HLAer: a System for Heterogeneous Log Analysis

Xia Ning, Geoff Jiang, Haifeng Chen and Kenji Yoshihira
Department of Autonomic Management
NEC Labs America
Princeton, NJ, 08850
{xning, gfj, haifeng, kenji}@nec-labs.com

Abstract

Logs are ubiquitous for system monitoring and debugging. However, there lacks a comprehensive system that is capable of performing heterogeneous log organization and analysis for various purposes with very limited domain knowledge and human surveillance. In this manuscript, a novel system for heterogeneous log analysis is proposed. The system, denoted as Heterogeneous Log Analyzer (HLAer), achieves the following goals concurrently: 1) heterogeneous log categorization and organization; 2) automatic log format recognition and 3) heterogeneous log indexing. Meanwhile, HLAer supports queries and outlier detection on heterogeneous logs. HLAer provides a framework which is purely data-oriented and thus general enough to adapt to arbitrary log formats, applications or systems. The current implementation of HLAer is scalable to Big Data.

I. INTRODUCTION

Modern systems are becoming ubiquitously more and more complicated, which sets up the insurmountable hurdles for system administrators to manually monitor system dynamics or fix any system issues in a timely manner. Thus, autonomic analysis of system logs is of a crucial demand. System/application logs represent real-time snapshots of system content and dynamics, and meanwhile, they have semantic meanings that administrators or developers can understand logically. Research and development for automatic log analysis has started attracting a considerable amount of attentions from both academia and industry.

However, in this era of Big Data, automatic log analysis faces highly non-trivial challenges that prevent conventional data mining or machine learning methods being applied immediately. Examples of such challenges include, but not limited to, the following items.

- Extremely huge amount of system logs are accumulated over periods. Processing such huge data volume itself requires significant amount of CPU times and memories.
- Various log formats, either standard or nonstandard, from different applications result in many difficulties in extending any existing methods to adapt to new log formats transparently, whereas the requirement of such adaption is tremendously common.
- Domain knowledge about the system of interest is typically lacking or not sufficient for people to fully understand the system details. Meanwhile, it is very expensive to acquire sufficient detailed domain knowledge. Thus, any log analysis methods relying on handcrafted rules from domain knowledge are not effective any more for new or newly updated systems.
- The huge data volume, heterogeneous data formats and limited domain knowledge all together makes information retrieval and further analysis from such logs overwhelming and prohibitive.

There have been some research efforts and commercial products existing that attempt on autonomic log analysis to some extent. However, all of them only tackle the problem under strong hypotheses and thus they deviate far away from the reality and fall far behind the real requirements. Existing methods are either highly customized to a specific system/application with sufficient domain knowledge available, or designed specially for a pre-defined task in mind. Therefore, they are not extensible or scalable to new systems/applications or new log types. To the best of our knowledge, there hasn’t been an autonomic log analysis system that exhibits simultaneously the characteristics of being

- highly scalable for Big Data,
- for heterogeneous log formats,
- purely data-oriented with very limited prior knowledge used,
- able to support efficient information retrieval, and
extensible to arbitrary applications/systems.

In this manuscript, we present our first efforts and results towards a fully autonomic heterogeneous log analysis system that have all the above characteristics. This system is denoted as Heterogeneous Log Analyzer (HLAer). HLAer is designed to be able to perform 1) heterogeneous log categorization and organization; 2) automatic log format recognition; 3) heterogeneous log indexing; 4) information retrieval from heterogeneous logs and 5) outlier detection from heterogeneous logs. The major difference of HLAer from all the other existing log analysis methods and tools is that HLAer is purely data-oriented and requires no specific knowledge about the underlying system/applications. Due to this, HLAer is adaptable to arbitrary systems/applications.

The rest of this manuscript is organized as follows. In Section II, a literature review is presented. Section III defines the notations. The nature of log data is discussed in Section IV. In Section V, the overview of the system is presented. In Section VI, the technical details on building HLAer are presented. In Section VII, the functionalities of HLAer are discussed. Section VIII presents the experimental results. Finally, Section IX presents the conclusions and further discussions.

II. RELATED WORK

In this section, we give a detailed literature survey on the research and industry development for log analysis.

A. Research efforts on log analysis

The first research efforts towards log data analysis fall along the line of data clustering and concurrently frequent pattern mining.

1) Clustering methods on log data: Data clustering is a powerful tool for complex data analysis since it provides the intuition and insights on the geometric structures of underlying space, from which the data are generated, and thus the inherent distribution of the data of interest. It can also work as a preprocessing step that categorizes data so as to facilitate further processing. Moreover, clustering algorithms can also work for anomaly/outlier detection purposes, given that intuitively any data points that fall far apart from the majority clusters can be considered as anomalies or outliers.

Vaarandi [16] developed Simple Logfile Clustering Tool, i.e., SLCT \(^1\), which represents one of the first log data clustering algorithms. SLCT is essentially based on frequent words, that is, the log data which have common frequent words are clustered together. The intuition behind SLCT comes from the highly skewed distribution of word counts in log data, which is very different from that of natural language and text data. This intuition plays an important role, either explicitly or implicitly, for many of the following log data clustering algorithms. On the other hand, the use of frequent words makes SLCT very rigid and only identifies the clusters in which all the logs have exactly same words. SLCT also generates a significant number of outliers which have a few different words than the clusters.

Makanju et al have a series work on log data analysis [11], [10], [9]. In [11], they proposed IPLoM \(^2\), an iterative clustering algorithms for logs. IPLoM consists of the following 4 consecutive steps: 1). logs of same lengths are first clustered together; 2). each cluster is further partitioned by tokens with best information gains; 3). second-order token pairs are used for another cluster partitioning; 4). cluster descriptions are generated based on majority voting. They demonstrated that IPLoM outperforms other log clustering algorithms including SLCT, LogHound and Teiresias (introduced later). A potential problem with IPLoM is that it can easily result in small cluster fragments that are not statistically significant. Moreover, the clustering quality is hard to control. The underlying assumption for the first step, that is, the logs of same length very possibly have same formats, can be easily violated as more and more heterogeneous logs coming into the system. Even worse, the poor clustering results from the first step can be further cascaded into the final results. Moreover, IPLoM requires a significant amount of I/O and intermediate storage, which makes it not scalable for large data in real systems.

2) Frequent pattern mining on log data: Frequent occurring patterns (i.e., combinations/series of log records, log words or system status) from log data are representative signatures of the underlying systems. Such frequent patterns represent the regular behaviors of the systems that should not be violated in normal cases, whereas any violations would indicate anomalous scenarios. Therefore, frequent patterns are often used to calibrate system behaviors as outlier detectors.

\(^1\)http://kodu.neti.ee/~risto/slct/
\(^2\)https://web.cs.dal.ca/~makanju/iplom/iplom_C.zip
Vaarandi [17] developed LogHound[^1] a tool for frequent itemset mining in log data. In LogHound, event logs are considered as transaction databases, that is, each log record represents a transaction, and thus frequent event sequence pattern mining becomes frequent itemset mining. LogHound adapts a breadth-first algorithm to find such frequent patterns, which involves heuristics to control memory usage, frequent itemset sizes, etc. A potential issue with LogHound is that its hypothesis on considering events as transactions may not always hold for many different log natures.

Capri[^2] is a new mining tool for log patterns. It generates frequent lines, terms and rules from log lines. The issue with Capri is that it generates rules without further utilizing them, and thus the use of the tool is limited.

3) Other research on log data: Research on log data also includes identifying log formats, outlier detection based on logs, and using logs to model system behaviors, etc. A very unique research direction is along the line of learning log formats. Zhu et al.[20] proposed to learn the log formats for arbitrary well-formatted logs from the perspective of Natural Language Processing (NLP) with pre-defined schemes.

Fu et al. [7] proposed to use Finite State Automaton (FSA) to model the execution behavior of system models based on logs. Logs are first clustered into log keys by considering their templates with all parameters removed based on empirical rules. Then each log is labeled by their log key type so as to construct a sequence. FSA is learned from such a sequence to capture the behaviors of the system.

