# Blockchain for High Performance Data Integrity and Provenance

## Phase 1 Final Technical Report

January 23, 2018

**Jason Wampler & Garrett Payer**

## Funding Opportunity Number: DE-FOA-0001618

## CFDA Number: 81.049

### HPC Security - Cybersecurity Technologies (3a)

## Executive Summary

The Department of Energy has some of the world‘s largest High Performance Computing (HPC) systems. Science and research funded and performed by the DoE relies significantly on the modeling and simulations that these HPC systems provide. Research performed on these systems can include energy, climate modeling, nuclear weapons and other government funded research by other agencies. The ubiquity of High Performance Computing, computational analysis has become a foundation for many policy and strategic decisions. While largely in the purview of national governments such as the United States Federal Government, the advent of cloud computing and concepts such as Big Data have made massive distributed computation available to industry.

A major concern as increased reliance on the information derived from highly complex computation is unverifiable trust in the integrity of the raw data, transformed data and the underlying processing algorithms. Solutions intended to secure large clusters or HPC systems should not require a separate set of high performance computational resources but should integrate into the computing cluster directly. Overhead should be low or manageable as to not impact the running time of the computations a cluster must perform.

To keep overhead to a minimum, integrate as part of the computing systems, and provide robust options in integrity checking, the solution utilizes a blockchain technology in order to record the provenance of information or data produced from high performance computing. To investigate the feasibility blockhain technology for this purpose a test bed was constructed with to emulated, at a small scale, high performance computation on a smaller. Using this test bed, a number of methods were evaluated to determine which platforms and software could produce a usable solution.

The test bed utilized only a few nodes with each node only being given limited resources. The computational resources of each node were easily maxed similar to many HPC workloads. This cluster has been created using an Openstack Private Cloud instance which makes use of several servers. Virtual machines are created with the requisite software and deployed an environment where experimentation, development, and testing of designs became possible. A subset of the software typically found within HPC systems is used to emulate a workload that emphasize computational complexity as well as utilized a shared storage to enable non-uniform distributed dataset loading.  In order to verify the integrity of a file of any size, in an efficient fashion, a Merkle Tree capability was developed that splits a file into blocks which are each individually hashed allowing this process to be distributed and it’s possible to verify individual blocks rather than the entire file at a time.

Blockchain platforms were evaluated to determine which platform would fit the solution the best. Openchain appeared to be viable candidate, however, Ethereum’s readily available documentation, tools, and relative maturity had the needed capabilities. Ethereum facilitates the running of distributed applications called smart contracts, allowing integrity requirements to be coded directly into its public blockchain, or through a private instance. A number of rudimentary smart contracts were constructed to receive input such as integrity data, however, only preliminary development on sending integrity information to the blockchain was completed.

## Contents

# 1. Introduction

The Department of Energy (DoE) has some of the world's largest High Performance Computing (HPC) systems. Science and research funded and performed by the DoE relies significantly on the modeling and simulations that these HPC systems provide. Research performed on these systems can include energy, climate modeling, nuclear weapons and other government funded research by other agencies. Research missions go beyond local agency use and HPC platforms within DoE and other agencies are national assets for use in any US funded research. Much of this research directly supports the United States security, economic, energy and health/bioscience postures. The Nation's policy makers and policy decisions depend on the research performed on these HPC systems. Compromise of these systems could significantly undermine our Nation's security and safety.

## 1.2 Integrity of High Performance Systems

The increased reliance on the information derived from highly complex computation has led to the unverifiable trust in the integrity of the raw data, transformed data and the underlying processing algorithms. The ubiquity of High Performance Computing, computational analysis has become a foundation for many policy and strategic decisions. While largely in the purview of national governments such as the United States Federal Government, the advent of cloud computing [1] and concepts such as Big Data [2] have made massive distributed computation available to industry.

The scale of information that could be monitored for integrity verification does not scale and the ambiguity of the actual remote user within a remote organization poses unique security challenges**.** Controls on user access at a remote organization rely on the policy and environment of that organization. So, even though a user or set of users may have undergone a vetting process with the Government for access to a particular system, there is no guarantee in an open academic or commercial environment that another unvetted user is not accessing those systems. The Department of Defense model in restricting function to the minimum functionality and adding privileges upon request does not lend itself well to the flexibility required of open research.

While not at the same scale of super computing, technologies such as Big Data enable businesses to harness significant amounts of parallel, distributed computing for data driven decision making such as business intelligence [3], predictive analytics [4] or data analyses performed as part of their normal operations. While DoE climate modeling can have profound implications for national priorities on energy, secure and reliable computation and data processing will significantly influence large and small businesses alike as they consume increasing amounts of information in order to deliver up to the minute insights into their operations and customer markets.

In order to trust the analysis performed using massive parallel computation, a number of unique issues will arise when evaluating the integrity of the operation:

- **Integrity of Computation** − Individual nodes performing computation are influenced in order to alter or influence the outcome. An attacker running authorized or unauthorized

tasks, could cause changes in how the computation is performed tainting the results that are produced.

- **Errors with Initial Data** − When raw data is to be analyzed, the process involves a number of transformations incorporated derivative or new raw data until a final outcome is reached. In the event that the raw data has been found to be incorrect or tampered with, results derived from this data must be identified
- **Hardware Errors** − Faulty hardware or the MTBF (Mean Time between Failures) of commodity devices can cause computations to provide inaccurate results. While many systems rely on hardware based mechanisms such as Error-Correcting Code (ECC) memory, not all hardware is protected, and not every possible hardware error is measurable or predictable.

### 1.3 Solution Considerations

We acknowledge that these considerations are not unique involving any process that requires computation. However, processes that run as HPC operations have unique considerations. HPC typically run on very large clusters, a few of which DoE currently operates. Not only are these systems utilizing the latest, high performance hardware, the computations themselves are designed to utilize much, if not all, the capacity provided by these systems.

Solutions targeting HPC systems must take into account that these systems will impact the ability to perform their function to a degree. Running software on the cluster may lead to performance degradation. Additionally, moving large amounts of data out or around a cluster could also significantly impact the HPC interconnecting network, degrading performance.

