Context-Aware Cloud Topology
Optimisation and Simulation

Parallel Trace Analysis

Project Deliverable D4.3

Zafeirios Papazachos, Sakil Barbhuiya, Dimitrios Nikolopoulos **(QUB)**,

Ahmed Ali-Eldin, Olumuyiwa Ibidunmoye, Amardeep Mehta, Ali Rezai (UMU),

Athanasios Tsitsipas (UULM),

Gabriel Gonzalez Castañé (DCU)

Due date: 30/06/2015
Delivery date: 30/06/2015

This project is funded by
the European Union under
grant agreement no. 610711

Dissemination Level

X	PU	Public
	PP	Restricted to other programme participants (including the Commission Services)
	RE	Restricted to a group specified by the consortium (including the Commission Services)
	CO	Confidential, only for members of the consortium (including the Commission Services)

Version History

Version	Date	Change	Author
0.1	04.05.2015	Structure of document defined	Papazachos Zafeirios(QUB)
0.2	26.05.2015	Added input on Spike detection	Ahmed Ali-Eldin (UMU)
0.3	01.06.2015	Input on Trace analysis Framework	Athanasios Tsitsipas(UULM)
0.4	02.06.2015	Initial input on draft	Papazachos Zafeirios(QUB)
0.5	03.06.2015	Input merging	Papazachos Zafeirios(QUB)
0.6	04.06.2015	Revised and commented structure	Dimitrios Nikolopoulos(QUB)
0.7	05.06.2015	Restructured part of document	Zafeirios Papazachos(QUB)
0.8	15.06.2015	Finalised draft for first review	Zafeirios Papazachos(QUB)
0.9	18.06.2015	1st internal review	Gabriel Gonzalez Castañé (DCU)
0.91	22.06.2015	Addressed internal review pointed issues	Papazachos Zafeirios(QUB)
0.92	24.06.2015	2nd internal review	Jörg Domaschka (UULM)
0.93	26.06.2015	Addressed the internal review pointed issues	Athanasios Tsitsipas(UULM)
0.94	26.06.2015	Fixes and changes based on 2nd review	Papazachos Zafeirios(QUB)
0.95	29.06.2015	Addressed pointed issues and added references	Ahmed Ali-Eldin(UMU)

EXECUTIVE SUMMARY

CactoScale provides monitoring and data analysis functionality to CACTOS. This deliverable presents the framework and algorithms used by CactoScale for parallel trace analysis. We describe different CactoScale framework extensions which enable the implementation of parallel correlation analysis of system utilisation metric traces and cloud data logs. We also present the implementation of Lambda Architecture into CactoScale which parallelises several aspects of monitoring and exchanging information in CACTOS.

CactoScale trace analysis tackles parallelism on various dimensions. We describe a hierarchical log analysis and anomaly detection framework. The anomaly detection utilises parallel data analysis frameworks such as Spark and mapreduce framework for parallel analysis of workload traces and system logs, coupled with HDFS for in-memory processing of the data. The trace analysis also involves the pre-processing of raw data logs for storage in HDFS. It allows executing anomaly detection algorithms hierarchically, both utilising the compute nodes in situ and the parallel HDFS monitoring facility. This is feasible by pairing the CactoScale agents with in situ analytics modules to cover the cases such as workload spike detection, but also to filter the data that flows to the database for post-processing. An in situ analytic module is a process designed to run locally in a node. This tactic provides the advantage of data locality. The data are pre-processed by the local node before being collected by a remote distributed service for further processing. In this way, the hierarchical design of data analysis allows for an additional level of real-time processing which is much closer to the data source.

CactoScale has different features and capabilities for parallel trace analysis which are demonstrated in this deliverable by using different algorithms for anomaly detection. Anomaly detection involves the use of trace analysis algorithms that detects outliers (numerical, textual, or correlation based) in data traces. Detecting outliers can trigger actions in resource management and for this reason we focus in anomaly detection as a use case. We demonstrate a Lightweight Anomaly Detection Tool based on correlation analysis. This tool utilises a monitoring cluster to perform parallel trace analysis using Spark and mapreduce. The online data analysis modules that we demonstrate include a log analysis module and several spike detection methods. Workload spikes are one of the main causes of QoS degradation in cloud applications. The log analysis demonstrates how information on cloud platform can contribute in reducing any false positive alerts.

TABLE OF CONTENTS

EXECUTIVE SUMMARY I

TABLE OF CONTENTS II

LIST OF FIGURES III

LIST OF TABLES IV

ABBREVIATIONS V

I. INTRODUCTION 1

II. PARALLEL TRACE ANALYSIS IN CACTOSCALE 3

1. OVERVIEW 3

2. CACTOSCALE MONITORING AND INTEGRATION 4

A) ARCHITECTURE 5

B) SCALING 6

C) PERFORMANCE 7

3. CACTOSCALE DATA ANALYSIS FRAMEWORK 8

III. TRACE ANALYSIS ALGORITHMS 10

1. LIGHTWEIGHT ANOMALY DETECTION TOOL 10

A) ANOMALY DETECTION TOOLS 10

B) LIGHTWEIGHT ANOMALY DETECTION TOOL ALGORITHM AND FUNCTIONALITY 11

C) EVALUATION 12

2. TEXTUAL LOG ANALYSIS 16

A) ANOMALY DETECTION TOOLS BASED ON CONSOLE LOG ANALYSIS 16

B) ALGORITHM DESCRIPTION 17

C) EVALUATION 18

3. PERFORMANCE AND MODEL ANOMALIES 20

A) ANOMALY DETECTION: STATE-OF-THE-ART 20

B) ONLINE SPIKE DETECTION IN CACTOSCALE 22

C) EVALUATION 27

IV. SUMMARY AND CONCLUSION 29

REFERENCES 30

LIST OF FIGURES

FIGURE 1 OVERVIEW OF THE CACTOS TOOLKIT 3
FIGURE 2: THE LAMBDA ARCHITECTURE APPLIED IN RUNTIME MODEL UPDATER 5
FIGURE 3 : PARALLEL INSTANCES OF RUNTIME MODEL UPDATER 6
FIGURE 4: RUNTIME MODEL UPDATER EXECUTION TIMES (HBASE ACCESS AND GENERATION OF CACTOS CLOUD INFRASTRUCTURE MODEL INSTANCES) 7
FIGURE 5 HIERARCHICAL DATA ANALYSIS AND DATA FILTERING WORKFLOW 8
FIGURE 6 TIME-SERIES OF NODE CPU UTILISATION(BLUE) AND AGGREGATED VM CPU UTILISATION (RED) 13
FIGURE 7 TIME-SERIES OF NODE DISK IOPS (BLUE) AND AGGREGATED VM DISK IOPS (RED) 13
FIGURE 8 CORRELATION COEFFICIENTS BETWEEN THE TWO CPU TIME-SERIES MEASUREMENTS. 14
FIGURE 9 CORRELATION COEFFICIENTS BETWEEN THE TWO DISK IOPS TIME-SERIES MEASUREMENTS 14
FIGURE 10 TIME SPENT ON THE DATA PROCESSING PHASE. 14
FIGURE 11 TIME SPENT FOR THE WHOLE ANALYSIS PROCESS. 14
FIGURE 12 SPEEDUP OF PIG AND SPARK IMPLEMENTATION OF LADT 15
FIGURE 13 NUMBER OF ROW-KEYS PROCESSED PER SECOND DURING THE DATA PROCESSING PHASE. 15
FIGURE 14 NUMBER OF ROW-KEYS PROCESSED PER SECOND FOR THE OVERALL TIME. 15
FIGURE 15 SPARK SPEEDUP OVER PIG 16
FIGURE 16 PROCESSING OF LOG DATA 17
FIGURE 17 TIME-SERIES OF NODE CPU UTILISATION(BLUE) AND AGGREGATED VM CPU UTILISATION (RED) 18
FIGURE 18 TIME-SERIES OF NODE DISK IOPS (BLUE) AND AGGREGATED VM DISK IOPS (RED) 18
FIGURE 19 CORRELATION COEFFICIENTS BETWEEN THE TWO CPU TIME-SERIES MEASUREMENTS. 18
FIGURE 20 CORRELATION COEFFICIENTS BETWEEN THE TWO DISK IOPS TIME-SERIES MEASUREMENTS. 18
FIGURE 21 VM ACTIVITY RECORD DERIVED FROM IN-SITU LOG ANALYSIS 19
FIGURE 22 TIME SPENT ON THE DATA PROCESSING PHASE. 19
FIGURE 23 TIME SPENT FOR THE WHOLE ANALYSIS PROCESS. 19
FIGURE 24 NUMBER OF ROW-KEYS PROCESSED PER SECOND DURING THE DATA PROCESSING PHASE. 20
FIGURE 25 NUMBER OF ROW-KEYS PROCESSED PER SECOND FOR THE OVERALL TIME. 20
FIGURE 26 FIFA 1998 WORLDCUP SERVERS' WORKLOAD 23
FIGURE 27 THE EFFECT OF AGGREGATING WORKLOAD MEASUREMENTS ON A MORE COARSELY GRANULATED TIMESCALE. LEFT: THE ORIGINAL MEASUREMENT SEQUENCE AND RESIDUALS WHEN CALCULATING ONE-STEP AHEAD PREDICTION USING AN AR MODEL. RIGHT: SAME AS LEFT, BUT FOR DATA AGGREGATED ON 25

LIST OF TABLES

TABLE 1 TUNABLE PARAMETERS IN LADT ALGORITHM 12
TABLE 2 RESULTS FOR THE EVALUATION OF THE ALGORITHMS FOR SPIKE DETECTION 28

ABBREVIATIONS

Abbreviation	Description
CACTOS	Context-Aware Cloud Topology Optimisation and Simulation
VM	Virtual Machine
HDFS	Hadoop Distributed File System
I/O	Input / Output
IOPS	I/O Operations Per Second
SLA	Service Level Agreement
QoS	Quality of Service

I. INTRODUCTION

The increasing size and complexity of modern data centres has brought upon the demand for new tools that will assist cloud operators to overcome the difficulties associated with the management of physical resources and hosted applications. The vast amount of heterogeneous resources in modern cloud platforms implies an equally diverse and large in volume amount of information that is related with different sources of data and log files. In order to make use of these data, the cloud operator must possess scalable tools which will allow handling and processing massive volumes of data in real time.

One of the goals of CACTOS project is to deliver a framework for the automated management and analysis of the cloud data centre resources. This allows cloud data centre operators to utilise their resources efficiently in order to maintain a high level of quality of service. To achieve this CACTOS toolkit (Ostberg, et al., 2014) combines the functionality of three different components which; monitor and analyse different data traces; optimise and provision resources to virtual machines; and simulate different scenarios. These tools are CactoScale, CactoOpt and CactoSim respectively.

CactoScale is a component of CACTOS toolkit which provides the cloud operator with the necessary tools to collect and analyse data traces from different sources. The framework of CactoScale allows for scalable and parallel data analysis. It is based on utilising parallel data processing frameworks such as Hadoop mapreduce (Dean & Ghemawat, 2008) and Spark (Spark, 2015). In this deliverable we demonstrate parallel filtering, correlation and analysis of traces. Parallelism is tackled on various dimensions. This deliverable demonstrates the use of these frameworks using an anomaly detection algorithm based on correlation analysis. It also demonstrates distributed processing of data traces based on in situ trace analysis modules for the purpose of online spike detection and textual log filtering. The processing of traces in the latter case occurs in a level which complements the monitoring cluster and works in parallel with it. To summarise, the following are some of the major features of CactoScale analysis tools:

- CactoScale tackles parallelism on various dimensions. It allows for parallel trace analysis based on parallel data processing tools such as Hadoop mapreduce and Spark. It also allows for online data analysis which is distributed on each node and is done in parallel to the monitoring infrastructure.
- We introduce Lambda Architecture (λ lambda-architecture, 2015) concept for the generation of the Infrastructure Model Instances (D5.4 Evaluation Methodology for the CACTOS Runtime and Prediction Toolkits) by the Runtime Model Updater of the monitoring system. This enhances with a parallel flavour the capabilities of the Runtime Model Updater in CactoScale.
- We present a lightweight anomaly detection tool (LADT) (Barbhuiya, Papazachos, Kilpatric, & Nikolopoulos, 2015) which implements parallel correlation analysis using Spark and Hadoop mapreduce framework.
- We implement a filtering mechanism of textual data log information for storing into a distributed monitoring database.
- An hierarchical analysis framework is demonstrated which allows in-situ pre-processing and filtering of information before storing them in a database. In this way, part of the workload is off loaded locally to the cloud nodes.
- Workload spikes are one of the main causes of QoS degradation in cloud applications. We have designed a number of new methods for early spike detection in a workload. These online analysis methods allow data to be processed locally on each node. This tactic provides the advantage of data locality and is done in a distributed way on each corresponding node in the cloud.

In this deliverable we present different algorithms for anomaly detection to demonstrate different features and capabilities of CactoScale for parallel trace analysis. We present a Lightweight Anomaly Detection Tool (LADT) which utilises a correlation analysis algorithm to detect anomalies on the host node. LADT bases its functionality on the correlation analysis between VM-level and node level metrics. Furthermore, we present an extension of the LADT tool which utilises filtered log information from cloud nodes to limit the number of false positive alerts that are triggered by LADT analysis when the performance of the VMs is affected by normal cloud activity, such as the allocation of a new VM instance on the same node. Finally, we present the case of in-situ data trace analysis where the data are processed locally on each node to detect any workload spikes.

The remainder of the document is structured in four parts. First, Section II describes the CactoScale framework for scalable trace analysis. In Section III we present different anomaly detection algorithms that are used as different scenarios for performing parallel trace analysis. In the final Section IV we provide concluding remarks and plans for the future.

II. PARALLEL TRACE ANALYSIS IN CACTOSCALE

CACTOS runtime toolkit (Ostberg, et al., 2014) is a holistic approach for the monitoring, analysis, optimisation and predictive analysis of IaaS cloud data centres. The main goal is to provide autonomic and interactive tools to support and improve the efficiency of cloud data centre operations. To accomplish this goal CACTOS toolkit combines three components CactoOpt, CactoSim and CactoScale to build the foundation to craft, evaluate, and improve data centre design, operation, and placement optimization algorithms. CactoOpt is designed to facilitate development of advanced optimization mechanisms capable of both resource-level scheduling optimization as well as holistic data centre-level optimization. CactoScale provides data filtering and correlation analysis tools which will run on vast volumes of data generated from large data sets from both physical nodes and virtual machines. CactoSim enables the evaluation of optimization strategies at design time based on various load and data centre models. In the following subsections, we describe the role of CactoScale in the CACTOS toolkit.

