Global Information Assurance Certification Paper

Tech Refresh for the Forensic Analysis Toolkit

GIAC GCFA Gold Certification

Author: Derek Edwards, derekedw@yahoo.com
Advisor: Richard Carbone
Accepted: March 2, 2015

Abstract

The most widely used commercial forensic tools have not undergone major architectural change since their market introduction in the late 1990s. Meanwhile, architectural change elsewhere has brought fast, powerful and inexpensive search, data visualization, and collaboration capabilities to users of all ages and computing experience levels. If the Internet is being indexed for search, could not forensic images be likewise indexed also? Could there potentially be relief from image size limits and storage barriers? Could forensic analysis be performed faster? What are the risks? “Big data” open-source tools like Apache Hadoop, Apache HBase and Apache Spark were used to develop a new architectural foundation proof of concept for digital forensics. While this framework did not improve performance on tasks that require serial processing, like hashing images for verification hashes, it has shown improved performance on a basic parsing task – finding ASCII strings.

1. Introduction

Many have written about the "digital forensics crisis" caused by growing caseloads and storage device sizes. (Garfinkel, 2010) Attempts to address these issues have focused primarily on triage or data reduction. While triage prioritizes forensic tasks so that the analyst is presented the most important artifacts or products first, data reduction eliminates items of low relevance to the case, such as "known-good" operating system and application files. (Casey, 2011)

While the digital forensics community seems to agree that more images in more formats each month are being submitted for analysis, few efforts discuss managing the increased "volume, variety and velocity" (Laney, 2001) of incoming data as a "big data" issue. Laney focused on the growth of transactional data that came from e-commerce. Some of the same data management strategies and tools that facilitate the analysis of e-commerce data may apply to digital forensics.

In digital forensics, a large amount of computer time and effort go towards parsing. Tools like Snort, WireShark and *tcpdump* are used to parse packet data from packet structures, various tools and scripts to parse expected text structures from log files, and a variety of other tools to parse disk and RAM artifacts from lots of other structures like partition tables, boot sectors, file allocation tables (FAT), master file tables (MFT), etc.

Most parsers process the structures serially (one at a time). Proof is found in the number of cores kept busy on the forensic workstation. Most often, the number is the same as the number of parse jobs initiated. Most parsers have this style (in pseudocode):

```
while(input.hasNext()) {
        x = input.next()
        output.add(parse(x))
}
return output
```

The larger the input, the larger the wait time for output.

A large amount of analyst time goes toward selecting what to parse because there is insufficient time to wait for the output or space to store it.

Author Name, email@address

2. Related Work

2.1. BinaryPig: Scalable Static Binary Analysis over Hadoop

Endgame Inc. introduced BinaryPig at a talk at BlackHat USA 2013. (Endgame Inc., 2013) About their data management challenge, they wrote:

> "Over the past three years, Endgame received 40 million samples of malware equating to roughly 19TB of binary data." (Cloudera, Inc., 2013)

Issues they were experiencing include running out of storage space and losing malware samples due to node failures. They also sought a better way of running analysis modules on all samples and sharing results.

Endgame's solution was the open source BinaryPig framework. Endgame set up an Apache Hadoop cluster to organize their samples so that their feature extraction modules could analyze them more efficiently. If a new module were developed, it could be run on all samples in short order. (Endgame Inc., 2013)

BinaryPig is available on GitHub: https://github.com/endgameinc/binarypig.

2.2. The Sleuth Kit Hadoop Framework

The Sleuth Kit Hadoop Framework is a prototype digital forensic analysis framework whose development was funded by the US Army Intelligence Center of Excellence (USAICoE). It seems to be a capable project, but it lacks installation scripts and instructions. The project's last commit occurred in 2012.

The project's code can be found on GitHub https://github.com/sleuthkit/hadoop_framework.

3. A New Architectural Foundation for Digital Forensics

Since digital forensic workload is outgrowing the forensic workstation, what capabilities would any alternative need?

Author Name, email@address

Table 1: Current vs. future platform for digital forensics

	Current	Alternative
Forensic tool	Mostly serial	Mostly parallel
CPU	Single node	Multiple redundant nodes
Memory	Single node	Multiple redundant nodes
Storage	Drives, RAID, Network Storage	Multiple redundant nodes
Failure recovery	Usually manual	Mostly automatic

The digital forensic workstation is limited to the CPU and memory resources of a single node. Its storage is local, attached to an array, or available over the network. Most of its tools parse information serially. RAID prevents data loss in case of hardware failure, but disasters other than loss of the site may require restoration from backups.

By contrast, the alternative system is a multi-node system. There is redundancy in its CPU, memory and storage resources. Its tools generally work in parallel across the nodes. Its software handles most failures other than loss of the site automatically.

3.1. Parallel Processing Concepts

MapReduce has proven its worth as a programming model for parallel processing since Google engineers Dean and Ghemawat published *MapReduce: Simplified Data Processing on Large Clusters*. (Dean & Ghemawat, 2004) Google, now known as Alphabet, Inc. earned $16.3 billion in net income for the year ending December 31, 2015. (Alphabet Inc. and Google Inc., 2015)

3.1.1. The *map* Method

In a number of programming languages, a collection is a group of elements, like an array or a set. Scala's *map* method, for example, applies a function to each element of the collection. The map function then returns a new collection containing the result of the function on each element. (Odersky & Spoon, 2010)

To illustrate this, define *upper*, which takes a string and changes its case to upper case.

Author Name, email@address

```
scala> def upper(s : String) : String = {
    | s.toUpperCase
    | }
upper: (s: String)String
```

Now, define the "items" collection, an array of strings, to which the *upper* function will be applied.

```
scala> val items = Array("One","Two","Three")
items: Array[String] = Array(One, Two, Three)

scala> items.map(upper)
res0: Array[String] = Array(ONE, TWO, THREE)

scala> items
res1: Array[String] = Array(One, Two, Three)
```

Therefore, the *map* function returns the array of items processed through the *upper* function. There is no requirement for looping or iteration in the code to apply the function to the elements.

Now, using some syntactic sugar (called a closure) to simplify coding, another simple *map* function changes the case of each element of the "items" array to lower case.

```
scala> items.map(s => s.toLowerCase)
res3: Array[String] = Array(one, two, three)
```

3.1.2. The *reduce* Method

Scala's *reduce* method processes the collection's elements through a function that takes two elements at a time.

For example, to calculate the total lengths of the words:

```
scala> items
res1: Array[String] = Array(One, Two, Three)

scala> items.map(s => s.length)
res26: Array[Int] = Array(3, 3, 5)
```

Now, to add the lengths of the elements together, use addition as a *reduce* function with two of the elements at a time, like so:

```
scala> items.map(s => s.length).reduce((a,b) => a + b)
res27: Int = 11
```

So the *reduce* function added (3 + 3) + 5, or 3 + (3 + 5), or 5 + (3 + 3) (there is no way to tell or predict) but the correct answer was returned!

Do not assume that "adjoining" elements will be sent to a reduce function, or that elements will be considered in a given order, or the results will surprise or disappoint. A *reduce* function must be *commutative* and *associative*, or else the result may be unexpected.

A function is *commutative* if the order of the operands does not matter. Addition, for example, is a commutative function because: 3 + 5 = 5 + 3 = 8; on the other hand, subtraction is not a commutative function because: 3 – 5 = -2, which is not equal to 5 – 3 = +2.