Any solution intended to secure large clusters or HPC systems should not require a separate set of high performance computational resources but should integrate directly into the existing system as its own process or use a system at a slower computational rate than that of the HPC being monitored. To improve the integrity of HPC processing, we investigated recording data provenance to track information creation and to utilize blockchain technology to reduce the overhead of saving and using this information compared to other solutions.

#### 1.3.1 Data Provenance

Establishing the provenance [5] of information is to understand how it was derived. For scientific information, this involves a specific workflow in which raw information is ingested and a number of processes run on one or more compute nodes in which a final result is created. A workflow can use a number of different sources of raw data and many different processes as illustrated in Figure 1.

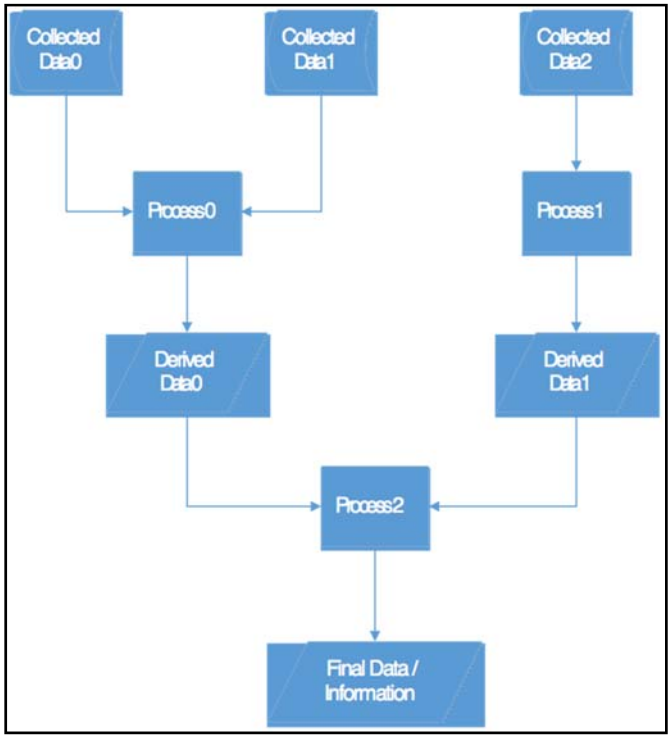


*Figure 1 – Raw Data is consumed by a number of processes in a specific order compute a final result.*

For some workflows, process monitoring could scale to literally thousands of processes with hundreds of different data sources. Additionally, each process has a computational load requiring itself one or more nodes within a cluster or HPC system. The integrity of the final, produced information is influenced by the integrity of the data, the processes, and nodes where processing took place.

Supporting data provenance within an HPC system or large clusters presents a unique challenge. Any solution provided will need to scale with the HPC system as well as integrate as part of infrastructure. A solution should not require its own, separate HPC system or cluster in order to operate. Additionally, a solution should not be tied to any specific architecture or piece of code in order to support data provenance. Many of the workloads run on these systems can vary significantly; even using completely different programming languages and tools.

Our solution focuses on providing the information needed to understand that when a compromise to integrity has been detected, it is possible to determine what derived information has become tainted. If an attacker has compromised a single node for a particular span of time, with data provenance, it would be possible to determine which processes were run on the node and at which times. This information can then be used to determine which derived information used these specific processes and they can be flagged as tainted. Additionally, we expect the final version of the solution to scale from a cluster of a few nodes to HPC systems with thousands of nodes while being agnostic enough to support multiple platforms such as MPI [6], Hadoop [7], and other large scale computational technologies.

### 1.3.2 Blockchain

Blockchain [8] technology underpins the distributed nature of the Bitcoin [8] cryptocurrency. While Bitcoin uses cryptographic functions, the blockchain technology itself is not encryption. Bitcoin uses the blockchain cryptographic processes to mathematically generate its currency but the overall innovation of blockchain is the distributed nature in which Bitcoin transactions are generated, logged, and validated through the Bitcoin blockchain through an asynchronous Distributed Ledger. Figure 2 illustrates the basic blockchain mechanism.

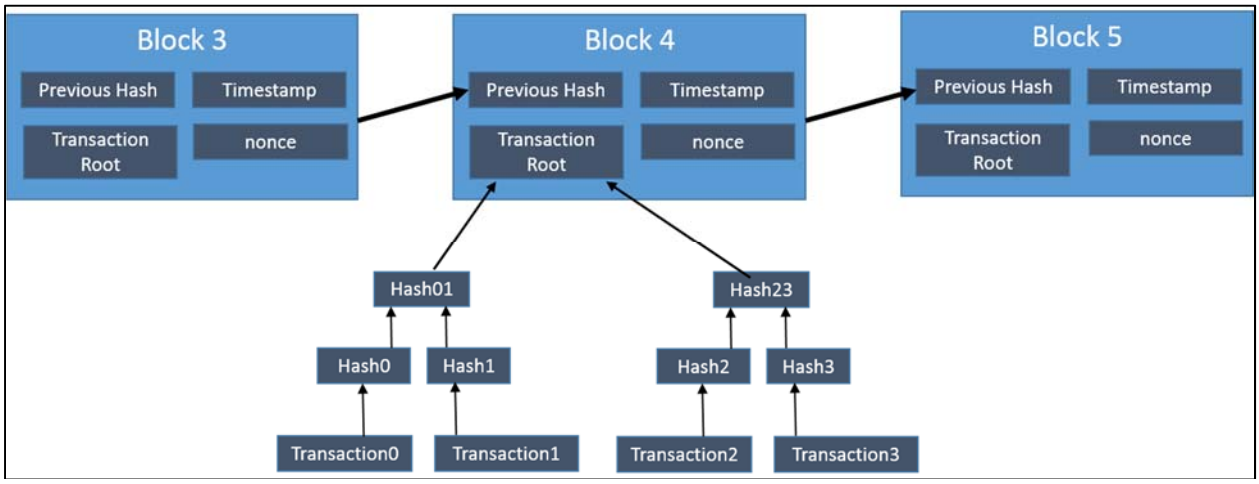


*Figure 2 – Illustration blockchain and storing hashes of data and processes as linked transactions.*

With the blockchain leveraging Distributed Ledger Technology, every Bitcoin transaction ever taken place since its inception is tracked, available, and viewable. By examining the Bitcoin blockchain, it is possible to see every exchange of currency between parties all the way back to the genesis of that currency.