1. OVERVIEW

The role of CACTOS toolkit (Ostberg, et al., 2014) is to tackle the complexity of orchestrating modern cloud data centres and enable cloud operators to efficiently monitor, analyse and optimise the cloud platform. Figure 1 illustrates how CACTOS toolkit combines CactoScale, CactoOpt and CactoSim components for capturing and analysing data centre information, optimising the topology and performing simulations.

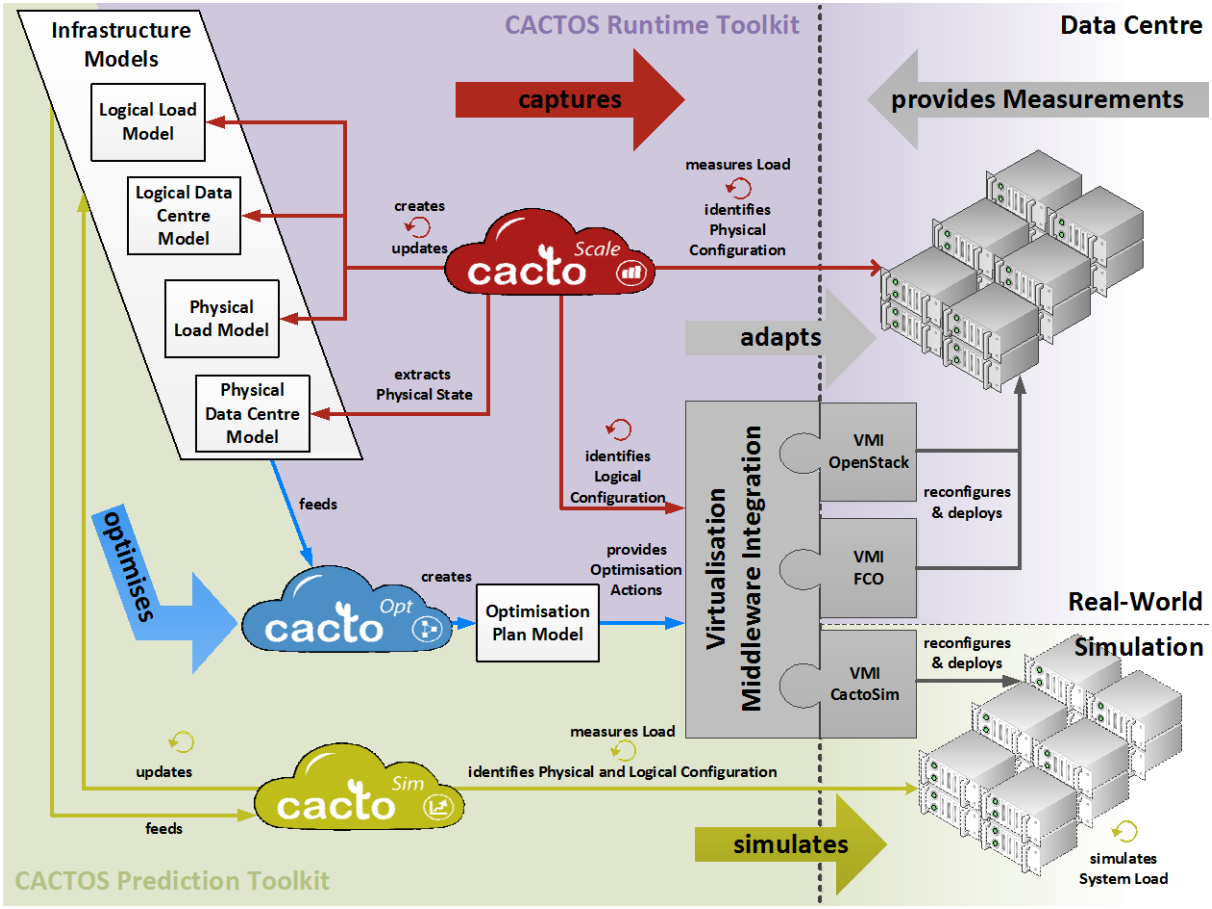

Figure 1 Overview of the CACTOS toolkit

In this deliverable we will focus on the CactoScale component. CactoScale is designed to monitor cloud data centres efficiently. It is a set of tools and methods to acquire and analyse application behaviour and infrastructure performance data. CactoScale utilises an agent-based monitoring architecture to retrieve load and infrastructure information from the data centre at runtime. A monitoring agent is responsible for extracting performance and utilisation samples from the nodes at regular time intervals. The data extracted by the agents are then gathered by collector processes which are designed to provide high availability and data transfer reliability. The agents and collectors are provided by Apache Chukwa's (Rabkin & Katz, 2010) runtime monitoring. A Chukwa agent runs on every monitored node. It collects data from physical devices (cpu, memory, storage, network) and VM usage information from different monitoring utilities such as iostat, sar, Virt-Top, Sigar, top, ps and Df. The role of these monitoring tools is further described in D4.1 (D4.1 Data Collection Framework). The output is then collected by one of the available collectors. The collector processes the data and registers the input to HBase.

CactoScale is also capable of gathering and extracting information from application and error logs. The agents can be paired with in situ analytics modules to cover the cases where high sampling rates of numerical indicators (e.g. utilisation) are needed, but also to filter the data that flows to the database for post-processing. An in situ analytic module is a process designed to run locally in a node. This tactic provides the advantage of data locality. The data are pre-processed by the local node before being collected by a remote distributed service for further processing. In this way, the hierarchical design of data analysis allows for an additional level of real-time processing which is much closer to the data source. To avoid interference of the trace data processing infrastructure, we plan to explore scheduling and buffer allocation methods that isolate the resources (cores, memory, interconnect) used by the logging infrastructure from the resources used by actual workloads.

The following sections consist of two parts. The first part describes the role of CactoScale with regard to data collection and integration with CACTOS. It describes an extension on the original CactoScale monitoring framework (D5.1 Model Integration Method and Supporting Tooling) for enhanced integration and performance with the CACTOS toolkit. The second part of this section, describes the additional modules of CactoScale which allow in-situ and also parallel trace analysis using a monitoring cluster based on Spark (Spark, 2015) and Hadoop (Hadoop, 2015).

2. CACTOSCALE MONITORING AND INTEGRATION

CactoScale is able to exchange information with CactoOpt and CactoSim by creating infrastructure model instances of the cloud data centre. A detailed description of the Infrastructure Model Instances is given in (D5.4 Evaluation Methodology for the CACTOS Runtime and Prediction Toolkits). The Runtime Model Updater component of CactoScale is responsible for persisting the current system state in CACTOS Cloud Infrastructure Model instances. For this purpose the Runtime Model Updater periodically queries the database for current data centre information. Once it has collected this information, it updates the Logical Data Centre Model with changes in the virtual topology. Additionally, both Physical and Logical Load Models are updated with recent utilisation metrics. Changes to the models are persisted in the Runtime Model Storage. This section aims to present the parallelism flavour of Runtime Model Updater in CactoScale. Diving in subsection level, in Architecture the Runtime Model Updater topology is described and presented, Scaling includes information about the scalability of Runtime Model Updater functionality and finally Performance depicts the results of executing Runtime Model Updater using an improved version of Chukwa developed by CACTOS. However, background information is available in (D4.1 Data Collection Framework) and (D4.2 Preliminary offline trace analysis).

a) ARCHITECTURE

CactoScale, utilises Chukwa (D4.1 Data Collection Framework) to serve as the log collection and analysis system for monitoring large distributed systems. The data collection scheme works as follows: Data logs are generated in each node. Several adaptors capture the data logs from different types of sources and send them to the agent. The agent connects to one of the available collectors and sends the data. The collectors gather the logs from the agents and store them in HDFS. The Runtime Model Updater with respect to the stored data in HBase builds up the CACTOS Cloud Infrastructure Models. The most intriguing aspect, is the Lambda Architecture (λ lambda-architecture, 2015) , a concept that is applied in the core code and combines the large-scale batch-processing in the batch layer with the real-time responsiveness of stream processing in the speed layer. The current architecture in Figure 2 , aims to persist rapidly an accurate view of a Cloud environment.

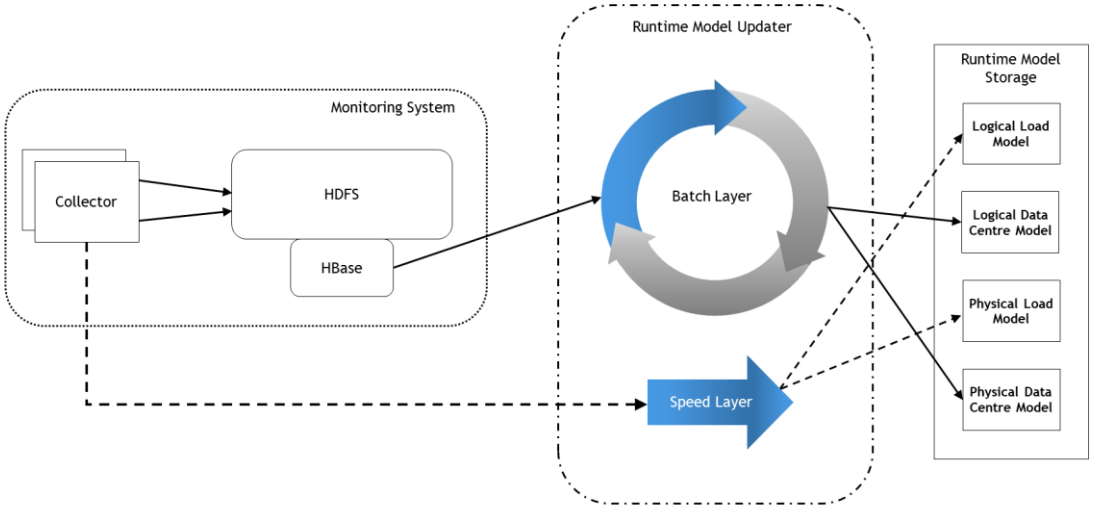

Figure 2: The Lambda architecture applied in Runtime Model Updater

BATCH LAYER

The batch layer in Runtime Model Updater runs in configurable intervals, creating and updating the models in Runtime Model Storage from raw data in HBase; for instance the UULM testbed used an interval of 60 seconds. Each time a batch job finishes, the Cloud Infrastructure Models reflect the real world Cloud environment. Because it only ever operates on immutable raw data, the batch layer can easily exploit parallelism which will be explained in the subdivision Speed layer. The batch layer drives all the four different kinds of Cloud Infrastructure Models in the Runtime Model Storage.

The major drawback of a batch layer is latency. Assuming the batch layer takes five minutes to complete a single run, the models will always be at least five minutes out-of-date. This is where speed layer comes into play.

SPEED LAYER

The expense of latency in batch layer, emerge the need for the speed layer in Runtime Model Updater. Optimally, the Physical and Logical Load Models are updated at real time in order to provide the latest and most recent values to CactoOpt. In particular, they require the new raw load data to be available faster than the update time provisioned by the batch mode. As new data arrives, the speed layer updates both the Load Models which is combined with the batch layer to create fully up-to-date models. For CACTOS, we implemented an extension to Chukwa that enables this feature. This implementation is based on a mechanism of the Chukwa Collector that allows tracing the stream of chunks passing through the system.

b) SCALING

A real world Cloud environment, consists of hundreds of thousands physical machines and even millions of virtual machines. Hence, the Runtime Model Updater must maintain a snapshot of such an infrastructure. Currently (D5.3 Operational Small-Scale Cloud Testbed Managed by the CACTOS Runtime Toolkit), a single instance of the Runtime Model Updater is responsible for this operation and apparently cannot handle such a large infrastructure. Due to its execution delays to handle and process the vast information and the immutable data in HBase, there is an emerged need to scale out the concept of its behaviour.

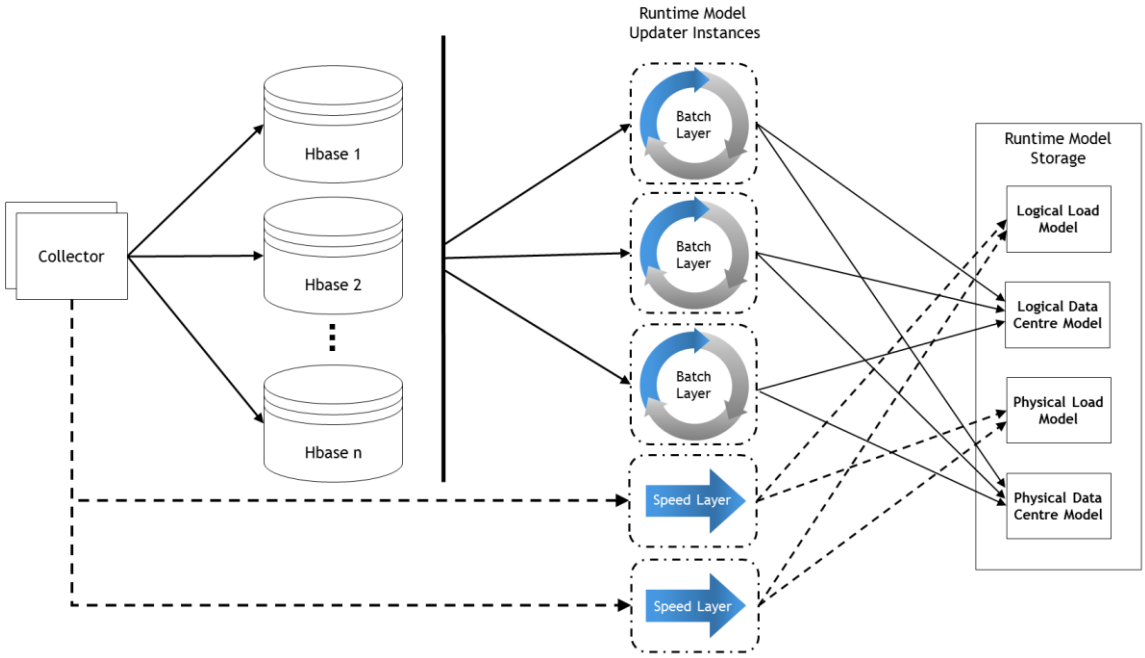

Figure 3 : Parallel instances of Runtime Model Updater

As mentioned in II.2.a), the Runtime Model Updater applies the Lambda architecture for its core functionality. In addition to that, we implemented a partitioning mechanism that lets one Runtime Model Updater only operate on a configurable subset of physical hosts and virtual machines. Hence, as depicted in Figure 3, the Runtime Model Updater is used either in batch or speed layer. Each instance is responsible for a set of physical machines; a configuration issue, though, depending on the size of the data centre and the HBase cluster. Specific guidelines will be developed in a future deliverable (D5.5 Performance evaluation of the CACTOS toolkit on a small cloud testbed) depicting the best configuration of the system depending on the size of the physical and virtual set-up and the demanded set of monitoring information.

