RESTful Correlation and Consolidation of Distributed Logging Data in Cloud Environments

Christian Pape, Sven Reissmann, Sebastian Rieger
Applied Computer Science
University of Applied Sciences
Fulda, Germany
{christian.pape, sven.reissmann, sebastian.rieger}@informatik.hs-fulda.de

***Abstract*—Due to the availability of virtualization technologies and related cloud infrastructures, the amount and also the complexity of logging data of systems and services grow steadily. Automated correlation and aggregation techniques are required to support a contemporary processing and interpretation of relevant logging data. In the past, this was achieved using highly centralized logging systems. Based on this fact, the paper introduces a prototype for an automated semantical correlation, aggregation and condensation of logging information. The prototype uses RESTful web services to store and analyze the logging data of distributed logging sources. In this context we will also present the special requirements of handling logging systems in highly dynamic infrastructures like enterprise cloud environments, which provide dynamic systems, services and applications.**

***Keywords*—Monitoring; Enterprise Cloud; Web Services; Log Analysis; Log Correlation;**

I. INTRODUCTION

The vast rise of virtualization technologies and the related wide availability of virtual machines increased the amount of logging data over the past years [1]. In addition to virtual machines themselves, cloud infrastructures, in which they are deployed, also deliver new services and applications in a fast and highly dynamic manner, producing logging data that is needed to monitor their states and service qualities. This leads to a growth of logging sources and the demand for logging systems to dynamically handle new sources and collect the corresponding data. Each new source provides detailed logging information and increases the overall amount of logging data. Typically logging data will be compressed and also anonymized at short intervals if individual-related data is included. Also, outdated log entries can be removed, but the number of logging sources (e.g., the number of virtual machines) themselves can't be reduced. For instance, in a virtualized cloud infrastructure where servers, storage and also the network is virtualized, each system, service and application should at least provide a minimal set of logging data to allow an effective analysis of the status and relevant events during service operation.

To support this analysis and evaluation across logging information originating from a large number of different distributed source systems, correlation techniques offer a way to group similar systems and applications. Furthermore, correlation can be used for the aggregation of logging data hence providing a condensation based on its relevance. In this paper, we introduce a solution to persist logging data that originated from syslog sources in a NoSQL-based database by enhancing existing solutions and using RESTful web services. For correlation and consolidation purposes, this data will also be enriched with meta information before providing the data for distributed analysis and evaluations.

The paper is layed out as follows. In the next section, the state-of-the-art of distributed logging in cloud environments is described. Section III gives examples of related work and research projects that also focus on improving the management and analysis of logging data in distributed or cloud environments. Requirements for the correlation and consolidation of logging data in enterprise clouds are defined in Section IV. Using RESTful web services and NoSQL-based storage, the prototype presented in Section V was implemented. It provides aggregation and condensation of logging data in cloud environments by correlating individual events from distributed sources. The prototypic implementation is evaluated and compared to the state-of-the-art and related work in Section VI. In the last section of this paper, a conclusion is drawn and aspects for future research are outlined.

II. STATE-OF-THE-ART

The following sections give an overview on logging in distributed environments using aggregation and consolidation techniques for standard logging mechanisms like syslog. Also, the advantages of evolving NoSQL databases, which are typically backed by RESTful webservices, are outlined.

A. Distributed Logging in Cloud Environments

Current cloud service providers offer a variety of monitoring mechanisms. For example, Amazon AWS and RackSpace both provide monitoring and alarms for their virtual machines. In the basic version, these services monitor several performance metrics (e.g., CPU, I/O and network utilization). Advanced versions (e.g., Amazon CloudWatch) allow the customers to check the current status of services running in the virtual machines and to define custom metrics and alarms that can be monitored using individual APIs of the cloud service provider. While these APIs could be used to send specific events and alarms, there is no specific service to handle the aggregation, correlation and management of logging data generated and provided by the operating systems and services running in the virtual machines. Furthermore, the individual APIs currently vary from provider to provider. Hence it is not possible to use a unified monitoring across different

 ISBN: 978-1-61208-280-6

cloud service providers. This also hinders the establishment of enterprise clouds that should allow the integration of private or hybrid cloud services operated by public cloud service customers, as these solutions again use individual monitoring techniques. An appropriate standard to address the issue of a cloud service provider independent open logging standard, is currently in the works at the IETF [2].

