ABSTRACT
In cloud environments, resources are being shared between different organizations, and service providers of cloud resources require an efficient mechanism for metering resource usage to be able to charge customers fairly, based on actual resource utilization rather than resource rental time.

One of the main challenges is the incompatibility of different usage data formats generated from different software and cloud resources, which need to be correlated and processed seamlessly to achieve a streamline metering process. In this paper, an architecture is proposed for collecting usage data and presenting it in a portable interoperable format. Usage data is presented in Predictive Model Markup Language PMML, where data models exchange is adopted rather than the traditional data unification approaches. An extensible interpreted Cloud Metering Markup Language, CMML, is proposed for distributed usage data collection and processing. Finally, a prototype for CPU usage correlation is presented to demonstrate the applicability of the framework.

KEY WORDS
Data mining, cloud computing, event collection, data modeling, extensible markup languages, PMML, metering, event processing, event correlation.

1 Introduction
Cloud computing is the so-called dream of providing computing resources as a service, following on demand utility computing concepts, where resources are consolidated and shared by different organizations allowing for maximum performance. In this context, a cloud environment is a very large distributed system running heterogeneous software and utilizing a wide range of hybrid types of resources. A producer-consumer model is adopted, especially in public cloud environments, where a service provider, the producer, avails a pool of resources, and an organization, the consumer, utilizes the resources through resource sharing and isolation techniques. A metering mechanism is very crucial in cloud environments, where fair charge-back mechanisms can be achieved. Moreover, predictive planning for resource usage and hence planning for resource availability is an important application for the service provider to be able to plan for the consumers resource demands.

Cloud computing environments are service oriented distributed environments, and hence collecting usage data for processing from system dump areas, log files, and event repositories has always been a challenging task to achieve. The two main challenges in the cloud metering domain are data transport and data interoperability. To be able to evaluate the resource usage by a specific organization, usage data of all applications acquired by that specific organization needs to be collected, preprocessed, and relevant data should be transported to common locations for correlation. Such applications, as well as cloud resource usage data are designed by different software vendors and were not designed initially to be compatible, further more, such metering raw data are collected from different architectural levels which are not compatible by nature. Hence, in addition to the need of an efficient data transport mechanism to transport related usage data to the relevant location for processing, usage data needs to be presented in a compatible format for processing.

In this paper, we are more concerned with the second challenge, which is the interoperability of the metering data format. The usage data generated from different cloud resources and applications that can coexist across a cloud environment is very analogous to the type of data being processed in a mining environment. The two aspects of similarity are data size and data incompatibility which raises the issue of interoperability. The data mining field has a lot of techniques for processing data, and the motivation behind this research is to try to apply some of those techniques in the domain of cloud resource metering and research the benefits of utilizing such techniques. In our research, we will try to deploy the PMML modeling language to build a cloud resource usage data model.

An architecture for collecting metering data from dif-
different sources with different initial raw format is proposed. Unlike the traditional custom data unification approaches adopted in the area of event correlation, we will adopt a model exchange mechanism where the metering data are presented in a portable data mining model. The power of representing the usage data in a data model is that data are coupled with data dictionary for variables definition, data transformation dictionary, and mining schema defining the type of mining algorithm that will be applied on the data.

An extensible markup interpreted metering language is introduced as part of the framework, to collect data from different sources and generate different PMML models. The PMML model will then be fed into a PMML-enabled data mining software for processing. A prototype is developed, as a proof of concept, to model the CPU usage of cloud nodes, and present the usage data in a PMML Association Model. Moreover, an association engine based on Apropri is built to correlate different running processes and identify the relation between them.

This paper is organized as follows: the needed background is established in section 2. A summary of related work is presented in section 3. The proposed architecture is presented in section 4. The prototype is presented in section 5. Experiments, results, and observations are presented in section 6. Analysis and discussion of the results and observations are presented in section 7. In section 8, we present conclusions and possible future work.

