Detecting Distributed Scans Using High-Performance Query-Driven Visualization

Abstract

Modern forensic analytics applications, like network traffic analysis, perform high-performance hypothesis testing, knowledge discovery and data mining on very large datasets. One essential strategy to reduce the time required for these operations is to select only the most relevant data records for a given computation. In this paper, we present a set of parallel algorithms that demonstrate how an efficient selection mechanism – bitmap indexing – significantly speeds up a common analysis task, namely, computing *conditional histogram* on very large datasets. We present a thorough study of the performance characteristics of the parallel conditional histogram algorithms. As a case study, we compute conditional histograms for detecting distributed scans hidden in a dataset consisting of approximately 2.5 billion network connection records. We show that these conditional histograms can be computed on interactive time scale (i.e., in seconds). We also show how to progressively modify the selection criteria to narrow the analysis and find the sources of the distributed scans.

Keywords: query-driven visualization, network security, network connection analysis, data mining, visual analytics

1 Introduction

A typical day's worth of network traffic at an "average" government research laboratory may consist of tens of millions of connections comprising multiple gigabytes worth of connection records. These connection records can be considered as "conversations" between two hosts on a network. They are generated by routers, traffic analyzers or security systems, and contain information such as source and destination IP addresses, source and destination ports, duration of connection, number of bytes exchanged and so forth. A year's worth of such data currently requires on the order of tens of terabytes or more of storage. According to Burrescia [Burrescia and Johnston 2005], traffic volume over ESnet, a production network servicing the U. S. Department of Energy's research laboratories, has been increasing by an order of magnitude every 46 months since 1990. This trend is expected to continue into the foreseeable future.

The steady increase in network traffic volume exacerbates the difficulty of forensic cybersecurity and network performance analysis. Current network traffic analysis toolsets often rely on simple utilities like grep, awk and gnuplot. While sufficient for analyzing small amounts of network traffic data, these utilities do not scale nor perform to the level needed for analyzing current or future levels of network traffic. To address the need for rapid forensic analysis capabilities, our work presents advances in two complementary technology areas, namely scientific data management and visual analytics.

Interactive network traffic data analysis is often based on histogram methods. One key feature distinguishing histograms in data mining applications from other types of applications [Ioannidis 2003] is the use of ad-hoc conditions to reduce the number of data records being analyzed. For example, to identify unusual activities on TCP ports, we may compute a histogram over destination ports and time to discover high or unusual levels of activity on a particular port. However, rather than computing a histogram over all data records, we might restrict the computation to those network sessions to a set of criteria like those that originate from a specific IP address or address range. The restriction on originating IP or other conditions is referred to an *external condition*. Analyses of data records subject to such external conditions are commonly known as *conditional analyses*. Our objective here is to develop efficient algorithms for a special type of conditional analysis, i.e., computing multi-variate histograms under arbitrary external conditions. We call this type of histogram *conditional histogram*.

The main contributions of this paper are as follows:

- We build upon recent previous work that combines state-of-the-art indexing with a novel approach to visual analytics [Bethel et al. 2006]. In this paper we introduce a new family of parallel algorithms for efficiently building 2D conditional histograms. The key feature of these parallel algorithms is that they operate on bitmap indices for evaluating the external conditions to efficiently populate histogram bins.
- We present a detailed performance analysis of these conditional histogramming algorithms on a 32-way parallel SMP platform. We show that this work helps accelerate the most data-intensive and time-consuming part of forensic analysis, namely data mining and knowledge discovery based upon multi-dimensional queries.
- We apply the rapid conditional histogramming technology in conjunction with a specialized visual analytics application for detecting and analyzing distributed scans. The data for this case study covers a 42-week period at a US government science laboratory. The case study and its forensic analysis would not be possible without the combination of technologies from scientific data management and visual analytics.

2 Previous Work

Our work is based on a multi-disciplinary approach of techniques in network traffic analysis, efficient querying and indexing, and query-driven visualization. In this section we give a brief overview of the related work in these areas.

2.1 Network Traffic Analysis and Visualization

A network connection can be thought of as a set of packets passing between two hosts within a given time interval that have common characteristics. An example of a network connection is a single communication session or an interaction between two hosts on the Internet. Several standard tools exist for capturing network connection data. For larger environments, routers and switches can provide connection data in specialized formats such as NetFlow [Systems 2005] or SFlow [Phaal et al. 2001]. Another tool for analyzing network connection

data is Bro IDS [Paxson 1998]. Bro can output connection records, which are slightly different than traditional flow records, but that contain substantially the same information. Other systems generate a single full-duplex connection record that also contains byte and packet counts for the reverse direction [Uphoff and Criscuolo 2004].

For network connection data analysis, the special purpose systems and software provided by network equipment vendors for analyzing NetFlow and SFlow records are typically optimized for aggregate network usage and trend analysis rather than forensic analysis. Freely available tools are typically architected as a collection of command-line utilities for collecting, concatenating, filtering, and summarizing network connection data. Fullmer's OSU Flow-Tools [Fullmer and Romig 2000] is a good example of such a collection that is best suited for small-scale analysis. Plonka's FlowScan suite [Plonka 2000] consists of a collection of perl scripts and modules that provide stateful flow inspection and charting/graphing capabilities. Both these utilities store data in flat files in a binary format (FlowScan uses a compressed binary format). Both use a sequential scan through data to accomplish filtering operations. Gates' SiLK Suite [Gates et al. 2004] is a collection of software utilities for flow data collection and analysis suite that uses a reduced-size flow record in conjunction with data compression and file/directory hierarchies to maximize filtering performance and minimize data footprint size on disk. SiLK uses bitmaps to accelerate filtering operations based upon IP address, but does not use bitmap indexing or any other form of indexing structure, other than a tree-like directory and file hierarchy, to accelerate query operations.

Intrusion Detection Systems (IDS) like Bro are typically good at analyzing events that are closely correlated in time where analysis can rapidly yield actionable results. However, over large time scales and/or very large data sets, the design decisions that make IDS software appropriate for rapid response limit applicability to problems of scale. Typical failure modes include unbounded memory consumption or computational overload that inhibits a rapid response.

Traditional high volume network traffic analysis usually begins after the IDS provides an alert, such as when one or more IP addresses are associated with a possible attack. The tool most typically used for searching text-format connection records is grep. Search results post-processing is typically performed with tools such as perl, shell scripts and gnuplot. These tools work well for analyzing moderate amounts of network traffic data. However, they do not scale up to terascale data sizes.

Previous work on network traffic visualization has focused on filtering and visual presentation of different types of network traffic data, including connection data, routing information, IDS alerts and so forth.

One of the most widely deployed tools for visualizing network traffic levels and statistics is MRTG [Oetiker 2006a], which relies on RRDtool [Oetiker 2006b] for charting and graphing. These tools are in use at nearly all sites that run production networks for the purpose of showing traffic levels at multiple temporal resolutions. One shortcoming of RRDtool is that detail in data temporally distant from the present is lost due to its being summarized and averaged in the round-robin database.

More recently, [Lau 2004; McPherson et al. 2004; Lakkaraju et al. 2004] describe applications that map network connection variables to axes, then present activity, or lack thereof, at the appropriate grid location (a form of histogram). The basic idea is to facilitate rapid visual discovery of features, patterns or activity in network traffic or network connection data. Other applications use a linked node concept to convey the presence of traffic or traffic levels between addresses mapped to a grid [McPherson et al. 2004; Yin et al. 2004; Goodall et al. 2005], to geographic locations or some other representation of network topology [Cox et al. 1996; Koutsofios et al. 1999], or to a task-oriented metaphorical representation of internal and external networks [Fisk et al. 2003; Livnat et al. 2005; Goodall et al. 2005]. Several other applications focus on visualization of IDS alerts [Komlodi et al. 2005; Livnat et al. 2005]. Within this constellation of previous visualization research, the work we describe here focuses on the interactive drill-down to first compute then display network connection statistics using histograms that show levels of activity satisfying user-defined query conditions to implement knowledge discovery and hypothesis testing on large collections of network connection data.

While these previous works in network traffic visualization have produced novel and useful presentation and interaction techniques, they were tested using small amounts of network data. In contrast, our work focuses on techniques suitable for use with large collections of network connection data. Our "filtering" is implemented by efficient compressed bitmap indices designed for rapid data mining of realistic-sized datasets.

2.2 Indexing and Querying

To search quickly through large amounts of network connection data, we use bitmap indices [O'Neil 1987; Chan and Ioannidis 1998; Johnson 1999; Shoshani et al. 1999; Stockinger et al. 2000], an indexing structure optimized for read-only data. Bitmap indices are well suited for use in indexing and querying network connection data, which is typically not modified once stored. Bitmap indexing has been implemented in many commercial database systems. Its efficiency is well accepted for low cardinality attributes. We recently showed that bitmap indices are also efficient with high cardinality attributes by using Word Aligned Hybrid coding (WAH) [Wu et al. 2004]. FastBit [Scientific Data Management Group 2006] is a research code that implements a number of different forms of bitmap index compression, including WAH.

