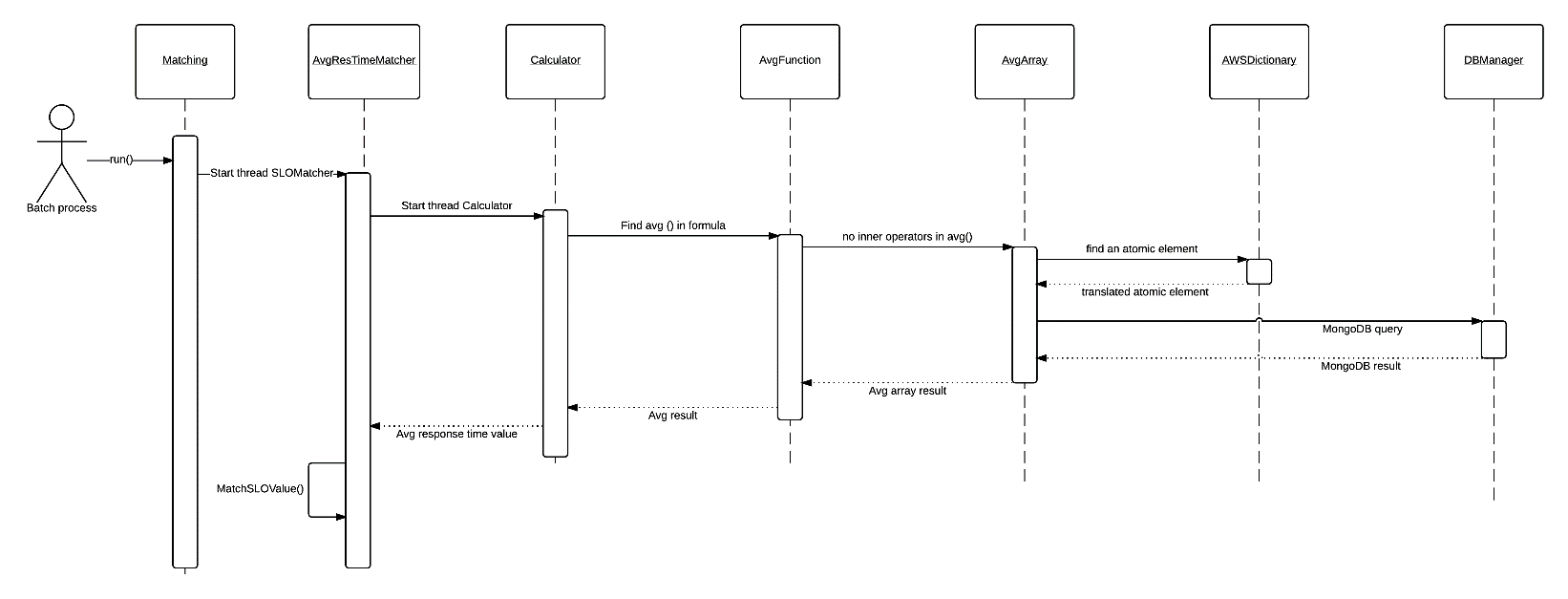


Figure 10 - Average response time evaluation and matching.

**Number of Simultaneous Clients:** Exactly as the ART parameter, the Number of Simultaneous Clients (NSC) is not present in the public Amazon S3 SLA. We define it using the European guidelines definition. According to them, it is the number of separate cloud service customer users that can use the service in the same time for each period. Its value could be negotiated because depends on the website importance. In our scenario Bob has a personal wiki website and the number of connections couldn’t be too high. Then, the number of connections in a single minute is supposed to be 500, a little period of time that allows us to catch a short attack (that can be a DDoS). NSC formula is a simple count of all the different IP numbers that perform a request in the period time:

The SLO implementation is similar to the previous parameters. A thread Calculator is launched every 5 minutes evaluating the formula using JEval. A new Count function use a Callable Task that extracts from logs database the count of the distinct IP numbers (field Remote IP in MongoDB) each minute. The result return to Count and from it to Calculator for the matching between real value and SLO value.

Matching between real value and SLO value: After calculating every real value, the system sends it and the corresponding SLO value object to the match function: it extracts value and operator (like <= or =) from SLO value object and builds the match formula. This is evaluated by JEval that returns the boolean result of the match. If the result is negative, we calculate the difference between real value and SLO value and, after store the violation into MongoDB, notify the violation to user.

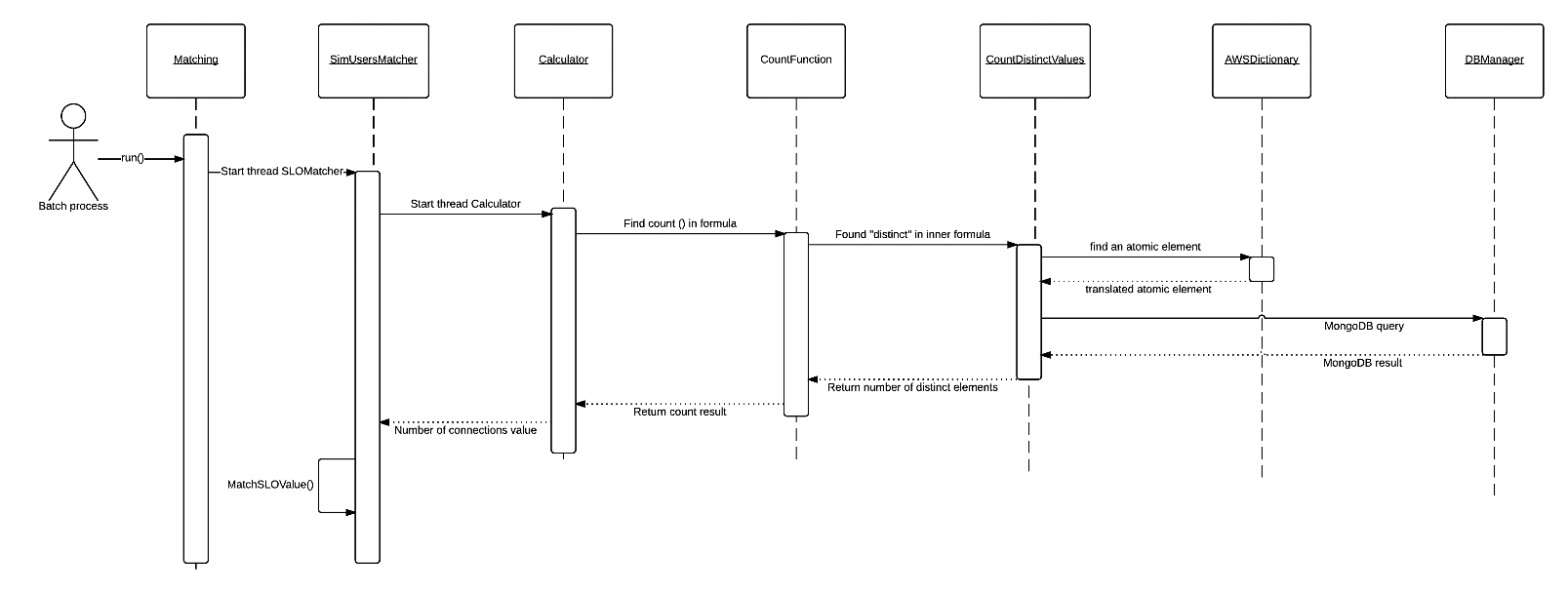


Figure 11 - Number of simultaneous client evaluation and matching.

**Attack Checker:** The final purpose of CFRT system is to recognize a possible attack using information from SLO violations. The system uses information stored into violation database (they are saved when a matching results negative). An anomaly in all the implemented SLOs could be a suspect of DDoS attack (according to SLO classification in table 5).

The attack checker functionality is realized using an independent threat. After a waiting time equal to the maximum period of time among all considered SLOs, the attack-checker-thread checks if during the last period HUP, NSC and ART are violated and, if true, notifies it to user. In the following flow chart is depicted the attack checker logic.

