2015 IEEE First International Conference on Big Data Computing Service and Applications

Scalable Security Event Aggregation for Situation Analysis

Jinoh Kim, Ilhwan Moon, Kyungil Lee, Sang C. Suh
Department of Computer Science
Texas A&M University, Commerce
Commerce, TX, 75428 USA
Email: Jinoh.Kim@tamuc.edu, mih0718@gmail.com,
klee15@leomail.tamuc.edu, Sang.Suh@tamuc.edu

Ikkyun Kim
Network Security Research Division
ETRI
218 Gajeong-ro, Yuseong-gu, Daejeon, 305-700, Korea,
Email: ikkim21@etri.re.kr

***Abstract*—Cyber-attacks have been evolved in a way to be more sophisticated by employing combinations of attack methodologies with greater impacts. For instance, Advanced Persistent Threats (APTs) employ a set of stealthy hacking processes running over a long period of time, making it much hard to detect. With this trend, the importance of big-data security analytics has taken greater attention since identifying such latest attacks requires large-scale data processing and analysis. In this paper, we present SEAS-MR (Security Event Aggregation System over MapReduce) that facilitates scalable security event aggregation for comprehensive situation analysis. The introduced system provides the following three core functions: (i) periodic aggregation, (ii) on-demand aggregation, and (iii) query support for effective analysis. We describe our design and implementation of the system over MapReduce and high-level query languages, and report our experimental results collected through extensive settings on a Hadoop cluster for performance evaluation and design impacts.**

I. Introduction

Cyber-attacks have been evolved in a way to be more sophisticated by employing combinations of attack methodologies with greater impacts. For instance, the Conficker worm in 2008 overwhelmed almost 25 million computers [1], while the number of victims by the SQL Slammer worm in 2003 was tens of thousands of computers [2]. More seriously, recent Advanced Persistent Threats (APTs) target not only personal computers and servers but also social infrastructures such as nuclear plants with greater intelligence [3]. Some examples of APTs are Stuxnet attacked Iran's nuclear facilities in 2010 [3] and Operation Aurora targeted dozens of IT giant companies including Google [4]. According to a recent report, the total cost of cyber-attacks is over $100 billion each year with growing impacts [5]. The commonality of these attacks is the utilization of combined attack methods over a long period of time for stealthy infiltration, making it much hard to identify [4].

For effective identification of such sophisticated attacks, it is essential to analyze massive security data (e.g., audit logs) that could span over several years. Hence, the size of the event data set to be processed could easily scale up to hundreds of tera-bytes or even more. Due to this reason, big-data security analytics has taken greater attention in the security community. For instance, a DARPA project (known as the Cyber Genome Project) develops security analytics tools and techniques to effectively deal with large-scale security data sets to search common characteristics of attacks [6]. In this work, we focus on scalable security event aggregation that groups events based on certain attributes to compute "situations," as the target application for security analytics. A situation is defined as a group of security events with certain characteristics in common and provides a condensed view of security issues [7].

MapReduce[8] (MR) has been widely accepted as a big-data computing platform with its powerful data processing capability using distributed computing resources, for many big-data computing applications in diverse domains such as bioinformatics [9], [10] and astronomy [11], [12]. Several works in security also considered this parallel computing framework for a certain type of log processing [13], [14], [15]. In this paper, we present SEAS-MR (Security Event Aggregation System over MapReduce) that utilizes MR technologies for providing scalable event aggregation and situation analysis. The presented system provides the following as core functions: (i) *periodic aggregation* against newly collected data within the last interval of time, (ii) *on-demand aggregation* initiated by users for their own analysis, and (iii) *query support* to retrieve aggregated information (i.e., situation instances). We introduce our design and implementation of the system, and explore performance and scalability impacts through extensive experiments conducted on a Hadoop cluster.

Our key contributions is twofold:

- We introduce our design of the system for security event aggregation and situation analysis using big-data computing technologies. The presented system provides periodic and ad hoc aggregation of security events and query support for analytics.
- We present extensive experimental results conducted in a Hadoop cluster for evaluating functional components in the system. For periodic aggregation, we compare the native MR implementation with the Pig script-based implementation. For on-demand aggregation, we consider two different techniques based on Pig [16] to see the performance impact. For query support, we employ several high-level query languages, including Pig [16], Hive [17], Tajo [18], and present their performance impacts.

The paper is organized as follows. In Section II, we

978-1-4799-8128-1/15 $31.00 © 2015 IEEE
DOI 10.1109/BigDataService.2015.28

introduce the concept of MR and high level query languages for big-data analytics. We then present our security event aggregation model and the overview of the system with core functions in Section III. In Section IV, we report our observations and implications through experiments with diverse settings. We finally conclude our presentation in Section VI with a summary.

II. MapReduce and Big Data Analytics

MapReduce is a programming model and framework for parallel, data-intensive computing, which has been widely accepted for many data processing applications in both science and industry [8]. The framework is attractive to users as it hides details of executing such data-intensive tasks in a parallel fashion. For example, users do not need to consider allocation of a set of tasks to multiple resources, fault tolerance in case of node or process failure, job scheduling, and so forth, since the framework provides rich functionality for parallel computing. Consequently, users do not need to care about such complicated parts, but they can focus only on their application itself and data processing strategies.

The MR framework consists of two core functions, $map()$ and $reduce()$. Since the framework assumes parallelism in computing, the $map()$ function is executed by multiple concurrent processes (*mappers*), and similarly the $reduce()$ function is run by a set of processes (*reducers*) for a single job. The number of mappers and reducers thus depends on the number of worker nodes of the cluster. When mappers are assigned to computing resources, $map()$ reads a disjoint input data block from the underlying distributed file system such as GFS [19], and produces intermediate key/value pairs (i.e., $(key1, value1) \rightarrow list(key2, value2)$). As soon as all the map tasks complete the execution, reducers take over the control and run the $reduce()$ function with the intermediate results. In case of using more than a single reducer, intermediate key space is partitioned (e.g., by hashing), and based on the partitioning information each reducer copies the intermediate results whose keys belong to its key space to the local storage. Thus, each intermediate record in mapper nodes is assigned to only a single reducer. The transferred intermediate key/value pairs are then merged in the form of $list(key2, value2) \rightarrow (key2, list(value2))$ by the given $reduce()$ function. The reduce outputs are finally loaded back to the file system and the map/reduce job terminates.