Xu et al.[18] proposed a method for console log mining. Their method has four steps: 1). identify log structure from source code; 2). cluster messages and generate a feature vector for each message type; 3). use Principal Component Analysis (PCA) for outlier detection; 4) use a decision tree to visualize the results. This method does not deviate much from conventional outlier detection methods but it gives a principled way for outlier detection from log data.

Other work includes Yamanishi et al. [19] that models log sequences using a mixture of Hidden Markov Models (HMMs) so as to identify system failures, etc.

B. Log analysis tools and products

In-use/commercialized log analysis products include Splunk, Prelert, Sisyphus, LogRhythm, Capri, etc. Splunk[^3] is one of the most successful commercialized log analysis products. It performs log data collection, indexing, search and visualization, etc. It generalizes to many different log types including both standard server logs and non-standard application logs. A notable problem with Splunk is that it requires tremendous manual configuration so as to tune the system with prior knowledge such as log data formats, alert definition, event definition, etc. Thus, data indexing, event extraction and knowledge discovery cannot be conducted without user specification. Splunk distinguishes itself by its versatility in performing various tasks, but its heavy reliance on user input deviates itself from being automatic.

Prelert[^4] focuses on outlier detection of event sequences from logs. Sequence analysis techniques inspired from Bioinformatics research have been applied in Prelert so as to identify frequent event sequences, and these frequent sequences are used as a normal model for outlier detection. However, a significant drawback of Prelert is that domain knowledge about logs including log format, semantic meanings and logging system properties has to be fully available, as well as that of the management system. In particular, the concept of “event” from logs is well defined, and thus Prelert knows exactly what to look for. The well calibrated log data that Prelert performs on dramatically reduce the difficulty of the problems, and meanwhile leaves Prelert not scalable to other systems.

Sisyphus[^5] is an open-source toolkit and specifically developed for super-computer syslogs. It is able to retrieve information content from logs regarding system outliers based on information theories. In particular, Sisyphus adopts Teiresias[^6], a pattern discovery algorithm developed from Bioinformatics research, so as to identify patterns from logs. However, the methods of Sisyphus are highly customized according to the nature of super-computers with the central hypothesis that similar/homogeneous normal computers produce similar logs. However, this hypothesis may not necessarily generalize to other systems, and thus the application of Sisyphus is limited.

LogRhythm[^7] specifically its SIEM tools, is particularly designed for defense of business data based on log data analysis. It processes IT system logs and meanwhile leverages information from disparate sources with the main

[^1]: http://ristov.users.sourceforge.net/loghound/
[^2]: http://research.cs.queensu.ca/home/farhana/capri.html
[^3]: http://www.splunk.com
[^6]: http://logrhythm.com/
purpose to detect security related issues. However, its detection is based on prescribed rules that are highly specific to the application domain, i.e., business data defense. Therefore, the rule-based approaches may not generalize to other application domains or systems.

Other existing tools include flow-tools\(^9\) logsurfer\(^10\) etc. A common issue with such existing tools is their limited generalizability.

III. DEFINITIONS AND NOTATIONS

We define the following terms for the purpose of clear presentation. A log or a log record is a line of sentences or multiple lines of sentences corresponding to one time stamp that record the system/application events or status of that moment of time. Log data refer to a collection of such log records. A log record typically has multiple fields, separated by a well-defined delimiter. Each field has certain meanings defined explicitly, even though not accessible usually. Typically the fields fall into two types. The first type is for housekeeping purpose, for example, the time stamps. The second type is for various information that is specific to that certain time stamp, referred to as “log message” here. The ordering of these fields together with the definitions on each field is referred to as the log format, log layout or log template. For instance, examples of IIS logs and log formats can be found in the according specifications\(^11\).

Outliers are the data points that are distant from the majority of the others. Anomalies are the abnormal-behaving data points. In the content of this manuscript, outlier and anomaly are used interchangeable, indicating log records that are different from others based on some criteria.

IV. THE NATURE OF LOG DATA

The inherent natures of log data determine the type of methods and approaches that can be applied properly. Compared to the conventional text/natural language data including documents, articles and web pages, log data have very different natures even though they are also composed of words. Such natures are summarized as follows, which are not always discussed in existing literature.

a) Log records have very weak syntactical structures. In order to concretely record the application/system status and behaviors, log records are typically short, succinct and abstract. In addition, log records are usually in a tabular format with a pre-defined delimiter. Each field of a certain log format, as well as all the words that can appear in that field, has a pre-defined meaning that is specific to the application/system. There are no standard or basic grammatical structures/relations among each field of log records, and thus an NLP parser will fail to identify any meaningful syntactics among log fields. Even though, there still exist strong logical/semantic relations among each field of a log format and its eligible words. System administrators can well understand the semantic meanings of a log record by filling up the contextual gaps among log fields based on their domain knowledge and logical thinking.

b) Logs have very limited vocabularies but extremely skewed word count distribution. Very commonly, log records are generated from source code using a “printf” statement with all the variables replaced by parameters specific to the current system status. Such “printf” statements define the limited but also most frequent words in the logs. On the other hand, the parameters, which may not necessarily repeat significantly frequently with same values, represent the system momentum, and thus they have richer and more critical content than the most frequent words. The skewed distribution of the word counts in log data is the fundamental reason for many frequent itemset-based log mining algorithms and clustering algorithms. Meanwhile, the imbalance of the information content between frequent and infrequent words becomes a challenge to such algorithms if they are originated from conventional text mining research.

c) Log records are generated from templates. As mentioned above, log records are generated from source code and thus have clear layouts, and such layouts will not change across log records from a same source, i.e., homogeneous logs. The existence of such regulated layouts or formats of log data, even unknown as to how they look explicitly, makes log data clustering intuitively easier than that for documents and other text data. Most of the existing log data clustering algorithms implicitly or explicitly utilize such knowledge about log data.

\(^9\)http://www.splintered.net/sw/flow-tools/
\(^10\)http://www.crypt.gen.nz/logsurfer/index.html
d) Identical log records can appear redundantly. This is due to the fact that some log records are used for a housekeeping purpose, and thus in normal cases same logs can be repeated many times. Due to this, same logs can be down-sampled so as to reduce the burdens on memory and CPU usage. The redundancy of log records is a significant difference of log data from conventional text data that makes the down-sampling very possible and meaningful in the Big Data environment.

V. THE HLAer SYSTEM: AN OVERVIEW

Before digging into the technical details of the heterogeneous log analyzer HLAer, an overview of the system is presented here. HLAer is an autonomic heterogeneous log analyzer, which analyzes huge amount of heterogeneous logs from any black-box system with arbitrary applications. It provides a principled way to categorize and organize logs and an automatic approach to recognize various log formats/layouts. It is also able to support retrieval of useful information from the logs regarding the system status.

A. HLAer construction

Figure 1 demonstrates the construction flow of HLAer. Basically there are five major components in order to construct HLAer:

- Hierarchical clustering of heterogeneous logs. During this process, the heterogeneous logs are first clustered into a hierarchy based on the inherent geometric structures of all the logs. The logs are organized accordingly.
- Pattern recognition. Within the hierarchical structure, log patterns/formats are recognized and common templates are extracted as an abstract of corresponding logs.
- Field analysis. With the recognized patterns, each field of a certain pattern is analyzed by looking into, for example, the distributions of eligible words, the range of numerical values, etc. This analysis is to understand some global semantic of all logs.
- Indexing. All the logs are indexed based on the hierarchy and recognized patterns. In specific, the recognized patterns are utilized to index which logs have certain fields of interest.

Figure 3 demonstrates the internal data structure of HLAer. Basically a tree structure is retained after hierarchical clustering. Details about this tree structure will be discussed later in Section VI-D.

B. HLAer functionalities

The following examples demonstrate the functionality of HLAer.

- Log indexing: Figure 4 shows an example of HLAer indexing logs. Given a set of heterogeneous logs, HLAer can generate a hierarchical indexing structure on top of the logs so as to organize the logs in a meaningful way in terms log formats and other information.
- Log format identification: Figure 5 shows an example of HLAer identifying common log formats. Given a set of heterogeneous logs, HLAer recognizes the formats of each log type. The log format recognition/identification can tolerate non-identical words.
- Log query: Figure 6 shows an example of HLAer supporting queries. The support of queries is implemented based on the format recognition and log indexing. Given a specified query, HLAer will return all the logs that satisfy this query. For now only queries on time is implemented but queries on other fields can be implemented in a same way.
- Outlier detection: Figure 7 shows an example of HLAer detecting outliers in an offline setting. Given a set of new logs, HLAer is able to identify logs that appear anomalous.