2 Background

In this section we will establish the needed background to be able to describe and explain different subsystems of the proposed framework. The background will cover three main areas, which are cloud computing, log data collection and unification, and data modeling based on Predictive Model Markup Language PMML.

2.1 Cloud Computing

Cloud computing environments lack a precise definition as a consequence for the lack of a strict standard for cloud computing. What we are facing here is more precisely a lack of a unified taxonomy of what cloud computing environments are. Nevertheless, a lot of scattered definitions do exist that describe cloud computing environments in a general approach through enumerating the cloud computing environments characteristics, benefits, and problems.

The most important characteristic of cloud environments, that all the references agree on [2][18][9][21], is that cloud environments share their consolidated resources as a service. In cloud environments, everything is as a service X-as-a-Service; Software as a Service SaaS, Infrastructure as a Service IaaS, Platform as a Service PaaS, etc. Cloud computing environments are the descendants of grid computing environments, it is viewed as the realization of the dream of utility computing; which means that computational power can be leased as a public utility that can be shared on demand and released for others when available.

Cloud computing environments are by nature built on top of clustered architectures of commodity computers that are connected via very high speed networks. Hardware resources are being consolidated to be viewed as a large environment representing a pool of resources that can be leased on demand. Virtualization is a key feature in cloud environments that enable two important features which are isolation and ad-hoc scalability.

A cloud middleware is basically responsible for managing consolidated resources and availing them to be shared transparently irrespective of the resource location. Usage optimization is a key responsibility for the cloud middleware, which is availing the right resource for the corresponding tasks in terms of location, data accessibility, and other resources dependability.

In summary, regardless of the fact that there is no standard and firm taxonomy for cloud environments, cloud computing is all about utility computing, which is encapsulating computing resources and providing them to the end user as a service in a transparent manner in terms of location and resources; the end user should not know where the services are running and which resources have been dedicated to serve them. In that manner, cloud computing is a very complex distributed system that utilizes a set of distributed resources to provide various heterogeneous types of services in a transparent manner providing features like reliability, fault tolerance, expandability, and extendibility, better resource utilization, better return on investment ROI, in addition to all the features provided by distributed systems. Moreover, virtualization is a key feature that is being utilized by cloud environments to provide infrastructure services as virtualized resources, targeting resource isolation, better utilization, and ad-hoc scalability. Figure 1 best describes a cloud node with all of its components. Notice that virtual machines running in the userspace have a similar architecture of the node layout without the upper virtualization layer.

2.2 Log Collection and Unification

A distributed system is a set of independent computers that appear to its users as a single coherent system, and the middleware of the distributed system is responsible for managing the collaboration of the applications, running on such an environment, to serve a common target. Distributed systems essentially utilize an underlying clustered architecture, consolidating resources to gain processing power, as well as fault tolerance, extendibility, expandability, and scalability [19]; a distributed environment providing SaaS is built up of hybrid types of software including, and not limited to firewalls, dispatches, web servers, legacy software, ERPs, business logic, databases, and more.

On each node, a software instance runs to carry out a task that complements the execution of the whole system, utilizing underlying resources, either physical or vir-
To be able to meter and monitor an application, event recording mechanisms must be adopted. Each application generates logs, a means of event recording, that indicates the operations being carried out by the application. Applications usually generate different levels of log details based on the runtime configuration of the application, and there is a tradeoff between performance and the details of the log generated [10][20]. Other means of usage data can be generated by system-wide continuous profilers which generate valuable usage data that can be used in distributed systems metering [17][1]. Moreover, special techniques are applied to collect profiling data from within virtual machines, taking into consideration the virtual timing problems as described in [5].

Techniques of log collection and unification are adopted such as in [10][20] to efficiently collect events data and apply preprocessing phases whenever possible before data transport, to save network bandwidth. A distributed event processing and correlation is adopted like in [20] to achieve that. Since logs coming from different applications, and different system layers need to be correlated and unified, such logs need to be compatible. Post the unification level, all logs collected from different applications should be presented in a unified format. Log parsing, conversion, and unification need to be applied for the correlation engine to be able to process and correlate the events. Although multiple standards are available by Common Event Expression (CEE), and there are recommendations for adopting extensible logging formats like XML and data definition schemas, yet it is very difficult to impose a standardized log format on software vendors[10].