Figure 1 demonstrates an abstract model for event aggregation against a large intrusion log (e.g., a 1 TB intrusion alerts collected over a month), based on the MR framework. A large-scale intrusion log is loaded to the distributed file system before computation. The input data is partitioned into multiple pieces (called "chunks"), each of which is assigned to a mapper. The mapper then reads the assigned data partition and produces intermediate key/value pairs for the aggregation. Reducers wait until all the mappers completed their tasks. Finally, reducers take the intermediate results based on the assigned key space, perform the aggregation function specified, and load the aggregated results to the file system, as shown in the figure.

Hadoop [20] is an open source implementation of the MR framework with a set of parallel computing functions

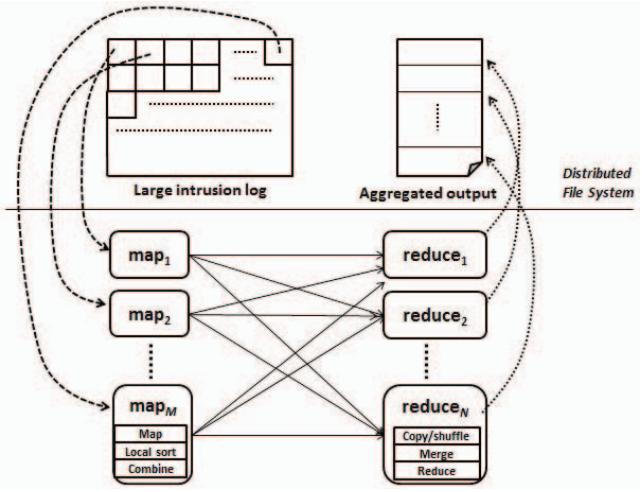


Fig. 1. An example of event aggregation against a large intrusion log using MR. The input data is partitioned into multiple chunks, and each chunk is assigned to a single mapper. The mapper then reads the assigned data partition and produces intermediate key/value pairs for the aggregation, and reducers take the intermediate information based on the assigned key spaces and perform the aggregation function. Finally, the aggregated results are loaded back to the file system.

(e.g., scheduling, monitoring, and management). In addition, Hadoop DFS (HDFS) is a block-structured file system like GFS implemented in the Hadoop software package. With its openness and rich functionality with a variety of auxiliary software such as HBase, Hive, and Pig, to list a few, Hadoop has been widely utilized in many big-data computing applications for parallelization [21]. In this work, we use Hadoop for our exploration.

For big-data analytics in the MR framework, high-level query support is essential. Pig [16] is a high-level computing platform for MR scalable dataflow processing on top of low-level Hadoop Java based API. The domain specific language for this platform is called *Pig Latin*. Hive [17] and Tajo [18] are relational and distributed data warehouse systems (for Hadoop) structured for query and analysis. Hive provides a mechanism to query the data using SQL-like language called HiveQL, and Tajo supports SQL standards. Pig and Hive are higher level query languages (HLQL) over the Hadoop MR platform, whereas Tajo has its own query execution engine on top of HDFS.[1]

Despite the fact that Pig Latin script describes a directed acyclic graph and is procedural (so it can support pipeline splits in the pipeline paradigm), Pig is SQL equivalent in computational power like Hive and Tajo, which implies that they are all relationally complete. Also they offer a set of aggregation functions, such as *average*, *count*, and *sum*, and furthermore, they are easily extended with user defined MR functions written in Java, which gives Turing completeness as Java is a Turing complete language [23]. Some other analytics tools, such as Spark [24] and Impala [25], have also been developed and available to the public. We considered Pig and

[1] A set of the most significant weaknesses and limitations of MR is discussed in detail at a high level, along with solving techniques, which include Pig and Hive [22]. Tajo was introduced as one of the solutions shortly afterward.

Hive in this work since they have long been used, and also picked Tajo as a recent analytics tool but there is no bias and plan to explore other analytics tools.

In this study, we employ the MR framework and related big-data analytics tools for security analytics, specifically for security event aggregation, which summarizes massive security events to uncover correlated activities. In the next section, we introduce our system for security event aggregation with respect to its core functions and design consideration in detail.

III. SEAS-MR: Security Event Aggregation System over MapReduce

In this section, we first discuss the concept of situation analysis based on event aggregation, and describe the system that implements functions for situation analysis using big-data computing technologies.

A. Situation analysis by event aggregation

Event aggregation is a similarity-based analysis technique widely used to identify correlated activities based on frequency information. While isolated events may not be considered significant, any repeatedly occurring ones could be eligible to take more attentions. For example, a distributed denial of service attack may produce a significant number of security events to a *specific* target host within a finite amount of time. A well-known security event aggregation model was introduced in [7] that aggregated events into seven "situations" with three attributes of $< S, T, C >$, where S is a source (e.g., a source IP address), T is a target (e.g., a target IP address), and C stands for an event class (e.g., a CVE number [26]). By definition, a situation is a group of events that have certain common properties, and the seven situations are defined as follows:

- Situation 1: aggregates events based on $< S, T, C >$
- Situation 2-1: aggregates events based on $< S, T >$
- Situation 2-2: aggregates events based on $< T, C >$
- Situation 2-3: aggregates events based on $< S, C >$
- Situation 3-1: aggregates events based on $< S >$
- Situation 3-2: aggregates events based on $< T >$
- Situation 3-2: aggregates events based on $< C >$

Figure 2 shows an example of event aggregation with three input events based on the seven situation model. From the figure, we can see 11 situation instances. Each situation instance consists of "key" as a unique identifer and "cnt" to indicate the number of frequencies.

In this work, we consider an extended situation model for aggregating events with two additional attributes of *source group* (G_S) and *target group* (G_T) to consider collective characteristics of events. For example, a source/target group can be a subnet address in which the source/target host is located. Table I demonstrates the extended model with 14 situations, which is twice the number of situations defined in the basic model. This leads to the fact that our situation analysis model is further data intensive with a greater volume of output since a single event can create up to 14 situation instances. To consider such a highly data-intensive property, we construct our system on the MR platform as will be described next.

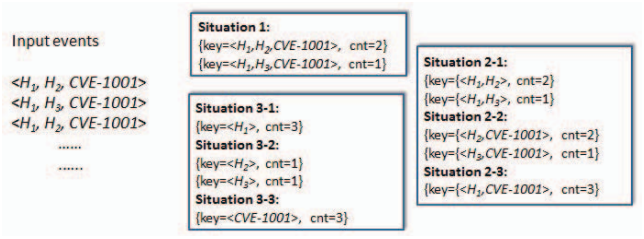


Fig. 2. An example of aggregation with three input events. For situation 1, two situation instances are created by using the three attributes. Five situation instances are created based on two out of three attributes for level 2, while four instances are created for level 3 based on each attribute. Here, Hn refers to host n, and the CVE numbers indicate event classes.

