

Using NoSQL Databases for Streaming Network Analysis

Brian Wylie*, Daniel Dunlavy*, Warren Davis IV*, Jeff Baumes**

*Sandia National Laboratories, **Kitware Inc

ABSTRACT

The high-volume, low-latency world of network traffic presents significant obstacles for complex analysis techniques. The unique challenge of adapting powerful but high-latency models to real-time network streams is the basis of our cyber security project. In this paper we discuss our use of NoSQL databases in a framework that enables the application of computationally expensive models against a real-time network data stream. We describe how this approach transforms the highly constrained (and sometimes arcane) world of real-time network analysis into a more developer friendly model that relaxes many of the traditional constraints associated with streaming data. Our primary use of the system is for conducting streaming text analysis and classification activities on a network link receiving ~200,000 emails per day.

Keywords: NoSQL, database, network, streaming, analysis, informatics, email, real-time.

Index Terms: C.2.3 [Computer-Communication Networks]: Network Operations – Network Monitoring; H.2.8 [Database Management]: Database Applications— data mining

1 INTRODUCTION AND RELATED WORK

The challenges associated with the processing and analyses of a live network stream are formidable. There are a myriad of open source toolkits used for the ingestion and display of network packet data; Snort [1], Wireshark (<http://www.wireshark.org>), and Bro IDS [2] are a few of the popular ones, with more listed at <http://sectools.org>. Such tools often provide a comprehensive set of filters or rules that can be applied to the network stream, which in the case of Snort can include upwards of 20,000 rules. Work focused on the analysis of network logs is also common. Splunk [12] is a popular commercial tool and Analysis Farm [13] uses a NoSQL database to improve aggregation and query performance. Although powerful, these low-level localized rule-based approaches often break down for higher-level analysis functions such as trend analysis of organizational email traffic or identification of high-risk web behavior.

The other extreme to these constrained streaming tools is to save the network captures to disk and utilize one of the analytically rich, publicly available toolkits such as WEKA [6], Orange [3], Titan Toolkit [7], etc., on the historical data. The significant downside to this file-based approach is that the analysis becomes forensic in nature and the developers may find themselves describing their new algorithm results against static data sources perhaps limiting the efficacy of the results when compared to approaches running on “live” production network streams.

The NoSQL database community is actively using these data stores in increasingly novel ways [8], [9], [10] and [11]; in particular our work is focused on the use of NoSQL databases to cross the chasm between traditional offline analysis and real-time network systems. This approach enables the application of analytical models to streaming network data with the results presented to a network defender within 10-20 seconds. In addition, our system provides a flexible environment where

different programming languages and scripts are welcome, components are interchangeable, and most importantly it’s remarkably resilient to unstable research-focused code.

2 APPROACH

Initially our project investigated a development model within an existing real-time packet processing toolkit but we quickly encountered constraints that would not allow us to work on our higher-level analysis goals. We also struggled with the unfamiliar and inflexible development environment. The team then moved to an offline development model, using our favorite toolkits and scripting languages. After several meetings with the network incident response team we realized that our techniques would have substantially less impact if not incorporated into the production network system. Thus the project had three significant tension points:

- Familiar development environment *in tension* with a constrained and unfamiliar packet processing library
- Agile research software (read: “buggy”) *in tension* with its deployment into a production system
- Analysis goals requiring large windows of data *in tension* with the transient, streaming network data

Our approach incorporates the use of a NoSQL database as the key element in the mitigation of these three tension points (see Figure 1).

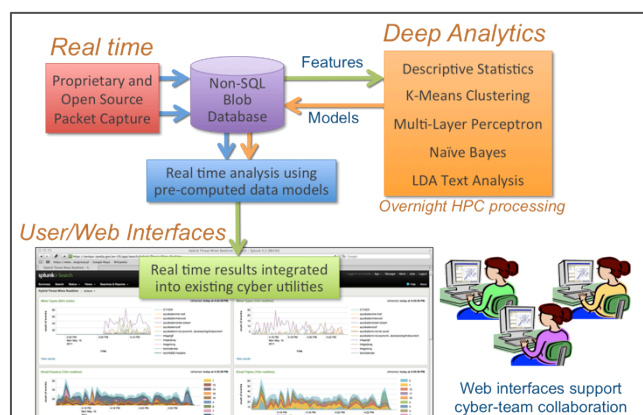


Figure 1: Conceptual diagram of the approach used for our cyber defense system illustrating the central role served by our NoSQL database.