2.3 PMML

Data used in data mining is collected from different sources in different amounts and at different sampling rates. More importantly, in very large mining environments, data are collected from different sources and different applications whose developers and inventors had no prior notion about how the data will be mined and what applications the data will be fed into.

Normally, data that is used for data mining is generated in what we call log files. Log file formats unification is the main obstacle in the data mining preprocessing phase. Theoretically, if all applications can read and write a unified data format, that would solve the problem radically. In reality, this is not the case, different applications generate data and logs in their own format. Moreover, a lot of drawbacks exist with respect to data logging such as inconsistency, lack of standard, and that logs were meant initially to be human readable [10]. Most of the work done on processing data logs, depends on that the data are provided in a standard format [13][15][20], to be able to be read and processed reliably, leaving this task to be done manually as part of the deployment process, yet this is one of the most difficult and time consuming tasks, and needs to be tailored per application and it’s requirements.

We consider the data format representation as one of the core aspects when it comes to introducing data to data mining engines, yet in data mining environments the problem is more radical as not only does the data need to be unified but rather the whole mining model needs to be unified to be able to process through different hybrid subsystems, that all collaborate in a comprehensive mining environment. This is considered as the default in large scale data mining environments that operate on the basis of pipeline schemes.

Extensible Markup Languages were invented for this specific purpose, to provide a standard interface between different systems that have internal non-standard data structures and need to exchange data with a predefined standard format. Based on the extensibility of XML, Predictive Model Markup Language PMML [7] was introduced by the Data Mining Group DMG as a derivative of XML to represent, not only data, but also to represent the data model.
as a whole. What is meant by the model is the operation or algorithm type that will be applied on the data together with the data laid out in a structure most suitable for processing. Data mining algorithms that can be applied here are association, clustering, classification, regression, ...etc [6]. For example, an association data model can be represented by a PMML file and can be fed into two different applications that perform two different algorithms and the output PMML file will be of the same expected format.

Practically speaking, a PMML file is an XML file with a schema that defines all the supported data mining algorithms and data structures that can be supported by PMML. The whole trick is in the definition of the XML schema file, and since XML is extensible by nature, the schema can be extended to support more data mining models [22].

PMML has a very well established structure that is defined within the standard. A typical PMML file consists of a data dictionary, mining schema, data transformation elements, and model elements. The data dictionary basically describes the input to the model, as well as validation schemas. The mining schema describes the usage type for each input field. The data transformation elements, defines the mechanism that will be used to preprocess the input data such as normalization, discretization, aggregation, and mapping of values. Finally, the model elements describe the algorithm type that will be applied on the data with all the parameters needed to run such an algorithm [16][6].

To summarize, PMML targets model sharing and interoperability as a priority. Its main objective is to make multiple heterogeneous applications collaborate on solving a data mining problem, by exchanging their partial results in a collaborative mode of operation for solving complex problems. The solution of such complex problems, if presented in PMML, can also be used as an input for other data mining environment. Hence, it helps divide a mining problem into small independent tasks that can be run by different applications and can communicate together, which is best suited to run in a distributed cloud environment.

3 Related Work

3.1 Log Collection and Unification

The guidelines of logging in a SaaS-based cloud environment are discussed in [14]. Important argumentative questions are raised like When To Log, How to Log, and What to log. The challenges of log analysis are enumerated as:

1. Decentralization
2. Volatility
3. Multiple tiers and layers
4. Archival and retention
5. Accessibility
6. Non-existence of logs
7. Absence of critical information
8. Non-compatible and random log formats as it is very crucial in distributed environments to be able to assess logs generated from different systems, which usually have different incompatible log formats. This problem arises from the fact that log generators highest priority is that they are human readable, rather than being machine process-able.