Network analysts usually explore network connection data iteratively and in an ad-hoc manner, i.e. the queries often do not follow a predefined pattern. Moreover, queries are often complex and contain several attributes (dimensions) like "Find all network connection records where StartTime $>$ 20050501 AND 10.102.0.0 $<=$ SourceIP $<=$ 10.105.255.255." Bitmap indices have a unique advantage over traditional tree-based indices for these kind of multi-dimensional queries since the dimensions can efficiently be combined without suffering from the "curse of dimensionality" [Bellman 1961].

2.3 Query-Driven Visualization

As data grows larger and more complex, building and using larger and more scalable visualization systems simply produces a greater amount of output, which in turn increases cognitive load on the viewer without any explicit guarantee of an increase in potential or actual insight. The goal of *Query-Driven Visualization* is to limit visualization processing and subsequent visual interpretation to data that is deemed to be "interesting."

Within the context of network traffic visualization and analysis, many previous works include some notion of "data filtering" as part of their knowledge discovery process. The OSU Flow-Tools [Fullmer and Romig 2000], FlowScan [Plonka 2000] and SiLK [Gates et al. 2004] all have command-line interfaces to filtering utilities that find records matching a set of criteria. Filtering selection criteria include connection record variable values like source or destination IP or port number, IP protocol, interface number, and autonomous system number. For most of these systems, filtering is performed using a sequential scan through flat files. The computational complexity of such an approach is $O(n)$ where n is the number of connection records. SiLK uses a tree-like directory and file hierarchy to partition data to accelerate searches.

Swift-3D [Koutsofios et al. 1999] is an integrated data visualization and exploration system. It supports C-style query expressions that are translated into C code, compiled into shared objects and dynamically linked into the application. This approach offers flexibility similar to that of *awk* and *sed* but with the speed of compiled code. This particular work achieves high performance by using Direct-IO to bypass the kernel, but does not appear to use any form of indexing to accelerate the search process. This approach is also $O(n)$ in computational complexity with respect to data size.

The VisDB system combines a guided query-formulation facility with relevance-based visualization [Keim and Kriegel 1994]. Each data item in a dataset is ranked in terms of its relevance to a query, and the top quartile of results is then input to a visualization and rendering pipeline. Data is presented in a way to cluster more relevant items closer together, and less relevant items further apart. It is especially well-suited to display the results of "fuzzy queries" in that inexact matches are ranked and visually displayed in a way to convey relevance. This approach results in $O(n)$ complexity.

The TimeFinder system described in [Hochheiser and Shneiderman 2001] supports interactive exploration of time-varying data sets by providing the ability to quickly construct queries, modify parameters, and visualize query results. The query results are presented in a fashion that implements a form of data mining – more detailed information about the items satisfying the query are presented in a separate window. TimeFinder reads all data into memory and is therefore able to operate on only modest-sized datasets. Again, the complexity of this filtering approach is $O(n)$.

2.4 Our Contribution

The notion of coupling visualization with high performance index/query technology was introduced in [Stockinger et al. 2005b]. We showed that the computational complexity of visualization processing can be constrained to the number of items returned by a query. As such, that approach is the most suitable for use in query-driven visualization and analysis of very large multi-dimensional datasets. Those principles were subsequently applied to rapid network traffic analysis in [Bethel et al. 2006].

The work we present in this paper extends that of both [Stockinger et al. 2005b; Bethel et al. 2006] in two key ways. First, the iterative data mining queries rely on a new family of parallel algorithms for building two-dimensional *conditional histograms*. We study the design and implementation of these novel parallel algorithms and provide a detailed performance characterization. Second, we apply these new histogramming algorithms to a 42-week collection of network connection data to detect and characterize distributed scans.

3 Parallel Conditional Histogram Algorithms

In our previous work [Bethel et al. 2006] we used a simple parallelization strategy for building conditional histograms. In this paper we introduce three parallelization strategies for building conditional histograms that offer significantly improved performance compared to our initial algorithm. These new algorithms achieve better performance by eliminating redundant work and leveraging the regular structure of the histograms to streamline the process of computing histogram bin values. One key idea of these novel algorithms is to use automatic *task* or *data parallelism* for populating histogram bins.

First, we provide a running example for motivating the problem of conditional histograms. Then we discuss the initial parallelization strategy that we used in [Bethel et al. 2006]. Next, we provide a detailed description of the three novel parallelization algorithms for building 2D conditional histograms.

3.1 Running Example

To simplify the description of the algorithms, we use an example where we compute a simple two-dimensional (2D) histogram. Assume that each histogram dimension (variable) is binned separately. Further, assume that the overall histogram is defined on bins formed by the cross product of the two one-dimensional bins. We denote the two variables as **A** and **B**. Next, we denote the number of bins for **A** as n_A and the number of bins for **B** as n_B. Finally, the histogram is built by computing the number of records that fall in each of the $n_A n_B$ bins.

The following is a specific 2D histogram example that we will use throughout this section.

```
histogram DP:5000:6000:125, tsyday:127:183:7
          where (state == 1) and (12 <= tshour < 14)
```

The given example contains a *bin specification* in the first line and a set of *external conditions* in the second line. The bin specification is defined by the triple `< lowerBinBoundary : upperBinBoundery : binWidth >`. In this example, the lower bin boundary for the variable `DP` is 5000, the upper bin boundary is 6000 and the bin width of 125. Hence, the total number of bins for variable `DP` is 8. Similarly, the bin boundaries for variable `tsday` are 127 and 183, respectively. The bin width is 7. Again, the total number of bins for variable `tsday` is 8.

3.2 Details of the Parallel Algorithms

First, we revise the original conditional histogramming algorithm in our previous work [Bethel et al. 2006]. Next, we provide a detailed description of three novel conditional algorithms. The names of these algorithms and their brief descriptions are given in Table 1.

Algorithm	Explanation
BAQ	Bin-A-Query: use a query to count the number of records in a bin
LAQ	Line-A-Query: use a query to filter the right data for a *column* of the multi-dimensional histogram
RAQ	Rectangle-A-Query: use a query to filter the right data for a *rectangular subset* of the multi-dimensional histogram
FEP	Filter-External-Partitions: - filter the externally partitioned records - count each partition separately - accumulate the results

Table 1: A summary of the conditional histogram algorithms.

In the discussion that follows, we are addressing task and data decomposition alternatives for each of the four algorithms. The software performs automatic parallelization on behalf of the calling application and significantly affects the overall performance. With respect to automatic parallelization, each of these algorithms will partition the data or tasks for the running example query in different ways. One of the algorithms, BAQ, can compute histograms using only the query capability of FastBit. The other three algorithms – LAQ, RAQ, FEP – use a combination of FastBit query capabilites to first select a set of records appropriate for a particular data decomposition, then use a second-phase algorithm to compute histogram subsets using raw data records. We take such an approach for the LAQ, RAQ and FEP algorithms after having observed substantial processing time in BAQ that appears to be due to redundant work associated with query processing.

The BAQ, LAQ and RAQ algorithms are all *task parallel*, while the FEP algorithm is *data parallel*. In the task parallel family of algorithms, each PE will fill a histogram bin or bin range; each PE will operate on the complete dataset to complete its work task. In the data parallel algorithm, each PE is computing all histogram bins on a subset of the data.

For the task parallel algorithms, we store each variable's raw data in a separate file. Additionally, each variable's bitmap index is also stored in a separate file. If there are V variables, then there would be $2V$ files. For the data parallel algorithm, each variable's raw data and associated bitmap indices are stored in p separate files where p is the number of PEs. Each of the p data files contains approximately the same amount of data. Therefore, if there are V variables and p PEs, there are a total of $2Vp$ files. Our performance experiments are conducted on an SMP platform so all PEs have access to a single, fast filesystem.

3.2.1 Algorithm Bin-A-Query (BAQ)

The BAQ algorithm uses only the query capability of FastBit to compute histograms. It builds a query for each bin to count the number of records that fall in the bin. Take the above example of filling a 8x8 histogram for `DP` and `tsyday`. The conditions for each query are the combination of the external conditions and the boundaries of the bins. For example, the first bin would have `DP` in the range of 5000 and 5125, and `tsyday` in the range of 127 and 134. The query to fill the first bin would be:

```
select count(*) where
 (state == 1) and (12 <= tshour < 14) and
 (5000 <= DP < 5125) and (127 <= tsyday < 134).
```

Similarly, for a bin with `DP` between 5125 and 5250 and `tsyday` between 134 and 141, the query would be:

```
select count(*) where
 (state == 1) and (12 <= tshour < 14) and
 (5125 <= DP < 5250) and (134 <= tsyday < 141).
```

In this case, a total of 64 queries are executed to compute an 8x8 histogram. If we a assume a parallel computer with 4 PEs, each PE executes 16 queries as illustrated in Figure 1(a). Since these queries can be executed independently, this algorithm is easily parallelized.

However, one inefficiency we see in this approach is that it executes an excessive number of FastBit queries: Since each query evaluates the external conditions `((state == 1) and (12 <= tshour < 14))` independently, this approach performs redundant work and could thus be inefficient. The remaining algorithms – LAQ, RAQ and FEP – can be viewed as alternative strategies to reduce these repeated evaluations of the external conditions.

