DFS^3: Automated Distributed File System Storage State Reconstruction

FirstName Surname[†]
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

ABSTRACT

Distributed file systems present distinctive forensic challenges in comparison to traditional locally mounted file system volume. Storage device media can number in the thousands, and forensic investigations in this setting necessitate a tailored approach to data collection. The Hadoop Distributed File System (HFDS) produces and maintains partially persistent metadata that is pursuant with a logical volume, a file system, and file addresses on the centralized server. Hence, this research investigates the viability of using a residual central server digital artifact to generate a history model of the distributed file system. The history model affords an investigator a high-level perspective of low-level events to narrow investigative process obligations. The model is generated through set-theoretic relations of the file system essential data structure. Graph-theoretic ordering is applied to the events to provide a history model. The research contribution is a rapid reconstruction of the HDFS storage state transitions generating timelines for system events to forensically assess HDFS properties with conceptual similarity to traditional low-level file system forensic tool output. The results of this research provide a prototype tool, DFS^3, for rapid and noninvasive data storage state timeline reconstruction in a big data distributed file system.

CCS CONCEPTS

• Applied Computing • Computer Forensics • Data Recovery

KEYWORDS

Digital Forensics, Reconstruction, Formal Modeling, Automation

1 Introduction

Digital forensics is a broadly-applied term referring to the identification, acquisition, and analysis of digital evidence [1], [2]. The general process can be defined as the identification of digital evidence using scientifically derived and proven methodology. The process undertaken is crucial to the investigative outcome. Ultimately, evidence credibility is the essential element of an investigation. The traditional means of performing digital forensics follows four primary phases: Collection, Examination, Analysis, Reporting [3]. The collection phase is typically reduced to digital evidence search and recognition.

The digital forensic process model has undergone significant modifications and adaptations to accommodate the rapidly changing computing landscape. Despite the numerous changes and varied proposals to address an increasing volume of data, very little has been published about methods of applying techniques to facilitate efforts of evidence collection reduction in a distributed environment.

Distributed File Systems (DFS) often provide storage repositories for the cloud environment. Attackers and academics are actively investigating scenarios to abuse or attack cloud environments [4], [5]. This activity prompts researchers to investigate the ability to recognize and examine attacks in cloud environments as well as understanding the legal implications associated with cloud investigations [6]–[14]. One of the chief difficulties in the digital reconstruction of a complex system is induced by the many rapidly changing variables and multiple abstraction layers that enter into every single operation [15]. Event materialization techniques take the high-level events and attempt to deduce lower-level events. The typical forensic reconstruction process moves bottom-up [16].

The objective of this research is to automate the generation of a storage state transition timeline in the DFS using prototype software, Distributed File System Storage State (DFS^3). To extract information that is typically made available later in the sequenced investigative process, DFS^3 emulates low-level methods of file system forensic examination and analysis tools as an initial enhanced preview of the file system. It is preferred to do this without having to access the individual files and datanodes. In this paper, we present an approach to apply causal ordering and Event Calculus (EC) [17] reasoning to identify and verify file system operations using a single DFS central server image file. The history reconstruction is made more efficient by namespace partitioning through topological ordering, and properties of the data units are shown to persist over intervals.

The structure of this paper is as follows. Section 2 provides background on distributed file systems. Section 3 discusses relevant work in forensic reconstruction. Section 4 chronicles the methodology. Section 5 presents the formal modeling of the system and the generation of facts and rules. Section 6 discusses the analysis of the findings. Section 7 draws conclusions from the research and discusses future work.

2 Background

Cluster-based distributed file systems (e.g., Google File Systems (GFS), Lustre File System, and Hadoop Distributed File System (HDFS)) often have a single master server with multiple data servers [18]. The single master server controls the data servers by having data management, and metadata management separated [19]. The file system is ordinarily object-based, allowing for unstructured data supporting a flexible schema-on-read implementation. Therefore, data is applied to a schema as it is retrieved from a stored location rather than when written, although many distributed file systems support schema on write for a specific application.

Files are distributed over data servers that handle the read and write operations. The master server, a meta-data server, maintains the directory tree and manages the data placement. This architecture allows for incremental scaling, and the capacity of this system is a function of the meta-data load. As the load increases, the metadata server must be able to manage the additional data servers as extensions of the file system capacity. A DFS with only a single metadata server is called centralized, whereas a DFS with distributed metadata servers is totally distributed [20].

Traditionally, in digital investigations, the file system media is collected, imaged, and preserved to maintain its integrity. A forensic image in this context is a bit-by-bit copy of data on the storage volume [21]. Once the evidence is identified and collected, file system examination and analysis methods are applied. Forensic tools are utilized to examine and extract information. Many tools exist for file system information extraction and analysis, but all operate on the notion of a volume image to examine. Many of the examination tools have built-in analysis capabilities. It's in this analysis phase when reconstruction of file system events occurs typically.

The centralized distributed file system is unique from traditional file systems because the metadata is not co-located with the content data. The traditional investigative process and current tools are not conducive to a distributed file system investigation. The delay from start to finish is prohibitive in all phases, from collection to analysis. Exhaustively collecting or imaging disk drives is not a good option. Trying to analyze the low-level file system for every drive in the system would be counterproductive and induce extremely high latency in the process. Imaging and extracting information from only the central server drives would not provide meaningful information to survey the system rapidly. However, the central server provides a natural starting point to emulate the low-level tools given it's the metadata manager for the entire centrally managed distributed file system. The central server uses a specific file for system restarts, persisting partial point in time file system metadata.

3 Related Work

The reconstruction of digital events is traditionally considered a performance of analysis using varied collected evidence [16], [22]–[25]. Most traditional approaches assume the lower level evidence identified and collected. There have been a relatively small number of attempts to apply formal methods to the reconstruction of digital events [26].