Likewise, a function is *associative* if any pair of elements could be chosen to be processed first. For example, addition is also an associative function because: (1 + 2) + 3 = 1 + (2 + 3) = 6.

3.1.3. Example Using the Scala Programming Language

Returning to parsing in digital forensics, most parsers have this style (in pseudocode):

```
while(input.hasNext()) {
        x = input.next()
        output.add(parse(x))
}
return output
```

Provided *parse* is a good candidate for a map function and x represents an individual record structure to parse, the new parser could have this style instead:

```
output = input.map(x => parse(x))
```

Since the execution environment controls invocation of the *parse* method, not the programmer, this style lends itself to parallelization. Different *parse* methods could parse a record structure like a log file row, a File Allocation Table (FAT) entry, an NTFS

Master File Table (MFT) file record or an Internet cache entry in parallel. Multiple cores or even multiple machines could collaborate on the workload.

3.2. Parallel Processing Tools

3.2.1. Apache™ Hadoop®

Doug Cutting and Mike Cafarella developed Apache Hadoop to improve their Nutch web search engine. It worked fine. However, it would not scale beyond a handful of machines and it could not operate without constant monitoring. Inspired by two Google research papers, *The Google File System* (Ghemawat, Gobioff, & Leung, 2003) and *MapReduce: Simplified Data Processing on Large Clusters*, (Chang, Dean, Ghemawat, Hsieh, & Wallach, 2006) they created the Hadoop MapReduce framework and the Hadoop Distributed File System and ported Nutch atop it. Cutting's son named his toy elephant "Hadoop," which is the origin of the name. Powered by Hadoop, Nutch ran with much less human intervention on 20-40 surplus machines. (Harris, 2013)

According to the Apache Software Foundation (ASF), Hadoop software is:

> "The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation, and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly available service on top of a cluster of computers, each of which may be prone to failures." (The Apache Software Foundation, 2016)

In Hadoop, data is dealt redundantly across the cluster of machines for availability and fault tolerance. Hadoop executes tasks in parallel and sends tasks, as close to the data as CPU, RAM, and network resources will allow.

Simple Programming Model: Hadoop MapReduce

Hadoop MapReduce differs from scala's *map* and *reduce* methods in the following ways:

- Map and reduce tasks are defined in user-defined *Mapper* and *Reducer* classes, vs. the *map* and *reduce* methods.
- Both *Mapper* and *Reducer* classes take key-value pairs as input instead of elements of a collection.

About MapReduce jobs, the ASF says:

> "A MapReduce job usually splits the input data set into independent chunks that are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically, both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks." (Apache Software Foundation, 2016)

The canonical example, Hadoop's WordCount program, reads text files and lists each word found in the file, followed by the number of times it appears in the input data set.

First, the *TokenizerMapper* class' *map* method accepts key-value pairs composed of the input filename as the key, and a line of text from the file as the value.

```
36  public static class TokenizerMapper
37       extends Mapper<Object, Text, Text, IntWritable>{
38
39    private final static IntWritable one = new IntWritable(1);
40    private Text word = new Text();
41
42    public void map(Object key, Text value, Context context
43                    ) throws IOException, InterruptedException {
44      StringTokenizer itr = new StringTokenizer(value.toString());
45      while (itr.hasMoreTokens()) {
46        word.set(itr.nextToken());
47        context.write(word, one);
48      }
49    }
50  }  (Apache Software Foundation)
```

So if the the input key-value pair is:

<"a.txt", "The quick brown fox jumps over the lazy dog">,

As Hadoop calls the *map* method in line #42, *key* is set to "a.txt" and *value* is set to "The quick brown fox jumps over the lazy dog." Line 44 splits the string into tokens. Between lines 45 and 48, the map function iterates over the tokens in the list. For each word, the variable *word* is set to the value of the token, and a new key–value pair with the token as the key and a count of 1 is returned to the framework by line 47.

Likewise, the framework sorts the maps by key and groups them by key. It then sends the new key-value pairs <token, <1, 1, 1, 1>> to the *IntSumReducer* class' *reduce* method:

```
52  public static class IntSumReducer
53       extends Reducer<Text,IntWritable,Text,IntWritable> {
54    private IntWritable result = new IntWritable();
55
56    public void reduce(Text key, Iterable<IntWritable> values,
57                       Context context
58                       ) throws IOException, InterruptedException {
59      int sum = 0;
60      for (IntWritable val : values) {
61        sum += val.get();
62      }
63      result.set(sum);
64      context.write(key, result);
65    }
66  }  (Apache Software Foundation)
```

Therefore, by line 56, invocations of the *TokenizerMapper* class' *map* method capture *key* as the token, and a list of number 1s emitted by maps for that *key* token as *values*. The loop at lines 60-62 adds the list of number 1s together.

Here is a simplified example.

```
scala> val txt = "The quick brown fox jumps over the lazy dog"
txt: String = The quick brown fox jumps over the lazy dog

scala> txt.split(" ")
res4: Array[String] = Array(The, quick, brown, fox, jumps, over, the,
lazy, dog)

scala> txt.split(" ").map(s => (s,1))
res6: Array[(String, Int)] = Array((The,1), (quick,1), (brown,1), (fox,1),
(jumps,1), (over,1), (the,1), (lazy,1), (dog,1))
```

Note the way (s, 1) was used to create a type of key-value pair in Scala, called a tuple. The key of the tuple can be accessed using the *_1* element of the tuple, and the value using *_2*.

```
scala> txt.split(" ").map(s => (s,1)).head
res12: (String, Int) = (The,1)

scala> txt.split(" ").map(s => (s,1)).map(s => s._1)
res8: Array[String] = Array(The, quick, brown, fox, jumps, over, the,
lazy, dog)

scala> txt.split(" ").map(s => (s,1)).map(s => s._2)
res9: Array[Int] = Array(1, 1, 1, 1, 1, 1, 1, 1, 1)
```

Hadoop groups (and sorts) the elements by their keys. It is just grouped by key below, for ease of illustration.

```
scala> txt.split(" ").map(s => (s,1)).groupBy(s => s._1)
res7: scala.collection.immutable.Map[String,Array[(String, Int)]] =
Map(lazy -> Array((lazy,1)), dog -> Array((dog,1)), The ->
Array((The,1)), over -> Array((over,1)), brown -> Array((brown,1)),
quick -> Array((quick,1)), jumps -> Array((jumps,1)), fox ->
Array((fox,1)), the -> Array((the,1))
```

This creates another type of key-value pair that is stored in a Scala *Map* collection. The *groupBy* method returns a *Map* collection that associates the given key with all the elements that shared that key. The two instances of the word "the" have different character case, so they are separate. Normalizing to lower case better illustrates the *groupBy*.

```
scala> txt.split(" ").map(s => s.toLowerCase).map(s =>
(s,1)).groupBy(s => s._1)
res6: scala.collection.immutable.Map[String,Array[(String, Int)]] =
Map(lazy -> Array((lazy,1)), dog -> Array((dog,1)), over ->
Array((over,1)), brown -> Array((brown,1)), quick -> Array((quick,1)),
jumps -> Array((jumps,1)), fox -> Array((fox,1)), the -> Array((the,1),
(the,1)))
```