3.2.2 Algorithm Line-A-Query (LAQ)

The LAQ algorithm uses a filtering capability to compute the histogram with less queries than BAQ. In essence, LAQ slices a multi-dimensional histogram into many 1D histograms. The basic idea of LAQ is as follows: (1) use FastBit to find the records for all bins in a 1D "slice" of a 2D histogram and (2) implement a counting function for populating the respective bins.

Assuming that we choose to slice the above 2D histogram along `DP` (see Figure 1(b)), the queries we use will select `tsyday` as output. This will lead to 8 queries of the form:

```
select tsyday where
 (state == 1) and (12 <= tshour < 14) and
```

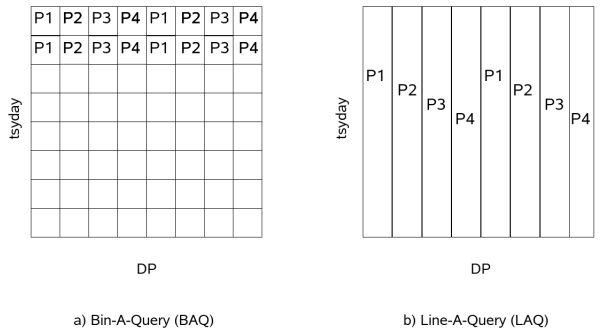


Figure 1: A schematic diagram depicting how algorithms BAQ and LAQ compute a 2D conditional histogram.

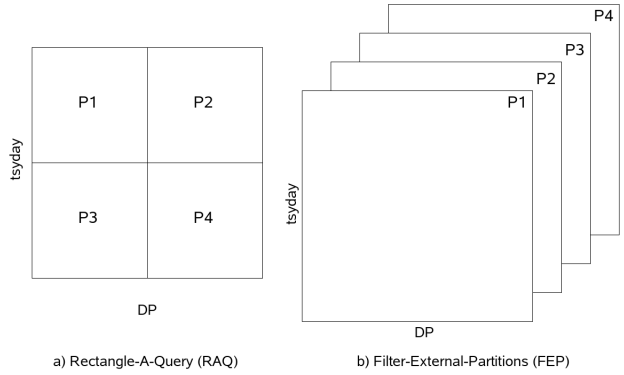


Figure 2: A schematic diagram depicting how algorithms RAQ and FEP compute a 2D conditional histogram.

```
 (5000 <= DP < 5125) and (127 <= tsyday < 183).

select tsyday where
 (state == 1) and (12 <= tshour < 14) and
 (5125 <= DP < 5250) and (127 <= tsyday < 183).

...
```

In contrast to algorithm BAQ, LAQ needs to read part of the raw data records in addition to reading the bitmap index. Because of this additional access to raw data, we expect each LAQ query could take more time than would be required in a BAQ approach. However, because the LAQ algorithm executes less queries overall, it is possible it may perform better than BAQ. We discuss this issue in more detail in Section 4.

After LAQ first uses FastBit to select the set of records that fall within its 1D "slice," it must next invoke its counting function to compute all histogram bin values within the slice. The bulk of computation in this function is to locate the right bin for each record and increment the bin's count. If the bins for `tsyday` are arbitrarily defined, we may need to use $\mathcal{O}(\log(b))$ time to place each record to the right bin, where b denotes the number of bins. However, for bins specified with the triple `< lowerBinBoundary : upperBinBoundery : binWidth >`, LAQ uses a simple computation to determine in which bin a record falls. In this case, the time needed to determine the bin for any data record is considered to be constant since it does not depend on the number of bins, and the time required to compute the 1D histogram is proportional to the number of records FastBit selects for the 1D slice.

3.2.3 Algorithm Rectangle-A-Query (RAQ)

As in algorithm LAQ, RAQ first uses a FastBit query to filter a subset of records and then uses a counting function to populate a subset of bins of the final histogram. Whereas LAQ partitions the 2D histogram task space into 1D histogram slices, RAQ partitions the task space into "tiles" or "rectangles" of smaller 2D histograms. RAQ partitions the task space into as many rectangles as there are PEs (see Figure 2(a)). In the implementation we present here, our partitioning strategy aims to have all partitioned rectangles have the same number of bins to evenly divide the tasks amongst PEs. Other 2D partitioning schemes could result in better load balance.

Assuming we are computing the above histogram on 4 PEs, we could partition the 64 bins into 4 rectangles of 16 bins each, e.g., "`5000 <= DP < 5500 and 127 <= tsyday < 155`," "`5000 <= DP < 5500 and 155 <= tsyday < 183`," "`5500 <= DP < 6000 and 127 <= tsyday < 155`," and "`5500 <= DP < 6000 and 155 <= tsyday < 183`." The query for filtering the records that fall in the first rectangle is:

```
select DP, tsyday where
 (state == 1) and (12 <= tshour < 14) and
 (5000 <= DP < 5500) and (127 <= tsyday < 155).
```

The counting function to fill a multi-dimensional histogram needs to place the incoming records into appropriate bins. Let us assume that 2D bins are a cross product of bins for each 1D variable. Thus, by determining in which 1D bin each variable falls, one can also determine in which 2D bin a record falls. Given the bins are specified using the triple `< lowerBinBoundary : upperBinBoundery : binWidth >`, the time required by this basic histogram function is proportional to the number of records selected for each 2D rectangle.

3.2.4 Algorithm Filter-External-Partitions (FEP)

An alternative to dividing the *task* according to the histogram bins is to divide the *data records* among the PEs (see Figure 2(b)). In the data parallel approach, each PE computes *all* histogram bins for a subset of n/p data records, where n is the total number of data records and p is the number of PEs. Finally all histogram computations are combined with the final result.

In FEP, each PE executes a FastBit query to first filter the records from a different data partition and then computes a set of partial counts for every bin. A final step accumulates all the partial counts for every bin from all PEs (*result aggregation*). Since this final step is relatively simple, we would expect this approach to parallelize well. Here, the external conditions are evaluated on each PE, but on different subsets of records. Therefore, there is no redundant evaluation of the external conditions with respect to a given partition of data. All PEs use the same query to filter the data records. In our example, the query is as follows:

```
select DP, tsyday where
 (state == 1) and (12 <= tshour < 14) and
 (5000 <= DP < 6000) and (127 <= tsyday < 183).
```

FEP uses the same counting function is an algorithm RAQ to fill the arrays of partial counts on each PE. The total cost of this counting function is proportional to the number of records selected.

4 Run-Time Complexity

In this section we discuss the run-time complexity of the four parallel algorithms BAQ, LAQ, RAQ and FEP for building conditional histograms. Let us briefly summarize the key ideas of the four parallel algorithms before we proceed with an in-depth analysis.

In Section 3 we showed that BAQ computes histograms using only the query capability of FastBit. The other three algorithms – LAQ, RAQ, FEP – use a combination of FastBit query capabilites to first select a set of records appropriate for a particular data decomposition *(record selection)*, then use a second-phase algorithm to compute histogram subsets using raw data records (*record counting*).

We start our discussion with the algorithmic phase that is common to all four algorithms. Then we will discuss the remaining algorithmic phases for LAQ, RAQ and FEP. The main thesis motivating our development of these algorithms is:

- The cost of performing *redundant queries* for computing conditional histograms exceeds the cost of eliminating redundant queries by using FastBit queries to *evaluate external conditions* and then *compute* the records that fall into particular histogram bins.

4.1 Evaluate Query Conditions

From earlier analysis and timing measurements [Wu et al. 2004; Wu et al. 2006], we know that the average query processing time is proportional to the size of the bitmap indices. Since all queries in algorithm BAQ involve the same set of attributes, we would expect the total time to be proportional to the number of bins in the histogram. For a typical 2D conditional histogram containing 100×100 bins, there are plenty of queries for a moderate size parallel system to ensure that each PE executes about the same number of queries (*query load balancing*).

In terms of number of queries, algorithm BAQ evaluates $n_A n_B$ queries, LAQ evaluates n_A (or n_B) queries, RAQ evaluates p queries (where p is the number of PEs), and FEP evaluates 1 query (distributed onto to p PEs). On a moderate size parallel machine where $n_A > p$, we would expect BAQ to use more time for processing queries than LAQ; LAQ to use more time than RAQ; and RAQ to use more time than FEP.

4.2 Record Selection

After algorithms LAQ, RAQ and FEP first use FastBit to select a set of records that match the filtering criteria, each must next read records from disk and then compute histogram bins. In this section, we examine issues that may affect performance for our two different partitioning strategies.

For our shared-memory implementation of the task-parallel algorithms (BAQ, RAQ and LAQ), all PEs access the *same set* of files, even though each PE reads a different set of record values. Because RAQ and LAQ read records from the same set of files, it is likely that some of the records read by different PEs will reside in the same pages on disk. Since all disk I/O operations are performed in pages internally, these pages would be read multiple times to satisfy the read operations issued from different PEs. When using more than one PE, LAQ and RAQ may require more time to read the same set of record values than algorithm FEP due to redundant disk page I/O.