To commit transactions to the blockchain, clients must submit their transaction to a participating node or nodes hosting the blockchain. Several transactions are then aggregated into a single block. The block is "mined" which is the "Proof of Work" [8] needed before being committed to blockchain. Unlike email, where the cost of sending email is essentially zero, "Proof of Work" requires an amount of work or computational effort, before amending blockchain with the current block of transactions. The node that performs this work is called a miner [8]. This "Proof of Work provides an easily verifiable result but is computationally difficult to perform.

By mining a block, a miner can send the block of transactions to other nodes with the result. This will allow the nodes to verify that the miner did perform the work and that the block can be committed to the Distributed Ledger. If the transaction was submitted to other nodes, the nodes will stop mining their current block if the transaction is contained within. If other transactions were in this block but were not located in the accepted block, those are re-aggregated into a new block, and the mining process starts over again.

Outside of transaction details, the Bitcoin blockchain can also contain various amount of metadata which can include other details or give the transaction more context with examination. This ability was included in later versions of Bitcoin as this was being carried out using unused

fields in the transaction. One of the major issues with expanding metadata use within transactions is that this can balloon the transaction size.

With millions of transactions a day, adding ten or a hundred bytes to every transaction can yield significant growth in the transaction log. It becomes imperative that only the minimal amount of information is required in each transaction. Additionally, the unspent transaction output (UTXO), which is the latest blocks that indicate unspent currency, are stored within memory rather than disk, as these are typically most frequently accessed blocks.

### 1.3.3 Report Organization

In the section "3. Methods and Results", the work we accomplished is detailed. Each section describes a effort made to accomplish a part of a task. The discussion section associated with each parent section discusses the implications of the work accomplished and what resulted from them. Additionally, it may contain additional detail in how the work performed will be utilized in future work.

## 2. Purpose

In order to develop a usable solution for the assessing the integrity of High Performance Computing, our effort focused on:

- Sufficiently replicating high performance computation on a smaller scale.
- Assess existing blockchain technology to determine if existing technology will provide capability of saving sets of transactions.
- Evaluate existing data provenance technology.
- Investigate methods for hashing large scale data sets.
- Attempt to create initial prototype using blockchain technology to evaluate the possible overhead produce for this type of solution.

While there are many options and solutions available to start addressing the unique issues of integrity within the final products of HPC or massively parallel computation, our proposal seeks to specifically address the issue of data provenance and its use in determining which raw or derived data may be tainted in the event of integrity compromise. We seek to accomplish this using blockchain technology to establish a distributed ledger within the same set of computational resources performing the analytic workload.

## 3. Methods and Results

The DoE Blockchain project is an exploration of the use in how Blockchain technology might be used to increase the integrity of HPC workloads and their effects on the performance of these workloads. While the most ideal situation would be to utilize an HPC system in order to perform this research, the resources for the project are limited. Instead of building, or accessing an HPC system, our team built a smaller scale version of a HPC cluster using performance starved virtual

machine. However, typical HPC workloads are designed to utilize HPC clusters and, therefore, are not suited for small scale, performance starved clusters.

### 3.1 Cluster Construction and Deployment

The DoE Blockchain project is an exploration of the use in how Blockchain technology might be used to increase the integrity of HPC workloads While the most ideal situation would be to utilize an HPC system in order to perform this research, the resources for the project were limited. Instead of building, or accessing an HPC system, our team built a smaller scale version of a HPC cluster.

To simulate the taxing nature of HPC workloads our "HPC Cluster" utilizes only a few nodes with each node only being given limited resources. The computational resources of each node are easily maxed similar to many HPC workloads. This cluster has been created using an Openstack Private Cloud instance which makes use of several servers. Virtual machines are created with the requisite software and deployed in the environment where we can experiment, develop, and test our designs.

Virtual Machine Configuration:

- 1 vCPU
- 2048MB of RAM
- 20GB of Disk space.

Shared Storage:

- Filesystem: NFS
- Size: 200GB
- Interconnect Speed: 1Gb/s

The severe amount of resources available to the virtual machines in the cluster sufficiently emulated the performance constraints that can be found on HPC systems by running workloads of mediocre computational requirements.

#### 3.1.1 HPC Cluster Instance

The "HPC Cluster" instance exists a set of virtual machines located on the Openstack private cloud. An instance of the cluster can be generated on the fly and, once finished, be released so that the resources in Openstack can be freed for other projects. To achieve this, we've defined our instances as an Openstack Heat Template. The Heat template defines the networking, virtual machines, and other configuration options. A Heat template allows for the entirety of the cluster's infrastructure and software deployment to be defined as a text file. Openstack can then be directed to consume these directives and construct the cluster on an as needed basis.