The parallelism flavour is enabled by exploiting the scalability capabilities of the other components of CactoScale. Due to the fact that the speed layer relies on Chukwa Collectors features, the quantity of available Chukwa Collectors determines the maximum possible quantity of speed layer instances of Runtime Model Updater. Then, each speed layer is subscribed to a specific collector bound to a set of physical machines and the virtual machines hosted by that physical machine. On the other side, the amount of batch layer instances of Runtime Model Generator will be dependent on the number of HBase nodes, where each one contains information for different set of physical machines. In CACTOS project, Chukwa is refactored to offer more performant writes to the HBase, which is explained in the following section Performance. Finally, the involvement of multiple instances will result in a sustainable solution for snapshotting a large scale infrastructure.

c) PERFORMANCE

Chukwa (D4.1 Data Collection Framework) is structured with distinct phases including the collection and processing of data that aims to provide clean and narrow interfaces between these phases. Within the monitoring system, each collector receives data from up to several hundred hosts, writing data directly to HBase. The current distribution of Chukwa provides index by primary key, managing data compaction. Also, Chukwa has a toolbox of MapReduce jobs for organizing and processing incoming data, based on the source of the data (i.e Sigar, virt-top, df). But, all the data even though they are pre-processed, they are stored in one table and it is difficult to retrieve a particular result fast enough, because the entire table has to be scanned. The HBase API, provides some filtering techniques, such as specifying the time range of the preferred query, but still the query time is not adequate.

Chukwa, in the context of CACTOS has been improved[1], supporting the following features:

- Enabling the query actions to HBase to become more performant
- Store aggregated values
- Distribute evenly the information to HBase cluster

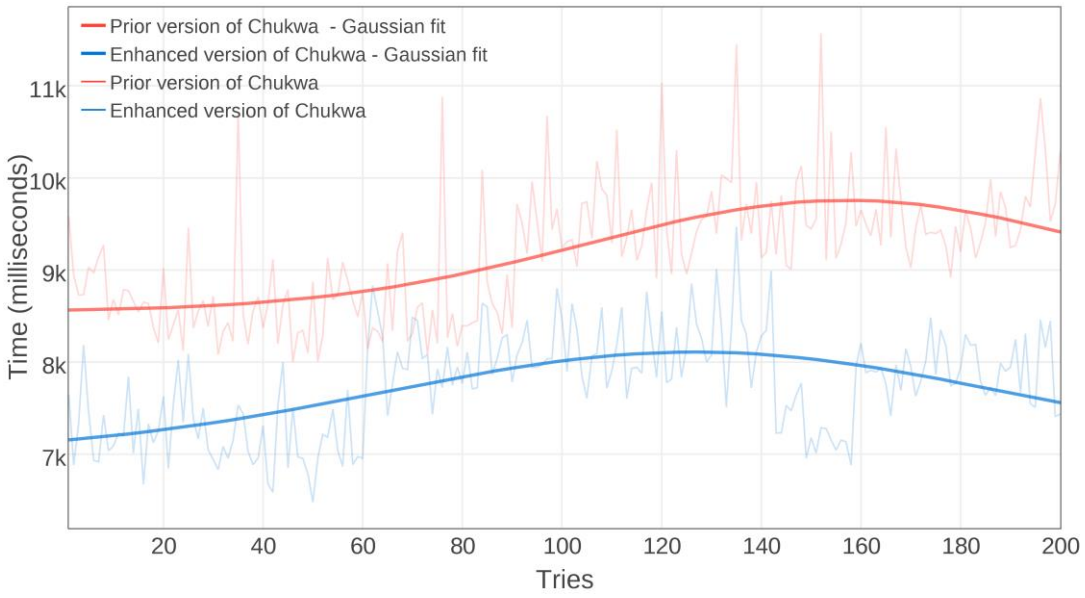

Figure 4: Runtime Model Updater execution times (HBase access and generation of CACTOS Cloud Infrastructure Model instances)

The graph in Figure 4 depicts the variation and the normal distribution (Gaussian fit) in execution times of Runtime Model Updater, using the prior and the enhanced version of Chukwa. In addition, the time measurements are extracted monitoring the HBase access and the generation of CACTOS Cloud Infrastructure Model instances; core functionality of Runtime Model Updater. The amount of total execution time is not significant at this point, but what is more important is the decrease of 25% in time at accessing the HBase and generating the models. The evaluation was done against the UULM testbed (D7.2.1 Physical Testbed), utilizing 16 physical hosts and 100 VMs running in the environment. In the future deliverable (D5.5 Performance evaluation of the CACTOS toolkit on a small cloud testbed) , the observed fluctuation in execution times of Runtime Model Updater will be examined.

[1] https://svn.fzi.de/svn/cactos/code/scale/trunk/chukwa-incubating-src-0.5.0/src/main/java/eu/cactosfp7/datacollection/chukwa/

The Chukwa refactor done by utilizing an index table, named *Index*, using as primary key the identifier of the physical machine and as value the "static" information (i.e CPU architecture) that cannot change during runtime, plus the last timestamp when the information was updated. The rest of the "dynamic" information is stored in a different table, named *Metrics*, which can be queried by using the last update timestamp of a physical machine for a specific metric. Evading a rigorous query to HBase, enables the Runtime Model Updater to become even faster than using the current distribution of Chukwa. Moreover, Chukwa is equipped with a feature of storing aggregated values from historical data from different metrics and a trace of patterns are utilized by other CACTOS components as described in the current deliverable.

3. CACTOSCALE DATA ANALYSIS FRAMEWORK

CactoScale framework has been extended with the necessary modules to allow for parallel trace analysis. This includes in-situ modules which run locally in each compute node and perform pre-processing and analysis of local data before storing them in the monitoring database. The advantage of this technique is that part of the analysis workload is off loaded to the compute nodes which provide an additional level of real-time processing much closer to the data source.

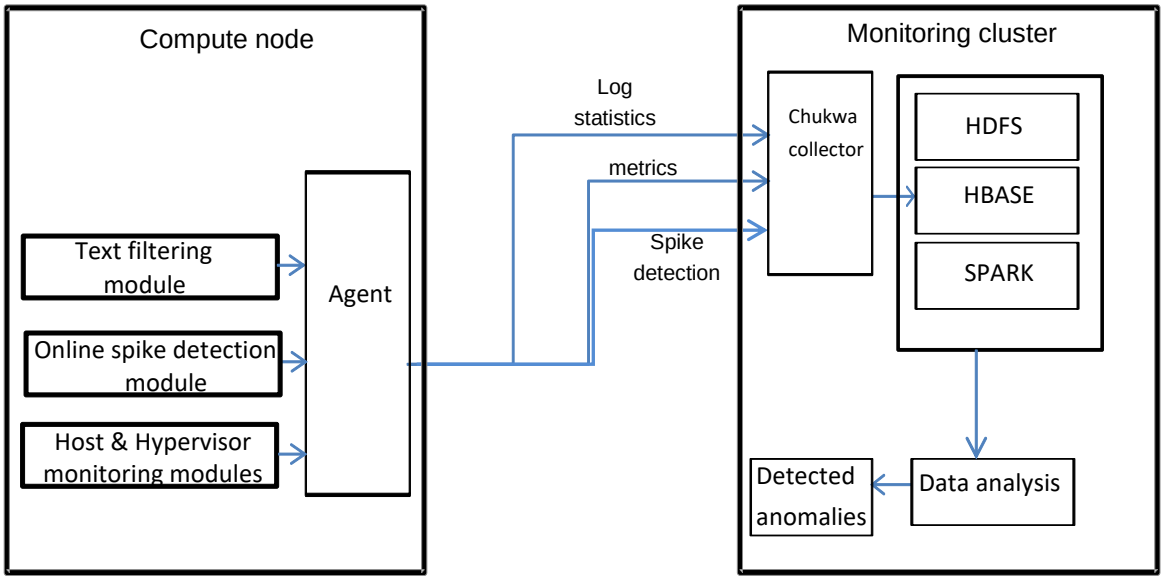

Figure 5 Hierarchical data analysis and data filtering workflow

In Figure 5 we illustrate the data analysis mechanisms of CactoScale that extend the framework described in (D4.1 Data Collection Framework). The extended framework includes in-situ modules which run locally on each compute node of the cloud. These application modules are connected to the agent along with the existing performance monitoring adaptors for the host node and the hypervisor.

The log filtering module enables CactoScale to detect VM operations on the node such as creation of a new VM or migration of a VM. It also allows detecting erroneous entries inside the log files. The filtering module is assessing the number of these occurrences over a selected time period and registers to the database a statistical number of them along with a timestamp. In this way, only a few entries are required to be recorded in the database, instead of a full log record. More information on this mechanism is given on Section III.2.

Online spike detection is a CactoScale in-situ data analysis module which allows CactoOpt to proactively deal with workload spikes as they develop. The interaction of CactoScale with CactoOpt and CactoSim is done using the mechanism described in section III.2. Since workload spikes are one of the main causes of QoS degradation in cloud applications, we have designed a number of new methods for early spike detection in a workload. The results from the in-situ spike detection are stored to the database of the monitoring facility, where they can be further utilised. Section III.3.b) describes in more detail the algorithms for the online spike detection.

To analyse the stored information in HBase, we use a Lightweight Anomaly Detection Tool (LADT) which is based on the apache Spark framework. The Spark framework has an advanced DAG execution engine that supports cyclic data flow and in-memory computing. It is a fast and general engine for large-scale data processing. Section III.1 describes the implementation of this tool and also compares the performance to an alternative implementation based on Hadoop using Pig scripts (Olston, Reed, Srivastava, Kumar, & Tomkins, 2008). The Apache Pig platform provides an abstraction over the MapReduce model. The motivation of transitioning from pig scripts to Spark is the better performance provided by Spark by enabling applications to reliably store the data in memory. The functionality and performance of LADT is detailed in Section III.1. A refinement of the correlation analysis by utilising cloud data logs is described in Section III.2. The use of the log analysis in the correlation analysis is giving us the advantage in discerning any fault anomaly alerts that might have been caused by normal activity of the cloud operator.

III. TRACE ANALYSIS ALGORITHMS

CactoScale has different features and capabilities for parallel trace analysis which are demonstrated in this deliverable by using different algorithms for anomaly detection. Anomaly detection involves the use of trace analysis algorithms that detects outliers (numerical, textual, or correlation based) in data traces. It is the cornerstone of all log analytics tools out there because these outliers trigger actions in resource management and for this reason we focus in anomaly detection as a use case. We present a Lightweight Anomaly Detection Tool (LADT) (Barbhuiya, Papazachos, Kilpatric, & Nikolopoulos, 2015) which utilises a correlation analysis algorithm to detect anomalies on the host node. LADT bases its functionality on the correlation analysis between VM-level and node level metrics. LADT is utilising the mapreduce framework to perform parallel trace analysis. We also measure the performance of an alternative implementation of LADT which uses Spark (Spark, 2015). The motivation for transitioning from Pig scripts to Spark is the better performance provided by Spark by enabling applications to reliably store this data in memory. The in-memory processing capability of Spark is achieved using the concept of a Resilient Distributed Dataset (RDD), which allows to transparently store data on memory and persist it to disc if it's needed. Furthermore, we present an extension of the LADT tool which utilises filtered log information from cloud nodes to limit the number of false positive alerts that are triggered by LADT analysis when the performance of the VMs is affected by normal cloud activity, such as the allocation of a new VM instance on the same node. Finally, we present the case of in-situ data trace analysis where the data are processed locally on each node to detect any workload spikes. Since workload spikes are one of the main causes of QoS degradation in cloud applications, we have designed a number of new methods for early spike detection in a workload.

1. LIGHTWEIGHT ANOMALY DETECTION TOOL

In this section we present CactoScale Ligthweight Anomaly Detection Tool (LADT) which monitors system-level and virtual machine (VM)-level metrics in Cloud data centres to detect node-level anomalies using performance metrics and correlation analysis. LADT addresses the complexity of implementing efficient monitoring and analysis tools in large-scale Cloud data centres, by collecting and storing the metrics generated by nodes and VMs using Apache Chukwa (Rabkin & Katz, 2010). To perform parallel analysis of the data traces we utilise two different implementations of the LADT algorithm which are based on the Hadoop mapreduce and Spark framework. The motivation for transitioning from the original Hadoop implementation to Spark is the better performance provided by Spark by enabling applications to reliably store this data in memory. The in-memory processing capability of Spark is achieved using the concept of a Resilient Distributed Dataset (RDD), which allows to transparently store data on memory and persist it to disc if it's needed.

a) ANOMALY DETECTION TOOLS

A number of anomaly detection tools use system metrics (Tan, Kavulya, Gandhi, & Narasimhan, 2012), (Wang, 2009), (Ward & Barker, 2013), (Kang, Chen, & Jiang, 2010), (Jiang, Munawar, Reidemeister, & Ward, 2009), (Bodik, Goldszmidt, Fox, Woodard, & Andersen, 2010), which can be collected with minimum overhead and without requiring any access to the source code of hosted applications. Using system metrics for detecting anomalies in Cloud systems has advantages over traditional log-based anomaly detection tools due to consideration of elasticity and workload evolution in cloud computing, due to provisioning, scaling, and termination of services in short periods of time. EbAT (Wang, 2009), is a tool that uses entropy based anomaly detection. EbAT analyses metric distributions and measures the dispersal or concentration of the distributions. The metrics are aggregated by entropy distributions across the cloud stack in order to form entropy time series. EbAT uses online tools like spike detection, signal processing and subspace methods to detect anomalies in the entropy time series. The tool incurs the complexity of analysing the metric distributions and also requires third party tools to detect anomalies.