It is not exactly what Hadoop passes to the reducer. Let us drop the token words from the list of map inputs. Note that Scala Map collection elements can still be accessed using the *_1* and _2 members.

```
scala> txt.split(" ").map(s => s.toLowerCase).map(s =>
(s,1)).groupBy(s => s._1).map(kvp => (kvp._1, kvp._2.map(s
=> s._2)))
res8: scala.collection.immutable.Map[String,Array[Int]] = Map(lazy ->
Array(1), dog -> Array(1), over -> Array(1), brown -> Array(1), quick ->
Array(1), jumps -> Array(1), fox -> Array(1), the -> Array(1, 1))
```

The map class passes something like this. Note that the word "the" is associated with an array of two (2) numbers one (1) because it appears twice in the input.

```
scala> txt.split(" ").map(s => s.toLowerCase).map(s =>
(s,1)).groupBy(s => s._1).map(kvp => (kvp._1, kvp._2.map(s
=> s._2).reduce((a,b) => a + b)))
res9: scala.collection.immutable.Map[String,Int] = Map(lazy -> 1, dog -
> 1, over -> 1, brown -> 1, quick -> 1, jumps -> 1, fox -> 1, the -> 2)
```

Scalable & Distributed: Hadoop Distributed File System (HDFS)

A Hadoop cluster uses HDFS to distribute data. HDFS was designed with the following assumptions in mind:

- Hardware Failure: A cluster of hundreds or thousands of machines may host an HDFS file system. Each machine has a number of components that could fail. Larger clusters have more machines, so the probability is higher that some component of the cluster is not working. Therefore, HDFS must detect failed components and automatically recover.
- Streaming Data Access: HDFS supports "short-circuit local reads" to get data to tasks faster when the data and the task are on the same node. (McCabe, 2013)
- Large Data Sets: HDFS is optimized to store up to tens of millions of files, each up to terabytes in size, even as the cluster may scale to hundreds of nodes.
- Simple Coherency Model: To allow many jobs and users to access the same data concurrently, without needing locks, semaphores or mutexes to prevent corruption, HDFS applications support a write-once-read-many access model for files. Files can be truncated or appended, but not modified between the file's beginning and its end.

- "Moving Computation is Cheaper than Moving Data": tasks, made of kilobytes or megabytes of Java class files, can be easily transferred throughout the cluster. Transferring mega-, giga- or terabytes of input data to a task will not be network- and storage-efficient, especially if the task is a parser and its input is a digital forensic image that is hundreds of gigabytes in size.
- Portability across Heterogeneous Hardware and Software Platforms: HDFS was designed to be portable across a variety of platforms. (Apache Software Foundation, 2016)

"Distribution" does not mean wide-area geographic distribution, or even distribution beyond a single data center in a Hadoop context. HDFS supports "rack awareness," meaning that data can be distributed outside the current rack to ensure that it remains accessible even in the event of failure of an entire computer equipment rack. (Apache Software Foundation, 2016)

The NameNode daemon implements HDFS' File Name (file and directory naming) and Metadata Layers (allocation, addressing and bookkeeping).

There is one NameNode per cluster and multiple DataNodes. The NameNode is a single point of failure that cannot be recovered without operator intervention. The "Secondary Namenode" described in the documentation does not provide automatic failover. High availability strategies for Hadoop are available. A secondary NameNode machine can stand in for a failed primary NameNode, at a risk of data loss. (Karanth, 2014)

The network URL of the NameNode, such as, hdfs://ip-10-0-0-27.ec2.internal:8020/ is the root of the HDFS filesystem. In jobs, the absolute path hdfs://ip-10-0-0-27.ec2.internal:8020/user/hadoop/piggybank.jar could be referenced using "/user/hadoop/piggybank.jar" or just "piggybank.jar," for the user "Hadoop."

DataNode daemons implement the Data Layer (management of data clusters).

HDFS replicates file blocks across the cluster. Files are composed of sequences of blocks. All blocks of an HDFS file are the same size, except the last. The NameNode ensures that the number of replicas of each file block is maintained, as defined by either

the cluster-wide replication factor (3 by default), or a per-file replication factor given later. The NameNode ensures that each HDFS file has only one writer at a time. Create a new file, and the NameNode will decide where to create and replicate the file's blocks. The first DataNode that receives the data successfully copies it to the replica locations. The default block size is 64 MiB, (Apache Software Foundation) but Amazon Web Services configures Elastic MapReduce (EMR) servers for its cloud clients with a default block size of 128 MiB. (Amazon Web Services, 2016)

On an hourly basis, each DataNode sends a heartbeat and a block report to the NameNode. A successful heartbeat lets the NameNode know that the DataNode is alive, functioning properly and capable of serving future block storage requests. The NameNode marks DataNodes that do not send a heartbeat as dead, and their data becomes unavailable. The block report contains an inventory of the blocks the DataNode is storing. The NameNode checks the block report to see which blocks the DataNode did not report. The NameNode then replicates any blocks that are replicated fewer times than the cluster's or the file's replication factor.

Pertaining to RAID, the best practice is installing RAID only on operating system drives of Hadoop nodes to lengthen their service life; otherwise, DataNode storage should be just a bunch of disks (JBOD). (Loughran, 2012) (Holoman & O'Dell, 2015) (Sammer, 2012) First, the HDFS software replicates blocks, so the data is available elsewhere. In addition, using RAID to store data redundantly means less total storage for the cluster. Hadoop workloads involve large sequential I/O reads and writes, so dividing the I/O operations across the drives' individual controllers and spindles actually performs better than having them queue in the RAID controller. For these reasons, DataNode daemons use the DataNode machines' underlying local filesystems for HDFS storage. For example, a DataNode could be set up to use UNIX filesystems on /disk1, /disk2, and /disk3 for storage.

The "hadoop fs" command does most basic HDFS file management. For example, to list the contents of the "/user/hadoop" directory on HDFS use the –ls option:

```
hadoop fs -ls /user/hadoop
<add>
```

Author Name, email@address

To copy the piggybank.jar file from the local UNIX to the "/user/hadoop" directory on HDFS, use the –copyFromLocal option:

```
hadoop fs -copyFromLocal /usr/lib/pig/piggybank.jar /user/hadoop
<add>
```

Likewise, to copy the piggybank.jar file from the local UNIX to the "/user/hadoop" directory on HDFS, use the –copyToLocal option:

```
hadoop fs -copyToLocal /user/hadoop/output/part-r-00000 .
<add>
```

Reliable: Hadoop Yet Another Resource Negotiator (YARN)

The YARN ResourceManager manages requests for the cluster's CPU, RAM, disk and network resources. The ResourceManager also can schedule cluster aware applications that do not use Hadoop directly to execute on the cluster. NodeManager processes running on the DataNode machines report resource usage to the ResourceManager process. Jobs negotiate with the ResourceManager to allocate work to the cluster through the ApplicationManager interface code. If the ResourceManager agrees that work can be allocated to a node, the ApplicationManager can ask the node's NodeManager to start resource container on the node. The ApplicationManager monitors the progress of the job's containers, and the ResourceManager monitors the health and resources of the nodes.

3.2.2. Apache HBase

HBase is a database that makes use of the distribution and fault-tolerance of Hadoop HDFS. It is derived from Google's BigTable (Chang, Dean, Ghemawat, Hsieh, & Wallach, 2006).

Although HBase is defined in terms of tables, rows, and columns, its data model is not at all like a spreadsheet or relational database. Some more suitable descriptions are a multidimensional map, a nested associative array or a list of key-value pairs that contain key-value pairs. Within an HBase table, the location of a data cell is specified by its row key, column family, column key, and timestamp or version number.