Until such open standards are available, distributed logging in cloud environments could be carried out by developing specific logging mechanisms for the infrastructures, platforms or applications (IaaS, PaaS, SaaS) used in the cloud. The drawback of this approach would be the effort that is needed for the software development and maintenance. Moreover, the individual APIs developed by the customers are likely to need a migration to upcoming cloud logging standards in the near future. Therefore, the more appropriate approach could be to extend existing and established logging services to support distributed cloud scenarios. The de-facto standard logging service offered in every predefined Linux-based virtual machine image by existing cloud providers is syslog, which is described in the next section. As a matter of fact, syslog is also the basis for the upcoming Internet-Draft [2] focusing on cloud-based logging services. Logging data can be stored and structured in NoSQL-based databases using RESTful web services as described in Section II-C.

B. Log Aggregation and Consolidation with Syslog

Syslog [3] defines a distributed logging solution for generating, processing and persisting host- and network-related events. Since its introduction, the syslog protocol evolved into the de-facto standard for the processing of logging events on UNIX-based systems and several network devices. A syslog message consists of multiple parts. First part is the so called PRI part, which contains a numeric priority and the facility that generated the message. Second part is a header, which includes a timestamp and the hostname of the producer of the message. The latter allows the grouping of different messages originating from the same individual machine. The closing MSG part consists of the message itself and can also include additional informations like the id of the process that produced the event. In a default configuration syslog messages of a host system are stored in files on the host's local filesystem. As outlined above, the impact of virtualization technologies and the corresponding growth of logging sources indicate that a centralized collection and analysis of syslog data is of essential importance. Otherwise, an overall rating of nearly identical messages originating from different sources would be a difficult task.

A centralized logging infrastructure and the utilization of relays to cascade logging servers in large environments, were also design goals of the syslog development. Originally, syslog [3] defines the User Datagram Protocol (UDP) to transport messages. Today the reliable Transmission Control Protocol (TCP) is preferred [4]. Also, additional security features like Transport Layer Security (TLS) assure integrity and authenticity of the data during the transport [5]. Figure 1 shows an example of a centralized logging environment.

The rsyslog server [6] provides an open source implementation of the syslog protocol and is among other solutions like

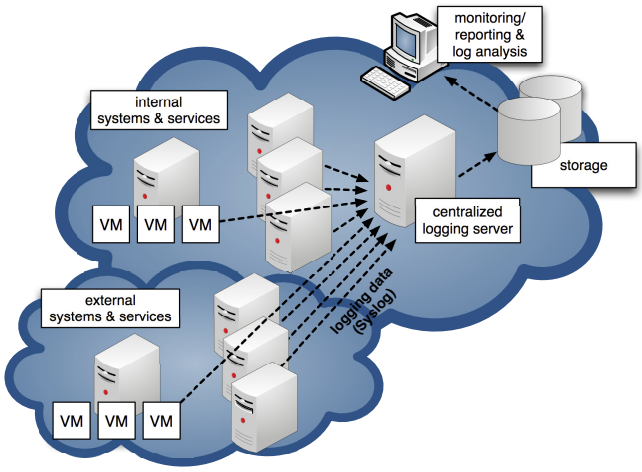

Fig. 1. Centralized logging of distributed systems and services

syslog-ng one of the most popular syslog servers. Also, a large number of plugins are available for rsyslog to support different message normalization techniques and new storage backends like MongoDB, HDFS or ElasticSearch. For these reasons, the popularity of rsyslog increased over the past years. Therefore, we use rsyslog in our research to provide centralized logging of syslog messages.

C. RESTful Log Management utilizing NoSQL Databases

The term NoSQL, standing for Not only SQL, refers to a type of databases that became an interesting alternative to SQL databases over the last couple of years. Although there are different implementations of NoSQL databases that fit different needs, they all share one aspect: They are not relational databases. A reason for the popularity of such new databases might on one hand be their performance. On the other hand the new requirements on storing unstructured data have changed with new concepts like BigData and full-text search. While relational databases demand the structure of the data to be specified when creating the database, NoSQL databases do not have such a need, allowing the structure of the data to be modified or extended at any time. Also, creating a relational database for data that does not easily map into a table-layout (e.g., different log formats from distributed sources) is not easy. Regarding the storage of logging data, the most important issues are the performance of the database system, especially as in many cases there is a tremendous amount of data to be stored, and the flexibility to add new log sources that may introduce new data structures.