2.1 NoSQL vs. Traditional RDBMS

Here we briefly discuss our use of a NoSQL database (or data store) as opposed to a traditional database (often referred to as an RDBMS). We chose NoSQL primarily because its schema-less key-value storage allows arbitrarily organized data into the database, and secondly because of its simple mechanisms for storage of binary data. The processing and analysis of malicious emails is one of our project’s major use cases and the storage of email serves as a good exemplar for both of these issues.

Field	Value
_attachments	<ul style="list-style-type: none"> payload_2: 1.1 KB, application/msword SESSION: 4.5 KB, application/msword; name="A CONSOLATION PRIZE.rtf" payload_8: 252 bytes, text/plain payload_1: 381 bytes, text/html
_id	"bba50a8de259940a936b9aa7012a75ad"
_rev	"5-2412ffa44fb08961b876f580c6eac8e4"
CLIENTPORT	44721
CONTENTENCODING	"base64"
CONTENTTYPE	"multipart/mixed; boundary=""B-429355456-1298187185--91866""

Figure 2: Screenshot of an email stored in one of our NoSQL databases (CouchDB <http://couchdb.apache.org>).

The storage of emails exercises both the schema-less architecture (arbitrary metadata fields) and the binary storage (for attachments). The email depicted in Figure 2 has been cropped but this particular email contains 15 metadata fields of various origin, an attachment (a Microsoft Word doc) and two bundled mime-types (html and plain text).

2.2 Benefits of a loosely coupled system

Traditionally network packet processing and analysis is accomplished with a tightly coupled software system. The “always-on” world of network packets will often impose a fairly rigorous set of system constraints around latency, throughput, performance and resources. In our experiences these constraints often lead to frustrations either with the development environment or in our inability accomplish a higher-level analysis task. The fundamental role of the database in our system is to change the traditional tightly coupled software system into a loosely coupled federated system. The benefits of this transformation are substantial and reflect themselves throughout the rest of the paper; in particular the loose coupling helps us address many of the key challenges associated with our tension points.

2.3 Packets to database

Packet processing is a production world where services and systems are literally expected to run 24/7 bug-free. As developers in a research project, we struggled with the pragmatics of deploying new algorithms within one of these environments. We have to balance our enthusiasm for new algorithms and tools with maintaining flawless production-system stability. Our approach was to completely decouple the packet processing from the rest of the software pipeline; we are currently using the open source Bro IDS and an internal proprietary system. Both systems are interchangeable and primarily perform packet reordering and re-sessionization. Email capture is a good example of what functionality is required during packet processing (see Figure 3).

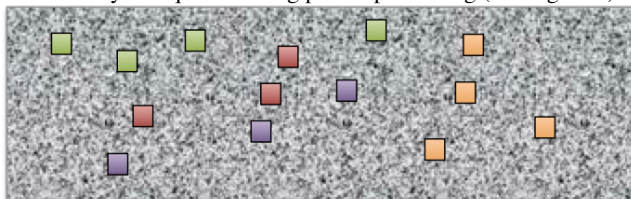


Figure 3: Four emails (green, red, purple and orange) scattered within the deluge of data packets transported by a network link. A typical corporate network supports a large number of protocols: HTTP(web), FTP(file transfer), DNS(name resolution), and SMTP(email).

In Figure 3, four different emails, each split across several packets, exist within the entire set of captured network traffic. The packet processing system simply aggregates the packets associated with a particular network session and places them into our NoSQL database along with a small amount of metadata (source_ip, dest_ip, port, flow_id, and datetime).

2.4 Feature Extraction

The next step in our pipelined process is feature extraction, which clearly demonstrates the benefits afforded by both the NoSQL database and the loosely coupled system. The aggregated packets are stored in the database as raw bytes by the packet capture engine. The organization of bytes in an email message, which in actuality is a fairly sophisticated hierarchical container, can be complex and is covered in numerous RFCs. Fortunately there is a Python module (email) that helps decipher, traverse and unpack an email into its constituent parts: plain text, html, attachments, and the long, *variable length*, set of metadata tags. The feature extraction script simply loops over all new incoming emails, pulls the raw bytes from the database, organizes the email into its constituent parts and places those back into the database. This approach enables the developers (and more importantly the network defenders) to see everything about that email from within a single record interface, including, if necessary, the raw bytes associated with the network packet capture (see Figure 2). At this point the email is ready for downstream processing by various analytic algorithms.