When a table is created, it has a name, which is a text string and a column family, which is also a text string. Cells that share the same column family are stored together in HDFS. (Apache HBase Team, 2016)

Row and column keys are byte arrays. HBase sorts them in alphabetical order.

Each data cell has a version number. If no value is specified when a cell is saved, the current UNIX epoch time is used.

HBase has a Master Server and multiple Region Servers. Each Region Server serves regions, or ranges of row keys. Using row keys that cluster together, like numbers in a series, dates, or times, can cause storage and retrieval traffic to "hot spot" on one region server. Big data vendor MAPR has produced a short "Whiteboard Walkthrough" about HBase row key selection for time series data. (MAPR, 2015)

3.2.3. Apache Spark

Apache Spark is a powerful advancement in Hadoop tooling. It is multi-platform, does not require Hadoop in test, and has language bindings for Java, Python, Scala and the R mathematics language. It offers the developer the ability to create powerful massively-parallel distributed processing applications with the ease of use of the Scala API and tools.

The Apache Spark website describes Spark as:

> "Apache Spark is a fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Spark Streaming." (Apache Software Foundation)

The main component of Spark programs is the Resilient Distributed Data Set (RDD). An RDD is a distributed collection whose elements are distributed to memory of machines in the cluster. If an RDD will not fit in the cluster's RAM, elements can be configured to be cached to disk, otherwise they are re-generated when needed again.

RDDs can undergo transformations and actions. Transformations, like *map*, are fast, and can be chained and/or done in parallel. Actions are operations that require generating an output, like *reduce, count,* or *collect*. The *collect* method sends the elements of the RDD to the Spark client as an array. If the RDD is sufficiently large, *collect* could easily cause an OutOfMemoryError.

When a Spark job is submitted using "spark-submit," the submitter can request a number of YARN application containers to start, called "executors," and how much RAM each executor should have. This log snippet shows recovery of a "Lost executor":

```
16/02/27 21:37:01 ERROR cluster.YarnScheduler: Lost executor 6 on
ip-10-0-0-156.ec2.internal: remote Akka client disassociated
16/02/27 21:37:01 WARN remote.ReliableDeliverySupervisor:
Association with remote system [akka.tcp://sparkExecutor@ip-10-0-0-
156.ec2.internal:39097]
has failed, address is now gated for [5000] ms. Reason is:
[Disassociated].
16/02/27 21:37:01 INFO scheduler.TaskSetManager: Re-queueing tasks
for 6 from TaskSet 1.0
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 92), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 88), so marking it as still running
16/02/27 21:37:01 WARN scheduler.TaskSetManager: Lost task 146.0
in stage 1.0 (TID 1151, ip-10-0-0-156.ec2.internal): ExecutorLostFailure
(executor 6
 lost)
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 93), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 89), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 87), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 90), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 91), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 85), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 86), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Resubmitted
ShuffleMapTask(1, 94), so marking it as still running
16/02/27 21:37:01 INFO scheduler.DAGScheduler: Executor lost: 6
(epoch 0)
```

```
16/02/27 21:37:01 INFO storage.BlockManagerMasterActor: Trying to
remove executor 6 from BlockManagerMaster.
16/02/27 21:37:02 INFO storage.BlockManagerMasterActor: Removing
block manager BlockManagerId(6, ip-10-0-0-156.ec2.internal, 34952)
16/02/27 21:37:02 INFO storage.BlockManagerMaster: Removed 6
successfully in removeExecutor
16/02/27 21:37:02 INFO scheduler.Stage: Stage 1 is now unavailable
on executor 6 (536/598, false)
16/02/27 21:37:17 INFO scheduler.TaskSetManager: Starting task
146.1 in stage 1.0 (TID 1152, ip-10-0-0-154.ec2.internal, RACK_LOCAL,
1396 bytes)
16/02/27 21:37:17 WARN scheduler.TaskSetManager: Lost task 123.0
in stage 1.0 (TID 1148, ip-10-0-0-154.ec2.internal): java.io.IOException:
Failed to
connect to ip-10-0-0-156.ec2.internal/10.0.0.156:34952
```

4. Proof of Concept

The proof of concept test was conducted in the Amazon Web Services (AWS) cloud. AWS offers, through its Amazon Elastic MapReduce (EMR) service, pre-configured Hadoop clusters.

The test cluster was composed of 1 (one) "m1.large" master host and eight (8) "m2.2xlarge" hosts. Specifications for the instances, excerpted from the AWS website, are given below.

Table 2: AWS instance types used in the proof of concept

Instance Family	Instance Type	Pro-cessor Arch	vCPU	Memory (GiB)	Instance Storage (GB)	EBS-optimized Available	Network Performance
General purpose	m1.large	64-bit	2	7.5	2 x 420	Yes	Moderate
Memory optimized	m2.2xlarge	64-bit	4	34.2	1 x 850	Yes	Moderate

Note: Adapted from "Previous Generation Instances" **(Amazon Web Services)**

The software configuration is listed below:

Table 3: AWS Elastic MapReduce installed software

AMI Version	Includes	Notes	Release Date
3.11.0	• AWS SDK for Java 1.10.41 • AWS SDK for Ruby 2.2.8 • Amazon Linux version 2015.09 • Hadoop 2.4.0 • Hive 0.13.1 • Hue 3.7.1 • Pig 0.12.0 • Hbase 0.94.18 • Impala 1.2.4 • Mahout 0.9 • Java: Oracle/Sun jdk-7u76 • Perl 5.16.3 • PHP 5.6.14 • Python 2.6.9 • R 3.2.2 • Ruby 1.8.7 • Scala 2.11.1 • Spark 1.3.1	This Amazon EMR AMI version provides the following bug fixes and changes: • Fixed a bug that prevented MapReduce JobHistory logs from pushing to Amazon S3. • Fixed bugs that prevented YARN container logs from being pushed to Amazon S3.	4 January 2016

Note: Adapted from "AMI Versions Supported in Amazon EMR" **(Amazon Web Services)**

The image used for performance testing is a 70 GB EWF image of a computer owned by the author.

Author Name, email@address

4.1. Security

DISCLAIMER: Testing digital forensic tools in the cloud is not without risk. Please get permission to use be prepared for possibly losing control of all data sent to the cloud. Also, before experimenting, please research the vendor's security features, take responsibility and mitigate risk to the extent possible.

The AWS Cloud Security Whitepaper is a great place to start to learn.

Images and other static files were saved to a private AWS S3 bucket. Code and scripts were checked out from a private GitHub repository using SSH with a key. The cluster was set up in an AWS Virtual Private Cloud. Inbound connections to the master host occurred over port 22 (SSH) only. SSH to the master host was accomplished through an SSH key. Access to web services on the master host, like the HBase console and Ganglia monitoring were tunneled through the SSH connection. All SSH keys used require passphrases.

4.2. Importing the Image

Introducing the image to the cluster means more than copying the image files. The goal is to stage the image for efficient access.

Navigating a forensic image requires alternating between two ways of accessing the image. Starting with a master boot record (MBR), for example, there are some structures and values to parse in this header, followed by the record structures of the partition table. One of the partitions may lead to an NTFS volume on the drive. The NTFS volume boot record has some header fields that lead to the record structures of the MFT. The MFT records represent files and may contain resident and non-resident file attribute records. So parsing the structures properly not only requires referencing offsets in the image and reading the structures sequentially, but it also offers the opportunity to parallelize the parsing of record structures. Once the locations of the structures have been parsed from headers and placed on a list, a *map* function could be used to parse the structures in parallel. Both types of access can be accomplished with the data blocks stored in HBase.