TABLE I. The extended situation table

Situation	Source (S)	Target (T)	Source Group (G_S)	Target Group (G_T)	Class (C)
1-1	✓	✓			✓
1-2		✓	✓		✓
1-3	✓			✓	✓
1-4			✓	✓	✓
2-1	✓	✓			
2-2		✓			✓
2-3	✓				✓
2-4		✓	✓		
2-5	✓			✓	
3-1	✓				
3-2		✓			
3-3					✓
3-4			✓		
3-5				✓	

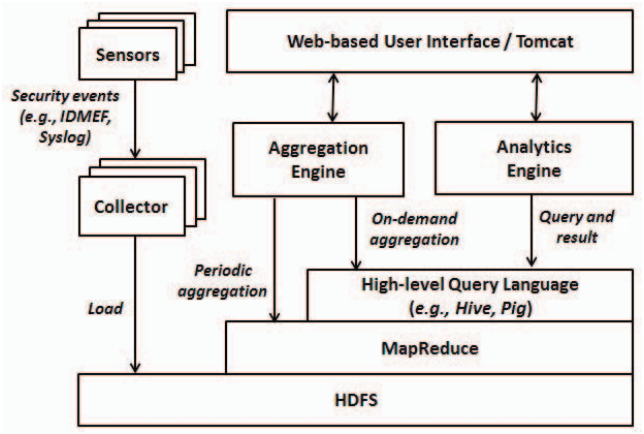


Fig. 3. The system model: It consists of four components: *Sensors* generate raw security event data, such as IDMEF and syslog messages; *Collectors* receive the events from sensors and store the events to the file system after refining; *AggregationEngine* performs event aggregation; *AnalyticsEngine* allows users to search and retrieve aggregated results.

B. System Description

Now we introduce SEAS-MR that we develop for scalable situation analysis. Figure 3 illustrates the system model consisting of four components: *Sensor*, *Collector*, *AggregationEngine*, and *AnalyticsEngine*. *Sensors* generate security events, and *Collectors* receive and store the events to the file

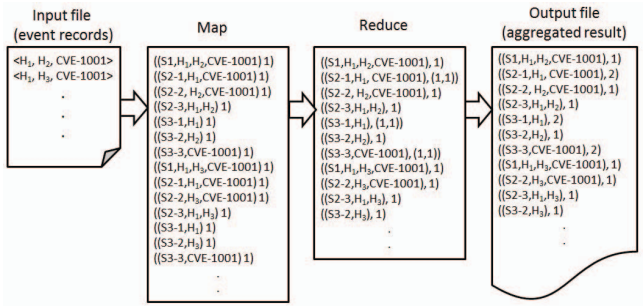


Fig. 4. An example of security event aggregation using MR: $map()$ produces $(situation\_key, 1)$, and $reduce()$ reduces $(situation\_key, list(1)) \rightarrow (situation\_key, frequency)$.

system after refining. There can be multiple sensors dispersed that generate raw security event data, such as IDMEF [27] and syslog [28] messages. A collector periodically loads the events gathered in the last time interval to the file system, with a standardized form of $< S, T, C, G_S, G_T >$ for any single event for future aggregation.

The component of *AggregationEngine* performs event aggregation based on the situation model described in Table I. *AnalyticsEngine* allows users to search and retrieve aggregated results by submitting queries. For example, the security administrator may want to browse situation instances that occurred more than a certain threshold within a specific amount of time. For interfacing with users, the system provides a web-based user interface (running over Apache Tomcat [29]), through which the user not only can initiate event aggregation but also submit queries for analysis.

C. Event Aggregation

As mentioned, *AggregationEngine* provides a set of functions for event aggregation. Figure 4 shows an example of the aggregation process using MR. For simplicity, the example shown in the figure is based on the basic aggregation model with seven situations. Given an input file with a set of events, $map()$ produces key/value pairs for all possible situation instances. A situation instance key consists of a situation category (e.g., Situation 1) and the proper combination of $< S, T, C >$ based on the associated situation category, while the associated value is '1' every time. We refer to the key and value as "situation key" and "frequency". By shuffling, the intermediate results are transformed to $(situation\_key, 1) \rightarrow (situation\_key, list(1))$. $reduce()$ finally merges the frequency values based on the situation key. The reduced results will then be stored back to the distributed file system (e.g., HDFS in the Hadoop cluster).

For event aggregation, SEAS-MR provides the following two types of operations:

- *Periodic aggregation*: Aggregation operations take place regularly against newly collected event data in the last time interval. Thus, the input for an individual run should be disjoint without overlapping with any other run. We assume that the collector creates a single file for each time interval, in which events generated within that time interval are stored.
- *On-demand aggregation*: An aggregation request can be initiated by a user. For example, one may want to launch an aggregation job other than the periodic task (e.g., for a long-term analysis). The user initiating an aggregation job needs to specify the event data sets that he/she wants to analyze (e.g., by providing the beginning and end time).

For the different types of aggregation operations, we implemented two versions: a native map/reduce program for periodic task (as it is better for performance) and a Pig script for on-demand task (better for user interactivity).

There can often be more than a single periodic aggregation task (e.g., 5-minute aggregation and 1-hour aggregation tasks). In that case, there can be two options for any longer term aggregation: one that processes the raw input data and the other that utilizes already aggregated information (if any).

In detail, suppose I_i is a set of input events collected in i-th time interval, and O_i is the associated output (i.e., aggregated results) for I_i:

$$Aggregation : I_i \rightarrow O_i \tag{1}$$

Suppose an aggregation task against multiple time intervals between i and j ($i < j$). And the first option can be defined:

$$Option_1 : I_{i,j} \rightarrow O_i^j \tag{2}$$

where $I_{i,j} = \{I_k | i \leq k \leq j\}$ and O_i^j is the aggregation result against security events collected between time i and j.

The second option is to reuse the already aggregated information, defined as follows:

$$Option_2 : O_{i,j} \rightarrow O_i^j \tag{3}$$

where $O_{i,j} = \{O_k | i \leq k \leq j\}$. Since our application here is commutative and associative, the results of Eq. 2 and Eq. 3 should be identical.

This holds the same for on-demand aggregation. In particular, on-demand aggregation requests can often be made for long-term analysis, we can also consider the both options defined in Eq. 2 and Eq. 3. We will discuss the performance impact of the options in Section IV in detail.