When looking at the different approaches of NoSQL databases, three main types can be identified. Key-Value-Stores allow to store unstructured data simply in form of key-value pairs. These databases achieve high performance when querying for keys, but are not very well suited to perform searches on the stored values. Going one step further, column-oriented databases allow the storage of closely related data in an extendable column. Besides using a column layout, they are not bound to the restrictions of the highly structured table layout SQL uses, but also allow to store data in a more detailed structure compared to key-value pairs. A third type of NoSQL

 ISBN: 978-1-61208-280-6

databases is referred to as document-based datastores. These types of databases are able to store collections of documents, each of which having a completely independent structural layout. The structure of new documents can be extended at any time, meaning that documents may consist of any number of key-value pairs of any length. Most of the document-based databases provide a RESTful web service interface allowing to store and retrieve documents using the JSON-standard. Therefore, document-based datastores provide a high degree of flexibility and interoperability.

Regarding the requirements of storing logging data, not all of the previously mentioned NoSQL technologies are suitable for a centralized logging data storage. Key-Value stores as well as column-oriented databases allow highly efficient queries for the data using their keys, while not being suitable for doing full-text searches and correlation on the stored data. Document-based datastores in contrast, allow highly efficient search queries on the full data and also efficient queries using the documents' keys. For our work we used ElasticSearch [7] as a document-based NoSQL datastore, which also offers a high-performance, full-featured text search engine based on Apache Lucene.

The decision to use REST instead of SOAP for our web service was driven by two reasons. First and foremost, the ElasticSearch search and analytics engine primarily offers REST APIs, which mainly use JSON [7]. Also, the rsyslog daemon we chose offers corresponding output modules. Second, the usage of REST perfectly fits the logging in the cloud environments we evaluated, because it is not as strictly tied to XML as SOAP [20]. Therefore, the overhead to transfer logging messages between the logging server and the prototype to correlate and consolidate the logging data, that we present in this paper, can be minimized. Otherwise, for each syslog message being sent to our prototype, the plain text logging data would need to be embedded in an XML structure. While it would have been possible to address this issue, e.g., with SOAP Message Transmission Optimization Mechanism (MTOM), this would still increase the overhead of requests and responses from the the logging server, which also leads to a decrease in performance. On the other hand, SOAP would provide alternative transport protocols and a strict definition of the interface and used data types [21]. The latter ones are not an issue for this paper, as data types and interfaces are already defined by rsyslog output modules and ElasticSearch. Overall, the simplicity of REST [20] allows a rather light-weight implementation for the communiation needed between the central logging server and the prototype that we will present in Section V.

III. Related Work

The challenge of persisting and evaluating decentralized logging data has been in the focus of many research publications. For instance, the evaluation of decentralized logging informations in IaaS, PaaS and SaaS cloud environments were described in [8] and [9]. Also, an internet draft is in development [2], covering logging of syslog messages from distributed cloud applications. Besides the requirements by these new highly distributed applications, there is also a challenge for analysis and structuring of logging information. Existing solutions for automated log analysers only comply with some of these requirements [10]. Therefore, Jayathilake [10] recommends the structuring of logging data and to extract the contained informations. In this context, NoSQL databases are best suited for handling these variable fields. These databases provide an adaptive approach of persisting data and allow the use of different table schemata or, e.g., an document-based approach storing key-value pairs. As outlined in [11] and [12], the evaluation and rating itself can be automated by event correlation and event detection techniques. Both publications also describe the use of the correlation solution Drools, that we use for our research. Correlation techniques help to reduce (and consolidate) the logging data so that only a condensed representation including relevant information, required for analysis and evaluation, will be persisted. As described in [12], a reduction of syslog data by up to 99% is possible. A solution based on the NoSQL database MongoDB using MapReduce to correlate and aggregate logging data in distributed cloud analysis farms is described in [13]. This solution however lacks event correlation and detection techniques.

IV. Requirements for Correlation of Logging Data in Enterprise Clouds

We initially described that centralized logging environments tend to produce a tremendous amount of logging events at the central logging server. To manage the storage of all these data and provide a way to perform a fast analysis on the stored data, the use of the previously mentioned NoSQL datastores seems obvious. However, looking at the amount of data that has to be manually analysed and evaluated, the question arises whether it is possible to automate the process of evaluating the relevance of certain syslog events or even reduce the amount of data that will be stored. The latter is only reasonable if it can be guaranteed that no valuable information will be lost by the reduction of messages. In the next sections, we are going to describe our approach for automatic evaluation and reduction of syslog events in detail.

