A software architecture for the analysis of large sets of data streams in cloud infrastructures

Mauro Andreolini, Michele Colajanni, Stefania Tosi
Dept. of Information Engineering
University of Modena and Reggio Emilia
Via Vignolese, 905/b - 41125 Modena, Italy
Email: {mauro.andreolini,michele.colajanni,stefania.tosi}@unimore.it

***Abstract*—System management algorithms in private and public cloud infrastructures have to work with literally thousands of data streams generated from resource, application and event monitors. This cloud context opens two novel issues that we address in this paper: how to design a software architecture that is able to gather and analyze all information within real-time constraints; how it is possible to reduce the analysis of the huge collected data set to the investigation of a reduced set of relevant information. The application of the proposed architecture is based on the most advanced software components, and is oriented to the classification of the statistical behavior of servers and to the analysis of significant state changes. These results guide model-driven management systems to investigate only relevant servers and to apply suitable decision models considering the deterministic or non-deterministic nature of server behaviors.**

***Keywords*-cloud computing; monitoring; distributed architecture; stream processing**

I. Introduction

Data centers for cloud services are characterized by a huge number of hardware resources (processors, memories, storage elements, virtual machines) and software components (applications, business processes) that interact in unpredictable ways. System management algorithms supporting these infrastructures must be able to prevent performance degradations, unavailability, and energy waste. They should be able to operate at different time scales (from seconds to days) and should support prompt reconfigurations motivated by changes in system, client, and business policies.

The monitors installed in cloud infrastructures produce thousands and thousands of heterogeneous data streams at the level of hardware resources, applications, events, and services [1]. We should also consider that some data streams may contain missing or faulty measures. Hence, any management decision needs software and modeling supports that are able to gather these heterogeneous data streams, to filter and aggregate multiple flows in order to extract from a multitude of raw performance measures those really relevant for a given business or system objective.

In this paper, we propose a novel architecture that supports models and methodologies for the efficient system management of large enterprise data centers offering external services at different levels (infrastructure and platform) as in a cloud-based infrastructure. The monitoring and analyzer components for management of cloud-based infrastructures are influenced by two main factors: *scalability* for increasing numbers of hardware and software components; *reliability* of the processes supporting collection and analysis operations. Both issues are addressed through the deployment of a highly distributed and redundant infrastructure that integrates several state-of-the-art software frameworks. Our design addresses the scalability challenge in several ways. The system is logically divided into several subsystems according to different spatial and time spans. 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 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. We show that the proposed approach is able to operate at different time scales and at different space scales (from single resources to servers to the entire cluster). It is able to reduce the dimension of the problem by discarding data streams that are negligible in terms of system management, and to classify the most relevant data streams in two classes: deterministic and non-deterministic. This classification allows a model-driven management system to apply the right model to the relevant data set and to receive just the signals about the most significant changes in the stochastic evolution of the server behaviors. It is important to observe that in this paper we consider one data center of a cloud infrastructure and we leave to future work the extension to geographically dispersed data clusters.

The paper is organized as follows. Section II introduces the on-line distributed analysis framework and its main components. Section III provides an analysis of different linear and non linear models to resources operating at medium-term time scales. Section IV discusses related work in the area of large-scale system monitoring. Section V concludes the paper with some final remarks.

II. A HIGHLY DISTRIBUTED ARCHITECTURE FOR SUPPORTING SYSTEM MANAGEMENT

High level architecture. High scalability and availability are the main goals driving the design of the proposed architecture for system monitoring and analysis that must support system management decisions. These goals are achieved through the deployment of a highly distributed and redundant infrastructure that is outlined in Figure 1. Although in many data centers, it is possible to use a single data source for the entire system, for the sake of scalability we decide to use several databases from the beginning of the design architecture. We assume to have one distributed storage for the collected samples and another distributed storage for longer term information for each set of resources that we denote as a cluster. The information stored in these databases is related to the status of the cluster (underutilized, healthy, loaded), to resource utilization and availability. The reason behind this choice is straightforward: we believe that it is easier to process system-level indications and trends from several, independent sources rather than from a unified central database. Decoupling cluster information also helps in identifying bottlenecks in a faster and clearer way.