Author Name, email@address

The image input program is an Apache Spark Program that uses the EWFImageInputFormat to load the image into an RDD. The EWFImageInputFormat divides the image into logical "splits" of up to 128 MiB (the HDFS block size) for processing. Then the EWFRecordReader actually reads the image into records of up to 64 KiB (HBase's default block size). The key-value records have the offset into the image as the key and the bytes of the image as the value. The RDD is then transformed into RDDs that are saved directly to HBase using HBase's TableOutputFormat.

To promote uniform distribution of the data across the HBase region servers, the row key is a SHA-1 hash of the image filename and a block ID (the starting offset divided by 128 MiB). This ensures that 128 MiB, plus up to 64 KiB are in each row (since the *starting* offset is used to calculate the block ID). For more, see the diagram below:

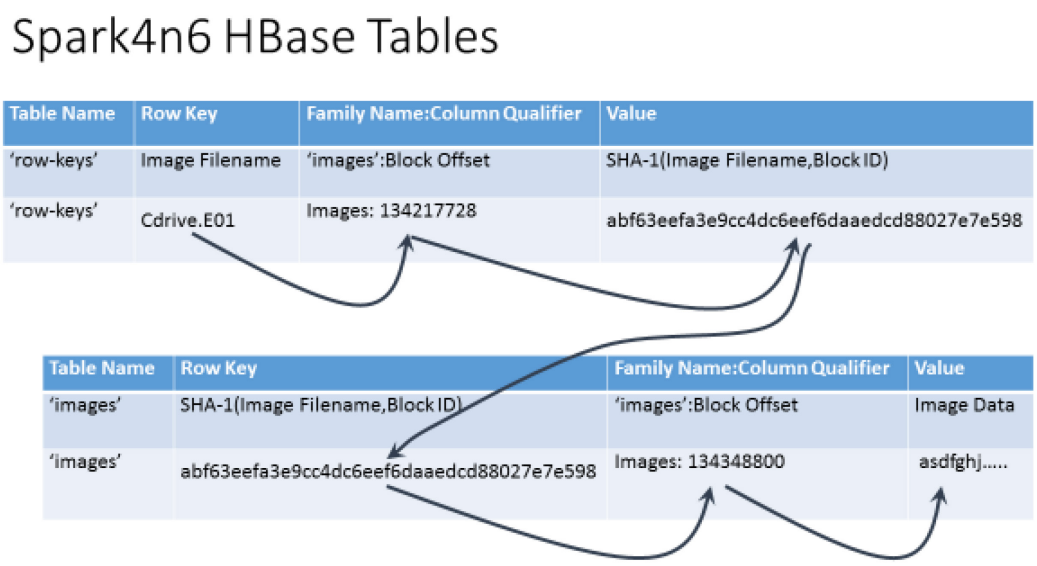

Figure 1: Sample navigation to an image block of interest in HBase

To get data beginning with a given offset of 134348800, we can start a *scan* operation on the 'row-keys' table and the 'images' column family name with the image filename 'CDrive.E01' as the row key and the desired offset divided by 124 MiB as the starting column qualifier. That gives the row key hash in the 'images' table. So we initiate another scan operation on the 'images' table with the row key hash as the starting

row and look for the greatest column qualifier less than or equal to the starting offset and retrieve the column.

As the HBase web UI shows below, the data are evenly distributed in 24 regions across all 8 region servers.

Table Regions

Name	Region Server	Start Key	
images,,1456664355146.17f40883044b90c94e5db16ae34ec915.	ip-10-0-0-153.ec2.internal,60020,1456693858424		0bc9acd
images,0bc9acd22385af4f7eeddb3d6f39865f91e483ee,1456664900259.c0839e41262f75dfa2e4545a687343ff.	ip-10-0-0-156.ec2.internal,60020,1456693853890	0bc9acd22385af4f7eeddb3d6f39865f91e483ee	12124c9
images,12124c961ea55333d82feeea83e4a7d1f2658da1,1456664900259.6a53879bf5e5511da8eb1fec0fea15a5.	ip-10-0-0-154.ec2.internal,60020,1456693849126	12124c961ea55333d82feeea83e4a7d1f2658da1	2000000
images,2000000000000000000000000000000000000000,1456664428302.36b2228d08fdfd656cdcd1193be33996.	ip-10-0-0-151.ec2.internal,60020,1456693850110	2000000000000000000000000000000000000000	2bee548
images,2bee548522c8a52a49403a10de25da3ed11faf3e,1456665015160.cb5a7fc277810befbdc0094b19945f87.	ip-10-0-0-155.ec2.internal,60020,1456693848078	2bee548522c8a52a49403a10de25da3ed11faf3e	3ab9cce
images,3ab9cce0ceded7a98af16c20773bd14d2f41a828,1456665015160.6a34a9b0115c1e25f9216fa6bff298e8.	ip-10-0-0-157.ec2.internal,60020,1456693848747	3ab9cce0ceded7a98af16c20773bd14d2f41a828	4000000
images,4000000000000000000000000000000000000000,1456664836296.0eb95285fa733d9d3df5e69621f2d6b5.	ip-10-0-0-150.ec2.internal,60020,1456693873035	4000000000000000000000000000000000000000	4e50aa2
images,4e50aa2781195f859219b0d82d905086f1c0ab23,1456664836296.e61487e27b49562d96219767b6d2ef53.	ip-10-0-0-154.ec2.internal,60020,1456693849126	4e50aa2781195f859219b0d82d905086f1c0ab23	567826a
images,567826aca69991e0f8bd71e848053458a880f811,1456664349483.e9f4fa31bdc661d87f6edb878688280d.	ip-10-0-0-151.ec2.internal,60020,1456693850110	567826aca69991e0f8bd71e848053458a880f811	6000000
images,6000000000000000000000000000000000000000,1456664364876.166c9abd30589ea92611ae9eb9b40df8.	ip-10-0-0-155.ec2.internal,60020,1456693848078	6000000000000000000000000000000000000000	6d0479d
images,6d0479dc9d594ae83564a66485cb7a0d2bb47456,1456664859905.15400100455f431caaa59b3c57be9d33.	ip-10-0-0-153.ec2.internal,60020,1456693858424	6d0479dc9d594ae83564a66485cb7a0d2bb47456	75985f2
images,75985f24a465124d7b54a8b57d01bfce5f5663b2,1456664859905.73ff3db50ab90aeca38aa0ed49835533.	ip-10-0-0-150.ec2.internal,60020,1456693873035	75985f24a465124d7b54a8b57d01bfce5f5663b2	8000000
images,8000000000000000000000000000000000000000,1456664457639.cf6ce246dba32db8b9705d137d22e156.	ip-10-0-0-152.ec2.internal,60020,1456693841069	8000000000000000000000000000000000000000	93ae157
images,93ae15790f12ad4ca161f906c21f6b9ac89770f1,1456665307403.dd216ee770852192957502l7f919a05f.	ip-10-0-0-150.ec2.internal,60020,1456693873035	93ae15790f12ad4ca161f906c21f6b9ac89770f1	97fb758
images,97fb75869dec5be1af317fd55d569acd748bbcbe,1456665307403.70ffd5fb870dc2848d3222e82c54ccf2.	ip-10-0-0-156.ec2.internal,60020,1456693853890	97fb75869dec5be1af317fd55d569acd748bbcbe	a000000
images,a0000000000000000000000000000000000000000,1456664393719.c2ec6cc8e234bc9f8d06416cf2467f5d.	ip-10-0-0-154.ec2.internal,60020,1456693849126	a0000000000000000000000000000000000000000	a5538b2
images,a5538b21b7110d9a87c558d8510f6223c9cc62c1,1456664624112.f063aa4486a9427bd1779eeaabe1d1b5.	ip-10-0-0-157.ec2.internal,60020,1456693848747	a5538b21b7110d9a87c558d8510f6223c9cc62c1	b0afcfd3
images,b0afcfd3f32fbfda89113ee08326b8a2b1cb0a2f,1456664624112.26a7b17d1bc6f299897368908e517114.	ip-10-0-0-156.ec2.internal,60020,1456693853890	b0afcfd3f32fbfda89113ee08326b8a2b1cb0a2f	c000000
images,c0000000000000000000000000000000000000000,1456664397814.123028be4a9bf48f9ae11378650a94f2.	ip-10-0-0-151.ec2.internal,60020,1456693850110	c0000000000000000000000000000000000000000	c8ce2cb
images,c8ce2cb58e9845baefa242ec3fff8914194ee719,1456664842207.17e1140f6739cacf821757f1b91fb02e.	ip-10-0-0-152.ec2.internal,60020,1456693841069	c8ce2cb58e9845baefa242ec3fff8914194ee719	d581063
images,d581063f95f46cb3dde4dd40bb38d442e2b00abc,1456664842207.389fbeaea66ec3e562fe37b7736fa3ec.	ip-10-0-0-155.ec2.internal,60020,1456693848078	d581063f95f46cb3dde4dd40bb38d442e2b00abc	e000000
images,e0000000000000000000000000000000000000000,1456664994238.a6c84652aa851cf36ba32f27f2d20870.	ip-10-0-0-157.ec2.internal,60020,1456693848747	e0000000000000000000000000000000000000000	e74f1c5
images,e74f1c5f0474653a150cc50af45757f440eea80b,1456664994238.4fe53e69bbfba901cf9e950807be1d8d.	ip-10-0-0-152.ec2.internal,60020,1456693841069	e74f1c5f0474653a150cc50af45757f440eea80b	ed9d51f
images,ed9d51f89118790af911df6b90f2a4c4ed71d9cc,1456664362008.d002b89f4b37da034172b50c5dd7a9fd.	ip-10-0-0-153.ec2.internal,60020,1456693858424	ed9d51f89118790af911df6b90f2a4c4ed71d9cc	