A. Correlating Distributed Logs in Enterprise Clouds

The core objective of this paper is the processing of the data provided from syslog and to identify important events of the network or individual hosts. For instance, the sequence of messages of an ongoing SSH brute force attack illustrates the demand for an automated rating of messages. During a brute force attack, the SSH daemon generates a log entry for each invalid login attempt. These messages are delivered to a centralized syslog server, indicating individual failed login attempts. However, the relative small number of events might become lost in the large total amount of syslog messages.

An IT operator analysing the logging data is not interested in displaying each individual login attempt, but rather wants to know whether the brute force attack led to a successful login. To answer this kind of question, the syslog messages must be filtered for the corresponding SSH daemons and searched for failed login attempts that are followed by a successful login. Thus, a system registering a large number of failed login attempts, and finally a successful login, might experience a successful brute force attack, while the possibility of an attack is rising along with the number of failed login attempts. The time-consuming and costly search for attack patterns like this can be simplified by an automated rating of syslog messages.

 ISBN: 978-1-61208-280-6

To identify individual messages describing similar events from different operating systems and platforms, it is required to normalize syslog data before correlating and persisting them. For the lookup of SSH login attempts, it is sufficient to examine single individual syslog messages. In order to identify a completed attack, a sequence of these matching messages must be investigated. If the conditions for a successful attack are met, it is possible to generate a new prioritized syslog message to support the immediate detection of these security threats in the network.

B. Consolidating Logging Data from Distributed Services

A second goal of our work was to reduce the amount of messages that actually get persisted into ElasticSearch. This may seem subordinate against the backdrop of increasing computing performance and concepts like BigData, but reducing the actual data still results in faster and easier analysis, even when using these new techniques. In practice, we actually see an advantage of the reduction of stored messages in long-term storage and data analysis. Such reduction techniques basically delete messages of a certain age or don't persist messages below a certain severity. However, these simple mechanisms result in a loss of valuable information, and for this reason are not practicable in our view.

Our approach first provides a grouping of messages. For example, same or recurring events are summarized. Based on those groups we are able to generate new summarized syslog messages containing a dense representation of all the valuable information of the initial messages and hence allowing us to actually drop those without loosing information. Using our solution, it is possible to reduce the amount of messages that needs to be stored at the central database server, and therefore improving the performance of the system without loosing information. Furthermore, it is possible to manipulate the severity of the newly generated messages, to even increase their value for later analysis.

An example of such a modification of syslog messages could be used to detect the previously mentioned brute force attack, that results in a flood of messages with a low priority. By generating a single message with a high priority - telling an administrator what the actual attack looked like, judging from the number of login attempts, the duration of the attack and the actual result - we produce information that helps to estimate the situation and the next steps to be taken. Also, regardless of waiving all the failed login attempts at the central database server, it is still possible to perform an exact analysis of the attack by looking into the logfiles of the actual server that was under attack.

A second example of consolidating messages would be the correlation of application access logs. For instance, in a cloud environment new machines will be spawned on demand, so several machines provide a single service in a cooperative way. An example could be a number of dynamically started HTTP servers receiving requests via a load balancer. The requests on the individual servers will be logged to the centralized syslog server, but these individual events must be aggregated, e.g., to support the decision process of starting new or stopping VMs running an HTTP server. The access log messages can be correlated to an access count per timeslot and it is also possible to count active HTTP servers by differentiating distinct logging sources. As already illustrated in the previous example it is again not necessary to persist the original access messages. The correlated logging information is useful to evaluate the load on all servers and can also be used to determine whether running machines have to be stopped or new ones need to be started.

Taking the logging information into account a thorough decision can be made that goes beyond the possiblities of network-based load balancing and failover techniques. A more generic approach would be to use Drools to count messages matching a set of rules for specific timeslots and to generate histograms for these kind of messages. This approach allows to compare different timeslots and answer questions like "Were the same number of crond jobs executed on monday and tuesday?". Also, a visual representation of these results, e.g., as presented in [14], could be possible with the benefit of easily identifying anomalies at first sight.