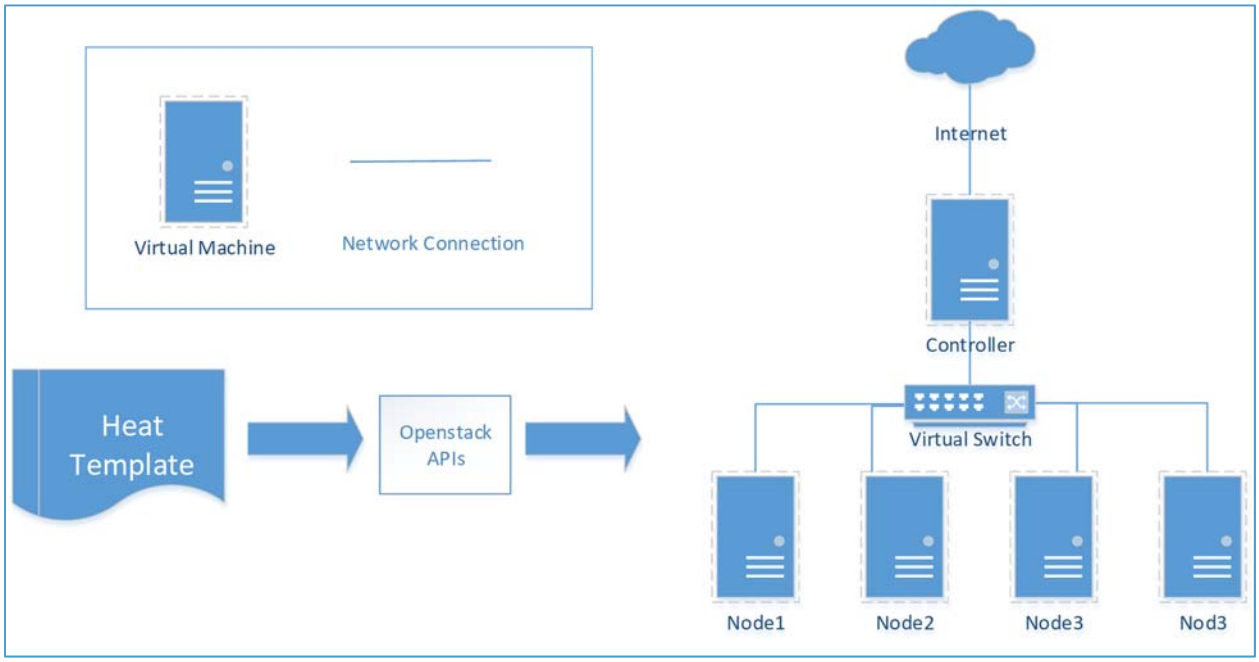


*Figure 3 - Heat Template Consumed through Openstack APIs, produces a "HPC Cluster" Instance.*

In Figure 3Figure 3, we utilize a Heat Template to generate the infrastructure for the project. The current heat template is configured to generate a controller node, and four worker nodes. The controller node hosts a number of other services for the worker nodes, including a Network Time Protocol (NTP) server to synchronize clocks for all nodes, routing and a Domain Name System service to facilitate access to the Internet, and an NFS shared file system. The shared file system is utilized so that all nodes can access the same set of data. A shared file system is also required to properly emulate an HPC system.

Using the Heat templates, we can configure user defined scripts that are run on the virtual machines in order to install and configure the software needed for the project. We utilize a number of Ansible templates in order to automate the installation of the cluster software and code needed in order to perform the experimentation.

When a cluster environment needs to be generated, the Heat template is sent through Openstack APIs referencing the Ansible playbooks. After heat has created a “stack”, a set of virtual machines and networking for a specific purpose, Heat runs the Ansible playbooks on these virtual machines within this stack. The nodes automatically have packages such as numpy [9], py4mpi [10], tweepy [11], etc. as well as the controller specific configuration item such as the NFS share. Additionally, our infrastructure also uses a Gitlab code repository to store the developed software for this project. The Ansible playbooks are also configured to pull that latest project software and install it within the cluster.

### 3.1.2 Cluster Stand Up and Tear Down Workflow

When a researcher wants to create a new cluster, the cluster Heat template is submitted to Openstack for infrastructure creation. Each researcher has a generated SSH Key for access to the

controller nodes and the NFS share is persistent storage. When a researcher has finished using the environment, the code is uploaded to the Gitlab code repository. A “stack delete” is performed and the virtual machines, virtual switches, IP addresses, etc. go back into the pool of resources for another user or future use. The researcher then updates the Heat template and Ansible Playbooks if they would like to automate the deployment of different or additional software and configuration, eliminating the need to setup manually when the environment is provisioned next time.

The Heat template was designed so that a researcher can have their own environment file containing their unique configuration items. The Heat template utilize the environment configuration as a set of parameters. With the environment file, the research can specify their own SSH Key limiting access to their system. Additionally, separate networks are generated for each cluster. This allows two researchers to generate two different HPC clusters, but be completely separate from each other. The researcher would not need to coordinate in order to utilize a single cluster, and eliminating the inherent issues with a shared environment.

### 3.2 Mock Workload

In order to perform our research, we built a workload to explicitly explore data and process integrity technology. We utilize a subset of the software available to HPC systems. The workload emphasizes HPC operations and shared storage to enable non-uniform distributed dataset loading and performing a number of analysis and data transformations. Our goal is to track the integrity of the final results which are derived from raw data introduced at the beginning and during the workload process. Furthermore, emphasis was placed on modeling the behaviors of HPC workloads at a reduced scale, not on the scientific rigor of the workloads and its results.

#### 3.2.1 Software

Many scientific workloads utilize MPI in order to use the massive number of nodes in a typical HPC cluster. We developed the workload using the Python scripting language due to its flexibility, ease of development, and ubiquitousness. Python has a module called py4mpi [10] that supports using MPI. The workloads utilize Python, while the inter node communication uses MPI, similar to existing HPC workloads.

#### 3.2.2 Data and Analysis

Much of the data required for analysis in HPC systems could range into several hundred gigabytes, to petabytes, and higher. Additionally, the computational overhead to process that data was designed with HPC systems in mind. With a resource restricted environment, our team decided that the analysis to be performed maximizes the resources available in our HPC environment, and the nature and amount of the data is simple and small enough to be processed.

Our workload consists of sentiment analysis of tweets from Twitter. We chose the use of Twitter data because a large dataset can be derived for testing by simply saving all tweets produced over a series of topics. Additionally, the data frequently contains the “sentiments” of millions of people, so it makes it ideal for sentiment analysis. Furthermore, datasets of varying sizes over varying topics can be produced.

### *Sentiment Analysis*

The basic premise of sentiment analysis a set of text, such as that which would be found in a tweet, is broken down and its parts usually words or n-grams. A relationship can be established between sets of words only or typically found in negative comments as well as the same relationship between certain words for positive comments.

Words such as “good”, “great”, “awesome” typically have a positive connotation similar to “bad”, “awful”, “disgusting” have a negative connotation. A model can be built in a way where parts of a set of text can be generally associated with the sentiment, whether it’s negative or positive. To build this model we use the “Large Movie Review Dataset” from Stanford (http://ai.stanford.edu/~amaas/data/sentiment/).

The movie review dataset has several thousand movie reviews each labeled as either positive or negative. Using this data, each review can be broken down into its components and used to train a model. Ideally, components closely associated with bad reviews are be negative, while components closely associated with good reviews are be positive.

### 3.2.3 KMeans Model

Our team fully admits that the KMeans Model for clustering is not the most effective model for assessing sentiment. The generated models don’t fit well, but we chose KMeans clustering as it will be computationally light enough to run on our cluster, and runs within a reasonable time. We are more concerned with loading raw data and transformations to generate integrity events for tracking rather than spending hours or days to compute more accurate sentiments.

