A HIERARCHICAL ARCHITECTURE FOR ON-LINE CONTROL OF PRIVATE CLOUD-BASED SYSTEMS

ABSTRACT

Several enterprise data centers are adopting the private cloud computing paradigm as a scalable, cost-effective, robust way to provide services to their end users. The management and control of the underlying hw/sw infrastructure pose several interesting problems. In this paper we are interested to evidence that the monitoring process needs to scale to thousands of heterogeneous resources at different levels (system, network, storage, application) and at different time scales; it has to cope with missing data and detect anomalies in the performance samples; it has to transform all data into meaningful information and pass it to the decision process (possibly through different, ad-hoc algorithms for different resources). In most cases of interest for this paper, the control management system must operate under real-time constraints.

We propose a hierarchical architecture that is able to support the efficient orchestration of an on-line management mechanism for a private cloud-based infrastructure. This architecture integrates a framework that collects samples from monitors, validates and aggregates them. We motivate the choice of a hierarchical scheme and show some data manipulation, orchestration and control strategies at different time scales. We then focus on a specific context referring to mid-term management objectives.

We have applied the proposed hierarchical architecture successfully to data centers made of a large number of nodes that require short to mid-term control and in our experience we can conclude that it is a viable approach for the control of private cloud-based systems.

KEYWORDS

Cloud computing, architecture, design, statistical models, anomaly detection, hierarchical.

1. INTRODUCTION

The design of a data center has come at an evolutionary crossroad. The increasing variety and complexity of end-user applications, the massive data growth, the challenging economic conditions, and the physical limitations of power, heat, and space are all issues which must be accounted for modern hardware and software infrastructure. Finding architectures that can take cost, complexity, and associated risk out of the data center while improving service levels has become a major objective for most enterprises (Armbrust, S. et al, 2009; Wood, T. et al, 2009; Meisner, D. and Gold, B. T. and Wenisch, T., 2009). In this scenario, the private cloud computing model is starting to be seen as an ideal paradigm for a service hosting platform. A private cloud applies the main concepts of cloud computing, such as on-demand resources, accounting, service oriented architectures, and the appearance of infinite scalability (Vaquero, L. M. et al, 2009; Armbrust, S. et al, 2009) to resources owned by the enterprise. Enhancing existing infrastructure with cloud computing capabilities leads to a reduction in operating costs, provides a scalable and robust execution environment and, ultimately, allows the adoption of an economy of scale through the Software-As-A-Service paradigm.

The adoption of a private cloud paradigm opens several issues at the level of system governance at real-time and off-line. In this paper, we focus on the former context. Enterprise data centers based on private clouds are characterized by a high number of heterogeneous hardware and software components (processors, memories, storage elements, virtual machines, applications, business enforcement modules) that operate at different time scales (spanning from seconds to entire weeks), interact in possibly unpredictable ways, can be subject to prompt reconfiguration due to changes in system, security, and business policies. In these conditions, it is impossible to take critical decisions by simply looking at the output of the performance and utilization measures provided by the cloud infrastructure monitors. There are too many distinct data flows (literally, thousands per second) to analyze; many of them may contain missing or faulty measures; often, in order to gain a meaningful insight into the state of a system, it is necessary to correlate and aggregate several flows. All these decisions must be taken in real-time. To make matters worse, the same algorithm that

operates well at a given time scale may fail at a different one. Thus it is not immediately clear which algorithms are more appropriate at different time scales. For these reasons, we are witnessing a shift from basic resource monitoring and management (allocation, activation, deactivation based on direct performance measures) to *service orchestration*, which shares the following goals:

- extract from a multitude of available raw performance measures those that are really relevant for a given business, security, or system objective;
- correlate and aggregate (both temporally and spatially that is, across different components) the most relevant time series in order to acquire an information about the internal system state.

In this paper, we propose a hierarchical architecture that supports models and methodologies for the efficient resource management of an on-line control enforcement mechanism operating on a private cloud-based infrastructure. Our design addresses the scalability challenge (related to the huge number of monitored information available from monitors) in several ways. The system is logically divided into several subsystems according to the different management time spans (short, medium, long term). A significant subset of the data processed by the monitoring framework at shorter time spans is made available to the runtime management modules operating at longer time spans. In this way, the duplication of preliminary operations such as data filtering is avoided. Furthermore, the modules operating at longer time scales identify from the myriad of available measures those that are really critical for the system. The goal is avoid the unnecessary and often computationally infeasible task of storing, monitoring and processing all the measures available from shorter time spans.
In the final part of the paper, we show an example of the difficulties behind on-line control enforcement at medium-term time scales.