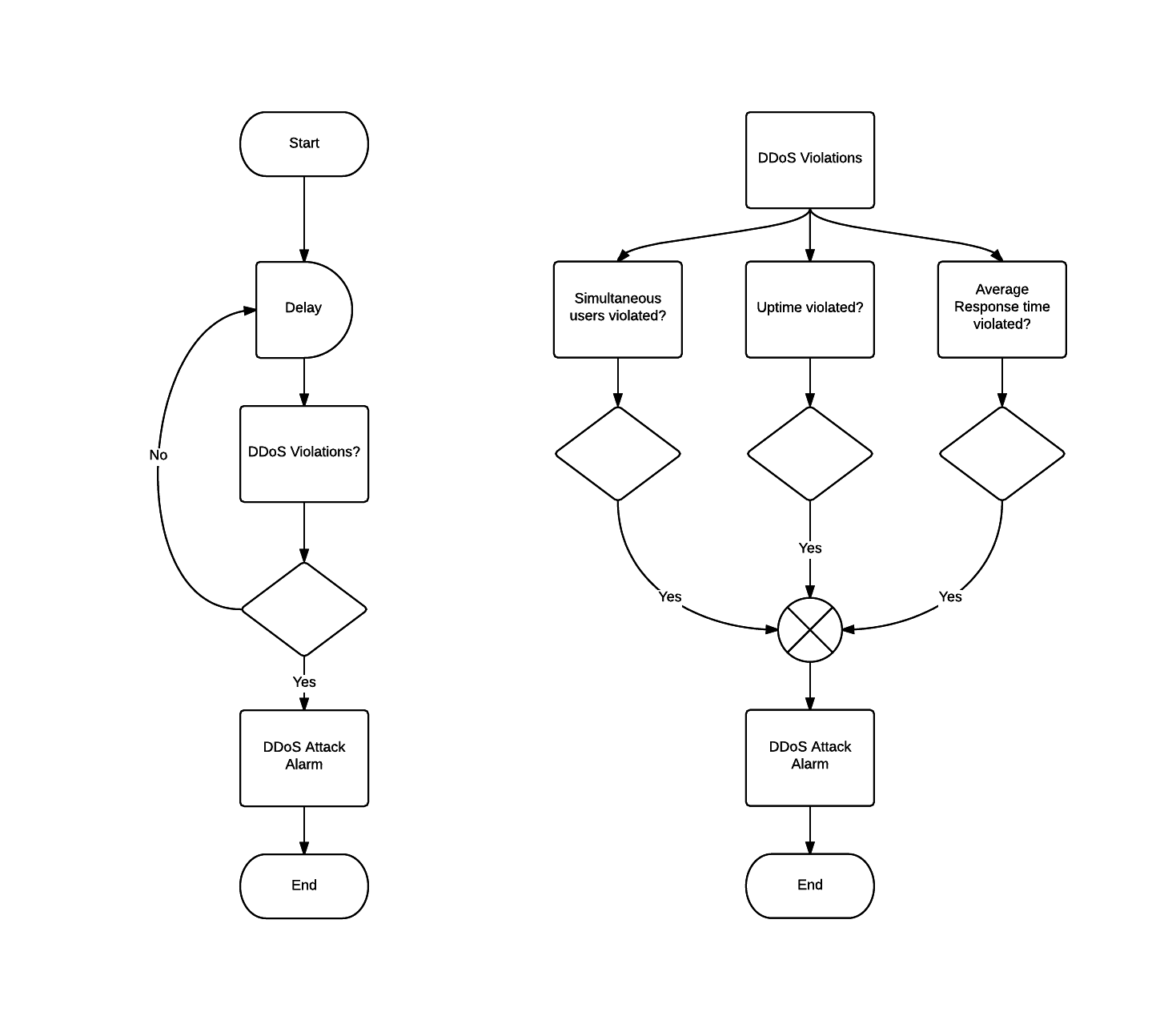


Figure 12 - Flow charts for attack checker.

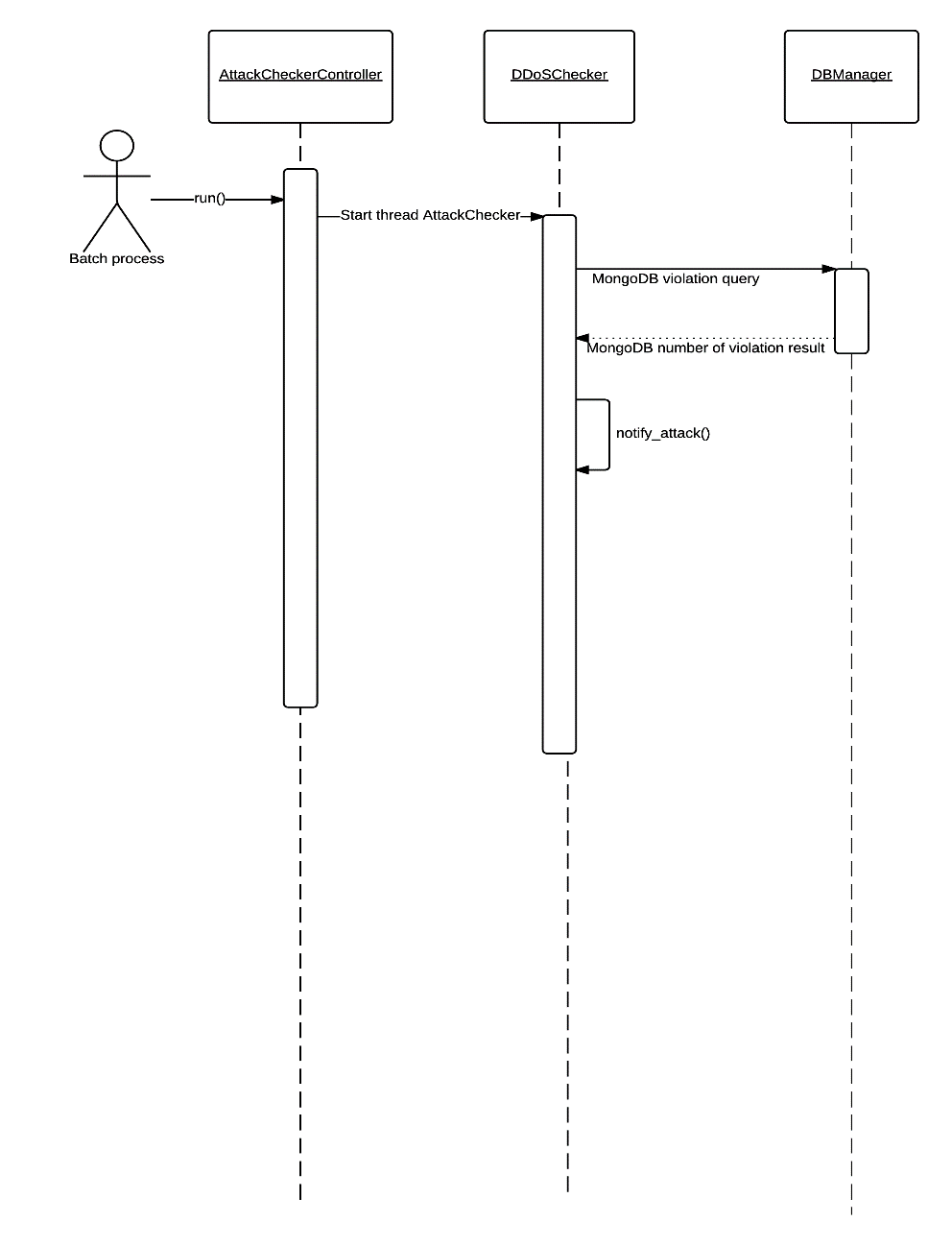


Figure 13 - Sequence diagram for DDoS attack check.

## Graphic User Interface

CFRT system needs a preliminary configuration before run. The user must download GATE Developer and execute GATE configuration that, starting from Gazetteer and Tagger files (v. 3.1.1) given by developers, produce as output an application file that extracts information from SLA documents. This configuration is reported in the user manual present in the system documentation. After this, the system can start setting some parameters. They are contained in a file called conf\_file.txt. It is decided, to make the system more user-friendly, to use a graphical interface to insert them in the configuration file. All the parameters are file path and therefore we use a file explorer for each one.