A feature vector is constructed from each of its components and how often it appears in that review. Words common to all reviews such as “the” would have a high occurrence in both sets of reviews, while “bad” would only have a high occurrence in negative reviews. Using KMeans, the model consists of clusters of negative, positive, and neutral components. Once the cluster of these words are established, each tweet is broken down into user selected sized n-grams. To simply the classification, a tweet’s sentiment is calculated by word counts associated with the positive cluster and subtracting the number of occurrences closest to the negative cluster. Simply stated, if a tweet has more negative components than positive, then its sentiment is negative and vice versa.

### 3.2.4 Twitter Data

To facilitate the capture of twitter data, we designed a Python developed service that runs over a period of time to collect tweets and saves them to files. This service, called twitter_hose, consumes a number of different configuration directives that can vary what and how data is collected:

- **File Size** − Tweets are be saved deliminated within a single file. The file size indicates the maximum amount of bytes that file occupies before a new file to save tweets is created. This allows us to vary the file sizes of the data set.

- **Topics** −Allows us to specify the topics that that the tweets are associated with. Allows us to examine the sentiment in specific topics.

### 3.2.5 Workload Process

Our process consists number of steps which utilize controller to pull information, followed by distributed feature extraction and analysis. The process is as follows:

#### *Step 1*

The Stanford movie review data is downloaded to the controller, extracted, and saved in the NFS shared storage.


*Figure 4 – Process Step 1 – Dataset download*

#### *Step 2*

Once the move review dataset has been extracted to the NFS shared storage, the controller gets a list of all the movie review files. In this dataset, each review is a separate file. These files are stored in task/job queue on the controller. The controller first processes all the positive reviews as they exist a separate directory from the negative reviews.

The worker nodes are placed in a ready state, and the controller sends them each a file name to process. Each worker node reads that file and build a feature vector using n-grams. The n-gram becomes the key, and the value is how many times that n-gram appears in that review.

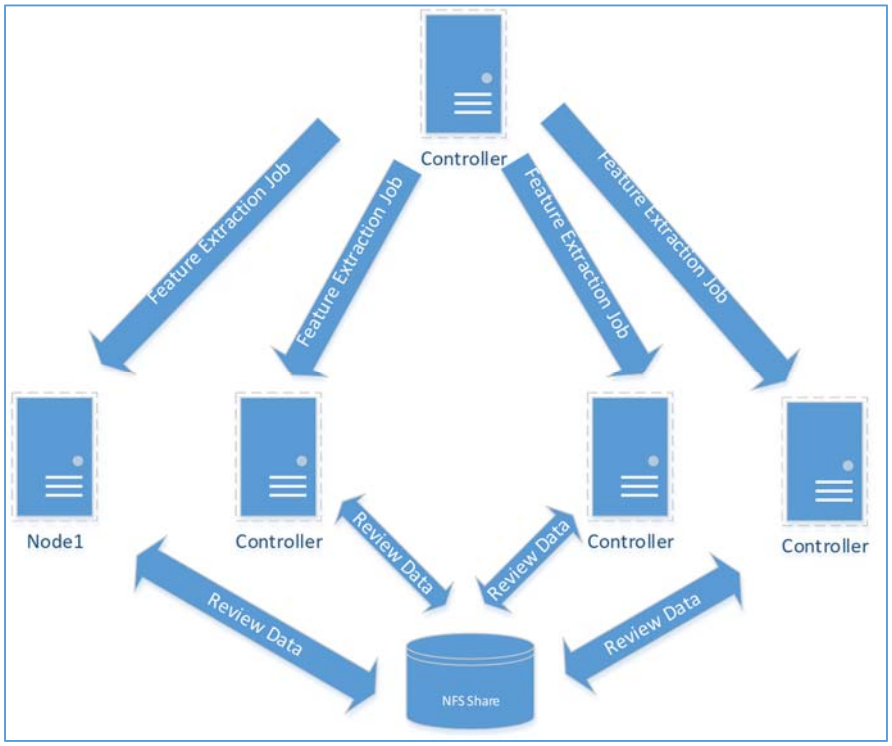


*Figure 5 - Feature Extraction Distributed to Worker Nodes*

Once a worker has finished constructing a feature vector, it is placed in the accumulator. The node then receives a new job, create a new feature vector, and then add it to the accumulator. The end result is that after all jobs are complete, each node has a dictionary of all n-grams observed, and how many time. Each worker node sends back their accumulated feature vector and they are be combined in one large feature vector.

The end result is two feature vectors: one containing all of the n-grams and occurrences in positive reviews, and the same for negative reviews. The point of this work is to distribute the feature hashing the same way data might be read and processed in an HPC system.

### *Step 3*

Once the feature vectors have been created, they are used to create one large set of coordinates to use for KMeans clustering. A total list of all n-grams is constructed and assigned an index. That index corresponds to the index of coordinates to be sent to be analyzed via KMeans. Each coordinate consists of an X and Y where the X access corresponds to the number of times an n-gram has been found in the positive set, and the Y access is for the n-gram occurrence in the negative data set.

Once the coordinates are assembled, they are then split among all nodes using a scatter operation. Each node evaluates the distances between each point and the centroids. Using the distances, the coordinates are associated to a centroid (the center point of a cluster). All distances have been calculated, they are collected via a gather operation to the controller. An array containing all distances are constructed, and the scattered back out to all nodes.

Each node calculates the mean of all distances to the closest points to the centroids get. These averages are collected by the master and an average is calculated from those averages. The average distance of all points considered closest to a centroid becomes the new coordinates for that centroid. Determining which points are closest to which centroid, and then averaging their distances to get a new centroid location is considered a single iteration. The process repeats, including the scattering, processing and gather of data from all nodes in the cluster, until the specified number of iterations has occurred. At this point, the model is complete.

Theoretically, n-grams that appear significantly more in negative review vs positive reviews have a larger Y, but a small X, and the opposite would be true for positive reviews. Words seen equally in both form a line of positive direction from (0,0) to the coordinates of n-grams with the highest occurrence between both positive and negative reviews.