2.5 MapReduce Views

As an alternative to the traditional relational query language (SQL), many NoSQL databases provide filtering and MapReduce frameworks. A variant of MapReduce called incremental MapReduce provides capability that becomes absolutely critical when working with streaming data stores. Incremental MapReduce means that the MapReduce can be incrementally updated with just the new data coming in and the results are valid for the entire data store. CouchDB supports incremental MapReduce natively and MongoDB supports it indirectly with some additional bookkeeping.

2.5.1 Pipeline Management

Incremental MapReduce views manage the entire pipeline process efficiency in the complex world where new data is continuously streaming in and existing data is continuously changing state. A concrete example of pipeline management is feature extraction, the second stage in our pipeline. When emails first come into the database they are large binary objects and do not contain much metadata. The feature extraction view in the NoSQL database emits documents that do not yet contain the *features_extracted* tag. A script polls that view, gets any new incoming documents, extracts features for those documents, updates their state (i.e., sets *features_extracted=True*) and moves on. Benefits to this approach include the following:

- If the feature extraction script crashes, the document causing the crash will not get marked and can be used for debugging. Because we are no longer tightly coupled with the packet processing, analysis scripts crashing at this point have no impact on the packet collection. This benefit in particular cannot be overstated as it squarely addresses one of our key tension points, allowing research software to run side-by-side with mission critical network processes in a production environment.

- After a crashing script is fixed, it can simply be restarted and it will automatically play catch-up on the emails that have piled up in the meantime due to explicit state handling.
- At any point scripts can be run to that reset the state of some of the documents to any point in the pipeline. If a new version of the feature extraction script is developed, resetting documents from the last several last days allows the new capabilities to easily be applied to historical as well as new, incoming data.

2.5.2 Text Analysis

The system’s real power is the ability to compute a set of analytical models on the corpus as a whole and then have those models applied to the real-time incoming network stream. Looking at Figure 6, we see that the input into our Latent Dirichlet Allocation (LDA) model for text analysis is a sparse feature frequency matrix that can be extremely large (number of emails X number of features). In particular, the use of two incremental MapReduce views above allow this matrix to be kept up-to-date in real-time for a large number of emails.

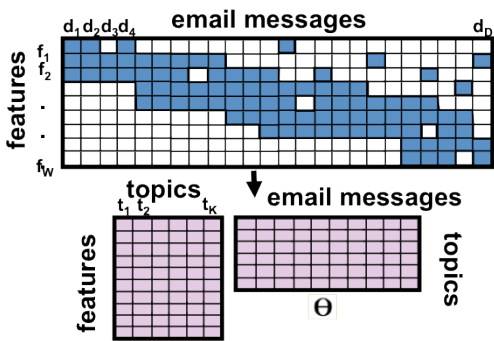


Figure 6: The sparse input feature matrix to our Latent Dirichlet Allocation (LDA) model is constructed from the incremental MapReduce views provided by our NoSQL database.

The views used for text analysis processing are more complex and illustrate the true power of incremental MapReduce. As incoming emails enter the system the views are automatically updated also with the changed-based engine driving the CouchDB views ‘exiting’ emails (those expired after some time window) are also properly accounted for.

Here we provide pseudo code for both feature dictionary and feature table mapreduce views (views in CouchDB are written in Erlang).

```
<<< Feature Dictionary View >>>
Map:
  if doc["include_in_model"]:
    for feature,count in doc["feature_vector"]:
      emit (feature, count)

Reduce:
  # Reduce sums the counts associated with each feature
  function (key, values) {
    return sum(values)
  }

<<< Feature Table View >>>
Map:
  if doc["include_in_model"]:
    for feature,count in doc["feature_vector"]:
      emit (doc["id"], feature, count)

Reduce:
  # None
```

2.6 Model Generation and Evaluation

The detailed coverage of the algorithms used for model generation and evaluation are beyond the scope of this paper, but we did want to illustrate one concrete example of how the database provides benefits for model generation, model storage, and real-time model evaluation against incoming network data.