VI. BUILDING THE HLAer SYSTEM

A. Log data tokenization

The first data preprocessing step on heterogeneous logs is to tokenize the log data such that lower level information from the words or phrases of each log record can be identified and retrieved. However, heterogeneous logs from different applications and systems have different formats, and thus different tokenizers/delimiters. Without specific knowledge or human inspection, it is unfair to use or not to use any pre-defined/popular tokenizer/delimiters for the
entire set of heterogeneous log data. Thus, a very general delimiter should be used. The use of such delimiter should avoid interfering with the possible, even unknown, delimiters from the log data, or should it introduce confusion into the original data.

In HLAer, empty space is used as such a general delimiter. All the words and special symbols except numbers will be separated by empty space. Examples of empty space as the delimiter are presented in Table I. Note that some units like IP addresses and URLs are all delimited by empty spaces. However, these special units will be recognized later as a whole.

### TABLE I: Examples of tokenization

<table>
<thead>
<tr>
<th>original log</th>
<th>tokenized log</th>
</tr>
</thead>
</table>
Input: heterogeneous logs

2012-07-09 20:32:46,864 INFO org.apache.hadoop.hdfs.util.GSet: recommended=4194304, actual=4194304
Jan 8 05:49:14 www httpd[7855]: 108.199.240.249 - - "GET /images/header/nav-pub-on.gif HTTP/1.1" 200 569
2012-07-09 20:32:46,904 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: fsOwner=hadoop_user
2012-07-09 20:32:46,905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: supergroup=supergroup
Jan 8 05:49:27 www httpd[7855]: 108.199.240.249 - - "GET /careers/internship.php HTTP/1.1" 200 11007
Jan 8 05:49:27 www httpd[7855]: 108.199.240.249 - - "GET /careers/images-careers/intern-title.gif HTTP/1.1" 200 1211
2012-07-09 20:32:46,905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: isPermissionEnabled=false
2012-07-09 20:32:46,909 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem dfs.block.invalidate.limit=100
Jan 8 05:49:27 www httpd[7855]: 108.199.240.249 - - "GET /images/home/current-bullet.gif HTTP/1.1" 200 131

Output: log indexing

Fig. 4: HLAer log data indexing

Input: heterogeneous logs

2012-07-09 20:32:46,864 INFO org.apache.hadoop.hdfs.util.GSet: recommended=4194304, actual=4194304
Jan 8 05:49:14 www httpd[7855]: 108.199.240.249 - - "GET /images/header/nav-pub-on.gif HTTP/1.1" 200 569
2012-07-09 20:32:46,904 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: fsOwner=hadoop_user
2012-07-09 20:32:46,905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: supergroup=supergroup
Jan 8 05:49:27 www httpd[7855]: 108.199.240.249 - - "GET /careers/internship.php HTTP/1.1" 200 11007
Jan 8 05:49:27 www httpd[7855]: 108.199.240.249 - - "GET /careers/images-careers/intern-title.gif HTTP/1.1" 200 1211
2012-07-09 20:32:46,905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: isPermissionEnabled=false
2012-07-09 20:32:46,909 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem dfs.block.invalidate.limit=100
Jan 8 05:49:27 www httpd[7855]: 108.199.240.249 - - "GET /images/home/current-bullet.gif HTTP/1.1" 200 131

Output: common log formats

Fig. 5: HLAer format identification
Input: a query
select all the logs recorded between 20:32:46 and 20:32:56, July 9, 2012

Output: all logs that satisfy the query

2012-07-09 20:32:46.864 INFO org.apache.hadoop.hdfs.util.GSet: recommended=4194304, actual=4194304
2012-07-09 20:32:47.905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: isPermissionEnabled=false
2012-07-09 20:32:47.909 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem dfs.block.invalidate.limit=100
2012-07-09 20:32:46.904 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: fsOwner=hadoop_user
2012-07-09 20:32:47.905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: supergroup=supergroup

Fig. 6: HLAer query support

Input: a batch of new logs
2012-07-09 20:32:46.864 INFO org.apache.hadoop.hdfs.util.GSet: recommended=4194304, actual=4194304
2012-07-09 20:32:47.905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: isPermissionEnabled=false
2012-07-09 20:32:47.909 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem dfs.block.invalidate.limit=100
2012-07-09 20:32:46.904 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: fsOwner=hadoop_user
2012-07-09 20:32:47.905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: supergroup=supergroup

Output: outliers

2012-07-09 20:32:47.905 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: isPermissionEnabled=false

Fig. 7: HLAer outlier detection

B. Log data clustering

Without domain knowledge with respect to the log formats, usage and sources, etc, a first step towards understanding and analyzing heterogeneous logs is intuitively to understand the geometric structure of the log data. As unsupervised data analysis methods, clustering algorithms serve as a way to categorize data purely based on their intrinsic properties and relations. Thus, within the framework of HLAer, a clustering algorithm is applied on the heterogeneous logs so as to present the initial depiction of the data. In specific, a hierarchical clustering algorithm is used in HLAer to generate a hierarchical structure of the heterogeneous logs. Hierarchical clustering is preferred not only because they can provide a coarse-to-fine view of the data, but also because the following data indexing and search is built up on a hierarchical tree structure for efficiency purposes. The hierarchical tree structure used in HLAer is denoted as Log Clustering Tree (LCT). The process for log data clustering is summarized in Figure 2. In Figure 2 since the calculation of pair-wise log similarity serves for the following clustering processes, the clustering method is described first as follows.

In HLAer, the hierarchical clustering algorithm Ordering Points To Identify the Clustering Structure (OPTICS) [1] is
implemented. The basic idea of OPTICS is based on DBScan [6], that is, the dense data regions with sufficient number of data points form clusters. The basic idea of DBScan algorithm is depicted in Algorithm 2 and Algorithm 3 in the Appendix. Intuitively, DBScan searches dense data regions by expanding from a certain data point towards all its neighboring data points that are sufficiently close under a pre-defined threshold. OPTICS performs DBScan but meanwhile outputs a certain ordering of the data points based on their smallest reachability distances. The reachability distance is defined as in Equation 5 in the Appendix, in which the core distance is defined in Equation 7. Intuitively, the reachability distance of a certain data point measures how close it is to its dense neighborhood, while the core distance of a certain data point measures how tight its neighborhood is centered around the point. In addition, OPTICS generates a hierarchical clustering structure from the data point ordering, in which the denser data region within a looser data region, which is still qualified as a cluster, becomes a lower-level child of the denser region (i.e., cluster). Thus, the hierarchical structure constructed from OPTICS represents the inherent data relations among all the data points.

The reason why OPTICS is selected is due to the fact that OPTICS has the following properties:

- It is scalable to large datasets and can be easily paralleled [2], which is a very favorable feature particularly for Big Data.
- It has only a few parameters (i.e., \( \epsilon, \text{minpts} \)) and these parameters can be easily obtained from historical data, given that the data are from a stable system.
- It is computationally efficient and runs significantly faster than other methods such as spectral clustering. However, the framework of HLAer is general enough to adopt various hierarchical clustering algorithms as long as they satisfy the above properties. For example, the hierarchical clustering algorithms in the software CLUTO [13] are also good options.

1) Setting Parameters: There are three critical parameters for OPTICS algorithm, i.e., the minimum number of data points \( \text{minpts} \) that can form a valid cluster, the maximum distance \( \epsilon \) between two data points that is allowed within a cluster and the distance function \( \text{dist}() \) that is used to measure the distance between two data points. Both \( \text{minpts} \) and \( \epsilon \) can be obtained based on empirical experiments. A small set of samples can be drawn first. Then a grid search on \( \text{minpts} \) and \( \epsilon \) can be performed on the samples so as to identify the optimal \( \text{minpts} \) and \( \epsilon \) that give satisfactory clustering results. The distance function \( \text{dist}() \) is discussed later in the next section.