The solutions recommended in [14] to solve the above problems are:

1. Centralization of all logs
2. Scalable log storage
3. Fast data access and retrieval
4. Support for any log format
5. Running data analysis jobs
6. Retention of log records
7. Archival of old logs and restoring on demand
8. Segregated data access through access control
9. Preservation of log integrity to relate logs from different systems in a distributed environment, integrity is a very important issue as logs need not to be modified and accessed in read-only mode.

The work in [10] is concerned with logging systems in distributed environments, mainly grid and cloud environments. The issue of logs incomparability and lack of interoperability is the main motive behind introducing a new solution for automatic LOG evaluation and automation. The solution proposed is based on XML representation of logs. The most important issue tackled is relating different logging events coming from different environments, which are deployed distributedly in GRID or cloud environments.

The concept of event correlation in a distributed environment that hosts different application types in different application layers is being introduced in [20], and a Run Time Correlation Engine (RTCE) was presented to provide an architecture for distributed log files correlation in a scalable manner. In the same paper [20], a solution is proposed for scaling the RTCE to run in a cloud environment which has more applications and generates more events than a normal distributed system. The solution proposed is to provide clustered RTCEs with a dispatching engine for load balancing that dispatches events coming from monitoring agents to be assessed distributedly. The results are then submitted to a centralized web server for inquiry and reporting. Along the same path, the dispatcher layer is being extended by providing a cluster of dispatchers, dividing the monitoring agents into groups, and assigning each group to
a dispatcher. The solution is characterized by being scalable and extendable, as the number of events increases, a corresponding increase in the number of RTCEs and dispatchers will be needed.

3.2 Interoperability and PMML

Zemintis Plugin [4] is a PMML converter that enables converting PMML models to proprietary formats. The paper presented the integration of Zemintis with GreenPlum. Zemintis Universal Plugin acts as a PMML gateway that converts PMML models to SQL user defined functions (UDFs), that can be executed within SQL statements. The SQL statements, which invokes the PMML UDFs, can then be executed in parallel on GreenPlum data segments. All PMML model validation is done within the plugin to ensure that the model is syntactically valid. Zemintis is also designed to export and import neural networks and support vector machines in PMML format.[6]

PDM, Planning for Data Mining, is presented in [16] as a data mining tool. PDM is designed to generate plans, which are data mining knowledge flows, to solve data mining tasks. PMML is used to define data mining tasks, which allows for openness and interoperability to produce a model that is portable on multiple systems. A data mining tool called WEKA [8] is used through which knowledge flow is described, and the overall system adopts a workflow model of execution, where every stage of execution feeds into another. The overall PDM system is a very good demonstration of huge data mining systems that span different technologies as well as data representations, and being able to execute through PMML interoperability approaches.

Another transformation model is suggested in [12] to build generic transformation modules based on PMML to integrate different data mining algorithms without rewriting code. The needed functionality is implemented through software adapters that convert PMML models to algorithm specific formats.

SPMML is introduced in [22] as a PMML extension to present spatial data models. SPMML is used to integrate spatial data mining and GIS systems and is based on spatial data types, spatial data transformation, and spatial data mining models. The spatial data model of SPMML inherits from the PMML general built-in model rules. For example, the SPMML Co-Location rule extends the PMML Association Rule. In that sense SPMML is a superset of PMML, and hence SPMML can be transformed to PMML by placing the definition of spatial data types, spatial data transformations, and spatial data models in the PMML extension part. The whole idea is that the spatial data mining engine generates its results in SPMML format, and integrates the results with GIS viewers equipped with XML parsers.

