British Journal of Mathematics & Computer Science
18(3): 1-28, 2016, Article no.BJMCS.27872
ISSN: 2231-0851

SCIENCEDOMAIN *international*
www.sciencedomain.org

Identifying Most Relevant Performance Measures for Root Cause Analysis of Performance Degradation Events on a Private Cloud Computing Application: Experiment in an Industry Environment

A. Ravanello[1*], A. April[1], A. Gherbi[1], A. Abran[1], J. M. Desharnais[1] and A. Gawanmeh[2]

[1]Ecole de Technologie Superieure, 1100, Notre Dame St. W, Montreal, QC, CA H3C 1K3, Canada.
[2]Khalifa University, P.O.Box 573, Sharjah, UAE.

Authors' contributions

This work was carried out in collaboration between all authors. Author A. April designed the study and supervised the work. Authors A. Abran, A. Gherbi and JMD reviewed and provided advice on standards, edition and analysis, while author A. Gawanmeh provided the state of the art on methods used for performance measures on related systems. Author AR edited the manuscript and conducted the statistical experiments. All authors read and approved the final manuscript.

Article Information

DOI: 10.9734/BJMCS/2016/27872
Editor(s):
(1) Dariusz Jacek Jakóbczak, Chair of Computer Science and Management in this Department, Technical University of Koszalin, Poland.
(2) Tian-Xiao He, Department of Mathematics and Computer Science, Illinois Wesleyan University, USA.
Reviewers:
(1) Anonymous, University of Carthage, Tunisia.
(2) Ahmet Sayar, Kocaeli University, Turkey.
(3) Jose Ramon Coz Fernandez, University Complutense of Madrid, Spain.
(4) Kexin Zhao, University of Florida, USA.
(5) Ucuk Darusalam, Universitas Nasional, Indonesia.
Complete Peer review History: http://www.sciencedomain.org/review-history/15931

Original Research Article

Received: 23rd June 2016
Accepted: 16th August 2016
Published: 26th August 2016

Abstract

Cloud computing applications (CCA) are defined by their elasticity, on-demand provisioning and ability to address, cost-effectively, volatile workloads. These new cloud computing (CC) applications are being increasingly deployed by organizations but without a means of managing their performance proactively. While CCA provide advantages and disadvantages over traditional client-server applications, their unreliable application performance due to the intricacy and the high number of multi connected moving parts of its underlying infrastructure, has become a major challenge for software engineers and system

**Corresponding author: E-mail: ravanello@gmail.com;*

administrators. For example, capturing how the end-users perceive the application performance as they complete their daily tasks has not been addressed satisfactorily. One possible approach for identifying the most relevant performance measures for Root Cause Analysis (RCA) of performance degradation events on CCA, from an end-user perspective, is to leverage the information captured in performance logs, a source of data that is widely available in today's datacenters, and where detailed records of resource consumption and performance logs is captured from numerous systems, servers and network components used by the CCA. This paper builds on a model proposed for measuring CC application performance and extends it with the addition of the end-user perspective, exploring how it can be used in identifying root causes (RC) for performance degradation events in a large-scale industrial scenario. The experimentation required adjustments to the original proposal in order to determine, with the help of a multivariate statistical technique, the performance of a CCA from the perspective of an end-user. An experiment with a corporate email CCA is also presented and illustrates how the performance model can identify most relevant performance measures and help predict future performance issues.

Keywords: Cloud computing; ISO 25010; holt-winters; performance; root-cause analysis.

1 Introduction

As Cloud Computing is more largely adopted by organizations, the measurement of the performance of this infrastructure becomes a major concern [1]. The utilization of Cloud Computing Applications (CCA) can be justified by financial reasons [2], even as this type of application contains more components and is geographically displaced than the standard client-server applications [3]. It is important to introduce the underlying concepts of CC as this experiment aims to address one of its performance characteristics.

One of the frequently cited sources for the definition of CC is the one by the US National Institute of Standards and Technology (NIST), that proposes that "Cloud computing (CC) is a model for enabling convenient, on demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [4].

From this definition CC is a technology that relies on the connectivity provided mainly by the Internet to allow access to shared pools of resources, whose utilization should be easily obtained and relinquished with good cost-effectiveness and without much administrative effort; these shared resources would permit a high degree of elasticity based on variable workloads and could be managed by a service level agreement (SLA) describing the expected behavior and performance of specific CCA. The notion of "Quality in Use" of a CCA is directly linked to the quality of the network infrastructure as well as the configuration of the pooled resources [5].

A number of authors cite the advantages [2] and disadvantages [1,3,6] of utilizing CC technology. It is important to identify that CC is not a solution for all the computing problems that exist and that it can be misused in lieu of other more adequate technologies. From the standard NIST definition, [4] it is possible to see that a CC relies on a number of different components, which can be configured, according to the needs of the users. This also means that the concept presupposes that the configuration of a cloud isn't static, meaning that at a certain moment a particular component might be hosting a different number of services and customers, depending not just on the requested workload by the end-users, but also on the availability of remaining resources, on the infrastructure, to host the said requests. In this case, when a performance issue happens, it is hard to pinpoint the exact point of degradation.

This research aims to address one of the cloud disadvantages that emerge from this greater number of inter-related components: The unreliability of CCA performance due to the intricacy of the multi connected dynamic parts of its underlying infrastructure. Since many nodes interact to process the data, it is hard to pinpoint performance issues. This research explores that characteristic with the investigation of the most

relevant performance measures for root cause analysis (RCA) of performance degradation events in the experiment.

Performance measurement of information systems is challenging: ensuring quality of information systems has long been a concern both for organizations and software engineers. Juran [7] had already identified in the 1970's that measuring the quality of software, systems and Information Technology (IT) services is challenging, in part caused by the immaturity of software engineering itself and by organizations seldom able to follow up with the rapidly evolving technologies [8].

Software measurement can be conducted from different perspectives: the internal perspective, which measures the quality of how well built and maintainable the software system is; and the external perspective that is interested in how well the software system fits its functional obligations and technological environment, and its quality when used. From the external perspective, performance log measures try to reflect the actual utilization of the system by its end-users, who are one of the stakeholders of the software and are using it to perform a task for achieving a particular business goal [5]. These software product measurement perspectives (i.e. internal and external) are documented in the ISO 25023 standard.

Measuring the performance as perceived by an end-user has been a concern for software engineering researchers since the early 60's [8] and some experiments have explored various proposals [9-16]. Initially, researchers have used end-user's surveys since it is easier than trying to actually measure the end-user's perceived performance using internal and external measures in a distributed computing environment such as a CCA environment characterized by the intricacy of the multi connected dynamic parts of its underlying infrastructure. However, surveys have a number of limitations, including not being good for following trends in real time, not providing a good source for cause and effect, and low response rates, vulnerability to responder's bias and being generally time-consuming efforts [17]. To overcome these limitations and complement survey data, it is possible to use automated and user-independent performance quality measurements as proposed over the years.

The literature describes that software systems performance measurement can be conducted in many ways. One popular approach is to use the datacenter logs [9]: An IT infrastructure typically offers readily available logs of different resource consumption and utilization levels for operating systems (OS), applications, computers and IT infrastructure components like telecommunication devices. Logs are often composed of binary files that include data from different components in a system, containing large quantities of detailed information and are typically stored in a file or database for later analysis.

Recently, many commercial, open source, and easily accessible log tools have been made available for collecting, analyzing and generating performance dashboards that render different measures of the infrastructure components that are used by an application [10-13] suggesting that these measures can be utilized to identify performance degradation events and to ascertain which are the potential root causes (RC) for these events.

At a high level, Croll [14] suggests that cloud performance should be approached from a business perspective initially, analyzing the positive and detrimental effects that it eventually brings to the company's objectives [14]. Others have proposed automated software tools to simulate access to services and to measure:

1. The response times [15];
2. The third-party performance evaluation services [16,17] and
3. Comparative tests amongst different providers [18].

In contrast, a number of other approaches propose the collection of internally derived measures of different service configurations over the same infrastructure [19], but do not provide details on how this can actually be done within an industry context. However, reporting performance from the end-user perspective has not been undertaken in previous works. To tackle this as unobtrusively as possible, we choose to monitor

performance logs and use the data collected to identify degraded or appropriate quality levels for an industry-based CCA.

The research reported here investigates how this data can be analyzed and interpreted, including the measurement results on the organizational goals, and in particular the performance as perceived by end-users. This work extends the Performance Measurement Framework for Cloud Computing Applications proposed by Bautista et al. [5] (Bautista's Framework), which is based on the COSMIC model, to support generic software as illustrated in Fig. 2.

This paper applies the measurement framework to represent the performance of an actual CCA from the end-user perspective and experiments on how to identify the most relevant performance measures for identifying RC for performance degradation on Cloud Computing Systems (CCS).

This paper is structured as follows. Section 1 introduces the research issue and the research environment. Section 2 presents the related works, and Section 3, the methodology. Section 4 reports on the experiment for collecting, analyzing and correlating the collected performance data as well as the data analysis. Section 5 presents the conclusion and future work.

2 Related Works

This section presents the related work approaches to measuring system performance, end-user performance measurement and the performance measurement framework for CCA, highlighting the elements relevant to this research.

2.1 Approaches to measuring system performance

The problem of measuring system performance has been explored by numerous authors and many tools are now available for collecting, measuring and displaying the values of performance log data [20], such as: memory use, processor use and at times telecom and disk usage.