The use of a sparse feature frequency matrix (features by documents) is common for text/feature analysis algorithms. Although conceptually simple, the construction and maintenance of this matrix can be challenging when placed into a system where new data is continuously coming in, old data is getting expired, and the columns of the matrix might actually contain only samples from a large database (our system supports all of the above). By utilizing the views described in section 2.5 we can properly construct and maintain this matrix. The *feature_dictionary_view* provides a global index for each unique feature and the *feature_table_view* provides the features associated with each observation (i.e., document). The generation and evaluation of models varies based on domain and model types, but for our example of using LDA on email features the method is as follows:

- Run a sampling process that uniformly samples without replacement a large number of the observations in the database (by simply adding the *include_in_model* tag to each of the samples).
- The views are then automatically updated with the new sampled observations.
- Run model generation process: pull the views described above that generate the feature matrix, which is used downstream to generate various models.
- The models are given a revision number and stored into the database (for provenance purposes).
- Run the model evaluation process: Pull the models plus any new emails (i.e., those not yet processed by the updated models) from the database, evaluate emails against the models and place the results of the evaluations back into the database.

The flexibility afforded by this approach is significant, squarely hitting our “familiar development environment” tension point. Using the database as an integration point means each part of the process is orthogonal to the rest; the sampling could be a Python script using an “R” interface, the model generation and evaluation could be done using a Java Library, and so on. Also, since the database drivers are socket based, each part of the process could be run on entirely separate machines. This is often helpful when developing or verifying particular components of the pipeline.

2.7 Web Interface

The results of this pipeline must be presented to the end user in order to make useful decisions based on the available data. We chose the web for our UI deployment target due to its increasingly rich set of expressiveness through HTML5 and its ease of deployment to a wide number of users. The web application environment in Splunk enables query management and data drill-down, in addition to a standard set of visualization tools. We augmented these tools with graph layouts and charts (see Figure 7) based on the D3 Javascript library [5]. The use of a NoSQL

database enables the web interface to interactively query the database for information through the use of asynchronous Javascript (Ajax).