Regions by Region Server

Region Server	Region Count
ip-10-0-0-150.ec2.internal,60020,1456693873035	3
ip-10-0-0-151.ec2.internal,60020,1456693850110	3
ip-10-0-0-152.ec2.internal,60020,1456693841069	3
ip-10-0-0-153.ec2.internal,60020,1456693858424	3
ip-10-0-0-154.ec2.internal,60020,1456693849126	3
ip-10-0-0-155.ec2.internal,60020,1456693848078	3
ip-10-0-0-156.ec2.internal,60020,1456693853890	3
ip-10-0-0-157.ec2.internal,60020,1456693848747	3

Figure 2: HBase Region Server Listing

Varying the number of executors or amount of RAM per executor did have an effect. More executors or more RAM only helped slightly.

Author Name, email@address

Figure 3: Image import performance, varying the number of executors on the cluster and RAM per executor

4.3. An Unsuitable Application: Verifying Integrity

Verifying the integrity of an acquired image is an important pre-analysis step. The goal is to hash the image to check to ensure that it matches the hash calculated at acquisition. Verifying the image's integrity is also an important test of whether the image importation software is working properly.

However, hash functions like MD5 and SHA-1 cannot be parallelized. The hash of any given block of a file except the first is a function of values accumulated by calculating the hash function on all of the preceding blocks. (Rivest, 1992) (National Institute of Standards and Technology , 2015) (Schneier, 2015) So they are neither commutative nor associative functions. So a sequential stream of the image has to be generated and passed to the hash function. Fortunately, HBase's scan operation can serve stored values in order, first by row key, then by column key with low latency. By comparison, the *sha1sum* UNIX command can be used to calculate a SHA-1 hash of a digital forensic image.

The proof of concept's Apache HBase SHA-1 hashing program used HBase's *scan* operation on two tables to get the image's blocks in order and calculate a SHA-1 hash. The SHA-1 implementation used is part of the java.security.MessageDigest Oracle Java runtime library. Results were compared with results of running the *sha1sum* UNIX

command on the dd image on the master node's local storage. Both programs successfully calculated hashes that match the acquisition hash.

Table 4: Hashing performance comparison

String Extraction Program	Running time (minutes)
Apache HBase-based SHA-1	82:53.858
UNIX *sha1sum*	18:34.942

Here is a snippet of the script that runs the tests:

```
# 'time' times the execution.  Curly braces were needed so that the
timing
# would be directed to the log file
# 'tee -a' appends to the test.log file (but lets me still see what's
happening)
#
# Does an HBase scan to get the contents of the image and calculate a
SHA1
{ time java com.edwardsit.spark4n6.HBaseSHA1 500GB-CDrive.E01
} 2>&1 | tee -a test.log
echo SHA1 should = 2bf1b3af2f049c0cda50d5c01c83f5a3dec3e43a
#
# Uses the sha1sum UNIX command to calculate a SHA1 of the image
{ time sha1sum /mnt/500GB-CDrive.E01.dd
} 2>&1 | tee -a test.log
echo SHA1 should = 2bf1b3af2f049c0cda50d5c01c83f5a3dec3e43a
```

This is an excerpt of the output:

```
16/02/29 04:08:06 INFO spark4n6.HBaseSHA1: 73.14 GiB read,
100.00% complete
500GB-CDrive.E01 = 2bf1b3af2f049c0cda50d5c01c83f5a3dec3e43a

real    82m53.858s
user    16m4.896s
sys     7m47.304s
2bf1b3af2f049c0cda50d5c01c83f5a3dec3e43a  /mnt/500GB-
CDrive.E01.dd

real    18m34.942s
user    4m45.456s
sys     0m46.348s
```

In both cases, Ganglia shows that only the master host (on the bottom) is under moderate load.

