Chapter 12
Forensics as a Service

Dener Didoné
Universidade Federal de Pernambuco (UFPE), Brazil

Ruy J. G. B. de Queiroz
Universidade Federal de Pernambuco (UFPE), Brazil

ABSTRACT

Cloud computing as a paradigm shift is transforming how services are being delivered. In this chapter, the authors present a Forensics as a Service (FaaS) model using cloud computing to deliver forensic services. This model leverages the flexibility, elasticity, and dynamics of cloud computing, and is affordable for business, government, or individuals in need, due to its reduced cost. It also addresses the challenge of processing a large volume of forensic data by using MapReduce and distributed computing.

INTRODUCTION

According to the report by Anderson and Rainie (2010), 71% of interviewed people agreed with the statement:

By 2020, most people won't do their work with software running on a general-purpose PC. Instead, they will work in Internet-based applications such as Google Docs, and in applications run from Smartphones. Aspiring application developers will develop for Smartphone vendors and companies that provide Internet-based applications, because most innovative work will be done in that domain, instead of designing applications that run on a PC operating system.

DOI: 10.4018/978-1-4666-2662-1.ch012

This report shows the growing trend of using elastic computing power offered by the Cloud provided as a commodity.

Garfinkel (2010) states that "the growing size of storage devices means that there is frequently insufficient time to create a forensic image of a subject device, or to process all of the data once it is found," which he has listed as one of the key challenges for computer forensics today. This chapter aims to propose a *Forensic as a Service*

(FaaS) model to address this challenge using the cloud computing infrastructure. Our model of FaaS shows how to use the MapReduce computation model for solving everyday forensics tasks, delivered through the Cloud. Our settings allow the use of public or private clouds to perform these tasks.

In this chapter, we will first briefly discuss *MapReduce* programming model and Forensics as a Service. We will then present our FaaS model with evaluation and validation, followed by discussions on the security, legal, economic and governmental implications of this proposed model.

BACKGROUND

MapReduce

MapReduce is a distributed programming paradigm, developed by *Google* to simplify the development of scalable, massive parallel applications that process terabytes of data on large commodity clusters.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system. (Dean & Ghemawat, 2004)

Users specify a map function that processes a *key/value* pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, including forensics tasks (string operations, image processing, statistical analysis, etc.) (Roussev, Wang, Richard III, & Marziale, 2009).

The canonical example application of MapReduce is a process to count the appearances of each different word in a set of documents (see Algorithm 1).

In this example retrieved from Dean and Ghemawat (2004), each document is split into words, and each word is counted initially with a "1" value by the Map function, using the word as the result key. The framework puts together all the pairs with the same key and feeds them to the same call to *Reduce*, thus this function just needs

Algorithm 1. Canonical example application of MapReduce

```
void map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    EmitIntermediate(w, "1");
void reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  int result = 0;
  for each pc in partialCounts:
    result += ParseInt(pc);
  Emit(AsString(result));
```