2) The format-specific similarity/distance function: In HLAer, a format-specific similarity function \( f_{ssim} \) is proposed as follows,

\[
f_{ssim}(\log_1, \log_2) = \frac{\sum_{i=1}^{\text{min}(|\log_1|,|\log_2|)} I_f(\log_1(i), \log_2(i))}{|\log_1||\log_2|},
\]

where \( \log_1 \) and \( \log_2 \) are two log records, and the identity function \( I_f(x, y) \) is defined as follows,

\[
I_f(x, y) = \begin{cases} 
1 & \text{if } x \text{ and } y \text{ are both numerical} \\
1 & \text{if } x \text{ and } y \text{ are identical words} \\
1 & \text{if } x \text{ and } y \text{ are identical symbols} \\
0 & \text{otherwise},
\end{cases}
\]

and \( \log(i) \) is the \( i \)-th word of the log record after tokenization. Intuitively, \( f_{ssim} \) measures how two log records are similar/identical from their very beginning to the end. The reason of summing from the very beginning towards the end of the two log records in Equation 2 (e.g., \( \sum_{i=1}^{\text{min}(|\log_1|,|\log_2|)} \)) is that very popularly the common information that each log record has to have following a certain pre-defined format goes first in the log record, whereas the specific information that varies with respect to each log record comes later. For example, the two log records

“Jan 8 05:49:14 www http[7855]: 108.199.240.249 - - GET /images/header/nav-pub-on.gif HTTP/1.1 200 569” and


have a same format and will have a large similarity value by \( f_{ssim} \), whereas the two log records

“Jan 8 05:49:14 www http[7855]: 108.199.240.249 - - GET /images/header/nav-pub-on.gif HTTP/1.1 200 569” and


\[\text{http://en.wikipedia.org/wiki/DBSCAN}\]
\[\text{http://glaros.dtc.umn.edu/gkhome/views/cluto}\]
have different formats and will have a smaller similarity value by \( f_{ssim} \). Thus, \( f_{ssim} \) is strong in capturing pattern similarity. In spirit, \( f_{ssim} \) is similar to edit distance, but it is highly adapted to log data natures.

The identity function \( I_f(x,y) \) in Equation 2 is defined in a way so as to leverage the fact that in a large volume of log records, the vocabulary is very limited compared to natural languages and in normal cases the words are highly repetitive, and thus binary values are sufficient to capture the word relations. Intuitively, however, the identity function \( I_f(x,y) \) can be replaced by a similarity measurement whose values fall in \([0, 1]\). Nevertheless, as the first step without any knowledge, binary values and identity function serve well as a good seed.

Given \( f_{ssim} \in [0, 1] \), a format-specific distance function \( f_{dis} \) is defined as

\[
    f_{dis}(\log_1, \log_2) = 1 - f_{ssim}(\log_1, \log_2), \tag{3}
\]

which is used in DBScan.

Adaptive format-specific similarity/distance function: note that \( I_f \) has a rigid format, that is, the similarity between two words (except numerical values and symbols) in two logs is only determined by whether they are identical or not, thus it is not flexible enough for words which are not identical but “similar” in some sense. For example, in log messages, the words “errors” and “mistakes” should have higher similarity based on their meanings rather than 0 calculated from \( f_{ssim} \). Thus, \( I_f \) and \( f_{ssim} \) are not strong enough to capture semantics. However, this drawback of \( I_f \) and \( f_{ssim} \) can be remedied via active learning as in [12]. The basic idea is to leverage the co-occurrence of word pairs over a large number of log records, and then estimate the word similarity using their co-occurrence frequency with a third common word.

Sequential ordering vs alignment: note that \( f_{ssim} \) calculates the similarity between two log records from the beginning to the end, that is, in a sequential order. This may not be the best to fully capture the structural and content similarity of the two logs if variables/parameters occur very early in the log records, which may significantly alter the alignment of template words and thus result in low similarity values. An alternative is to first align the logs and then calculate the similarity from the alignment. The idea of alignment will be addressed later, but as in the every first steps for data processing, a coarse-grained similarity/distance function is sufficient for the basic clustering as indicated in the experiments. However, systematically the similarity function can be improved iteratively via active learning so as to be better customized to the problem of interest.

C. Pattern recognition and Field Analysis

After clustering the log data, an overall structure of all the heterogeneous logs is generated. However, patterns within each cluster is still very necessary in order to semantically understand the logs in-depth. The conception of “pattern” is defined here as the layout/format of the log records. For example, in the log record

“Jan 8 05:49:14 www httpd[7855]: 108.199.240.249 - - “GET /images/header/nav-pub-on.gif HTTP/1.1” 200 569”,

its pattern is

date time www httpd[number]: IP – “GET URL” number number.

The purpose of pattern recognition is to identify which field of a log record after tokenization is about what specific information.

Since within each cluster, the log records have very similar formats due to the definition in Equation 3, pattern recognition is performed within a cluster using the idea of sequence alignment from Bioinformatics research. In specific, all the log records within a cluster are aligned together so as to identify the motifs (i.e, the most conserved and frequent common portion), and thus the common patterns, from the logs. Smith-Waterman algorithm [13] is used to perform pairwise alignment and then Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [14] strategy is used to perform multiple alignment. Note that the pattern recognition is done first in the leaf nodes, where the cluster purity is high and thus the alignment is clear cut, and then the pattern information is backpropagated back from the leaves to the root node.

The strategy of first conducting clustering and then performing multiple sequence alignment within clusters is also for computational efficiency purpose. For a dataset of \( n \) log records, the multiple sequence alignment on all the \( n \) log records requires the calculation of \( O(n^2) \) pairs of alignment as described later in Section VI-C1. However, if the \( n \) logs are clustered into \( m \) clusters equally, for example, it only requires \( O\left(\frac{n^2}{m}\right) \) pairs, which can dramatically decrease the computation load. In addition, the alignment task for each cluster is fully decoupled and thus it is trivial to parallel all the alignment tasks.

1) Sequence Alignment for Pattern Recognition: The building block of pattern recognition via multiple sequence alignment is pairwise sequence alignment. The basic idea of pairwise sequence alignment is to find an optimal arrangement of two sequences side by side by which the alignment score is maximized. In order to make such alignment, four different operations are available during pairwise alignment: match, mismatch, insert (gaps) and delete. Each of the operations contributes its own score to the alignment, and the alignment score is calculated as the sum of all such scores. Pairwise sequence alignment is typically formulated as an optimization problem in which the objective is to maximize the alignment score, and dynamic programming [3] is a popular method to solve the optimization problem. The Smith-Waterman method is the most popular method that utilizes dynamic programming for pairwise sequence alignment, which requires a scoring function that measures the total score of the alignment.
In HLAer, the scoring function for Smith-Waterman method is defined as follows.

\[
\text{score}(S_1(i), S_2(j)) = \max \left\{ \text{score}(S_1(i-1), S_2(j-1)) + \text{match}\_\text{score}(S_1(i), S_2(j)) \right. \\
\left. \text{score}(S_1(i), S_2(j-1)) + \text{insert}\_\text{score} \right. \\
\left. \text{score}(S_1(i-1), S_2(j)) + \text{delete}\_\text{score} \right. 
\]

(4)

where the function \text{match}\_\text{score} is defined as follows,

\[
\text{match}\_\text{score}(S_1(i), S_2(j)) = \begin{cases} 
10 & \text{if } S_1(i) = S_2(j), \\
1 & \text{otherwise}
\end{cases}
\]

(5)

and the insertion score \text{insert}\_\text{score} and deletion score \text{delete}\_\text{score} are both set to 0. The reason for setting the scores like this is to greatly encourage aligning identical words and reducing gaps. An example of aligning the following two tokenized logs is presented in Table II.

**TABLE II: Examples of pair-wise log alignment**

<table>
<thead>
<tr>
<th>logs</th>
<th>alignment</th>
</tr>
</thead>
</table>

"_" here represents an inserted gap.

Given all pairwise alignment, the UPGMA multiple sequence alignment is done as in Algorithm I. The key idea of UPGMA is to sequentially merge the pairwise alignment that has the best score and replace the merged sequences with a consensus sequence for the next merge iteration.

