Correlation and Consolidation of Distributed Logging Data in Enterprise Clouds

Sven Reissmann, Dustin Frisch, Christian Pape, and Sebastian Rieger
University of Applied Sciences Fulda
Department of Applied Computer Science
Fulda, Germany
{sven.reissmann, dustin.frisch, christian.pape, sebastian.rieger}@cs.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 relies on a NoSQL storage back-end that is used to persist consolidated messages of distributed logging sources in a highly performant manner. This step of consolidation includes strategies for minimizing long-term storage, and by using correlation techniques also offers possibilities to detect anomalies in the stream of processed messages. In this context, we will present the special requirements of handling scalable logging systems in highly dynamic infrastructures like enterprise cloud environments, which provide dynamic systems, services and applications.**

***Keywords*—Syslog Correlation; Log Analysis; Anomaly Detection; Monitoring; Enterprise Cloud.**

I. Introduction

The vast rise of virtualization technologies and the related wide availability of virtual machines (VM) has 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 state and service quality. 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 personally identifiable information is included. Also, outdated log entries can be removed, but the number of logging sources (e.g., the number of virtual machines) themselves often cannot be reduced. For instance, in a virtualized cloud infrastructure where servers, storage and also the network are virtualized, each system, service and application should at least provide a minimal set of logging data to allow an effective analysis of its 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 [1], we introduced a solution to persist logging data that originated from syslog sources into a NoSQL-based (Not only SQL) database by enhancing existing solutions. For correlation and consolidation purposes, this data was also enriched with meta information before providing the data for distributed analysis and evaluation. This article elaborates on the implementation and concepts outlined in [1] and presents extensions to use different, e.g., structured logging sources and increase the efficiency of our correlation engine. Furthermore, an evaluation test-bed to enhance the scalability of our implementation in OpenStack-based enterprise cloud environments is given. Additionally, we present possibilities to use information from external network and system management solutions to enrich the events being correlated.

The article is laid 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 cloud environments. Requirements for the correlation and consolidation of logging data in enterprise clouds are defined in Section IV. In Section V, the implementation of a prototype for log correlation and consolidation in cloud environments is presented. It provides aggregation and condensation of logging data in cloud environments by correlating individual events from distributed sources. At the end of the section, an example of the usage of our prototype for the log analysis in cloud environments is given. Section VI describes the deployment of our prototype in an OpenStack test-bed. The performance of our prototypical implementation is evaluated in Section VII using multiple test cases. In the last section of this article, a conclusion is drawn and aspects for future research are outlined.

II. State-of-the-Art

The following sections give an overview on the evolution of logging methods in distributed environments using aggregation and consolidation techniques for standard logging mechanisms like syslog. Also, the advantages of evolving NoSQL databases in this area are outlined.

A. Distributed Logging in Cloud Environments

Current cloud service providers offer a variety of monitoring mechanisms. For example, Amazon Web Services (AWS)

and RackSpace both provide monitoring and alarms for their virtual machines. In the basic version, these services monitor several performance metrics, like central processing unit (CPU), input/output (I/O), and network utilization. Advanced versions (e.g., Amazon CloudWatch [2]) 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 application programming interfaces (API) of the cloud service provider. While these APIs could be used to send particular 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 implementation across different 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 cloud service provider-independent logging, is currently in the works at the Internet Engineering Task Force (IETF) [3].

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 required software development and maintenance effort. Moreover, the individual APIs developed by the customers are likely to need a migration to upcoming cloud logging standards in the near future. The more appropriate approach, therefore, 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. In fact, syslog is also the basis for the upcoming Internet-Draft [3] focusing on cloud-based logging services. Logging data can be stored and structured in NoSQL-based databases as described in Section II-C.

B. Log Aggregation and Consolidation with syslog

Syslog defines a distributed logging solution for generating, processing and persisting host- and network-related events. As a key feature, it allows one to separate the software that generates events from the system that is responsible for processing, storing and analyzing those events. Since its introduction, the syslog protocol has evolved into the de-facto standard for processing and forwarding logging events on UNIX-based systems and network devices (i.e., routers, firewalls). However, there has not been any standardization of the protocol characteristics for quite a long time, which has led to incompatibilities across vendor-specific implementations.

The state of the protocol (*Berkeley Software Distribution (BSD) Syslog Protocol*), including the most commonly used message structure and data types, has been documented by the IETF in RFC 3164 [4]. Each syslog packet starts with a so-called PRI (priority) part representing the severity of the message as well as the facility that generated the message. Severity and facility are together numerically coded using decimal values; the priority value, which is placed between angle brackets at the very beginning of the message, is calculated by first multiplying the facility value by 8 and then adding the severity value. RFC 3164 specifies eight possible values for the severity of a message, as well as 24 facility values, which are assigned to some of the operating systems' daemons (i.e., the mail subsystem). The second part of a syslog packet, following immediately after the trailing bracket of the PRI part, is called HEADER and includes a TIMESTAMP and HOSTNAME field, each followed by one single space character. TIMESTAMP represents the local time when a message was generated using the format "Mmm dd hh:mm:ss", while HOSTNAME may only contain the hostname [5], IPv4 address [6], or IPv6 address [7] of the producer of the message. Finally, the MSG part of a syslog packet consists of two fields, known as TAG and CONTENT, where CONTENT contains the actual message, while TAG is the name of the program that generated it. The TAG value can be distinguished from the message or other optional information by a colon or left square bracket depending on the presence of an optional (but commonly used) combination of the program name with its process ID. Listing 1 shows an example of an RFC 3164 compliant syslog message.

```
<34>Oct 11 22:14:15 mymachine su: 'su root'
    failed for lonvick on /dev/pts/8
```

Listing 1. Example of an RFC 3164 compliant syslog message.

While syslog messages are typically stored in files on the generating host's local filesystem, we outlined above that the impact of virtualization technologies and the corresponding growth of logging sources indicate that a centralized collection and analysis of syslog events is of essential importance. Otherwise, an overall rating of nearly identical messages originating from different sources (i.e., from a cloud service that spreads over multiple hosts) would be a difficult task. As a matter of fact, centralized logging infrastructures and the utilization of relays to cascade syslog servers in large environments were considerations in the early development of syslog. Historically, the BSD Syslog Protocol [4] uses the User Datagram Protocol (UDP) to transport messages over the network, which may lead to the imperceptible loss of important events. To address this issue, the use of a reliable delivery mechanism for syslog has been proposed shortly after the documentation of the protocol characteristics [8]. Additionally, security features like Transport Layer Security (TLS) and cryptographic signatures have been proposed to assure the integrity and authenticity of the data during transport of the messages from the sending hosts [9][10][11]. Figure 1 shows an example of a centralized logging environment, where a number of physical or virtual servers send their messages to a central syslog server, which then stores these messages appropriately.