PeerWatch (Kang, Chen, & Jiang, 2010) uses canonical correlation analysis (CCA) to extract the correlations between multiple application instances, where attributes of the instances are system resource metrics such as CPU utilisation, memory utilisation, network traffic etc. PeerWatch raises an alarm for anomaly whenever some correlations drop significantly. As a result of analysing the application instance behaviours and correlating them, this tool is capable of detecting application-level or VM level anomalies. However, this approach requires statistical metrics analysis and knowledge of the hosted applications, which is a limitation in large-scale clouds, where hundreds of different types of applications run on the VMs. Varanus (Ward & Barker, 2013) uses a gossip protocol, which is layered into clouds, groups and VMs in order to collect system metrics from the VMs and analyse them for anomalies. This approach allows in-situ collection and analysis of metrics data without requiring any dedicated monitoring servers to store the data. The metric-based black box detection technique presented in (Tan, Kavulya, Gandhi, & Narasimhan, 2012) uses the LFD (Light-Weight Failure Detection) algorithm to detect system anomalies. LFD raises an alarm when there is a lack of correlation between two specific system metrics. The anomaly indicates a system problem and each such problem is associated with a specific system metrics pair. LFD is a lightweight algorithm with lower complexity than EbAT, PeerWatch and Varanus. Furthermore, LFD does not require any training or source code and understanding of hosted applications. The LFD follows a decentralised detection approach, where each node analyses its own system metrics in order to achieve higher scalability. However, the LFD algorithm does not consider anomaly detection in cloud data centres and only focuses on node level metrics.

b) LIGHTWEIGHT ANOMALY DETECTION TOOL ALGORITHM AND FUNCTIONALITY

A cloud data centre comprises of large pools of compute, storage, and networking resources. The VMs in a cloud are hosted on compute nodes which manage the VMs with the help of the hypervisor. The major task of these nodes is to host the VMs, hence the largest part of the workload of these nodes consists of the VM execution. In an anomaly-free compute node most of the supporting services are designed so as not to interfere with the execution of the VMs. However, an anomaly on the hosting services could cause the node to utilise resources which are not justified by the VMs' workload. Moreover, a malicious attack could compromise the system security and let the intruder to access the compute nodes resources. Security exploits of this kind have been recorded in the past and they have been always addressed with high priority. An example of these exploits is the Blue Pill attack which manipulates kernel mode memory paging and the instructions that control the interaction between the host and guest (virtual machine). This permits undetected, on-the-fly placement of the host operating system in its own secure virtual machine allowing for complete control of the system including manipulation by other malware. Accessing the host resources in this way could interfere with workload of the registered VMs.

The LADT algorithm is based on the premise that there is a strong correlation between the node level and VM level metrics in a cloud system. This correlation will drop significantly in the event of any performance anomaly at the node-level and a continuous drop in the correlation can indicate the presence of a true anomaly in the node.

ALGORITHM

LADT is used to correlate node-level and VM-level metrics. The anomaly detection algorithm correlates disk I/O and CPU utilisation between the hosting node and the VMs in order to detect anomalies in IaaS Cloud environments. LADT computes the Pearson correlation coefficient (ρ) between the hosting node disk IOPS and the aggregated VM disk IOPS. Pearson's correlation coefficient is the ratio of the covariance between the two metrics to the product of their standard deviations as described in the following equation and ranges between -1.0 and 1.0.

$$\rho_{N,V} = \frac{covariance(N,V)}{\sigma_N \sigma_V}$$

where N = time-series of node disk IOPS and V = time-series of VM disk IOPS.

The LADT tool is based on the LFD algorithm (Tan et al., 2012) which is using a correlation analysis to find anomalies on a node by detecting significant changes in the relationship between the compute phase and the communication phase. The parameters for this tool are summarised on table 1.

Table 1 Tunable parameters in LADT Algorithm

Parameter	***Description***
LW, low-pass window width	raw samples to take mean
LS, low-pass window slide	raw samples to slide
W, correlation window width	samples to correlate
S, correlation window slide	samples to slide detection window
D, diagnosis window	correlation coefficients to consider

Before calculating the correlation coefficient between the metrics, their mean values are taken in small windows (*LW* = 5) in order to reduce noise. An anomaly alarm is raised when the average of *D* consecutive values of the correlation coefficient drops below the threshold T in order to detect the true anomalies by considering a larger range of system behaviour. We keep the same parameter values as the LFD algorithm for four parameters (*LW* = 5;*LS* = 5;*W* = 60;*D* = 10) (Tan, Kavulya, Gandhi, & Narasimhan, 2012), which we experimentally found to achieve best performance. However, the correlation window slide, *S* is changed from 12 to 60 as this amortises better the overhead of a Pig (Hadoop) program, which is used to execute the LADT algorithm. Therefore, each correlation coefficient value considers the last LW×W = 300 seconds (5 minutes) of system behaviour.

c) EVALUATION

In this section we demonstrate the functionality of the implemented tool and we analyse the performance of the tool. We describe the set up and the workload used for the experiment.

To test the functionality of our tool we used a compute node from an OpenStack cloud testbed. The host is a Dell PowerEdge R420 server which runs CentOS 6.6 and has 6 cores, 2-way hyper-threaded, clocked at 2.20 GHz with 12GB DRAM clocked at 1600 MHz. Each node includes two 7.2K RPM hard drives with 1TB of SATA in RAID 0 and a single 1GBE port. KVM is the default hypervisor of the node. The sampling rate of the performance metrics from the node is one measurement every three seconds. The node level metrics are generated using Sigar (Sigar, 2015) and the vm-level metrics are generated from Virt-Top (Virt-Top, 2015).

Four active instances on the node are running the Data Serving and the Graph Analytics benchmark from Cloud-Suite (Ferdman, et al., 2012). The Data Serving benchmark relies on the use of Cassandra, an open-source NoSQL datastore, stimulated by the Yahoo! Cloud Serving Benchmark. YCSB is a framework to benchmark data store systems. This framework comes with the interfaces to populate and stress many popular data serving systems. The second benchmark from Cloud-Suite is the Graph Analytics benchmark which uses the GraphLab machine learning and data mining software. Graph Analytics performs data analysis on large-scale graphs, using a distributed graph-processing system. Graph Analytics becomes increasingly important with the emergence of social networks such

as Facebook and Twitter. The graph analytics benchmark implements TunkRank on GraphLab, which provides the influence of a Twitter user based on the number of that user's followers.

The experiment runs over a time period of 60 minutes, where we inject an anomaly in the node at the end of the first 30 minutes using a disk and CPU stressing application, which periodically increases the disk read/write operations and CPU load runs for the remaining 30 minutes. This anomaly reflects a Blue Pill or a Virtual Machine Based Rootkit (VMBR) attack on a Cloud system, where the attacker introduces fake VMs via a hidden hypervisor on the victim hosting node to get access to the hardware resources such as CPU, memory, network or disk (Dahbur et al., 2011). Research such as (Antunes, Neves, & Verissimo, 2008) has already used system resource-level anomaly analysis to deal with such attacks. That research analyses system resource utilisation to explore the normal system behaviour and builds a model, based on which it detects the abnormal behaviours in the system, and subsequently, the attacks. However, this approach of detecting the attacks requires a large amount of historical data and use of machine learning techniques. LADT can detect the attacks using the correlation analysis between the node-level and the VM-level metrics. LADT is implemented in the testbed, which uses Chukwa agents in each of the hosting nodes to collect both the hosting node and VM disk read/write metrics. The tool stores the collected metrics in the HBase running in the monitoring node, which is a dual AMD Opteron 6272 server. Finally, in the monitoring node, LADT analyses the stored metrics by running the algorithm, which calculates the correlation between the individual hosting node metrics and the aggregated performance measurements of all the VMs in that node, to detect the injected anomaly. We tested two different implementations of the LADT algorithm. The first version was programmed in Pig Latin which is a high-level language for creating mapreduce programs used with Hadoop. The second version of LADT is based on Spark, which has an advanced DAG execution engine that supports cyclic data flow and in-memory computing. It is a fast and general engine for large-scale data processing. Spark uses the concept of resilient distributed datasets (RDD), which represent a read-only collection of objects partitioned across a set of machines that can be rebuilt if a partition is lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations.

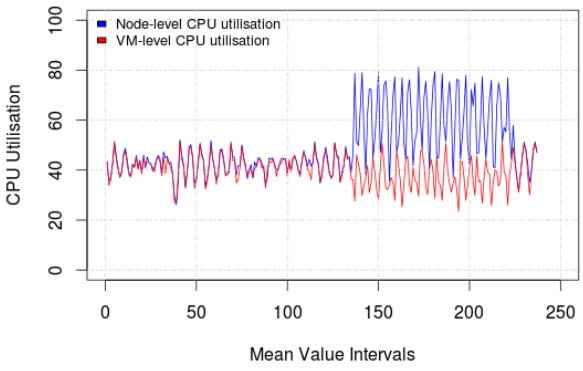

Figure 6 Time-series of node cpu utilisation(blue) and aggregated VM cpu utilisation (red)

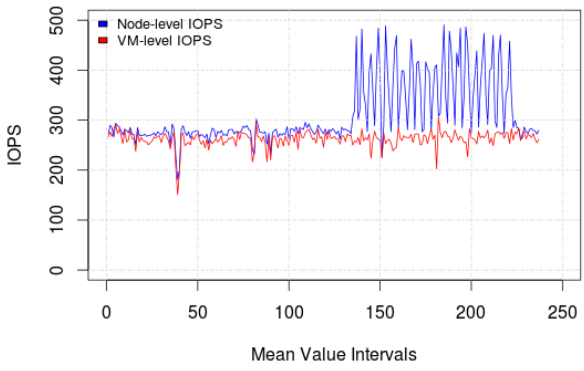

Figure 7 Time-series of node disk IOPS (blue) and aggregated VM disk IOPS (red)

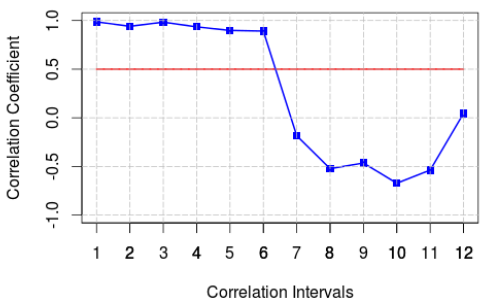

Figure 8 Correlation coefficients between the two cpu time-series measurements.

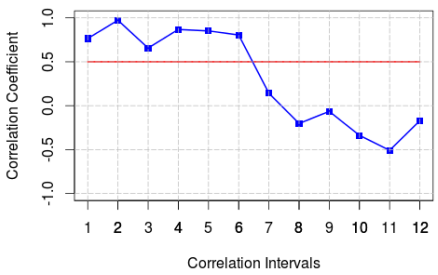

Figure 9 Correlation coefficients between the two disk IOPS time-series measurements.

In Figure 6 and Figure 7 we illustrate the data collected for the CPU utilisation and Input/Output Operations Per Second(IOPS) of the hosting node (blue line) and the aggregated VM CPU utilisation and IOPS (red line) respectively. We present the normalised values of the IOPS, which are the mean values in small windows of 15 seconds, including 5 samples of metrics data (the frequency of metrics collection is 3 seconds). The correlation coefficient values are calculated in correlation windows of 5 minutes, covering 5 minutes of metrics data. Hence, there are 12 correlation intervals in the 60 minutes of experiment. The fault injection in both cases appears after 30 minutes. In the case of IOPS , the node IOPS appear to be higher than the aggregated VM IOPS because of the overhead that is inflicted by the extra software layer of the hypervisor, which is interposed between guest operating systems and hardware (Li, Jin, Liao, Zhang, & Zhou, 2013). The hypervisor multiplexes I/O devices by requiring guest operating systems to access the real hardware indirectly and hence induces an overhead in the I/O context. The IOPS measurements are also taken with a different tool in each case (virt-top and Sigar), which means that the accuracy of each tool might differ. However, the correlation trace analysis is not significantly affected by these factors, since the major factor is the pattern and the relation of the two measurements.

The correlation analysis for CPU utilisation and IOPS in Figure 8 and Figure 9 respectively, clearly shows that there is a strong correlation between the hosting node metrics and the aggregated metrics of the VMs for the first half of the experiment, where the correlation coefficient values are above 0.5. However the correlation value drops below 0.0 suddenly at the sixth interval when the fault injection starts. The coefficient value remains very low throughout the second half. We consider correlation coefficient values above 0.5 (marked with red line) to

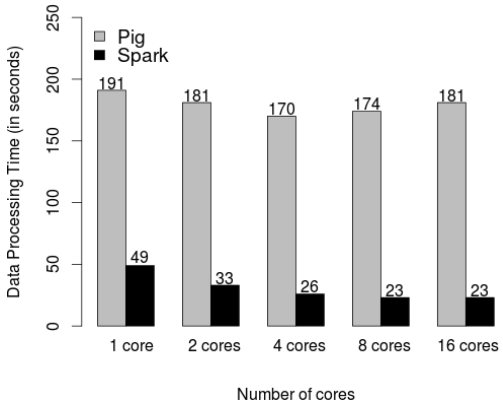

Figure 10 Time spent on the data processing phase.

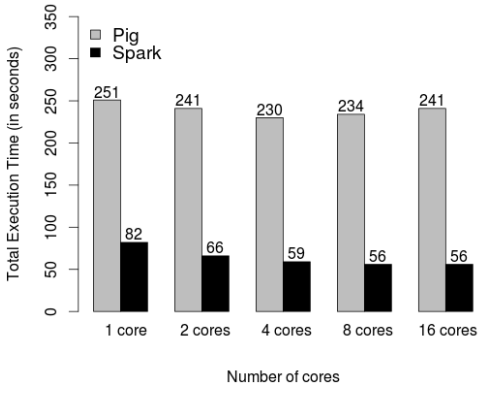

Figure 11 Time spent for the whole analysis process.

indicate a strong correlation in this case. This is a clear reflection of the injected anomaly in the host node, which distorts the correlation in both cases of CPU utilisation and IOPS.