**Algorithm 1: UPGMA for Multiple Sequence Alignment**

```plaintext
input : n sequences \{S_1, S_2, \ldots, S_n\}
output: Multiple sequence alignment order Align_order

// Pairwise sequence alignment
score_matrix = -inf \times \text{ones}(2n, 2n);

m \leftarrow n;

for i \leftarrow 1 \ to \ n do
    for j \leftarrow 1 \ to \ n do
        score_matrix(i, j) = \text{pairwise_alignment_score}(S_i, S_j);
        score_matrix(i, i) = -inf;

Align_size = \text{zeros}(2n, 1);
Align_size(1 : n) = 1;
Align_order = \text{zeros}(2n, 1);

// UPGMA multiple sequence alignment
while m < 2n - 1 do

    // find the alignment profiles to align
    (max_score, i, j) = \max(\max(score_matrix));
    \text{push}(Align_order, (i, j));

    // align two alignment profiles
    \text{align}\_\text{alignment}\_\text{profile}\_i\ and \text{alignment}\_\text{profile}\_j;

    // update alignment scores
    score_matrix(:, :) = -inf;
    score_matrix(:, i) = -inf;
    score_matrix(:, j) = -inf;
    score_matrix(:, :) = -inf;
    m \leftarrow m + 1;
    score_matrix(m, m) = -inf;
    Align_size(m, m) = Align_size(i) + Align_size(j);

    foreach k \neq m do
        score_matrix(k, m) = (score_matrix(k, i) \times Align_size(i) + score_matrix(k, j) \times Align_size(j))/(Align_size(i) + Align_size(j));

return Align_order
```
Adaptive scoring scheme: similarly as in the situation for \( fssim \), the \( \text{match\_score} \), \( \text{insert\_score} \) and \( \text{delete\_score} \) are rigid. However, the scoring scheme can also be adaptive using the idea from active learning. For example, when \( S_1(i) \neq S_2(j) \), the \( \text{match\_score} \) between \( S_1(i) \) and \( S_2(j) \) can be replaced by their similarity score that is adaptively learned for \( fssim \).

Computational efficiency and scalability of sequence alignment: the computational complexity for pairwise sequence alignment for two sequences of length \( m \) and \( n \), respectively, is \( O(mn) \), which is expensive. The situation is even worse for multiple sequence alignment. However, it can be remedied by the following two options: 1) parallel multiple sequence alignment since parallel algorithms for multiple sequence alignment has been well developed from Biinformatics research [5]; 2) down-sample the sequences and this can be done starting from the calculation of \( fssim \), that is, if \( fssim \) of two sequences (i.e., logs) is close to 1 enough, then only one of them can be sampled and used for further alignment and pattern recognition. The second option is quite practical due to the nature of log data.

2) Pattern profiling: After multiple log alignment, from each conserved motifs (i.e., well-aligned segments of multiple logs), two different types of patterns are recognized. The first is application-irrelevant fields such as time stamps, IP addresses, URLs, internet application protocols, etc. Such fields are hard-coded within the source code and will appear regularly within a common log format. The second type is application-specific fields that contain vocabularies only specific to the applications and its runtime.

For the first type of patterns, a tag is dedicated to each field. For example, for time stamp “2013-07-08 11:18:10”, tag “time_yyyymmdd hhmmss” is used to represent this field. This type of patterns is standard, even though each of them may have a finite set of different formats. For example, for time stamps, the format can be like “2013-07-08 11:18:10” or “11:18:10 07/08/2013”. However, since the formats are limited, each of them can be properly handled. Individual parsers for these formats are implemented. In addition, the information is saved in tree nodes in a histogram so as to keep track of the range.

For the second type of patterns, the pattern profile is retained. For all the words that fall in the field, their distribution is calculated and stored in the hierarchical tree structure. Since it is assumed that the log vocabulary is limited and the logs are well formatted, the distribution of all the words in one field is in heavy tail. This nature makes the storage requirement not a particularly critical issue. Also even when in special cases there are many words occurring in a same field, the distribution list can be truncated so as to only keep the top most frequent words in the list.

3) Pattern of patterns: As mentioned in Section VI-C, the pattern recognition is done in the leaf node, where relatively the clustering is pure and the patterns are clear. After the pattern recognition is done, the patterns are backpropagated to upper-level nodes from the leaf nodes. The backpropagation is done by aligning the profiles of the patterns based on the idea of profile alignment from Bioinformatics [4], for example, profile Hidden Markov Models can be used to align profiles. However, for simplicity, the most possible sequence from each pattern profile, that is, the sequence of words that are most frequent at each field, is used as a representative of the profile and then sequence alignment is conducted. In each intermediate node, profiles from lower-level nodes are first aligned and then a consensus profile is generated as the profile for this node. In addition, each node has pointers directed to the profiles of its lower-level children nodes. This pattern backpropagation is done in a bottom-up fashion from the leaf nodes up to the root node.

D. Indexing

After the hierarchical clustering of all the logs, the logs are sorted according to the DBScan algorithm and also a tree structure LCT is built. Figure 8 demonstrates a LCT node. In specific, each node in LCT has the following information.

- Log indices: indices of the logs that are clustered as in the corresponding clusters.
- Pattern profile: including the field tag (e.g., timestamp, URL, IP, numerical values etc), word distribution (e.g., as in Figure 8 “GET” happens 53% out of all the times in field 5), and a representative log of this cluster.
- Pointers to the children nodes: the log indices of the children nodes are the subsets of the log indices of the parent node.

Figure 3 demonstrates how the indexing is implemented. Basically all the logs are first sorted based on DBScan algorithm. Then in the hierarchical tree structure LCT, each node contains a set of logs after the re-ordering. The LCT structure organizes all the logs such that similar logs are within a same sub-tree. The more similar the logs are, the lower level sub-tree they fall within.

VII. IMPLEMENTING THE HLAER FUNCTIONALITIES

A. Online pattern recognition

After the LCT is constructed, given a new log record, its format can be recognized using LCT in a top-down fashion. In specific, once a log record is routed to an LCT node, the log record is aligned with all the pattern profiles in this node. If the alignment score is higher than a pre-specified score, the log is considered as to have the corresponding pattern. The log-profile alignment is done in a similar fashion as in pairwise sequence alignment (via dynamic programming) except in the scoring function, the profile distribution is also considered, that is, the score is calculated as the sum of all the words happening in the corresponding field weighted by their frequencies.

For a given log record, if a cluster is found which has the pattern most similar to the pattern of this log record, this log is considered as normal pattern. Otherwise, this log is considered as abnormal/rare.
Note that the online pattern recognition is done here using alignment, which is quadratic with respect to the length of the input log, and alignment has to be done multiple times as the incoming log is routed down to a certain node. Due to these reasons, this process can be expensive. However, this drawback can be overcome by implementing a parser that recognizes all the profile patterns at one round. On the other hand, without the significant efforts on parser development, alignment is still a good option which is able to implement the recognition purely based on data.

B. Search

The search is also done in a top-down fashion in the LCT. First the pattern profiles are checked to identify if the corresponding patterns have the field of question. If they do, then the search is routed down to the corresponding children. The organization of all the log records grants the opportunity that the search process can quickly filter out a lot of irrelevant log records. Query on timestamps of various formats has been implemented. Particularly, queries such as all logs before/after a certain timestamp, or between two timestamps can be executed.

Note that the top-down search from the LCT can be slow when the tree hierarchy is deep or there are many patterns. A walk-around is to build an indexing on the popular query items, e.g., time, IP, URL, etc, and thus the query can be directed immediately to the right patterns/logs. Figure 9 demonstrates how the query is done in a top-down fashion. The basic idea is at each sub-tree check whether the node has patterns which have the query items.

Currently, queries on time are implemented in HLAer and the supported types of queries are listed as in Table III which represents the most complicated queries on single fields. For the fields of numerical values, the four operations as in Table III, that is, =, >, < and in, can be easily implemented in a similar way as for times. For the fields of words, IP addresses and URLs, etc, the only operation = is also easy to deal with.