V. Implementation of Log Correlation and Consolidation in Cloud Environments

To facilitate the analysis and storage of logging data in distributed cloud environments, this paper presents a log correlation and consolidation prototype. The prototypic implementation uses rsyslog [6] as a central syslog server that receives and normalizes syslog messages originating from distributed sources, e.g., VMs in the cloud. After normalizing the data and hence allowing to process the data from different sources in a unified way, the logging information is serialized to JSON and sent using a RESTful web service to our prototype, which we implemented in Java. The prototype embodies a correlation engine, which analyses the messages and afterwards persists them, again using JSON via a RESTful web service, in an ElasticSearch cluster. Our implementation of the correlation is based on the Complex Event Processing (CEP) Engine Drools Fusion [15]. This engine allows the definition of rules using temporal reasoning that we use for the correlation of messages.

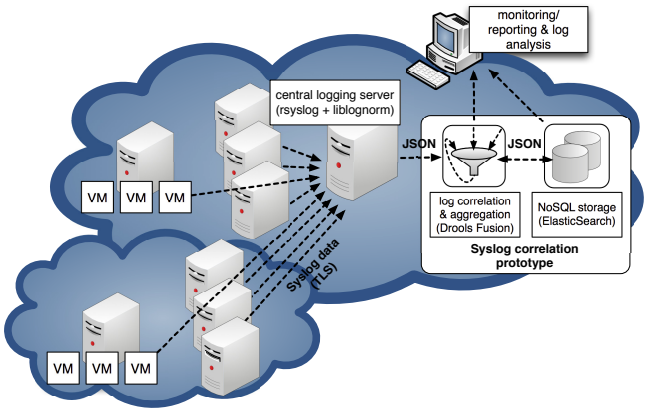

Fig. 2. Correlation and aggregation of centralized logging data

Figure 2 shows the setup of our testbed. Syslog messages are sent from the distributed clients to the rsyslog server using a secure TLS connection. The server normalizes the messages using liblognorm [16] and transmits the normalized messages in JSON format using RESTful web services

to the correlation prototype. By using REST, our prototype implements a flexible interface that can easily be used from a variety of cloud services. Moreover, the majority of the compute cloud providers offer syslog-based logging in their VMs and therefore our entire approach using rsyslog can be used to correlate the logging data across multiple and heterogenous cloud environments. Having received the messages, our prototype temporarily stores the logging data locally, but also instantly transfers them to the ElasticSearch cluster, again using RESTful JSON-based web service requests. The temporal storage is currently needed for the evaluation of the defined rules in the Drools engine, as it is configured to be done in-memory to achieve a better performance. As described in the last section of this paper, we're also evaluating persisting the messages directly and execute the correlation in the NoSQL storage to overcome the size limitation of our in-memory approach. The rules can be easily extended to provide multiple correlation and consolidation techniques. An example is presented in Section VI. Messages that did not match any of the rules will be removed from the in-memory cache. On the other hand, messages matching at least one rule are correlated, and the result is stored with a flag representing the successful correlation and a reference to original messages that have been correlated in the ElasticSearch cluster. By cyclic searching for successful correlation flags in the ElasticSearch cluster and pruning the original messages they refer to, a consolidation is achieved.

VI. Scalable RESTful Log Analysis in Cloud Environments

In this section we give an example of the usage of our solution for the correlation of logging data being generated during an SSH brute force attack as described in Section IV-A. The aim is to generate a new syslog message with a higher priority in the case of a successful SSH login immediately after a certain number of failed logins during possible SSH brute force attacks. To detect this scenario we defined the rules shown in the following listings in the correlation engine of our prototype. Hence, the detection of the successful SSH brute force attack will be done automatically by our prototype. For the correlation of the syslog messages that indicate a successful brute force attack on the password during an SSH login, the corresponding syslog messages need to be isolated and filtered from the stream of logging data originating from the syslog server. Examples for the failed and successful SSH login messages are shown in listing 1. The messages shown here have already been normalized by rsyslog and are therefore independent of the individual host that generated them.

```
% Failed password for root from 192.168.1.1
    port 34201 ssh2
% Accepted password for root from 192.168.1.1
    port 34201 ssh2
```