Healthy Housing Evaluation and Rules Discovery Decision Support System is another system that illustrates the use of PMML in data mining environments that are built of heterogeneous subsystems that run on a clustered environment. The whole model is based on decision trees, and a PMML representation is designed to represent the model. PMML files can be imported by other data mining software such as SPSS, WEKA, Oracle, ...etc.[3]

4 Proposed System Architecture For Mining Cloud Usage Data

The architecture that we are proposing is based on three main subsystems. The Cloud Metering Markup Language CMML interpreter, PMML Merger, and data mining subsystem. The diagram in Figure 2 describes the framework and its subsystems. The raw usage data and a CMML script is fed into the CMML interpreter which produces a PMML model based on the CMML script. The PMML files generated are collected and merged selectively, and finally PMML files are fed into the corresponding mining algorithm to generate the needed usage analysis.

4.1 CMML

We propose the Cloud Metering Markup Language CMML as an interpreted language based on XML designed for metering data collection and processing. Although the CMML language is in its early stages of development, yet its parsing utilities are strong enough to parse usage data and log files efficiently. The constructs of the language are defined in XML, and commands of the language are presented as XML tags. The CMML interpreter makes it very easy to write plugins for different usage data formats. Two main benefits for CMML other than its strong parsing constructs are: 1) The ability of recursive execution allowing embedded construct tags inside others. 2) The ability to embed native C code to be compiled on the fly as shared libraries and run on tokens generated from the CMML parsing engine. The flexibility of CMML allows customized parsing of complex data input formats and the generation of different formats which will be the PMML model in our case.

The sample code above was used in our prototype, that will be presented shortly, to generate PMML from the usage data generated by the UNIX pidstat command line.
Figure 3. CMML Code

As can be seen, the language XML-based and all the commands are represented by XML tags. The parameters for each command are passed as XML attributes. The language allows executing shell commands, embedded interpreted code like SED/AWK, and compiled languages code like C. The embedded code is compiled or interpreted on the fly and loaded into the current script execution through the dynamic shared loadable library mechanisms supported by the CMML interpreter.

As mentioned before CMML is in its early stages of development and a lot can be integrated into it to enable powerful parsing constructs. Concise code that can achieve a lot functionalities is one of the most important targets for CMML, as it is desirable that a lot of functionality be achieved with small amount of code.

Currently CMML supports the embedding of C code as well as SED/AWK, but it is intended to integrate more languages such as PHP, perl, JAVA, ...etc. CMML was designed initially to allow for integrating different code snippets within the same CMML script which is a powerful feature allowing to integrate ready made code already designed for parsing specific log formats, and hence correlation for different logs, generated by different sources, can be done within the same CMML script easily.

4.2 PMML Merger

Since the PMML model will be generated by the CMML interpreter on each node in the cloud presenting the usage of the nodes resources per a specific duration of time, we propose a PMML merger engine that will be essential to merge related PMML models. The merged PMML models will be used to mine the usage data based on the mining schema within the PMML model.

4.3 Mining Engine

A mining engine that can accept input in PMML format is needed to carry out the data mining part. In the proposed architecture, the mining engine is a plug-and-play component that can be integrated into the framework based on the requirements, with the only constraint of being PMML enabled. An example of mining engines are association engines which can associate and relate usage of different services on the cloud nodes and predict different levels of execution. Another example of mining engines is clustering, which can be used to cluster related resources to run together which can help a service provider to better manage the cloud environment resources, and to achieve reduction in the number of nodes running through what is called cloud over subscription and service capping[11]. A lot of mining engines do exist, and which engine is deployed depends on the application and the requirements of the target environment.

5 A Prototype For Mining CPU Utilization Data

A prototype to test the architecture proposed is setup to model the CPU utilization of a cloud environment as an association model. The target of the prototype is to be able to predict association relationships between related processes as well as the level of that relation.

Although the cloud resource usage model can be applied generically on any usage data across the cloud, for prototype feasibility issues we are going to concentrate on CPU usage of a specific set of processes, whose usage data are collected through OS usage collection tools such as TOP or SysStat, preprocessed, and converted to a PMML model.

The purpose of the model in our case study is to relate usage of different processes and find the processes that initiate change in usage of other processes. This will allow us to understand different applications behavior per en-
vironment, and consequently will allow predicting overall resources usage needs based on different users’ behavior.