<table>
<thead>
<tr>
<th>query</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>=.2012-5-25 5:49:6</td>
<td>to find all the logs which are recorded at exactly 2012-5-25 5:49:6</td>
</tr>
<tr>
<td>&gt;.2012-7-9 20:50:5</td>
<td>to find all the logs which are recorded after 2012-7-9 20:50:5</td>
</tr>
<tr>
<td>&lt;.2012-7-9 21:29:53</td>
<td>to find all the logs which are recorded before 2012-7-9 21:29:53</td>
</tr>
<tr>
<td>in,.2012-7-9 20:50:5,.2012-7-9 21:29:53</td>
<td>to find all the logs which are recorded between 2012-7-9 20:50:5 and 2012-7-9 21:29:53</td>
</tr>
</tbody>
</table>

C. Outlier detection

The outlier detection from HLAer can be done in three different ways. The first way is to do outlier detection during HLAer construction, that is, during clustering of the historical log data. This relies on the clustering algorithms that have the mechanism to figure out the outliers. Outlier detection from historical data gives the idea of how the system normally runs and what typical problems the system may have, and thus gives the intuition on the problems to fix for the future. The second way is to do outlier detection offline in a batch setting after HLAer is constructed. This is done by calculating the $f_{dis}$ distances of the testing data with respect to the training data (i.e., historical data for the HLAer construction). If the testing log record is far from any other
training data up to a certain threshold, then the log is considered as an outlier. This method can be used to detect most recent outliers compared to the normal history. The third way is to do outlier detection online, that is, as a new log record comes in, the outlier detection is performed concurrently. For now, HLAer performs online outlier detection by comparing the format of the new coming log record with all the templates in HLAer. Essentially, this is the idea of comparing the new coming log record with the most frequent log formats in history, and this can be done by aligning the new log with pattern profiles. If this log cannot be aligned well with any pattern profiles, then this log is considered as outlier. Meanwhile, the content of the logs should also be considered for outlier detection. This alternative is left for further exploration.

**VIII. EXPERIMENTAL RESULTS**

**A. Datasets**

We use 12,000 log records of 13 different types to evaluate our methods and framework. Table IV summarizes the datasets. Failure and HPC are both downloaded from Los Alamos National Laboratory, High Performance Computing Research Projects: Operational Data to Support and Enable Computer Science Research[14] all systems failure/interrupt data 1996-2005[15] which
TABLE IV: The Datasets Used in Evaluation

<table>
<thead>
<tr>
<th>dataset</th>
<th>samples</th>
<th>examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Failure</td>
<td>1,000</td>
<td>2,cluster,49,6152,80,0,5-Apr,5-Jun,2005,part,80,1,1,0,graphics,fe/9/6/2005 14:50,9/6/2005 15:08,18,Other Software,No</td>
</tr>
<tr>
<td>HPC</td>
<td>1,000</td>
<td>4033293.resourcemgmt daemon node-31,server,subsys,1145553621,1,failed to configure resourcemgmt subsystem error = 10</td>
</tr>
<tr>
<td>ISS</td>
<td>1,000</td>
<td>2012-05-25 05:43:50 192.168.10.180 GET /proxy.txt - 80 - 192.168.10.196 - 200 0 0 229</td>
</tr>
<tr>
<td>Apache</td>
<td>1,000</td>
<td>2,cluster,49,6152,80,0,5-Apr,5-Jun,2005,part,80,1,1,0,graphics,fe/9/6/2005 14:50,9/6/2005 15:08,18,Other Software,No</td>
</tr>
<tr>
<td>Hadoopjob</td>
<td>1,000</td>
<td>Decommissioning JobTracker: Decommisioning 0 nodes</td>
</tr>
<tr>
<td>Hadoopname</td>
<td>1,000</td>
<td>2012-07-09 20:33:36:08 INFO org.apache.hadoop.mapred.JobTracker: Decommissioning 0 nodes</td>
</tr>
<tr>
<td>Syslog</td>
<td>500</td>
<td>May 16 07:23:10 ns1 ftpd[12015]: wu-ftpd - TLS settings: control allow, client_cert allow, data allow</td>
</tr>
<tr>
<td>BGL</td>
<td>1,000</td>
<td>2012-07-09 20:32:47:749 INFO org.apache.hadoop.ipc.Server: starting</td>
</tr>
<tr>
<td>Liberty</td>
<td>1,000</td>
<td>2012-07-09 20:32:47:749 INFO org.apache.hadoop.ipc.Server: starting</td>
</tr>
<tr>
<td>Redstorm</td>
<td>1,000</td>
<td>2012-07-09 20:32:47:749 INFO org.apache.hadoop.ipc.Server: starting</td>
</tr>
<tr>
<td>Spirit</td>
<td>1,000</td>
<td>2012-07-09 20:32:47:749 INFO org.apache.hadoop.ipc.Server: starting</td>
</tr>
<tr>
<td>Thunderbird</td>
<td>1,000</td>
<td>2012-07-09 20:32:47:749 INFO org.apache.hadoop.ipc.Server: starting</td>
</tr>
</tbody>
</table>

records the system failure/interrupt information of high-performance computing clusters, and system event information. IIS contains logs from Microsoft Internet Information Services 7.5 from our internal web server. Apache contains logs from our internal Apache web server. Hadoopjob, Hadoopname and Hadoopdata are logs from an internal hadoop system, particularly from job tracker, hadoop name node and hadoop data node. Syslog is a set of system logs from an internal server. BGL, Liberty, Redstorm, Spirit and Thunderbird are downloaded from Sandia National Laboratories, Supercomputer Event Logs for automatically detecting and diagnosing failures in large-scale computer systems. 1,000 samples from each dataset are randomly drawn (except 500 from Hadoopdata and Syslog due to limited dataset size). Examples of each dataset are presented in Table IV. Since the three Hadoop datasets are from a same system, they have very similar format. The last five datasets are also similar to each other in terms of their unique formats (e.g., logs start with “.” and a big number). The diversity of the different log formats and natures (e.g., Failure is particularly for logging system failures and interrupts whereas Apache is for server events) provides a good testing case for the generality of LCT.

B. Similarity/distance measurements

Figure 10 and Figure 11 show the pairwise similarities of the datasets calculated from fssim as defined in Equation 1 and cosine similarity from bag-of-word representation of log records, respectively. Bag-of-word representation is a way to present a document as a vector of words and their frequencies in the document, where there is no ordering among the words. It is widely used in text mining and other problems where an entire entity can be represented as an orderless ensemble of small pieces. For demonstration purposes, all the log records are ordered in a way such that the logs from a same dataset are arranged together. Figure 10 has clear block structures along the diagonal. This demonstrates that fssim has significant power to distinguish logs from different formats. However, as in Figure 11 the conventional cosine similarity on bag-of-word features of the logs cannot capture the structure of log data. This is due to a special characteristics of log data, that is, each log record is much shorter than a document, whereas the cardinality of the word dictionary is relatively larger than the log record length, and thus all the cosine similarity tends to be very close to 0. Figure 12 demonstrates the reordered pairwise distances after clustering as well as the hierarchical structures after clustering. In Figure 12, ε = 0.5 and minpts = 10. Accordingly, the reachability distance is presented in Figure 13. Figure 12 clearly shows the blocks with small distances along the diagonal. In specific, there are three big blocks along the diagonal. The first block occupies the region with x-axis [0:2000] and y-axis [0:2000] and includes the majority of the datasets Failure and HPC. The second block occupies the region with x-axis [2000:8500] and y-axis [2000:8500] and includes the majority of the datasets BGL, Liberty, Redstorm, Spirit and Thunderbird. The third block occupies the region with x-axis [8500:12000] and y-axis [8500:12000] and includes the majority of the datasets Hadoopjob, Hadoopname and Hadoopdata. IIS, Apache and Syslog form clusters that fall within the second and third blocks. IIS forms two clusters around x-axis 8500 and [11000:12000], Apache forms two clusters around x-axis 6000 and [7300:7800], Syslog forms one cluster around x-axis [4380:4880]. In addition, hierarchical structures have been clearly demonstrated in Figure 12 as well. For example, there are two small clusters along the diagonal in the region of x-axis [1:500] and [500:1000], respectively, both of which belong to a larger cluster occupying x-axis [0:1000] region, and this cluster is part of even higher-level cluster in region x-axis [0:2000]. The visualization of the pairwise distances from fsdis shows that the similarity function fssim and the distance function fsdis can easily capture both the formats and content of heterogeneous logs.