Let us compare the read operations performed by LAQ and RAQ for building d-dimensional conditional histograms. The first difference is that LAQ reads the data records of only *one* variable while RAQ reads d variables. This fact suggests that LAQ could spend less time reading its data records. However, because the record values are randomly selected from data files, the reading order may have a substantial influence on the actual execution time. When reading the same variable, algorithms LAQ and RAQ could read the same record values. However, each algorithm performs these reads in a different order through a different number of queries. In general, reading these record values in a smaller number of queries will take less time than reading them a larger number of queries. In order to read the selected record values, LAQ executes a

total of n_A (or n_B) such queries while RAQ only executes p queries. In this performance study we assume $n_A > p$. Thus, we expect algorithm RAQ to use less time to the read selected record values than algorithm LAQ.

For most conditional histograms we expect that the selected records are randomly scattered throughout the dataset. In this case, algorithm RAQ would show relatively small changes as the number of PEs p increases for the following reasons: (1) Since all PEs perform the same evaluation of external conditions, this part of the query time is unlikely to reduce as the number of PEs increases. (2) Likewise, the I/O time for retrieving the records might also not change significantly due to I/O contention of simultaneous disk page accesses. In contrast, the query response time for LAQ would decrease as each PE works on fewer number of 1D histograms. On a single PE, we may see LAQ to take longer than RAQ, but as the number of PEs increases, LAQ may catch up with RAQ.

With the data parallel algorithm (FEP), there are no issues with respect to read contention or redundancy since all PEs are reading different data from different files.

4.3 Record Counting

Finally we discuss the time required for counting the number of records per bin. Remember that the total number of records to be counted for a conditional histogram is defined by the external conditions, and is the same for all algorithms. The procedure used to locate the bin for each record value is also the same. The time required for the counting functions after the variables have been read into memory is proportional to the number of records and the number of variables. LAQ, RAQ and FEP work on the same total number of records. However, LAQ only selects and counts one variable while RAQ and FEP select and count two variables in our example. We expect that RAQ and FEP would spend nearly twice as much time as LAQ on the counting functions. However, because the counting functions operate on in-memory data, the total time spent in these functions may be relatively small.

In summary, we expect algorithm BAQ to be the slowest because it performs considerable amount of redundant work. We expect algorithm RAQ to be the fastest on a single PE, and algorithm FEP to be the fastest on multiple PEs.

5 Performance Study

In this section we evaluate the performance of four parallel algorithms for building conditional histograms. All experiments were executed on a 32-processor SGI Altix with 192GB of RAM and 40TB of fiberchannel RAID. Because of its large amount of memory and substantial I/O performance rates, this platform is well suited for data intensive analysis and visualization tasks.

5.1 Data Characteristics

The data set consists of 42 weeks' worth of network connection data that was collected by Bro [Paxson 1998] at NERSC (National Energy Research Scientific Computing Center, Berkeley Lab). There are 2.5 billion records, each with 22 attributes such as source and destination IP addresses, source and destination ports, start time, duration, number of bytes sent along with additional network connection information. All the data is stored in raw binary format. This data set is more than two times the size that we used for our initial experiments for the SC05 HPC Analytics Challenge [Stockinger et al. 2005a]. In order to increase query efficiency, we have split IP addresses into their historical A, B, C and D octets. The total size of the data set is 281GB.

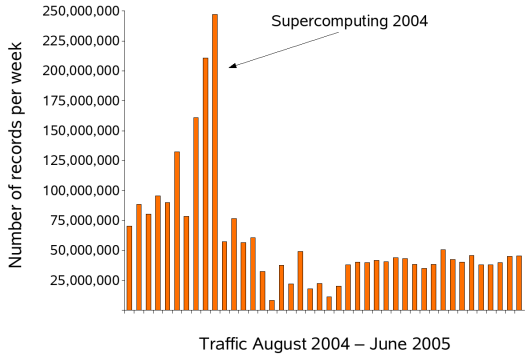


Figure 3: Network traffic data collected at NERSC.

Figure 3 shows the network traffic that was collected at NERSC between August 2004 and June 2005. We can see a significant traffic increase before Supercomputing 2004.

For each of the 22 attributes we built an equality-encoded bitmap index with WAH bitmap compression [Wu et al. 2004]. The size of the bitmap indices is 78.6GB, which is less than a third of the raw data. This size is fairly modest compared with B-tree-based indices, which are on average three to four times the size of the raw data.

For our benchmarks we experimented with two different kinds of data partitioning strategies. First, we stored the values of each attribute in a single large file. In total, we have 22 files (one file per attribute). We call this case the *contiguous data set*. Next, we divide the data into 32 partitions (one per PE), where each partition consists of about the same number of data records (*partitioned data set*).

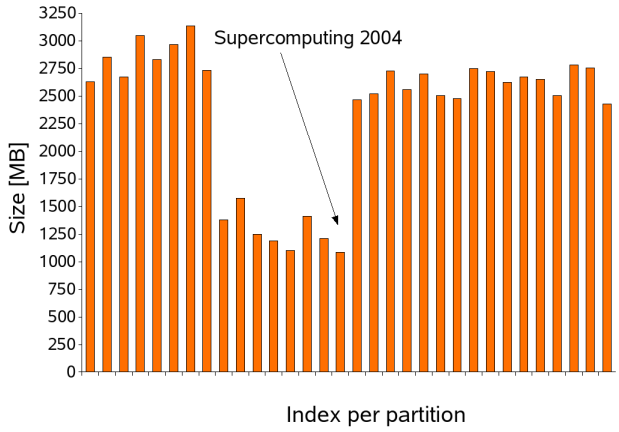


Figure 4: Size of the compressed bitmap indices for 32 partitions. Each partition contains 22 bitmap indices (one per data attribute).

Figure 4 shows the total sizes of the WAH compressed bitmap indices for the 32 partitions. Even though each data partition contains the same number of records, the sizes of the compressed bitmap indices vary. This variance is due to the fact that the access patterns in the network traffic varies. As we saw in Figure 3, there was an increase in network traffic intensity a few weeks before Supercomputing 2004. Since traffic in that week is more homogenous than the traffic during the rest of the year, the bitmap indices compress better for that part of the data. We observe that an even data partitioning strategy based on the number of records does not necessarily result in evenly sized indices.

5.2 Parallel Histogramming Performance

In our first set of experiments we evaluated a multi-dimensional query for producing 2D conditional histograms. This query is typical for studying the hosts that were contacted over a certain period of time. The query we used for our performance benchmarks is as follows:

```
Histogram 1:
plot histogram DP:1000:11000:100,
               tsyday:50:350:3
where (state == 1) and (12 <= tshour < 14)
```

This query looks for unsuccessful connections (state==1) within the time window of 12:00 and 14:00. `DP:1000:11000:100` refers to *destination port* in the range [1000;11,000) with a step size of 100. `tsyday:50:350:3` refers to the time stamp given in days with the range [50;350) and a step size of 3. As one can easily calculate from the ranges and the step sizes, the number of histogram bins for both attributes is 100.

The time to evaluate the query with our four parallel algorithms is shown in Figure 5. The graph shows the performance of these four algorithms with up to 32 PEs. This query selects about 28 million records out of 2.5 billion, corresponding to a selectivity of about 1%. Algorithm RAQ shows the best performance on one PE. For two PEs and greater, algorithm FEP is the winner. Note that algorithms BAQ, LAQ and RAQ operate on the contiguous data set whereas algorithm FEP operates on the partitioned data set.

Let us now analyze algorithm FEP in more detail. We can see that it shows nearly linear scalability. One of the reasons for not observing optimal scalability for algorithm FEP is that the sizes of the compressed bitmap indices for each data partition vary – the bitmap compression algorithm performance is strongly correlated with the data distribution statistics [Wu et al. 2004]. Since the network access patterns change over time, we expect that some parts of the data compress better than others as can be seen in Figure 4.