The paper is organized as follows. Section 2 details the reference architecture based on private clouds, explains the available measurements and the control enforcement operations, and formalizes the problem of service orchestration. Section 3 introduces the on-line analysis framework and its main components. Section 4 provides an analysis of different linear and non linear models to resources operating at medium-term time scales. Finally, Section 5 concludes the paper with some final remarks.

2. ARCHITECTURE

Figure 1 describes our model of an enterprise data center driven by business and system policies. The system is split into several layers, whose components (technologies, mechanisms and algorithms) are typically spread across the private cloud infrastructure. The *System Infrastructure Layer* is responsible for the fruition of the information stored in the data center. This information may have different origins; for example, it may pertain to user applications, or it may refer to system resources performance data. All the components are monitored and tested continuously; a control enforcement module makes sure that the proper system policies are operated correctly. The *Business Layer* hosts all the components responsible for the enforcement of proper business policies, such as ERP, supply chain management, Finance, Sales. The *Governance Layer* is responsible for the orchestration of the entire infrastructure at the system and business level. In this layer, the information coming from the system and the business layers is processed in order to detect policy violations or anomalies. The Governance Layer also produces reports and dashboards which the corporate management uses to define the business, system and security policies (Kaliski, B. S., 2010).

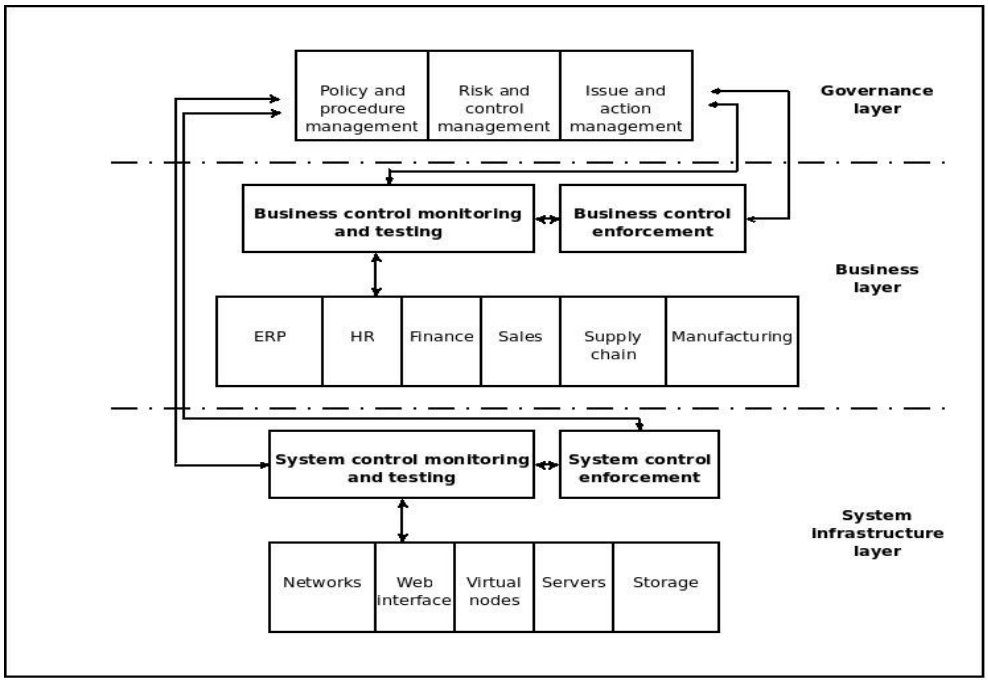

Illustration 1: High level architecture of a private cloud-based data center

In this paper, we focus on the architectural supports operating at the System infrastructure layer. These models support the on-line management tasks that are executed at different temporal scales (ranging from seconds up to a few hours). In a private cloud-based architecture, these management tasks use several control actions available to the underlying architecture to enforce control, including: admission control, dispatching and scheduling, load (re)balancing, addition and removal of a physical host from the pool of available resources, (live) migration of virtual machines, resource limitation through traffic shaping and container-based techniques, resource reallocation (to accommodate high priority tasks, to reduce power consumption). The control actions are executed as a reaction to different events at different time scales. For example, a request enters the system, or an anomaly has been detected in the last hour, or a daily forecasting analysis suggests the re-configuration of a whole subsystem. To enforce control, the management tasks require some measurement from performance monitoring tools operating at different levels (physical hosts, virtual machines, applications), including CPU utilization, memory utilization, network throughput, power consumption, storage-related metrics such as read and write throughput and disk utilization, application-level data (application throughput, response times, failure rates). We can assume that these performance monitoring samples are available periodically. Previous literature (Andreolini, M. and Casolari, S. and Colajanni, M., 2008), shows how the collection of several performance samples in distinct time series can provide the basis for efficient monitoring and prediction of system resource behavior. Thus, we will also assume to have a reasonably small amount of past sample history, in the form of fixed-window time series. Unfortunately, choosing the right sampling interval is a challenging task, because different management tasks run at different time scales and may not need (or, worse, not operate properly in presence of) frequent monitored data streams. It is therefore also necessary to place the management tasks at the "right" time scale. In this paper, we distinguish three different time scales:

- short time scale;
- medium (mid-term) time scale;
- long time scale.

Tasks operating at short time scales must take decisions and enforce them in the range of seconds, typically under a minute. These decisions allow the system to serve user requests in a best-effort fashion, given the actual configuration. Common control actions include admission control, resource reallocation due

to the execution of some high priority application, dispatching of a request to the proper subsystem or server. Short time scales are critical for two reasons. First, there is typically no time for complex analyses on the monitoring data obtained from the probes. Experience shows that only data filtering is viable. As a consequence, the decisions taken by a short time task must be taken using the data as-is, cannot be optimal, and must focus on avoiding disasters than on achieving optimal performance. Second, the volume of monitoring data can be very high. Thus, even the storage of the performance samples in a database for later analysis can be a problem due to the high computational overhead and to the disk space consumption involved (Ousterhout, J. et al, 2010). Algorithms operating at short time spans must often treat the monitoring data as a stream of numbers.

Tasks operating at mid-term time scale must take decisions and enforce them in the minute range (typically, under a hour). These decisions aim to adapt and improve the configuration of the system according to changed operating conditions. Common control actions include dynamic load balancing, virtual machine migration, node activation, dynamic resource allocation. These tasks have the time to process the monitored data, but, in order to optimize the execution of short-time tasks, they often have to aggregate different pieces of information both in time (through time series smoothing techniques) and in space (across different system components).

Tasks operating at long time scale must take decisions and enforce them in the hours range. These decisions share the goal of optimizing the system at real-time but in a longer horizon. Common control actions include resource reallocation due to some optimization (a typical goal is to pack the applications into the smallest number of nodes minimize to power consumption), and capacity planning activities. At this time scale, tasks have the time to fully analyze their raw data, produce reports and store the relevant results (since the required disk space is not an issue). However, the complexity of the decision process increases as, besides time and space aggregations, these tasks also need to compute predictions, analyze what-if scenarios, encompassing the whole system.

Any infrastructure for the on-line control enforcement of a private cloud-based architecture must support these operations in a scalable way. Scalability can be achieved by reducing the number of nodes handled by each management task and by reducing the overhead due to storage and processing to a minimum. This translates to the following architectural requirements:

- make sure that any task (in particular, a short-time task) is responsible for a reasonable number of system components;
- support persistent storage only at longer time scales;
- make the results of shorter analyses available as a starting point for longer analyses
- support more sophisticated aggregation techniques (time and space) only at longer time scales.

In the next section, we propose a hierarchical architecture that can integrate solutions for most real-time management problems.

3. A HIERARCHICAL ARCHITECTURE FOR RESOURCE MANAGEMENT

In Figure 2 we propose our hierarchical architecture for the support of on-line management tasks. (We note that the hierarchy is logical, but each layer can be implemented through several servers and databases). In this scheme, the system consists of different, smaller, manageable subsystems, each of which is controlled by a set of management tasks operating at different time spans. At the lowest level, we have subsets of hardware and software resources which can be associated to subnets, racks, distinct production areas, and the like. Each subset is governed through the control actions operated by a control enforcement module. Every control enforcement module is driven by an orchestration module, which is at the heart of every management task. The main purpose of the orchestration module is to take the “right” decisions based on some measure of the subsystem's internal state. Depending on the chosen time scale, the nature of the monitored data varies. At short time scales, a task has at most grossly filtered performance measures available (to avoid out-of-scale values, outliers and missing data). At longer time scales, tasks usually

perform more sophisticated statistical operations based on time and space aggregations, anomaly detection, predictions.