D. Query support

After the aggregation process, the output is stored in a file in the distributed file system. The aggregated information would then be accessed for future analysis. For example, there can be an additional entity that is supposed to raise an alarm whenever it sees any situation instance, the frequency of which exceeds the predetermined threshold. One may also want to access the aggregated information for their own analysis. To enable this, we considered high-level query languages running on the Hadoop platform, including Pig [16], Hive [17], and Tajo [18]. By the help of the query language, the user can submit a query to the system to retrieve the associated situation

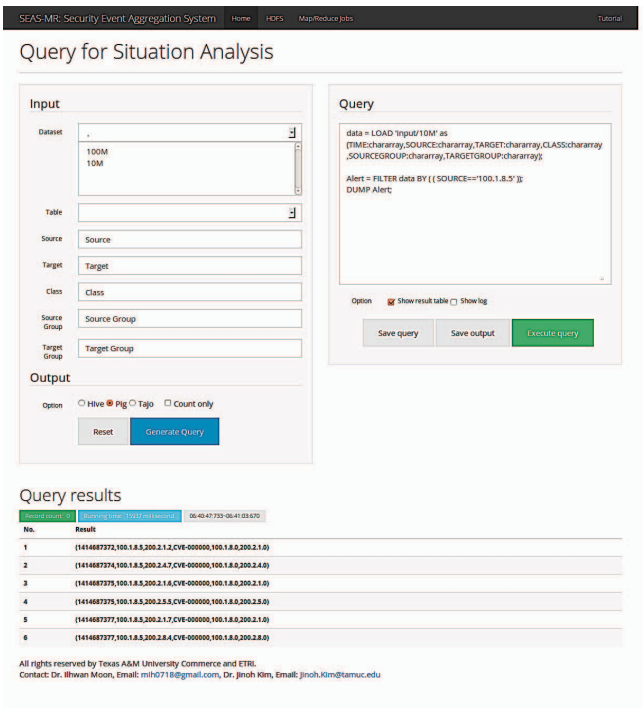


Fig. 5. The web-based user interface for query support

instances. In the following experiment section, we compare performance of the three query languages for our application.

One concern could be that security administrators may not be familiar with high-level query languages. For this reason, we provide a web-based interface demonstrated in Figure 5. Using the GUI, the user can specify search conditions to create a query based on the five tuples of $< S, T, C, G_S, G_T >$. It is also possible to determine one or more files that contain aggregated situation instances for one's analysis. In the figure, the left pane is for query composition, while the right side is a pane for query script. When a user gives one's input and clicks the "Generate Query" button in the left pane, the corresponding query script is displayed in the query pane. Then, the generated query can be executed directly ("Execute Query") or saved for future use ("Save Query"). In case of execution, the query result is displayed in the bottom pane. The result can either be a set of situation instances or a simple counter that indicates the number of situation instances satisfying the given condition.

E. Lazy aggregation

As described earlier, our aggregation process results in a large volume of intermediate and final data, which can be 14 times as many records as the number of input records at max. It is thus a bottleneck point with tremendous data exchange between mappers and reducers and writing to the file system. Optionally, we can consider *lazy aggregation* that performs only exact reductions based on the entire attributes in the aggregation phase. And situation instances are inferred in the analytics phase.

Here is an example of lazy aggregation. For ease of exposition, suppose the basic event aggregation with three attributes of $< S, T, C >$. If we only keeps an associated counter value for each $< S_i, T_j, C_k >$ by lazy aggregation, the size of output of the aggregation process should be less than or equal to the size of input (with respect to the number of records). Inferring the frequency of individual instances can then be performed by simple computations. For example, the frequency of a Situation 2-1 instance for a specific source/target pair, S_i and T_j, can be computed by:

$$\text{Situation 2-1}(S_i, T_j) = \sum_{\forall k} < S_i, T_j, C_k > \qquad (4)$$

By lazy aggregation, it would be possible to reduce the output size dramatically. On the other hand, there would be a trade-off since it may require massive computation for every single query in the analytics phase. We will discuss the trade-off in the following experiment section with our observations.

IV. Experiments

In this section, we present our experimental results conducted on a Hadoop computing cluster. We describe the experimental setting first and then present our observations and implications obtained through experiments.

A. Experimental setting

The evaluation was conducted in a computing cluster that consists of 22 nodes mounted in a rack. Each node has 4 CPU cores, 8 GB memory, and 2TB hard drive, and hence, the cluster system is equipped with 88 CPU cores, 176 GB memory, and 44 TB disk space, collectively. The nodes in the cluster are interconnected through Gigabit Ethernet Switch The Operating System for the cluster is CentOS release 6.5. We installed two versions of Hadoop, v1.2.1 and v2.4.0, because of analytics tools' constraints. For high-level query languages, we installed Pig (v0.11.1), Hive (v0.12.0), and Tajo (v0.9.0), for the experiments. Pig and Hive were operated on Hadoop v1.2.1, while Tajo worked with the later version of Hadoop.

For thorough evaluation, we considered multiple input properties such as size and aggregation ratio. We define *aggregation ratio* ($A.R.$) as the ratio of the number of output records to the number of input records (i.e., $A.R. = \frac{\text{number of input records}}{\text{number of output records}}$). With many more events having identical characteristics, the associated aggregation ratio will become higher since the number of output records should be relatively small in this case, and vice versa. $A.R.$ should be critical to overall performance as it determines intermediate and final output size. To consider various input characteristics, we used synthetic data sets with different size and aggregation ratio.

Table II shows three aggregation ratios we considered in our experiments. From the table, *Agg-L* has a very low aggregation ratio ($A.R. = 0.17$), and the number of output records is six times greater than the number of input records for any *Agg-L* data file. In contrast, *Agg-H* would produce almost the same number of output records as the number of records in the input file with $A.R.$ = 1.0. By default, we used *Agg-M* with $A.R. = 0.4$ unless otherwise noted, with which the output size is 2.5 greater than the input size with respect to the number of records. Aggregation ratios for the three types of data (i.e., *Agg-L*, *Agg-M*, and *Agg-H*) are fairly consistent

TABLE II. AGGREGATION RATIOS USED IN THE EXPERIMENTS

Type	$A.R.$	# output records (for 1M input data)
Agg-L	0.17	6M
Agg-M	0.4	2.5M
Agg-H	1.0	1M

regardless of input data size and the maximum variation is 0.027.

In the input data, we assume that each security event consists of six attributes: <timestamp, source IP address, target IP address, CVE ID, source network address, target network address>. To see scalability, we employed a variety of data sets with a different number of event records from 1 million ($\approx$ 60 MB) to 1 billion ($\approx$ 60 GB) as input.