Next, we varied the query condition to study the impact of the number of result records on the performance of the algorithms. In particular, we varied the time window of the query in the following way:

```
Histogram 2:
plot histogram DP:1000:11000:100,
               tsyday:50:350:3
where (state == 1) and (13 <= tshour < 14)

Histogram 3:
plot histogram DP:1000:11000:100,
               tsyday:50:350:3
where (state == 1) and (11 <= tshour < 15)
```

The number of result records for Histogram 2 is 12.6 million (which is half of the number of result records of Histogram 1). The number of result records for Histogram 3 is 54.6 million (double the size of the result records of Histogram 1). The performance for building these

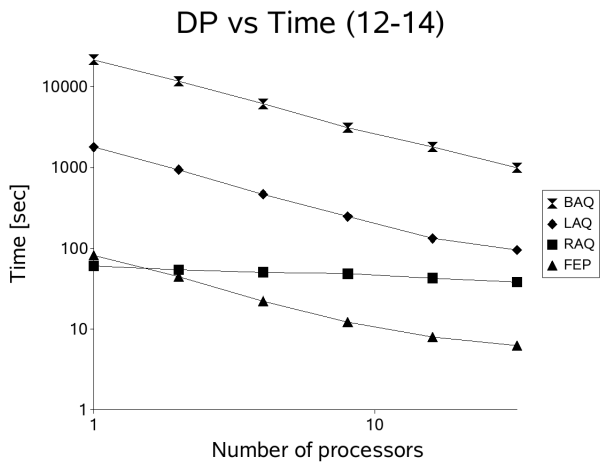
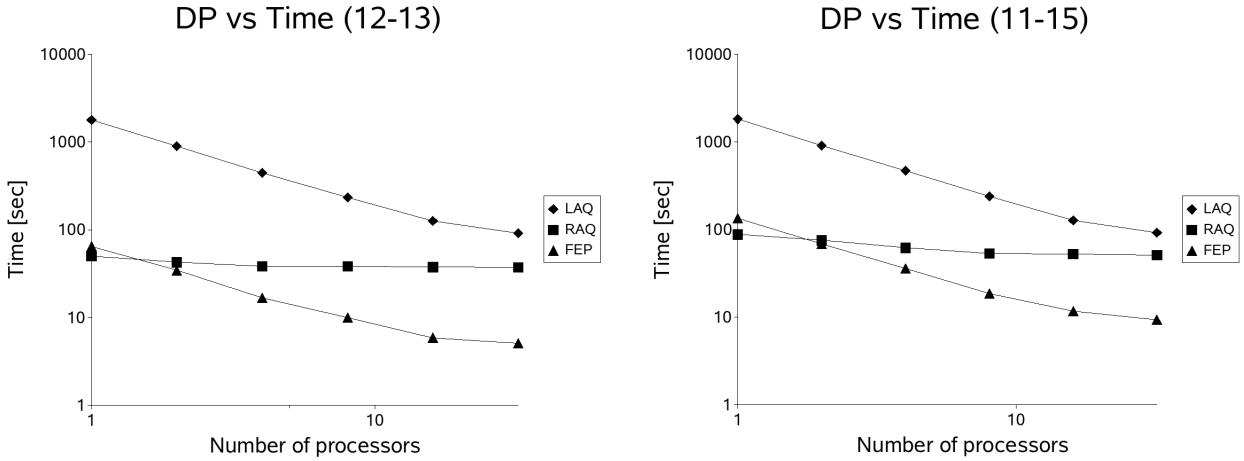


Figure 5: Processing time for conditional 2D histogram over 2.5 billion records with up to 32 PEs. The query condition is (state == 1) and ($12 \leq$ tshour < 14). Note the log-scale on both the x and y axes.

conditional histograms is shown in Figure 6. Since algorithm BAQ turned out to have the worst performance across all experiments, we only plot the results for algorithms LAQ, RAQ and FEP. We can observe that the absolute run time of the algorithms changes slightly with the number of result records. However, the relative performance of the three algorithms stays the same. Again, algorithm FEP is the overall winner.



(a) Result records: 12.6 million.

(b) Result records: 54.6 million

Figure 6: Processing time for conditional 2D histogram over 2.5 billion records for up to 32 PEs with various numbers of result records. Note the log-scale on both the x and y axes.

To determine if FEP's "tail off" in parallel performance with increasing numbers of PEs is due to I/O limits, we next conduct an experiment comparing FEP with a sequential data scan. If the sequential scan becomes I/O bound, its relative performance improvement will also begin to degrade with increasing parallelism. The performance results of the sequential scan and FEP are shown in Figure 7. For the sequential scan, we read the four attributes that are contained in Histogram 1, namely *DP*, *tsyday*, *tshour* and *state*. Each attribute is stored in a separate file with 32 equally sized partitions. We observe that the parallel sequential scan performance indeed scales linearly, but algorithm FEP shows a slight decrease in performance improvement with increasing parallelism. This result indicates that the sequential scan has not become I/O limited at 32-way parallelism since its relative performance improvement does not degrade. We conclude that FEP, which is performing less I/O than a sequential scan, is not I/O bound at this level of parallelism on our experiment's computing platform. We see that the query response time of algorithm FEP is, on average, a factor of 10 faster than the sequential scan.

Finally, we compared the performance of algorithm FEP with ROOT (developed at CERN) and ROOT-FastBit [Stockinger et al. 2006]. We have chosen ROOT since it is currently one of the most powerful database systems used in production for scientific data management analysis. For instance, that Babar [Experiment 2006] experiment at the Stanford Linear Accelerator Center stores nearly up to one Petabyte of data

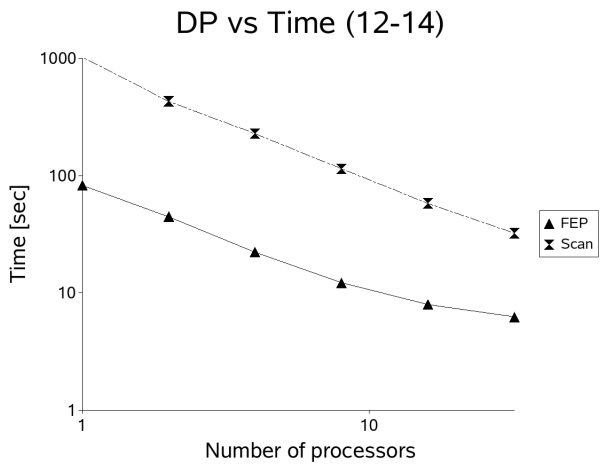


Figure 7: Processing time for conditional 2D histogram over 2.5 billion records for up to 32 PEs. Note the log-scale on both the x and y axes.

in ROOT. Recently we also integrated FastBit into ROOT so that we can perform a direct comparison of the conditional histogramming capabilities.

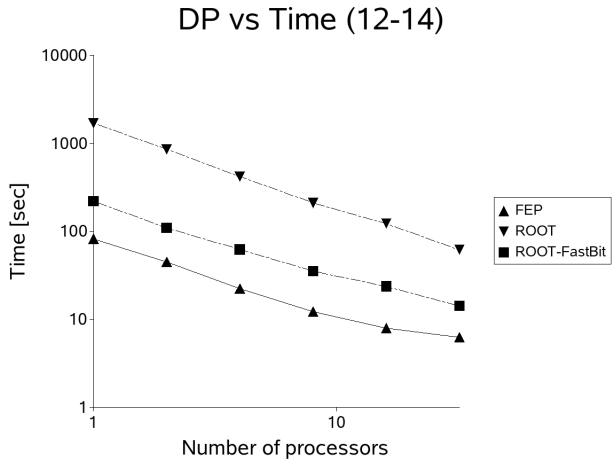


Figure 8: Processing time for conditional 2D histogram over 2.5 billion records for up to 32 PEs. Note the log-scale on both the x and y axes.

Figure 8 shows the performance of two different kinds of histogramming algorithms in ROOT. One uses ROOT's internal projection indices [O'Neil and Quass 1997], the other one uses FastBit indices. We observe that ROOT has nearly optimal scalability for up to 32 PEs in this study. ROOT-FastBit and algorithm FEP both scale well up to 16 PEs with a marginal performance penalty for 32 PEs. However, algorithm FEP is, on average, a factor of 3 faster than ROOT-FastBit.

6 Network Connection Analysis Case Study

It is common for attackers to use sets of previously compromised hosts to collectively scan a target network. This form of attack, known as a "distributed scan," is typically accomplished by dividing the target address space up amongst a group of 'zombie' machines and directing each zombie to each scan its portion of the target network. The term 'zombie' refers to a computer system that has been enslaved for the purpose of carrying out some action, typically malicious. The scan results from each zombie are then aggregated at a master host thereby creating a complete picture for the attacker. An example attack is a search for unsecured network services on a given port or port range. Identifying sets of hostile hosts under common control is helpful in that the group of hosts can be blocked from access to critical infrastructure, and the group can be reported to the larger community for further analysis or action.

The initial indication of a security incident typically comes from a source other than data mining, e.g. an email alert, an IDS alert, or a phone call from a colleague at another institution. For example, we might receive an email indicating that a new vulnerability has been publicized, affecting a network service that listens on a particular TCP port. We can then examine historical data to see if there is evidence of malicious activity targeting this port – this activity could well be an indication of an attacker with pre-publication knowledge of the vulnerability.

In the discussion that follows, we implement a sequence of data mining steps aimed at detecting a distributed scan and identifying the set of

remote hosts participating in the scan. The target network is of "class B" size, meaning it contains 65536 unique IP addresses covering the full range of [0;255] in both the C and D octets (the target network in this case is identified by its unique values in the A and B octets). We use a custom visual analytics application that combines a GUI for constructing complex multi-dimensional queries, FastBit for answering the queries, and OpenRM Scene Graph [R3vis 1999-2006] for performing visualization and rendering. In our analysis, we first determine the temporal characteristics of the distributed scan, then identify the set of remote hosts participating in the scan.