In the following table are described the four parameters needed to start the system.

|  |  |
| --- | --- |
| Parameter | Description |
| GATE\_home | This is the path of the main folder where is installed GATE Developer. Usually is C:\Program Files\GATE\_Developer\_8.0 but it can be modified during GATE installation. |
| GATE\_app | This is the path of the application created following the procedure in user manual. |
| SLA\_file | This is the location of SLA document to analyze. This parameter is an URL: if the document is online you can set SLA\_file with the URL of the web page where it is; if it is a local file the parameter must be set as “file:C://[yourpath]”. In GUI, user can write an URL in text box or use explorer to find the file in his computer |
| Logs\_file | This is the path of the file where are memorized the provider logs entries |

Table 4: Run parameters for FRS system

In the figures below are reported some GUI screenshots:

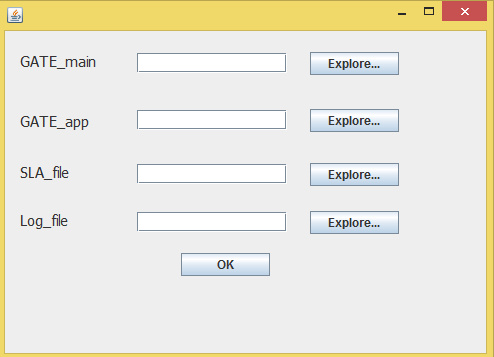


Figure 14 - "Set parameters" window in FRS system

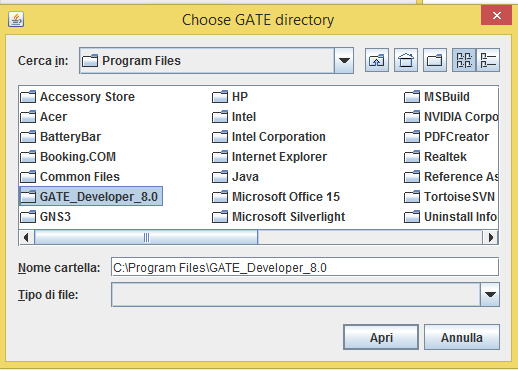


Figure 15- File explorer window

## System Output

The system output will be the alert of the recognized violation. When a an SLO value is violated the system will show the following alert.

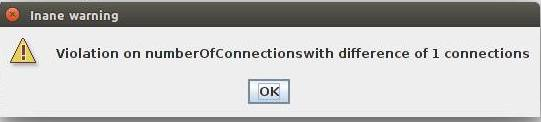


Figure 16 - SLO violation alert.

It contains information about violation name of and violation gravity. As it is explained before these information are saved into the non-relational database.

In case of attack, the following alert will be show to the user.

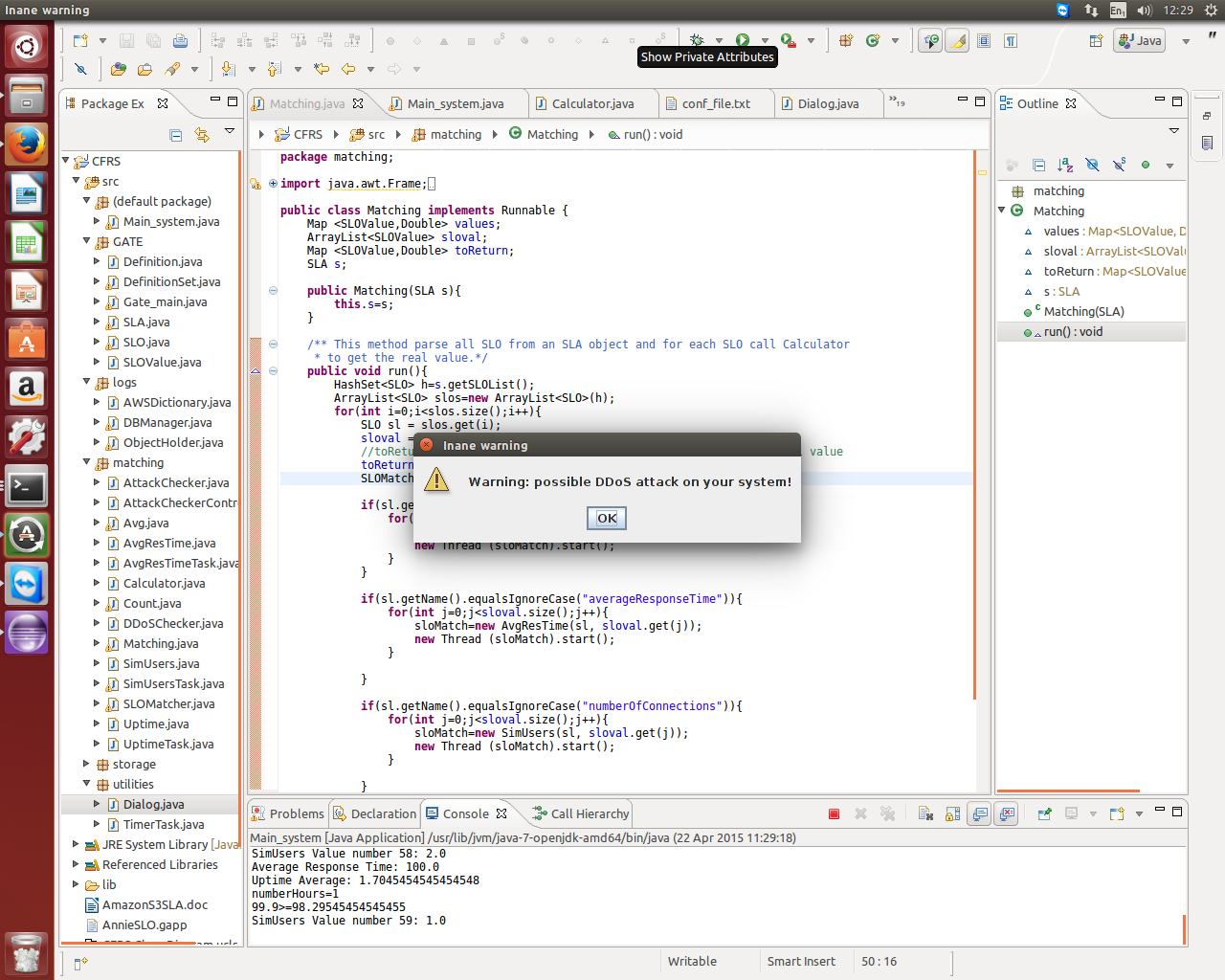


Figure 17 - DDoS attack alert.

In this case only the type of attack is notified. All detailed information are stored into the system database.

### System Testing

The last part of the work is the system testing. In all test cases input and output are MongoDB entries. Input entries are from logs database, output from violations and attacks database.

We individuated seven test cases:

### Violations:

#### DDoS attack

**Input specifications: 3 violations (response time violation, number of connections violation, hourly uptime violation)**

The first 11 entries of S3 log have 11 different IPs in the first minute, response time of each request is 999 ms and the response has ErrorCode as InternalError.

Example:  5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [15/02/2015:03:07:53 +0100] 66.249.78.20 - UGBXDJ9QZGSG2EX5 WEBSITE.GET.OBJECT fanarea.html "GET /fanarea.html HTTP/1.1" 500 InternalError 0 0 9999 99999 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" -

**Output specifications**

Three entries in violations database, one in attacks database because we have an DDoS attack

**Results:**

**Violations database**

{ "\_id" : ObjectId("553664d6e4b050717035da6c"), "SLO" : "numberOfConnections", "Difference" : 1, "Unit" : "num", "Time" : NumberLong("1423969733000") }

{ "\_id" : ObjectId("553665c6e4b050717035da71"), "SLO" : "averageResponseTime", "Difference" : 20013.32, "Unit" : "ms", "Time" : NumberLong("1423969973000") }

{ "\_id" : ObjectId("55366fcbe4b0ad64725155d5"), "SLO" : "hourlyUptimePercentage", "Difference" : -1.6, "Unit" : "%", "Time" : NumberLong("1423973273000") }

**Attacks database**

{ "\_id" : ObjectId("55378610e4b0245276203cad"), "Attack" : "DDoS", "Time" : NumberLong("1423973273000") }

#### *Number of connections and response time violations*:

**Input specifications: 2 violations (response time violation, number of connections violation)**

**Input specifications**

The first 11 entries have 11 different IPs in the first minute and response time of 999 ms.