Direct measures are extracted straight from the data-source and are usually self-sufficient to represent any given state, such as processor utilization. Indirect measures are measures that are obtained from combining, deriving and calculating other measures: for example, number of completed jobs per second.

Different data collection approaches are implemented in these tools: Some install agents on the equipment that report the log data back into a performance management database [21,22]; others use a popular internet standard protocol, SNMP[1] [11,23], that collects the data directly. Other tools store the results of the measurements locally on performance logs [10,12,13]. These results are typically used for monitoring purposes or stored and processed for posterior use.

The data processed as they are collected usually have some level of pre-defined thresholds that, when exceeded, trigger an action, such as restarting a service, rebooting a computer or generating an alarm. The data processed or analyzed *a posteriori* (secondary data) are frequently used for troubleshooting, root cause analysis, management and decision making purposes. Some of these data collection tools are capable of doing a first level of processing (i.e. alerts, restarts and reboots) as well as storing the data for further analysis [24,25].

[1] *SNMP is a "Internet-standard protocol for managing devices on the first approach offers a IP networks".* [11]

Two main concerns have been documented on the current measurement collection approaches when the processing is performed at the moment of the data collection:

1) Is the actual processing affecting the collected measures? More specifically, is the measuring process intruding on the measured system?
2) When the processing is done *a posteriori*, is there a risk of impacting the data quality and, consequently, not being able to take decisions in a timely fashion? [26-28].

CC increases the difficulty of understanding the behaviour of the service delivery infrastructure. It is harder to identify what are the paths that the service follows inside a CCS infrastructure in order to reach the client and, accordingly, it is harder to identify clearly each point of failure. Additionally, defining quality characteristics and effectively managing such infrastructures is hard and its measurement process involves many steps and both computational and human effort to complete.

To study CCS performance, two approaches from the literature are promising for this research:

- Design and set up a set of experiments that deploy a customized application on different commercial Cloud Computing Platforms (CCP), thus measuring the performance of the infrastructure [29].
- Use stochastic methods to simulate the user interaction with a customized system, comparing the effects of network I/O on user response [30].

The first approach offers a set of equations that allow the comparison of different systems. On the other hand, the second approach demonstrates the cause-effect of user interactions with a system. The following limitations have already been identified by the authors of both papers [29,30]:

- Customizing an application is not always feasible within a CCS context [29], considering that many applications are constructed by third parties and integrated into a CCS.
- Stochastic simulations are, by definition, non-generalizable [30], given that the components that compose that particular stochastic scenario are not necessarily translatable to other experiments.

For measuring CCS performance, there should be an alternative that offers a fair compromise of time to decision and generalization capacity.

2.2 End-user performance measurement

The concern with the user's ability to efficiently interact with computer systems has long been raised. For instance [31], in the 60's suggests that different users can have different backgrounds and abilities for exploring different degrees of information and, even if the application user's interfaces can be standardized, the same is not guaranteed for user behaviour.

During the 80's and the 90's, researchers [32-34] discussed that end-user perceived performance of a software based system is a complex attitudinal construct, involving four key concepts describing what would be a software based system that displays good quality from their perspective:

1. It should improve the average quality of the user's daily work;
2. It should make the user's job easier;
3. It should save the user's time and effort; and
4. It should fulfill the needs and requirements of the user's tasks.

Fagan and Neil [35] discussed the potential impact of self-efficacy, including to fully exploit a technology. They report the impact of user anxiety, experience, IT support and utilization of computer systems on the end-user experience. One of their key findings is that, whenever a software-based system was described by an end-user as having a good technical response (for example, *saving the user's time and fulfilling the needs and requirements of the user's tasks*), the actual end-user performance was also reported as good.

The individuality of the performance perception and user experience has been explored in [36], where the authors reported that different individuals (i.e. end-users) would display variable responses to fluctuating user experience when using software-based systems. The perceived usefulness and task-related expectation of a system has also been positively related to the end-user satisfaction [37]. Five performance characteristics are also proposed as relevant for a typical end-user: 1) *task success*; 2) *time-on-task*; 3) *errors; 4) efficiency;* and 5) *learnability* [38]. These research results reinforce the concept that a typical end-user wants to be able to perform the required tasks in a timely fashion, with the least possible amount of effort.

Two studies have also reported that:

1) When an end-user *feels* prepared for the task to be performed with the utilization of a specific software-based system, that user will frequently present a greater degree of satisfaction and tolerance to failure whilst engaging the task [39];
2) The application delivery chain, which is "the sum of all interconnected components that are involved in making an application available to the end-user" is getting increasingly more complex every year (i.e. this makes "seeing the big picture", understanding all its elements and the origin of problems, a harder task for the datacenter technical analysts [40]).

From these related works, a number of findings are useful to guide the development of this research:

- Understanding cloud computing performance is a technically challenging task, and being able to identify, comprehend and act upon a performance-degrading event on a timely fashion is difficult.
- Users are able to identify degraded performance of systems by interacting with them. System analysts do not usually have the same interaction with systems and, without the aid of measures and indicators, it might be impossible to remediate performance degradations without impacting the end-user performance as well.
- Monitoring cloud computing performance using datacenter logs involves interpreting a large amount of data: to be able to provide useful information for timely decisions, an automated approach for the data collection, processing and presentation is necessary.

2.3 Performance measurement framework for cloud computing

This research leverages the performance measurement model proposed by Bautista et al. [5] (Bautista's Model), which is based on the COSMIC model for generic software [41], represented in Fig. 1. Bautista's model is represented in Fig. 2.

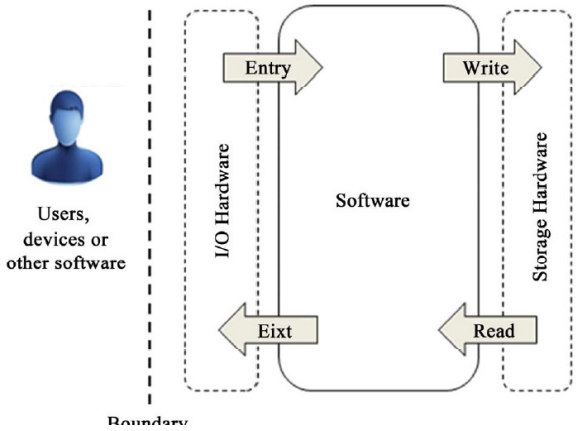

Fig. 1. COSMIC model of generic software (adapted from [42])
Legend: I/O: Input/Output

The COSMIC generic software model presented in Fig. 1 also enables the connection between the CCA – the application, with the cloud computing system (CCS). The CCA is the software layer which consumes resources from the I/O and Storage Hardware in order to provide services to the users, devices and other software. The CCS is the combination of the hardware, software, telecommunications equipment and its specific configurations that allow a CCA to perform the requested services.

It describes the cloud computing system (CCS) and its detailed attributes that can determine the system's performance, specifically quantifying the efficiency and reliability concepts, through measurement functions that assign values to the properties, thus allowing actual measurement and evaluation. This framework proposes to use and share different performance log measures across different concepts, using a mechanism of entries and exits as communication channels. Performance log measures are obtained as the result of a measurement method, and are analysed by analysis functions that read the data on a storage system.

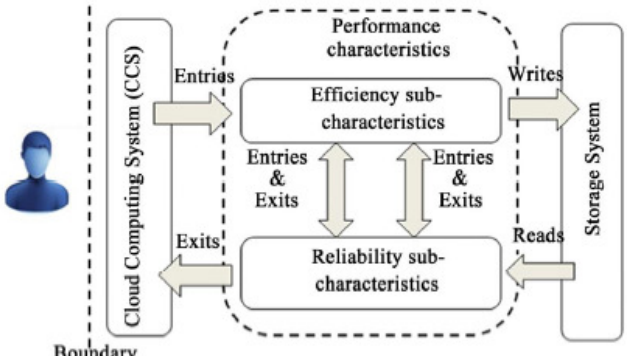

Fig. 2. Generic performance measurement framework - Bautista et al. [5]

This proposed model considers that three fundamental outcomes are generated by a software-based system whenever a request is made: it either a) performs the service correctly, b) incorrectly or c) does not perform at all. If the system performs correctly, it is feasible to measure its response speed. If the system performs incorrectly, it is feasible to measure its system's reliability. If the system does not perform, it is feasible to identify a system's availability issue. These outcomes are represented in Fig. 3.

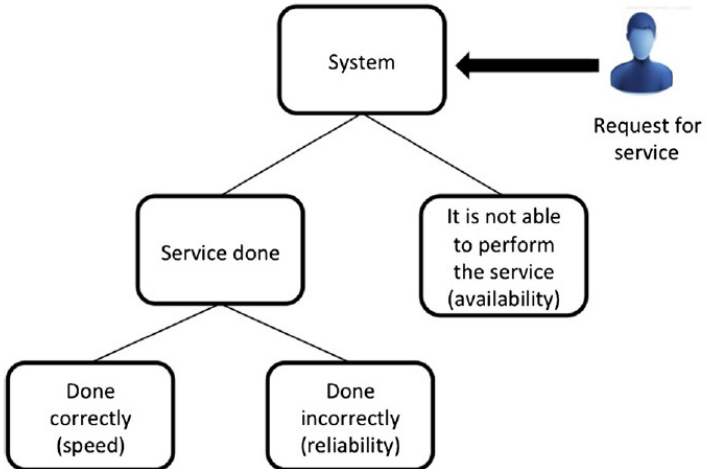

Fig. 3. Bautista et al. [5] model: The 3 feasible outcomes for a system request

These three types of outcomes have been mapped to the definition of quality as described in both use model and the ISO software product quality model in ISO 25010. The related performance sub-concepts identified in this mapping are: efficiency (i.e. time behavior, resource utilization, capacity) and reliability (i.e. maturity, availability, fault tolerance, recoverability) as the key determinants of the performance for a CCA.