6.1 Understanding Temporal Characteristics

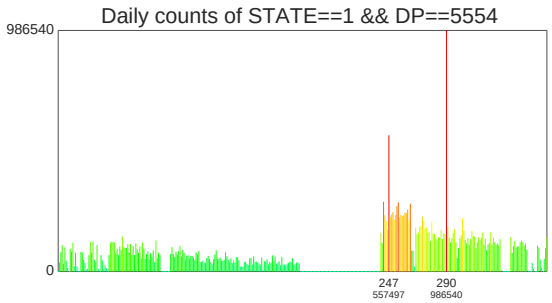


Figure 9: Unsuccessful connection attempts to port 5554 over a 42-week period are shown here in histogram form. While the source data is sampled at per-second resolution, we created this histogram by posing a query requesting the number of connection attempts over a 24-hour time window: each histogram bin is one day wide. We see "similar behavior" in the region around day 247, where "similar behavior" in this case is characterized by the number of connection attempts. Each histogram bar is color-coded according to the number of standard deviations from the mean. Green bars are "close to" the mean number of connections, while red bars indicate bins having count values that are about three standard deviations from the mean number of counts.

Figure 9 is a histogram showing daily counts of unsuccessful incoming TCP connections directed at port 5554 over a 365 day period. The data set contains 42 weeks of data – the "missing" 10 weeks are visible as a gap in the middle of the histogram. Note the large counts on days 247 and 290 visible as tall red spikes. Note also that around day 247 there is a consistent increase in daily activity, as evidenced by similar levels of daily connection counts. The activity on day 290 appears different, since there appears to be a large increase in scanning activity – possibly indicating a new scanning tool or technique. The activity surrounding day 247 appears at first glance to be careful work over time by a set of hosts, whereas the day 290 is likely a single host scanning at a very high rate with the intent to 'get in and get out' before its presence can be detected and its access blocked. Such high speed scans are quite common, and are often the reconnaissance phase of a larger attack mechanism, or a combined scan and attack tool. We choose to focus on the 4-week period around day 247, where there is evidence of temporally regular and systematic suspicious activity as shown in Figure 10. In contrast, Figure 11 shows a histogram of suspicious activity sampled at one-hour intervals over a two-day period around day 290. The access patterns visible in Figure 10 are much different than those in Figure 11.

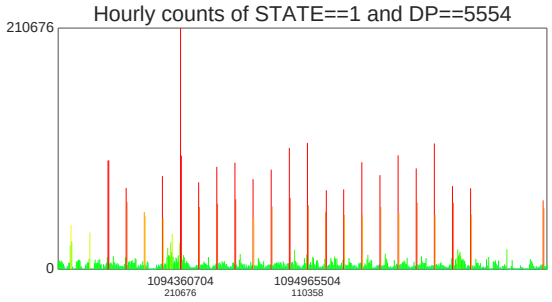


Figure 10: This histogram shows the per-hour number of unsuccessful connection attempts over a four-week period around day 247.

Figure 10 shows the four-week period surrounding day 247 with finer granularity in time – this histogram shows connection counts at hourly resolution. We can see that the activity is temporally periodic, with a period of approximately one day, repeating for twenty-one days. We will therefore direct our analysis toward the activity over this 21-day period.

We drill into the data at a finer temporal resolution; by posing a query that requests counts of per-minute unsuccessful connection attempts over a five-day period within the four-week window. The histogram, which is depicted in Figure 12, shows a distinct pattern of increased activity on a precise 24-hour interval. This image shows a spike in activity occurring at about 21:15:00 local time each day. Each such primary spike is followed by a secondary spike that occurs about 50 minutes later. Each of the top five spikes is labeled on the bottom of the figure with the raw Unix timestamp and the number of connection attempts at that time.

At this point, we conclude that an organized scanning event is occurring daily at 21:15 local time within a four-week window of our 42-week dataset. While there also appears to be a secondary event occurring about 50 minutes later within the same four-week period, for the rest

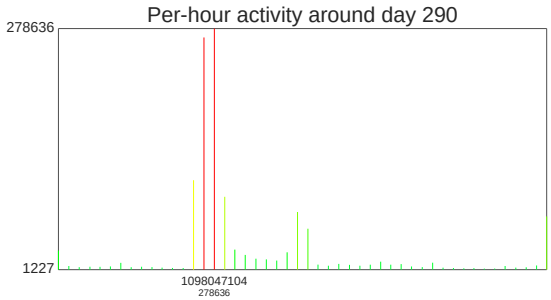


Figure 11: This histogram shows the per-hour number of unsuccessful connection attempts over a two-day period around day 290.

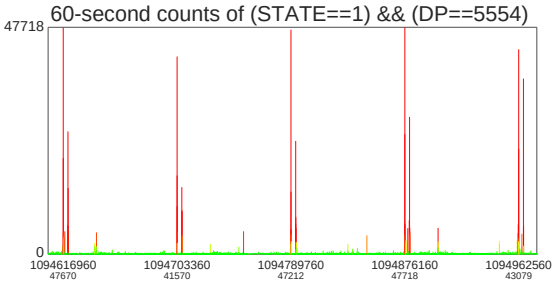


Figure 12: This histogram shows the number of unsuccessful connection attempts to port 5554 over a five-day period of time sampled over one-minute temporal windows. The histogram indicates a repeating pattern of a spike in unsuccessful connection attempts that occurs on a twenty-four hour interval. Each spike is followed by a secondary, smaller spike fifty minutes later.

of this analysis we will focus only on the event occurring at 21:15 in order to simplify the method that we are presenting. We refer to the seven-minute window during which the daily scanning event occurs as the "event horizon."

So far, our analysis has focused on understanding the temporal characteristics of the attack. Next, we take a different approach in looking at the coverage of the target network by the attack. As shown in Figure 13 we create a 3D histogram where two of the axes are the C and D octets of the destination address space and the third axis is time. The range of the IPR_C and IPR_D axes is $[0;255]$, which corresponds to the complete range of addresses in the target network. The range of the time axis, labeled ts, is a two-hour period within the larger time window of interest sampled at a one-minute granularity. The resulting histogram has 256 by 256 by 120 "bins" that contain the number of connection attempts on port 5554 for each destination (C,D) address within a one-minute time window. Each point in Figure 13's 3D scatterplot is color-coded by the number of connection attempts at each destination address. Points in blue indicate a single connection attempt; points in green represent two connection attempts and indicate an overlap in target address space amongst the attacking hosts. Outside of the "sheet-like" structures, we see an occasional scan through a sequence of D octet values.

In Figure 13, we see coarse structures that approximate "sheets." Each sheet represents a nearly complete scan of the target network, and each corresponds to some pair of large spikes visible in Figure 12. Were we to broaden the time window in Figure 13 to include the entire time range of Figure 12, we would see the "sheet" structure repeat over and over again. We focused on a specific two-hour window to look at the fine-level time/address structure that might not be visible with a coarser temporal view. For the sake of completeness, Figure 14 shows a 1D histogram over the same two-hour window sampled at one-minute intervals. The "spikes" in Figure 14 correspond to the temporal location of the "sheet-like" structures in Figure 13.

Drilling into the data at a finer resolution, Figure 15 shows a histogram of unsuccessful connection attempts over a seven-minute window at one-second temporal resolution. At this fine temporal resolution – the maximum possible with this particular dataset – we see the sharp spikes in the one-minute resolution histogram from Figure 12 is a distribution that spans approximately two minutes from about 21:14 to 21:16 PST.

6.2 Discovering Attacking Host IP Addresses

The next step in the analysis is to determine the source addresses of the attacking hosts. We will use an iterative approach in which we first identify the A octet of an attacking host's address followed by the B, C and D octets.

We construct a complex multi-dimensional query that aims to answer the following question: over the event horizon, how many hosts for each value in the A octet attempt to connect to all destination C octets during each scanning event? Each call to the 2D histogram query function asks for the number of unsuccessful connection attempts from each source A octet to each destination C octet within a one-second time window. We perform one such query for each of the 420 seconds within the event horizon. The results of these queries are presented in Figure 16.

Figure 16 contains two different views of a 3D histogram along with a related 1D histogram. The 1D histogram shows the number of

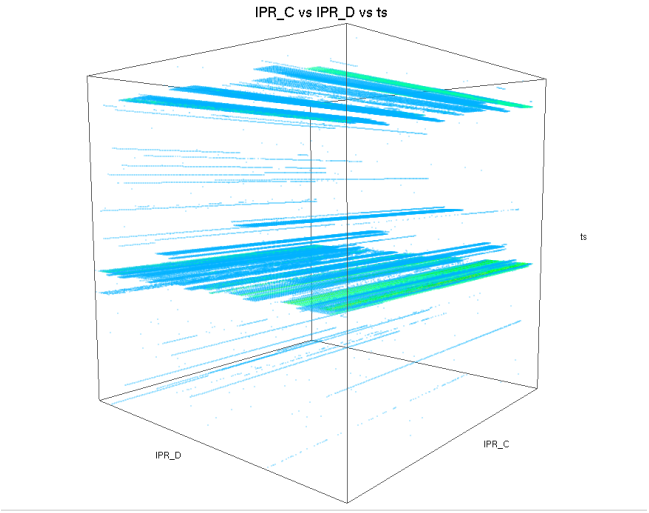


Figure 13: Focusing on a two-hour window using a one-minute temporal resolution, we generate a histogram showing the number of connection attempts across all (C,D) addresses of the target address space. Two of the axes are all addresses within each of the C and D octets of the target address space. The third axis is time over a two-hour window sampled over one-minute windows. The "sheet-like" structures correspond to a pair of temporally close spikes in Figure 12.