16LA-UR-06-0803-MX20_NODES_0_TO_255_EVENTS.csv
C. Clustering performance

For the optimal parametrization $\epsilon = 0.2$ and $\text{minpts} = 10$, out of 12000 log records, 975 records (8.1%) have been clustered as outliers, which are either far away from other data points, or only have a small amount. Table VIII shows some examples of the detected outliers. Although no ground truth is available to determine whether these records should be considered as outliers in their specific system environment, these log records can still be interpreted as to represent either the system status change (e.g., the first few logs in Table VIII) or system failures or errors (e.g., the last few logs in Table VIII). Thus, detection of such informative log records is still meaningful. In addition, for the datasets used in experiments, OPTICS only takes 1.379 seconds to generate clusters. Totally 128 clusters have been generated, out of which 118 clusters only contain logs from a same dataset. Clusters are plotted in Figure 13 and Figure 14 (the green lines; -1 as cluster ID corresponds to noise/outliers). In specific, Figure 14 shows the clustering results for the combination of Failure, HPC and BGL, where, even after the reordering, the first 1000 log records are all of Failure and the next 773 log records are from HPC, and last 219 log records are from BGL. 10 clusters are generated in total, 2 for Failure and 7 for HPC, and 1 (cluster 10) for BGL, with 55 outliers in total, 9 for Failure, 38 for HPC and 8 for BGL, respectively. There is a clear cut between the three datasets. The 9 outliers for Failure are listed in Table VII which are the records that have missing information in the first few fields where comma is the delimiter. Examples of outliers for HPC are listed in Table VII where comma is also the delimiter. The outliers correspond to the system errors that
Comparison with SLCT and IPLoM: SLCT is a popular log clustering tool as discussed in Section II. We applied SLCT on our datasets for comparison purpose. SLCT requires the delimiter to be specified as input and only works on homogeneous log types. Even though, we applied SLCT on the combination of Failure and HPC with comma as the specified delimiter and support 10. SLCT generates 46 clusters and 987 outliers. Interestingly, almost all the logs from HPC are classified as outliers. This may because although HPC has more intrinsic formats and less frequent words. In addition, Table VII shows some examples of clusters from SLCT. Clearly, the first three clusters should be condensed into one cluster, since the only difference among the three is the numbers (actually the parameters) after “ambien =”. Similarly, the last five clusters should be condensed into one as well. In addition, in order to generate template, the clusters have some words, e.g., “empera”, “curren”, etc, which themselves do not have accurate
TABLE VI: Outlier examples for HPC by HLAer

<table>
<thead>
<tr>
<th>Outlier examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2567867, Interconnect-1T00, switch_module, bcast-error, 1074100627, 1, Link error</td>
</tr>
<tr>
<td>29900, Interconnect-1T00, switch_module, bcast-error, 1075552735, 1, Link error</td>
</tr>
<tr>
<td>69834, Interconnect-1T00, switch_module, bcast-error, 1077038819, 1, Link error</td>
</tr>
<tr>
<td>101758, Interconnect-1T00, switch_module, bcast-error, 1077734600, 1, Link ok</td>
</tr>
<tr>
<td>225719, Interconnect-1T00, switch_module, bcast-error, 1078593128, 1, Link error</td>
</tr>
<tr>
<td>226540, Interconnect-1T00, switch_module, bcast-error, 1078708751, 1, Link error</td>
</tr>
<tr>
<td>287239, Interconnect-1T00, switch_module, bcast-error, 1132212255, 1, Link error</td>
</tr>
<tr>
<td>414921, Interconnect-1T00, switch_module, bcast-error, 1142873824, 1, Link error on broadcast tree</td>
</tr>
</tbody>
</table>

TABLE VII: Clusters for Failure from SLCT

<table>
<thead>
<tr>
<th>Cluster examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>* node-* node empera ure 1* 1 ambien =30</td>
</tr>
<tr>
<td>Support: 17</td>
</tr>
<tr>
<td>* node-* node empera ure 1* 1 ambien =29</td>
</tr>
<tr>
<td>Support: 18</td>
</tr>
<tr>
<td>* node-* node empera ure 1* 1 ambien =32</td>
</tr>
<tr>
<td>Support: 11</td>
</tr>
<tr>
<td>20 clus er 512 2048 4 * 1-Oc 1-Dec curren par 16 2 2 compu e * * * * * CPU No</td>
</tr>
<tr>
<td>Support: 37</td>
</tr>
<tr>
<td>19 clus er 1024 4096 4 * 2-Aug 2-Oc curren par 16 2 2 compu e * * * * * CPU No</td>
</tr>
<tr>
<td>Support: 20</td>
</tr>
<tr>
<td>19 clus er 1024 4096 4 1* 2-Aug 2-Oc curren par 8 2 2 compu e * * * * * CPU No</td>
</tr>
<tr>
<td>Support: 49</td>
</tr>
<tr>
<td>18 clus er 1024 4096 4 * 2-Mar 2-May curren par 16 2 2 compu e * * * * * CPU No</td>
</tr>
<tr>
<td>Support: 14</td>
</tr>
<tr>
<td>18 clus er 1024 4096 4 * 2-Mar 2-May curren par 8 2 2 compu e * * * * * CPU No</td>
</tr>
<tr>
<td>Support: 38</td>
</tr>
</tbody>
</table>

semantic meanings. In contrary, HLAer clusters such logs into one cluster due to the flexibility of the similarity measurement to encapsulate the different words in the template, whereas SLCT is very rigid relying only on frequent words. IPLoM has exactly the same problem as SLCT, since it only considered frequent identical words. For example, IPLoM generates two clusters with the templates “169563node25nodesstatus1140990821running” and “231564node70nodesstatus11177219541running” for the combined dataset Failure and HPC, but obviously these two should be condensed into one. In addition, IPLoM generates a huge amount of intermediate files during the process and puts a very high demand on I/O. Comparison with CLUTO: CLUTO is also applied for clustering purpose. Recursive bisection algorithm is used in CLUTO with 64 clusters with $fssim$ as a similarity measure. The results are presented in Figure 13 and Figure 14 (the blue plots). 64 clusters are chosen for demonstration purposes. In the figures, the cluster ID has an offset to avoid overlap with other plots (the green plots). Both Figure 13 and Figure 14 demonstrate that in general, CLUTO also generates clusters that correspond to the datasets. The difference is CLUTO will not identify outliers but generate small clusters for those outliers (the blue dots between x-axis [1600:1800] in Figure 14). The agreement between OPTICS and CLUTO, both of which perform the clustering based on the inherent structure (i.e., pairwise similarity) of the data, indicates that $fssim$ is effective in capturing the inherent structure of log data.

D. Outlier detection

The entire dataset of 12000 log records is randomly split into 5 folds equally, 4 as the training data and the last one as the testing data. It takes 0.053 seconds including I/O for the offline detection on a single CPU, 2.27e-5 seconds on average per each testing log record, given the pairwise similarity between the training data and testing data is available. Otherwise, it takes 35.10 seconds to calculate the pairwise similarity between 2341 testing data and 9659 training data, on average 0.015 seconds per each testing log. Examples of detected outliers in the offline setting are presented in Table VIII. These outliers correspond to either system status change or system errors/exceptions. In total, 161 outliers are detected. Please note that we use Table VIII to show the outliers detected during both clustering process as in Section VIII-C and the offline outlier detection process. In Table VIII
the outliers are the interaction of those from the two processes. For the clustering, all the 12000 logs records are used while in the offline outlier detection process, 4 out of 5 folds are used, and thus the outliers in the left \textbackslash fold are also the outliers in the entire set.