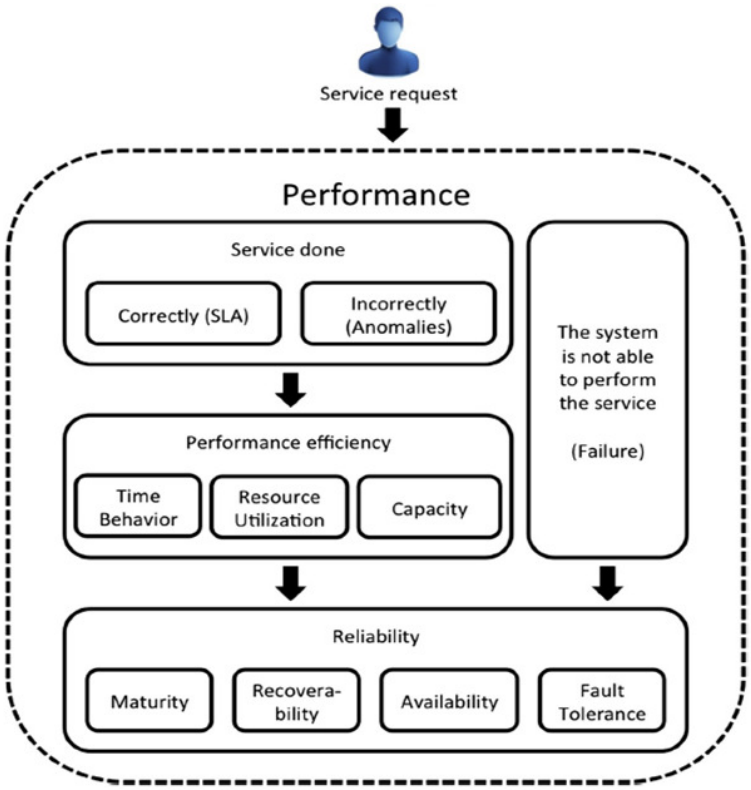

Fig. 4. Bautista model: Performance concepts and sub concepts relationships [5]

Bautista et al. [5] describe performance as a service that either completes or not a request, and attains different degrees of efficiency and reliability while doing it – see Fig. 4.

3 Methodology

The experiment conducted in this work involved 700 CCS components within this corporate email CCA, located in a single physical location, and collecting anywhere from 80 to 570 different performance data sources, per component, at a specific time. The total size of data collected is 2.4 GB, with approximately 500.000 rows and 483.000.000 cells. These CCS components are disposed in a topology (see Fig. 5) composed of desktops, servers and network components producing the log data listed in Tables 1 and 2. The processing has been executed by 5 virtual machines (VMs) where one is configured as a host and four are configured as slave nodes running the Anaconda – Python for Hadoop – package through which the experimental data is collected, organized and processed.

From this high-level network diagram, it is possible to see that between the VM-Host, which is the node where the end-user interaction happens and the data that is stored in the CCA database, represented by the Mbx/Db CCS component, at least 14 different CCS components can influence the end-user perceived performance at any time. Each of the individual CCS components can generate a number of measures

depending on its specific characteristics. Sub-sections 3.1-3.4 provide further information on how the performance characteristics are associated, organized and extracted.

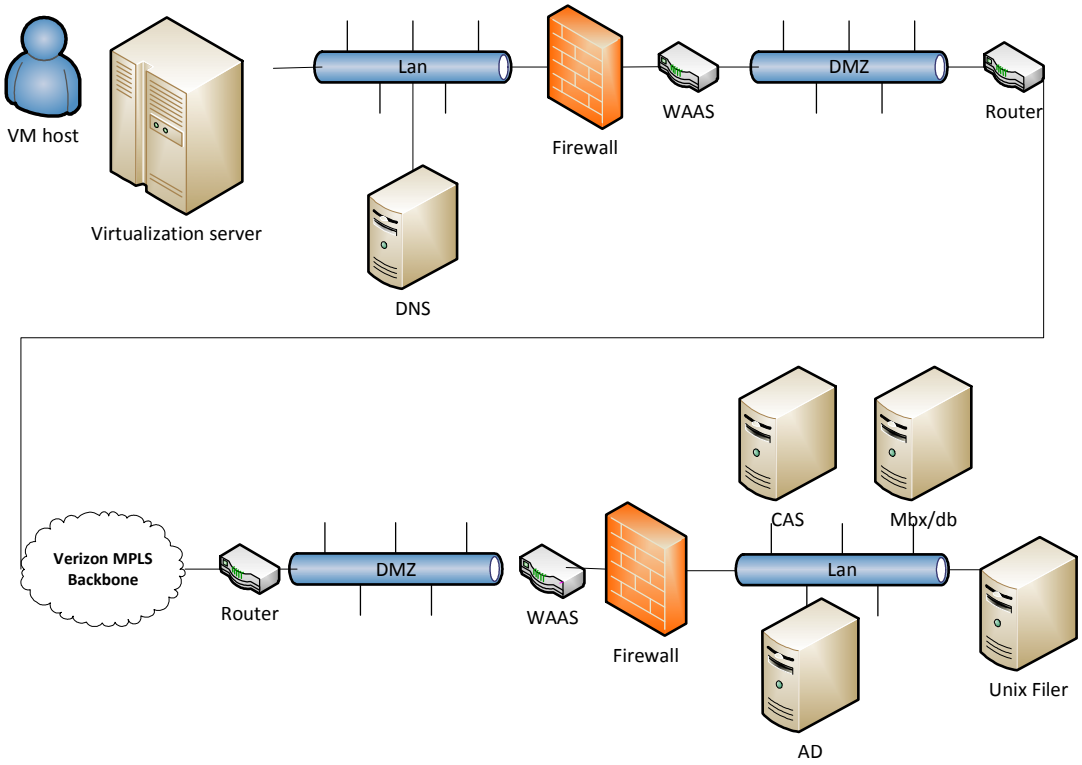

Fig. 5. High-level network topology of a CCS

During the experiment, 2.4 GB of data is collected from 34 nodes on a single physical location, as per the following steps:

1. Collect real life data from the cloud based e-mail application.
2. Identify the most relevant performance measures for RCA of each time series.
3. Explore any potential anomalies in the measures identified
4. Describe intra-component and trans-component correlations.
5. Provide any additional information that can guide decisions on the situation of the end-user perceived performance at any given time.

3.1 Description of the industry case study

The environment of this experiment is defined by a hybrid cloud which provides the end-users with e-mail functionality. The cloud hosts databases (Microsoft SQL Server), e-mail (Microsoft Exchange), directory services (Microsoft Active Directory), DNS servers (Red Hat Enterprise Linux Bind), virtualization servers (Citrix XenDesktop); the clients run Microsoft Outlook on Microsoft Windows 7 machines.

The Network infrastructure is provided by third parties, and is mainly comprised of Cisco switches, routers and firewall appliances. High availability is attained by load balancing and redundancy of resources. All the performance and application logs are centralized in a logging service, in order to provide accountability and compliance to the particular industry regulators.

The total amount of machines on the network is approximately 120.000, and the experiment targeted 700 at a single physical location. These 700 machines share the same networking components, while the cloud infrastructure is shared by all users.

The experiment collected the performance logs generated by the 700 machines of the targeted physical location, as well as those from the cloud infrastructure; those logs are already collected as part of incident management approaches and the data used is a secondary copy of the original one. The volume of processed data was very large, and the processing was completed on an Apache-Spark machine cluster of 2 master and 4 slave nodes. The respective configurations of these machines are Xeon E31230 3.2 GHz (8 cores) 16 GB of ram and each VM running on 4 virtual cores with 8 GB of ram. The OS used was Red Hat Enterprise Linux 64bit. The language selected for the creation of the processing algorithms is Python, mainly due to the convenience of using this language for statistical calculation with the Pandas module.

This environment is a live production setting, and the hardware used for the processing is part of an innovation lab that is used to test and develop new services.

3.2 Relationship between performance measures for CCA, platform and software engineering concepts

Mapping the performance measures from the CC platform into the quality concepts of the Bautista's model is a needed step in order to establish a relationship between performance data and ISO quality concepts. This data is typically collected using the performance logs. In this list we are trying to have the most granular possible performance data provided by the log tools. Table 1 presents an excerpt of the full list of performance data and from which CCS component types the data are collected; Appendix1 presents the complete list.

It is important to notice that while there are only 57 types of log data in the table presented in Appendix 1, 24 of them have a name containing "*": this means that this particular data source is applied to multiple disks, processors, processes, network interfaces and will fluctuate depending on the activity level at any given moment. An expanded description of each measure can be found in the documentation of the original sources, as created by the owner of the appropriate systems [10,12,13].