There exists a collection of research addressing the areas of automating heterogeneous data evidence extraction and event correlation. Chen et al. [27] developed the Event Correlation for Forensics (ECF) as a means by which a consolidated repository of data evidence is created from various log file structures. The repository can then be queried for post hoc event correlation. Necessary information is captured in 4-tuple event abstractions (Time, Subject, Object, and Action). Schatz et al. [28] developed Forensics of Rich Events (FORE) to store events in an ontology. They endeavored to explore methods of forensic investigation of heterogeneous event log based records. The methods included a human-guided search, automated correlation, and hypothetical reasoning.

Hargreaves and Patterson [25] used an approach to search for patterns of events in the low-level timeline based on pre-determined rules. The method consists of two phases: low-level event extraction and high-level event reconstruction. The generation of low-level events includes file system times, and times extracted from within files by the analysis of a mounted file system. This approach strives to develop high-level events; however, the method relies on accumulated low-level data evidence.

Formal modeling for digital event reconstruction has generally taken the approach to model the system as a transition system [26]. The most common method is the overall notion of a Finite State Machine (FSM). Gladyshev and Patel [29] modeled a hacked system as an FSM to explore the possible scenarios leading to the hacking incident. They utilized a back-tracing of transitions from the discovered state of the system, and then discarded scenarios which did not agree with discovered evidence. Gladyshev and Patel presented findings of an automated search of the state space for a simple print program given the knowledge of the system functionality, which included 25 states and 75 possible transitions. One drawback to their method is the model doesn't provide a solution for the state-space explosion in a more realistic system, although the authors believe the model could be utilized for reconstruction with an appropriate choice of models.

James et al. [30] presented a novel approach to formally defining the system as an algorithmic representation of a Deterministic Finite Automaton (DFA). The system computations are encoded as sets of strings mapped as an FSM. They use a propriety witness statement to restrict possible computations of the FSM by restricting the possible events of the model by applying known observations. This helps to limit the state space explosion, albeit limited and augmented by witness statements. FSM modeling presents challenges in scoping the state space, specifically with complex systems. Generalizing an FSM and determining the appropriate level at which to model with practicality to digital forensics has yet to be realized in general [26].

Khan and Wakeman [31] proposed a neural network-based event reconstruction of application activity from disk image input parameters. The activity parameters included log files, registry entries, file system properties, and free blocks on disk. Evidence must be derived from events within the log file, temporary files detected either directly on the file system, or by searching through the free blocks of the file system. As the reconstruction moves further into the past, the evidence becomes much less reliable.

Willassen [32] uses the Simplified Event Calculus (SEC), a form of propositional logic to reconstruct digital events from observed states using hypotheses about actions. In that work, a simple file system model is presented, which demonstrates the resolution of observed states by means of Selective Literal Definite clause with Negation as Failure (SLDN) resolutions. Willassen extends his theoretical approach to forming and testing hypotheses about actions and deriving system invariants and concludes SEC could be a reasonable tool for system model building and property determination, but the approach in his work is purely theoretical.

The complexity and search space of many of the formal models reviewed make these approaches ineffective or impractical for the modeling of the DFS [33]. With respect to the other methods that appeared in this review, they have several qualities in common, making them inappropriate for the formal modeling of the DFS. First, they require the collection of low-level data evidence or log file extraction to reconstruct event timelines. The latency induced to piece together and analyze voluminous log data or extract low-level file system data is prohibitive given the architecture of the DFS. Second, some of the methods mostly lack the formal theoretical foundation behind the automation of the event extraction. Although machine learning appears to be a viable solution in a specific context, neural network reasoning can be unclear and not explicit, which could limit evidentiary value to some degree. If there is no understanding of the process behind the actions to infer events and make interpretations, the validity is questioned.

4 Methodology

The method chosen is to extract information from a central server image file. The process of reconstructing file system operations normally flows bottom-up as we described, but we'll use a top-down approach and work at the higher abstraction layer. At this high-level layer of abstraction, we adopt many of the common concepts that a low-level file system forensic tool would address. We use those concepts and build structures like master file tables that are extractable on lower-level file systems.

The HDFS was formatted, and we started with a clean cluster volume and performed file system operations on the cluster over a period of months. A series of central server images were collected during the process over a time period of several months, and each image is a partition of the final image space. The cluster consisted of a central server and three data nodes in a fully distributed mode.

A prototype software tool, DFS^3, has been developed to demonstrate the model construction and reasoning. The model is built on the data structure that is essential to the core purpose of the HDFS (3.2.0) consistent with low-level approaches. Logic programming is ideal for modeling any sort of knowledge and its use. Modus ponens (if/then) rules make available understanding of the hypotheses linking cause (antecedent) and effect (consequent) assertions in a digital investigation. This is comparable to the approach of Hargreaves and Patterson [25]; however, this work does not utilize accumulated low-level artifacts for the high-level reconstruction. We employ logic statements to describe the properties and behavior of the domain being modeled and generate event occurrence hypotheses. The timeline is innately made in the temporal logic of the system data structures.

There exist various techniques to acquire or generate the central server binary image file, and we omit that discussion and assume the image has been acquired upstream through a sound forensic method. The binary file is converted to XML file format as input into our Python 3 prototype software DFS^3.

5 Modeling the Centralized DFS

HDFS presents inode, block, and generation stamp variables as integer types. HDFS is mainly developed in the Java programming language, and the source code is readily available through Apache Software Foundation (ASF). The Java long data type is a 64-bit two's complement integer. The Java programming language is statically typed, stating the variable's type and name, which means that all variables must first be declared before they can be used. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of $2^{64}-1$. [34]. This subset of integers forms the value set from which the inode, block, and generation stamp variables are permitted to hold. This subset of permitted integers is a subset of the positive integers. The values can be quantified over the *natural numbers*, and that is the domain of discourse.