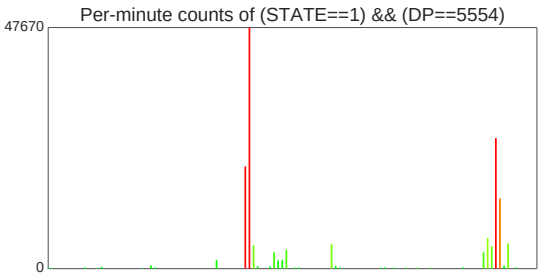


Figure 14: A 1D histogram of connection attempts over a two-hour time period at a one-minute temporal resolution. The spikes in this image correspond to the temporal location of the "sheets" visible in Figure 13.

unsuccessful connection attempts from all A octets for each daily event time horizon over the total event time horizon. In the 3D histogram views, one axis is the destination C octet space, another is the source A octet space. Time in both views corresponds to the vertical axis with earlier time being at the bottom of the histogram image. We draw several conclusions from Figure 16. First, it appears that most of the organized activity is originating from within three specific A octets. Interestingly, there are three highlighted spikes visible in the 1D histogram of Figure 16 corresponding to high levels of suspicious traffic across all A-octet addresses. Second, we characterize "organized" traffic in this image as a systematic progression through the destination C octet space over time. The top left panel of Figure 16 clearly shows slanted line segments where the destination C octet changes as a function of time. Based upon these images as well as other output from the visual analytics application (not shown here), we conclude that most of the attacking traffic originates from the A octet 220, but there are also significant levels of traffic from the A octets 60, 61, 211, 218, 221 and 222 that exhibit correlating characteristics, namely time and duration of activity.

We next perform an iterative sequence of steps that converge on completely identifying the full address of all attacking hosts. Whereas the query producing the data for Figure 16 aims to identify the source A octets, the next three steps in the analysis each identify the B, C and D octets of the complete IP address of all attacking hosts. Each step uses information from the previous step so that by the time we mine data for the source D octets, we are using results from previous steps that identified the A, B, and C octets. At the end of these steps, we identified a total of twenty IP addresses that appear to be part of a distributed scan.

Using those twenty IP addresses, we next compute a "coverage map" showing the destination C and D octets for each zombie host. The result is Figure 17.

7 Conclusions

This paper emphasizes the importance of a multi-disciplinary approach in large-scale analytics. By combining state-of-the-art scientific data management technology – efficient compressed bitmap indices – with simple yet effective visualization in a visual analytics and data mining

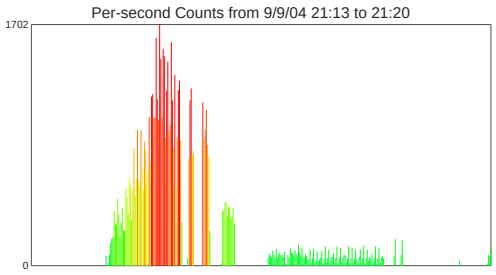


Figure 15: By increasing the temporal resolution, we see in this histogram having per-second resolution over a seven-minute window, the sharp spikes in previous images reflect activity that occurs over about a two-minute period.

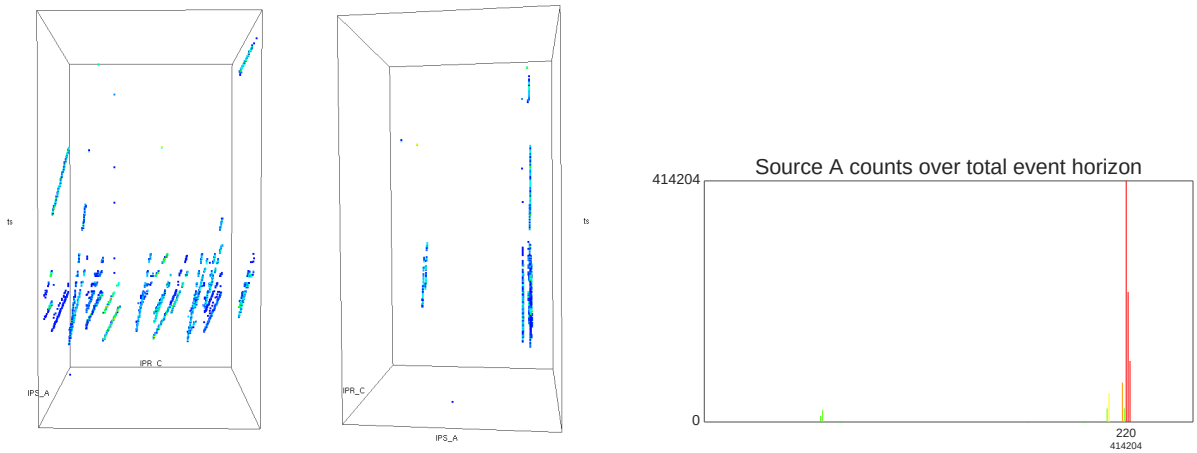


Figure 16: The top two panels are different views of the same 3D histogram showing the number of connection attempts to destination C octets from source A octets over a seven-minute window at a one-second granularity for a periodic event occurring on 9/9/04. "Organized" behavior is visible in the top left panel as line segments that systematically probe destination C octets. The start of the scanning event is visible as the sudden increase in activity in the bottom one-third of the top-left image. The bottom 1D histogram shows the number of connection attempts from all source A octets regardless of the destination C octet.

application, we demonstrate using the combination to detect distributed scans in 42-weeks worth of network connection data. Since the visual analytics application focuses on generating and displaying multi-dimensional conditional histograms, this paper includes a discussion of the performance characteristics of a new family of algorithms for quickly producing one- and two-dimensional histograms from bitmap indices. While the context of our presentation is on a cybersecurity application, the techniques and technology are sufficiently general to be applicable to a wide range of data analysis and mining topics in science, engineering, medicine, business and security.

Acknowledgement

This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

AMSDEN, P., AMWEG, J., CALATO, P., BENSLEY, S., AND LYONS, G., 1997. Cabletron's lightweight flow admission protocol specification, version 1.0. IETF RFC 4124, http://www.ietf.org/rfc/rfc2124.txt.

BELLMAN, R. 1961. *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

BENTLEY, J. 1975. Multidimensional binary search trees used for associative search. *Communications of the ACM 18*, 9, 509–516.

BERCHTOLD, S., JAGADISH, H. V., AND ROSS, K. A. 1998. Independence diagrams: A technique for visual data mining. In *Proc. 4th Int. Conf. Knowledge Discovery and Data Mining, KDD*, AAAI Press, R. Agrawal, P. E. Stolorz, and G. Piatetsky-Shapiro, Eds., 139–143.

BETHEL, E. W., CAMPBELL, S., DART, E., STOCKINGER, K., AND WU, K. 2006. Accelerating network traffic analysis using query-driven visualization. In *IEEE Symposium on Visual Analytics Science and Technology*, IEEE Computer Society Press.

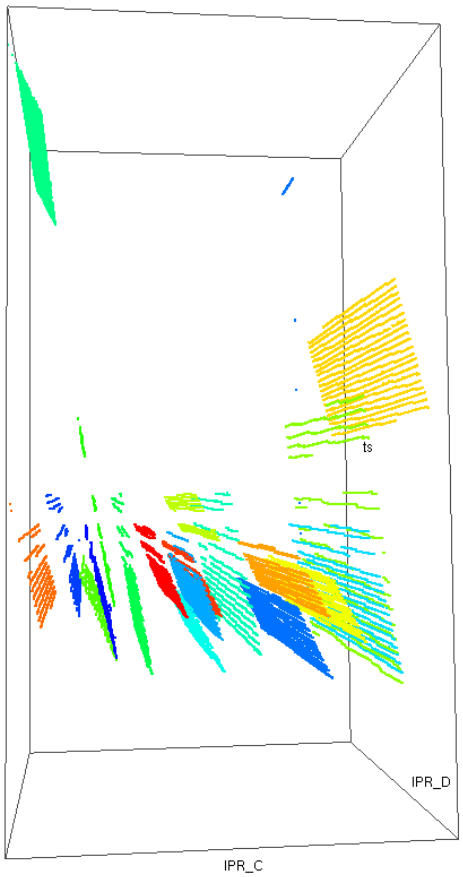


Figure 17: Here we see the destination C and D octets contacted by each of the twenty hosts participating in the distributed network scan in one of the periodic scanning events.

BRUN, R., AND RADEMARKERS, F. 1997. Root – an object oriented data analysis framework. In *Proceedings of the AIHENP 1996 Workshop*, 81–86.

BURRESCIA, J., AND JOHNSTON, W., 2005. Esnet status update. Internet2 International Meeting.

CHAN, C.-Y., AND IOANNIDIS, Y. E. 1998. Bitmap index design and evaluation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*.