A major drawback of the previously described syslog protocol regarding correlation and analysis capabilities is the unstructured nature of the messages' format. In 2009, a standard for a syslog protocol was proposed in RFC 5424 [12], which obsoletes the previously described but still widely used BSD Syslog Protocol. The new protocol specifies the PRI part of a syslog packet in the same way as its predecessor, but includes it into the HEADER part of the packet together

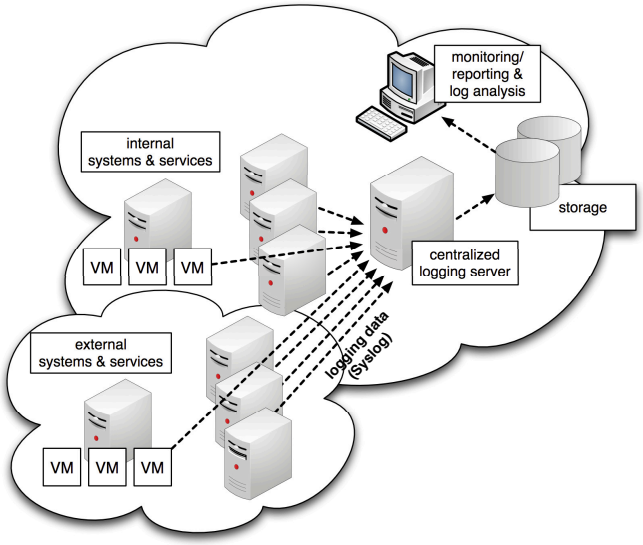

Fig. 1. Centralized logging of distributed systems and services.

with additional information such as the name and process ID of the generating application and a message ID. Another feature of the new HEADER format is the use of a formalized TIMESTAMP value as standardized in RFC 3339 [13]. Most important, however, is the possibility to add structured data into a syslog packet, allowing to express information in a well defined and easily machine-parsable format. An example of a syslog message containing both structured data and a regular free text message is expressed in Listing 2.

```
<165>1 2003-10-11T22:14:15.003Z
       mymachine.example.com
    evntslog - ID47 [exampleSDID@32473 iut="3"
    eventSource="Application" eventID="1011"]
    BOMAn application event log entry...
```

Listing 2. Example of an RFC 5424 compliant syslog message.

Among the wide range of available syslog servers capable of processing the previously delineated protocols, syslog-ng [14] and rsyslog [15] are some of the most widely used solutions on UNIX-based operating systems. Both implement RFC 3164 as well as RFC 5424 message transport, various security features such as cryptographic signatures, message encryption, and the ability to convert messages from one format to another. The rsyslog server, which provides an open-source implementation of the syslog protocol, was selected for our research as it provides interesting features like a large number of input and output modules to support a wide range of logging sources and novel storage back-ends like MongoDB, Hadoop Distributed File System (HDFS), and Elasticsearch. Furthermore, message normalization and modification techniques make it possible to parse incoming messages and add structured information to them, which enables post-processing applications to apply correlation techniques and filtering rules. In our test-bed, which we describe in detail in later sections of this article, we make use of rsyslog's input filters, message normalization provided by liblognorm, and message forwarding capabilities using various output modules.

C. Log Management utilizing NoSQL Databases

Regarding the persistence of logging data, important aspects are performance and scalability of the storage system, especially as in many cases there is a tremendous amount of data to be stored. Further, the flexibility to add new logging sources that may introduce new data structures (i.e., when providing individual structured data as described in the previous section) is a crucial requirement. While various storage back-ends are available for centralized logging environments, we would argue that only a certain type of these systems qualifies by offering reasonable write performance and, even more important, allowing complex analysis on the stored data. That class of storage systems is called NoSQL, which refers to a type of data storage that became an interesting alternative to Structured Query Language (SQL) databases over the past couple of years, especially for storing huge amounts of information such as logging data.

Relational databases demand the structure of the data to be specified at the time the database is created. This means that creating a relational database for data that does not easily map into a fixed table-layout (e.g., different log formats from distributed sources) is not a simple task. NoSQL datastores, in contrast, provide little or no pre-defined schema, allowing the structure of the data to be modified or extended at any time. This property qualifies NoSQL for persisting structured syslog information, where on the one hand, the data is well formatted, but at the same time, the structured key-value pairs of the individual syslog sources may vary depending on the type of service or system that generated the message.

While the name NoSQL makes it appear that the lack of SQL is the most important difference, there are a number of other characteristics that distinguish this type of storage system from the well-known SQL database management systems (DBMS). In almost all cases, a simple query interface is used for storing, retrieving or modifying data, rather than an SQL processor. While the query language of an SQL DBMS itself does not necessarily have negative effects on performance, it can be quite difficult to write efficient queries when trying to do complex operations on the data, e.g., joining a large number of tables. Another important difference of NoSQL compared to SQL DBMS is the scalability over hundreds of hosts, which is achieved in exchange for giving up 100% ACID (Atomicity, Consistency, Isolation, Durability) semantics. Instead, NoSQL guarantees consistency only within certain boundaries or within a specific record. At the same time, scalability leads to high availability by doing transparent failover and recovery using mirror copies. Of course, not all copies are guaranteed to be up to date, because of the previously mentioned lack of the ACID compliance.

Although there are different implementations of NoSQL datastores that fit completely different needs, all are designed to fit new requirements, like storing unstructured data or performing full-text search queries. In [16], Cattell presents an overview of the available types of NoSQL technologies and names some of the actual implementations of the various technologies. Looking at the distinct approaches of NoSQL, three main types can be identified. Key-value stores allow one to store unstructured data in a single distributed key-value index for all the data. The data is typically stored as binary large object (BLOB) without being interpreted and

can be accessed by a client interface, which provides basic operations like insert, delete and index lookups. Key-value stores achieve high performance when querying for keys, but are not very well suited to perform searches on the stored objects themselves. Going one step further, extensible record stores, also referred to as column-oriented datastores, allow the storage of closely related data in an extendable row-and-column layout. Scalability is achieved by splitting both rows, through shards on the primary key, and columns, by using pre-defined column groups. Besides using a row and column model, extensible record stores are not bound to the restrictions of the highly structured table layout SQL uses, as new attributes can easily be added at any time. Still, they allow one to store data in a more detailed structure compared to the previously mentioned key-value stores. A third type of NoSQL is referred to as document-based datastores or document stores. Here, document is not to be confused with an actual document in the traditional sense. Instead, these types of databases are able to store collections of objects, each of which may have a completely independent number of attributes of various types. 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. Like the previous, document stores can partition the data over multiple machines for scalability and replication of the data. Document stores provide a high degree of flexibility and interoperability, but like other NoSQL systems, they do not provide ACID transactional conformance.

Looking at the requirements for storing logging data and being able to do complex analysis of the data later on, not all of the previously named NoSQL technologies are suitable. Key-value stores as well as column-oriented databases allow highly efficient queries for the data using their keys, while not being applicable for doing full-text search queries and correlation on the stored data. Document stores, in contrast, allow highly efficient search queries on the stored data objects as well as the documents' keys. Further, the flexibility to add any newly formatted document at any time is an important feature when storing individually structured syslog messages. For the implementation of our prototype, we use Elasticsearch [17], a document store that offers a high-performance, full-featured text search engine based on Apache Lucene [18].

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 information in IaaS, PaaS and SaaS cloud environments was described in [19] and [20]. Also, an Internet-Draft is in development [3] covering processing 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 analyzers comply with only some of these requirements [21]. Therefore, Jayathilake [21] recommends structuring logging data and extracting the contained information. 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., a document-based approach to storing key-value pairs. As outlined in [22] and [23], the evaluation and rating itself can be automated by event correlation and event detection techniques. Both publications also describe the usage of the correlation solution Drools, which 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 [23], 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 [24]. This solution, however, lacks event correlation and detection techniques.