5.1 System Properties

It is convenient to leverage natural number properties in the formation of the prescribed notation. This includes the concept that for a finite set of positive natural numbers (integers), a total (well) ordering exists. The HDFS uses the Unix-style inode concept of a file. There exists an inherent causal ordering in the structure that can be exploited to reason about the file system event sequences.

The in-memory representation of the directory/file/block hierarchy is kept in a base class, *Inode,* containing common fields for files and directories. When an inode is generated, the inode is assigned an ID, uniquely identifying the inode. Inodes are allocated sequentially from an initial static value, LAST_RESERVED_ID+1, in the *InodeID* class. The initial value is assigned to the root directory as 16385. The primitive Java type *long* is used to represent the inode ID. The inode ID won't be recycled and is not expected to wrap around for a very long period. File inode IDs are immutable through all subsequent file operations, including location changes (data transfers) and renaming operations. File inodes contain chunks of data in the form of blocks. Block IDs are generated sequentially as immutable keys identifying blocks and generation stamps as an operation sequence. Figure 1 depicts the relationships of the inode, block, and generation stamp structures in HDFS. File inodes include blocks, and blocks include generation stamps. These object relations, their orderings, and the values they hold present complex event causes and effects.

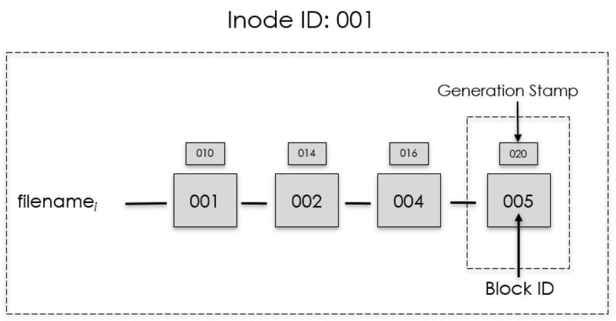

Figure 1: HDFS Inode Structure

Inodes, blocks, and generation stamps are essential to the metadata management of the central server. These unique identifiers are generated as monotonic sequences of integers, and the instantiations can be abstracted and treated as ordered sets of integers. The operations affect the allocation of object sets in the abstraction. The structure is summarized into sets with file operations equating to set operations. For the remainder of this paper, inode and data block references are the high-level HDFS abstractions.

It can be established that the set of blocks B in the namespace is totally ordered with each $b \in B$ represented by a unique integer value. By definition, there exists a least element and greatest element from a subset of B. Suppose $b_1, b_2 \in B$. Let $B' = \{b_1, b_2\} \neq \emptyset$. By hypothesis, B' has a smallest element. If it is b_1, then $b_1 \leq b_2$, and if it is b_2, then $b_2 \leq b_1$. The same logic applies to a set of generation stamps G' in the namespace. Generation stamps G and blocks B form relations since generation stamps are created if and only if (*iff*) block operations occur. Within HDFS, the following block operations rules affect the generation stamp to block relation in precisely the manner described:

1. OP_ALLOCATE_BLOCK_ID: creates a new block to write data. Adds one block and one generation stamp to namespace pools with OP_SET_GENSTAMP, OP_ADD_BLOCK
2. OP_DELETE: removes data block. Removes one block and one generation stamp from namespace pools with OP_DELETE
3. OP_APPEND: appends data to end of file; the last data block within file Inode. Overwrites previous namespace generation stamp with OP_SET_GENSTAMP
4. OP_TRUNCATE: removes data from the block. Overwrites previous namespace generation stamp with OP_SET_GENSTAMP

From the block operation rules, we define three data block content properties that can be used to determine consistency for 'events' in the system. Observed events are those explicit in the discovered namespace.

- Events are considered 'original' events when rule 1 is the cause of the observed event
- Events are considered 'deleted' events when rule 2 is the cause of the observed event. Hypotheses events are generated to infer rule 1 'original' events from rule 2 events
- Events are considered 'modified' events when rules 3 and 4 are the cause of the observed event. Hypotheses events are generated to infer rule 1 'original' events from rule 3 and 4 events

5.2 Modeling the System Operations

The four block operation rules define the characteristics of the block to generation stamp relation. Formally, each relation R is defined with $\wp$ representing the power set:

$$\wp R := (G \times B): \forall p \in R: \\ p \in G \times B\ (g \in G,\ b \in B: p = \{g, b\}) \quad (1)$$

Let G_{obs} be a set of generation stamps in the discovered namespace: $\{g_1, g_2, \ldots, g_n\}$ where g_1 is the least element, and g_n is the greatest element. The generation stamp integer value domain is 1001 - 2^{64}-1. $|G_{obs}|$ = the number of elements in G_{obs}. This value is the total number of block operations explicitly enumerated in the discovered namespace from $\{g_1, b_1\}$ to $\{g_n, b_k\}$. However, since some of our block operations remove generation stamps from G_{obs}, this does not represent the total number of block operations over the namespace after the event associated with g_1. The generation stamp values belong to the set $\mathbb{Z}^+$, the set of positive integers, and are allocated sequentially and monotonically. The relation gRb is asymmetric, transitive, complete, and acyclic.