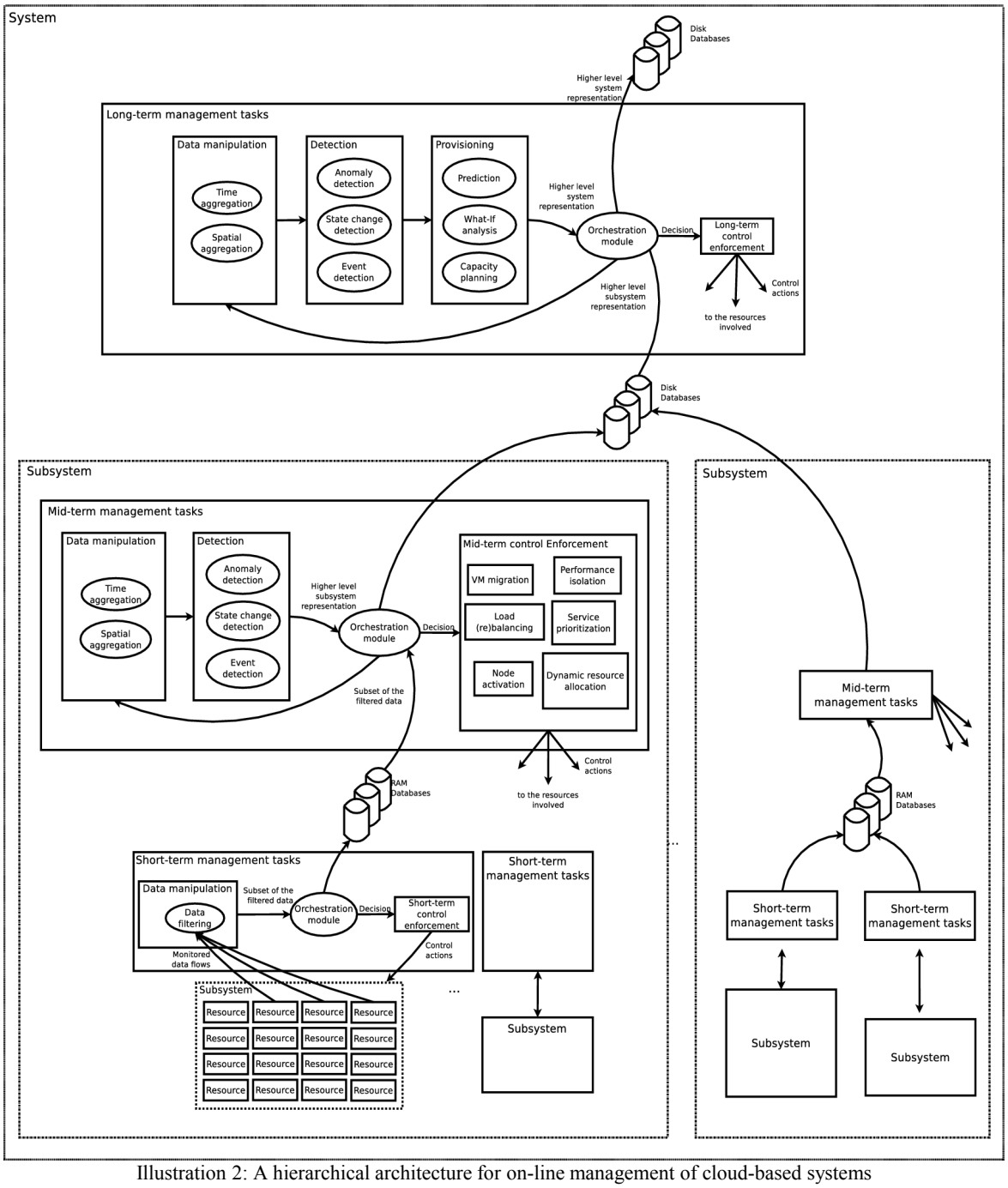

Illustration 2: A hierarchical architecture for on-line management of cloud-based systems

In order to keep the control enforcement process scalable, each management task must orchestrate the control using the least amount of resources available. This implies the monitoring and control of only a limited subset of the entire system (this holds particularly true for short-time tasks). We decide not to permanently store each raw performance measure obtained from the subset, since this would be unfeasible, given the literally hundreds of time series that would need to be monitored. Instead, we treat each monitor as a stream

of data which is filtered in a basic fashion and made directly available to the orchestrator. This operation is straightforward, since low-level system monitors such vmstat and sar can be easily instrumented to pipe their output to different processes. For some proprietary systems, it is possible to exploit the SNMP querying capabilities offered by more sophisticated tools such as Cacti, Nagios, Zenoss, to extract the same information from several system components.