Table 1. Excerpt of the data collected

Performance log data measure name	CCS component type
\LogicalDisk(*)\Free Megabytes	Client, Server
\Netlogon(*)\Average Semaphore Hold Time	Server
\Memory\Page Faults/sec	Client, server
\Memory\Available Bytes	Client, server, network
\Memory\Pages/sec	Client, server
\Paging File(*)\% Usage	Client, server
\System\File Read Bytes/sec	Client, server
\System\File Write Bytes/sec	Client, server
\System\System Up Time	Client, server
\System\Processor Queue Length	Client, server

The data presented in Table 2 describes the association of the above measures with the quality concepts of Bautista's Model for efficiency (i.e. time behavior, resource utilization, capacity) and for reliability (i.e. maturity, availability, fault tolerance, recoverability). It is possible to identify some imbalances in the quantity of performance log data types associated with each concept, similar to what has been already reported by Bautista et al. This could lead to a discussion on how to effectively design a CCA, which is not within the scope of the research reported here. The complete list of associations is also part of Appendix 1.

Table 2. Association of performance log data with PMFCCA quality sub concepts (Excerpt)

Performance log data measure name	ISO 25000 quality concept
\LogicalDisk(*)\Free Megabytes	Capacity
\Netlogon(*)\Average Semaphore Hold Time	Maturity
\Memory\Page Faults/sec	Maturity
\Memory\Available Bytes	Capacity
\Memory\Pages/sec	Time behavior
\Paging File(*)\% Usage	Time behavior
\System\File Read Bytes/sec	Resource utilization
\System\File Write Bytes/sec	Resource utilization
\System\System Up Time	Availability
\System\Processor Queue Length	Time behavior

The asterisks in Table 2 refer to the same meaning as in Appendix 1 and Table 1. It may be the case where multiple processes or service interactions span different instances of counters, each collecting data for a particular process or service. In this case, multiple counters for the same processes are named with an # and a number, according to the order under which each instance of the same process is invoked. For example, if a CCS component, such as an end-user desktop computer, has two (2) Internet Explorer applications running, the performance counters would be \Process(Iexplore)\% Processor Time and \Process(Iexplore#1)\% Processor Time.

3.3 Principal component extraction based on variance and kurtosis

The original research and experimentation conducted by Bautista uses a combination of particular statistical methods in order to obtain a list of the most relevant measures to be analyzed. In the next iteration of this research, a different approach is implemented, which is the time series analysis: the data sets of values are considered not as a matrix of self-contained values but as an interactive result considering previous values. In this type of data, the selection of random samples from the data contained in the matrix of performance log values only makes sense if random times, containing all the types of data, can be selected. In the experimental setup reported in Section 3, the performance log file is considered as potentially endless, with each of the lines of the file being a new interaction of the system. This requires a more proactive and online process where each of the features, which will be stored in the columns in the files, consists of a string of multiplexing values that can disturb other values within the same log column – vertically, in a matrix perspective - and also across multiple measures in the same time interval– horizontally, again, using the matrix.

An example of such interactions would be "\Paging File(*)\% Usage". This performance log measure describes how much of the page file is being used by the local operational system. In practice, it means that a) there is a need for paging (e.g., the main memory is exhausted) and b) the act of reading this file is causing I/O reads and writes on the local disk. Ultimately, the higher the utilization, the more indication will be given that there is a significant time degradation on the task. This time degradation is being caused by disk I/O that are slower than memory I/O.

As the counter collects data, the time series will show a "history", the values within itself representing different relevancies as the time goes by. For example, a sequence of values, such as: 5, 6, 50, 100 would suggest that the performance is degrading, whereas the same values, in the reverse order: 100, 50, 6, 5 would represent an improvement in the time behavior. Intrinsically, each of these values would have an impact on the measures collected for the same sampling according to its values. The "100" value should also contain more Disk I/O Read/Write, as long as the values of multiple features had been sampled on the same time axis.

In the end-user performance theory described by Davis [32], software-based systems have to help the users to perform the required tasks in a timely fashion. We consider that the linearly-performing features, the ones

with more symmetry and reduced variance, would induce the end-user to expect a standardized response. Consequently, disturbances to the end-user perspective can be identified by the peaks or asymmetries in the resulting time series.

Considering these effects, it is possible, using the end-user performance theory and Independent Component Analysis (ICA) [43], to justify choosing both Kurtosis and Variance as initial approaches for feature selection. These symmetry and variability fit the description of ICA: in neural networks, amongst many other disciplines, the identification of the multivariate data requires a large amount of calculations. ICA has been recently developed as a way to represent the non-Gaussian data in terms of independent, or as independent as possible, components. The variance and kurtosis analysis is a preliminary approach for this technique and leaves room for further improvements to the initial technique.

3.4 Detecting anomalies on seasonal and time series

Considering that one of the concerns of this research is how the end-user perceived performance of CCA can be modeled in a way that timely analysis can be enacted upon the data, it is important to describe an approach through which "timely analysis" can be done. The objective is to provide, as early as possible, some kind of information to a datacenter analyst that would help in identifying issues potentially affecting the end-user performance.

One of the key differences of this research from [5] is the analysis of the view of performance data as a time series: a time series has some well-known properties such as seasonality, momentum, interval and unicity of the data points [44]. The idea of interpreting the data as a time series also allows the possibility of forecasting, cycling and trending.

The concept invoked here, though, is that of the anomaly detection. An anomaly is any feature value that is positioned outside of an expected model. In this research, the Holt-Winters technique of exponential smoothing [45] is used on the code in Algorithm 2 presented in Section 3.6. This statistical method was selected because it is one of the simplest forms of exponential smoothing and thus serves as a starting point for approaching the problem of anomaly detection.

The algorithms for Holt-Winters exponential smoothing are available in multiple languages such as R and Python. This algorithm in particular has 2 important drawbacks: first, it tends to produce lagging results, meaning that burst-type variations take at least one full cycle interaction to compute. Additionally, seasonal data is hard to be taken into account unless the moving average windows can be fixed around the seasonality. This leads to a poor forecasting capability for this particular method, as exposed in the conclusions as an opportunity for further research.

4 Results and Discussion

The experiment has used Bautista's model [5] in a real world setting: A widely used corporate e-mail CCA. This section presents the setup of the experimental environment, data preparation, feature extraction, correlation, anomaly detection, example of correlations and an indicator.

4.1 Setup

To activate the log collection, each different operational system (either a desktop OS's, servers or network component) has to be configured to generate these logs. This can be achieved in different ways depending on the CCS. For the experimental setup, we chose a centralized Hadoop Distributed File System (HDFS) configured to be the main data collector.

As described in Section 3, for the definition of the scope of the experiment, we must be able to collect the data from a real life application. This can be performed by leveraging the OOZIE Coordinator which

implements workflow execution on multiple triggers. Algorithm 1 demonstrates, in pseudocode, the high level configuration for the OOZIE Coordinator for this task.

Algorithm 1 – Oozie coordinator Algorithm
For (1){
    If (file exists, extension=csv) Run (http://url:9000/perfman/organization);
}

This configuration will invoke the organization algorithm when a new file, having a .csv extension, is added on /perfman/. The workflow of the processes is shown in Fig. 6 and the relevant algorithms are discussed in section 4.3 and 4.5.

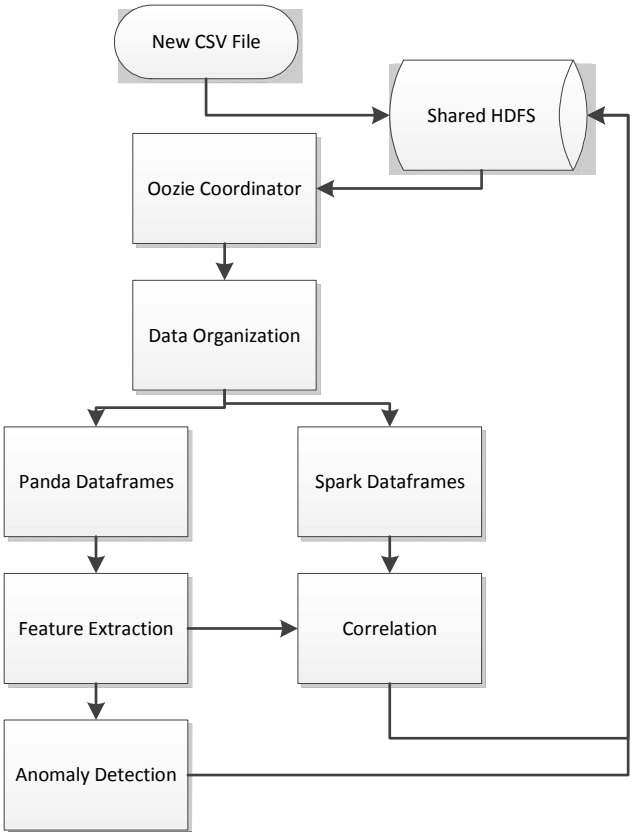

Fig. 6. Experiment components and relationships

Fig. 6 presents the experiment components and relationships as a sequence of executions of the algorithms to produce the desired outcomes. It also shows that not all operations could be performed using the Spark dataframes and had to be done using Pandas dataframes because of Spark limitations for calculating the kurtosis.

4.2 Data preparation

Since the volume of data will be very large (see Section 4.1), an approach for processing very large amounts of data in real-time is through the use of the recent Big Data technology Apache Spark [46], which is both faster than Hadoop and simpler to learn and program. Spark supports APIs for Scala, Java and Python. This chosen algorithmic approach allows processing a very large amount of log data, coming from multiple CCS

components, within decision time, including being able to analyze anomalies and to describe potential sources of problems as early as possible to act upon them.