In order to clarify the goal and the purpose of the CPU Utilization Model, lets illustrate with an example. Consider an environment that hosts a traditional web application which is based on open source middleware, where the Apache web server is its front-end, MySQL is the application backend database, and the application is written in any scripting language like Perl or PHP. In this case, requests coming from users to the web server will initiate execution threads for the applications deployed under the web server, and some of those threads will initiate database connections to the backend. In such a case, our model can detect the relation between the two middlewares, Apache and MySQL, and the level of utilization needed at different web usage traffic, initiated from the user.

Moreover, the model should be able to come up with such relations without being given definitions of the existence of those relations. So, for a pool of different processes, the model should identify related processes and the characteristics of the relation between those related processes, rather than depend on pre-established information about the existence of such relations, being passed to it as an input.

To be able to achieve such functionality, the PMML Association schema will be adopted to represent different processes usage through presenting usage snapshots as association itemsets. Since the CPU utilization per process is a continuous decimal value, discretization techniques will be applied to define ranges of CPU utilization to be presented as different items as per process.

We developed a tool, that we will refer to as top2pmml, to convert usage data to a PMML model. The top2pmml tool is basically a CMML script that is used to parse the output of the UNIX Top command and extract the CPU utilization of the user defined processes upon which the association will be applied. Figure 3 shows the raw top command data representation and the highlighted data columns that will be extracted.

As the CPU usage data generated by TOP command line UNIX tool is very granular as per process, the top2pmml will then group each process type, e.g HTTP, MySQL, ...etc, and will sum up all their usage to be presented once with the total amount of execution. Moreover, since the CPU utilization is a percentage decimal value, it is considered as a continuous value, and hence a discretization process will be applied by the top2pmml on the data by defining ranges of execution, which in our case will be increments of 10%; for example if the HTTP process utilization at a specific snapshot is 38.4%, it will be represented in the items and itemsets in the model as HTTP 40, and a process having a CPU utilization of 0% will not appear in the itemset of this specific transaction. The top2pmml will run at fixed predefined intervals, each of which will generate a snapshot of the CPU utilization, each of which is represented as an itemset. The PMML is then generated after parsing all the data. Figure 4 shows a snapshot of parts of the PMML being produced by the top2pmml tool. The association engine used is based on the famous Apropri Algorithm. The engine was written from scratch in C++, following the primitive basic Apropri algorithm, to be able to do some optimization based on the nature of the problem, and to be able to freely play around with the code. The main feature in our problem that provides the mentioned optimization to the algorithm, and is very well known and confirmed from the beginning, is that items of the same processes types and different CPU utilization cannot coexist in the same itemset; for example HTTP 10 and HTTP 30 cannot coexist in the same itemset. The impact of this kind of valid presumption restricts the ceiling of the size of the itemset to be equal to the number of processes types being associated and defined by the user. So instead of designing the algorithm to accommodate all possible combinations of the available items, which in this case will be very large, the problem domain size is reduced dramatically to the size of the available processes types. The impact of that is a boost in performance and a speed up in the overall association engine processing.

The diagram in Figure 5 best describes the overall process of the CPU Usage Association Model, and it’s phases of execution. It is very important to highlight the fact that the prototype presented is not the only way to model this problem, and that it is designed for demonstration purposes; meaning that, a lot of complexity can be added through introducing more combinations of parameters as we will discuss in the future work section later.

### 6 Experiment & Observations

The experiment performed to test the prototype and validate its results is run on an online environment that hosts traditional internet services, and hosts web applications that are based on Apache/PHP as a web frontend and MySQL as its backend database. The environment provides email services with most of its protocols where Exim is used as the SMTP agent for email transport, Dovecot is used for POP3, and SPAMD is used as a spam detection agent. Some com-
mand line applications are written in PHP, which perform scheduled tasks that need to run on regular schedules with the crontab scheduler daemon.