Author Name, email@address

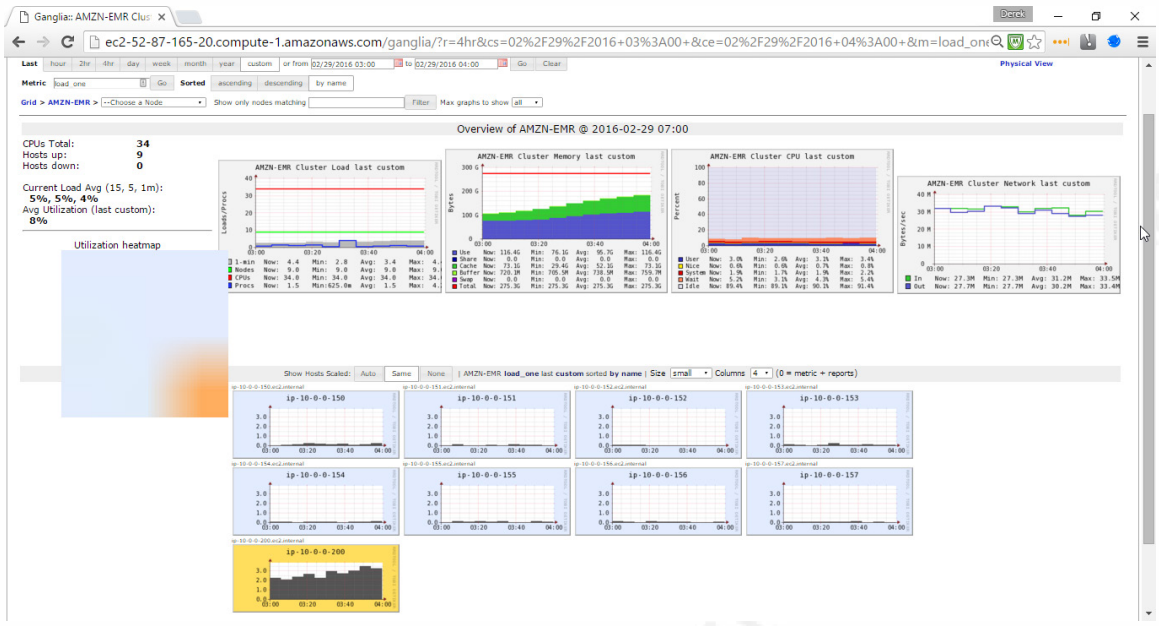

Figure 4: Ganglia monitoring during hash calculation

4.4. A Fitting Application: Finding Strings

Searching for text strings of interest is a key requirement of a digital forensic system. (NIST Computer Forensic Tool Testing (CFTT), 2008) the most important proof that could be obtained may be text the user has written. Text hidden other places can also be important clues.

One way of searching is to search the image for specific key words as needed. Another way involves two steps: first extracting all readable text to an index file using a program like the *strings* UNIX command, then using a text-searching program like the *grep* UNIX command or the *find* command on Windows to the desired string in the index file.

The *strings* UNIX command is used to extract readable ASCII or Unicode text from binary images. The *strings* command, when run on a suspicious executable, may reveal instructions for users of the program, error messages, IP addresses or even user identities and passwords. Likewise, when run on forensic images, *strings* likewise can reveal documents, e-mail messages, and web pages and stored Personally Identifiable Information (PII) like names, e-mail addresses, telephone numbers, credit card numbers and social security numbers.

Author Name, email@address

The proof of concept used an Apache Spark program running on the master node to extract readable ASCII text from the image. The program uses HBase's TableInputFormat to load the image's blocks into a Spark RDD in parallel. The RDD is then transformed to an RDD of strings and offsets into the image where the strings were found. The RDD is then *collect*ed to send the strings to the standard output on the master node. The proof of concept program's running time was compared with running the *strings* UNIX command on the dd image on the master node's local storage. Both programs' output was discarded to save testing time and disk space on the master node.

Table 5: 'strings' programs performance comparison

String Extraction Program	Running time (minutes)
Apache Spark-based string extraction program	13:8.572
UNIX *strings*	108:11.512

Here is the snippet of the script that runs the tests:

```
{ time spark-submit \
        --num-executors 24 --executor-memory 8g \
        --class com.edwardsit.spark4n6.Strings \
        target/scala-2.10/spark4n6_2.10-1.0.jar >/dev/null
} 2>&1 | tee -a test.log
{ time strings –t d /mnt/CDrive.E01.dd >/dev/null
} 2>&1 | tee -a test.log
```

This shows Ganglia monitoring activity during the run of the Apache Spark-based string extraction program. The whole cluster is engaged, primarily the master (in red on the bottom), which is collecting the input, but the remainder of the cluster is not overburdened.

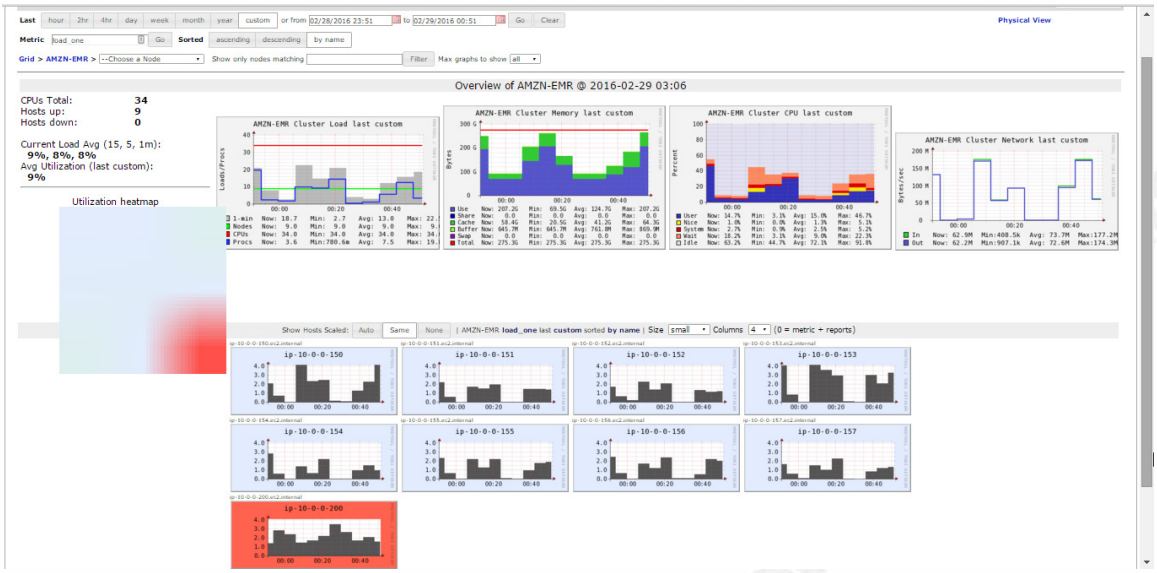

Figure 5: Ganglia monitoring during Spark strings calculation

This shows Ganglia monitoring activity during the run of the *strings* UNIX command. Only the master host is busy (in red at the bottom).

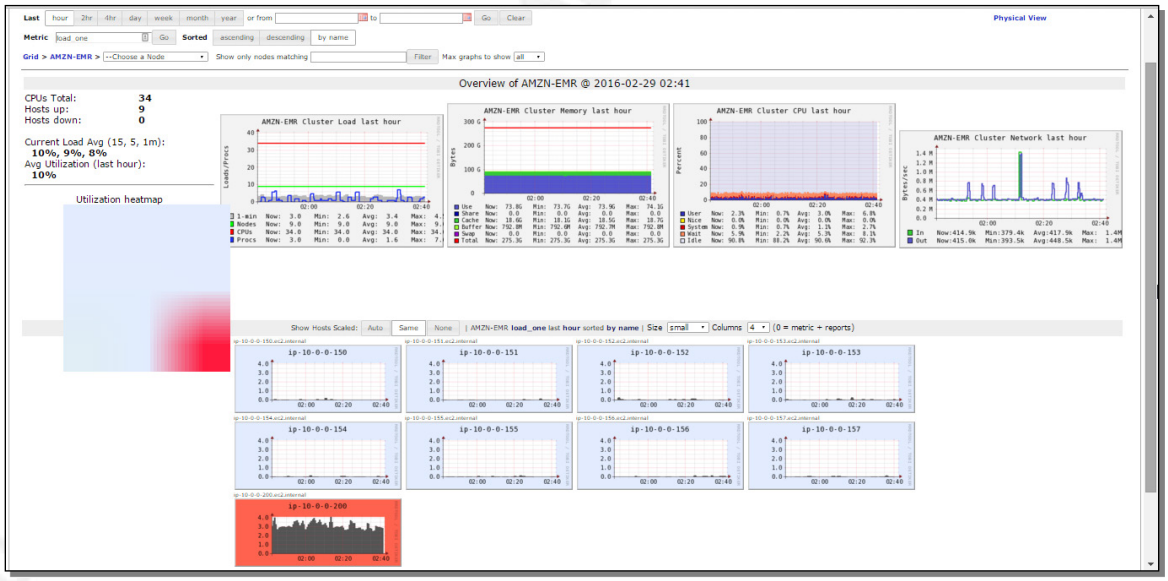

Figure 6: Ganglia monitoring during UNIX 'strings' calculation

5. Future Directions

Amazon has developed a ruggedized 50 TB storage device called Snowball with an attached Kindle reader that acts as a management interface when the device is in use or as a shipping label otherwise. It is essentially a portable S3 bucket. (Barr, 2015) The