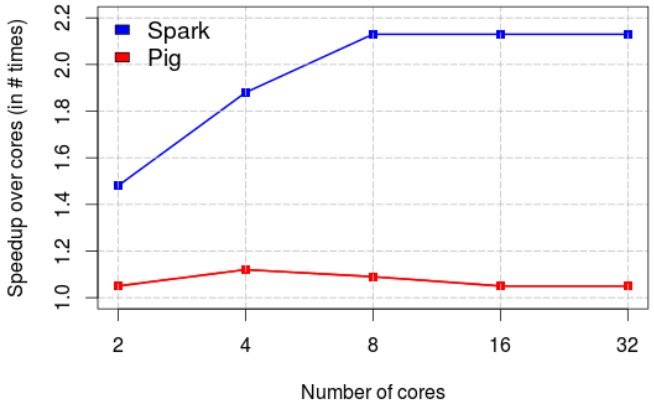

Figure 12 Speedup of Pig and Spark implementation of LADT

The time taken to process one hour of collected data with a different number of assigned cores is presented in Figure 10 and Figure 11. We observed experimentally that different runs did not yield any significant difference in execution time. The first figure is giving us the execution time that is needed for the section of the program that does the analysis on the collected data. This section does not include loading the values from HBase into proper structures that can be processed by Hadoop or Spark. The reason we exclude this part in the first figure is that it cannot be simply parallelised by just adding more working cores to the node and the procedure can be limited by having to access the data from the node's disk. This becomes clearer when we do a comparison with Figure 11 where we realise that there is a fixed difference between the graphs of the two figures which remains the same for different number of cores. As we notice, the Spark implementation has an advantage over running Pig Latin. Spark uses resilient distributed datasets (RDD), which can be cached in memory across machines and be reused across multiple parallel operations. Furthermore, Pig Latin adds an abstraction layer over Hadoop which could further increase the competition time of the map-reduce analysis. Another observation based on the number of assigned cores is that the performance of the implementation in Pig improves by increasing the core number up to

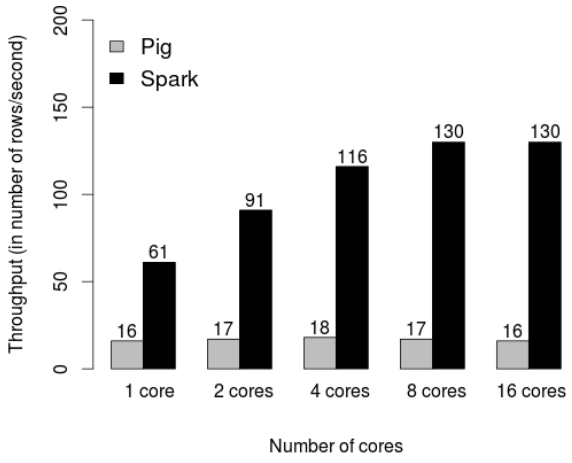

Figure 13 Number of row-keys processed per second during the data processing phase.

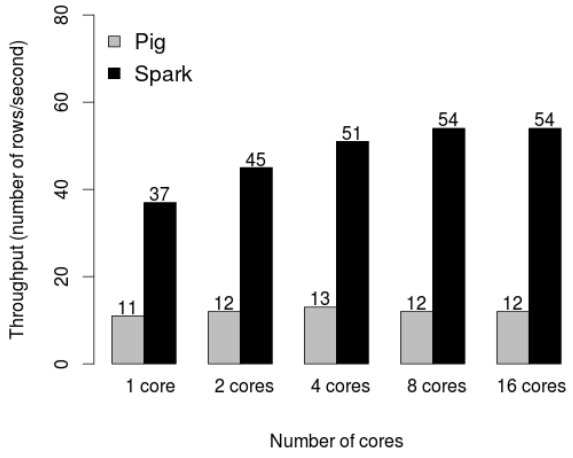

Figure 14 Number of row-keys processed per second for the overall time.

4, but after that the performance is deteriorating for the examined amount of data. Spark implementation is improving while adding up to 8 cores, but remains the same for 16 cores. This observation is clearer when we observe the speedup of the two implementations of LADT in Figure 12.

The throughput of LADT is depicted in Figure 13 and Figure 14. In the first case we calculate the throughput in terms of total HBase rowkeys processed per second when we consider only the data processing part of the application. The second case is showing the number of records processed per second if we include loading the data from HBase into structures which can be processed by Spark and Pig.

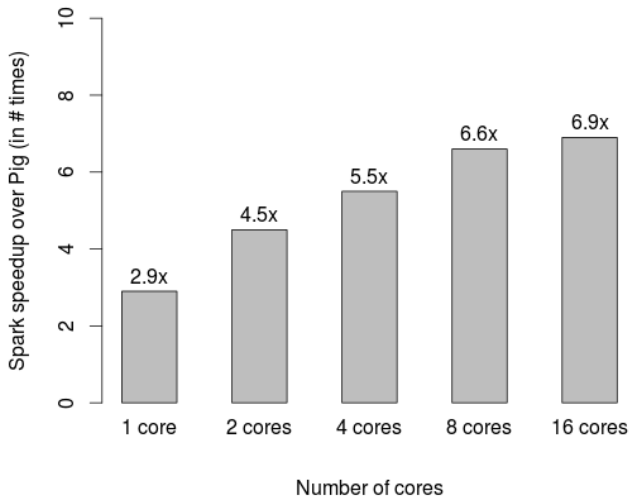

Figure 15 Spark speedup over Pig

To compare the two different implementations of LADT in Spark and Pig Latin, we present in Figure 15 the speedup achieved when transitioning from mapreduce with Pig scripts to Spark. In this figure we notice that the Spark implementation performs better as the number of cores increases. Therefore, performing correlation analysis in Spark has more potential and is more scalable. In future, we plan to increase the speedup ratio of LADT by exploiting opportunities for parallelisation that come from grouping performance data from separate nodes.

2. TEXTUAL LOG ANALYSIS

Cloud data centres are implemented as large-scale clusters with demanding requirements for service performance, availability and cost of operation. As a result of scale and complexity, data centres exhibit large numbers of system anomalies caused by operator error (Oppenheimer, Ganapathi, & Patterson, 2003), resource over/under provisioning (Kumar, Cooper, Eisenhauer, & Schwan, 2007), hardware or software failures (Pertet & Narasimhan, Causes of failure in web applications, 2005) and security issues. These anomalies are inherently difficult to identify and resolve promptly via human inspection (Kephart & Chess, 2003). Thus, automatic system monitoring that captures system state, behaviour and performance becomes vital. Computer system logs are the main source of information for anomaly detection.

a) ANOMALY DETECTION TOOLS BASED ON CONSOLE LOG ANALYSIS

Analytic tools for anomaly detection based on console logs, such as SEC (Rouillard, 2004), Logsurfer (Prewett, 2003) and Swatch (Hansen & Atkins, 1993) check logs against a set of rules which define normal system behaviour. These rules are manually set by developers based on their knowledge of system design and implementation.

However, rule-based log analysis is complex and expensive because it requires significant effort from system developers to manually set and tune the rules. Moreover, modern systems consisting of multiple components developed by different vendors and the frequent upgrades of those components make it difficult for a single expert to have complete knowledge of the total system and set the rules effectively. This complexity gave rise to statistical learning based log analytic tools like (Lou, Fu, Yang, Xu, & Li, 2010), which extract features from console logs and then use statistical techniques to automatically build models for system anomaly identification. (Lou, Fu, Yang, Xu, & Li, 2010) proposes a statistical learning technique which consists of a learning process and a detection process. The learning process groups the log message parameters and then discovers the invariants among the different parameters within the groups. For new input logs, the detection process matches their invariants among the parameters with the learned invariants from learning process. Each mismatch in the invariants is considered as anomaly. (Xu, Huang, Fox, Patterson, & Jordan, 2009) proposes a new methodology to mine console logs to automatically detect system problems. This first creates feature vectors from the logs and then applies the PCA (Principal Component Analysis) algorithm on the feature vectors to detect anomalies. However, the learning based tools require a custom log parser for mining the console logs in order to create the features for the learned model. The log parsers requires source code of the hosted applications to recover the inherit structure of logs.

b) ALGORITHM DESCRIPTION

In a cloud data centre which supports automatic management of the resources, there is a series of different actions happening on the background in order to achieve different goals set by the operator, some of the most important goals are to keep the workload balanced, maintain Service Level Agreements (SLAs) and optimise energy consumption. These macroscopic targets imply certain VM actions happening on each node of the cloud. The following are some examples of a VM action on a node: migration, new VM creation, suspend, terminate, resize. The VM actions and any potential errors on a node are registered into different log files. The amount of information on the log file and the large number of nodes included in a cloud platform makes it difficult for cloud operators to track important events.

Tracking VM actions efficiently is important for the analysis of the system performance and allows cloud data centre operators to manage their platform. In order for a VM action to complete, it requires different amount of resources before it boots up. For example, creating a new instance on a node would require claiming memory, disk and VCPUs from the node by the VM. It would also create a temporal network and disk I/O bandwidth consumption on the node for the VM image to be transferred. All the aforementioned resource demands could cause an analysis on the node to lead to faulty results if they are not taken into account. For this reason we examine coupling the parallel correlation analysis of LADT with the functionality of a cloud log tracking mechanism to reduce any false positives by the detection mechanism.

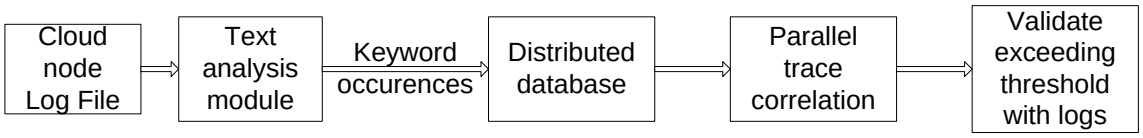

Figure 16 Processing of log data

The procedure of log processing is displayed in Figure 16. The process starts from each compute node of the cloud data centre where the log file is generated. A text analysis module is triggered periodically by an agent resided in each node. The module is tailing the file for the logs of the most recent time interval. Then it searches for a set of specific patterns of words and phrases that indicated certain VM actions or errors. For example, the phrase "Attempting claim: memory *X* MB, disk *Y* GB, VCPUs *Z*" indicates that a new VM allocation is in progress on the

current node. Other examples of interesting occurrences are: “Instance destroyed successfully”, “Migrating instance to X finished successfully”, “QEMU: error”. These occurrences are then organised into an HBase row-key with the key *time_period-node* and a column family *VMActions* which has different columns such as “Placement” and “migration”. Each of these columns has a value equal to the number of occurrences that “placement” and “migration” related keywords/phrases appeared during the period and by the node marked by the row-key. In this way the actual log files are filtered and only a statistical row-key is recorded in the datastore. Part of the information in the log file is unique and contains information relevant to the service that produced the log. This means that it is difficult to retrieve this information by another way, since this information might not be available elsewhere. The LADT tool performs the correlation analysis using Spark and in the case of detected anomalies it tries to validate them by looking on the entries of the related period to see if these alarms are justified by other VM management activities on the current node.

c) EVALUATION

In the following section we examine the effect of a new VM placement on the node. We identify any correlation drops and examine if these are justified by any activities that are recorded by the text analysis tool. We also analyse any overhead that the text analysis adds to the LADT tool. We use the same experimental setup and workload as in the LADT evaluation for testing purposes.

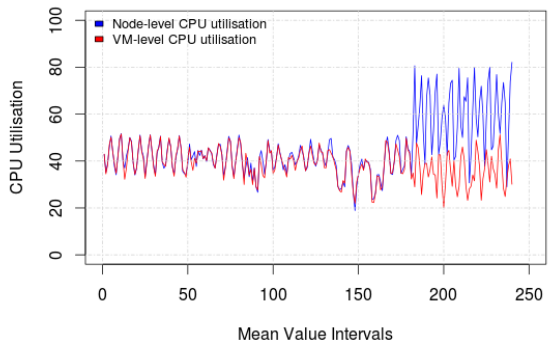

Figure 17 Time-series of node cpu utilisation(blue) and aggregated VM cpu utilisation (red)

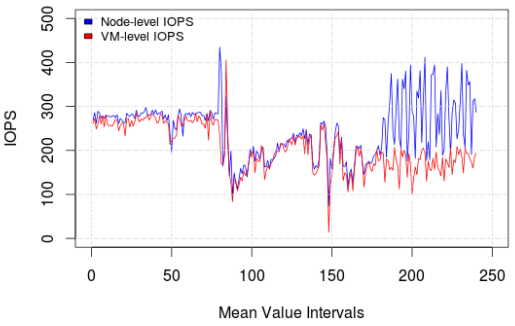

Figure 18 Time-series of node disk IOPS (blue) and aggregated VM disk IOPS (red)

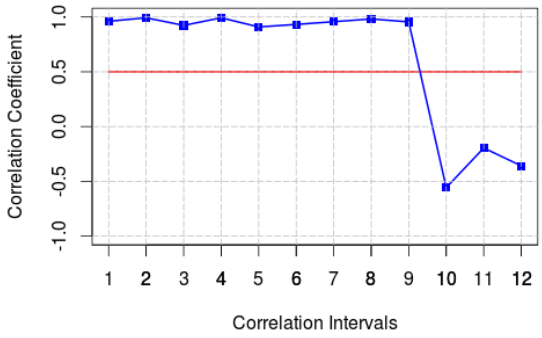

Figure 19 Correlation coefficients between the two CPU time-series measurements.

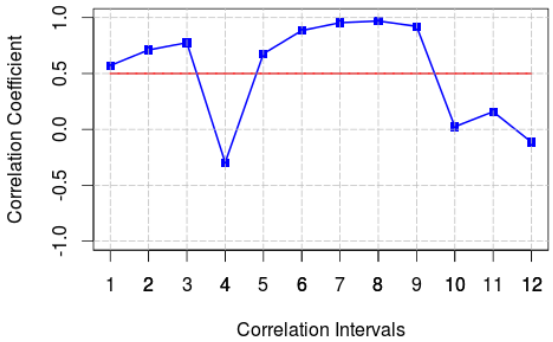

Figure 20 Correlation coefficients between the two disk IOPS time-series measurements.