Three data preparation activities have been carried out:

1. Data organization: the files are loaded in HDFS and shared by the individual nodes as they are created. When a new file is detected, the workflow coordinator invokes the organization algorithm, which scans for new files and loads them in memory. This starts the data cleanup step.
2. Data cleanup: two main processes happen: conversion to percentiles, so that all measures can be plotted in the same space, and conversion of qualitative measures characterized as H according to Table 3, so that all measures conform to the "lowest is better" qualitative approach. Some dataframes might contain null values: these must also be cleaned up. Once completed, the dataframe segregation algorithm is invoked.
3. Dataframe segregation: conversion of the .csv data into either a Pandas or a Spark dataframe. Pandas dataframes are used for the calculation of kurtosis, variance and anomaly detection, and the Spark dataframes are used for the vertical correlation. The dataframes also do not necessarily have the same format (i.e. meaning, the number of rows and columns) as it depends on the ability of a particular node to log information. This activity returns the Pandas and Spark dataframes to the correlator function for the feature extraction, described next.

4.3 Feature extraction

The feature extraction activity, represented as an intermediary step in Fig. 5, is necessary in order to provide the 30 most significant features that are extracted via the kurtosis and the variance analysis of each performance data source, identifying, for the datacenter analyst, the elements in which the highest peaks and variance are located.

This process reduces the dataframes to a more manageable size, with the fixed width of the 30 most relevant columns. These reduced dataframes are part of the desired outcomes, as they contain the most variant performance log measures which could potentially represent the measures that better explain any degraded performance and are passed into the correlator function, described in 4.5. Algorithm 2 contains the pseudocode for the features extraction.

Algorithm 2 – Feature extraction via Variance and Kurtosis analysis.

```
for column in df:
temp = (df[column].copy()
mean = temp.mean()
if mean is not None and mean!= 0:
kurt = stats.kurtosis(temp)
results[column] = [kurt, temp.var()/mean]
Transpose results by Kurtosis, Variance/Mean
```

The initial approach of the 30 most relevant performance log measures follows the findings of previous research where 38 features explained most of the variance of performance log data samples [47]. In this particular experiment, the 30 features contained at least 98% of the cumulative variance of the data set. This algorithm also has a dependency on the Pandas dataframes, because the kurtosis and variances formulae are not implemented on Spark as of the writing of this paper. It also means that this particular code cannot be distributed to the child nodes, which reduces the processing turnaround.

4.4 Correlation analysis

The correlation analysis activity of the experiment allows the datacenter analyst to identify candidate relationships between different performance measures across the CCS components or internally within a particular component. Although not inferring causality, it sheds light on similarity, patterns and equivalences

between the values of the performance log measures. The correlation analysis of the most relevant performance measures is executed in two ways: intra-component and trans-component:

4.4.1 Intra-component performance data correlation

On a performance log following the .csv format, the performance measures are displayed each in a column. When correlating performance measures on the same time-frame, these performance measures are located side-to-side, referred to here as an intra-component correlation.

4.4.2 Trans-component performance data correlation

This activity is akin to the intra-component correlation, only that, in this case, the columns with the same name are correlated across all the nodes that contain that particular performance log measure. Again, not inferring causality, it is possible to recognize the similarity of loads, resource consumption and even widespread issues which affect more than one node. The computations necessary in order to calculate the trans-node correlations can be distributed to the Spark cluster and are significantly faster when running in multi-node tenancy than in single node.

4.4.3 Intra-node correlations

Intra-node correlations will clarify the relationships between performance measures within the same CCS component, thus helping to identify issues that affect a single component.

4.4.4 Trans-node correlations

Trans-node correlations would assist in investigating issues that involve the same performance measure but affect multiple nodes.

4.5 Anomaly detection

The anomaly detection implementation has the potential for providing useful information for the datacenter analyst with regards to providing solutions to performance and resource utilization incidents. The anomalies are calculated by the Holt-Winters (second order) algorithm, which considers that the most recent values have a larger weight on the current measures than more distant ones (i.e. older events): empirical observations of datacenter analysts often assume that a recent fluctuation in processor utilization would impact end-user performance more – at that moment – than more distant fluctuations.

This implementation uses an anomaly ratio of 25% above the predicted level, as well as a time span of 100 observations. As for any autoregressive method, the longer the data is collected, the greater its forecasting precision will be [48]. Algorithm 3 presents the pseudocode for its implementation.

Algorithm 3: Anomaly Detection employing Holt-Winters second-order algorithm

```
alpha = 2.0 / (1 + span)
s = np.zeros((N, ))
b = np.zeros((N, ))
s[0] = arr[0]
for i in range(1, N):
s[i] = alpha * arr[i] + (1 - alpha) * (s[i - 1] + b[i - 1])
b[i] = beta * (s[i] - s[i-1]) + (1 - beta) * b[i-1]
return s
hw = _holt_winters_second_order_ewma(series_clean, 2, 0.5)
ratio_series = series_clean.apply(_lambda_get_ratio, args=(hw, cpt))
return ratio_series[ratio_series > _ANOMALY_MAX_RATIO]
```

This algorithm reads the list of performance logs and uses the exponential smoothing technique to identify the candidate anomalies, marking that particular feature, in time, with an X. This could help the datacenter analyst in identifying potential sources for issues on the time series that need to be focused upon.

4.6 Application of the model

In this activity the measures emerging from the extracted features are mapped back to the concepts described in Table 2. This mapping helps in interpreting the results where the results follow a textual presentation such as:

Anomaly detected (Time X, Measurement Name Y, Performance concept Z)

The Bautista performance model includes a set of formulae for representing the performance concepts presented in Table 2 and fully listed in Appendix 1. Table 3 shows an excerpt of the extensive list present in the same appendix of all the adapted formulae, following the same philosophy. In the next section these formulae are discussed in more detail. Since this research focuses on the performance measures obtained from different components and OS's, considering that the collected measures are all quantitative values, the formulae are qualitative evaluations of the values of these measures on an ordinal scale. In Table 3, L means "lowest is better" – e.g., a value closer to zero represents a better performance to the end-user, whereas H means that the higher the value, the best will be the performance to the end-user.

Table 3. Excerpt of collected measures and qualitative evaluations

Collected measures	Evaluation scale
\LogicalDisk(*)\Free Megabytes	H
\Netlogon(*)\Average Semaphore Hold Time	L
\Memory\Page Faults/sec	L
\Memory\Available Bytes	H
\Memory\Pages/sec	L
\Paging File(*)\% Usage	L
\System\File Read Bytes/sec	L
\System\File Write Bytes/sec	L
\System\System Up Time	H
\System\Processor Queue Length	L

Only 8 performance log measures were evaluated within the H scale category, whereas 49 others were evaluated within the L category. This research has no control on this characteristic of the log measures but a search for a greater balance between the 2 categories could be explored by the log tool developers and vendors that use such counters [9-11,21-23]. In order to simplify the calculation process, a conversion is made whenever a measure fits the H description, so that, after being transformed into a percentage, these 8 measures are then rewritten in terms of 100-value, so that they can be "the lowest, the better".

5 Discussion

This first experiment processed a large amount of data: 2.4 GB, with approximately 500.000 rows and 483.000.000 data points. Processing the full data set for feature extraction, anomaly detection and correlation required 7 hours on the utilized processing infrastructure.

In order to verify if the 7 hours for the processing of the total data points was a constant, we attempted the segmentation of data in 5 chunks of ~300 MB as represented in Fig. 7, reporting the time that it took to process the data in each of the cases. The base dataset had a size of 983 MB and took 16 minutes to process. Each additional data set contained the previous ones, so the first trial only had 983 MB, with a length of 16

minutes, whereas the 3rd trial had 1500MB and took 100 minutes to be processed by the algorithms described in section 4.

Fig. 7 shows that in this experiment, the relationship between time and data size is not linear. This could be caused by large number of possible data relations and data length. This could be further explored in terms of searching for an optimum ratio of processing time, size and time series accuracy. This data has been processed using the algorithm from section 3.4 which led to the extraction of the most relevant features. Table 4 represents an excerpt of the extracted components: it contains the most frequently extracted features as well as the frequency of their representations.

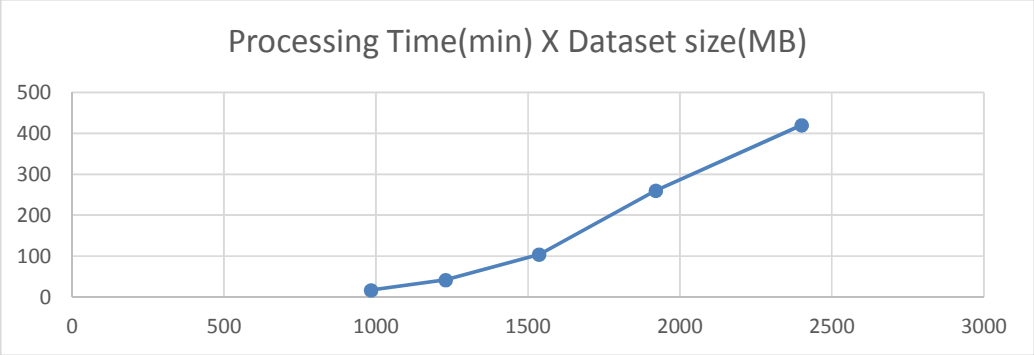

Fig. 7. Non-linear processing lengths, 5 trials, 500 MB chunks
X axis: Megabytes processed
Y axis Processing Time, in minutes

The measures represented in Table 4 are good candidates for an initial investigation of the most frequent root-cause sources for end-user performance degradation events.