TABLE VIII: Detected outlier examples: offline setting

<table>
<thead>
<tr>
<th>Outlier examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>1136304262 2006.01.03 an 706 jan 3 08 : 04 : 22 an 706 / an 706 portmap : portmap startup succeeded</td>
</tr>
<tr>
<td>1150819446 2006.06.20 tsqe 2 jun 20 09 : 04 : 06 tsqe 2 / tsqe 2 sshd [ 24580 ] : local disconnected : connection closed</td>
</tr>
<tr>
<td>2581058 , interconnect _ 0 n 02 , switch _ module , error , 1074217265 , 1 , linkerror event interval expired</td>
</tr>
<tr>
<td>461389 , node - 54 , unix . hw , net . niff . up , 1145552319 , 1 , niff : node node - 54 has detected an available network connection on network 5 . 5 . 224 . 0 via interface alt 0</td>
</tr>
<tr>
<td>1131747220 2005.11.11 bn 689 nov 11 14 : 13 : 40 bn 689 / bn 689 instsvcdrv : dcdipm device driver loaded</td>
</tr>
<tr>
<td>1152133261 2006.07.05 cn 397 jul 5 14 : 01 : 01 cn 397 / cn 397 crond [ 22451 ] : ( root ) cmd ( run - parts / etc / cron . hourly )</td>
</tr>
<tr>
<td>1131057247 2005.11.03 r 63 - m 0 - nd - c : j 09 - u 01 2005 - 11 - 03 - 14 . 34 . 07 . 738996 r 63 - m 0 - nd - c : j 09 - u 01 ras kernel info 640764 floating point alignment exceptions</td>
</tr>
<tr>
<td>1120260785 2005.07.01 r 20 - m 0 - n 9 - c : j 03 - u 11 2005 - 07 - 01 - 16 . 33 . 05 . 490139 r 20 - m 0 - n 9 - c : j 03 - u 11 ras kernel info 1146800 double - hummer alignment exceptions</td>
</tr>
</tbody>
</table>

E. Patterns

Pattern examples are presented in Table IX. Compared with the patterns that are generated from SLCT on each individual log type, HLAer finds the patterns that are more general.

TABLE IX: Pattern examples

<table>
<thead>
<tr>
<th>log example</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012 - 5 - 25 15 : 2 : 46 get / proxy . txt - 80 - - 200 0 0 0</td>
<td>(yyyymmdd) (time) get W proxy \ W txt \ W (numerical) \ W \ W (numerical) (numerical) (numerical) (numerical) (numerical) (numerical)</td>
</tr>
<tr>
<td>2012 - 7 - 9 21 : 21 : 6 , 975 info org . apache . hadoop . hdfs . server . datanode . datanode : receiving block blk _ - 2807933493831396937 _ 15274 src : / 138 . 15 . 164 . 121 : 57053 dest : / 138 . 15 . 164 . 127 : 41010</td>
<td>(yyyyymmdd) (time) \ W (numerical) info org \ W apache \ W hadoop \ W hdfs \ W server \ W datanode \ W datanode \ W receiving block blk _ W (numerical) _ (numerical) src W (IPv4) _ W (numerical) dest W (IPv4) \ W (numerical)</td>
</tr>
<tr>
<td>2012 - 7 - 9 21 : 45 : 38 , 845 info org . apache . hadoop . hdfs . server . datanode . datanode : receiving block blk _ - 109842255273914538 _ 15274 src : / 138 . 15 . 164 . 124 : 40852 dest : / 138 . 15 . 164 . 127 : 41010</td>
<td>(strnddyyyy) (time) \ W (numerical) info org \ W apache \ W hadoop \ W hdfs \ W server \ W datanode \ W datanode \ W receiving block blk _ W (numerical) _ (numerical) src W (IPv4) \ W (numerical) dest W (IPv4) \ W (numerical)</td>
</tr>
</tbody>
</table>

F. Search results

Examples of search results are presented in Table X. On average, it takes 6.44e-5 seconds to return a query results.
TABLE X: Query examples

<table>
<thead>
<tr>
<th>query</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>=2012-5-25 5:49:6</td>
<td>1 result 2012 - 05 - 25 05 : 49 : 06 get / virtualserver / vswebapp . exe view = 17 &amp; vm = k % 2 dkato % 2 ddc 3 80 k - kato - dc 2 \0000010816740 mozilla / 4 . 0 ···</td>
</tr>
</tbody>
</table>

IX. CONCLUSION & DISCUSSION

A. Conclusion

In this manuscript, we proposed HLAer, a novel system for heterogeneous log analysis. HLAer achieves the following goals concurrently: 1) heterogeneous log categorization and organization; 2) automatic log format recognition and 3) heterogeneous log indexing. Meanwhile, HLAer supports queries and outlier detection on heterogeneous logs. HLAer is constructed in a purely data-oriented fashion and it is very general adapt to arbitrary log formats, applications or systems. The current implementation of HLAer is scalable to Big Data. Technical details of HLAer are documented in this manuscript together with comprehensive experiments. Comparison between HLAer with other log analysis tools is also presented. The current implementation of HLAer is demonstrated to be able to achieve the above goals and meanwhile outperforms other log analysis tools in terms of clustering/outlier detection performance, scalability and generalizability.

B. Further Discussions

1) Efficiency of outlier detection: The efficiency for the offline outlier detection is not considered as a big issue as long as clustering algorithm and similarity/distance calculation are efficient. However, the online outlier detection has much more critical demand on high efficiency. Currently, the online outlier detection is implemented via sequence-profile alignment. Such alignment has a quadratic complexity, which is not preferable in an online setting. The improvement can be achieved by implementing a specific parser for each pattern, and assigning the parsers that most favorably match the new log to parse the new log.

2) HLAer incremental updates: HLAer can be updated via two different ways. The first way is via offline upgrade. Once the most recent logs are accumulated after a certain period of time, a new HLAer can be built from such logs. This method will automatically adopt the most recent system status into the HLAer. The second way is via online upgrade. Once a new log comes in, it is routed down to a certain node of the HLAer tree, and the statistic of that node, e.g., pattern profiles, will be updated correspondingly. In addition, new nodes can be grown if there are enough simulated logs that satisfy the density or other condition so as to form a cluster. The dynamic growth of the tree can also be utilized as a real-time reflection of the system dynamic.

APPENDIX A

DBScan ALGORITHM

A. Definitions of distances

reachability-distance_{\epsilon,\text{minpts}}(d, p) = \begin{cases} 
\text{undefined} & \text{if } |\text{neighborPts}(d)| < \text{minpts}, \\
\max(\text{core-distance}_{\epsilon,\text{minpts}}(p), \text{distance}(d, p)) & \text{otherwise} 
\end{cases} 
(6)

core-distance_{\epsilon,\text{minpts}}(d) = \begin{cases} 
\text{undefined} & \text{if } |\text{neighborPts}(d)| < \text{minpts}, \\
\text{distance the minpts-th closest point} & \text{otherwise} 
\end{cases} 
(7)

B. The DBScan algorithm

REFERENCES


Algorithm 2: DBScan algorithm

**Input:** dataset $D$, Neighborhood$_D$, $\epsilon$, minpts

**Output:** Clusters $\{C\}$

foreach unvisited point $d \in D$ do

visited[$d$] ← 1

// Get all $d$'s neighboring points
neighborPts$_d$ = getNbrPts($d$, Neighborhood$_D$, $\epsilon$)

if sizeof(neighborPts$_d$) < minpts then

isNoise[$d$] ← 1

else

$C$ ← new cluster

expandCluster($d$, Neighborhood$_D$, neighborPts$_d$, $C$, $\epsilon$, minpts)

Algorithm 3: DBScan expandCluster algorithm

**Input:** data point $d$, Neighborhood$_D$, neighborPts$_d$, $C$, $\epsilon$, minpts

**Output:** The expanded cluster $C$

$C$ ← $C$ $\cup$ $d$

foreach $d'$ $\in$ neighborPts$_d$ $\&\&$ $d' \neq d$ do

visited[$d'$] ← 1

neighborPts$_{d'}$ = getNbrPts($d'$, Neighborhood$_D$, $\epsilon$)

if sizeof(neighborPts$_{d'}$) $\geq$ minpts then

neighborPts$_d$ ← neighborPts$_d$ $\cup$ neighborPts$_{d'}$

if $d'$ does not belong to any cluster then

$C$ ← $C$ $\cup$ $d'$