IV. Requirements for Correlation of Logging Data in Enterprise Clouds

In the introduction of this article, we described that centralized logging environments tend to produce a tremendous number of logging events at the central logging server. To manage the storage of all these events 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 analyzed 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 after the reduction of messages. In the next sections, we are going to describe in detail our approach for automatic evaluation and reduction of syslog events. Also, a method for identifying interesting types of events for which a correlation or consolidation is possible is proposed at the end of this section.

A. Correlating Distributed Logs in Enterprise Clouds

The core objective of our previous work was the processing of data provided via syslog and the identification of important events in the network or on individual hosts. For instance, the sequence of messages of an ongoing Secure Shell (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 relatively small number of such specific events might become lost in the large total number of syslog messages received from all distributed sources. At the same time, a single message indicating that an SSH login attempt was successful will definitely be hard to identify among the huge number of syslog messages denoting failed login attempts while the brute-force attack is running. Figure 2 shows a visualization of authentication messages collected by a central syslog server used to monitor servers, network devices and storage systems for students' web servers at the University of Applied Sciences Fulda.

The resulting graph gives an overview of the general behavior and basic noise of these logging messages. The Holt-Winters [25] algorithm was used to compute and adapt a confidence band over time, which represents the normal behavior of the time-series data. If specific values are violating this confidence band for a number of periods in a given time window, these values are marked as failures or aberrant

Fig. 2. Holt-Winters based vertical bands representing high occurrences of SSH login attempts resulting in ”Failed password” messages.

behavior. This is also visualized in the graphical output by yellow vertical areas (vertical bands in the graph). In this example, the peaks were results of distributed brute-force attacks with common user account names, such as root, admin or test.

An IT operator analyzing the logging data, however, is not interested in displaying each individual login attempt, but rather wants to know whether the brute-force attack led to a successful login into one of his systems. To answer this kind of question, the syslog messages must be filtered for data of 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 rises 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. 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. By using the flexibility of structured data in syslog messages we described earlier, it is possible to add certain tags to the parsed or newly generated messages. These tags may be used later on, in order to support analysis by allowing the application of filters, e.g., to separate newly injected messages from the normal syslog messages after they are persisted all together into the Elasticsearch cluster.

B. Consolidating Logging Data from Distributed Services

A second goal of our work was to reduce the number of messages that actually get persisted into the Elasticsearch cluster. This may seem subordinate against the backdrop of increasing computing performance and concepts like BigData. Reducing the actual amount of data that gets persisted, however, still results in faster and easier analysis, even when using the novel NoSQL techniques. In practice, we actually observe the reduction of stored messages in long-term storage as a requirement for many companies. Such reduction techniques basically delete messages of a certain age or do not persist messages below a certain severity. However, this results in a loss of valuable contextual information, which is why we would argue that such simple consolidation mechanisms are not practicable for logging data. Our novel approach first provides a grouping mechanism, where identical or recurring events are summarized. Based on those groups we are able to generate new syslog messages containing a dense representation of all the valuable information of the initial messages, thus allowing us to actually drop those messages without losing any contextual information. Using our solution, it is possible to reduce the number of messages that need to be stored at the central database server, and therefore we are able to improve the performance of this system without losing information. Furthermore, it is possible to manipulate the severity of the newly generated messages, in order to increase their value for later analysis.

An example application of such modification of syslog messages might be the detection of the previously mentioned brute-force attack, which results in a flood of low-priority security event messages. By generating a single message of 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 supports estimating the situation and the next steps to be taken. In addition, regardless of waiving all the failed login attempts at the central database server, further investigation is still possible by performing an exact analysis of the attack using the logfiles stored at the individual 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 systems provide a single service in a cooperative way. An example could be a number of dynamically started Hypertext Transfer Protocol (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 or stopping virtual machines running the HTTP servers. 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 confident decision can be made that goes beyond the possibilities of network-based load balancing and failover techniques. A more generic approach would be to use correlation engines like Drools to count messages matching a set of rules for specific timeslots and to generate diagrams for these kinds of messages. This approach allows one to compare different timeslots and to answer questions like “Were the same number of cron jobs executed on Monday and Tuesday?”. Also, a visual representation of these results, e.g., as presented in [26], could be possible with the benefit of easily identifying anomalies at first sight.

C. Identifying logging events of interest

When trying to automatically process syslog messages with the objective of evaluating their importance by using correlation techniques, one has to start by identifying sequences of messages that point to certain events. We did a manual analysis of such events by using large amounts of syslog information provided by the General Students' Committee of the University of Applied Sciences Fulda and a number of virtual servers we previously set up to collect and simulate specific attacks. To analyze the dataset, we implemented a small Java-based application that counts similar messages in five-minute intervals based on regular expressions. The results of these calculations are stored in a round-robin archive using RRDtool [27], which is able to generate a graphical representation of the data. This simplified approach to represent logging data in a diagram allows us to use algorithms applicable for time series data, e.g., forecasting or anomaly-detection algorithms. The RRDtool utility itself implements an exponential smoothing algorithm for forecasting and anomaly detection also known as Holt-Winters Aberrant Behavior Detection [28]. We used this algorithm to identify aberrant behavior in the syslog data of specific applications we monitored. An example of this method based on counting warning messages from the postfix/smtpd process on mail servers is shown in Figure 3. Investigating the logging data to find the cause of the peaks indicated in the graph led back not only to temporary name resolution problems, but also to authentication issues on this host.

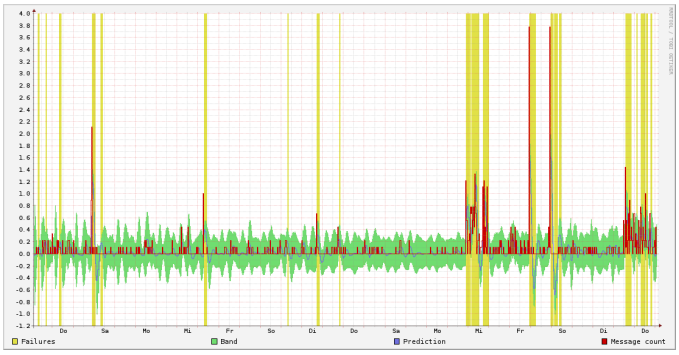

Fig. 3. Holt-Winters based vertical bands representing high occurrences of postfix/smtpd warning messages.

On one hand, the visualization of logging data, as shown in Figure 3, can be used to support network operators trying to find and investigate problems in the monitored network. On the other hand, it is a convenient approach to identify interesting logging data in order to construct suitable rules for aggregation, correlation, and of course anomaly detection. It is also imaginable that this use of time-series algorithms provides a way to realize an automated generation of rulesets for detecting aberrant behavior in logging data. One prerequisite, however, is the feasibility of such algorithms to work on highly-structured data, which allows them to unambiguously identify certain events, as a similar method shows [29].

V. Implementation of Log Correlation and Consolidation in Cloud Environments

To facilitate the analysis of security event messages in distributed cloud environments and to reduce the amount of logging data that needs to be permanently stored, we presented a log correlation and consolidation prototype in [1]. In this section we will give an overview of the functional principle of the prototype and an example application of our solution by showing the correlation of logging data being generated during an SSH brute-force attack as described in Section IV-A.