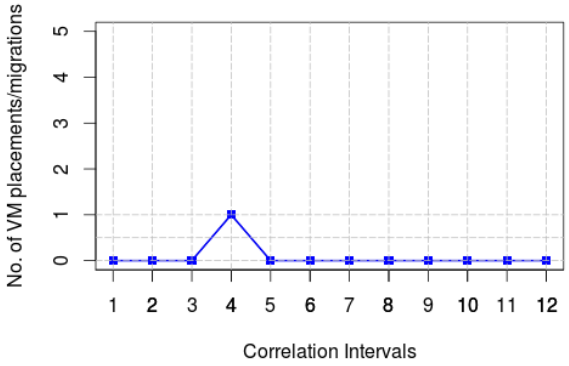

Figure 21 VM activity record derived from in-situ log analysis

The graphs in Figure 17 and Figure 18 present the CPU utilisation and IOPS for the node and the VMs. In the case of the VMs the measurements are aggregated for all the active VMs. In this scenario, we examine the correlation analysis using the Spark implementation of the extended LADT under the influence of a new VM allocation and an injected anomaly occurring at two different time frames. The VM allocation is taking place after 15 minutes and the fault injection is happening on the 45th minute. The VM allocation requires only a few minutes (approximately 5 minutes) to be fulfilled in this case.

The parallel trace analysis tool is examining the correlation results (Figure 19 and Figure 20) along with the HBase records of the VM placement. We notice that the first occasion where the correlation values for the IOPs (Figure 20) drops is when the VM placement takes place (at the 4rth time interval of Figure 21). The correlation drop is explained if we consider that the VM allocation required a significant amount of disk to store the new VM instance. In the case of the correlation analysis of the CPU utilisation in Figure 19 , we observe that the correlation values remain above the 0.5 threshold during the allocation of the new VM on the node. The reason is that the process of allocating the new VM does not require significant amount of CPU utilisation. Therefore, the correlation analysis of the CPU utilisation remains unaffected. The second drop in Figure 20 is matching the appearance of the injected fault. The same fault also affects the CPU correlation in Figure 19 and there is no indicated VM action in Figure 21. Furthermore, the duration of the fault is longer that the VM allocation correlation drop and also affects both IOPS and CPU utilisation. All these factors are indicating that this is an actual anomaly instead of a false positive generated by normal cloud operation activity. We conclude that the analysis of textual log files plays an important role in deciding whether an anomaly alert is true or just a false positive.

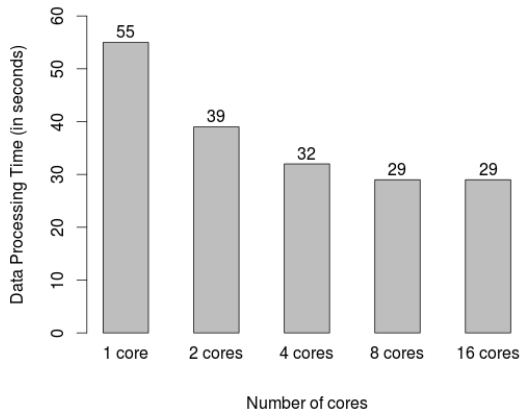

Figure 22 Time spent on the data processing phase.

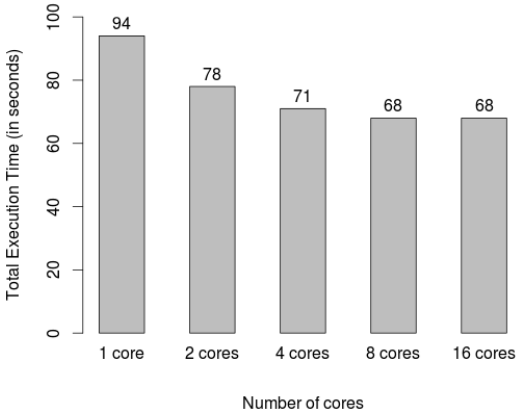

Figure 23 Time spent for the whole analysis process.

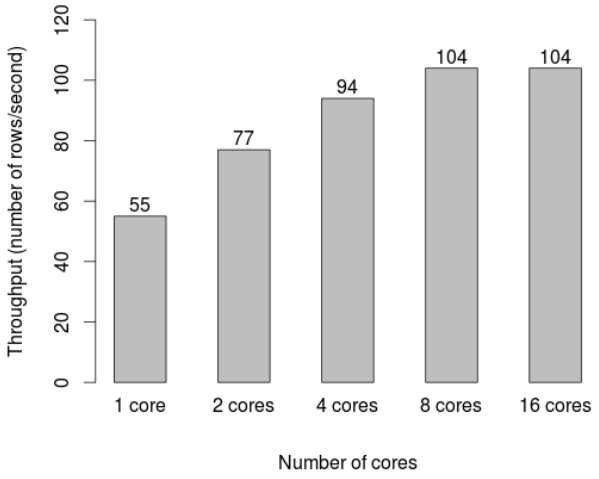

Figure 24 Number of row-keys processed per second during the data processing phase.

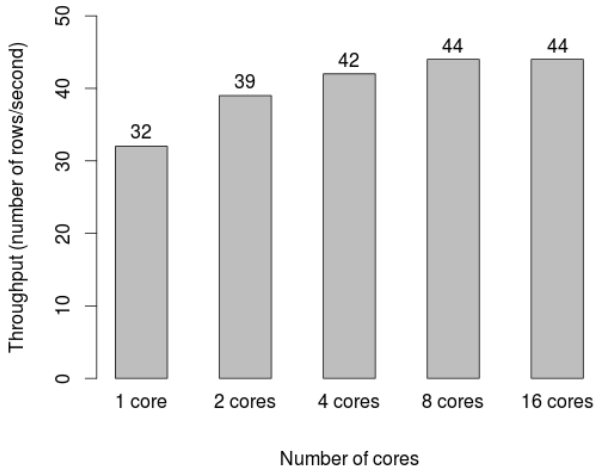

Figure 25 Number of row-keys processed per second for the overall time.

In this scenario we interested in examining the overhead on the Spark LADT parallel trace analysis tool when this is modified to take into account log statistics which are stored in HBase. For this reason we measure the time taken to process one hour of collected data with a different number of assigned cores in Figure 22 and Figure 23. When compared to the results of plain LADT in Figure 10 and Figure 11 we realise that the log analysis requires a fixed amount of overhead which involves retrieving the information from HBase and doing the log comparison. We believe that this amount of time can be further reduced in the future by restructuring the log statistics information in separate HBase table where the search for the desired data can be reduced. In the last set of figures (Figure 24 and Figure 25) we observe the impact of this overhead over different number of cores on the data throughput when compared to Figure 13 and Figure 14. We believe that this overhead does not affect scalability significantly because it remains the same for 1 to 16 cores used (approximately 6 seconds for the data processing phase and 8 seconds in total). Finally, further optimisations in distributing this analysis over multiple hosts could lead to further improvement on the performance of this tool in the future.

3. Performance and Model Anomalies

To be able to optimize the infrastructure proactively, CactoOpt requires early detection of workload spikes or performance anomalies. This section provides an overview of the methods developed within CactoScale to allow CactoOpt to proactively deal with workload spikes (Mehta, Durango, Tordsson, & Elmroth, 2015) as they develop. The problem of uncovering and understanding performance anomalies and their causes in different system and application domains has been studied for decades. In the context of cloud infrastructures, new complications are added due to the scale, dynamics, and heterogeneity of the infrastructure. In this section, we provide an overview of the state-of-the-art on anomaly detection in general giving special focus to spikes. Since workload spikes are one of the main causes of QoS degradation in cloud applications, we have designed a number of new methods for early spike detection in a workload. This section summarizes two recent publications describing the methods used in CactoScale (Ibidunmoye, Hernandez-Rodriguez, & Elmroth, 2015) (Mehta, Durango, Tordsson, & Elmroth, 2015).

a) Anomaly Detection: State-of-the-art

Many factors such as varying application load and spikes, application issues (e.g. bugs and updates), architectural features, hardware failure, have been found to be sources of performance degradation in large scale systems (Joao Paulo & Silva, 2010). According to some reports, Amazon, experiences 1% decrease in sales for additional 100 ms delay in response time while Google reports a 20% drop in traffic due to 500 ms delay in response time. These

implications show not only the importance but also the potential economical value of robust and automated (Chandola , Banerjee, & Kumar, 2009)solutions for detecting workload spikes in real time. Bottleneck conditions such as system overload, and resource exhaustion have been reported to cause prolonged and intermittent system downtimes (Pertet & Narasimhan, Causes of failure in web applications , 2005). Global web services such as Yahoo Mail, Amazon Web Services, Google, LinkedIn, and Facebook have recently suffered from such failures (McHugh., 2013). To achieve guaranteed service reliability, performance, and Quality of Service (QoS) timely detection of performance issues before they trigger unforeseen service downtime is critical for service providers.

Generally, anomalies can be seen on a graph, as a point or group of data points lying outside an expected normal region. Chandola et al (Chandola , Banerjee, & Kumar, 2009) identifies three basic types of anomalies; point, collective and contextual anomalies. These types only capture anomaly in terms of individual or contiguous data points, however, performance metrics are also known to commonly exhibit characteristic shapes when a resource is saturated. Therefore we present one more type of anomalies, the pattern anomaly, which characterizes performance behaviours in terms of the structure or shapes of their curves rather than finite data points (Gunther, 2011).

- *Point anomalies.* A point anomaly is any point that deviates from the range of expected values in a given set of data. For example, a workload where the number of requests increases at just one monitoring interval by 3 standard deviation from the mean (i.e. $>= \mu \pm 3\sigma$) may be considered a point anomaly if the expected behaviour is 1 standard deviation from the mean (i.e. $<= \mu \pm 1\sigma$). Point anomalies are the dominant type of anomalies in majority of literature that we reviewed. They commonly manifest as spikes in application latency or system resource utilization measurements. While common, in many occasions these very short spikes are monitoring errors.
- *Collective anomalies.* Collective anomaly is a homogeneous group of data points deviating from the normal regions of the rest of the data. Though the individual data points may not be anomalous with respect to the group, their occurrence together as a collection is anomalous.
- *Contextual anomalies*. Some performance anomalies manifest only under specific execution environments or contexts. The contexts may be defined by load levels (e.g. high, moderate, load, or bursty), type of loads (e.g. IO-bound, CPU-bound, read-heavy, write-heavy or mixed), system states (e.g. system configurations), or by the nature of underlying computing infrastructure (e.g. virtualized or shared-hosting environments) etc.
- Pattern anomalies. The shapes of some performance metrics when plotted are known to exhibit a specific pattern that can be used to identify anomalous behaviours (Gunther, 2011), e.g., a workload with diurnal patterns. The shape or pattern of performance metric may be anomalous if it does not conform to the shape of typical behaviour. Pattern anomalies may be considered a generalized form of collective anomalies since the anomalous shape is made up of a set of data points that are as well collective anomalies.

Bursty application loads are characterized by periods of continuous peak arrival rates that significantly deviate from the average or expected workload intensity. Internet phenomena such as flash-crowds culminate into workload burstiness (Mi, Casale, Cherkasova, & Smirni., 2008). The undesirable effects of such load behaviour include congested queues, oversubscribed threading resources, present of short and uneven peaks in resource and performance measurements (Casale, Kalbasi, Krishnamurthy, & Rolia., 2009) .

Conventionally, the approach to detecting performance problems involves continuous estimation of models of normal system behaviours at specific points of interest. New performance observations that fail to match (within some acceptable confidence levels) existing models are flagged anomalous. However, many solutions employ

more complicated techniques (such as statistical and learning methods) while following one or more strategies to achieve some detection goals.

Existing anomaly detection systems often follow one or more strategies for robustness. A strategy defines the set of policies to achieve a detection goal. The choice of strategy is greatly influenced by system observability as well as whether the detection is to take place in either offline or online mode. Offline detection is a "post-mortem" identification and analysis of performance issues. Online detection is performed at run-time. All strategies use thresholding to prune and complement detection decisions in one way or the other. A threshold is a limit value or range of values for parameters or metrics of interest beyond which an event is raised. Setting thresholds becomes cumbersome when many parameters and metrics are involved. Modern systems exhibit dynamic behaviours that consistently violates the ideal set thresholds. It is therefore expedient that the right thresholds are estimated. Threshold values are also expected to evolve with respect to change in underlying execution environment. In addition, it is crucial to understand the sensitivity of varying thresholds on detection accuracy.

b) ONLINE SPIKE DETECTION IN CACTOSCALE

Unanticipated changes in workload characteristics can potentially lead to service slowdown and end in service-failure due to insufficient resource allocation. For this reason, having the ability to early detect the build up of workload spikes is beneficial for making proactive resource management decisions.

The problem of detecting workload spikes is complicated by the fact that no coherent definition of workload spikes exists. This, of course, makes any attempt to characterize them harder. Spikes manifest themselves by sudden changes in a workloads statistical properties. This can include changing mean, variance and autocorrelation structure. The abnormality that generates spikes can be due to either internal or external events. The occurrence of events can be well-known in advance, such as a sporting event generating large amounts of traffic for a streaming service, or completely unexpected. For CactoScale, we focus on the detection of unanticipated spikes in cloud workloads.

Detection and identification of spikes is tightly coupled to workload modeling. In CactoScale, we adopt the view that spikes can be seen as significant deviations in any aspect of a workload from a predicted output of a model of said workload. Such models might already be readily available for some workloads, or may be derived as part of designing spike detection algorithms. In this paper we employ the latter approach. The view that spikes are considered deviations from a model implies that the problem of detecting spikes is heavily application dependent: while a large sudden burst in traffic for one application could be considered a spike, if it occurs regularly due to some well understood reason, it could probably be modeled and therefore not be considered a spike.

In CactoScale, we perform online workload spike detection using adaptive signal processing techniques. In this section, we derive a number of workload models that are coupled with a stopping rule for detecting the onset of a spike and evaluate them using the well-known and publicly available FIFA 1998 World Cup workload (Arlitt & Jin, 2000), shown in Figure 26. Of special interest is the ability to detect the onset of a spike as early as possible, thus leaving enough time to mitigate its effects. We explore the trade-off between the ability to correctly detect large traffic increases as spikes and how prone the detectors are to generate false alarms when no spike is present. Also explored is the trade-off between how early detections can be made, and the number of false alarms generated. We conclude that these are tradeoffs that need to be carefully considered by an application owner, as for example some applications can withstand a fairly large fraction of undetected spikes, while it can only make use of a detection if it arrives well in time.