B. Aggregation performance

We first report scalability our security application on the MR platform. For the experiments, we largely employed Hadoop default parameters except for the some fine tunes as follows. For the number of reduce tasks, we determined it by using the equation of $1.75 \times N \times T_{max}$, where N is the number of nodes and T_{max} is a Hadoop configuration variable (`mapred.tasktracker.tasks.maximum`). Since our application produces a large volume of intermediate data in the map phase, we enabled the combiner function but did not use any compression, based on our preliminary observations (not shown here).

Figure 6 shows performance for scale-up (with respect to data size) and scale-out (with respect to the number of worker nodes). As shown in Figure 6(a), the MR platform is highly scalable for our security application for greater data sets, showing over 18x speedup with 16x computing resources for an input file with one billion records (60 GB). For an input file with 100 million events, the speedup is around 13 with 16x worker nodes. There were no significant benefits of parallelism for relatively small input files ($\leq$ 600 MB). This indicates that *our application of security event aggregation running over the MR platform is highly scalable with greater benefits for larger input data sets with the increasing number of computing resources*.

Figure 6(b) shows the impact of aggregation ratio. As mentioned earlier, smaller A.R yields greater number of intermediate and final output records. While showing very good scalability regardless of $A.R.$, we can see that *Agg-L* requires more time than *Agg-M* that also requires more time than *Agg-L* for job completion. Compared to *Agg-L*, *Agg-M* creates roughly half number of situation instances as output and yielded up to 20% better performance. *Agg-H* generates around 20% compared to *Agg-L* in terms of the number of output records, and we observed up to 30% difference in performance. This indicates that communication is an important part for overall performance while computation is still a dominant element.

We next examine performance with Pig for on-demand aggregation. We used the default setting of Pig without tuning. The left one In Figure 7 illustrates scalability. The right-side figure compares performance for aggregation when using three different techniques for aggregation: (i) by the MR code

TABLE III. QUERIES USED FOR EXPERIMENTS

Query ID	Query	# Hits (%)
Q1	Search a specific S1-1 instance	1 ($< 10^{-9}$)
Q2	Search a specific S2-1 instance	1 ($< 10^{-9}$)
Q3	Search a specific S3-1 instance	1 ($< 10^{-9}$)
Q4	Search entire S1-1 instances	951M (38%)
Q5	Search entire S2-1 instances	10K ($< 10^{-5}$)
Q6	Search entire S3-1 instances	100 ($< 10^{-7}$)
Q7	Search entire S1 instances	2.3B (92%)
Q8	Search entire S2 instances	200M (8%)
Q9	Search entire S3 instances	1M (0.04%)
Q10	Search any instance with counter ≥ 5	300M (12%)

("Native MR"), (ii) by the Pig script against the raw input data ("Pig"), and (iii) by the Pig script against the previously aggregated data ("Pig (reuse)"), based on the discussion in Section III-C.

As shown in Figure 7(a), the trend is similar with the one with MR in Figure 6, but the speedup is fairly limited. The max speedup we observed is 12x with 16x computing resources. Figure 7(b) shows that utilizing existing aggregated information dramatically improves performance for Pig-based aggregation, showing almost comparable performance to native MR. This implies that *using Pig with existing aggregated information can be a vital option for on-demand aggregation.*

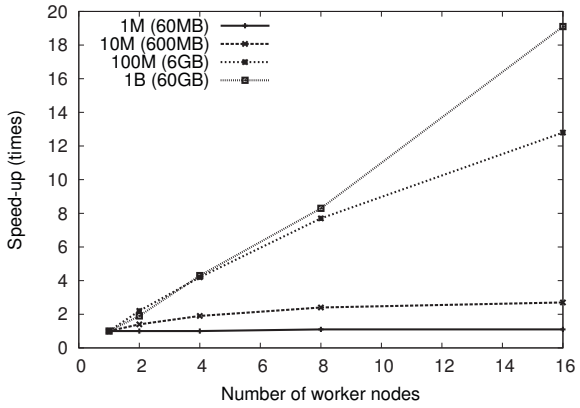
C. Query performance

To evaluate query performance, we considered Pig, Hive, and Tajo, as mentioned. There was no fine tuning for the tools and used the default settings. We set up a diverse set of queries that have the different number of hits as a result of queries, as shown in Table III.

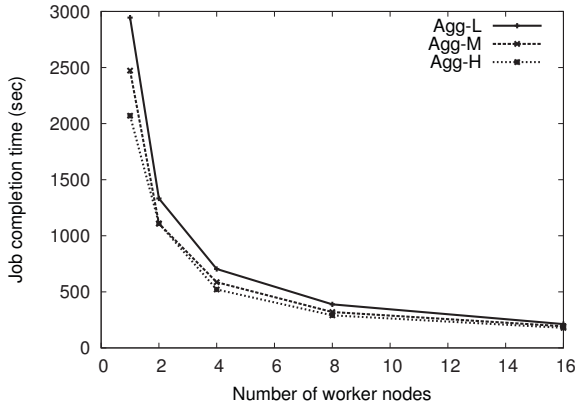
Figure 8 demonstrates query performance when using different analytics tools. In this experiment, we configured a cluster with 16 nodes, and used 1B records with *Agg-M* as input. The input file size is roughly 60 GB, and the size of the corresponding output file after aggregation is about 90 GB with 2.5B situation instances. The left figure shows query response time for retrieval of situation instances, while the right side figure demonstrates the results of count-only queries. For retrieval queries, the resulted instances are written back to a file in DFS when the queries are completed. On the other hand, a count-only query returns the number of situation instances as a result.

As can be seen from the both figures, Tajo significantly outperforms the other two, and Hive follows. For queries for instance retrieval, Tajo yielded 2–6.2x speedup compared to Pig (4.8x on average) and 1.9–4.8x speedup to Hive (3.2x on average) in Figure 8(a). We observed that Q7 and Q4 require much time than the other queries with respect to response time. This is because the two queries (Q4 and Q7) result in significant writing to the file system with a substantial number of hits (38% and 92%, respectively). As in Figure 8(b), on the other hand, we can see that Tajo and Hive work fairly consistent through the queries, while Pig still shows a high degree of variations. With respect to count-only performance, Tajo yielded much better performance: 5–9.8x speed up to Pig and 3–4.6x speed up to Hive.