A. Mode of Operation

As we already mentioned, our prototypical implementation uses rsyslog [15] as the central syslog server, which receives and normalizes syslog messages originating from distributed sources. Since the majority of the compute cloud providers offer syslog-based logging in their VMs, our entire approach using rsyslog can be used to correlate the logging data across multiple and heterogeneous cloud environments (e.g., cloud federations or hybrid clouds). The Complex Event Processing (CEP) Engine Drools Fusion [30] is used as a basis for a correlation and consolidation prototype, which was implemented in Java. Finally, we use the Elasticsearch [17] document store for permanent persistence of all the received and correlated syslog messages.

As pictured in Figure 4, syslog messages are sent from the distributed clients — either virtual machines or physical hosts — to the rsyslog server using a TLS connection over TCP. While this will slightly increase the overhead for processing and transmitting the messages, it guarantees reliable delivery as well as authenticity and privacy of the received messages.

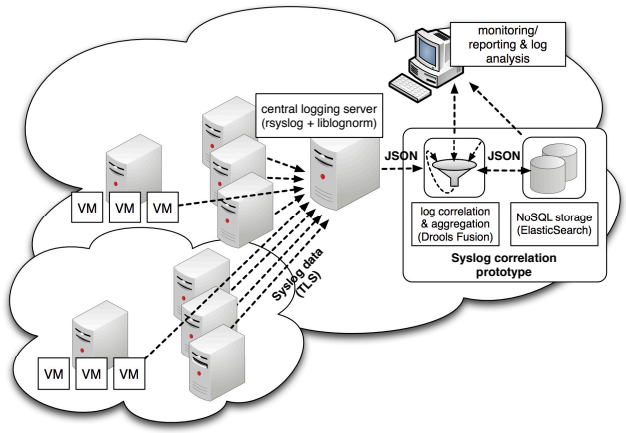

Fig. 4. Correlation and aggregation of logging data at a central syslog server.

Normalization of the received messages is performed at the central syslog server using liblognorm [31], a high-performance log normalization library for rsyslog. The use of this library enables us to parse incoming messages from different sources by specifying normalization rules in a syntax similar to regular expressions before relaying the messages to the designated output module. From the parsed messages, we then extract and process valuable information and add it back to the original syslog message using an RFC 5424 [12] compliant structured log format. It is obvious that this normalization step is needed to assure that messages originating from different clients, which may use a slightly different message format for the same type of event, can be handled in a unified way. More important, the identification of diverse

messages having the same meaning in terms of correlation can be identified and denoted by adding unique tags to the structured information of the original messages, which can easily be interpreted by our correlation prototype.

Listing 3 shows an example of rules that can be used with liblognorm to normalize syslog messages indicating successful or failed SSH password login attempts. During normalization, the values for username, source IP address and TCP port number as well as the SSH protocol version used in the authentication attempt are identified and provided as structured data in the processed syslog messages. Most important, a list of tags (labels) can be added to the message, indicating the type of event that was denoted by the parsed message. In our example, the tags SSHSUCCESS or SSHFAILURE are added to messages depending on the type of event, whereas these tags may be assigned to multiple various messages pointing to the same security event. We illustrate this by the two latter rules, which both get assigned the tag SSHFAILURE. For performance reasons, the complete rule base gets merged into an optimized parser tree by liblognorm, allowing effective analysis of the received messages.

```
rule=SSHSUCCESS: Accepted password for %user:
    word% from %ip:ipv4% port %port:number% %
    protocol:word%

rule=SSHFAILURE: Failed password for %user:
    word% from %ip:ipv4% port %port:number% %
    protocol:word%

rule=SSHFAILURE: Failed password for invalid
    user %user:word% from %ip:ipv4% port %port:
    number% %protocol:word%
```

Listing 3. Example normalization rules for matching SSH password authentication attempts.

The normalization step still has a heavy impact on performance due to the string matching operations that must be performed on all incoming syslog messages. For this reason, we make use of the ruleset feature in rsyslog, which allows us to have multiple input queues for message submission. In each queue we apply only a subset of specific normalization rules based on the type of the incoming messages. On the one hand, this minimizes the number of rules that need to be matched against the incoming messages; on the other hand, it allows us to distribute our whole prototype over multiple cloud instances as described in more detail in Section VI. The decision to use rulesets would also allow us to forward messages directly to the storage back-end in case they are not affected by correlation. However, an evaluation study we present in Section VII shows that in case of an Elasticsearch back-end this has a negative impact on performance.

After normalizing, the logging information is serialized using JavaScript Object Notation (JSON) and forwarded to an appropriate rsyslog output module, which connects to our correlation prototype. Listing 4 depicts an example of a structured syslog message, providing the most useful information from the originating syslog message through a data substructure and the type of event through its list of tags. The prototype embodies a correlation engine, which analyzes the messages and also instantly transfers them into an Elasticsearch cluster for permanent persistence. Therefore, our prototype utilizes the Elasticsearch Java API to become part of the cluster as a transparent node not storing data itself, but forwarding it to an appropriate data nodes.

```
{
  "data": {
        "protocol": "ssh2",
        "port": "54548",
        "ip": "10.0.23.4",
        "user": "root"
  },
  "time": "2014-01-29T16:06:00.000",
  "host": "test.example.com",
  "facility": "auth",
  "severity": "info",
  "program": "sshd",
  "message": "Failed password for root from
      10.0.23.4 port 54548 ssh2",
  "tags": [ "SSHFAILURE" ]
}
```

Listing 4. Structured syslog message of a failed SSH password authentication attempt.

Our implementation of the correlation prototype is based on the Complex Event Processing (CEP) Engine Drools Fusion [30], which supports temporal reasoning on a stream of data, allowing us to extend events with a property containing the time of occurrence provided by the syslog message's timestamp value. This enables us to define rules that may consider a number of similar messages in a specific time interval when evaluating the importance of incidents at the monitored systems. The rules can easily be extended to provide arbitrarily complex correlation and consolidation techniques. A more detailed example of possible rules used in the correlation engine is shown in Section V-B.

Drools automatically keeps track of all events that any of the rules may apply to, using an in-memory cache. Messages that do not match any of the rules, in contrast, will be removed from the in-memory cache. However, 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 periodically searching for successful correlation flags in the Elasticsearch cluster and pruning the original messages they refer to, we can achieve a consolidation.

B. Example Application

As an example application of our prototype we are showing 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 of high priority in the case of a successful SSH login, which follows immediately after a certain number of failed logins, hence pointing to an SSH brute-force attack, which possibly succeeded. The detection of this scenario should be carried out completely autonomously by our prototype. In order to detect such a scenario, first we need to match the corresponding syslog messages of failed and successful SSH logins. These messages need to be isolated and filtered from the stream of logging data originating from the syslog server to trigger certain operations on them. Since we

already normalized incoming syslog messages at the central syslog server using liblognorm, we can easily recognize all interesting events by their list of tags, which we previously added to the structured data values of the messages. An example of an unsuccessful SSH authentication attempt message including its additional tag was already shown in Listing 4.