The experiment is designed to find the association of the different CPU utilizations for five main services: httpd, mysqld, php, exim, and spamd. These specific services were chosen because practically three of them are dependent and two of them are standalone. The httpd, and the php scheduled command line scripts both invoke mysql for processing database queries so it is expected that the association rules should detect this. On the other hand, although the exim and spamd intuitively appear to be related yet in reality this is not the case as we will see in the discussion section below.

Data collection was done once over a total period of one week, and the top2pmml tool was run on the overall data to generate the PMML association model once. The PMML model was used to run the association algorithm with different minimum support and confidence values to be able to assess the generated association rules with respect to changing the calibration variables.

Figure 6 shows some statistics about the nature of the raw data used in the experiment, for later comparison, with the predicted association rules. It is very important to highlight that the reason behind the maximum CPU utilization being close to 400% for the HTTP and the MySQL processes is that the CPU model used in the experiments is a multicore architecture CPU, the CPU can be visualized on the OS level as 4 Virtual CPUs, and since we are collecting data per service type, the service can have more than one process and not all of them are necessarily running on the same virtual CPU.

Figures 7 and 8 present the experiment setup environment and parameters followed by the results of the five runs of the experiment. The association algorithm was run on the PMML association model 5 times. We started with a minimum support and confidence that is equal to 0.1 (10%), and we kept increasing them together by 0.1 (10%) in every experiment run until we reached the value 0.5 (50%) after which the algorithm stopped to return rules. The following was observed after running the experiments:

1. The number of rules generated are inversely proportional with the minimum support and confidence; less rules generated with higher minimum support and confidence.
2. At lower minimum support and confidence some rules that do not make sense are generated, and as the minimum confidence and support increases those rules diminish.
3. The correct direction of the rules is preserved, meaning that rules having higher support and confidence indicate the correct processes that starts the initiation of the CPU usage; as we can see in the last run, HTTP implies MySQL, which confirms with the reality of execution as the traffic is initiated by the http requests which results in initiating queries to the database servers.
4. The difference between the HTTP and MySQL transaction was expected to be higher than what is shown in the statistics graph.
5. Exim rarely appeared in the rules generated, and its appearance is irrelevant to the reality of the nature of the environment. Also spamd, did not appear at all in the rules although it intuitively appears to be related to the email services.

6. The rules at the highest minimum confidence and support indicates that the CPU utilization in the environment was dominated by the HTTP process 40-50% and the MySQL process at 10%.

7 Discussion

The rules generated by the association engine are realistic and confirm with the graph in Figure 6 in the sense that they do not equally occupy the CPU. If the five processes were equally loaded into the CPU, they would have equally appeared in the generated rules for minimum support less than or equal to 0.2, and would have all disappeared as we increase the support above 0.2; actually the HTTP and MySQL processes dominate the occupation of the CPU compared to the other 3 processes. It is shown in the experiments results table that the generated rules decrease as we increase the minimum support cutoff value, and as we increase the support and the confidence minimum cutoff values the rules which remains are corresponding to processes which occupy the CPU more.

It is expected that the number of rules decreases as the minimum confidence and support increase. This indicates that some processes are more active than others, execute more frequently and occupy the CPU for longer durations of time, and this confirms with the statistics graph presented above. If the five processes were equally loaded into the CPU they would have appeared throughout the five runs of the experiment.

Some rules do not make sense, such as EXIM 10 -> MySQL 10, where exim is completely unrelated to MySQL, but at lower minimum support and confidence such rules might appear as there might be a coincidence of the two processes being run together. This is expected to happen, and this can be overcome by defining an affinity value between processes that instructs the association engine not to relate them even if they match.