We also conducted experiments with different data sizes and observed consistent results of Tajo $>$ Hive $>$ Pig for

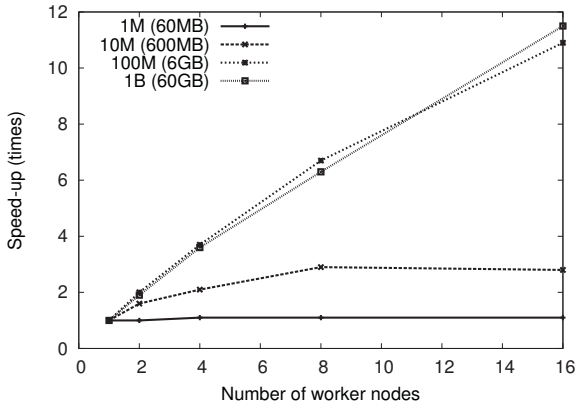


(a) Scale-up/scale-out speedup (Data type: *Agg-M*)

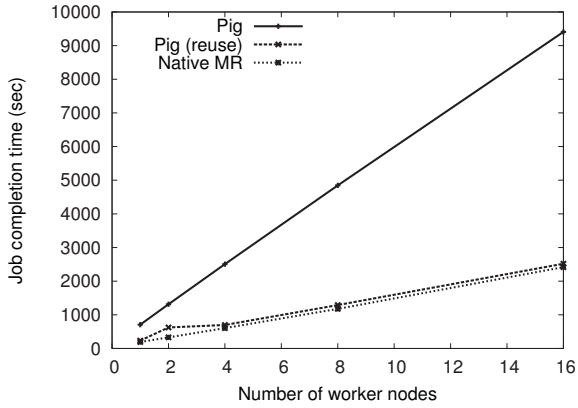


(b) Impact of aggregation ratio (Data size: 1B records)

Fig. 6. MR aggregation performance: (a) It is highly scalable for greater data sets, showing over 18x speedup with 16x computing resources for the biggest data set, while there were no significant benefits of parallelism for small input files; (b) $A.R.$ has a significant impact to overall performance — compared to *Agg-L*, *Agg-M* creates roughly half number of situation instances as output and yielded up to 20% better performance, and *Agg-H* generates around 20% number of output records compared to *Agg-L* with up to 30% difference in performance.

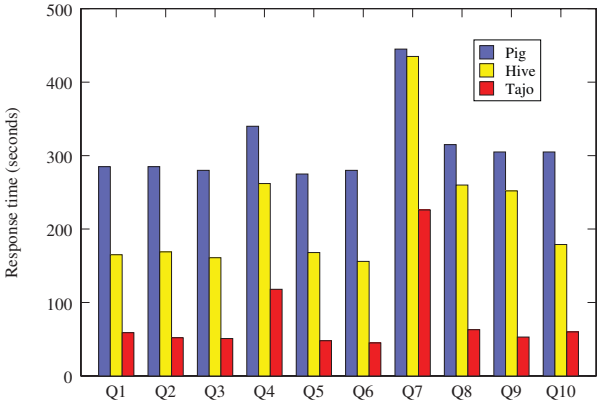


(a) Scale-up/scale-out speedup (Data type: *Agg-M*)

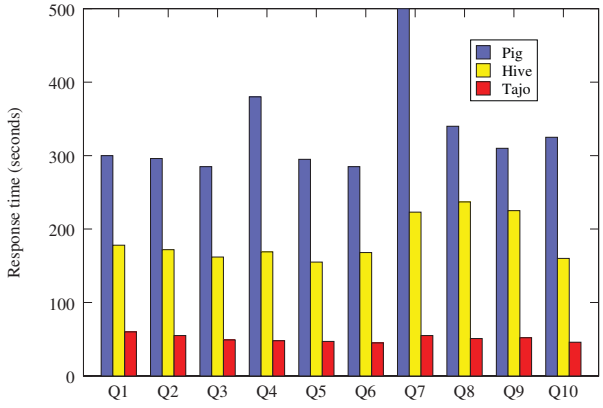


(b) Performance comparison (Data size: 1B records)

Fig. 7. Pig aggregation performance: (a) The speedup is fairly limited up to 12x with 16x computing resources; (b) utilizing existing aggregated information dramatically improves performance for aggregation by Pig, showing almost comparable performance to native MR.



(a) Query performance for retrieval



(b) Query performance for count-only

Fig. 8. Query performance (16 nodes, 1B records, *Agg-M*): Tajo works the best, yielding 2–6.2x speedup compared to Pig (4.8x on average) and 1.9–4.8x speedup to Hive (3.2x on average) for retrieval queries; For count-only queries, Tajo yielded much better performance: 5–9.8x speed up to Pig and 3–4.6x speed up to Hive.

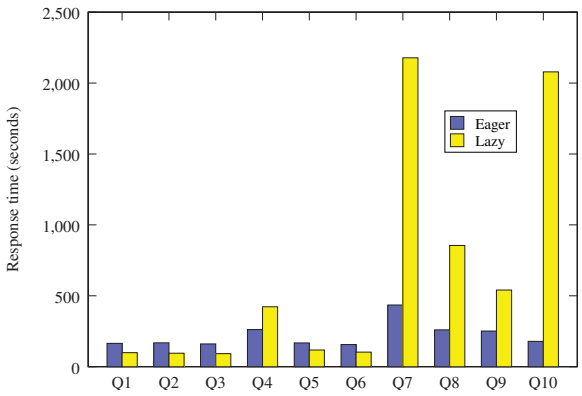


Fig. 9. Query performance for eager vs. lazy aggregation (16 nodes, 1B records, *Agg-M*): Lazy aggregation leads much smaller aggregation time than the non-lazy method, but query performance can be significantly degraded for some queries (e.g., Q7 and Q10) requiring massive computation.

query performance. We conjecture that Tajo works well for our application since the data for queries are well structured fitting to the relational model, for which Tajo is optimized. In sum, *Tajo with its own query engine optimized for the relational data model shows better query performance for situation analysis than Pig and Hive that rely on MR operations.*

D. Lazy aggregation performance

We next examine performance of lazy aggregation that performs exact reductions only in the aggregation phase. Situation instances can then be inferred in the analytics phase instead. As mentioned, the basic intuition behind lazy aggregation is that a large volume of (intermediate and final) aggregated data would be a bottleneck for aggregation performance. Lazy aggregation postpones the computation of individual situation instances information to the query phase. Thus, it would be beneficial to improve performance in the aggregation phase, but there can be a trade-off and it would be expected that it takes more time for queries. In this experiment, we investigate the trade-off.