Since longer tasks tend often to use the results of shorter tasks, we instrument the orchestrator to extract proper subsets of the filtered data (typically, the last n performance measures coming from the most important monitor probes) and to store them in a lightweight database. In this way, we reduce considerably the set of monitored data available to the mid-term and long-term tasks, and we provide a persistent data storage which is necessary to hold data spanning in longer ranges. A RAM database such as Firebird would fit perfectly, since it provides excellent performance over a moderately high volume of stored data. Our experience shows that this approach allows to monitor tens of hardware and software resources per node, in a subsystem consisting of a few hundreds nodes.

The mid-term management tasks take the filtered data, aggregate it, detect (and possibly correct) anomalies, and produce a meaningful representation of a subsystem's internal state, which is used to enforce control decisions aimed at improving the performance and optimizing the present behavior relevant subsystems. These operations can be implemented through any standard math environment (Matlab, R Octave) or through mathematical libraries available for the most popular general purpose languages (Scipy, Numpy in Python). With respect to the whole set of time series available from the monitors, the representations produced by mid-term management tasks are much more compact in terms of size and are computed less often; thus, they can be stored into a DBMS, such as MySQL, PostgreSQL, Oracle.

Finally, the long-term management tasks also perform more sophisticated tasks oriented improve the performance and optimize the behavior of the whole system in the future. To this purpose, they retrieve the state representations computed by the mid-term management tasks and perform sophisticated computations involving long-term predictions, what-if analysis, capacity planning. The resulting models drive the decisions of the orchestration modules.

Our architecture is designed in such a way that the control actions can be implemented through off-the-shelf hardware and software components. For example, request dispatching and load balancing can be operated through standard setups based on Apache2 (through the mod_rewrite and mod_proxy modules), Tomcat (through AJP connectors in a clustered setup) and, more recently, on nginx. There are several viable alternatives for the virtualization of services; the most popular are Xen (version 3.0 and above) and KVM (the hypervisor officially adopted by the Linux community). All these solutions support (live) migrations, dynamic resource re-allocation, ballooning, paravirtualized device drivers for close-to-native performance. Another alternative is the use of resource containers (OpenVZ), which provide native performance at the expense of executing a shared operating system kernel for all running services.

4. MID-TERM ANALYSIS

In this section, we show several examples pointing out different problems related to on-line management of cloud-based architectures. Due to space constraints, we focus on the mid-term temporal scale. At this level, control enforcement is pursued through some very specific control actions, such as virtual machine migrations, load balancing, resource reallocation, node activation. The decisions leading to these actions are taken by the proper orchestration module, which has at its disposal a subset of the data filtered by shorter-term management tasks and available efficiently through RAM databases. Unfortunately, the monitored data available to the orchestrator cannot be used to fulfill the management goals; it must be first transformed into a higher level representation of a subsystem through data manipulation and anomaly detection techniques. For example, let us consider a set performance metrics commonly available through off-the-shelf monitoring tools. Figure 3 shows the behavior of CPU utilization performance samples (obtained at intervals of 5 minutes) of a server node. The samples have been filtered to exclude outliers and out-of-scale values.

A first problem with this data set is its marked oscillatory behavior, which makes it very difficult (if not impossible) to deduct a clear representation of the server load conditions. If this data were to be used by the orchestrator module to enforce load balancing across different servers, the result would likely be an

unstable subsystem where the load balancer constantly bounces process from one server node to another, in an endless effort to even the load. Hence, at the mid-term level our architecture must support the extraction of a more clean trend from the available data. This is possible through the adoption of *time aggregation models*. If we consider the subset of performance data as a fixed-window time series, we can apply aggregation models to it in order to retrieve a synthetic representation of the resource behavior. In the considered mid-term context, time aggregation is generally pursued through the adoption of linear smoothing that is, moving-average techniques, such as exponential moving average (D. J. Lilja, 2000), through regression and auto-regressive models (P. Dinda et al, 2000), and through interpolation techniques, such as the cubic spline (D. J. Poirier, 1973).

These models have been shown to provide a good level of precision at reasonable computational costs and in reasonable time (Andreolini, M. and Casolari, S. and Colajanni, M., 2008, Dinda, P. et al, 2000), compatible with the constraints imposed by mid-term management (usually, under one hour). Our experience shows that the majority of time aggregation functions require a CPU processing time well below 1 ms. This implies that, in one minute, a mid-term management task can aggregate at least 60000 time series. Our hierarchical architecture is capable of handling up to 300000 aggregations per minute by adopting a simple Exponential Weighted Movie Average (EWMA) model based on short, fixed windows. Figure 4 shows a time aggregation of the CPU utilization shown in Figure 3, based on the Exponential Weighed Moving Average of the last 30 measures. The smoothing effect of the EWMA makes it easier to detect an oscillating server load and lowers the risk of unnecessary control enforcement.