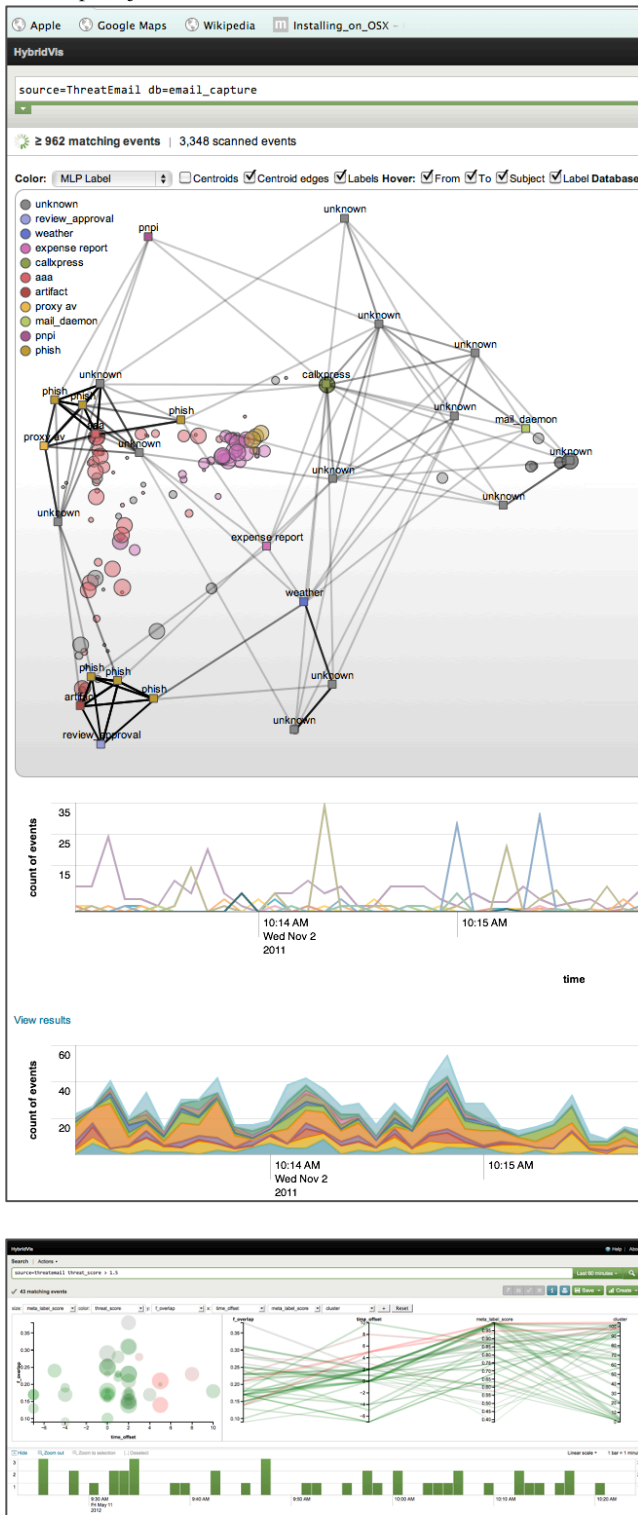


Figure 7: (Top) Web interface used to communicate the analysis results to the network defender, where dots fade in and out in real-time with model-based location. (Bottom) Another form of the web interface with scatter plots, parallel coordinates, and a histogram showing emails within the last 60 minutes.

3 CONCLUSIONS AND FUTURE WORK

In this paper we have discussed an approach for real-time data analytics on a network data stream. We found that the approach of using a NoSQL database allowed us to use decoupled components and incorporate familiar toolkits and development environments against real-time streaming network data. Our goal of conducting text analysis and classification experiments against a real-time email stream (~200,000 emails / day) was greatly simplified by using this loosely coupled system. Follow up work is investigating a variant of this system to track web traffic. The variant approach uses MongoDB as the data store and incremental mapreduce to keep site wide aggregate counts of web requests for both individual sites and domains (~50 Million HTTP requests/day).

4 ACKNOWLEDGEMENTS

This work was funded by Sandia National Laboratories, a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

5 REFERENCES

- [1] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks", Proceedings of LISA '99: 13th Systems Administration Conference, Seattle, Washington, USA, November 7-12, 1999.
- [2] V. Paxson, "Bro: A System for Detecting Network Intruders in Real-Time Computer Networks", 31(23-24), pp. 2435-2463, 1999.
- [3] T. Curk, J. Demšar, Q. Xu, G. Leban, U. Petrović, I. Bratko, G. Shaulsky, B. Zupan. "Microarray data mining with visual programming". Bioinformatics. 2005 Feb 1;21(3):396-8. [Orange]
- [4] S. Bird, E. Klein, and E. Loper, "Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit", O'Reilly Media, 2009.
- [5] M. Bostock, V. Ogievetsky, J. Heer, "D3: Data-Driven Documents", IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis), 2011.
- [6] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, "The WEKA Data Mining Software: An Update", SIGKDD Explorations, Volume 11, Issue 1, 2009.
- [7] B. Wylie, J. Baumes, "A Unified Toolkit for Information and Scientific Visualization", IS&T/SPIE Electronic Imaging 2009, Visual Data Analytics (VDA 2009), 2009.
- [8] P. Näsholm, "Extracting Data from NoSQL Databases-A Step towards Interactive Visual Analysis of NoSQL Data," 2012.
- [9] T.A.M.C Thanriwatt, C.I. Keppetiayama, "NoSQL query processing system for wireless ad-hoc and sensor networks," International Conference on Advances in ICT for Emerging Regions (ICTer), vol., no., pp.78-82, 1-2 Sept. 2011.
- [10] E. Meijer and G. Bierman, "A Co-Relational Model of Data for Large Shared Data Banks," Commun Acn, vol. 54, no. 4, pp. 49-58, 2011.
- [11] R. Cattell, "Scalable SQL and NoSQL data stores," SIGMOD Record, vol. 39, no. 4, May 2011.
- [12] Burns, Bryan; Killion, Dave; Beauchesne, Nicolas; Moret, Eric; Sobrier, Julien; Lynn, Michael; Markham, Eric; Iezzoni, Chris; Biondi, Philippe; Granick, Jennifer; W. Manzuik, Steve; Guersch, Paul. "Security Power Tools." O'Reilly Media, Inc., ISBN 0-596-00963-1.
- [13] Jianwen Wei, Yusu Zhao, Kaida Jiang, Rui Xie, Yaohui Jin, "Analysis farm: A cloud-based scalable aggregation and query platform for network log analysis," International Conference on Cloud and Service Computing (CSC), vol., no., pp.354-359, 12-14 Dec. 2011