The extracted features have been intra and trans correlated, as described in Section 4.4. The Correlation Table for the intra-node correlated performance measures of one node is presented in Table 5. The list of the trans-node correlations between different machines for the same performance measure is presented in Table 6. These are candidate performance measures for being part of the RCA process for performance degradation events.

The data in Table 5 allows to identify, for example, that BESClientUI has an 89% correlation ratio to both (pnamain)\% Processor Time, (pnamain)\% User Time and (nvvsvc)\IO Read Operations/sec. This could indicate that, whenever troubleshooting issues involve one of these particular processes, the other processes could potentially be likewise affected.

Table 6 shows the trans-node correlations for one particular performance measure. We can see that, for the (svchost#1)\IO Read Operations/sec performance measure, there is a high correlation level between nodes 0,1,2,3 and 6: this means that if an issue was to be found in one of these CCS components, which impacted the selected measure, it would be appropriate to also investigate the correlated CCS components for similarities.

Table 7 contains an index of all the measures that have been trans-node correlated during this experiment. Some of them present a large number of correlations, while others correlate only to a single component. When investigating systemic issues or when evaluating distinct users and locations, the findings in this table could be used to guide the identification of possible similarities in otherwise segregated scenarios, thus simplifying the RCA for the degradation events.

Table 4. Most frequently extracted features

Extracted feature name	Frequency
\Process(WmiPrvSE#3)\% User Time	0.41
\Process(EACommunicatorSrv)\IO Read Operations/sec	0.35
\Process(CirratoClient)\% User Time	0.32
\Process(svchost#1)\IO Read Operations/sec	0.32
\Process(sftvsa)\% Privileged Time	0.32
\Process(svchost#3)\IO Read Operations/sec	0.32
\Process(nvvsvc)\% Privileged Time	0.29
\Process(CirratoClient)\% Processor Time	0.29
\Process(csrss)\IO Read Operations/sec	0.29
\Process(concentr)\% Processor Time	0.24
\Process(wfcrun32)\% Processor Time	0.24
\Process(EACommunicatorSrv)\% Privileged Time	0.24
\Process(CentralizedLogPusher)\% User Time	0.24
\Process(CUI)\% Privileged Time	0.24
\Process(wdp)\% Privileged Time	0.24
\Process(AERTSr64)\% Privileged Time	0.21
\Process(concentr)\% Privileged Time	0.21
\Process(CirratoClient)\% Privileged Time	0.21
\Process(wfcrun32)\% User Time	0.21
\Process(winlogon)\% User Time	0.18
\Process(spoolsv)\% Privileged Time	0.18
\Process(idarchive)\IO Read Operations/sec	0.18
\Process(taskhost)\IO Read Operations/sec	0.18
\Process(SCNotification)\% Privileged Time	0.18
\Process(wfcrun32)\% Privileged Time	0.18
\Process(svchost#11)\% Privileged Time	0.18
\Process(svchost#10)\% Privileged Time	0.18
\Process(svchost#8)\IO Read Operations/sec	0.18
\Process(CentralizedLogPusher)\% Privileged Time	0.18
\Process(sftlist)\IO Write Operations/sec	0.18

The anomaly detection algorithm was able to detect different occurrences of anomalies when an anomaly is defined as a difference of 25% of the value observed versus the forecasted value using the Holt-Winters algorithm. Table 8 presents a list of the measures which displayed anomalies in 3 different CCS components. The anomaly detection, as well as the time stamp of each individual event, can be mapped back to the end-user reports of degraded performance to positively map the possible causes to the performance degrading events. These can be candidate RC's for the degradation events.

From Tables 7 and 8 it can be observed, for example, that the measure \Process(Svchost) is a possible candidate for being a RC on performance degradation events: it is seen as trans correlated (i.e. affecting multiple CCS components) as well as intra correlated (on a particular component). It would be possible to narrow the investigation over this particular measure and, in solving this issue, potentially improving the end-user perceived performance for multiple end-users.

Table 5. Correlation table for the intra-node correlated performance measures of one node

	(CirratoClient)\% Processor Time	(CentralizedLogPusher)\%Processor Time	(pnamain)\% Processor Time	(pnamain)\% User Time	(nvvsvc)\IO Read Operations/sec	(WmiPrvSE#3)\% User Time	(SCNotification)\% Processor Time	(AcroRd32)\% Privileged Time	(nvxdsync)\% Privileged Time
CirratoClient)\% Privileged Time	1								
(CentralizedLogPusher)\% User Time		1							
(pnamain)\% User Time			1						
(nvvsvc)\IO Read Operations/sec			1	1					
(SCNotification)\% Processor Time						0.9			
(SCNotification)\% User Time						0.9	1		
(BESClientUI)\% User Time			0.89	0.89	0.89				
(nvxdsync)\% Processor Time									1
(AcroRd32)\% Processor Time								0.95	

Table 6. Trans-node correlation ratios, (svchost#1)\IO read operations/sec

(svchost#1)\IO read operations/sec	node0	node1	node2	node10	node3	node4	node11	node5
node1	1							
node2	1	1						
node3	1	1	1					
node4				1				
node6	1	1	1		1			
node7				.99		.99		
node5							1	
node8							1	1

Table 7. Trans-node correlated performance measures

\Process(CirratoClient)\% Processor Time
\Process(idarchive)\IO Write Operations/sec
\Process(svchost#1)\IO Read Operations/sec
\Process(encsvc)\% Privileged Time
\Process(svchost#5)\IO Write Operations/sec
\Process(csrss)\IO Read Operations/sec
\Process(CirratoClient)\% Privileged Time
\Process(svchost#3)\IO Read Operations/sec
\Process(idarchive)\IO Read Operations/sec
\Process(WmiPrvSE#3)\% User Time

The application of Bautista's model involves the manual association of the measures to the performance concepts proposed in the ISO 25010 standard which, in this experiment, has produced the following distribution:

- "capacity": 10 measures
- "availability": 5 measures
- "time behavior": 27 measures
- "fault tolerance": 9 measures
- "resource utilization": 780 measures
- "maturity": 58 measures

Table 8. Anomaly sources – 3 machine sample

1\Security Per-Process Statistics(1168)\Context Handles
1\Process(svchost#5)\IO Write Operations/sec
1\Process(chrome#7)\IO Read Operations/sec
1\Process(chrome#8)\IO Write Operations/sec
2\Process(chrome)\IO Read Operations/sec
2\Process(TrustedInstaller)\IO Write Operations/sec
3\Security Per-Process Statistics(3428)\Context Handles
3\Security Per-Process Statistics(3464)\Credential Handles
3\Security Per-Process Statistics(8024)\Context Handles
3\Process(iexplore)\% User Time
3\Process(iexplore)\% Processor Time
3\Process(iexplore)\% Privileged Time
3\Process(System)\IO Read Operations/sec
3\Process(iexplore#2)\% Processor Time
3\Process(iexplore#2)\% Privileged Time
3\Process(iexplore#2)\% User Time

It is possible to identify an imbalance between the quantity of measures. This is because the resource utilization measures are multiplied by the number of processes running on a CCS component which represents, at a macro scale, that more resources are being consumed by a particular application. Given the imbalances in the quantity of components for each concept, it was necessary to create a form of aggregator – an indicator – for each performance concept. In order to construct such an indicator, the following process was selected:

1- All the values were initially converted to percentages so that they can be represented on the same scale. For the values with a quality evaluation of "H" in Table 3 the value was further converted into a "100-value" so that the lower the number, the best would be the expected end-user performance.
2- After normalizing all the values to percentages, all the values were plotted on a virtual plane.
3- These points were distributed concentrically over a radial graph, so that the distance in between each and the 0, central point, were equivalent to their values (point 10, being for example 5 units closer – or shorter – to the center than point 15).
4- Using the formula in Algorithm 3 for calculating the area of a circumscribed polygon of N sides, written in Python, the value for the area of each point, as represented in Fig. 8, was calculated. The values for the calculated areas were then represented across the time for the experiment, representing different levels of capacity, availability, time behavior, fault tolerance, resource utilization and maturity.