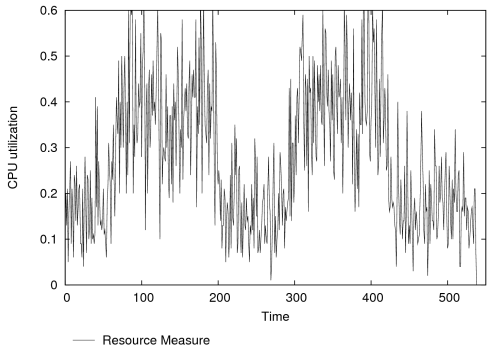

Illustration 3: Available Monitored Data

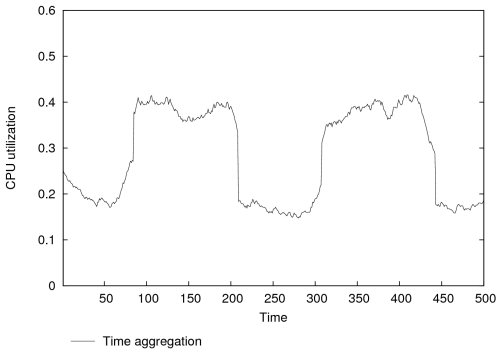

Illustration 4: Time aggregation (EWMA)

In a complex system made up of several thousands of hardware and software components, even identifying the failing nodes may become a computationally intensive task. Thus, a scalable monitoring framework needs to reduce the number of relevant system state information made available to the orchestration module. This goal can be achieved by distinguishing the components that are critical for the performance of the entire systems from those that are not. To make matters worse, it is very difficult if not impossible to infer the internal conditions of a subsystem from several, distinct internal state representations of (albeit relevant)a single components. We may not understand the reasons behind the failure, and we could simply not tell whether that failure is about to cause further misbehavior in the system. Hence, our monitoring framework must also aggregate different, heterogeneous, time-aggregated series in order to produce a higher level view of a subsystem, which allows to tell whether a subsystem is performing suboptimally and why. To these purposes, *spatial aggregation models* are often used to analyze and combine multiple heterogeneous data sources into a single, coherent view of a system. Techniques such as the multi-variate analysis (K.V. Mardia et al, 1979) and the Principal Component Analysis (PCA) (H. Hotelling, 1933) are effective in capturing the salient features of a subsystem's internal state, thus drastically cutting the amount of data used to perform decisions. Our experience shows that the PCA fits well in a hierarchical architecture where each subsystem can be efficiently handled by a subset of tasks and can provide meaningful insight to the longer-term tasks. We have handled the reduction of more than one million samples pertaining to 1050 different components (21 hardware and software resources of 50 nodes) to a subset of 12 principal components in less than a minute. A simple weighted aggregation model applied on the selected principal features allows to estimate a reliable representation of the system's internal state. Figure 5 shows, on the left, 2 out of the 12 principal components characterizing an entire subsystem. The 12 principal

components are aggregated through a weighed regression model, shown on the right of Figure 5. Ultimately, the status of an entire subsystem can be represented through a single time series, available to the mid-term orchestrator and to the longer-term management tasks.

Spatial aggregation of different resources in an the context of an on-line management system for distributed systems is still an open research problem, due to the the high number of time series available and to the heterogeneity of business-level and system-level measures (Zhu, X. et al, 2009).