client interface for copying data to one or more devices over a local network appears to implement a subset of the functionality of the *hadoop distcp* program. Acquisition could be a "killer app" for a device with this capacity, especially if its API could support acquisition in parallel using tools like Spark.

6. Conclusion

Digital forensic workloads seem to have outgrown the digital forensic workstation, which is limited to the CPU and memory resources of a single node. Adding storage usually offers little help.

On the other hand, the new architectural foundation for digital forensics shows that "Big Data" tools and technologies may be able to help with many aspects of the data challenge in digital forensics. It makes use of Apache Hadoop's reliable, expandable storage, HBase's, sequential access and and Spark's distributed parallel in-memory computation. Both a fitting application of the technology, finding strings, and an unsuitable application, hashing, were demonstrated.

Author Name, email@address

7. References

Amazon Web Services. (2016). *HDFS Configuration.* Retrieved from Amazon Elastic MapReduce: http://docs.aws.amazon.com/ElasticMapReduce/latest/ReleaseGuide/emr-hdfs-config.html

Apache HBase Team. (2016, February 24). *Apache HBase ™ Reference Guide.* Retrieved from Apache HBase ™: https://hbase.apache.org/book.html#columnfamily

Apache Software Foundation. (2016, 01 26). *HDFS Architecture.* Retrieved from Apache Hadoop: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

Apache Software Foundation. (2016, January 26). *HDFS Users Guide.* Retrieved from Hadoop: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

Apache Software Foundation. (2016, 01 26). *MapReduce Tutorial.* Retrieved from Apache Hadoop: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Apache Software Foundation. (n.d.). *GC: WordCount.* Retrieved from grepcode.com: http://grepcode.com/file/repo1.maven.org/maven2/org.apache.hadoop/hadoop-mapreduce-examples/2.7.1/org/apache/hadoop/examples/WordCount.java#WordCount.main%28java.lang.String%5B%5D%29

Apache Software Foundation. (n.d.). *hdfs-default.xml.* Retrieved from Apache Hadoop: https://hadoop.apache.org/docs/r2.6.3/hadoop-project-dist/hadoop-hdfs/hdfs-default.xml

Casey, E. (2011). *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet, 3rd Edition.* Boston: Elsevier Inc.

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., & Wallach, D. A. (2006). *Bigtable: A Distributed Storage System for Structured Data.* Retrieved from Google: http://static.googleusercontent.com/media/research.google.com/en//archive/bigtable-osdi06.pdf

Author Name, email@address

Cloudera, Inc. (2013, November 15). *BinaryPig: Scalable Static Binary Analysis Over Hadoop.* Retrieved from Cloudera Engineering Blog: https://blog.cloudera.com/blog/2013/11/binarypig-scalable-static-binary-analysis-over-hadoop/

Dean, J., & Ghemawat, S. (2004, December). *MapReduce: Simplified Data Processing on Large Clusters* . Retrieved from Google: http://research.google.com/archive/mapreduce.html

Endgame Inc. (2013, December 2). *Black Hat USA 2013 - BinaryPig - Scalable Malware Analytics in Hadoop*. Retrieved from YouTube: https://www.youtube.com/watch?v=fcLkTvnBpIw

Garfinkel, S. L. (2010). Digital forensics research: The next 10 years. *Digital Investigation*, S64-S73.

Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003, October). *The Google File System* . Retrieved from Google: http://research.google.com/archive/gfs.html

Harris, D. (2013, March 4). *The history of Hadoop: From 4 nodes to the future of data.* Retrieved from Gigaom: https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/

Holoman, J., & O'Dell, K. (2015, January 16). *How-to: Deploy Apache Hadoop Clusters Like a Boss.* Retrieved from Cloudera Engineering Blog: http://blog.cloudera.com/blog/2015/01/how-to-deploy-apache-hadoop-clusters-like-a-boss/

Karanth, S. (2014). HDFS high availability. In S. Karanth, *Mastering Hadoop.* Birmingham, UK: Packt Publishing.

Laney, D. (2001, February 6). *3D Data Management: Controlling Data Volume, Velocity and Variety.* Retrieved from Gartner Web site: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwj5rPLQjpPLAhUIVT4KHcbNAzAQFgg5MAA&url=http%3A%2F%2Fblogs.gartner.com%2Fdoug-laney%2Ffiles%2F2012%2F01%2Fad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and

Author Name, email@address

Loughran, S. (2012, November 9). *Why not RAID-0? It's about Time and Snowflakes.* Retrieved from Hortonworks: http://hortonworks.com/blog/why-not-raid-0-its-about-time-and-snowflakes/

McCabe, C. (2013, August 16). *How Improved Short-Circuit Local Reads Bring Better Performance and Security to Hadoop.* Retrieved from Cloudera Engineering Blog: http://blog.cloudera.com/blog/2013/08/how-improved-short-circuit-local-reads-bring-better-performance-and-security-to-hadoop/

NIST Computer Forensic Tool Testing. (2008, January 24). *Forensic String Searching Tool Requirements Specification.* Retrieved from National Institute of Standards and Technology: http://www.cftt.nist.gov/ss-req-sc-draft-v1_0.pdf

Odersky, M. (2002). *What is Scala?* Retrieved from The Scala Programming Language: http://scala-lang.org/what-is-scala.html

Odersky, M., & Spoon, L. (2010, September 7). *Trait Traversable.* Retrieved from scala-lang.org: http://www.scala-lang.org/docu/files/collections-api/collections_3.html

Sammer, E. (2012). Daemons. In E. Sammer, *Hadoop Operations.* Sebastopol: O'Reilly Media, Inc.

Schinz, M., & Haller, P. (2011). *A Scala Tutorial for Java Programmers.* Retrieved from The Scala Programming Language: http://docs.scala-lang.org/tutorials/scala-for-java-programmers.html

Schneier, B. (2015). 18.5 MD5. In *Applied Cryptography: Protocols, Algorithms and Source Code in C, 20th Anniversary Edition.* Indianapolis: John Wiley & Sons.

Subramaniam, V. (2011, September 29). *Why Scala? ...by a hilarious Indian guy.* Retrieved from YouTube: https://www.youtube.com/watch?v=LH75sJAR0hc

The Apache Software Foundation. (2016, February 13). *Welcome to Apache™ Hadoop®!* Retrieved from Apache™ Hadoop®!: http://hadoop.apache.org/