Algorithm 3: Circumscribed polygon of N sides area calculation, Python

```
count = len(number_of_columns)
  for i in range(count - 1):
     result += arr[i] * arr[i+1]
  result += arr[0] * arr[count-1]
  result *= 0.5
  rad = ((360.0 / (count *1.0)) * math.pi) / 180.0
  return result * math.sin(rad)
```

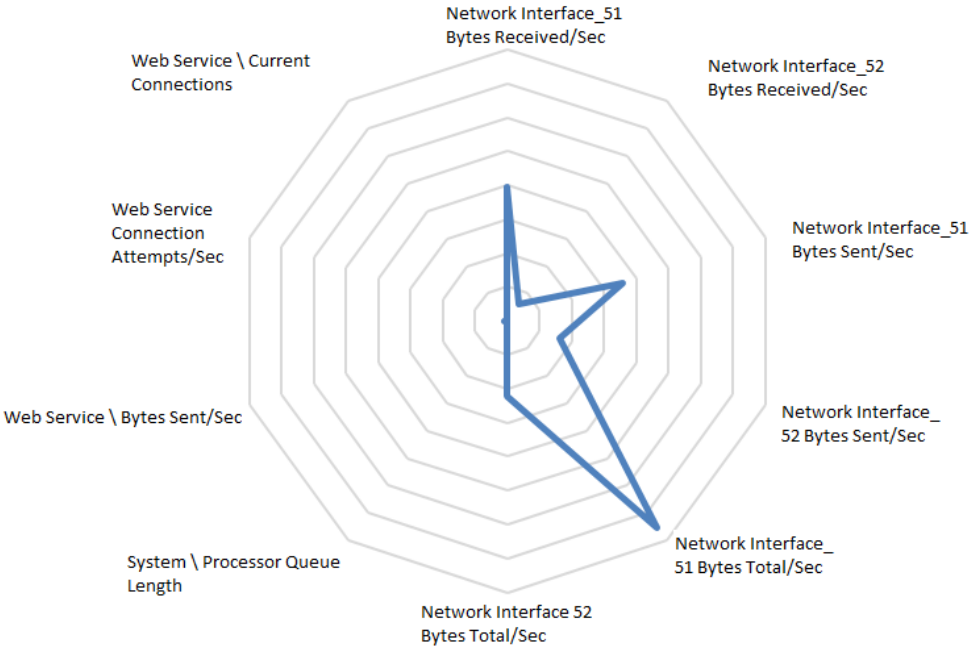

Fig. 8. One point, multiple time behavior measures displayed on a virtual plane

Scale: 0 (center) – 100% (border) utilization of a given measure

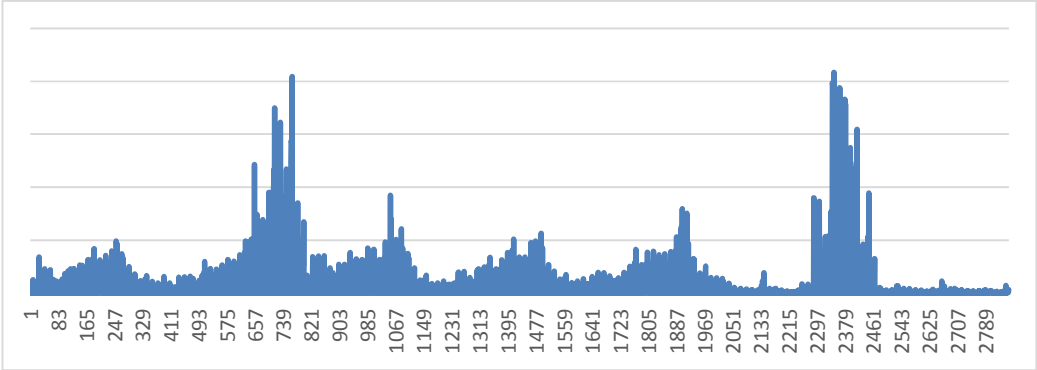

Fig. 9. Time behavior representing peaks in occurrence 765 and 2343

Scale: Calculated area of the N-Sided Polygon, smallest means less resources used, potential better time behavior

In Fig 8, closer to the center means that the resource is freer. The area of this polygon can represent the utilization of the resources.

The multiple areas can be represented in the form of a single time series that respects the quality evaluation of either L or H where appropriate. Essentially, this novel indicator would be able to demonstrate, across a timeline, the events with the highest potential to provide a degraded performance for the end-user.

Fig 9 shows particular points in time when the time behavior concept had high values. This could indicate times where a particular combination of measures presented a degraded behavior as well as displaying, to the datacenter analyst, points where intervention would be more effective. Additionally, the monitoring of these indicators could be part of the development of a service level agreement for CCA based on the end-user performance indicators as calculated in this experiment.

6 Conclusion

This experiment demonstrated that, for the studied case, it has been possible to extract the most relevant performance measures for identifying performance degradation of CCA (Tables 4, 5, 6) using only performance logs. These measures were correlated intra-node and trans-node in order to expose possible degradation events. Anomaly calculation using the Holt-Winters algorithm helped to identify the most relevant performance measures for root cause analysis of the performance degradation events (Tables 7 and 8). These measures should represent the end-user perspective of the performance degradation events given that they are associated with the relevant degradation events and the respective performance measures.

The experimentation modified the original theoretical proposal with the utilization of time series analysis on the performance data in order to determine the performance of a CCA from the perspective of an end-user. An indicator for each performance concept has been introduced, using a formula for calculating the area for an N-Sided circumscribed polygon.

The resulting measurement model can serve as a basis for continuous improvement and cost-effectiveness analysis of a CCA. Industry can also benefit by implementing these algorithms in internal quality measurement systems, possibly provoking the discussion around the topic of end-user performance measurement. Using the indicator to monitor the relative performance levels of the system is simpler than investigating the multitude of performance measures generated by the components of the CCA in order to identify possible performance degradation events.

As an experiment, the conclusions are limited to this particular case. Further applications of the same methodological approach should be performed in order to verify its generalization capacity. Any CCA that is capable of logging performance data in the compatible CSV format could be used in these further investigations. Additionally, the application of the experiment does involve costs in terms of work, processing time, and these have not been evaluated here.

A few challenges were also faced and some opportunities for further research have been identified. Starting with the challenges, we describe how they emerged, and what was done to circumvent them.

Challenges encountered during this research:

1) Theoretical Challenge: Manual association of performance log measures and performance concepts.

   This association of performance log measures and performance concepts, inspired from Bautista et al.'s original work [5], remains manual: it is therefore lengthy and prone to human error. An index would be useful for more easily associating the performance log measures and the concepts.

2) Technological Challenge: Spark.

   Spark allows the distribution of data to be computed on the slave nodes, aiming to increase performance. Unfortunately, Spark lacks support for the most interesting statistical methods

employed in this experiment and, therefore, had to be used in tandem with Pandas, a mathematical package which is very popular amongst Python developers. The processing time could be significantly reduced with a similar software tool supporting all the required functions.

3) Measure Design Challenge

We have observed a disparity in the number of performance log measures associated to each ISO concept. This could be because the discussion of how to properly design measures is not easily mastered in the software industry [42] or because the concern with performance has led to a growing number of measures of all sorts. A discussion with the creators of software with the logging capabilities could take place so that these measures are designed in order to support the quality evaluation.

Opportunities for further research include:

1) Different data sizes and experimental setup configurations (machine types, processing power, number of slave and controller nodes): as seen in Fig. 7, there appears to be a nonlinear relation between the data size and the processing time. This could be investigated further to verify if there is an optimum configuration that achieves the best timely response for calculating the indicator.
2) Data optimization: in this experiment, 30 performance log measures contained 98% of the sample variance. This could be used to reduce the total number of collected measures, potentially simplifying the required calculations and reducing processing time of the collected data.
3) Anomaly detection: In this experiment, we investigated anomaly detection using only the Holt-Winters algorithm. Other avenues could explore the identification of a better performing, more precise algorithm, or even forecasting algorithms.

Competing Interests

Authors have declared that no competing interests exist.

References

[1] Armbrust M, Fox A, Griffith R. A Berkeley view of cloud computing. UCB/EECS-2009-28 EECS Department, U.C.B, Berkeley CA; 2009.

[2] Creeger M. CTO roundtable: Cloud computing. Communications of the ACM; 2009.

[3] Gruschka N, Jensen M. Attack surfaces: A taxonomy for attacks on cloud services. Proceedings of the 3rd IEEE International Conference on Cloud Computing. 2010;276-279.

[4] NIST-The NIST definition of cloud computing. CSD - TIL - NIST, National Institute of Standards and Technology, Gaithersburg, MD; 2011.

[5] Bautista L, Abran A, April A. Design of a performance measurement framework for cloud computing. Journal of Software Engineering and Applications. 2012;5(2):69-75.

[6] Grobauer B, Walloschek T, Stocker E. Understanding cloud computing vulnerabilities. IEEE Security and Privacy. 2011;9(2):50-57.

[7] Juran J, De Feo J. Juran's quality Handbook: The complete guide to performance excellence. McGraw-Hill; 2010.

[8] HP. HP ITSM Transforming IT organizations into service providers; 2013.

[9] NetApp, Inc. CLUEBOX: A performance log analyzer for automated troubleshooting. Netapp, Inc; 2008.
Available:https://www.usenix.org/legacy/event/wasl08/tech/full_papers/sandeep/sandeep_html/
(Accessed 14 03 2016)

[10] Microsoft. Microsoft perfmon; 2013.
Available:http://technet.microsoft.com/en-us/library/cc749249.aspx

[11] Zenoss. Zenoss unified it operations monitor; 2013.
Available:www.zenoss.com

[12] Forster F. Collected open source project. Munich; 2013.
Available:http://www.collectd.org

[13] Weisberg J. Argus TCP monitor; 2013.
Available:http://argus.tcp4me.com

[14] Croll A. How should we measure clouds?; 2013.
Available:http://www.informationweek.com/cloud-computing/software/how-should-we-measure-clouds/240151231

[15] Suhono H, Supangkat S, Saragih R, Suakanto S. Performance measurement of cloud computing services. International Journal on Cloud Computing: Services and Architecture (IJCCSA). 2012;2(2).

[16] Cloudsleuth. Cloudsleuth project; 2013.
Available:http://cloudsleuth.net

[17] Cloudharmony. Cloudharmony project; 2013.
Available:http://cloudharmony.com

[18] Abel A. Benchmarking 5 cloud platforms; 2010.
Available:http://www.infoq.com/news/2010/07/Benchmarking-5-Cloud-Platforms

[19] Meijer G. How do you measure cloud performance; 2012.
Available:http://www.cloudproviderusa.com/how-do-you-measure-cloud-performance/

[20] Ravanello A, April A, Desharnais JM, Bautista LE, Gherbi A. Performance measurement for cloud computing applications using ISO 25010 standard characteristics. Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), Rotterdam, The Netherlands; 2014.