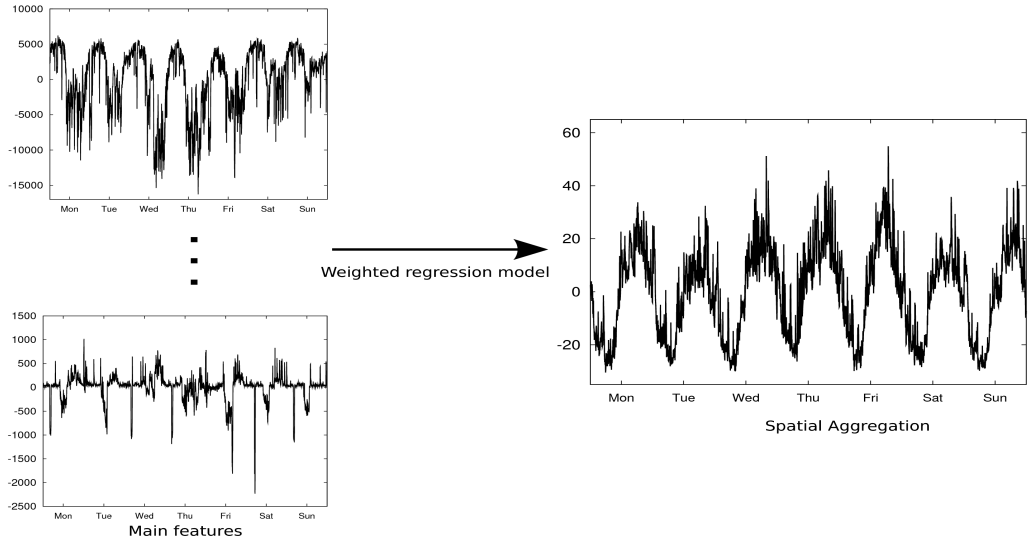

Illustration 3: Spatial Aggregation

Control enforcement is applied as a reaction to events that perturb the normal operating conditions of the system: anomalies in the workloads, changes in the offered load, events that normally do not occur. The orchestrator must be able to detect these events timely through the adoption of on-line *anomaly detection*, *state change detection* and *event detection models*. We have evaluated and integrate several on-line detectors in our hierarchical architecture. The models relying on a prior knowledge of all the possible states of a process (Lu, D. et al. 2004) are completely inadequate both statistically due to the non deterministic behavior of the resource measures and for their unfeasible computational costs. Other widely adopted methods that use one or more load thresholds for detecting relevant state changes (Ramanathan, P. 1999.) seem unsuitable to the highly variable context of interest for this paper. In the private cloud-based architectures that are characterized by high variability of the measures, by non stationary and by unpredictable behavior, a good state change detection model that is able to guarantee adequate solutions while respecting the time constraints of the mid-term management, is based on the Cumulative Sum (Cusum) statistics (Montgomery, D. C., 2008; Basseville, M. et al, 1993). Figure 6 and Figure 7 illustrate the behavior of the threshold-based and the Cusum-based state change detection models, respectively. Filtered and non-filtered data is overlaid point out relevant state changes; true and false positives are depicted with crosses and circles at the bottom of the figures.

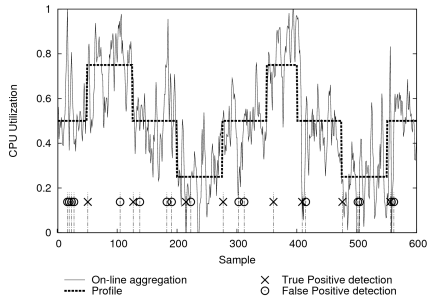

Illustration 4: Threshold-based model

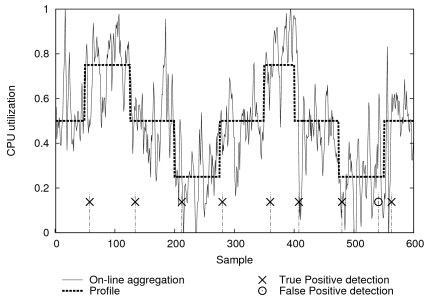

Illustration 5: Cusum-based model

For our evaluation purposes we consider the data aggregation generated by the data manipulation of the CPU utilization of the subsystem nodes. An off-line analysis allows us to determine that relevant state change to be signaled to the control-enforcement management system occurs at times 50, 125, 200, 275, 350, 400, 475, and 550, as shown by the horizontal/vertical lines in the figures. This scenario characterized by multiple state changes can be considered a good benchmark for on-line detection models. Traditional threshold-based detection models are capable of detecting all state changes, but their precision decreases, resulting in a high number of false detections during the stable states. This is a consequence of the inability of threshold-based models to guarantee reliable detections in highly variable contexts. On the other hand, the Cusum-based detection model exhibits a high detection quality. The proposed model detects timely the state change and it is affected by just one false detection at sample 540. Because of the computational cost of the considered model is able to provide reliable detections in the range of milliseconds it can be completely integrate in our hierarchical architecture.