One of the most important observations is that the correct direction of the relation between HTTP and MySQL indicates that HTTP is the process that initiates the processing. This observation propagated throughout the five runs.
of the experiment, which indicates the reliability of the algorithm to detect the relation. The explanation is that the HTTP requests usually fires a web script execution which reads data from the HTTP socket, executes some normal rendering transactions which are mostly processing, calls the database iteratively, then does some processing to prepare the output, and finally sends the web content back to the client over the HTTP socket. So it is mainly I/O, processing, database interaction, processing, and finally I/O, and this gives the HTTP process more occurrences in the CPU statistics collected as the duration of execution of the HTTP process overlaps and includes the processing of the MySQL, and hence it will appear in more transactions than the MySQL as it appears from the statistical graph, and this gives the HTTP process more support and confidence. The above description of the process of execution, can be interpreted simply as the HTTP is the initiator of the traffic. It was expected that the difference in the number of transactions between the HTTP and MYSQL processes be larger than what is shown above in the statistics graph, but this can be explained as the MySQL process is used not only by the HTTP processes, but also by scheduled PHP command line scripts which are fired via the crontab scheduler.

Although the exim process appeared extensively in the statistics graph, yet it rarely appeared in the rules, especially when we increased the minimum support and confidence and this is attributed to the fact that it is not related to any of the other processes, and the same applies to the spamd process. It is intuitive that the exim and spamd are very strongly related as the received emails are being checked by the spamd engine, yet the physical mode of operations in reality is different, the exim process receives emails from the SMTP port and processes it based on the rules in its configuration files and then adds the emails into the email queue. The spamd is started to check the emails in the queue, and it completely locks the queue until the emails are checked and approved, and this puts the exim process into sleep, and hence the two processes do not coexist in the transactions collected, and consequently they will not appear in the generated rules. This is considered a deficiency, as the spamd processing is based on the queue length that is being filled by exim, but it will not be detected in this scheme, and this can be overcome by introducing exim and spamd log file indicators in the model as will be suggested in the future work.

The rules generated at the highest confidence are showing that the http is executing most of the time between HTTP 40 and HTTP 50, and MySQL 10. It is very important to mention that those values are ranges of values, and in this case MySQL 10 means from 0.1% to 10%, and similarly the HTTP usage is from 30.1% to 50%. The MySQL value in the rules is confirming with the statistics graph above, but for the HTTP process it is a little bit deviated and this is because the CPU utilization of the HTTP fluctuates more than that of MySQL by nature; moving from I/O to processing to sleeping mode waiting for data from the database.

8 Conclusion

The main contribution of the research presented in this paper is to present an extensible architecture for processing raw usage data that is collected cross a cloud environment from different architectural layers and present it in a data model that can be processed by a data mining engine. The proposed architecture is based on the CMML interpreter that generates PMML models from raw usage data. A process correlation prototype is presented to demonstrate the applicability of the proposed architecture, where several experiment runs were performed and the results and observations were analyzed and discussed showing the benefits and the drawbacks of the architecture. The prototype was able to predict the relation between processes with respect to their CPU usage and the level of dependence of processes on others. On the other hand, some hidden relationships could not be inferred by the architecture, such as in the case of EXIM and SPAMD, which can be overcome by introducing more data from both applications logs to the model to enrich the model with information about their relevance. Finally, a set of future work approaches were suggested to overcome functionality that we overlooked and deficiencies in the proposed architecture.

1. The ability to set different usage ranges per process type; for example HTTP increments can be 5% and for MySQL increments can be in 10%. This will provide more accurate rules. Moreover, this can be done in a dynamic way which means that the range of increments can decrease as the value increases.

2. The ability to define affinity levels between different items so that unrelated items such as Exim and MySQL do not appear in the rulesets.

3. A PMML merger module would be very helpful to merge multiple PMML models before the association algorithm can execute them.

4. Transactions from application log files could be integrated into the model; for example, adding the number of HTTP transactions being executed to the model will provide a better measure and can relate the number of requests to the CPU usage of the HTTP process, which will allow an insight to the different application types hosted and their processing requirements.

5. In a high demand processing environment such as distributed environments in general, and cloud environments in particular, the speed of execution of the algorithm is very important, and hence an incremental association mechanism would need to be investigated to allow changing the rulesets dynamically over time as the nature of the execution changes.

6. Allow the integration of more in-line languages in the CMML interpreter, which would allow more flexibility and reliability in parsing raw data and generating PMML models.
References