Listing 1. Syslog message of failed and successful SSH logins

To correlate these events, our prototype implements the rules using Drools Fusion [15], which detect messages matching a successful SSH login after a certain amount of previously failed logins (based on the message format shown in listing 1). Listing 2 contains the rules we defined for our example. The rule matching failed messages requires a success message within 1 minutes after at least 10 failed messages.

```
success : Message(
  message matches
    "Accepted password for [^\\s]* from [^\\
        s]* port [^\\s]* ssh2")
failed : ArrayList( size >= 10 ) from
    collect(
  Message(this before[0,1m] success,
    message matches
      "Failed password for [^\\s]* from [^\\
          s]* port [^\\s]* ssh2") )
```

Listing 2. Drools fusion rules to detect successful SSH brute force attacks

If the failed rule that we defined in listing 2 matches, our current implementation generates a new syslog message with the facility "security" and priority "emergency" containing the message "Possible successful SSH brute force attack". It would also be possible to postpone the persistence of the logging data until the correlation is finished, to store all message related to the attack with a higher priority, e.g., "emergency", in the ElasticSearch cluster. Another possibility would be to only persist the new message that indicates the possible SSH brute force attack with a high priority and drop the other messages that have been correlated. Messages needed for the correlation are kept automatically in the in-memory cache by Drools Fusion according to the rules we defined. Drools automatically removes messages from the cache that do not match any of the rules anymore.

VII. Conclusion and Future Work

In the previous sections, we presented a solution to automatically correlate and consolidate syslog messages containing logging data from distributed sources in cloud environments. Besides evaluating the requirements for such implementations and defining an appropriate concept, a prototype based on RESTful web services and NoSQL database storage was developed. The prototype addresses the requirements for correlation and consolidation of distributed logging sources in today's enterprise cloud environments. It supports the proper condensation of log messages by grouping individual messages. The achieved reduction improves the performance of processing and analysing logging data especially in distributed environments with a lot of systems (typically virtual machines) sending similar logging information.

Existing monitoring solutions could be enhanced to use the presented prototype as a filter improving the quality and relevance of the logging data (e.g., by using escalation techniques, traps, or sending messages regarding detected events) as shown in the example of an SSH brute force attack in Section IV-A and VI. The integration of the prototype with existing network monitoring tools (e.g., OpenNMS, splunk) is one of the next steps for our research. An interesting starting point could be their interfaces to correlate events, i.e., to perform a root-cause-analysis, that could be extended to consume relevant events that were filtered from the distributed logging data by our prototype. Another option could be to use these interfaces bidirectionally to enrich the logging information, e.g., combining information in the logging data with the location or other details from asset, configuration, system or service management. For example, expected downtimes

 ISBN: 978-1-61208-280-6

could be resolved to ignore corresponding log events in the prototype.

While this paper focuses on the usage of the de-facto network-based logging standard syslog, the prototype presented in this paper could also handle different text-based logging sources (e.g., application specific log files, log4j, etc.). A current limitation regarding the amount of logging data that can be correlated is the available memory. Theoretically, the prototype could also use data that is already stored in the NoSQL storage for the correlation to overcome this limitation. While this approach has a negative impact on performance, it could on the other hand dramatically increase the accuracy of complex correlation over long-term data. The enhancement could be easily implemented using the RESTful search API not only for the analysis but also while filtering and before persisting the logged data in the NoSQL database. In the next version of our prototype, we will implement this extension and evaluate the performance impact (regarding latency to store a log entry and overall throughput of the correlation engine). Also, balancing the load of complex correlations across multiple instances of the prototype, e.g., elastically in the cloud, could be an option. Using OpenStack we currently set up an enterprise cloud environment to serve as a scalable platform for our prototype.

Our predefined rule set outlined in this paper can easily be generalized to fit the requirements of other use cases. In our ongoing evaluation we will therefore contrast the results of our prototype to comparative work being presented in [11], [12] and [13]. Another possible topic for future research could be the integration of existing knowledge-based systems and automated reasoning as developed, e.g., for network anomaly and intrusion detection systems (IDS). Even more interesting could be the integration of existing work that has been published regarding the detection of anomalies in syslog messages. Makanju et. al. [17] are describing a promising solution to detect anomalies in logging data of high performance clusters (HPC). Administrators can confirm the detected anomalies to correlate them with error conditions and trigger a consolidation. These techniques could also facilitate the definition of correlation rules as patterns are detected without prior configuration. Syslog-based event forecasting, as described, e.g. in [18], could be another promising option for our prototype. The prototype could be used to enhance the information being evaluated to generate the forecast, but can also consume the forcasting data. This way, existing rulesets could be augmented. Furthermore, the definition of rules could be simplified by automatically deriving rules from the forecasts, which have been submitted to our prototype. To detect failures and error conditions in cloud environments this has already been proposed in [19]. We will evaluate to extend this approach to allow for the correlation and aggregation of logging data in enterprise cloud environments.