To accomplish the detection of a successful SSH brute-force attack we utilize the temporal reasoning features of Drools Fusion [30], which allow us to construct rules that describe particular events by a sequence of specific syslog messages within a certain time. First, we need to detect an ongoing SSH brute-force attack, before we can then search for subsequent successful logins. Therefore, we specify a rule entitled ”SSH brute-force attempt” as shown in Listing 5, which will match syslog messages containing the value SSHFAILURE in the list of their tags. Furthermore, by evaluating the values of the producing host and the username used to authenticate, we build up multiple in-memory queues of various authentication attempts. If we can match a series of ten failed messages within a one minute time window concerning a specific user account at one specific system, we assume a certain chance of an ongoing brute-force attack. Our current implementation shown in Listing 5 then retracts all messages from the time frame and generates a new syslog message with the facility ”security” and severity ”warning”, containing the message ”SSH brute-force attack” together with further additional information. Beyond that, the new value BRUTEFORCE is added to the list of message tags, allowing us to easily identify the message in a second rule described below.

```
rule "SSH brute-force attempt"
no-loop
when
  Message ($host: host,
           $user: data["user"])
  $atts: CopyOnWriteArrayList(size >= 10)
    from collect(
      Message(tags contains "SSHFAILURE",
              host == $host,
              data["user"] == $user)
      over window:time(1m))
then
  Message last = (Message) $atts.get($atts.
      size() - 1);

  for (Object f : $atts) {
    retract(f);
  }

  insert(messageFactory(last)
    .setTime(last.getTime())
    .setSeverity(Message.Severity.WARNING)
    .setFacility(Message.Facility.SECURITY)
    .setMessage("SSH brute-force attack " +
        "for @{data.user} from @{data.ip}")
    .addTag("BRUTEFORCE")
    .message());
end
```

Listing 5. Drools fusion rule to detect running SSH brute-force attacks.

It would also be possible to postpone the persistence of the logging data until the correlation is finished, to store all messages related to the attack with a higher priority (e.g., emergency) in the Elasticsearch cluster. Another possibility would be to persist the generated messages in addition to the existing ones instead of retracting the original messages. As described earlier, messages still needed for correlation are automatically kept in the in-memory cache by Drools Fusion according to the rules we defined, whereas messages that no longer match any of the rules get removed from the cache, self-controlled by Drools.

A second rule ”Successful SSH brute-force attack”, which we illustrate in Listing 6 matches successful SSH logins after a brute-force attempt was recognized. This is done by detecting a successful authentication message with the tag SSHSUCCESS within a ten-second window after a message containing the tags SSHFAILURE and BRUTEFORCE was recognized, containing the same username respectively. In that case, the message indicating the successful login is altered by increasing the severity to ”emergency” and adding another value ”INCIDENT” to the list of tags, to ease traceability of the security event message. Additionally, a short description is appended to the textual message to make it more meaningful.

```
rule "Successful SSH brute-force attack"
no-loop
when
  $att: Message(tags contains "SSHFAILURE",
                tags contains "BRUTEFORCE",
                $host: host,
                $user: data["user"])
  $suc: Message(host == $host,
                data["user"] == $user,
                tags contains "SSHSUCCESS",
                this finishes[10s] $att)
then
  $att.addTag("INCIDENT");
  $att.setSeverity(Severity.EMERGENCY);
  $att.setMessage($att.getMessage()
    + " [bruteforce]");

  update($att);
end
```

Listing 6. Drools fusion rule to detect successful SSH brute-force attacks.

Figure 5 illustrates how the detection of a successful SSH brute-force attack is displayed in the user interface of our prototype. In the example shown, the correlated message has been logged in addition to the individual login attempt messages. Obviously, the increased severity of the security event is more likely to be recognized by an operator inspecting the syslog messages than events with the regular severity info.

Severity	Facility	Message
emerge	security	Accepted password for root from 10.0.23.4 port 54548 ssh2 [bruteforce]
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
warn	security	SSH brute-force attack for root from 10.0.23.4
info	auth	Connection closed by 10.0.23.4 [preauth]
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2
info	auth	Failed password for root from 10.0.23.4 port 54548 ssh2

Fig. 5. Extract of our prototype’s user interface showing the detection of a successful SSH brute-force attack.

Instead of altering the matching syslog message, it would also be reasonable to generate a completely new one as shown in Listing 5. An even more interesting approach that we briefly discuss in Section VIII is to trigger an event in an existing network management system like OpenNMS [32].

VI. Using the prototype to enable log correlation in an OpenStack Cloud Environment

The in-memory cache of Drools Fusion limits the number of events our engine is able to correlate. Also, a single engine instance limits the scalability, which is a major drawback in cloud environments that should rather scale elastically both up and down with respect to performance. To overcome these limitations and also to test our implementation in a cloud environment, we integrated our correlation engine in a Rackspace Private Cloud [33] test-bed based on OpenStack Havana [34]. Our syslog correlation prototype shown in Figure 4 was set up as an OpenStack Nova VM instance using an Ubuntu 12.04.3 LTS Cloud Image. Subsequently, Java 1.7 and the code for our prototype (named jCorrelat) were installed.

A. Scalability of our prototype in an OpenStack environment

While the CPU performance and memory capacity of the VM instance could be resized using OpenStack Nova, or respectively the hypervisor beneath it, a comprehensive scale-out solution in the cloud demands the ability to scale over multiple instances. Hence, multiple instances running our prototype should be launched. To balance the load across these VM instances, an additional implementation that distributes the log messages across the instances was evaluated. Furthermore, we considered as an option using the load-balancing facility of OpenStack's networking component Neutron. While both solutions are feasible, the practicability to enhance the scalability of our prototype using these approaches is rather limited. This is due to the fact that, if the correlation were carried out across multiple VM instances, these instances would need some sort of shared or distributed memory. While there are solutions for Java-based, distributed, in-memory data stores, e.g., Terracotta BigMemory [35] or Hazelcast [36], these solutions are rather expensive and also increase the complexity of the management of our solution.

Therefore, we chose a simpler approach that forwards logging information to the corresponding VM instance based on the application it was received from. To accomplish this distribution, multiple output modules were defined in the centralized rsyslog logging server. Figure 6 depicts the usage of multiple VM instances of our correlation engine in an OpenStack test-bed. Each instance receives logging data for a specific application. If the correlation of logging data originating from different applications is required, logging information from multiple applications could be sent to a single VM instance as shown in Figure 6. If the correlation needs more memory, e.g., due to complex rulesets that need to store the logging data for a long period in the in-memory database, distributed memory techniques as described above, e.g., using [35] or [36] could be added. This way, the in-memory cache for the correlation engine stores the logging data across multiple VM instances. Using queries to external data stores for the correlation, e.g., searching in the persisted messages in Elasticsearch, did not prove successful in our tests, as the latency added to each incoming log message is too high to handle bursts of logging information coming from the applications.