Example:  5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [15/02/2015:03:07:53 +0100] 66.249.78.20 - UGBXDJ9QZGSG2EX5 WEBSITE.GET.OBJECT fanarea.html "GET /fanarea.html HTTP/1.1" 200 - 0 0 9999 99999 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" -

**Output specifications**

2 entries in violations database

**Result:**

**Violations database**

{ "\_id" : ObjectId("553664d6e4b050717035da6c"), "SLO" : "numberOfConnections", "Difference" : 1, "Unit" : "num", "Time" : NumberLong("1423969733000") }

{ "\_id" : ObjectId("553665c6e4b050717035da71"), "SLO" : "averageResponseTime", "Difference" : 20013.32, "Unit" : "ms", "Time" : NumberLong("1423969973000") }

#### Uptime and response time violations:

**Input specifications: 2 violations (response time violation, hourly uptime violation)**

The first 5-minute entries has response time of 999 ms and  InternalError as ErrorCode..

Example:  5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [15/02/2015:03:07:53 +0100] 66.249.78.20 - UGBXDJ9QZGSG2EX5 WEBSITE.GET.OBJECT fanarea.html "GET /fanarea.html HTTP/1.1" 500 InternalError 0 0 9999 99999 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" –

**Output specifications**

2 entries in violations database

**Result:**

**Violations database**

{ "\_id" : ObjectId("553665c6e4b050717035da71"), "SLO" : "averageResponseTime", "Difference" : 20013.32, "Unit" : "ms", "Time" : NumberLong("1423969973000") }

{ "\_id" : ObjectId("55366fcbe4b0ad64725155d5"), "SLO" : "hourlyUptimePercentage", "Difference" : -1.6, "Unit" : "%", "Time" : NumberLong("1423973273000") }

#### Uptime and number of connections violations

**Input specifications: 2 violations (hourly uptime violation, number of connections violation)**

The first 11 entries has 11 different IPs and  InternalError as ErrorCode.

Example:  5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [15/02/2015:03:07:53 +0100] 66.249.78.20 - UGBXDJ9QZGSG2EX5 WEBSITE.GET.OBJECT fanarea.html "GET /fanarea.html HTTP/1.1" 500 InternalError 0 0 238 238 "-"  "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" -

**Output specifications**

2 entries in violations database

**Result:**

**Violations database**

{ "\_id" : ObjectId("553664d6e4b050717035da6c"), "SLO" : "numberOfConnections", "Difference" : 1, "Unit" : "num", "Time" : NumberLong("1423969733000") }

{ "\_id" : ObjectId("55366fcbe4b0ad64725155d5"), "SLO" : "hourlyUptimePercentage", "Difference" : -1.6, "Unit" : "%", "Time" : NumberLong("1423973273000") }

#### Uptime violation

**Input specifications: 1 violation (uptime violation)**

The first 11 entries have ErrorCode as InternalError.

Example: 5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [15/02/2015:03:07:53 +0100] 66.249.78.20 - UGBXDJ9QZGSG2EX5 WEBSITE.GET.OBJECT fanarea.html "GET /fanarea.html HTTP/1.1" 500 InternalError 0 0 238 238 "-"  "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" -

**Output specifications**

1 entry in violations database

**Result:**

**Violations database**

{ "\_id" : ObjectId("55366fcbe4b0ad64725155d5"), "SLO" : "hourlyUptimePercentage", "Difference" : -1.6, "Unit" : "%", "Time" : NumberLong("1423973273000") }

#### Number of connections violation:

**Input specifications: 1 violation (number of connections violation)**

The first 11 entries have 11 different IPs.

Example:.  5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [15/02/2015:03:07:53 +0100] 66.249.78.20 - UGBXDJ9QZGSG2EX5 WEBSITE.GET.OBJECT fanarea.html "GET /fanarea.html HTTP/1.1" 200  -  0 0 238 238 "-"  "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" -

**Output specifications**

1 entry in violations database

**Result:**

**Violations database**

{ "\_id" : ObjectId("553664d6e4b050717035da6c"), "SLO" : "numberOfConnections", "Difference" : 1, "Unit" : "num", "Time" : NumberLong("1423969733000") }

#### Response time violation:

**Input specifications: 1 violation (response time violation)**

The first 5-minute entries have response time of 999 ms.

Example  5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [15/02/2015:03:07:53 +0100] 66.249.78.20 - UGBXDJ9QZGSG2EX5 WEBSITE.GET.OBJECT fanarea.html "GET /fanarea.html HTTP/1.1" 200  -  0 0 999 9999 "-"  "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" -

**Output specifications**

1 entry in violations database

**Result:**

**Violations database**

{ "\_id" : ObjectId("553665c6e4b050717035da71"), "SLO" : "averageResponseTime", "Difference" : 20013.32, "Unit" : "ms", "Time" : NumberLong("1423969973000") }

# Empirical study

Although, Amazon provided a free account for S3, our investigation needed logs with real data, and our S3 was too young to have enough information on logs. Unfortunately, it was not possible to find any existing S3 access-log file on-line. For this reason, some S3 logs were generated using LittleS3 [18]. (The generation of S3 logs will be described with more details in the next paragraph). These logs were used not only to calculate SLO values, but also during the experimentation phase, to test the importance of violations as metrics to classify logs entries.

LittleS3 is similar to S3, the Simple Storage Service of Amazon. LittleS3 as S3 organizes file into structures called "Bucket".

These structures are similar to folders of a standard file system. Unlike LittleS3, where bucket have a limited capacity, with S3 every bucket have no capacity limit. These limit was taken in mind during the experimentation phase. Together with the differences of hardware used for host LittleS3. The adoption of a different and open source platform for experimentation phase was an obliged choice. As written before, to create data for testing CFRT it was necessary a big dataset. Hence, the choice of using a free and open source platform that that simulates the functionality of the original S3 to use as target of attacks simulation without bounds on costs and number of requests.

Since CFRT needed at least one month of access logs to calculate the uptime SLO, the first try was made using Apache access-logs. To obtain enough data to work with, some Tomcat access-logs have been gathered from a real running server, and then S3 logs were generated from Tomcat information using a simple Perl script to remove/generate all the plus/missing information. Moreover, in order to solve problems with long waiting times (1 month) for monthly uptime (defined in the Amazon S3 public SLA), the monthly uptime percentage (MUP) was placed side by side

with two other metrics: daily uptime percentage (DUP) and hourly uptime percentage (HUP).

With these generated data, it proved that it is possible evaluate data from logs having the SLO formula. Hence, it was necessary to have a real experimentation with a real S3 platform (or a similar service). To prove the idea, it was necessary to verify behaviour of system under attack and hence retrieve information from its logs in order to analyse them and evaluate if there were violations.

As quoted before, to generate logs of a cloud system under attack, it was used LittleS3[18].

LittleS3 is a simple web app that simulate the behaviour of a real S3 service. LittleS3 run on Tomcat, it takes HTTP request to GET, PUT or DELETE an object from a specified bucket, in the same way of what S3 does. Object organization are the same of S3. Only the limited hardware and the limited bucket size is the big difference between these two systems.

Using a Perl script (as done for the Apache access logs before), Tomcat access logs file were translated in a S3 access logs. The detail about the translation script are explained in the next chapter.

Amazon S3 access logs (as well as others Amazon services logs) are delivered on a S3 bucket. In order to keep logs file secure, this log-bucket needs to have a specific configuration. As described in the documentation [7], by default, only the bucket owner and the Amazon S3 Log Delivery group (which is the automatic system that delivers logs) have access to these files. In a theoretical use, CFRT should retrieve the file directly from the cloud service provider. In our case of study, generated logs were stored in Tomcat folder. Hence, from a local resource, log files were imported by CFRT into a non-relational DB as Mongo DB.

In order to validate the idea of a new attack prediction method based on SLO and violations, it is required a dataset that simulate accesses on S3 platform, the dataset has to be big enough to run one or more classifiers on it and have a more or less accurate prediction of the attack. One of the last test will be evaluate the importance of different SLO in DoS attack recognition.

## Generation of S3 logs Data

As described in the previous chapter, it was impossible to use a real S3 service to have access logs during an attack simulation to the Amazon platform. Hence, we used LittleS3, but before start the attack simulation and retrieve logs, it was necessary to configure Tomcat logging module.