Hadoop is chosen as the the development framework for scalable and reliable data collection and analysis, because it brings a dramatic scalability improvement with respect to traditional RDBMS-based data storage architectures [2] under the following conditions: (a) random data access patterns (stream processing proves much more efficient than individual read-write operations); (b) a non negligible fraction of the stored data is processed and stored concurrently (the Map-Reduce paradigm has been designed with this purpose in mind); (c) data reads are predominant with respect to data writes. These conditions are typically met by a large-scale monitor and analysis process that collects performance samples continuously and computes periodical component health status summaries.

At the lowest level, we have sets of hardware and software resources which can be associated to subnets, racks, distinct production areas, and logical or physical clusters. On each monitored node, a *collection agent* is responsible for extracting performance and utilization samples at regular time intervals. These samples are sent to the *distributed cluster data filter*, which performs preliminary validity checks on them and stores them into the *distributed sample storage*. From now on, the data is persistently stored and available as a *(key, value)* pairs, where *key* is a unique identifier of a measure and *value* is usually a tuple of values describing it (e.g., timestamp, host name, service/process, performance index, actual value).

The *distributed cluster analyzer* periodically extracts all the (key, value) pairs related to the resources in the cluster. It then performs more complex actions: identification of the relevant components in the system through the Principal Component Analysis technique, trend analysis through linear and non-linear timeseries aggregation models, anomaly detection, capacity planning. The goal of these scripts is to provide a "reduced view" of the entire cluster by discarding those data streams that are negligible in terms of system management, and to classify the most relevant data streams in two classes: deterministic and non-deterministic. This classification allows a model-driven management system to apply the right model to the relevant data set and to receive just the signals about the most significant changes in the stochastic evolution of the server behavior. The invocation frequency of the distributed cluster analyzer is higher than that of collection agent (hours to weeks), and depends largely on the time-scale of the associated management task. The (key, value) pairs resulting from the analysis are stored into the permanent *distributed analysis storage*.

The role of the *distributed system analyzer* is similar to that of the distributed cluster analyzer. It takes the (key, value) pairs from each cluster's distributed analysis storage and produces a global representation of the data center reduced in terms of data streams.

The collection agent. The collection agent is shown in Figure 2a. Each monitored resource has associated a *probe* process that collects a set of performance indexes, such as utilization, response time, throughput, at different time intervals (our considered sampling intervals are the order of one minute, but different time scales can be considered). All collected performance samples are stored and processed efficiently by the Chukwa monitoring subsystem. Chukwa is a highly scalable data collection and reporting system which has recently become part of the Hadoop framework. It has been designed to mine efficiently log data and to discover trends in large-scale systems. It can also be used to stream process data produced through live probes. Chukwa delegates the collection process to one single agent process per host, which collects data from several, specialized adaptor processes that act as an interface with the monitoring probes because each probe has its own adaptor. It then emits data periodically (if necessary, even in the order of seconds). We implemented several probes for the most popular system monitoring tools (vmstat, pidstat, sar, XenMon). We also configured the *ExecAdaptor* module to execute these probes at regular intervals of time.

The distributed cluster data filter. The data collected by the Chukwa agent is sent to the distributed cluster data filter shown in Figure 2b, where it is received by a Chukwa *collector* process. The collector is designed to scale up to several hundred distinct agents, and it is written to stable storage. It is important to remark that data is not written continuously but it is packed in large chunks (called *sink files*) and marked for later processing. These types of operations dramatically improve the throughput of the storage subsystem. Indeed, the number of write operations drops from one per adaptor per machine per unit time to