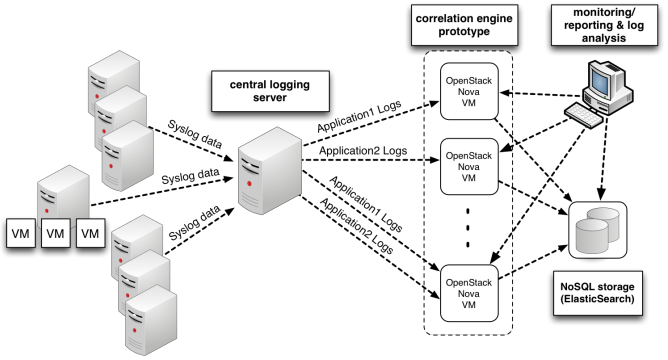

Fig. 6. Scale-out solution for our correlation engine using OpenStack.

As described in Section V, our correlation engine is based on Drools Fusion. Though discussions about a distributed setup can be found in the Drools forums (e.g., the discontinued Drools Grid), Drools is currently using an extended Rete algorithm [37] for its rule engine that relies on shared memory and is therefore limited to a single machine. Hence, our solution to use separate Drools VM instances in OpenStack Nova is currently sufficient for our experiments. Another interesting possibility could be the integration with the Storm distributed and fault-tolerant realtime computation framework [38].

Since we presented our first research results in [1], the OpenStack Havana release has brought two new components that can be used for our distributed syslog correlation engine shown in Figure 6. The first is Heat, which offers an OpenStack orchestration API. Using this API, the setup we described in Figure 6 can be implemented as a Heat template. This template describes the entire environment and VM instance configuration for our correlation engine prototype, hence allowing an automatic deployment within an OpenStack environment. This orchestration could also be combined with configuration management tools like Chef [39] that handle automatic preparation of the operating system and our correlation engine within the VM instance. Furthermore, Heat manages the lifecycle needs of the instance, e.g., scaling instances up and down, but especially stopping and starting new ones. This way, Heat supports an auto-scaling mechanism that enables us to automatically start more VM instances of our correlation engine, e.g., in case of an increasing logging volume.

The auto-scaling mechanism can also include the distribution of logging data originating from a specific application. For example, it is possible to include the name of the application in the name of the VM instance, which can then be used by the central logging server to distribute the logging data to specific correlation instances. As the auto-scaling could lead to high resource usage, the maximum number of instances should be limited. Also, an accounting mechanism for the VM instance usage is necessary. Such an accounting mechanism is offered by another component included in OpenStack Havana, named Ceilometer, which is closely integrated with Heat. Ceilometer offers a standardized interface to collect measurements and accounting information within OpenStack. This opens up an interesting approach to ensure the accounting of our instances.

A promising concept is also proposed by the integration of an Elasticsearch storage driver for Ceilometer in one of the forthcoming OpenStack releases [40]. This way, a standardized way to collect logging information in an OpenStack environment could be combined with the advanced log analysis and long-term storage capabilities offered by Elasticsearch as described in Section II-C. In [40], syslog and logging information coming from other applications within an OpenStack environment has already been considered. If this project would find its way into a new version of OpenStack, we could integrate our correlation engine in between, thereby getting a standardized interface to log data in an OpenStack environment, while also enabling the correlation and consolidation of logging events.

B. Integration of non-syslog logging data

Regarding the OpenStack enterprise cloud test-bed we described in the previous section, besides consuming syslog based logging data, OpenStack also produces its own individual type of log files. OpenStack log files use a structured log format that contains a timestamp, the process ID, the severity and the log message in each line [41]. The log message often spreads across multiple lines as the log is filled with Python tracebacks. One way of using this log data in our correlation engine would be to send it directly to our prototype. As the structured log format used by OpenStack is not fundamentally different from the syslog format described in Section II-B, the adaptation of our prototype to consume OpenStack's Python-based logs is easy to implement. However, the most appropriate way to integrate OpenStack logging in our correlation engine is the configuration of OpenStack to use centralized logging to our rsyslog server, as described in [41]. This can also be implemented for a variety of other application-specific non-syslog logging mechanisms, e.g., log4j, log4net [42]. Due to syslog being the de-facto standard especially for logging in Linux based environments, a large number of application-specific log files and formats can be sent to or consumed by current syslog server implementations. Another benefit of converting application-specific logs to syslog messages, beyond the centralization of the logging data, is normalization as described in Section V and an overall decrease in complexity due to the unified logging.

C. Correlation with information from external management systems

While logging data that is used for the correlation in our prototype should be converted to a syslog-based format (as described in the previous section), information originating from management systems like network management and monitoring, system management, service management or facility management could be valuable to correlate events in the log with events from these external management systems. An example could be the physical location of the node that runs the service in a data center or context information like scheduled downtimes for a service during which certain events should be ignored. This sort of filtering also increases the performance of the correlation as rules could be tailored to take such contextual factors (e.g., downtimes) into account.

Since we use JSON to send the logging data to our correlation prototype, as described in Section V, events coming from external management systems could be injected using a simple TCP or web-service interface. Such a service could be used either to push events from the management system to our correlation engine prototype or to periodically pull information from the management systems. In our OpenStack cloud environment, for example, we use OpenNMS [32] as a network management system that collects monitoring information from the systems and network equipment. By using the events from external network, system and service management systems, the output of the correlation engine could also in turn be used as a feedback for these systems. One example could be the automatic creation of a trouble ticket if a ruleset in our prototype is positively evaluated. Normally, such an automatic creation could lead to a large number of open tickets, but due to the correlation and especially consolidation of events offered by our prototype, the quality of the information and integration with trouble ticket or service management systems could instead be improved.

VII. Performance Evaluation

In this section, we present the results of our performance evaluation study in which we monitored the number of processed syslog packets considering various storage back-ends for permanent data persistence.

A. Test-bed setup

For the performance evaluation we set up a test-bed using the Rackspace Private Cloud as delineated in Section VI. Our prototype runs on an Ubuntu 12.04.3 LTS Cloud Image, which has been assigned four Intel i7 CPU cores at 2.80 GHz, a total of 8 GB memory and a solid-state drive (SSD) storage device. On the same system we have set up rsyslog as the central syslog server together with liblognorm, which we utilize to apply the normalization rules described in Section V. Permanent persistence of syslog messages is done using various storage back-ends that are also set up on the same virtual machine and available to our prototype and the rsyslog server through a TCP socket.

The transport of syslog messages from distributed sources is simulated using loggen [43], a syslog message generation tool provided by the syslog-ng project for testing purposes, which has been installed on a remote machine. The tool may be configured to generate random syslog messages as well as injecting real messages by doing loop-reads on a prepared file. For message submission we provide a dedicated 1 Gbit/s Ethernet connection to the generating machine, connecting it over TCP. The use of any security features like TLS or signatures was renounced in our test in favor of throughput performance.

In order to compare the performance impact of our correlation and consolidation prototype, we set up various storage back-ends and ran the test explained below several times, each time configuring an appropriate rsyslog output module. The individually used storage back-ends and the corresponding output modules are presented in Table I.