5. CONCLUSIONS AND FUTURE WORK

The private cloud computing model is a viable paradigm for a service hosting platform, capable of keeping up with the challenging requirements behind complex end-user applications, massive data growth, sophisticated business models and the physical limitations of power, heat, and space. In this paper, we have proposed an architecture that supports models and methodologies for the efficient resource management of an on-line control enforcement mechanism operating on a private cloud-based infrastructure. The design of the architecture addresses several interesting challenges:

- it is modular, easy expandable through off-the-shelf hardware and software components;
- it integrates an on-line monitor and analyzer, suitable for run-time decisional tasks at different temporal scales (short, mid-term, long);
- it is hierarchical, allowing for scalable monitoring and control enforcement of subsystems made of up to hundreds of components.

We have also outlined the difficulties of on-line management and control enforcement at a given temporal scale (mid-term). In particular, the monitored time series available to the orchestrator must be subject to time and spatial aggregations (in order to derive a clear view of the controlled subsystem). Furthermore, on-line detection algorithms are necessary to identify anomalies in an otherwise normally operating subsystem. Control enforcement is typically pursued as a reaction to these anomalies. Our experience shows that management can happen efficiently even at different time scales. In particular, in the time span of one minute our architecture can:

- seamlessly aggregate several tens of thousands time series over time, producing a clear view for every component;
- identify the most relevant components of a system made up of one thousand components.

As a next step, we plan to enrich our on-line management task with supports oriented to power consumption (Raghavendra, R. et al, 2008), for example to place the incoming workload in areas at lower temperature or with higher cooling capacity. We will also try to scale our testbed to larger sizes to evaluate the scalability limits of the proposed architecture.

REFERENCES

Armbrust, S. et al, 2009. *Above the Clouds: A Berkeley View of Cloud Computing.* UC Berkeley Reliable Adaptive Distributed Systems Laboratory.

Wood, T. et al, 2009. The Case for Enterprise-Ready Virtual Private Clouds. *Proceedings of the Workshop on Hot Topics in Cloud Computing.* San Diego, CA, pp. 10-15.

Zhu, X. et al, 2009. 1000 islands: An integrated approach to resource management for virtualized data centers. *Cluster Computing*, Special Issue on Autonomic Computing, Volume 12, Number 1, pp. 172-181.

Clark, C. et al, 2005. Live migration of virtual machines. *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI'05)*. Boston, MA, pp. 273-286.

Andreolini, M. and Casolari, S. and Colajanni, M., 2008. M. Models and framework for supporting run-time decisions in Web-based systems. *In ACM Transactions on the Web.* Vol. 2, no. 3, Aug. 2008.

Meisner, D. and Gold, B. T. and Wenisch, T., 2009. PowerNap: eliminating server idle power. *In ACM SIGPLAN Notices.* Vol. 44, Number 3, pp. 205-216.

Vaquero, L. M. et al, 2009. A break in the clouds: towards a cloud definition. *In ACM SIGCOMM Computer Communication Review.* Vol. 39, Number 1, pp. 50-55.

Kaliski, B. S., 2010. Towards Risk Assessment as a Service in Cloud Environments. *Proceedings of the 2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'10)*. Boston, MA, pp. 20-29.

Ousterhout, J. et al, 2010. The case for RAMClouds: scalable high-performance storage entirely in DRAM. *In ACM SIGOPS Operating Systems Review.* Vol. 43, Number 4, pp. 92-105.

Hotelling, H. 1933. *Analysis of a complex statistical variables into principal components*. J. Educ, Psy.

Lilja, D. J. 2000. *Measuring computer performance. A practitioner's guide.* Cambridge University Press.

Dinda, P. et al, 2000. Host load prediction using linear models. *Cluster Computing.* Vol. 3, Number 4, pp. 265-280.

Poirier, D. J. 1973. Piecewise regression using cubic spline. *Journal of the American Statistical Association.* Vol. 68 Number 343, pp. 515-524.

Mardia, K.V. et al, 1979. *Multivariate Analysis.* Academic Press.

Montgomery, D. C., 2008, *Introduction to Statistical Quality Control.* John Wiley and Sons.

Basseville, M. et al, 1993. *Detection of Abrupt Changes:Theory and Application.* Prentice-Hall.

Lu, D. et al. 2004. Change detection techniques. In *Int. Journal of Remote Sensing.* Vol. 25, Number 12, pp. 2365-2347.

Ramanathan, P. 1999. Overload management in real-time control applications using (m,k)-firm guarantee. *Performance Evaluation Review.* Vol. 10, Number 6, pp. 549-559.

Raghavendra, R. et al, 2008. No power struggles: Coordinated multi-level power management for the data center. *In Proceedings of the 13th International Conference on Architectural Support for Programming Languages (ASPLOS'08).* New York, NY, pp. 48-59.