As expected, we observed that lazy aggregation leads much smaller aggregation time. For 1B-record (60 GB) aggregation, the basic aggregation (called "eager aggregation") requires over 5,600 seconds with 16 nodes. It was dramatically reduced to 881 seconds using lazy aggregation ($>$ 6x faster). The number of output records after aggregating using lazy option is 950 millions (48 GB), while it is 2.5 billions (90 GB) with the non-lazy method.

Figure 9 compares query performance with the ten queries in Table III. For the queries resulting in the small number of matches, we can see lazy aggregation works well, yielding even better response time than eager aggregation. However, it is significantly degraded for queries that need much computation. For instance, lazy aggregation is over 4x worse for Q7 than the other. This is because that lazy aggregation requires a scan of the aggregated data to construct the entire situation 1 instances. The computational overhead is bigger than Q8 and Q9 with the greater number of attributes to compose a key (i.e., three out of $< S, T, C, G_S, G_T >$).

V. Related Work

Recent advances of data-parallel computing technologies have facilitated big-data computing in science and technology communities. MapReduce [8] is a framework enabling large-scale data analytics on thousands of commodity hardware, and used for many data-intensive applications for diverse applications, such as Internet search [8], social computing [30], and scientific computing [31]. Dryad [32] also provides a programming model for data parallel applications based on a directed acyclic graph (DAG) model. These frameworks have led to the active development of an extensive set of software, such as Hadoop, Hive, and Cassandra [20], to name a few, that could provide a means of constructing big-data computing platforms.

While the MR framework is attractive, Hadoop provided a strong motivation for researchers in diverse domains to launch their applications onto the parallel computing platform with its nature of openness. For instance, Hadoop was studied in bioinformatics as a platform for processing massive biological data sets in a scalable way for peptide [10] and sequence database search [9]. In Astronomy, as another example, there were several research activities to utilize Hadoop for astronomical image processing [11] and astrophysical simulations [12]. However, there could be distinctive requirements for each domain against parallel computing platforms since each domain may assume different types of data sets and usage patterns. For example, scientific applications often require handling binary image data, while traditional MR applications assume text-formatted data sets. Thus a degree of customization for each domain would be essential for constructing a computing platform.

In the network security community, Hadoop has also been considered for several purposes, such as intrusion detection [33], [34], [35], [36], [37], log analysis [13], [14], [15], and traffic monitoring [38], [39]. The studies utilizing Hadoop for intrusion detection are mostly focused on launching existing intrusion detection algorithms such as Snort [40], and the experiments conducted were also largely limited to scalability tests. Log file analysis using Hadoop showed how to integrate alerts from different sources using different presentation formats [14] and how it could be used for analyzing DDoS log events [24]. These studies were also limited without extensive experiments. The study in [26] employed Hadoop for HTTP GET flooding attack detection based on Counter-based method with fairly large sets of input files, 500GB and 1TB, and also conducted the scalability test with ten compute nodes. The research in [38], [39] showed how Hadoop could be used for analyzing network packets and NetFlow data, and the work in [38] developed MR programs for packet analysis for IP, TCP, and HTTP traffic, and NetFlow data analysis.

Alert aggregation is one of essential components in intrusion detection and analysis, which reduces the number of alerts by grouping them sharing common features with minimal loss of information. The study in [7] presented a simplified method using a sorting of alerts based on source, target, and attack type. Individual alerts are aggregated into situations based on the three fields above. For example, if an attacker launches a repeated set of email server attacks to a single email server, the alerts for the attack have the same values for all the three fields. On the other hand, if a large number of attackers are

testing a new type of attack recently introduced in the hacker community, there could be many alerts that have the same target but different source and target information. Recently, some efforts have been made for online alert aggregation based on probabilistic methods [41] and a fuzzy-based alert aggregation for anomaly-based detection systems [42]. In this work, we chose the classical method as it is still widely used as a basic aggregation method. In addition, the classical technique could create a massive volume of intermediate data that could be an order of magnitude larger than the size of the associated raw alert data, and in this work we are interested in utilizing big-data computing technologies for security analytics.

VI. Conclusions

With the increasing demands on massive event processing, security analytics would be a killer application for big-data computing. In this work, we developed a system for security event aggregation running over a MR platform as a security analytics application. We introduced our design of the system for security event aggregation using big-data computing technologies. The presented system provides (i) periodic aggregation against newly collected data in the last interval of time, (ii) on-demand aggregation initiated by users for their own analysis, and (iii) query support to retrieve aggregated information (i.e., situation instances). We also presented extensive experimental results conducted in a Hadoop cluster for evaluating individual functional components in the system. The key observations and findings through the performance studies in this work are summarized as follows:

- Our application of security event aggregation running over the MR platform is highly scalable with greater benefits for larger input data sets with the increasing number of computing resources. We observed over 18x speedup with 16x computing resources and the speedup is better with greater input.
- Using Pig for on-demand aggregation would be feasible if we can utilize existing aggregated information. The performance is competitive with the one with the native MR code.
- Tajo significantly outperforms Pig and Hive for queries accessing situation instances with its own query engine on top of HDFS.
- Lazy aggregation can dramatically reduce aggregation time, but we observed a trade-off with a significant degradation of query response time, particularly for queries requiring a scan of data with a high degree of computational overhead for inferring.

In this study, we examined feasibility of MR technologies for big-data security analytics with the application of security event aggregation. To consider multiple data characteristics, we used a variety of synthetic data sets with different properties in our experiments. Future planned work includes working our developed system with real security log traces for fine-grained optimization. We also plan to explore publicly available big-data analytics tools in addition to the ones we investigated for query performance evaluation.

Acknowledgment

This work was supported in part by the ICT R&D program of MSIP/IITP [13-921-06-002]. The authors would like to acknowledge Sam I. Saffer and Derek Harter for their support with Lion HPC cluster for our preliminary experiments. We are also grateful to Dae-Sung Moon and Hyun-Joo Kim for their valuable suggestions for technical merits of the project.

References

[1] S. Shin and G. Gu, "Conficker and beyond: A large-scale empirical study," in *Proceedings of the 26th Annual Computer Security Applications Conference*, ser. ACSAC '10. New York, NY, USA: ACM, 2010, pp. 151–160. [Online]. Available: http://doi.acm.org/10.1145/1920261.1920285

[2] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for internet worms," in *Proceedings of the 10th ACM Conference on Computer and Communications Security*, ser. CCS '03. New York, NY, USA: ACM, 2003, pp. 190–199. [Online]. Available: http://doi.acm.org/10.1145/948109.948136

[3] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security and Privacy*, vol. 9, no. 3, pp. 49–51, May 2011. [Online]. Available: http://dx.doi.org/10.1109/MSP.2011.67

[4] C. Tankard, "Advanced persistent threats and how to monitor and deter them," *Network Security*, vol. 2011, no. 8, pp. 16–19, 2011.