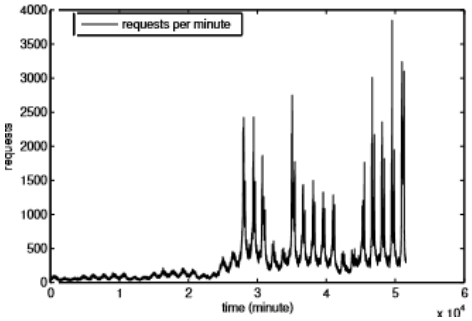

Figure 26 FIFA 1998 worldcup servers' workload

STATE-OF-THE-ART IN MODELING AND ONLINE SPIKE DETECTING

Previous efforts have been made on the topic of modeling and characterizing workloads and spikes, see (Downey & Feitelson, 1999), (Bodik, Fox, Franklin, Jordan, & Patterson, 2010) and (Arlitt & Jin, 2000) . Bodik et al. (Bodik, Fox, Franklin, Jordan, & Patterson, 2010) classify spikes into volume spikes (sudden increase in the total workload of a system) and data spikes (sudden increase in demand of a certain object). Spikes are characterized using seven statistical parameters, and modeled into four different phases: increase, plateau, decrease and normal. Ali-Eldin et al., (Eldin, et al., 2014) show how spikes on one Wikipedia entry can result in major spikes in other related Wikipedia entries. These collateral spikes can be larger than the spikes on the original page. In (Lassnig, Fahringer, Garonne, Molfetas, & Branco, 2010) the authors analyze the accuracy of different existing models for predicting workload spikes, continue by developing a new model, and finally present a novel metric for assessing the prediction accuracy of said model. Gusella (Gusella, 1991) introduces the concept of characterizing burstiness in time series of workload arrivals using the index of dispersion. In (Mi, Casale, Cherkasova, & Smirni., 2008)the authors describe how the index of dispersion can be used to model and parametrize bursty workloads when investigating service times in multi-tier applications. Caniff et al. (Caniff, Lu, Mi, Cherkasova, & Smirni, 2010) presents Fasttrack, a dynamic resource provisioning solution for multi-tiered applications that estimates the index of dispersion and utilizes it for determining when the workload is entering and exiting a bursty state.

ONLINE SPIKE DETECTION

There are several possible approaches for performing spike detection in cloud workloads. We use a commonly employed technique in adaptive signal processing: combining a model of the system or signal of interest, which in our case is the workload, with a stopping rule that signals whenever it detects that a system change has occurred. In particular, we employ the one-model approach described in (Gustafsson, 2000). In this approach, the workload measurements y_t, t = 0,1,..., are compared to the one-step ahead predicted output $\hat{Y}_{t|t-1}$ of a model that has been derived. If the model correctly captures the dynamics of the workload, taking the difference between the workload and prediction will result in the residuals,

$$\varepsilon_t = y_t - \hat{y}_{t|t-1} \qquad (1.1)$$

looking like a white noise sequence, i.e. as a sequence of independent, identically distributed (often Gaussian) random variables. If, at any point in time, an event takes place that changes the system in such a manner that the existing model is no longer able to correctly describe the system behaviour, this will show up as a change in the characteristics of the residuals (changing mean, variance, autocorrelation, etc.). In accordance with (Gustafsson, 2000), the residuals are used for deriving a so-called distance measure $s_t = f(\varepsilon_t)$, which is then used in conjunction with a stopping rule for detecting when a change has taken place. The particular choice of distance measure

depends on the type of change that we are interested in detecting. Some common choices include using the residuals directly, i.e. $s_t = \varepsilon_t$, which is useful for detecting changing means, and the squared residuals, $s_t = \varepsilon_t^2$, which is useful for detecting changes in variance.

STOPPING RULE- CUSUM TEST

The idea behind using stopping rules for change detection is simple: by monitoring critical aspects of interest of the system under investigation, we sound an alarm when these aspects exceed a threshold. Here we have adopted the well-known CUSUM test (Page, 1954), which can be considered a special case of the sequential probability ratio test (Wald, 1945). In the CUSUM test, the distance measure sequence s_t is cumulatively summed up (hence the name CUSUM – cumulative sum) to calculate the test statistic g_t until the sum hits a preset threshold *h*, at which time point an alarm is raised and the sum is reset to zero. During the calculation, g_t is bounded from below, so if at any time point $g_t < 0$, it will also then be reset to zero. If the distance measure s_t used is the residuals ε_t themselves, and the workload model used is correct so that the residuals become a white noise sequence, g_t will undergo a random walk. This will lead to g_t crossing the threshold h spuriously even though no change has taken place, leading to false alarms. For this reason, a drift term *v* is used in the calculation of g_t to slowly push it back to zero. Algorithm 1 outlines the pseudo code for the CUSUM test used. Both the threshold *h* and drift term *v* are concerned with determining the speed and robustness of the detector. For example, a careful choice of *v* should ensure that g_t is kept small enough to not cross the threshold randomly, but still allow for g_t to cross the threshold when a change has taken place. On the other hand, a large robustness to random alarms can come at the cost of relative large delays before a detection is made. This tradeoff is problem-specific. To avoid too frequent alarming, we implement a hanging window approach. When an alarm is fired, any subsequent detections during the duration of the hanging window length are ignored. When the hanging window expires, the next detection can trigger a new alarm. Another possible solution that is often employed to avoid too frequent alarming is to raise an alarm only if two or more detections during a short time period are made.

Algorithm 1 CUSUM test

```
Input: drift ν, threshold h
Output : alarm times
g_0 ← 0
for t = 1 to T do
    g_t ← max(g_{t−1} + s_t − ν, 0)
    if g_t > h then
        g_t ← 0
        alarm ← 1
```

TIME AGGREGATION OF WORKLOAD DATA

For modelling purposes, choosing a high enough sampling frequency for a signal is crucial for fully capturing the dynamics of the workload. On the other hand, a high sampling frequency comes with the drawback of increased noise levels. Yet another drawback is the need for increased model complexity when increasing the sampling frequency, e.g., in order to describe workload pattern variations on an hourly basis we require a higher model complexity if new measurements arrive every second, as opposed to every ten minutes. We propose an approach where we utilize incoming workload measurements aggregated on two timescales. Apart from using the actual workload measurements *y* we construct also the more coarsely-granulated measurement sequence $\{y^H_k\}$ by aggregating the measurements in bins of size t_s:

$$y_k^H = \sum_{i=s}^{k \cdot t_s} y_i \qquad (1.2)$$

where $s = (k - 1) \cdot t_s + 1$. From this sequence we derive fairly simple models that can capture the general workload dynamics. In Figure 27 we show the effect of aggregating measurements on a more coarsely-granulated timescale.

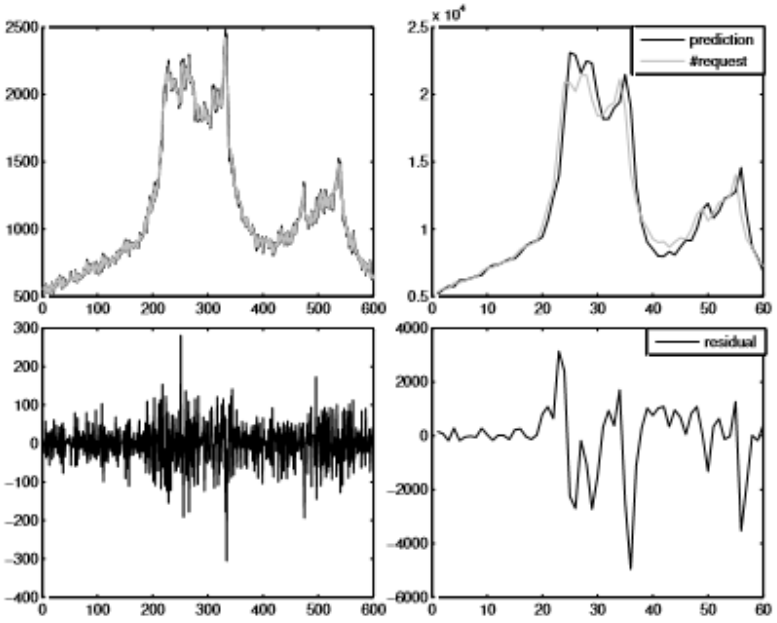

Figure 27 The effect of aggregating workload measurements on a more coarsely granulated timescale. Left: the original measurement sequence and residuals when calculating one-step ahead prediction using an AR model. Right: same as left, but for data aggregated on

ADAPTIVE FILTER MODELS

A workload model can be derived in a multitude of ways. Which one to choose is very much dependent on the purpose of modelling. What they should have in common is the ability to provide a model that encompasses the dominating features of the workload while being agnostic to features that are not considered important for the task at hand. Here we present a number of workload models based on adaptive filtering methods.

1. Auto-regressive modelling using time aggregation (AR-TA): in the first approach, we make use of the time aggregation technique described before and estimate an auto regressive (AR) model of order *p* using the coarsely-granulated data. This gives a model on the form:

$$y_k^H = \phi_0 + \sum_{i=1}^{p} \phi_i y_{k-i}^H + e_k \qquad (1.3)$$

where e_t is white noise. The parameters $\phi = [\phi_0,...,\phi_p]$ are estimated using ordinary least squares over a sliding time window of size *wl* and re-estimated every time step in the coarsely-granulated timescale. The one-step ahead prediction of the workload using this model is:

$$\hat{y}_{k|k-1}^H = \phi_i y_{k-i}^H \qquad (1.4)$$

The one-step ahead prediction on the coarsely granulated timescale is then up-sampled to the finer timescale at which workload measurements arrive using linear interpolation. The combination of two time scales allows us to predict, using the coarsely-granulated timescale, the general direction in which the workload is heading

and compare that to the behaviour of the workload on the finer timescale. If a spike is building up, the incoming workload measurements will start deviating from the prediction. The modified CUSUM test outlined in Algorithm 2 is used for detection purposes.

Algorithm 2 Modified CUSUM test

```
1: Input: drift ν, threshold h
2: Output : alarm times
3: g_prev ← 0
4: for k = 1 to K do
5:     g_0 ← 0
6:     for t = 1 to t_s do
7:         g_t ← max(g_{t−1} + s_t + g_prev − ν, 0)
8:         g_prev ← 0
9:         if g_t ≥ h then
10:            g_t ← 0
11:            alarm ← 1
12:        else if t = t_s then
13:            g_prev ← g_t
```

2. Double exponential smoothing using time aggregation (DS-TA): in the second approach, we combine the time aggregation technique with a double exponential smoothing model for computing the one-step ahead prediction. The model assigns exponentially decreasing weights over time and takes into account the trend of the workload. The smoothed data S_k and trend b_k for the coarsely-granulated measurements y^H_k at time instant $k > 1$ is estimated as

$$S_k = \alpha y_k^H + (1-a)(S_{k-1} + b_{k-1}) \qquad (1.5)$$

$$b_k = \beta(S_k - S_{k-1}) + (1-\beta) b_{k-1} \qquad (1.6)$$

where α is a data smoothing factor, $0 < \alpha < 1$, and β is the trend smoothing factor, $0 < \beta < 1$. We initialize $S_1 = y_1^H$ and $b1 = y_1^H - y_0^H$. The one-step ahead prediction is given by

$$\hat{y}^H_{k|k-1} = S_{k-1} + b_{k-1} \qquad (1.7)$$

As for AR-TA, the one-step ahead prediction is upsampled to the finer-granulated timescale using linear interpolation. The procedure for detecting spikes is also identical to that for AR-TA.

3. Lowpass filtered differentiation (Diff) implicitly assumes that the workload will change fairly slowly. By differentiating the workload measurements, we can estimate the rate of change in the workload. During normal traffic, the rate of change will be distributed around zero. In contrast, during a spike we expect the mean rate of change to transiently increase to a positive number, a fact that is exploited for spike detection. Since differentiating measurements often gives a noisy result, we improve the signal-to-noise ratio by lowpass filtering the differentiated signal using a second order Butterworth filter before feeding it to the CUSUM detector. This allows us to more robustly detect when the rate of change is deviating from zero.
4. Constant mean (CM): the final approach takes a fairly agnostic view of the workload by using a model that simply assumes a constant workload mean:

$$\hat{y}_t = \theta + e_t \qquad (1.8)$$

where ϑ is the workload mean and e_t white noise. The one-step ahead prediction then simply becomes

$$\hat{y}_{t|t-1} = \theta \qquad (1.9)$$

The mean ϑ is estimated online using recursive least squares as new measurements become available. In order to still allow for some slow variation in the mean when computing the estimate, a forgetting factor λ that puts exponentially decreasing weight on old data is used. During a spike, the measurements will significantly deviate from the estimated long term mean, leading to the residuals $\varepsilon_t = y_t - output\ \hat{Y}_{t\ |t-1}$ no longer being distributed around zero. The residuals are then used in the CUSUM test to detect spikes.

c) EVALUATION

For this evaluation, we have used a workload with significant spikes, the FIFA 1998 worldcup servers' workload, which consists of the count of requests made to the webservers serving the official website during the lead-up to and the final stages of the tournament. The sampling time of the measurement sequence is one minute. During data pre-processing, we manually identify $N_s = 19$ spikes throughout the workload duration. For each spike, we manually record the starting time, ending time and duration of the spike. The starting points in time are not picked according to a strict definition, but rather to cover the complete rising phase of all spikes. The ending times are consistently chosen to coincide with the peak of each spike. During evaluation we calculate a set of different metrics appropriate for describing the performance of the detectors. If an alarm is raised during one of the manually identified spike intervals, we count that as a hit (true positive). An alarm raised outside of the identified spike intervals is counted as a miss (false positive). Note that this implies that alarms raised during the decreasing phase of a spike are counted as misses, which is natural since an alarm that late into a spike will probably be useless from a proactive resource management point of view. As is commonly done in pattern recognition, we calculate the so-called precision and recall metrics, defined as

$$precision = 100 \times \frac{\#hits}{\#hits + \#misses} \qquad (1.10)$$

$$recall = 100 \times \frac{\#hits}{N} \qquad (1.11)$$

Using these metrics we can also calculate the F-score defined as the harmonic mean of recall and precision

$$F = 2 \times \frac{precision \times recall}{precision + recall} \qquad (1.12)$$

Additionally, in order to capture the performance of the detectors in terms of how early or late they detect a spike when a hit is generated we define for each detector the average time before peak (ATBP) as the average time interval between the time point at which a hit is generated and the time point at which the peak of the corresponding spike occurs. We also calculate the difference between the peak workload level during each correctly detected spike and the workload level at the time of detection, and define the average relative change (ARC) as the average ratio between said difference and the peak workload level. We make use of a hanging window to avoid too frequent alarming. We set this parameter to 205 minutes, which coincides with the average length of the manually identified spikes. We have compared out method to the Fano Factor used for describing burstiness in neural spike trains. The evaluation results are shown in Table 2.