B. Evaluation of the test-beds' peak performance

According to Gerhards [44], the rsyslog server is capable of processing up to 250k messages per second over the network. To prove the correctness of our test-bed setup, in a first

TABLE I. UTILIZATION OF STORAGE BACK-ENDS AND RSYSLOG OUTPUT MODULES

Backend type	Output module	Description
File	omfile	Persist messages to a single file on disk
MySQL	ommysql	Persist messages into MySQL using the corresponding TCP socket
Elasticsearch	omelasticsearch	Persist messages into Elasticsearch using the corresponding REST API
jCorrelat (1)	omtcp	Forward messages to our prototype, which persists them into Elasticsearch without applying any correlation rules
jCorrelat (2)	omtcp	Forward messages to our prototype, which persists them into Elasticsearch after applying the appropriate correlation rules

experiment we examined the actual peak performance without any bottlenecks like normalization and correlation or disk I/O latency. Therefore, we configured loggen to generate random syslog messages of different sizes (256, 512 and 1024 byte) and submit these to the central syslog server as fast as possible using six concurrent TCP stream-sockets. The syslog server immediately forwards the received messages to the /dev/null device using the file output module and does not apply any further processing like message normalization or correlation techniques.

Figure 7 depicts our result, showing that the number of messages processed by the central syslog server is dependent on the message size. At an average message size of 512 byte, we get rather close to the number that was stated in [44]. In addition, the graph on the right side of Figure 7 illustrates that we are able to saturate the 1 Gbit/s Ethernet link to about 85.5% of capacity in all of the test cases.

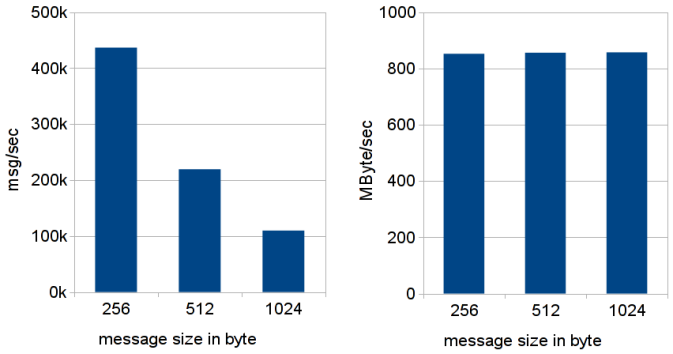

Fig. 7. Maximum syslog packet throughput (left) and bandwidth utilization (right) without any normalization and correlation.

C. Evaluation of the correlation prototype performance

This time, our test setup uses liblognorm at the central syslog server to apply normalization rules as described in Section V. For message generation we again use loggen, which we advised to do loop-reads on a prepared file containing real syslog messages that were previously collected in our test environment. The file contains a total of 20,000 messages including a vast variety of different syslog facilities, one-third of which are authentication related messages including various SSH brute-force attempts.

Figure 8 summarizes the results of our tests. It clearly shows that the best performance can be achieved when writing to a single syslog file on disk, which is not surprising as it does not involve the overhead of sending messages over a TCP socket, like in the other test cases. However, the use of the file storage back-end is included only for comparison, as it appears obvious that its application is not practicable in a centralized syslog environment due to the huge amount of messages from different sources and the lack of any reasonable analysis capabilities. Also, when persisting messages into MySQL, we experienced good performance, but analyzing the logging data afterwards is not easy as the read performance decreases with the number of stored messages. A countermeasure would be indexing, but this in turn creates a heavy burden on write performance. Message persistence into Elasticsearch was done using rsyslog's corresponding output module, which uses the Elasticsearch representational state transfer (REST) interface to insert logging data. As shown in Figure 8, the write performance of this method is poorer than writing to MySQL, most likely because the REST interface involves the overhead of using HTTP when submitting messages.

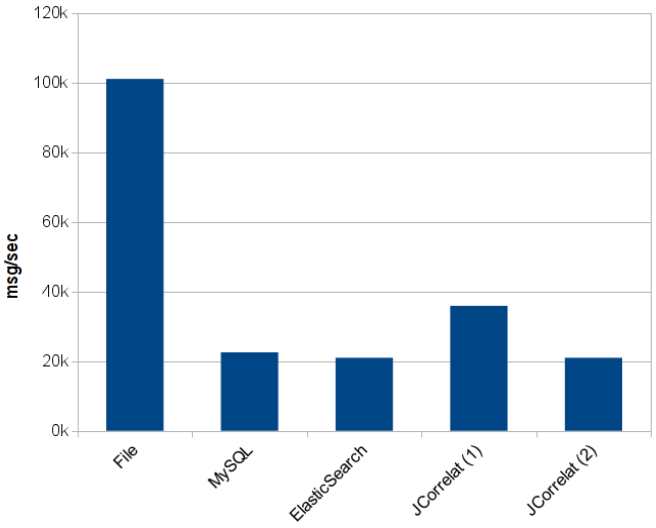

Fig. 8. Comparison of message throughput with different storage back-ends. Our prototype appears twice showing its performance without any correlation rules installed (jCorrelat (1)) and with the rules for SSH brute-force detection applied (jCorrelat (2)).

In the case of sending messages to our prototype, we first measured the message throughput without performing message correlation by simply removing any rules, hence operating in transparent mode. In contrast to using the Elasticsearch REST interface directly via an appropriate rsyslog output module, we get much better performance when using our prototype without applying any correlation. The reason is the usage of the Elasticsearch Java API in our prototype, which allows it to join the Elasticsearch cluster, acting as a transparent node by simply forwarding all messages to the actual data nodes. This way, we are able to decrease the overhead of sending via REST and consequently increase performance.

Finally, when activating message correlation in our prototype as described in Section V-B, we achieve roughly the same throughput we would get by sending messages using the Elasticsearch REST interface or the MySQL back-end, but with the benefit of increased significance of the persisted security event messages. For this reason, we argue that the

application of our prototype is perfectly suitable for logging environments that are already using one of the discussed SQL or NoSQL back-end types. In addition, a feature we described briefly in Section V-A but did not use in our performance evaluation is rsyslog's rulesets facility. It allows normalization rules to be applied only to messages matching a certain ruleset (i.e., matching the syslog facility auth), which would reduce the application of normalization rules to only one-third of the syslog messages we sent to our prototype. As normalization uses string parsing operations, which are known to be CPU intensive, using this method would allow us to reduce normalization overhead and increase performance even further.

VIII. 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 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 analyzing logging data, especially in distributed environments with many 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 Sections IV-A and V-B. 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.

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 Elasticsearch's 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). For example, we could integrate this approach into the OpenStack-based cloud environment presented in Section VI.

Our predefined ruleset 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 [22], [23] and [24]. Another possible topic for future research is the integration of existing knowledge-based systems and automated reasoning as developed, e.g., for network anomaly and intrusion detection systems (IDS) [45]. 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. [46] describe 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 in [29], 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 forecasting 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. A starting point for further research could be the use of confidence bands generated by the Holt-Winters algorithm shown in Section IV-C. To detect failures and error conditions in cloud environments this has already been proposed in [47]. We will evaluate the extension of this approach to allow for the correlation and aggregation of logging data in enterprise cloud environments.

Since we published our first research results in [1], Amazon Web Services released the Kinesis cloud service, which offers real-time processing of streaming data at large scale [48]. While this solution seems to offer a promising alternative for the correlation and consolidation of logging data within the Amazon Cloud, moving large amounts of logging data from enterprise clouds towards Amazon presents an obstacle. Additionally, Kinesis is based on streaming, rather than temporal reasoning or complex event processing as described in Section V-B. The scalability of Kinesis, however, is paramount, and we are evaluating integration of techniques like Twitter Storm [38] in our solution. In this respect, Esper [49] also presents an interesting alternative to Drools Fusion and offers high scalability when used in combination with Storm. The visualization and analysis of logging data in an Elasticsearch cluster could also perhaps be improved by enabling time-based comparisons and corresponding plots using Kibana [50], which integrates seamlessly with Elasticsearch.