[21] Omniti. Reconnoiter fault detection and trending; 2013.
Available:https://labs.omniti.com/labs/reconnoiter

[22] Munin. Munin monitoring open project; 2013.
Available:http://munin-monitoring.org/

[23] Cacti. Cacti RRD tool; 2013.
Available:http://www.cacti.net

[24] High scalability blog. Troubleshoot response time problems; 2011.
Available:http://highscalability.com/blog/2011/5/11/troubleshooting-response-time-problems-why-you-cannot-trust.html

[25] Real user monitoring blog. The complete list of end user experience monitoring tools; 2011.

[26] Microsoft. Performance analysis of logs tool; 2012.
Available:http://pal.codeplex.com/

[27] Friedl A, Ubik S. Perfmon and servmon: Monitoring operational status and resources of distributed computing systems. CESNET Technical report 1/2008; 2008.

[28] Kufrin R. Measuring and improving application performance with perfsuite; 2005. Available:http://perfsuite.ncsa.illinois.edu/publications/LJ135/

[29] Iosup A, Ostermann S, Nezih Y, Prodan R, Fahringer T, Epema D. Performance analysis of cloud computing services for many-tasks scientific computing. IEE TPDS Many-Task Computing; 2010.

[30] Mei Y, Liu L, Pu X, Sivathanu S. Performance measurements and analysis of network I/O applications in virtualized cloud. Georgia Institute of Technology, Atlanta, GA; 2010.

[31] Emery JC. The impact of information technology on organizations. 24th Annual Meeting, Academy of Management. 1964;1-27.

[32] Davis FD. Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly. 1989;13(3):319-339.

[33] Etezadi-Amoli J, Farhoomand A. A structural model of end user computing satisfaction and user performance. ELSEVIER, Information & Management. 1996;30:65-73.

[34] Davis S, Wiedenbeck S. The mediating effects of intrinsic motivation, ease of use and usefulness perceptions on performance in first-time and subsequent computer users. Interacting with Computers. 2001;13(5):549-580.

[35] Fagan M, Neill S. An empirical investigation into the relationship between computer self efficacy, anxiety, experience, support and usage. Journal of Computer Information Systems. 2004;44(2):95-104.

[36] Law E, Roto V, Hassenzahl M, Vermeeren A, Kort J. Understanding scoping and defining user experience: A survey approach. Proceedings of Human Factors in Computing Systems, CHI' 09. 2009;719-728.

[37] Mahmood M, Burn J, Gemoets L, Jacquez C. Variables affecting information technology end-user satisfaction: A meta-analysis of the empirical literature. IJHCS. 2010;54:751-771.

[38] Tullis T, Albert B. Measuring the user experience: Collecting, analyzing, and presenting usability metrics. Morgan Kaufmann; 2010.

[39] Marshall B, Mills R, Olsen D. The role of end-user training in technology acceptance. Review of Business Information Systems. 2008;12(2).

[40] Baer T. Application performance is in the eyes of the end-user. OVUM, CYIT0122; 2011.

[41] ISO. ISO/IEC 19761:2011 - Software engineering - COSMIC: A functional size measurement method. International Organization for Standardization (Geneva); 2011.

[42] Abran A. Software metrics and software metrology. New York: John Wiley & Sons Interscience and IEEE-CS Press; 2010.

[43] Hyvärinen A, Oja E. Independent component analysis: Algorithms and applications. Neural Networks. 2008;4-5(13):411-430.

[44] Wei WWS. Time series analysis: Univariate and multivariate methods. Victoria, AU AbeBooks; 1990.

[45] Winters PR. Forecasting sales by exponentially weighted moving averages. Management Science. 1960;3(6):324-342.

[46] Apache Foundation. Apache spark overview; 2013.
Available:http://spark.apache.org/
(Accessed 04 07 2014)

[47] Esmael B, Arnaout A, Fruhwirth R, Thonhauser G. A statistical feature-based approach for operations recognition in drilling time series. International Journal of Computer Information Systems and Industrial Management Applications. 2015;5:454-461.

[48] ISO. ISO 5725-1:1994(en) - Accuracy (trueness and precision) of measurement methods and results — Part 1: General principles and definitions. International Organization for Standardization (Geneva); 1994.
Available:https://www.iso.org/obp/ui/#iso:std:iso:5725:-1:ed-1:v1:en
(Accessed 22 04 2016)

Appendix 1

Complete list of Performance Measure types collected during the experiment, the type of component where the measure can be found and the association of the measure with a particular ISO 25000 quality concept.

Performance log data measure name	CCS component type	ISO 25000 quality concept
\ LogicalDisk(*)\Free Megabytes	Client, Server	Capacity
\Netlogon(*)\Average Semaphore Hold Time	Server	Maturity
\Memory\Page Faults/sec	Client, Server	Maturity
\Memory\Available Bytes	Client, Server, network	Capacity
\Memory\Pages/sec	Client, Server	Time behavior
\Paging File(*)\% Usage	Client, Server	Time behavior
\System\File Read Bytes/sec	Client, Server	Resource utilization
\System\File Write Bytes/sec	Client, Server	Resource utilization
\System\System Up Time	Client, Server	Availability
\System\Processor Queue Length	Client, Server	Time behavior
\System\Processes	Client, Server	Availability
\System\Threads	Client, Server	Capacity
\Processor Information(*)\% Privileged Time	Client, Server, Network	Resource utilization
\Processor Information(*)\% User Time	Client	Resource utilization
\Processor Information(*)\% Processor Time	Client, Server, network	Resource utilization
\LogicalDisk(*)\Current Disk Queue Length	Client, Server	Time behavior
\PhysicalDisk(*)\Disk Reads/sec	Client, Server	Capacity
\PhysicalDisk(*)\Disk Writes/sec	Client, Server	Capacity
\Processor(*)\% Processor Time	Client, Server, Network	Time behavior
\Processor(*)\% User Time	Client	Time behavior
\Processor(*)\% Privileged Time	Client, Server, network	Time behavior
\Search Indexer(*)\Master Index Level	Client	Maturity
\Client Side Caching\Application Bytes Read From Server (Not Cached)	Server	Fault tolerance
\Client Side Caching\Application Bytes Read From Server	Server	Fault tolerance
\Client Side Caching\Application Bytes Read From Cache	Server	Recoverability
\Client Side Caching\Prefetch Bytes Read From Server	Server	Fault tolerance
\Client Side Caching\Prefetch Bytes Read From Cache	Server	Fault tolerance
\Client Side Caching\Prefetch Operations Queued	Server	Fault tolerance
\Client Side Caching\SMB BranchCache Hash Bytes Received	Server	Fault tolerance
\Client Side Caching\SMB BranchCache Hashes Received	Server	Fault tolerance
\Client Side Caching\SMB BranchCache Hashes Requested	Server	Fault tolerance
\Client Side Caching\SMB BranchCache Bytes Requested From Server	Server	Time behavior
\Client Side Caching\SMB BranchCache Bytes Published	Server	Time behavior
\Client Side Caching\SMB BranchCache Bytes Received	Server	Time behavior
\Client Side Caching\SMB BranchCache Bytes Requested	Server	Fault tolerance

Performance log data measure name	CCS component type	ISO 25000 quality concept
\Offline Files\Bytes Received/sec	Client	Time behavior
\Offline Files\Bytes Transmitted/sec	Client	Maturity
\Offline Files\Bytes Transmitted	Client	Maturity
\Offline Files\Bytes Received	Client	Time behavior
\Terminal Services\Total Sessions	Server	Availability
\Terminal Services\Inactive Sessions	Server	Maturity
\Terminal Services\Active Sessions	Server	Availability
\Security System-Wide Statistics\NTLM Authentications	Server	Maturity
\Security System-Wide Statistics\Kerberos Authentications	Server	Maturity
\Distributed Transaction Coordinator\Active Transactions	Server	Availability
\Distributed Transaction Coordinator\Committed Transactions	Server	Time behavior
\Security Per-Process Statistics(*)\Credential Handles	Server, network	Maturity
\Security Per-Process Statistics(*)\Context Handles	Server, network	Resource utilization
\Authorization Manager Applications(*)\Number of Scopes loaded in memory	Network	Resource utilization
\Authorization Manager Applications(*)\Total number of scopes	Network	Resource utilization
\Network Interface(*)\Bytes Received/sec	Host, server, network	Capacity
\Network Interface(*)\Bytes Sent/sec	Host, server, network	Capacity
\Process(*)\% Processor Time	Host, Server	Resource utilization
\Process(*)\% User Time	Host	Resource utilization
\Process(*)\% Privileged Time	Host, Server	Resource utilization
\Process(*)\IO Read Operations/sec	Host, Server	Resource utilization
\Process(*)\IO Write Operations/sec	Host, server	Resource utilization

Appendix 2

Acronyms:

CCA – Cloud Computing Applications
CC – Cloud Computing
RC- Root Cause
RCA – Root Cause Analysis
NIST – U.S. National Institute of Standards and Technology
SLA – Service Level Agreement
IT – Information Technology
OS – Operating System

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://sciencedomain.org/review-history/15931