Opening the file *server.xml* edit the log format as following:

pattern="%h %l %u %t &quot;%r&quot; %s %b &quot;%{Referer}i&quot; &quot;%{User-Agent}i&quot; %D %q"

Adding *%D* and *%q* parameters at the end will add at the combined standard log, information about time taken to process the request, in millis (*%D*) and the query string (*%q*). This information are essential during the generation of S3 access logs. Because in S3 access logs these information are registered and the translation aims to have the best likelihood with a real S3 log system.

## Attack simulation and dataset building

The dataset used for classification was populated using a particular module of CFRT. Data were taken from logs information stored in MongoDB database. A particular collection of data, called “log” stores all field of access logs. Details about MongoDB collection are written in the section *Mongo DB table description* in the chapter about the *system implementation*.

Starting from these information, following values were evaluated with five minutes interval:

* AVG (response time): average of server response time (in milliseconds).
* Sim User: number of users simultaneously connected to the service. The value is calculated counting the number of different IP address that made some request within 5 minute.
* HUP: Hourly uptime percentage value, this value is calculated following the formula specified into the Amazon SLA, and scaled on a number of interval of one hour, instead of an entire month.
* Violation: a binary field that show if there is a violation within the five minute interval.
* Attack: a binary field that reveal if there were an attack during the five minute interval. This field was used to train the classifier and verify prediction.

Within the dataset, there are requests from both standard and malicious clients.

## Test environment

As quoted before, during these simulation it was used **LittleS3** (<https://code.google.com/p/littles3/>), a little web app that uses Tomcat as platform. As S3, it allows to manage, through a web interface, objects and buckets. Files are stored only on the server, using a not distributed file system. Moreover, the hardware was not shared and hence it is easier to have control on resources and server loads.

One of the strengths of LittleS3 is Tomcat as web server. Apache Tomcat has a customizable access log service called Valve Component. The Valve Component provide an easy way to add or remove information from access log. The access log customization is explained in the previous paragraph.

## Tools

Here is a summary of all tools and software used during the experimentation process:

* Tomcat & LittleS3. LittleS3 running on a server machine, waiting for requests on the port 8080. Every request was recorded on Tomcat access log.
* SimpleDoS, is a little Java program written to simulate behaviour of a real client, that makes requests to a S3 service through web API. This tool was run on different client machines, both standards or malicious.
* Perl conversion script. A simple Perl script that makes use of regular expression to re-write data in a S3 log format. This script is better explained in the following chapter.
* CFRT. The Cloud Forensic Readiness Tool, is the software developed during this work to identify SLA violations. A part of this tool evaluate data to populate a dataset for the classification phase.
* Rapid Miner. In the last phase it was used Rapid Miner to run different classifier for DoS attack prediction, evaluating the prediction with different subsets of feature enlisted before.

## Methodology

In the figure below is explained the role of all tools mentioned before and how they interact each other.

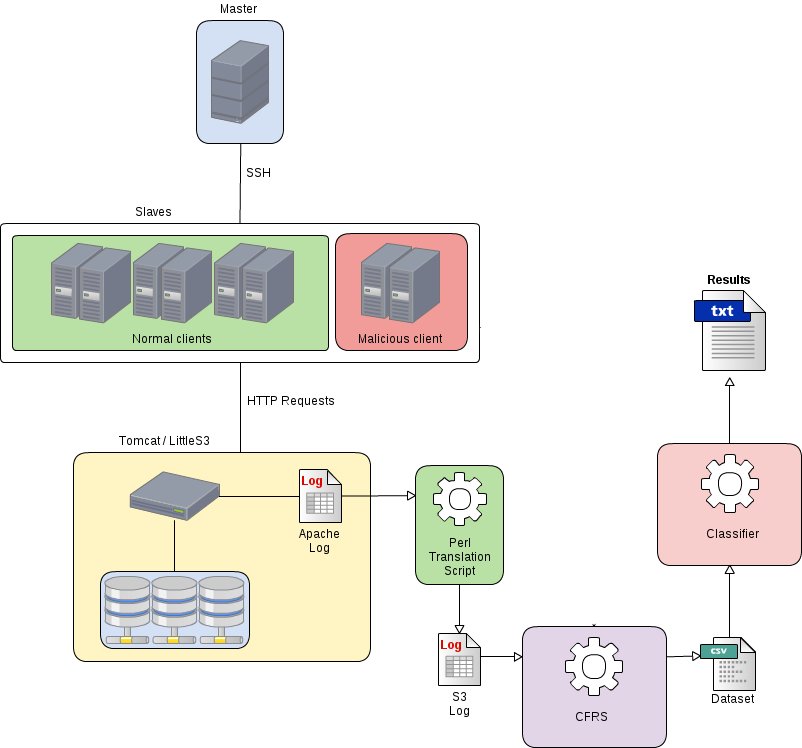


Figure 18 – Attack simulation architecture

Master sends to slaves the command to run SimpleDoS.jar with all required parameters. The command is sent using SSH protocol.

After few minutes, the malicious client starts to simulate a DoS attack, flooding Tomcat with PUT requests to LittleS3. The attack persist for five minutes and the attack routine is called every fifty minutes. After five minutes, malicious client stop the flooding routine and start to request files as a standard client. Other information about SimpleDoS tool are described in the paragraph *SimpleDoS* below.

Request done by malicious client, during the attacking routine are labelled using the query string in the URL requested. The parameter *atk* is always present, it is *atk=0* if the request comes from a standard client or a malicious client outside the attack routine. On the other hand the parameter is *atk=1* if the request belongs to a attack routine. This information is used to mark requests during the classification process.

In order to have enough data to classify, the simulation process have taken two days. After the first day, the process was restarted, Tomcat and LittleS3 were restarted and the target bucket was changed to avoid problems with storage limits of LittleS3.

Once the simulation ended, logs were translated into S3 access log format and then data were stored from CFRT into MongoDB, adding values calculated by the forensic readiness tool (details about evaluation of new values are described in the next chapter).

Hence, using the gathered data, the specified values were summarized into a csv file ready to be imported and classified using Rapid Miner suite. The classification will be repeated different times with different subsets of features, in order to understand which CFRT metrics is more important in order to predict flooding DoS attack.

## Technical specifications

In this section are explained all technical specifications about machines and how requests and attacks are automatized.

Firstly, during this simulation we will use virtual machines with following technical specs:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Normal client | Malicious client | Master | Server S3 |
| CPU | 1 core | 2 core | 2 core | 2 core |
| RAM | 512 MB | 2 GB | 2 GB | 2 GB |
| Disk | 20 GB | 20 GB | 20 GB | 20 GB |

Table 5- Hardware specification for machine used during the simulation process

During the data generation we will use:

* 1 Server S3 machine (IP: 192.168.60.17)
* 1 Malicious client machine (IP: 192.168.60.6)
* 1 Master machine (IP: 192.168.60.2 Public IP: 193.205.186.57)
* 10 Normal client machine (IP range : 192.168.60.[18-28])

### SLA used

Since the SLA document is a legal document signed from both the parties; the cloud service provider and the customer, starting from the original Amazon S3 SLA, it was created a new and more detailed SLA in order to have a private-like SLA with not only the single SLO of monthly uptime percentage(MUP). Inspired by the MUP definition, there were introduced two extra SLO, together with a definition and a formula to calculate them. Since the attack to recognize and predict was a DoS, it was chosen to implement some of SLOs identified in table 5 as feature of DoS attack, in order to evaluate the importance of each of this SLO violation during the classification process.

SLOs chosen for this evaluation were:

* Level of uptime, calculated as hourly uptime percentage (HUP).
* Average response time.
* Max number of simultaneous cloud service users.

Details about the evaluation of this SLO with formula used are better described further.

Declared SLA values for these SLO are stored into the column *value* of SQL table reported below:

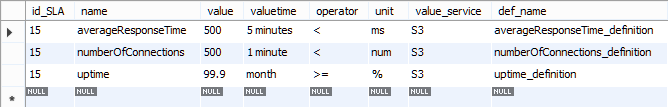


Figure 19 - SQL table of SLO used during the simulation.