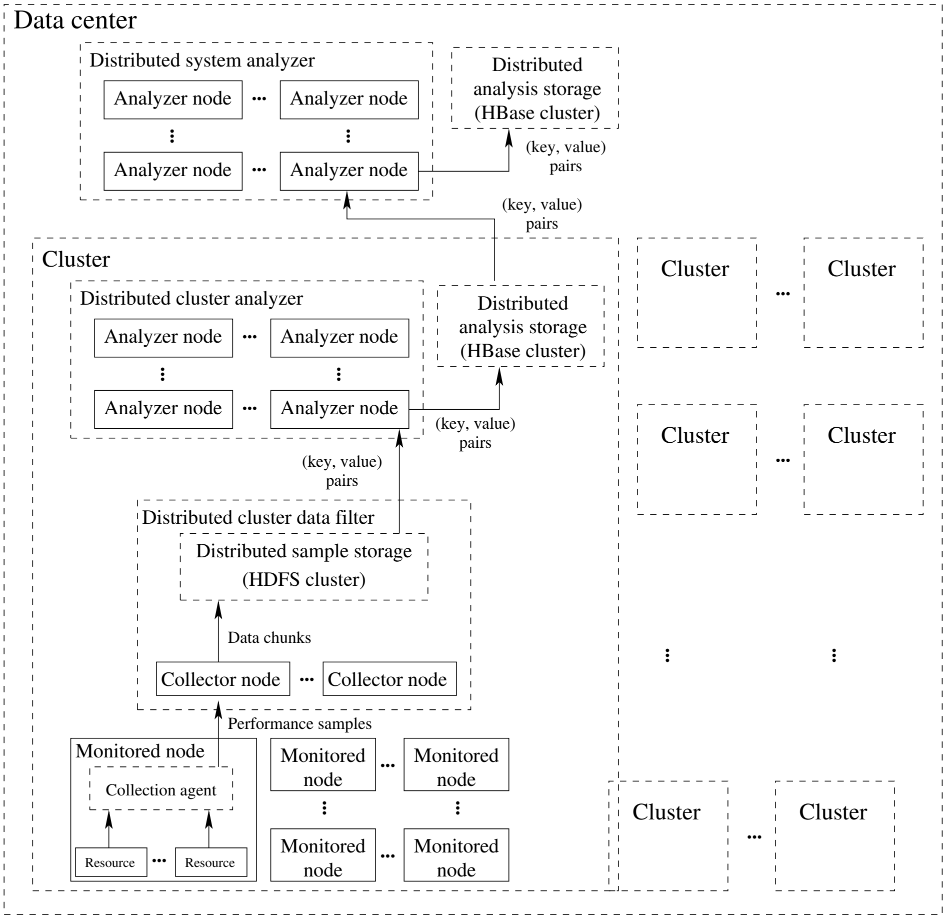
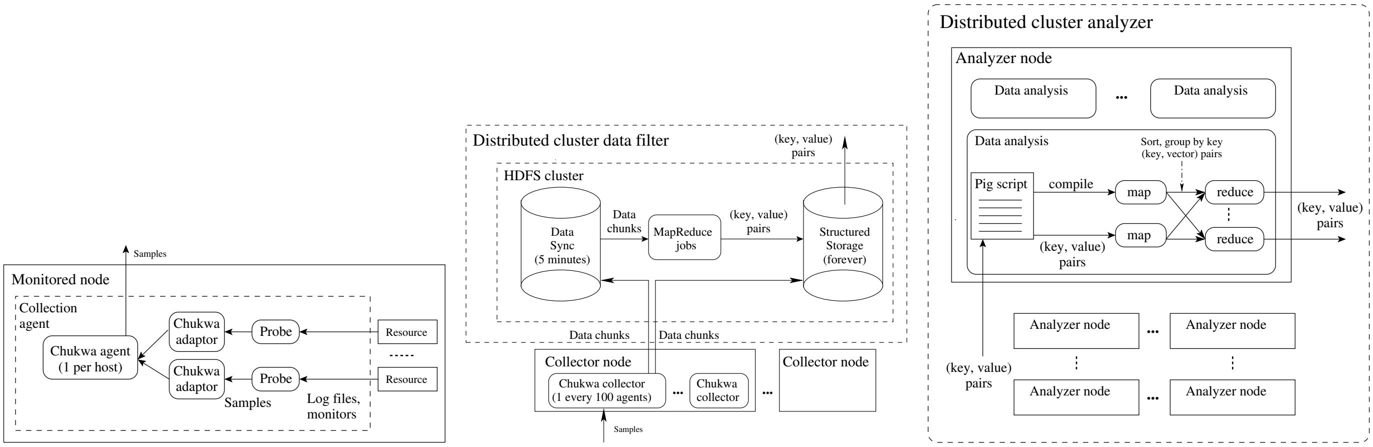


Figure 1. High level architecture



(a) The collection agent

(b) The distributed cluster data filter

(c) The distributed cluster analyzer

Figure 2. Detailed design of the components

a handful per cluster. The data chunks can be archived directly into the Hadoop Distributed File System (HDFS) without any modification, or pre-processed into sequences of (key, value) pairs through multiple map-reduce jobs [3]. HDFS is designed to run on commodity hardware, is highly fault tolerant, provides high throughput access to large data sets and makes stream batch processing very easy. However, these positive features come at one cost: low-latency access to single data structures is more difficult and slower than in the traditional RDBMS-based schemes. This fact does not represent a drawback in our architecture, for two main reasons: (a) we are not interested in very high sampling frequencies (once every minute is fine); (b) at this level, we are interested in scalable, batch checking and storing of a high volume of performance samples. With these premises in mind, we integrate the Chukwa framework with a set of custom map-reduce scripts that check the validity of the performance samples, and flag them appropriately. The goal behind these scripts is to verify that the probes are collecting meaningful data. Some examples of checks include: presence/absence of samples, belonging to a specified range, presence of a series of null samples.