Table 2 Results for the evaluation of the algorithms for spike detection

Method	Recall	Precision	ATBP
AR-TA	100	79.2	109.2
DS-TA	100	76	91.6
Diff	100	79.2	99
CM	100	79.2	101.9
FF	89.47	31.12	133.4

IV. SUMMARY AND CONCLUSION

This deliverable described CactoScale framework for fast and scalable analysis. We demonstrated new parallel trace analysis tools introduced into CactoScale based on anomaly detection use case scenarios and we described three different categories of anomaly detection algorithms. In each scenario we perform trace analysis using different techniques.

In the first scenario we perform parallel trace analysis on a monitoring cluster using Pig Latin which is script language for generating mapreduce jobs. We used the LADT algorithm for parallel correlation analysis and we also compared to an updated implementation based on Apache Spark. In both cases we examined the behaviour of the LADT tool for different numbers of utilised cores. The Spark implementation outperformed our previous implementation based on Pig. Spark is utilising the concept of Resilient Distributed Dataset which takes advantage of in-memory data caching for iterative data analysis. This is the key to Spark's performance, as it allows applications to avoid costly disk accesses by utilising the concept of Resilient Distributed Datasets (RDD).

In the second scenario we examine a use case where CactoScale takes advantage of in situ analysis modules and uses the results in conjunction with the LADT parallel correlation analysis. This use case demonstrates how the log data analysis is pre-processed and the filtered results are stored for further analysis. In this way, the analysis takes place in a hierarchical manner and part of the workload (i.e. the text log processing) is off loaded in the computing nodes, while the correlation analysis is done in parallel on the monitoring cluster using apache Spark. The results of the analysis indicate that the processing of the data logs has only a small fixed overhead over a different number of cores. Taking advantage of the textual logs when detecting anomalies limits the number of false positive alerts.

In the third scenario we present a number of new in situ spike detection methods. We perform online workload spike detection using adaptive signal processing techniques. In this use case the anomaly detection algorithm functions locally on each cloud node and the results are collected to the monitoring database. The analysis load is distributed on the cloud platform and each node performs spike anomaly detection on itself. Unanticipated changes in workload characteristics can potentially lead to service slowdown and end in service-failure due to insufficient resource allocation. For this reason, having the ability to early detect the build up of workload spikes is beneficial for making proactive resource management decisions.

In this deliverable we demonstrated different data analysis tools that run in parallel using Apache Pig and Spark framework. We combined the parallel correlation analysis of performance metrics with filtered data logs from cloud data nodes to improve the accuracy of the analysis. In the spike detection scenario, we examined in-memory processing of data within compute nodes to improve the efficiency and scalability of the analysis. In the future we plan to examine the interference of the in situ trace data processing to actual workload. We will further explore scheduling and buffer allocation methods that isolate the resources used by the logging infrastructure from the resources used by actual workloads.

REFERENCES

[1] Joao Paulo, M., & Silva, L. (2010). Adaptive Profiling for Root-Cause Analysis of Performance Anomalies in Web-Based Applications. *10th IEEE International Symposium on Network Computing and Applications*, (pp. 171-178).

[2] Antunes, J., Neves, N., & Verissimo, P. (2008). Detection and Prediction of Resource-Exhaustion Vulnerabilities. *19th International Symposium on Software Reliability Engineering*, (pp. 87-96).

[3] Arlitt, M., & Jin, T. (2000). A workload characterization study of the 1998 world cup web site. *IEEE Network*, 30–37.

[4] Barbhuiya, S., Papazachos, Z., Kilpatric, P., & Nikolopoulos, D. (2015). A Lightweight Tool for Anomaly Detection in Cloud Data Centres. *5th International Conference on Cloud Computing and Services Science, CLOSER 2015.* Lisbon.

[5] Bodik, P., Fox, A., Franklin, M. J., Jordan, M. I., & Patterson, D. A. (2010). "Characterizing, modeling, and generating workload spikes for stateful services. *Symposium on Cloud Computing.* ACM.

[6] Bodik, P., Goldszmidt, M., Fox, A., Woodard, D. B., & Andersen, H. (2010). Fingerprinting the Datacenter: Automated Classification of Performance Crises. *Proceedings of the 5th European Conference on Computer Systems* (pp. 111-124). Paris, France: ACM.

[7] CACTOS Consortium. (2014). *D4.1 Data Collection Framework.*

[8] CACTOS Consortium. (2014). *D4.2 Preliminary offline trace analysis.*

[9] CACTOS Consortium. (2014). *D5.1 Model Integration Method and Supporting Tooling.*

[10] CACTOS Consortium. (2014). *D5.3 Operational Small-Scale Cloud Testbed Managed by the CACTOS Runtime Toolkit.* Retrieved from D5.3 Operational Small-Scale Cloud Testbed Managed by the CACTOS Runtime Toolkit.

[11] CACTOS Consortium. (2014). *D7.2.1 Physical Testbed.*

[12] CACTOS Consortium. (2015). *D5.4 Evaluation Methodology for the CACTOS Runtime and Prediction Toolkits .*

[13] CACTOS Consortium. (2015). *D7.4.1 Validation and Result Analysis.*

[14] CACTOS Consortium. (2016). *D5.5 Performance evaluation of the CACTOS toolkit on a small cloud testbed.*

[15] CACTOS Consortium. (2016). *D7.4.2 Validation and Result Analysis.*

[16] Caniff, A., Lu, L., Mi, N., Cherkasova, L., & Smirni, a. E. (2010). Fastrack for taming burstiness and saving power in multi-tiered systems. *ITC.*

[17] Casale, G., Kalbasi, A., Krishnamurthy, D., & Rolia., J. (2009). Automatic Stress testing of multi-tier systems by dynamic bottleneck switch generation. . *In Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware.* Springer-Verlag.

[18] Chandola , V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing surveys*.

[19] Consortium, C. (2015). *Evaluation Methodology for CACTOS Runtime and Prediction Toolkits.*

[20] Consortium, C. (n.d.). *D5.5 Performance evaluation of the CACTOS toolkit on a small cloud testbed.*

[21] Dahbur, K., Mohammad, B., & Tarakji, A. B. (2011). A Survey of Risks, Threats and Vulnerabilities in Cloud Computing. *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications* (pp. 12:1-12:6). Amman, Jordan: ACM.

[22] Dean, J., & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM, 51*(1), pp. 107-113.

[23] Downey, A. B., & Feitelson, D. G. (1999). The elusive goal of workload characterization. *SIGMETRICS Performance Evaluation Review*, 14-29.

[24] Eldin, A. A., Rezaie, A., Mehta, A., Razroev, S., Luna, S. S., Seleznjev, O., . . . Elmroth, E. (2014). How will your workload look like in 6 years? analyzing wikimedia's workload. *IEEE International Conference on Cloud Engineering.* IEEE.

[25] Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafaee, M., Jevdjic, D., . . . Falsafi, B. (2012). Clearing the clouds: a study of emerging scale-out workloads on modern hardware. *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems* (pp. 37-48). London, England, UK: ACM.

[26] Gunther, N. (2011). *Analyzing Computer System Performance with Perl::PDQ.* Springer.

[27] Gusella, R. (1991). Characterizing the variability of arrival processes with indexesof dispersion. *IEEE Journal on Selected Areas in Communication*, 203-211.

[28] Gustafsson, F. (2000). *Adaptive Filtering and Change Detection.* Wiley.

[29] Hadoop. (2015). *Apache Hadoop*. Retrieved 2015, from https://hadoop.apache.org
[30] Hansen, S. E., & Atkins, E. T. (1993). Automated System Monitoring and Notification With Swatch. *Proceedings of the 7th USENIX Conference on System Administration* (pp. 145--152). Monterey, California, USA: USENIX Association.
[31] Ibidunmoye, O., Hernandez-Rodriguez, F., & Elmroth, E. (2015). Performance Anomaly Detection and Bottleneck Identification. *ACM Computing Surveys*.
[32] Jiang, M., Munawar, M. A., Reidemeister, T., & Ward, P. A. (2009). System Monitoring with Metric-correlation Models: Problems and Solutions. *Proceedings of the 6th International Conference on Autonomic Computing* (pp. 13-22). Barcelona, Spain: ACM.
[33] Kang, H., Chen, H., & Jiang, G. (2010). PeerWatch: A Fault Detection and Diagnosis Tool for Virtualized Consolidation Systems. *Proceedings of the 7th International Conference on Autonomic Computing* (pp. 119-128). Washington, DC, USA: ACM.
[34] Kephart, J. O., & Chess, D. M. (2003). The Vision of Autonomic Computing. *Computer*, 41-50.
[35] Kumar, V., Cooper, B. F., Eisenhauer, G., & Schwan, K. (2007). iManage: Policy-driven Self-management for Enterprise-scale Systems. *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware* (pp. 287-307). Newport Beach, California: Springer-Verlag New York, Inc.
[36] Lassnig, M., Fahringer, T., Garonne, V., Molfetas, A., & Branco, M. (2010). Identification, modelling and prediction of non-periodic bursts in workloads. *CCGrid.* IEEEE.
[37] Li, D., Jin, H., Liao, X., Zhang, Y., & Zhou, B. (2013, March). Improving Disk I/O Performance in a Virtualized System. *J. Comput. Syst. Sci.*, 187-200.
[38] Lou, J.-G., Fu, Q., Yang, S., Xu, Y., & Li, J. (2010). Mining Invariants from Console Logs for System Problem Detection. *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference* (pp. 24-24). Boston, MA: USENIX Association.
[39] McHugh., A. (2013). *Top 10 Web Outages of 2013* . Retrieved from http://blog.smartbear.com/performance/top-10-web-outages-of-2013/
[40] Mehta, A., Durango, J., Tordsson, J., & Elmroth, E. (2015). Online Spike Detection in Cloud Workloads. *IEEE International Conference on Cloud Engineering (IC2E 2015),*, (pp. 446-451).
[41] Mi, N., Casale, G., Cherkasova, L., & Smirni., a. E. (2008). Burstiness in multi-tier applications: Symptoms, causes, and new models. *In Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware.* (pp. 265–286). Springer-Verlag.
[42] Olston, C., Reed, B., Srivastava, U., Kumar, R., & Tomkins, A. (2008). Pig Latin: A Not-so-foreign Language for Data Processing. *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data* (pp. 1099--1110). Vancouver, Canada: ACM.
[43] Oppenheimer, D., Ganapathi, A., & Patterson, D. A. (2003). Why Do Internet Services Fail, and What Can Be Done About It? *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems. 4*, pp. 1-1. Seattle, WA: USENIX Association.
[44] Ostberg, P.-O., Groenda, H., Wesner, S., Byrne, J., Nikolopoulos, D., Sheridan, C., . . . Pap. (2014). The CACTOS Vision of Context-Aware Cloud Topology Optimization and Simulation. *In Cloud Computing Technology and Science (CloudCom)* (pp. 26-31). Singapure: IEEE.
[45] Page, E. (1954). Continuous inspection schemes. *Biometrika*, 110-115.
[46] Pearson. (2015). Retrieved 2015, from http://www.statstutor.ac.uk/resources/uploaded/pearsons.pdf
[47] Pertet, S., & Narasimhan, P. (2005). *Causes of failure in web applications.* CMU-PDL-05-109.
[48] Pertet, S., & Narasimhan, P. (2005). *Causes of failure in web applications* . Carnegie Mellon University.
[49] Prewett, J. E. (2003). Analyzing cluster log files using Logsurfer. *In Proceedings of the 4th Annual Conference on Linux Clusters.*
[50] Rabkin, A., & Katz, R. (2010). Chukwa: A System for Reliable Large-scale Log Collection. *Proceedings of the 24th International Conference on Large Installation System Administration* (pp. 1-15). San Jose, CA: USENIX Association.
[51] Rajasekar, N. C., & Imafidon, C. (2010). Exploitation of Vulnerabilities in Cloud Storage. *Proceedings of the First International Conference on Cloud Computing, GRIDs, and Virtualization*, (pp. 122-127).
[52] Rouillard, J. P. (2004). Real-time Log File Analysis Using the Simple Event Correlator (SEC). *Proceedings of the 18th USENIX Conference on System Administration* (pp. 133-150). Atlanta, GA: USENIX Association.
[53] Sigar. (2015). Retrieved 2015, from https://support.hyperic.com/display/SIGAR/Home
[54] Spark. (2015). *Apache Spark*. Retrieved from https://spark.apache.org/
[55] Tan, J., Kavulya, S., Gandhi, R., & Narasimhan, P. (2012). Light-weight Black-box Failure Detection for Distributed Systems. *Proceedings of the 2012 Workshop on Management of Big Data Systems* (pp. 13-18). San Jose, California, USA: ACM.

[56] Virt-Top. (2015). Retrieved 2015, from http://virt-tools.org/about/

[57] Wald, A. (1945). Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 117-186.

[58] Wang, C. (2009). EbAT: Online Methods for Detecting Utility Cloud Anomalies. *Proceedings of the 6th Middleware Doctoral Symposium* (pp. 4:1-4:6). Urbana Champaign, Illinois: ACM.

[59] Ward, J. S., & Barker, A. (2013). Varanus: In Situ Monitoring for Large Scale Cloud Systems. *Proceedings of the 2013 IEEE International Conference on Cloud Computing Technology and Science. 02*, pp. 341-344. Washington, DC, USA: IEEE Computer Society.

[60] Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009). Detecting Large-scale System Problems by Mining Console Logs. *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles* (pp. 117-132). Big Sky, Montana, USA: ACM.

[61] *λ lambda-architecture*. (2015). Retrieved from http://lambda-architecture.net/