### SimpleDoS

SimpleDos.jar is the little java program used to automate the experiment process. A copy of this program is installed on each client/master machine. The .jar file is launched in with different options, depending from the role of the client machine:

java -jar SimpleDoS.jar http://[server\_address] littleS3-2.3.0/[bucket\_name] [mode]

*[mode]* can have two different values:

* 0 for Normal client: it makes an HTTP request every 5 seconds with probability of 40% on a file choose randomly among the set of created files. The type of request will be chosen at random between GET (70% of requests) or PUT/DELETE (30% or requests);
* 1 for Malicious client, it acts as a normal client, every hour, the malicious routine is lauched. It makes a pool of 10 threads, each one makes PUT requests sending the biggest file among the set of provided files;

When the command run, 10 files of different size will be created using the following command:

**this**.executeCommand("dd if=/dev/urandom of="+name+i+" bs="+i\*10\*1024+" count=1");

The command generates files of variable size filled up with random data. /dev/urandom (or unlimited random) is, as the name suggests an unlimited source of random data. The quality of the randomness is not as high as that of /dev/random as it uses both input from /dev/random as well as pseudo random generation algorithms.

### ***S3 Log details***

Then information provided by Tomcat access logs are used to generate information to insert into S3 access log. The following table maps information between Apache log and S3 log:

|  |  |  |
| --- | --- | --- |
| Apache example entry | Apache access log Field | S3 access log Field |
| - | - | Bucket Owner |
| - | - | Bucket |
| [10/Oct/2000:13:55:36 -0700] | Time | Time |
| 127.0.0.1 | Remote host | Remote IP |
| - | - | Requester |
| - | - | Request ID |
| "GET /apache\_pb.gif HTTP/1.0" | Request line | Operation |
| "GET /apache\_pb.gif HTTP/1.0" | Request line | Key |
| "GET /apache\_pb.gif HTTP/1.0" | Request line | Request-URI |
| 200 | Status code | HTTP status |
| - | - | Error Code |
| 2326 | Size of object returned | Bytes Sent |
| 2326 | Size of object returned | Object Size |
| - | - | Total Time |
| 7 | Time to process the request | Turn-Around Time |
| "http://www.example.com/start.html" | Referrer | Referrer |
| "Mozilla/4.08 [en] (Win98; I ;Nav)" | User-Agent | User-Agent |
|  | - | Version Id |

Table 6- Apache/S3 logs mapping

Other information concerning Tomcat access-log are available on the official documentation page [6].

Some fields in the Apache column are marked with “-”, this means there is no value that it could be used to put in the S3 logs. Indeed, for different reasons, information in the S3 logs are more accurate than information in the Apache logs.

For fields that have no matching the following methods was used to generate a value that could fit the missing fields:

|  |  |
| --- | --- |
| S3 access log Field | Generation method |
| Bucket Owner | Static field – the bucket owner is a constant value corresponding to an alphanumeric string. |
| Bucket | Static field – the bucket name is a constant value corresponding to the name of bucket. I have used my own bucket name ID to fill this field. All events inside the log have the same Bucket ID. |
| Requester | Static field – for WEBSITE.GET.OBJECT operations on public object from a non-authenticated user, the requester ID is a “-“ character. In this case of study, all files have been accessed with unauthenticated WEBSITE.GET.OBJECT requests. All requests have the same “-” value for the requester field. |
| Request ID | A unique alphanumeric string. The Perl script generates a unique alphanumeric string for each request. |
| Error Code | Depending on the HTTP response, this field reports the error name. The Perl script generates the name of the error according to both the HTTP response number and the error code provided in the Amazon S3 documentation. |
| Total Time | Total time value depends from the size of the requested object. This value is calculated as: [Turn-Around Time]+[Network Write Speed]. Using some real S3 logs, I’ve calculated the mean of the Network Write Speed doing get requests on objects (text and images) of different size (1MB, 2MB and 5MB) in different hours of day. According to my experiments and results gathered from (http://cloudharmony.com/), the Perl script uses a download speed that float randomly in a range of values that goes from 30MB/s to 70MB/s. (More researches and real data analysis could improve this range. This is not the objective of this project.) |
| Version Id | This field in not used for Website service. For each event in the log it’s generated a “-” constant for this field. |

Table 7- Generation methods for missing fields

As additional information there is the table that explain all fields belonging the S3 access-log as showed in the Amazon S3 documentation page.

|  |  |  |
| --- | --- | --- |
| Field Name | Example Entry | Notes |
| Bucket Owner | 79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be | The canonical user id of the owner of the source bucket. |
| Bucket | mybucket | The name of the bucket that the request was processed against. If the system receives a malformed request and cannot determine the bucket, the request will not appear in any server access log. |
| Time | [06/Feb/2014:00:00:38 +0000] | The time at which the request was received. The format, usingstrftime() terminology, is [%d/%b/%Y:%H:%M:%S %z] |
| Remote IP | 192.0.2.3 | The apparent Internet address of the requester. Intermediate proxies and firewalls might obscure the actual address of the machine making the request. |
| Requester | 79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be | The canonical user id of the requester, or the string "Anonymous" for unauthenticated requests. This identifier is the same one used for access control purposes. |
| Request ID | 3E57427F33A59F07 | The request ID is a string generated by Amazon S3 to uniquely identify each request. |
| Operation | REST.PUT.OBJECT | Either SOAP.operation, REST.HTTP\_method.resource\_type or WEBSITE.HTTP\_method.resource\_type |
| Key | /photos/2014/08/puppy.jpg | The "key" part of the request, URL encoded, or "-" if the operation does not take a key parameter. |
| Request-URI | "GET /mybucket/photos/2014/08/puppy.jpg?x-foo=bar" | The Request-URI part of the HTTP request message. |
| HTTP status | 200 | The numeric HTTP status code of the response. |
| Error Code | NoSuchBucket | The Amazon S3 [Error Code](http://docs.aws.amazon.com/AmazonS3/latest/dev/ErrorCode.html), or "-" if no error occurred. |
| Bytes Sent | 2662992 | The number of response bytes sent, excluding HTTP protocol overhead, or "-" if zero. |
| Object Size | 3462992 | The total size of the object in question. |
| Total Time | 70 | The number of milliseconds the request was in flight from the server's perspective. This value is measured from the time your request is received to the time that the last byte of the response is sent. Measurements made from the client's perspective might be longer due to network latency. |
| Turn-Around Time | 10 | The number of milliseconds that Amazon S3 spent processing your request. This value is measured from the time the last byte of your request was received until the time the first byte of the response was sent. |
| Referrer | "http://www.amazon.com/webservices" | The value of the HTTP Referrer header, if present. HTTP user-agents (e.g. browsers) typically set this header to the URL of the linking or embedding page when making a request. |
| User-Agent | "curl/7.15.1" | The value of the HTTP User-Agent header. |
| Version Id | 3HL4kqtJvjVBH40Nrjfkd | The version ID in the request, or "-" if the operation does not take aversionId parameter. |

Table 8 - S3 access log fields description

There is an example of an entry taken from S3 log.

file79a59df900b949e55d96a1e698fbacedfd6e09d98eacf8f8d5218e7cd47ef2be mybucket [06/Feb/2014:00:00:38 +0000] 189.48.46.51 - 3E57427F33A59F07 WEBSITE.GET.OBJECT /photos/2014/08/puppy.jpg "GET /mybucket/photos/2014/08/puppy.jpg?x-foo=bar" 200 - 14200 14200 15 15 "http://www.puppypicturest.com/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/36.0.1985.143 Safari/537.36" -

### Pearl translation script details

The Apache access log is a text file where every entry (or event) is separated by the new line character. Moreover, every field is separated by a blank space. From this point of view, in S3 log files, events and fields have the same structure.

It’s used a regular expression to get every field separately from the Apache log entry and manipulating these values, they are written inside the new S3 log file.