The distributed cluster analyzer. At the heart of the distributed cluster analyzer shown in Figure 2c there is a set of analyzer nodes which execute Pig scripts. Pig [4] raises the level of abstraction for processing large datasets. With map-reduce, there is a map function and there is a reduce function, and working out how to fit the data processing into this pattern, which often requires multiple map-reduce stages, can be a challenge. With Pig the data structures are much richer, typically being multivalued and nested; the set of transformations that can be applied to the data are much more powerful (for example, they include joins, which are not easy to implement in bare map-reduce scripts). Each Pig script is compiled into a series of equivalent map-reduce scripts that process the input data and write the results in a parallel way. We let Pig operate in "Hadoop mode" by fetching data directly from the files in the distributed cluster data filter, through the `LOAD` statement. Each map script selects a subset of the data produced by Chukwa and feeds it to the map-reduce subsystem. Here, values are grouped by key, sorted by key and sent to the reduce scripts, that produce the desired output. We implemented scripts to aggregate data both temporally and spatially. Further analyses include anomaly detections, trend analyses and supports for capacity planning on a longer time scale.

The distributed analysis storage. The output of these map-reduce scripts is written to a HBase cluster. Apache HBase is a distributed column-oriented database built on top of HDFS. HBase is the Hadoop application to use when an application requires real-time read/write random-access to very large datasets. HBase is built from the ground-up to scale linearly just by adding nodes; it is not relational and does not support SQL, but thanks to the proper space management properties, it is able to surpass a traditional RDBMS-based system by hosting very large and sparsely populated tables on clusters implemented on commodity hardware. In our architecture, HBase fits perfectly the role of an intermediary storage between the local analyzer that produces information periodically, and the distributed system analyzer that consumes the produced information.

The distributed system analyzer. The distributed system analyzer fetches data from several distributed analysis storage clusters and processes in a parallel way the health status of each cluster in order to produce a few figures of merit that show the health status of the entire system. The architecture of the distributed system analyzes is almost identical to that of any distributed cluster analyzer. It differs only in the Pig scripts, which elaborate more sophisticated, system-wide models aimed at identifying the most relevant resources. Example scripts implemented at this level include, among the others, longer term predictions, Principal Component Analysis, capacity planning.

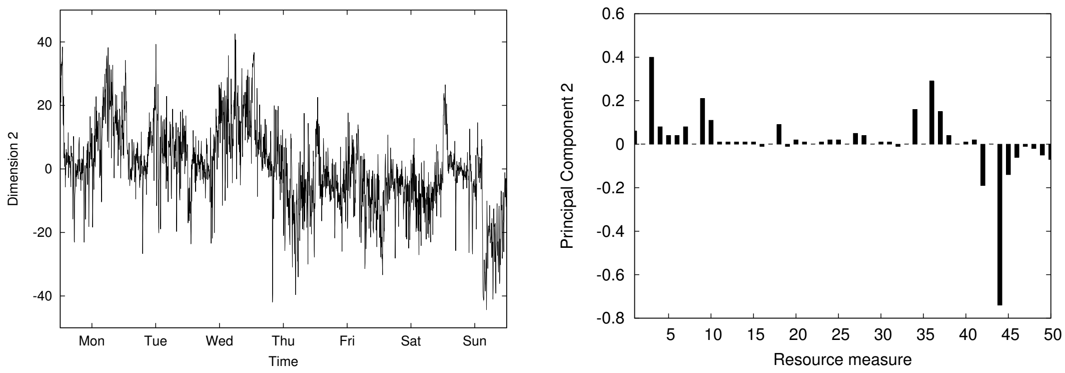
III. Statistical analysis

In this section, we present the results of preliminary monitoring on a data center of 50 nodes. We show that in the considered real data center characterized by a high number of heterogeneous hardware and software components (processors, memories, storage elements, applications) the proposed approach is able to operate at different time scales, to reduce the system dimensionality by discarding low impact data streams and passing to a model-driven management system only relevant data streams, to classify the main time series with the final goal of applying the right models for supporting the decision, to signal significant changes in the evolution of the resources and overall system.