Therefore, we can determine the total number of block operations implicit in the namespace with the set G_{all}: $\{g_1, g_2, \ldots, g_n\}$ and $|G_{all}| = g_{n+1} - g_1$. The overwritten, or *absent* generation stamps are defined by G_{abs}:

$$G_{abs} := G_{all} - G_{obs} = \{\, g \mid g \in G_{all} \wedge g \notin G_{obs} \} \\ G_{obs} \cup G_{abs} \subseteq G_{all} \quad (2)$$

We can similarly define sets of blocks B_{obs}, B_{abs}, B_{all} for the discovered namespace. The block numbers use a different initiating block numbering criterion but are allocated sequentially and monotonically, forming a finite set of integers.

$$B_{abs} := B_{all} - B_{obs} = \{\, b \mid b \in B_{all} \wedge b \notin B_{obs} \} \\ B_{obs} \cup B_{abs} \subseteq B_{all} \quad (3)$$

Let $R_{obs} \subset R$ where R_{obs} is the discovered or observed image namespace. Every generation stamp, including those absent in the discovered namespace, belongs to a relation $R_{all} \subset R$ and $R_{obs} \subseteq R_{all}$. R_{all} includes our discovered namespace relations and hypothesis-based relations developed from of the discovered namespace. $R_{all} := (G_{all} \times B_{all})$. The binary relations R_{obs} and R_{all} are functions F. By definition:

$$F \in \wp(G \times B) \wedge (\forall p_1, p_2 \in F: p_1 \neq p_2 \Rightarrow \pi_1(p_1) \neq \pi_2(p_2)) \wedge \{\pi_1(p) \mid p \in F\} = G \tag{4}$$

The operators π_1 and π_2 act on the first and second coordinates of ordered pair p. The function from G to B has a one-to-one correspondence and is bijective when applied to the discovered namespace R_{obs} and surjective when applied to R_{all}.

Returning to the four operations defining *R*, we can state some propositions about the HDFS system and expected observed relations in the discovered image. The first proposition established allows us to create additional propositions and form inferences about the sequences of relations.

Proposition 5.2.1: $|B_{obs}| \leq |G_{obs}| \wedge |B_{abs}| \leq |G_{abs}| \wedge |B_{all}| \leq |G_{all}|$

> The number of blocks must always be less than or equal to the number of generation stamps. Operation 1 must occur prior to Operations 2-4 for every block included in the HDFS namespace. Operation 1 is bijective in its function and is the only operation in which a block is added to the allocated set of blocks B_{all}.

From Proposition 5.3.1, an additional proposition is formed about the quantity of file system operations on data blocks and the number of data blocks from the discovered image properties.

Proposition 5.2.2: $|G_{all}| - |B_{all}|$

> The number of modification operations can be determined by cardinality differences of any given partition.

Propositions 5.3.1 and 5.3.2 enable the causal ordering and bounding of relation *gRb*, which are modeled as event occurrences.

5.3 Specifying Forensic Properties

The discovered namespace relation R_{obs} is a total preorder. R_{obs} is a bijection, and by definition, the composition of bijective functions is bijective. Partitioning R_{obs} creates the mutually disjoint non-empty sets: $P_{Robs} := \{\{R_1\}, \{R_2\}, \ldots \{R_n\}\}$. Each partition is a bijection, and the union of these partitions is R_{obs}. Creating partitions provides two important analysis functions. First, partitions of *original* blocks may be generated. The data block content was created and written to without subsequent modifications to the content. Second, partitioning facilitates the computation of a *maximal validity interval* (MVI). This is an interval over which a property holds uninterruptedly. The algorithmic approach to compute possible states is simplified by minimal partitions of modified sequences.

Creating partitions bounded by known *original* data block properties allows us to reduce the state space when checking whether a property holds on the model. The effect is to create a second proposition on a partition concerning the initial events defined by the binary relation R_{obs}:

Proposition 5.3.1: $\exists \{g, b\} \in R_{obs}: \forall p \in R_{obs} (g \in G, b \in B: p = \{g', b'\})\ g \leq g' \wedge b \leq b'$

> The first event formed by the relation between generation stamps and blocks must be the least element from each set in a partition of original events. The partitions can be classified into two types:
>
> Type 1: Contains only block operations 1 and 2. These partitions are identified by the straight-line equation of the binary relation. The partition is a total order.
>
> Type 2: Contains block operations 2, 3, and 4, and is bounded by type 1 partitions. The partition is a total order of the observed set with complex event space partial orders.

If we utilize the concept of a logical clock, *c(t)*, and apply that to our set of events E, a predicate **happens** can be evaluated with the clause **happens**(*e, c(t)*). An a priori order is established from the image as a set of event occurrences. This pair, (*e, c(t)*), uniquely identifies an event occurrence. For forensic investigative practicality, we prefer an approach that would constrain the event occurrence to those within a partition of the namespace. Therefore, the initial goal is to show that the **happens**(*e, c(t)*) relations in the discovered namespace could form disjoint sets and cannot be united. If the partition can be shown to be monotonic, then reachable states of the namespace can be more efficiently discovered.

The algorithm employed to specify the forensic properties first verifies the discovered system partition is in a normal state. Sequences of block creation events do not include block operations rules 3 and 4. Block creation events are an arithmetic order-preserving sequence in the form:

$$e_n = e_{n-1} + 2 \tag{5}$$

This formula is zero-based, and e_0 can be any $e \in R_{obs}$. Arithmetically, e is the sum of the integer values for the block ID and generation stamp.

Any event occurrence *e* in the discovered namespace set *o* **happens**(*e, c(t)*), implies every event *e'* with $c'(t) < c(t)$ **happens_before**(*e*). Assuming discrete time increments μ on the logical clock where $t \in T$, the next event in our sequence has a clock value $c(t+\mu)$. The addition of an original block $b \in B_{all,}$ which is defined by $B_{obs} \cup B_{abs} \subseteq B_{all}$. If $B_{abs} = \emptyset$ in the time interval $c'(t) < c(t) < c(t+\mu)$, then $B_{obs} \Leftrightarrow B_{all}$. An original block will not have any $g \in G_{abs}$ as a possible relation. If the block is created at *c(t)* it is not possible to associate any time $c'(t) < c(t)$ with the block creation. Also, no events with time greater than *c(t)* could possibly generate an original block less than block *b* in the event identified as original. Therefore, the interval $[c(t) < c(t+\mu)]$ contains the original block generation stamp and $G_{obs} \Leftrightarrow G_{all}$ in the interval if the events satisfy the arithmetic sequence in (5). An event *e'* is an immediate predecessor of event *e* if no other event could exist in the interval between the events. By determining *original* event sequences (block creations), it is possible to partition the event sequence on these blocks. Additionally, an event sequence defined as a Type 1 partition is a unique topological ordering and cannot be invalidated by any additional information.