#### *Step 4*

We utilize twitter_hose to construct a data set of tweets. Topics are selected based on the sentiment to be analyzed.

#### *Step 5*

After the model is constructed, the tweet data set is parsed in the same fashion as the review data. Each tweet is then being broken down into n-grams and a comparison as to how many of those n-grams are associated with each cluster of words.

### 3.2.6 Software

- **mpi_kmeans/kmeans.py** − this contains all the code used to manage the Kmeans model building. It has all the necessary code for the nodes to perform distance and average calculations.
- **mpi_kmeans/sentiment.py** − this contains all the code needed to create the feature hashes of data based on n-gram.
- **mpi_kmeans/scheduler.py** − This contains all the code to distribute and execute the feature hashing on multiple files over multiple nodes.
- **project1/kmeans.py** − The current main program that arranges all steps of the workload, from dataset download, to final comparison of tweets.
- **twitter/twitter_hose/service.py** − The twitter_host service command.

### 3.2.7 Discussion

We fully admit that the KMeans model used in this work is less than ideal for sentiment analysis. Our goal was to develop workloads that simulate what occurs on HPC systems but at a smaller scale on resource constrained hardware. We are more interested in making the nodes gather data and do work, and not the validity of the learning models produced by that work.

The current version of the software has a number of bugs and other oddities. As an example, the array containing coordinates for KMeans model construction must be divisible by the amount nodes that operate on it. A workaround was used to drop enough features to be divisible by the

number of nodes performed. While this is suboptimal, we are not interested in faithfully performing KMeans modeling, but rather mimicking its workload.

### 3.3 Merkle Hash

When verifying the integrity of the computations for a large-scale data analysis requires verifying the data has not been tampered with. This is typically achieved using a Hash function. A hash function can take data of any length and produce a corresponding output of fixed length called a hash.

Cryptographic hash functions produce output that is collision resistant: the output hash vastly changes when the given input is only changed slightly. Collision resistance significantly increases the difficulty in predicting output based on previous differing output. By producing a hash of a data file, it's possible verify that the data has not been tampered with, as producing a hash of the tampered data yields a vastly different output hash.

An issue arises when dealing with very large file sizes. Files in the gigabytes, terabytes, or even petabytes significant amount of time in order to perform an entire hash. Checking the integrity would require processes to wait for the hash function to run on entire file before even a small piece of that file can be used in computation.

#### 3.3.1 Merkel Tree

In order to verify the integrity of a file of any size, in an efficient fashion, a Merkel Tree is used. A Merkel Tree has two major advantages: A file is split into blocks which are each individually hashed allowing this process to be distributed and it's possible to verify individual blocks rather than the entire file at a time.

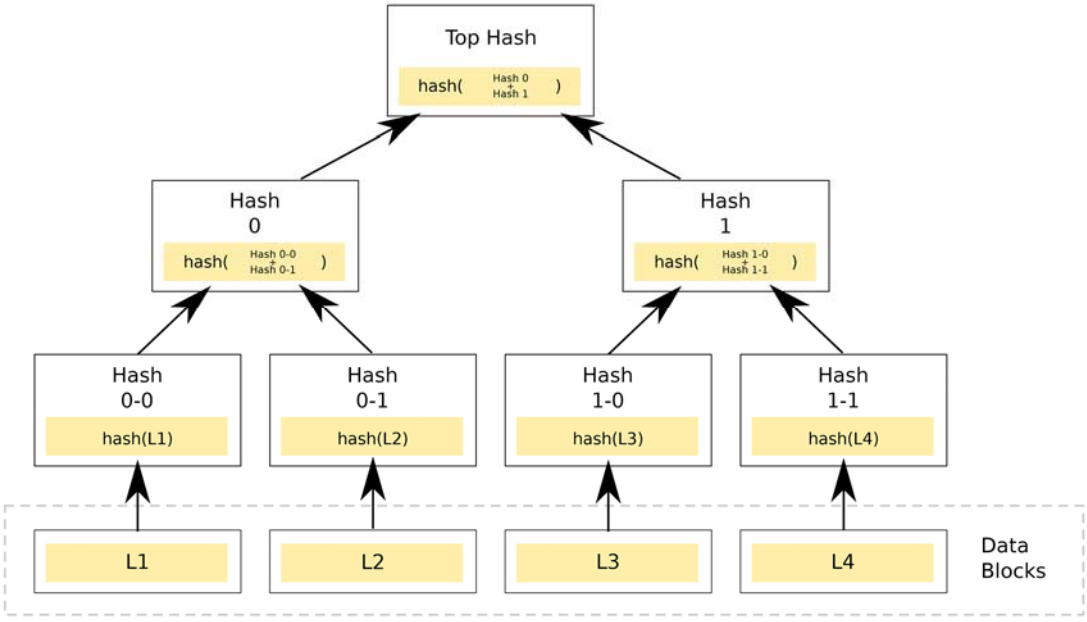


*Figure 6 - Merkel Tree (David Göthberg / Wikimedia Commons / Public Domain)*

Figure 6 provides an illustration of a basic Merkel Tree. For our use case, each file is be split into blocks of specified length. Each block is fed as input into a designated hash function. Once each block is hashed, the hash from the first block, and the hash from the second block are concatenated and used as input into the specified hash function to produce another output hash. This repeats for each pair of blocks. If there are an odd number of blocks, the last block's hash is used and is not concatenated with another block. This process begins again with the new concatenated hashes to produce a new set of hashes. This process ends once there is only one hash remaining: the Top Hash.

To limit the data necessary to verify a large file, verification can occur using only the Top Hash. The Merkel hash processes is performed again on the data, and the produced Top Hash can be compared to the known Top Hash to determine if the data has been altered. However, if some or all of the hashes for each block and concatenated hashes is available, it's possible to check the integrity of individual blocks. While much faster and more accurate in determining which piece of data may have been altered, the tradeoff is that significantly more hashes need to be stored compared to the single Top Hash.

### 3.3.2 Implementing the Merkel Hash

The Merkel Tree functionality has been integrated into the MPI workload library as the *hasher* functionality. Hasher provides two capabilities: to perform a Merkel on a file singularly and perform a distributed Merkel Hash using the MPI workload library scheduler. Both operate in similar fashion and require similar parameters:

- **Algorithm** − The algorithm is the hash function. Any hash function provided by Python's hashlib is supported.
- **Block Size** − The size in bytes of a block. When a file is divided up, the size of the file and the size of the block is determined the number of blocks to be used in finding the first set of hashes.