The first important goal is to evaluate the impact of each data stream on the overall system activity so to distinguish important and negligible sources of information. Eliminating negligible data sets diminishes the complexity of managing the system and provides a robust solution able to work at the different management time spans (short, medium, long term). The proposed methodology is based on the Principal Component Analysis that allows us to reduce the dimensionality of the problem. Figure 3 shows an example of a dimension of our data set and its corresponding principal component. The dimension in Figure 3(a) captures a pattern of the temporal variation common to the set of data streams referring to CPU utilizations of different servers. The extent to which this temporal pattern is present in each CPU utilization of the monitored servers is represented by the entries shown in Figure 3(b); we infer from the strongest peak that the dimension is most present in the server 44.

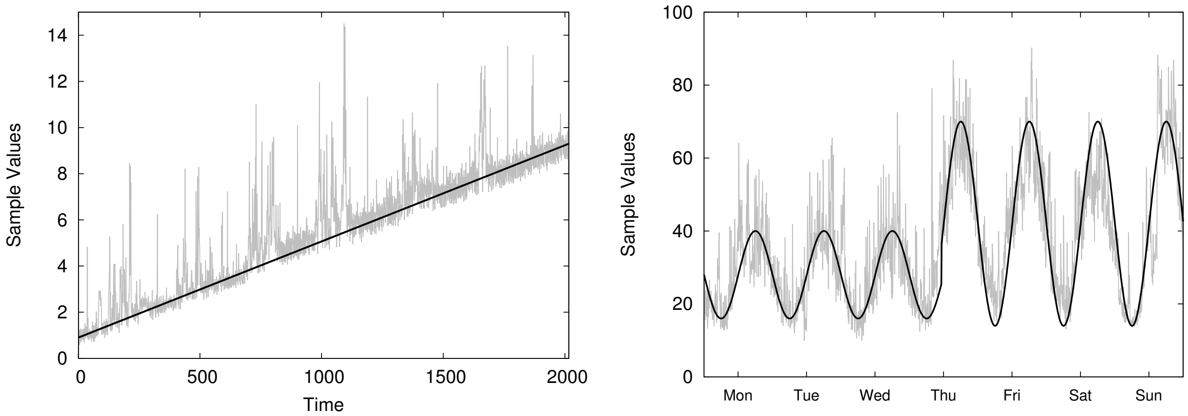
The classification of data streams on the basis of their deterministic or non-deterministic behavior is based on the evaluation of the autocorrelation function of each relevant dimension [5]. If a data stream contains trends and seasonal patterns, then the correlogram exhibits high values and an oscillation at the same frequency. Knowing the nature of the data streams guides the choice of suitable management algorithms and prevents the analysis of those components that cannot be modeled or do not provide relevant information to the overall system state.

Let us outline some statistical characteristics to motivate why only deterministic data streams can be used for data



(a) Dimension **(b) Principal component**

Figure 3. Example of dimension and corresponding principal component.



(a) Trend data stream **(b) Seasonal data stream**

Figure 4. Examples of deterministic data streams.

center management decisions at any time scale and the need of multiple filtering techniques to extrapolate meaningful information from non-deterministic data streams. Most deterministic patterns in data stream analysis can be described in terms of trend and seasonality. The *trend* represents a general systematic linear or (most often) non-linear component that changes over time and does not repeat or at least does not repeat within the time range captured by the data. When data streams are monitored for a sufficiently long time period, it is often the case that these series display *seasonal* patterns. A seasonal pattern has similar nature as a trend component, but it repeats itself in systematic intervals over time. This is typically the case of Internet-based services, where system measures increase during diurnal activities and decrease during the night or weekend. In Figure 4 (a) we give an example of an upward linear trend data stream. Figure 4 (b) shows a typical example of a time series displaying seasonality. Both trend and seasonal components can be predicted through parametric techniques that learn a model from the past and reproduce it in the future for management purposes.

On the other hand, non-deterministic data streams have stochastic behavior and therefore we do not deal with only one possible reality of how the data stream might evolve under time. In a stochastic data stream there is some indeterminacy in its future evolution is described by probability distributions. All non-deterministic data streams are mainly driven by stochastic errors, that are deviations of the data stream from the expected systematic pattern. Random errors of data sets coming from process monitoring typically include a noteworthy *spike component*. It collects short-lived bursts departing from data stream mean in correspondence of unexpected and uncommon events in the sampled resource measure. Also stochastic *noises* are deviations of the data stream from the expected deterministic pattern. In computing and information contexts, noise is typically considered unwanted data without meaning, that is, data that is not used to transmit a signal, but it is simply produced as an unwanted by-product of other activities.