Theorem 5.3.1: If the duration μ between values of T is defined as the smallest possible time interval for T, which an event may occur, then the maximum number of events happening over the interval is one.

Proof: If two events occur between $c(t) < c(t+\mu)$, then the duration between their occurrences is less than μ. This is not possible given the definition of the values in T.

Theorem 5.3.2: If the duration μ between values of T is defined as the smallest possible time interval for T which an event may occur, then no event occurrence *e''* could have existed between $e'_{(c(t))}$ and $e_{(c(t+\mu))}$.

Proof: If event *e''* occurred between $c(t) < c(t+\mu)$ then e'' and e occurred between $c(t) < c(t+\mu)$. This is not possible given Theorem 5.3.1.

Block operation rule 1 in section 5.1 describes the operation for block creation in the namespace. Establishment of *original* blocks as bookends of the partition means all rule 1 operations antecede the partition for every block ID less than the minimal bookend block, and all rule 1 operations postdate the partition for every block ID greater than the maximal bookend block. The partition is bounded by the clocks of e_0 and e_n. If the sequence of events satisfies the arithmetic sequence of (5), then the partition is Type 1. The method creates partitions of the discovered namespace into *original* and *modified* regions. In fact, we can form numerous partitions that help to illuminate operations and bound reachability of hypotheses for R_{all}.

This method is utilized to determine which blocks are *original* within the namespace. Original in the sense that data content was written to the block once, and the block has not undergone data content modification. The method is also utilized and applied to Type 2 partitions to determine *original* generation stamp sequencing possibilities of modified blocks. Type 2 partitions are differentiated because these partitions contain regions where possible states exist without known order, and event hypotheses must be formed and tested. DFS^3 automatically generates data structures with partitioned data sets.

5.4 Verifying Properties

Event calculus (EC) allows the derivation of time intervals over which properties hold [35]. EC builds on first-order predicate calculus. The calculus is a general approach in logic programming to represent and reason about events and effects. Default persistence is a notion expressed in EC. An assumption of persistence until termination exists for properties defined by event effects. A history is modeled by a set of event occurrences as a binary acyclic relation.

The event calculus algorithm can be applied to each partition separately and composed as necessary. An investigator may query a knowledge base on any number of partitions, and the algorithm is used to determine whether the property holds. EC expresses the properties of states visited as properties that can be queried against a knowledge base derived from the observed state.

Several versions of the EC have evolved from the original [17]. The EC forms a timeline about events and *fluents*, the-time varying properties, or states. One variant of the calculus is called the Simplified Event Calculus (SEC). Any number of supporting domain-dependent axiom definitions are appropriate; however, the SEC consists of three core effect axioms in conventional logic programming if-form [35]:

$$\text{holdsAt}(f, t) \leftarrow ((\text{initially}(f) \wedge \neg\text{clipped}(0, f, t)) \qquad \text{(SEC1)}$$

$$\text{holdsAt}(f, t_2) \leftarrow (\text{happens}(e, t_1) \wedge \text{initiates}(e, f, t_1) \wedge t_1 < t_2 \wedge \neg\text{clipped}(t_1, f, t_2) \qquad \text{(SEC2)}$$

$$\text{clipped}(t_1, f, t_2) \leftarrow \text{happens}(e, t) \wedge t_1 < t < t_2 \wedge \text{terminates}(e, f, t) \qquad \text{(SEC3)}$$

These axioms assert that a fluent that is initially true remains true and that a fluent that is initiated persists until termination by a subsequent event. Given:

- a conjunction *SEC* of SEC1 and SEC2 and SEC3
- a conjunction Σ of Initiates and Terminates formulae
- a conjunction Δ of Initially, Happens and temporal ordering formulae
- a conjunction Ω of uniqueness-of-names axioms for actions and fluents of interest
- HoldsAt predicate form: HoldsAt(f, t)

Definition 5.4.1: The event calculus program:

$\Sigma \wedge \Delta \wedge SEC \wedge \Omega \vDash \text{HoldsAt}(f, t)$

A partial simplified event calculus model is defined based on a data block object within the HDFS. This example system consists of data units, blocks, which can only be written. The write is considered as two acts, Create and Modify, that effect the fluent property State. The State has a logical clock timestamp. Create and Modify cause State updates through Initiates clauses. Changes are represented as an event calculus with fluents as states of the data unit. Actions are the operations Create and Modify changing the fluent State. The set of Initiates and Terminates clauses set the values from the logical system clock as timestamp c(t). The set Σ follows:

$$\text{initiates}(\text{create}(\text{Block}), \text{state}(\text{Block}, c(t)), t) \qquad \text{(E1)}$$
$$\text{initiates}(\text{modify}(\text{Block}), \text{state}(\text{Block}, c(t)), t) \qquad \text{(E2)}$$
$$\text{terminates}(\text{create}(\text{Block}), \text{state}(\text{Block}, c(t_1)), t) \leftarrow t_1 < t \qquad \text{(E3)}$$
$$\text{terminates}(\text{modify}(\text{Block}), \text{state}(\text{Block}, c(t_1)), t) \leftarrow t_1 < t \qquad \text{(E4)}$$

Since HDFS blocks always have a value assigned to the ID and modification time, we will define Initially clauses for these fluents to hold from the start. A subset of Δ follows:

$$\text{initially}(\text{state}(\text{Block}, t_0)) \qquad \text{(E5)}$$

Completing the set Δ with a set of **happens** clauses completes an event calculus program for this basic HDFS. For example: A block is created at $t = t_c$ and subsequently modified at $t = t_m$, therefore $t_c < t_m$. We will let c(t) be integers so that $t_c = 1076$ and $t_m = 1093$. Two **happens** clauses are added to Δ:

happens(create(Block), t_c) ∧
happens(modify(Block), t_m)

The fluents in Δ can be examined in a logic program by means of Selective Literal Definite-clause Negation as Failure (SLDNF) resolutions for specific points in time for possible event histories, or to test propositions at a certain time. In this case, an assessment is made of the observation of the modified timestamp at a specific time to see if it holds. If we want to determine whether the block was created at the time observed $t_{ob} = 1093$, we could resolve holdsAt(State(block, 1093), t_{ob}).