[5] "Internet security threats report. symantec, http://www.symantec.com/threatreport/, last accessed: June 2013."

[6] "http://www.darpa.mil/our_work/i2o/programs/cyber_defense_ (cyber_genome_dynamic_quarantine_of_computer_based_worm_attacks, _and_scalable_ network_monitoring).aspx."

[7] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID '00. London, UK, UK: Springer-Verlag, 2001, pp. 85–103. [Online]. Available: http://dl.acm.org/citation.cfm?id=645839.670735

[8] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[9] S. Lewis, A. Csordas, S. Killcoyne, H. Hermjakob, M. Hoopmann, R. Moritz, E. Deutsch, and J. Boyle, "Hydra: a scalable proteomic search engine which utilizes the hadoop distributed computing framework," *BMC Bioinformatics*, vol. 13, no. 1, p. 324, 2012.

[10] B. Pratt, J. J. Howbert, N. I. Tasman, and E. J. Nilsson, "Mr-tandem: parallel x!tandem using hadoop mapreduce on amazon web services." *Bioinformatics*, vol. 28, no. 1, pp. 136–137, 2012.

[11] e. a. K. Wiley, "Astronomical image processing with hadoop," *Astronomical Data Analysis Software and Systems*, 2011.

[12] S. Loebman, D. Nunley, Y. Kwon, B. Howe, M. Balazinska, and J. P. Gardner, "Analyzing massive astrophysical datasets: Can pig/hadoop or a relational dbms help?" in *CLUSTER*. IEEE, 2009, pp. 1–10.

[13] R. Khattak, S. Bano, S. Hussain, and Z. Anwar, "Dofur: Ddos forensics using mapreduce." in *FIT*. IEEE Computer Society, 2011, pp. 117–120.

[14] W.-Y. C. S.-F. Yang and Y.-T. Wang, "Icas: An inter-vm ids log cloud analysis system," in *IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, 2011.

[15] Y. Lee and Y. Lee, "Detecting ddos attacks with hadoop," in *Proceedings of The ACM CoNEXT Student Workshop*, ser. CoNEXT '11 Student. New York, NY, USA: ACM, 2011, pp. 7:1–7:2.

[16] A. F. Gates, O. Natkovich, S. Chopra, P. Kamath, S. M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, "Building a high-level dataflow system on top of map-reduce: The pig experience," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1414–1425, Aug. 2009. [Online]. Available: http://dx.doi.org/10.14778/1687553.1687568

[17] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: A warehousing solution over a map-reduce framework," *Proc. VLDB Endow.*, vol. 2, no. 2, pp. 1626–1629, Aug. 2009. [Online]. Available: http://dx.doi.org/10.14778/1687553.1687609

[18] H. Choi, J. Son, H. Yang, H. Ryu, B. Lim, S. Kim, and Y. D. Chung, "Tajo: A distributed data warehouse system on large clusters," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference*, 2013.

[19] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, ser. SOSP '03, 2003, pp. 29–43.

[20] "http://hadoop.apache.org/."

[21] K.-H. Lee, Y.-J. Lee, H. Choi, Y. D. Chung, and B. Moon, "Parallel data processing with mapreduce: A survey," *SIGMOD Rec.*, vol. 40, no. 4, pp. 11–20, Jan. 2012.

[22] C. Doulkeridis and K. Nørvåg, "A survey of large-scale analytical query processing in mapreduce," *The VLDB Journal*, vol. 23, no. 3, pp. 355–380, 2014.

[23] R. J. Stewart, P. W. Trinder, and H.-W. Loidl, "Comparing high level mapreduce query languages," in *Advanced Parallel Processing Technologies*. Springer, 2011, pp. 58–72.

[24] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113

[25] "http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/impala.html."

[26] "http://cve.mitre.org/."

[27] "http://www.ietf.org/rfc/rfc4765.txt."

[28] "http://tools.ietf.org/html/rfc5424."

[29] "http://tomcat.apache.org/."

[30] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu, "Data warehousing and analytics infrastructure at facebook," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: ACM, 2010, pp. 1013–1020. [Online]. Available: http://doi.acm.org/10.1145/1807167.1807278

[31] A. Matsunaga, M. Tsugawa, and J. Fortes, "Cloudblast: Combining mapreduce and virtualization on distributed resources for bioinformatics applications," in *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, ser. ESCIENCE '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 222–229.

[32] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, Mar. 2007.

[33] N. Guilbault and R. Guha, "Experiment setup for temporal distributed intrusion detection system on amazon's elastic compute cloud." in *ISI*. IEEE, 2009, pp. 300–302.

[34] I. Aljarah and S. A. Ludwig, "Mapreduce intrusion detection system based on a particle swarm optimization clustering algorithm." in *IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 955–962.

[35] M. Masataka, B. Shintaro, S. Martin, and M. Masaharu, "Implementation of multiple classifier system on mapreduce framework for intrusion detection," in *PDPTA'13*, 2013.

[36] S. Veetil and Q. Gao, "A real-time intrusion detection system by integrating hadoop," in *Dalhousie Computer Science In-House Conference (DCSI)*, 2013.

[37] L. A. Trejo and et al., "Using cloud computing mapreduce operations to detect ddos attacks on dns servers," in *4th Iberian Grid Infrastructure Conference, IBERGRID*, 2010.

[38] Y. Lee and Y. Lee, "Toward scalable internet traffic measurement and analysis with hadoop," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 1, pp. 5–13, Jan. 2012.

[39] W. K. Y. Lee and H. Son, "An internet traffic analysis method with mapreduce," in *Network Operations and Management Symposium Workshops (NOMS Wksps)*, 2010.

[40] "http://www.snort.org/."

[41] A. Hofmann and B. Sick, "Online intrusion alert aggregation with generative data stream modeling," *IEEE Trans. Dependable Secur. Comput.*, vol. 8, no. 2, pp. 282–294, Mar. 2011. [Online]. Available: http://dx.doi.org/10.1109/TDSC.2009.36

[42] F. Maggi, M. Matteucci, and S. Zanero, "Reducing false positives in anomaly detectors through fuzzy alert aggregation," *Inf. Fusion*, vol. 10, no. 4, pp. 300–311, Oct. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.inffus.2009.01.004