Here, there is the Perl regular expression used to parse the Apache log fields:

(/^([\w\.:-]+)\s+([\w\.:-]+)\s+([\w\.-]+)\s+\[(\d+)\/(\w+)\/(\d+):(\d+):(\d+):(\d+)\s?([\w:\+-]+)]\s+"(\w+)\s+(\S+)\s+HTTP\/1\.\d"\s+(\d+)\s+([\d-]+)((\s+"([^"]+)"\s+")?([^"]+)")?$/)

After that all fields are manipulated in the right way, everything is written in the output file:

print STDOUT "5927116389e7d406047097a41cba2ef5830ad74cdaf67351d74682eeaa07ea2b mybucketwebsite [$day/$month/$year:$hour:$min:$sec $tz] $host - $requestID WEBSITE.$method.OBJECT $keyRequest \"$method $uriRequest HTTP/1.1\" $code $error $bytesd $bytesd $time $processingTime \"$referer\" \"$ua\" - \n";

The script can be executed on Unix system using the following bash command:

$ perl parser.pl Apache\_access\_log\_file > S3\_log\_file

Where *parser.pl* is the Perl script, *Apache\_access\_log\_file* is the Apache log file and *S3\_log\_file* is the output file. The script can be executed also redirecting the STDIN like the following examples:

$ perl parser.pl < Apache\_access\_log\_file > S3\_log\_file

$ cat Apache\_access\_log\_file | perl parser.pl > S3\_log\_file

These commands will create a new file with the name “*S3\_log\_file*” that will contain all information likely a real S3 access log file.

The entire script can be found at this URL:

https://drive.google.com/file/d/0B-jErA\_Gs-QYS1Z5MTF5UmdPTHM/view?usp=sharing

MongoDB

A non-relational DB was considered the best choice, since the big number of entry and the high ratio of query on a big file. Moreover, the proposed system is a real time system so one of the not functional requirement was the efficiency.

Given these considerations, we chose to use a non-relational database: MongoDB (as said before).

MongoDB is not only one of the most popular non-relational databases, but it provide some useful features.

*Flexible data model* - MongoDB’s document data model makes it easy for us to store data of any structure and dynamically modify the schema. This feature fit perfectly our necessities, because logs usually have different structures and information in them. Indeed, information into log depend from source of log.

*Highly Scalable* - Scale up or scale out horizontally, from a single server to thousands of nodes. Deploy in the cloud and across multiple data centers. Because of the fast growth of logs, this feature is another motivation that leads us to choose MongoDB.

*Expressive Query Language* - MongoDB provide a query language that allows varied query types. Moreover, it provides drivers for any programming language in order to use MongoDB query in an easy way.

## Attack Prediction idea

Attacks prediction was the proposed objective of this research, together with [8]. It was made a mapping between SLO violations and attacks. Only a section of this idea was implemented in the CFRT in order to have a proof of concept. The idea was then validated through the simulation described in the previous chapter.

Here are reported all the entire mapping.

### Attacks/SLO Mapping

An additional collection “*Attacks*” was created and it contains all possible detected attacks based . The system guesses the attack basing on which violation were detected in the last period. The following table links together SLO violation and attacks. More detailed information on this mapping can be found in [8].

|  |  |  |
| --- | --- | --- |
| SLO Name | Possible Attack | Reason |
| Level of uptime | DoS or DDoS Hijacking | If the cloud system has been unavailable more than allowed there may have been an DoS or DDoS attack caused by a hijacking threat |
| Percentage of successful requests | DoS or DDoS Hijacking | If the percentage is lower than allowed the network could be blocked under DoS or DDoS attack caused by a hijacking threat |
| Average response time | DoS or DDoS Hijacking | If the response time is upper than allowed the network could be blocked under DoS or DDoS attack caused by a hijacking threat |
| Max response time | DoS or DDoS Hijacking | If the response time is near the maximum allowed the network could be blocked under DoS or DDoS attack caused by a hijacking threat |
| Max number of simultaneous connections | DoS Hijacking | A number of connections by a single client near the maximum allowed could be a sign of an DoS attack caused by a hijacking threat |
| Max number of simultaneous cloud service users | DDoS | A number of simultaneous users connected near the maximum allowed could be a sign of an DDoS attack |
| Percentage of service provisioning requests in time | DoS or DDoS Hijacking | If the percentage is lower than allowed in time the network could be blocked under DoS or DDoS attack caused by a hijacking threat |
| External connectivity | DoS or DDoS Hijacking | If the metrics of jitter and latency are out of bounds there may have been an DoS or DDoS attack caused by a hijacking threat |
| Backup method | Data breach | Knowing the backup methods can be useful to identify possible unwanted access to backup data |
| Data transfer rate | DoS or DDoS Hijacking | If the data transfer rate is lower the average allowed the network could be blocked under DoS or DDoS attack caused by a hijacking threat |

Table 9 - Mapping between SLO and cloud attack.

*Attacks* collection have the following structure:

{ "\_id" : ObjectId("553fab4be4b0a4f6dd850257"), "Attack" : "DDoS", "Time" : NumberLong("1423973273000") }

Where \_id is the auto-generated id, Attack is the name of the attack and Time is the date/time in milliseconds when the attack is detected and notified.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Description | Type | Value example | Source |
| Attack | the name of the attack guessed by the CFRT. | String | “DoS” | CFRT |
| Time | Time (in milliseconds) used to retrieve from log the last value used in evaluation process. | Long | "NumberLong("1440372901000")" | CFRT |

Table 10 - Description of fields of Attack collection

# SLA Violations and DoS Attacks evidence

Once the simulation phase was complete, it was possible to classify data and evaluate importance of each feature in DoS attacks prediction. Feature and methodology used are described before in the chapter **Empirical Study**.

The classification and validation process was realized using Rapid Miner [23].

This software suite allows to choice among different classifier, train them to retrieve a model for data prediction. Hence, using the X-Validatation operator it was performed a cross-validation in order to estimate the statistical performance of a learning operator on data set. The cross validation it is mainly used to estimate how accurately a model (learnt by a particular learning operator) will perform in practice.

The learning operators were the following:

**Linear discriminant Analysis**: this operator performs linear discriminant analysis (LDA). This method tries to find the linear combination of features which best separate two or more classes of examples. The resulting combination is then used as a linear classifier. Discriminant analysis is used to determine which variables discriminate between two or more naturally occurring groups, it may have a descriptive or a predictive objective.

**Naïve Bayes**: A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be 'independent feature model'. In simple terms, a Naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class (i.e. attribute) is unrelated to the presence (or absence) of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 4 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a

Naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple.

The advantage of the Naive Bayes classifier is that it only requires a small amount of training data to estimate the means and variances of the variables necessary for classification. Because independent variables are assumed, only the variances of the variables for each label need to be determined and not the entire covariance matrix.

**Decision Tree**: A decision tree is a tree-like graph or model. This representation of the data has the advantage compared with other approaches of being meaningful and easy to interpret. The goal is to create a classification model that predicts the value of a *target attribute* (often called *class* or *label*) based on several input attributes of the dataset. Each interior node of tree corresponds to one of the input attributes. The number of edges of a nominal interior node is equal to the number of possible values of the corresponding input attribute. Outgoing edges of numerical attributes are labeled with disjoint ranges. Each leaf node represents a value of the *label* attribute given the values of the input attributes represented by the path from the root to the leaf.

## Results

Result tables report values of precision and accuracy for each classification.

Items were divided in two distinct classes, using two labels:

* **A** for items that actually belong to an interval of attack. (*positive*)
* **B** for items that doesn’t belong to an interval of attack. (*negative*)

In this case, one gives more importance to have a high precision on predicted B elements and an high recall for true A items. Since the objective is to predict all attacks, but without too many false negative elements.

### Linear Discriminant

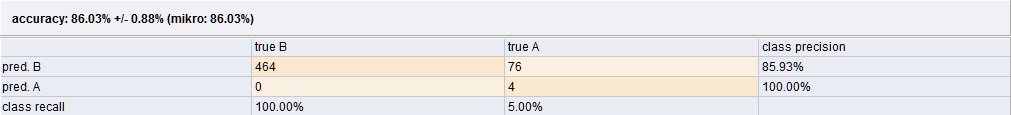
The first analysis was made using a linear classifier, the **linear discriminant analysis** operator, using the entire set of features (or attributes).

Figure 20 - Results for linear discriminant analysis (LDA) classifier.

As reported in figure 17, the LDA gives many false positive. 76 over the 80 of positive items, belonging to the A class. Since the recall of A class is so low, maybe **there is no linear dependency among features**. For this reason, it was used others not-linear classifier.