A resolution for the observed timestamp would show the only **happens** clause that can satisfy this is happens(modify(Block),t_m). Therefore, holdsAt (state(Block, 1093), t_{ob}) is determined to belong to a possible observation lacking refutation.

Building on the theory developed by Willassen [32], the discovered namespace, R_{obs}, forms an observation set O as a finite set of **holdsAt** clauses. Leveraging the modeled system, a finite set of **happens** clauses may be derived from O to explain the observed state. Partitioning the namespace event space simplifies the hypotheses generation and treats the partitions as disjunctive logic programs in the EC. The Type 1 partitions possess complete ordering information, and no additional event occurrences are possible in these partitioned event spaces.

In partitions with multiple *modified* data blocks, it is possible to generate multiple *original* block hypotheses. If we let E define the events in the partition, we can define the number of possible original creation events for each modified event e' in the partition with the set of absent generation stamps in the interval defined by $\forall g \in G_{abs}$: G_{abs} is defined over the interval $(g \mid g \in \min_{e \in E} \wp(E) \leq e')$. Possible actions can be used to form event hypotheses in the form of **happens** clauses. It is possible to recursively traverse previous partitions to reduce the set of event hypotheses if desired.

6 Analysis and Findings

The DFS^3 builds a logical view that inversely maps the blocks to file objects. The software generates structures for directories, files and blocks. The directory structures contain the inode ID, directory name, the modification time, and user and group permission information. File structures include inode ID, file name, replication status, modification and access times, size, and user and group permissions. Block structures include generation stamp, the block order within the file, size, replication, and pointer to the inode. An additional mutable field exists for the data state content of the block. For the observed set, those blocks that are enumerated in the image file, this field is updated through the ordering logic property verification with the initial pass over the block sequences.

The image file used in the final analysis contained 3977 transactions over the namespace consisting of 1,574 objects (files, directories, data blocks). DFS^3 correctly identified all original content blocks and all 147 modified content data blocks from 1069 total data blocks in the image. Deleted inode and data block and absent generation stamp data sets were generated for analysis and hypothesis production. On average, the data structures were generated in under 3.0 seconds once the XML file is input. Progressively larger images were collected over a time span of several months.

Figure 2 presents the initial sequence of events on the cluster defined by relations *gRb* discovered in the namespace on the central server image file after a modest number of file system operations. In this case, the initial series consists of data block creation and deletion events. The sequence is generated by the DSF^3 causal ordering logic.

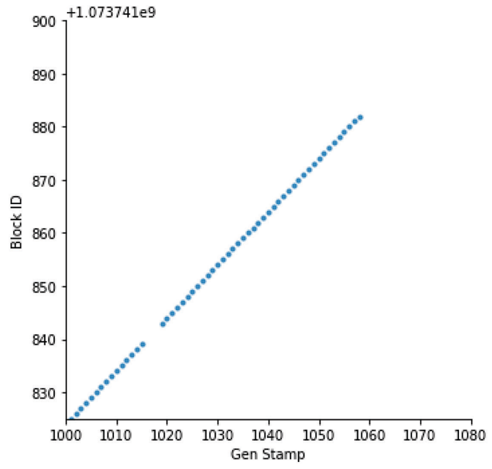

Figure 2. Initial cluster sequence of events, May 2019

Figure 3 presents the same initial sequence of events, however, from a progressively larger image space 70 days after the observation set generated in Figure 2.

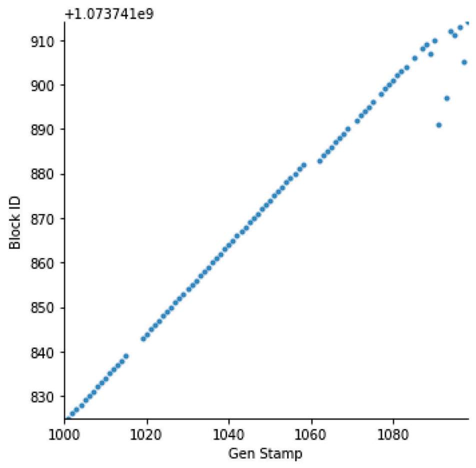

Figure 3. Initial cluster sequence of events, Jun 2019

An important property is illuminated in the initial sequence of events. Despite the addition of file system operations after the initial sequence, and 70 days between image captures, the properties of the initial sequence of events hold at a later observation. The EC logic programming exploits default persistence to reason about the events. Actions affect fluents, and the **holdsAt** clauses exist because the fluents were initiated or held initially. The model uses the data content state property to reason about the actions. The properties abstracted in the initial sequence of events will hold until the initiation of an action terminating the property.

Modeling the events as *gRb* relations with creation events as a function of (5) is the propositional statement equivalent of *the next data block in the namespace is assigned the next ID number in the block ID allocation set, and assigned the next ID number in the generation stamp allocation set.* If not, then the *gRb* where g is the next generation stamp in the allocation set must be further analyzed for possible action hypotheses.

Another important property illuminated in this example is the deletion of the data blocks will persist. It's an important concept, because the model will always reveal deletions in perpetuity. The possible creation events can be generated for all the deleted blocks and are bounded by events surrounding the deleted block creations. Therefore, write time bounds can be established for deleted files dating back over long periods of time. In the case of the initial sequence of events in Figure 2, it can be determined that three data blocks were generated on cycles 1016, 1017, and 1018. This is because theorems 5.3.1 and 5.3.2 allow for bounding the possible events around these cycles, and the causal ordering logic rules require data blocks to be generated in a specific sequence.