Once a Merkel hash has been executed, the file size is retrieved, and using the block size the following formula is used to get the number of blocks (file size) / (block size). The non-whole number remainder is considered a block where the size of the last block is less than the block size. As an example, a file size of 10 with a block size of 3 yields 4 blocks. The first three blocks are 3 bytes each, while the final block only have 1 byte.

Once the number of blocks has been found, the file is divided into a list of ranges called the block list. The block list is the index of the bytes in a block within the file itself. Continuing from our previous example, the first block using a block size of three bytes would be byte 0 through byte 2, the second block would be 3 through 5, the third 6-8, and the final block would only be byte position 9 which is the tenth byte.

### 3.3.3 Discussion

Using the local Merkel Hash Functionality allows the MPI workload functionality to perform hashes on local files on a singular system. It accepts the above parameters as well as the file

name of the file to be hashed. The block list is then created and the file is parsed one block at a time. For files in gigabytes on a reasonably modern system, takes less than a minute to produce a top hash.

### *Distributed Merkel Hash*

In the MPI workload functionality, a distributed Merkel Hash workload was created using the Scheduler functionality. It accepts the same parameters as the Local Merkel Hash, however, rather than accept a single file, it accepts a list of files. Scheduler Jobs are defined so that an instance of a Local Merkel Hash object is created for each file. These objects are then given to worker nodes to perform the hash of each file and return the result to the scheduler. The final result produced is a dictionary of file names as keys and their corresponding top hashes as the values.

### *Inherent Weakness in Distributed Approach*

The major weakness with the described distributed approach is that if there are large asymmetries in file sizes. In one case, the task of hashing an entire list of files can be bottlenecked if one file is larger than all others combined. Hashing all files takes at least as long as the hash of the single large file.

### *Storage on the Blockchain*

Implementation of a Merkel Tree is not inherently novel as it's used in a wide variety of protocols such as BitTorrent, Bitcoin, Ethereum, etc. This module was constructed to test how the Merkel hash could be stored within the blockchain to aid with integrity checking. The module can be modified to work with an existing implementation of blockchain so the same workloads that download raw data, and generate derived data as part of their processes can both produces and check hashes of that data. A number of approaches have been considered:

- **Merkel Hash Header** − There are two key parameters to confirming the integrity of a file in order to reproduce the top hash: the hash function and block size. This information would be saved as a transaction on the blockchain along with the file name.
- **Hash as a single transaction** − The top hash and the Merkel Hash Header would be saved as a single transaction on a blockchain. The main advantage is that only the minimal amount of information is saved in the blockchain, reducing the potential size and growth as more data integrity information is added. The major downside is that it becomes impossible to check the integrity of a single block. The entire file would need to have the Merkel Hash process performed to check integrity.
- **Hash as a set/block of transactions** − Each part of the Merkel Hash is recorded on the blockchain. The first set of transactions would be the filename, and file index of a specific block. A second set of transactions would be the hash of these blocks and the transaction IDs of those blocks. Each subsequent set of transactions would be the hash produced from the concatenation of two previous hashes and their transaction IDs. This process would culminate until a final transaction similar to the single transaction hash,

but also containing the transaction IDs for the two previous transactions containing the pair of hashes used to produce the top hash.

While a single transaction to publish the top hash of the file takes up significantly less space than storing the whole Merkel hash as a set of transactions, the major advantage to the latter approach allows checking the integrity of individual blocks rather than having to perform the hash for the entire file. Rather than specifically target either method, the method selected should be workload, or result dependent.

Depending on the integrity requirements required of a result for computation, one, the other, or a combination of both may be necessary. Certain security or integrity requirements could require the full set of transactions, while processes or results that require less integrity can use single transactions. Additionally, being able to check the integrity of individual blocks can provide processes the information necessary to take actions to self-heal, such as reissuing the same computation on a different system where the data passes integrity checks.

### 3.4 Blockchain Software Evaluation

We leveraged an existing blockchain platform in which to build our solution. There are a number of other technologies available for private blockchain development including Hyperledger [12], Openchain [13], Ethereum [14], etc. We evaluated the feasibility of these options for use in our solution, and while, initially, Openchain appeared to be viable candidate, after further investigation, we found the available documentation, tools, and relative maturity of Ethereum fit our needs.

#### 3.4.1 Ethereum

Ethereum, similar to Bitcoin, is a public blockchain which also uses a cryptocurrency of the same name. It differs from Bitcoin in that it facilitates the running of distributed applications called smart contracts [15]. For Ethereum, smart contracts are Turing complete scripts that run as part of the mining the transactions. If a transaction to be placed in a block for mining is associated with a smart contract, the code is run on the miner itself, and the output also recorded to the block. Since the transaction may be verified by a number of different miners, the code can potentially be run many times.

Ethereum comes with a number of compelling features. Unlike other blockchain platforms such as Openchain or Hyperledger is that there exists a well-supported, public blockchain in significant use. While initially, we expect our solution to use a gated, private blockchain, moving the product from the private blockchain to the public Ethereum blockchain relatively less difficult. Assuming we are successful in the implementation of it would be possible to move the integrity information from a private blockchain, to the public blockchain. This would eventually support the sharing of integrity information and automatic checks between organizations. Integrity information

##### *Smart Contracts*

One of the major advantages to using Ethereum over software such as Openchain is that Ethereum has had support smart contracts since its inception. Smart contracts are Turing

complete scripts that run as part of the mining the transactions. If a transaction to be placed in a block for mining is associated with a smart contract, the code is run on the miner itself, and the output also recorded to the block. Since the transaction may be verified by a number of different miners, the code can potentially be run many times.

#### *Ether*

Running smart contracts and mining requires what the Ether. Similar to Bitcoin, Ether is also cryptocurrency. Running smart contracts and transaction mining requires computational resources. The miners that provide these services are paid in ether. A smart contract will require a certain amount of "gas", ether needed to run the smart contract. The major advantage to accruing ether is that you can provide gas for your own smart contracts. Additionally, there are exchanges which will allow you to trade in ether for cash. If smart contracts are intended to be run on the public Ethereum blockchain, purchasing or mining ether will be required.