REFERENCES

[1] C. Canali and R. Lancellotti, “Automated clustering of vms for scalable cloud monitoring and management,” in Software, Telecommunications and Computer Networks (SoftCOM), 20th International Conference on, 2012, pp. 1-5.

[2] G. Golovinsky, D. Birk, and S. Johnston, “Syslog extension for cloud using syslog structured data - draft-golovinsky-cloud-services-log-format-03,” Internet-Draft, IETF, 2012.

[3] C. Lonvick, “RFC 3164: The BSD syslog protocol,” Request for Comments, IETF, 2001.

[4] R. Gerhards, “RFC 5424: The syslog protocol’,” Request for Comments, IETF, 2009.

[5] K. E. Nawyn, “A security analysis of system event logging with syslog,” SANS Institute, no. As part of the Information Security Reading Room, 2003.

[6] R. Gerhards, “The enhanced syslogd for linux and unix rsyslog,” http://www.rsyslog.com, [retrieved: 4, 2013].

[7] Elasticsearch Global BV, “The enhanced syslogd for linux and unix rsyslog,” http://www.elasticsearch.org/, [retrieved: 4, 2013].

[8] R. Marty, “Cloud application logging for forensics,” in Proc. 2011 ACM Symposium on Applied Computing, ACM, 2011, pp. 178-184.

[9] A. Rabkin and R. Katz, “Chukwa: A system for reliable large-scale log collection,” in Proc. 24th international conference on Large installation system administration, USENIX Association, 2010, pp. 1-15.

[10] D. Jayathilake, “Towards structured log analysis,” in Computer Science and Software Engineering (JCSSE), International Joint Conference on, 2012, pp. 259-264.

[11] A. Müller, C. Göldi, B. Tellenbach, B. Plattner, and S. Lampart, “Event correlation engine,” Department of Information Technology and Electrical Engineering - Master’s Thesis, Eidgenössische Technische Hochschule Zürich, 2009.

[12] M. Grimaila, J. Myers, R. Mills, and G. Peterson, “Design and analysis of a dynamically configured log-based distributed security event detection methodology,” The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology, vol. 9, no. 3, 2012, pp. 1-23.

[13] J. Wei, Y. Zhao, K. Jiang, R. Xie, and Y. Jin, “Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis,” in Cloud and Service Computing (CSC), International Conference on, 2011, pp. 354-359.

[14] K. Fukuda, “On the use of weighted syslog time series for anomaly detection,” in Integrated Network Management (IM), IFIP/IEEE International Symposium on, 2011, pp. 393-398.

[15] J. Community, “Drools - jboss community,” http://www.jboss.org/drools/, [retrieved: 4, 2013].

[16] R. Gerhards, “A syslog normalization library,” http://www.liblognorm.com, [retrieved: 4, 2013].

[17] A. Makanju, A. Nur Zincir-Heywood and E. E. Milios, “Interactive Learning of Alert Signatures in High Performance Cluster System Logs,” in Network Operations and Management Symposium (NOMS), IEEE, 2012, pp. 52-60.

[18] A. Clemm and M. Hartwig, “NETradamus: A forecasting system for system event messages,” in Network Operations and Management Symposium (NOMS), IEEE, 2010, pp. 623-630.

[19] Y. Watanabe, H. Otsuka, M. Sonoda, S. Kikuchi and Y. Matsumoto, “Online failure prediction in cloud datacenters by real- time message pattern learning,” in Cloud Computing Technology and Science (CloudCom), 4th International Conference on, IEEE, 2012, pp. 504-511.

[20] C. Pautasso, O. Zimmermann and F. Leymann, “Restful web services vs. ’big’ web services: making the right architectural decision,” in Proc. of the 17th international conference on World Wide Web, ACM, 2008, pp. 805-814.

[21] M. Zur Muehlen, J. V. Nickerson, K. D. Swenson, “Developing web services choreography standards - the case of REST vs. SOAP,” Decision Support Systems, 40.1, 2005, pp. 9-29.

 ISBN: 978-1-61208-280-6