IV. Related work

Monitoring systems. Existing monitoring systems can be divided into two distinct categories: log collection frameworks and lossy, low data-rate real-time systems for machine telemetry. Probably the oldest log collection framework is syslog [6], which supports streaming logs across the network. However, syslog has serious defects: no clear solution to the discovery, load balancing, or failure handing problems. The rise of datacenter-scale distributed systems has made these problems particularly glaring, and in recent years several newer systems have been developed. Scribe [7] apparently solves some of these problems, but unfortunately, no details about its architecture seem to have been published to date. Two popular network monitoring systems are as Nagios [8] and Ganglia [9]. They are capable of collecting and storing substantial volumes of metrics data. While Nagios has a centralized architecture, Ganglia is hierarchical and distributed, thus making it a good candidate for our collection subsystem. Unfortunately, Ganglia is oriented to numeric time-series data, and provides little to no support to log mining. Chukwa has one limitation: it cannot easily instrumented to do sophisticated processing on the collected data, due to lack of rich data structures that are available in the Pig Latin language. Our approach, which to the best of our knowledge is one of the first in its field, tries to solve this limitation with a hierarchical architecture, which simplifies the extraction of system-level indications and trends from several, independent sources.

Identification of relevant data streams. 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. Techniques such as the multi-variate analysis and the Principal Component Analysis are effective in capturing the salient features of a subsystem's internal state, thus drastically cutting the amount of data used to perform decisions. 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. To the best of our knowledge, this paper is one of the first attempts at reducing the system state information through these techniques.

V. Conclusions

The cloud computing paradigm allows us to keep up with requirements behind complex applications, massive data growth, sophisticated business models, but it requires novel approaches to support model-driven resource management systems. In this paper, we have proposed a scalable and reliable architecture that is based on highly distributed technologies operating at computational and data access level. These choices are mandatory when you have to support gathering and analysis operations of huge numbers of data streams coming from cloud system monitors. The proposed architecture is already integrated with on-line analyzers working at different temporal scales. Preliminary experiments include the identification of the most relevant data streams for system management purposes, and the possibility of distinguishing between deterministic and non-deterministic data sets. All these operations for hundreds of data streams are carried out within real-time constraints in the order of few minutes thus demonstrating that huge margins of improvement are feasible.

Acknowledgment

The authors acknowledge the support of the MIUR-PRIN project AUTOSEC "Autonomic Security".

References

[1] V. Soundararajan and K. Govil, "Challenges in building scalable virtualized datacenter management," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 95–102, December 2010. [Online]. Available: http://doi.acm.org/10.1145/1899928.1899941

[2] J.-S. Leu, Y.-S. Yee, and W.-L. Chen, "Comparison of map-reduce and sql on large-scale data processing," *Parallel and Distributed Processing with Applications, International Symposium on*, vol. 0, pp. 244–248, 2010.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150. [Online]. Available: http://www.usenix.org/events/osdi04/tech/dean.html

[4] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: a not-so-foreign language for data processing," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08. New York, NY, USA: ACM, 2008, pp. 1099–1110. [Online]. Available: http://doi.acm.org/10.1145/1376616.1376726

[5] M. Andreolini, S. Casolari, and M. Colajanni, "Models and framework for supporting runtime decisions in web-based systems," *ACM Trans. Web*, vol. 2, pp. 17:1–17:43, July 2008. [Online]. Available: http://doi.acm.org/10.1145/1377488.1377491

[6] "Guide to computer security log management," 2006, – http://csrc.nist.gov/publications/nistpubs/800-92/SP800-92.pdf.

[7] "A scalable streaming log aggregator," 2008, – https://github.com/facebook/scribe.

[8] E. Imamagic and D. Dobrenic, "Grid infrastructure monitoring system based on nagios," in *Proceedings of the 2007 workshop on Grid monitoring*, ser. GMW '07. New York, NY, USA: ACM, 2007, pp. 23–28. [Online]. Available: http://doi.acm.org/10.1145/1272680.1272685

[9] F. D. Sacerdoti, M. J. Katz, M. L. Massie, and D. E. Culler, "Wide area cluster monitoring with ganglia," *Cluster Computing, IEEE International Conference on*, vol. 0, p. 289, 2003.