We turn to a sequence of events from a subsequent image extraction in Feb of 2020. The image is processed by DFS^3, and the output produces 15 directories, 92 files, and 235 data blocks in the observed set of events. The set-theoretical abstractions for the image reveal the deleted inodes and data blocks and the number of deletion and modification events to account for in the image. Figure 4 reveals a portion of the reconstructed space with multiple data block content modifications identified. Fourteen data blocks in this sequence have been identified as 'modified,' and the DFS^3 generates holdsAt and happens clauses initiating the states for the observation set. In addition, through the causal ordering logic, possible sets for creation cycles of the logical clock are generated for each modified block in the observation set. The blocks are then mapped to inodes and file objects generated to now reason about the file as an aggregation of the data blocks related to each inode. The software prompts the investigator to query the system for file state properties and possible action hypotheses.

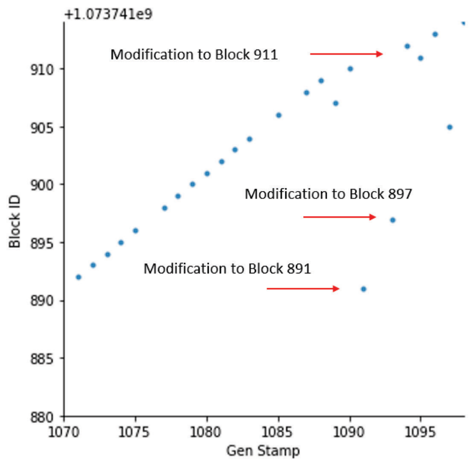

Figure 4. Partitioned space with multiple modified data blocks

The three data blocks highlighted by the red arrows in Figure 4 belong to two different files, inodes 16405 and 16408. When queried about these two files, the logic output is generated and produces the following via the command line interface:

```
Output:

Inode 16405 contains following blocks:
1073741890 Original
1073741891 Modified
1073741911 Modified
1073741913 Original
Original Block Bounds:
1073741890 1073741892 1073741910 1073741912
Possible creation cycles:
Block: 1073741891, Cycles: [1070]
Block: 1073741911, Cycles: [1092]

Inode 16408 contains following blocks:
1073741896   Original
1073741897   Modified
1073741912   Original
Original Block Bounds:
1073741896 1073741898
Possible creation cycles:
```

Block: 1073741897, Cycles: [1076]

In this context, the files are presented as objects with content history state representation. The original block bounds are identified, which reduce the space in which to reason about causal events. Possible creation cycles are generated for those data blocks identified as modified.

Another file in the image, inode 16396, contains two data blocks. During the first pass, the following information is generated for this file:

blocks: [1073741883, 1073741884]
block states: [Modified, Original]

The clock cycle for the modification event of block 1073741883 is 1062 in the observed set. Upon generating a query for inode 16396, the system produces the following output logic:

Inode 16396 contains following blocks:
1073741883 Modified
1073741884 Original
Original Block Bounds:
1073741882 1073741884
Possible creation cycles:
Block: 1073741883, Cycles: [1059, 1060, 1061]

There exist three possible cycles for the 'original' data content creation events for block 1073741883. This unusual case for the HDFS was intentionally generated in the test image and is utilized to highlight the logic behind the program. The data block was created, and original data content written to on logical clock cycle 1059. Subsequently, and in sequence, three modifications were made to the data block. A truncate operation followed by two append operations with the latter append generating a new block of data, 1073741884, within the file. All three possible logical creation cycles would present as valid and would be accepted in the logic as valid; however, they do not imply full knowledge in this case, only as possible valid hypotheses actions. There may be more than one **Initiates** clause initiating this fluent, and all must be considered. In this instance, if the valid logical clock cycle of 1059 is established, the other two possibilities of 1060 and 1061 are constrained to modification events for inode 16396 given the causal ordering of events.

6.1 Forensic Value

The reconstruction is a history of the file operations modeling the state of the data blocks. One benefit is data causal consistency can be established given the file operation history. For example, if the file presented in the previous section as inode 16408 is of interest and an operation in question is prior to event 2, then the single block of 1073741896 on disk is representative of the file state at that time, and we've reduced the number of blocks to recover or analyze. Similarly, if the operation is after event 4, then all three blocks on disk are available and representative. Figure 5 shows the reconstructed state of the file and the validity intervals.

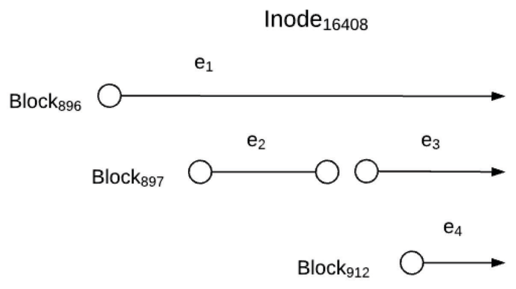

Figure 5. Inode 16408 data block content state validity intervals

The reconstruction provides an enhanced preview of the file system with output common to logically acquiring and extracting file system volume. The volatility of the data in the unique context of the HDFS could be better understood at an early point in the process. If data blocks exist on the current system with a persisted state property of interest, a decision could be made to quarantine datanodes and identify files for recovery before additional operations modify files. The amount of effort that is required to recover data may be established much earlier.