COX, K. C., EICK, S. G., AND HE, T. 1996. 3D Geographic Network Displays. *SIGMOD Rec. 25*, 4, 50–54.

EXPERIMENT, B., 2006. The babar experiment. http://www-public.slac.stanford.edu/babar/.

FISK, M., AND VERGHESE, G. 2002. Agile and scalable analysis of network events. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, 285–290.

FISK, M., SMITH, S. A., WEBER, P., KOTHAPALLY, S., AND CAUDELL, T. 2003. Immersive Network Monitoring. In *Proceedings of the 2003 Passive and Active Measurement Workshop*.

FULLMER, M., AND ROMIG, S. 2000. The OSU Flow-tools package and Cisco Netflow Logs. In *Proceedings of the 14th Systems Administrator Conference (LISA 2000)*, 291–303.

GATES, C., COLLINS, M., DUGGAN, M., KOMPANEK, A., AND THOMAS, M. 2004. More NetFlow Tools: For Performance and Security. In *Proceedings of the USENIX18th Systems Administration Conference (LISA 2004)*, 121–131.

GOODALL, J., LUTTERS, W., RHEINGANS, P., AND KOMLODI, A. 2005. Preserving the Big Picture: Visual Network Traffic Analysis with TNV. In *Proceedings of the 2005 Workshop on Visualization for Computer Security*, 47–54.

GRINSTEIN, G., KEIM, D., AND WARD, M., 2002. Information visualization, visual data mining, and its application to drug design. IEEE Visualization 2002 Course #1 Notes, October.

HOCHHEISER, H., AND SHNEIDERMAN, B. 2001. Visual specification of queries for finding patterns in time-series data. In *Proceedings of Discovery Science*.

IOANNIDIS, Y. 2003. The history of histograms (abridged). In *International Conference on Very Large Data Bases*.

JACOBSEN, V., LERES, C., AND MCCANNE, S., 1989. tcpdump. ftp://ftp.ee.lbl.gov/.

JOHNSON, T. 1999. Performance measurements of compressed bitmap indices. In *International Conference on Very Large Data Bases*.

KEIM, D., AND KRIEGEL, H.-P. 1994. Visdb: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications 14*, 4, 40–49.

KINDLMANN, G. 1999. *Semi-Automatic Generation of Transfer Functions for Direct Volume Rendering*. Master's thesis, Cornell University.

KITWARE, INC. 2003. *The Visualization Toolkit User's Guide*, January.

KNUTH, D. 1998. *The Art of Computer Programming, 2nd Ed., Volume 3*. Addison-Wesley.

KOMLODI, A., RHEINGANS, P., AYACHIT, U., GOODALL, J., AND JOSHI, A. 2005. A user-centered look at glyph-based security visualization. In *Proceedings of the 2005 Workshop on Visualization for Computer Security*.

KORNEXL, S., PAXSON, V., DREGER, H., FELDMANN, A., AND SOMMER, R. 2005. Building a time machine for efficient recording and retrieval of high-volume network traffic. In *Internet Measurement Conference*.

KOUTSOFIOS, E. E., NORTH, S. C., TRUSCOTT, R., AND KEIM, D. A. 1999. Visualizing large-scale telecommunication networks and services (case study). In *VIS '99: Proceedings of the conference on Visualization '99*, IEEE Computer Society Press, Los Alamitos, CA, USA, 457–461.

LAKKARAJU, K., YURCIK, W., AND LEE, A. 2004. NVisionIP: NetFlow Visualizations of System State for Security Situational Awareness. In *Internet Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC-2004)*.

LAU, S. 2004. The spinning cube of potential doom. *Communications of the ACM 47*, 6, 25–26.

LEVOY, M. 1989. *Display of Surfaces from Volume Data*. PhD thesis, University of North Carolina at Chapel Hill.

LIVNAT, Y., AGUTTER, J., MOON, S., ERBACHER, R., AND FORESTI, S. 2005. A visual paradigm for network intrusion detection. In *IEEE Workshop on Information Assurance And Security*.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21.

MAX, N. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2 (June).

MCCANNE, S., LERES, C., AND JACOBSEN, V., 1994. libpcap. ftp://ftp.ee.lbl.gov/.

MCCORMICK, P., INMAN, J., AHRENS, J., HANSEN, C., AND ROTH, G. 2004. Scout: A hardware-accelerated system for quantitatively driven visualization and analysis. In *Proceedings of IEEE Visualization*, 171–178.

MCPHERSON, J., MA, K.-L., KRYSTOSEK, P., BARTOLETTI, T., AND CHRISTENSEN, M. 2004. Portvis: A tool for port-based detection of security events. In *Proceedings of CCS Workshop on Visualization and Data Mining for Computer Security, ACM Conference on Computer and Communication Security*.

NIELSON, G. M., AND HAMANN, B. 1991. The asymptotic decider: Removing the ambiguity in marching cubes. In *Proceedings of IEEE Visualization*.

OETIKER, T., 2006. Multi router traffic grapher. http://mrtg.hdl.com/.

OETIKER, T., 2006. Round robin database tool. http://oss.oetiker.ch/rrdtool/.

O'NEIL, P., AND QUASS, D. 1997. Improved query performance with variant indices. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, ACM Press.

O'NEIL, P. 1987. Model 204 architecture and performance. In *Second International Workshop in High Performance Transaction Systems*, Springer Verlag.

PAXSON, V. 1998. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*.

PHAAL, P., PANCHEN, S., AND MCKEE, N., 2001. Inmon corporation's sflow: A method for monitoring traffic in switched and routed networks. IETF RFC 3176, http://www.app.sietf.org/rfc/rfc3176.html.

PLONKA, D. 2000. FlowScan: A Network Traffic Flow Reporting and Visualization Tool. In *Proceedings of the 14th Systems Administrator Conference (LISA 2000)*, 305–317.

PRODUCTS, E. S., 2006. The fast light toolkit. http://www.fltk.org.

R3VIS, 1999-2006. OpenRM Scene Graph. http://www.openrm.org.

SCIENTIFIC DATA MANAGEMENT GROUP, L. B. N. L., 2006. Fastbit. http://sdm.lbl.gov/fastbit.

SHOSHANI, A., BERNARDO, L., NORDBERG, H., ROTEM, D., AND SIM, A. 1999. Multidimensional indexing and query coordination for tertiary storage management. In *International Conference on Scientific and Statistical Database Management*, IEEE Computer Society.

1998. Proceedings of the 1998 ACM SIGMOD: International Conference on Management of Data, ACM Press, New York, NY, USA.

STOCKINGER, K., DUELLMANN, D., HOSCHEK, W., AND SCHIKUTA, E. 2000. Improving the performance of high-energy physics analysis through bitmap indices. In *Proceedings of the 11th International Conference on Database and Expert Systems Applications*, Springer Verlag.

STOCKINGER, K., WU, K., CAMPBELL, S., LAU, S., M, F., GAVRILOV, E., KENT, A., DAVIS, C. E., OLINGER, R., YOUNG, R., PREWETT, J., WEBER, P., CAUDELL, T. P., BETHEL, E. W., AND SMITH, S. 2005. Network traffic analysis with query driven visualization - sc 2005 hpc analytics results. In *SC05, HPC Analytics Challenge*, ACM Press.

STOCKINGER, K., SHALF, J., WU, K., AND BETHEL, E. W. 2005. Query-driven visaulization of large data sets. In *Proceedings of IEEE Visualization*.

STOCKINGER, K., WU, K., BRUN, R., AND CANAL, P. 2006. Bitmap indices for fast end-user physics analysis in root. *Nuclear Instruments and Methods in Physics Research, Section A – Accelerators, Spectrometers, Detectors and Associated Equipment 559*, 99–102.

SYSTEMS, C., 2005. Cisco netflow collection engine. http://www.cisco.com/en/US/products/sw/netmgtsw/ ps1964/.

THOMAS, J. J., AND EDS., K. A. C. 2005. *Illuminating the Path – The Research and Development Agenda for Visual Analytics*. IEEE Computer Society Press.

UPHOFF, B., AND CRISCUOLO, P. 2004. A framework for collection and management of intrusion detection data sets. In *Proceedings of the 16th Annual FIRST Conference on Computer Security Incident Handling*.

WARE, C. 2004. *Information Visualization: Perception for Design*, second ed. Morgan Kaufmann Publishers.

WU, K., OTOO, E., AND SHOSHANI, A. 2001. A performance comparison of bitmap indices. In *Proceedings of the ACM CIKM International Conference on Information and Knowledge Management*, ACM Press.

WU, K., OTOO, E., AND SHOSHANI, A. 2004. On the performance of bitmap indices for high cardinality attributes. In *Proceedings of the International Conference on Very Large Data Bases*.

WU, K., OTOO, E. J., AND SHOSHANI, A. 2006. An Efficient Compression Scheme For Bitmap Indices. *ACM Transactions on Database Systems 31*, 1–38.

YIN, X., YURCIK, W., TREASTER, M., LI, Y., AND LAKKARAJU, K. 2004. VisFlowConnect: NetFlow Visualizations of Link Relations for Security Situational Awareness. In *Internet Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC-2004)*.