## Naïve Bayes

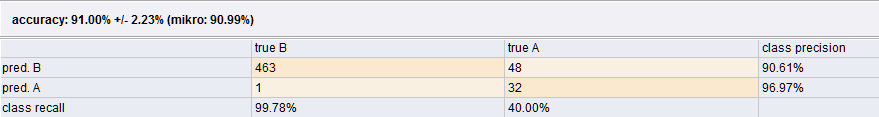
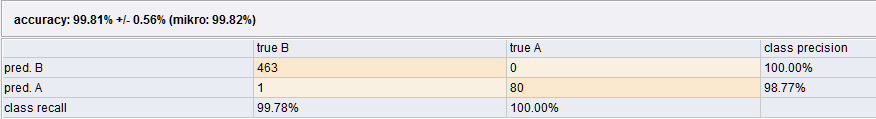
Indeed, using a stochastic classifier the recall index increase drastically, using the same set of attributes and the same dataset.

Figure 21 - Results for naive Bayes (NB) classifier.

Changing the approach to classification, the accuracy of predicted classes increase. Using the Bayes classifier, the recall value for A class increase, even though, the class precision for A class decrease for a bad classification of a single value.

## Decision Tree

Finally, using a decision tree classifier one manages to reach the best classification results.

Figure 22 - Results for decision tree classifier.

With the accuracy value of 99.81%, this classification is the best one could have with the provided dataset. All the positive case of DoS attack were predicted, only one case over 463 of negative cases was predicted as positive. This is an acceptable condition for have a recall value of 100%.

With the results below, it was proved that it’s possible to achieve the attack prediction for DoS attacks, using the small set of provided SLO, with a good amount of accuracy.

## Attributes subset

Since the secondary objective of the simulation was find the importance of each feature for the DoS attack prediction, the classification was launched again, using the decision tree classifier, with different subsets of attributes.

In order to predict a DoS attack, one finds that the most relevant attributes from the provided set was the average response time. This feature, gives the major contribution to items classification.

Following the mapping between SLOs and attacks, it was also proved that the feature of **Max number of simultaneous cloud service users** (referring to the table 5) does not influence the prediction of DoS attack.

# Conclusion

Investigating about the actual cloud providers and their policy about the information disclosure, we tried to find a common point from where it was possible to start the realization of a scalable system that it could implement the concept of forensic readiness. The main purpose of our system was give as many information as possible about a possible attack, for possible future forensic investigation.

As found in the actual state of art, all the forensic analysis processes start from the log analysis. A forensic investigator can start his analysis querying our violation and attack databases to find the exact moment in which the system detected a violation or an attack. This precious information could save a lot of time in the investigation process.

CFRT tries to recognize a possible attack using information disclosed to customers: service logs and SLA. Log files give information regarding the status of both services and physical machines. These files contain many information, but they are hard to read and understand because important information are scattered across the entire document. Thanks to our matching module we are able to query information directly from logs file summarize important data as SLO value and save all the suspected entries to be used further for some investigation. All what we need to know is the structure of logs to write queries and expand the provided dictionary.

The main project idea was not only retrieve information from logs, but also use SLOs violation in order to recognize possible attacks, matching real values from logs with theoretical value declared in the SLA to have a point of matching. The idea of use SLO violation was already present in the literature, but it was never used in order to identify an attack.

With the simulation, one proved that there are some SLO value, which are fundamental to predict an attack. Evaluation of the importance of each SLO for identify a specific attack should be investigated deeply. Once the entire mapping between SLO and attack will be proved, it will be possible for CFRT recognize a probable attack using classification. With this classifier it will be possible to guess a possible attack after only one or few SLO violation.

An important step ahead in for this research will be the standardization of SLA. Improve SLA document with additional and computable SLO, will give more chance to forecast an attack and achieve a better result in cloud forensic readiness.

In future, we will try to implement all the proposed mappings in table 5 and prove the importance of each SLO during the attack prediction process.

A better dataset for DoS and DDoS attack can be created, using following extra metrics:

* Success rate: the ratio between HTTP response “200” and the number of total requests.
* AVG (object size) : average of size (in byte) of requested objects.
* Error rate: the ratio between HTTP response “500” and the number of total requests.
* PUT rate: the ratio between the number of PUT requests and the total number of requests.
* GET rate: the ratio between the number of GET requests and the total number of requests.

Violation metrics, is actually a Boolean value, in the future this value can be splitted into differents attributes, one for each SLO:

* Uptime violation: Boolean value that indicates a violation of the uptime SLO.
* Simoultaneous clients violation: Boolean value that indicates a violation of the number of maximum connected clients.
* Average response time violation: Boolean value that indicates a violation of the number of average response time.

References:

1. – CSA, “Top Threats of Cloud Computing V1.0”, march 2010.
2. – Bernd Grobauer, Tobias Walloschek, and Elmar Stöcker (Siemens), “Understanding Cloud Computing Vulnerabilities”, march/april 2011.
3. - Luis M. Vaquero et al., “Locking the sky: a survey on IaaS cloud security”.
4. - Amazon S3, “Access Log documentation”, <http://docs.aws.amazon.com/AmazonS3/latest/dev/LogFormat.html>
5. – Amazon , “Key Differences Between the Amazon Website and the REST API Endpoint”, <http://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteEndpoints.html>
6. – Apache Tomcat, “Tomcat documentation page”, <https://tomcat.apache.org/tomcat-5.5-doc/config/valve.html>
7. – Amazon, “S3 Server access log”, <http://docs.aws.amazon.com/AmazonS3/latest/dev/ServerLogs.html>
8. – Gennaro Napoli, “Sviluppo e Sperimentazione di un Tool Per L’estrazione Automatica di Vincoli Contrattuali da Service Level Agreement”, Università Degli Studi di Salerno, 2015.
9. – Salman A. Baset, ”Cloud SLAs: Present and Future”, IBM Research.
10. – P. Mell and T. Grance, “The NIST Definition of Cloud Computing”, NIST Special Publication 800-145, September 2011.
11. – Robert Rowlingson , “A Ten Step Process for Forensic Readiness” , Journal of Digital Evidence Winter 2004.
12. – John Tan, “Forensic Readiness”, July 17, 2001.
13. – Zafarullah, Faiza Anwar, Zahid Anwar, “Digital Forensics For Eucalyptus”, 2011 Frontiers of Information Technology.
14. – Alecsandru Patrascu1 and Victor Valeriu Patriciu, “Logging Framework for Cloud Computing Forensic Environments”, Military Technical Academy, Computer Science Department, Bucharest, Romania.
15. – Raffael Marty, “Cloud Application Logging for Forensics”, San Francisco.
16. – F.R. Van Staden and H.S. Venter, “Implementing Digital Forensic Readiness for Cloud Computing Using Performance Monitoring Tools”, University of Pretoria.
17. – Vincent C. Emeakaroha et al., “CASViD: Application Level Monitoring for SLA Violation Detection in Clouds”, 2012 IEEE 36th International Conference on Computer Software and Applications.
18. – Jesse Peterson, “LittleS3” - Github - <https://github.com/igorhvr/littles3>
19. – Euromean Commission, “Cloud Service Level Agreement Standardisation Guidelines”, http://ec.europa.eu/digital-agenda/en/news/cloud-service-level-agreement-standardisation-guidelines
20. – HoneyNet, [HoneyNet project forensic challenge], <http://www.honeynet.org/challenge/results/index.html>
21. – The Notorious Nine, “Cloud Computing Top Threats in 2013”, Cloud Security Alliance. https://downloads.cloudsecurityalliance.org/initiatives/top\_threats/The\_Notorious\_Nine\_Cloud\_Computing\_Top\_Threats\_in\_2013.pdf
22. – K. Modi School, “Detection and Prevention of DDoS Attacks on the Cloud using Double-TCP Mechanism and HMM-based Architecture”, Computer Science and Engineering, VIT University, Chennai – International Journal of Cloud Computing and Services Science - April 2014
23. – Rapid Miner - <https://rapidminer.com/>
24. – J. J. Barbara, “Cloud computing: Another digital forensic challenge”, Digital Forensic Investigator News, October 2009.
25. – D. Birk and C. Wegener, “Technical issues of forensic investigations in cloud computing environments”, Ruhr-University Bochum, Horst Goertz Institute for IT Security Bochum, Germany, January 2011.
26. – G. Palmer, “A road map for digital forensic research”, First Digital Forensic Research Workshop (DFRWS), Tech. Rep. DTR - T001-01 FINAL, August 2001.