When running Ethereum as private network, how much Ethereum is generated, and the ease at which new Ethereum can be mined is configurable. When creating a new private chain, ether can be configured to be generated in quantities that largely remove the need to accrue it. This allows for experimentation and development without any fear of wasting ether.

#### *Tokens*

Another major benefit to using Ethereum is that it natively supports the generation of custom tokens. Distributed applications utilizing smart contracts can be setup to generate, impart, and consume their own set of tokens. It's possible to construct a separate cryptocurrency that utilizes Ethereum. Technically, the tokens are generally consumed by the distributed application (DApp) that produces them, so they can't directly be used as gas. Ethereum has a standard for specifying a set of functions to generated tokens called ERC20.

#### *Running an Ethereum Node*

Ethereum nodes are that participate in mining operations and running smart contracts use Geth to run the full Ethereum node software. Geth is the primary console in which interact with the Ethereum client. With Geth you can mine ether, transfer funds, create smart contracts, and explore transactions recorded on the blockchain. Geth can be used to create a private or custom blockchain. A custom “genesis block”, first block in a blockchain, can be defined and with the characteristics such as gas limits or the difficulty in mining new ether

### 3.4.2 Discussion

The major benefit to using this platform is two-fold. As we’ve mentioned, the relative maturity software and documentation simplifies the development and deployment of the platform. We’ve successfully deployed a local Ethereum instance and have written a number of basic smart contracts which accept input and produce output stored on the private blockchain.

The other major benefit is the ability to leverage Ethereum’s public blockchain. While expect our solution, in the early iterations, to make use of the ability to create a public blockchain. However, in the solution can be extended so that the transactions and integrity information is available globally, the solution would require little adapting to use the public blockchain.

An additional benefit is the ability to generate tokens. There are a number of platforms that exist on the Ethereum blockchain which produce and consume their own tokens. Tokens could be used as an incentive as part of our solution. For instance, if we encoded integrity requirements for an analysis that includes being run by another user or organization, this behavior could be influenced by providing those who voluntarily run the same analysis to confirm results by providing tokens they could use or sell to other organizations. Tokens would provide the means to offer some sort of monetary incentive to participate in the platform.

### 3.5 Blockchain storage of Integrity Data

Our investigation into saving integrity data was not as comprehensive as we would have liked. While we were able to create a number of rudimentary smart contracts to receive input such as integrity data, however, we only started the preliminary work needed to build in the ability to send this information to a private Ethereum.

#### 3.5.1 Discussion

We started investigating how we were to integrate our workload with the Ethereum private blockchain. The primary mechanism to send and receive data from a smart contract is through JSON-RPC calls. There exists a number of libraries that would facilitate communication from Python applications to JSON-RPC, however, there is an Ethereum project called web3.py [16] that provides much of this functionality.

Using web3.py it's not only possible to interact with existing smart contracts, but web3.py will take contracts written in Solidity, compile and then deploy them on the blockchain. Once a contract has been placed on the blockchain, the contract is given a specific contract address. With the address, applications using the web3.py will be able to send data to the requisite contract and receive any relevant outputs. This interaction is then recorded as a set of transactions on the blockchain. For integrity data to be saved on the blockchain, several sets of smart contracts will need to be created and placed on the blockchain to provide the necessary functionality required. Additionally, each of the smart contracts may need input or output from other smart contracts so functionality may require facilitating chaining of smart contracts together to provide more complex behavior.

## References


[1] S. Akioka and Y. Muraoka, "HPC benchmarks on Amazon EC2.," in *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*, Perth, Austrailia, 2010.

[2] H. Chen, R. H. Chiang and V. C. Storey, "Business Intelligence and Analytics: From Big Data to Big Impact," *MIS quarterly,* vol. 36, no. 4, pp. 1165-1188, 2012.

[3] M. Minelli, M. Chambers and A. Dhiraj, Big data, big analytics: emerging business intelligence and analytic trends for today's businesses, John Wiley & Sons, 2012.

[4] M. A. Waller and S. E. Fawcett, "Data science, predictive analytics, and big data: a revolution that will transform supply chain design and management.," *ournal of Business Logistics,* vol. 34, no. 2, pp. 77-84, 2013.

[5] S. L. Yogesh, B. Plale and D. Gannon, "A framework for collecting provenance in data-centric scientific workflows.," in *IEEE International Conference on Web Services (ICWS'06)*, Las Vegas, Nevada, 2006.

[6] W. Gropp, E. Lusk, N. Doss and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard.," *Parallel computing,* vol. 22, no. 6, pp. 789-828, 1996.

[7] T. White, Hadoop: The Definitive Guide (1st ed.), O'Reilly Media, 2009.

[8] A. M. Antonopoulos, Mastering Bitcoin, Sebastopol, CA: O'Reilly Media, Inc., 2015.

[9] S. van der Walt, S. C. Colbert and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering,* vol. 13, no. 22, pp. 10-20, 2011.

[10] L. Dalcin, "MPI for Python," 8 November 2017. [Online]. Available: http://mpi4py.scipy.org/. [Accessed 23 1 2018].

[11] J. Roesslein, "Tweepy Documentation," Read the Docs, 2009. [Online]. Available: http://docs.tweepy.org/en/v3.5.0/. [Accessed 23 January 2018].

[12] The Linux Foundation, "Hyperledger," The Linux Foundation Projects, 2017. [Online]. Available: https://www.hyperledger.org/. [Accessed 23 January 2018].

[13] Coinprism, "Openchain," Coinprism, 2015. [Online]. Available: https://www.openchain.org/. [Accessed 30 September 2016].

[14] Ethereum Foundation, "Ethereum," Ethereum Foundation, 2018. [Online]. Available: https://www.ethereum.org/. [Accessed 23 January 2018].

[15] N. Szabo, "Formalizing and Securing Relationships on Public Networks," *First Monday,* vol. 2, no. 9, 1997.

[16] P. Merriam and J. Carver, "Web3.py," Read the Docs, 2017. [Online]. Available: http://web3py.readthedocs.io/en/stable/. [Accessed 23 January 2018].