References

[1] C. Pape, S. Reissmann, and S. Rieger, "RESTful Correlation and Consolidation of Distributed Logging Data in Cloud Environments," In Proceedings of the Eighth International Conference on Internet and Web Applications and Services (ICIW), 2013, pp. 194–199.

[2] Amazon Web Services Inc., "Amazon CloudWatch," https://aws.amazon.com/cloudwatch/ 2014.05.30.

[3] 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.

[4] C. Lonvick, "RFC 3164: The BSD syslog protocol," Request for Comments, IETF, 2001.

[5] P.V. Mockapetris, "RFC 1034: Domain names - concepts and facilities," Request for Comments, IETF, 1987.

[6] P.V. Mockapetris, "RFC 1035: Domain names - implementation and specification," Request for Comments, IETF, 1987.

[7] R. Hinden and S. Deering, "RFC 2373: IP Version 6 Addressing Architecture," Request for Comments, IETF, 1998.

[8] C. Lonvick, "RFC 3195: Reliable Delivery for syslog," Request for Comments, IETF, 2001.

[9] F. Miao, Y. Ma, and J. Salowey, "RFC 5425: Transport Layer Security (TLS) Transport Mapping for Syslog," Request for Comments, IETF, 2009.

[10] K. E. Nawyn, "A security analysis of system event logging with syslog," As part of the Information Security Reading Room, SANS Institute, 2003.

[11] F. von Eye, D. Schmitz, and W. Hommel, "SLOPPI - A Framework for Secure Logging with Privacy Protection and Integrity," In Proceedings of the Eighth International Conference on Internet Monitoring and Protection (ICIMP), 2013, pp. 14–19.

[12] R. Gerhards, "RFC 5424: The syslog protocol," Request for Comments, IETF, 2009.

[13] G. Klyne and C. Newman, "RFC 3339: Date and Time on the Internet: Timestamps," Request for Comments, IETF, 2002.

[14] BalaBit IT-Security, "Multiplatform Syslog server and logging daemon," http://www.balabit.com/network-security/syslog-ng 2014.05.30.

[15] R. Gerhards, "The enhanced syslogd for linux and unix rsyslog," http://www.rsyslog.com, 2014.05.30.

[16] R. Cattell, "Scalable SQL and NoSQL data stores," ACM SIGMOD Record 39, no. 4, 2011, pp. 12–27.

[17] Elasticsearch Global BV, "Open Source Distributed Real Time Search & Analytics," http://www.elasticsearch.org, 2014.05.30.

[18] Apache Software Foundation, "Apache Lucene," http://lucene.apache.org, 2014.05.30.

[19] R. Marty, "Cloud application logging for forensics," In Proceedings of the 26th Symposium on Applied Computing, ACM, 2011, pp. 178–184.

[20] A. Rabkin and R. Katz, "Chukwa: A system for reliable large-scale log collection," In Proceedings of the 24th International Conference on Large Installation System Administration, USENIX, 2010, pp. 1–15.

[21] D. Jayathilake, "Towards structured log analysis," In Proceedings of the International Joint Conference on Computer Science and Software Engineering (JCSSE), 2012, pp. 259–264.

[22] 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.

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

[24] 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 Proceedings of the International Conference on Cloud and Service Computing (CSC), 2011, pp. 354–359.

[25] J. D. Brutlag, "Aberrant Behavior Detection in Time Series for Network Monitoring," In Proceedings of the 14th Systems Administration Conference (LISA), 2000, pp. 139–146.

[26] K. Fukuda, "On the use of weighted syslog time series for anomaly detection," In Proceedings of the International Symposium on Integrated Network Management (IM), IFIP/IEEE, 2011, pp. 393–398.

[27] T. Oetiker, "RRDtool," http://oss.oetiker.ch/rrdtool/, 2014.05.30.

[28] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," In International Journal of Forecasting, vol. 20, no. 1, 2004, pp. 5–10.

[29] A. Clemm and M. Hartwig, "NETradamus: A forecasting system for system event messages," In Proceedings of Network Operations and Management Symposium (NOMS), IEEE, 2010, pp. 623–630.

[30] JBoss Community, "Drools - The Business Logic integration Platform," http://www.jboss.org/drools/, 2014.05.30.

[31] R. Gerhards, "A syslog normalization library," http://www.liblognorm.com, 2014.05.30.

[32] OpenNMS Group, "The OpenNMS project," http://www.opennms.org 2014.05.30.

[33] Rackspace Inc., "Private Cloud Computing, Storage & Hosting by Rackspace & Openstack," http://www.rackspace.com/cloud/private/, 2014.05.30.

[34] OpenStack Foundation, "OpenStack Open Source Cloud Computing Software," http://www.openstack.org, 2014.05.30.

[35] Terracotta, Inc., "BigMemory Terracotta," http://terracotta.org/products/bigmemory/, 2014.05.30.

[36] Hazelcast, "Hazelcast: In-Memory Data Grid," http://www.hazelcast.com, 2014.05.30.

[37] R. B. Doorenbos, "Production matching for large learning systems," PhD Thesis, University of Southern California, 1995.

[38] Apache Software Foundation, "Storm, Distributed and fault-tolerant realtime computation," http://storm-project.net, 2014.05.30.

[39] Chef, "Chef, IT automation for speed and awesomeness," http://www.getchef.com/chef/, 2014.05.30.

[40] OpenStack LaunchPad, "OpenStack Telemetry (Ceilometer), Elasticsearch Storage Driver Support," https://blueprints.launchpad.net/ceilometer/+spec/elasticsearch-driver/, 2014.05.30.

[41] OpenStack Foundation, "OpenStack Operations Guide - Chapter 13. Logging and Monitoring," http://docs.openstack.org/trunk/openstack-ops/content/logging_monitoring.html, 2014.05.30.

[42] Apache Software Foundation, "Apache logging services," http://logging.apache.org, 2014.05.30.

[43] BalaBit IT Security, "The syslog-ng Open Source Edition Guide," http://www.balabit.com/sites/default/files/documents/syslog-ng-ose-3.3-guides/en/syslog-ng-ose-v3.3-guide-admin-en/html/loggen.1.html, 2014.05.30.

[44] R. Gerhards, "rsyslog: going up from 40K messages per second to 250K," http://www.gerhards.net/download/LinuxKongress2010rsyslog.pdf, 2014.05.30.

[45] M. Salem, S. Reissmann, and U. Buehler, "Persistent Dataset Generation using Real-Time Operative Framework," IEEE International Conference on Computing, Networking and Communications (ICNC), IEEE, 2014, pp. 1023–1027.

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

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

[48] Amazon Web Services Inc., "Amazon Kinesis," http://aws.amazon.com/kinesis/, 2014.05.30.

[49] EsperTech Inc., "Esper - Complex Event Processing," http://esper.codehaus.org, 2014.05.30.

[50] Elasticsearch Global BV, "Kibana," http://www.elasticsearch.org/overview/kibana/, 2014.05.30.