Timelines are established without the reliance on physical clocks. The event bounding helps to verify modified, accessed, and created times on files and other subevents of the system using the event time bounds. Additionally, log file analysis is not only simplified, but log file timestamps can be verified through the event bounding. In the case of inode 16408 in Figure 5, the only possible logical cycle for e_2 is determined to be clock cycle 1076. The log files should contain a data block creation event for block 1073741897 within the physical clock time bounds of logical cycle 1076. Since this is the only possible valid scenario, in this case, discovering this event in the log file established a high level of confidence in the validity of the log file. Times across log files and system events can be synchronized using the event sequences and consistent timelines established and verified.

Considering inodes 16405 and 16408 again, though 16405 is created before 16408, the last block in 16405 is block 1073741913, and it is original data content. Inode 16408 last block is 1073741912, also original data content. This means that data was more recently written to inode 16405, and we would expect to find consistent modified timestamps for these two files. In fact, the modified timestamps for the two files support this reasoning:

16405: 2019-12-18 22:27:35.167000
16408: 2019-12-18 22:12:47.148000

The timestamps can be synchronized to the events and the physical timestamp bounds become tighter with higher event density.

7 Conclusions and Future Work

Although the top-down technique could produce multiple possibilities or undefined states of the lower-level data, we believe this approach is well suited for an initial analysis of the HDFS. The method produces many common output concepts of traditional lower-level file system forensic tools without the need to produce low-level volume images. Further, if the possible storage states are formally defined and bounded, the top-down approach aids in the production of supporting evidence once the investigation proceeds into lower-level analysis.

Event bounding allows the formation and testing of hypotheses to capture all possible high-level events of interest with associated global state space explosion mitigation. The model forms a collection of trustworthy observations building a foundation for hypothesis-based investigation of the HDFS. This type of automated high-level reconstruction is not intended to replace all low-level analysis but instead reduce the need or narrow the focus of low-level file system extraction and analysis. The high-level events can highlight areas of interest and expedite the overall investigative process. The reconstructed events could also provide verification and evidentiary support to raw data evaluation, such as log file analysis. The reconstructed state properties capture the system invariant properties and thus provide a formal foundation to establish validity to the reconstruction.

By conducting a hierarchical approach which embeds lower-level primitive events, the method offers several advantages:

- The hierarchical embedding of subevents into higher-level complex events forms a framework for traceability to lower-level analysis
- Asynchronous events are modeled as synchronous complex events simplifying the temporal component to the model
- Model simplification means that any 'concurrent' events can be treated as complex events and time bounds applied

Future work includes improving DFS^3 to provide more abundant tools to enhance the analysis features. Further broadening the test data set and cluster environment to include very large numbers of data blocks, complete operations sets, and fuller metadata associations. The incorporation of a suitable knowledge base and a broader rule reasoner to include additional properties of the DFS is also left for future work.

ACKNOWLEDGMENTS

This work being reported partially supported by the National Science Foundation; the grant number is being withheld as part of the double-blind review.

REFERENCES

[1] "Defining Digital Forensic Examination and Analysis Tool Using Abstraction Layers.," *Int. J. Digit. Evid.*, vol. 1, no. 4, pp. 1–12, 2003.

[2] R. Ayers, W. Jansen, and S. Brothers, "Guidelines on mobile device forensics (NIST Special Publication 800-101 Revision 1)," *NIST Spec. Publ.*, vol. 1, no. 1, p. 85, 2014, doi: 10.6028/NIST.SP.800-101r1.

[3] [ISO/IEC 27037:2012]. 1st ed, "Guidelines for identification, collection, acquisition and preservation of digital evidence," 2012. http://www.iso27001security.com/html/27037.html (accessed Mar. 13, 2018).

[4] C. Kynogis, W. B. Glisson, T. R. Andel, and J. T. McDonald, "Utilizing the Cloud to Store Camera-Hijacked Images," in *Hawaii International Conference on System Sciences*, 2016, vol. HICSS-49.

[5] G. Grispos, W. B. Glisson, J. H. Pardue, and M. Dickson, "Identifying User Behavior from Residual Data in Cloud-based Synchronized Apps," *J. Inf. Syst. Appl. Res.*, vol. 8, no. 2, pp. 4–14, 2015, [Online]. Available: http://jisar.org/2015-8/.

[6] T. Watts, R. Benton, W. Glisson, and J. Shropshire, "Insight from a Docker Container Introspection," in *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[7] J. Shropshire and R. Benton, "Container and VM Visualization for Rapid Forensic Analysis," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[8] G. Grispos, T. Storer, and W. Glisson, "Calm Before the Storm: The Challenges of Cloud Computing in Digital Forensics," *Int. J. Digit. Crime Forensics*, vol. 4, no. 2, pp. 28–48, 2012, doi: 10.4018/jdcf.2012040103.

[9] A. J. Brown, W. B. Glisson, T. R. Andel, and K. R. Choo, "Cloud Forecasting: Legal visibiliy issues in saturated environments," *Comput. Law Secur. Rev. 2018/06/18/*, 2018, doi: https://doi.org/10.1016/j.clsr.2018.05.031.

[10] D. Quick and K. K. R. Choo, "Big forensic data reduction: digital forensic images and electronic evidence," *Cluster Comput.*, vol. 19, no. 2, pp. 723–740, 2016, doi: 10.1007/s10586-016-0553-1.

[11] N. D. W. Cahyani, N. H. A. Rahman, W. B. Glisson, and K.-K. R. Choo, "The Role of Mobile Forensics in Terrorism Investigations Involving the Use of Cloud Storage Service and Communication Apps," *Mob. Networks Appl.*, vol. 22, no. 2, pp. 240–254, 2017, doi: 10.1007/s11036-016-0791-8.

[12] N. H. A. Rahman, W. B. Glisson, Y. Yang, and K.-K. R. Choo, "Forensic-by-Design Framework for Cyber-Physical Cloud Systems," *IEEE Cloud Comput.*, vol. 3, no. 1, pp. 50–59, 2016, doi: 10.1109/MCC.2016.5.

[13] G. Grispos, W. B. Glisson, and T. Storer, "Chapter 16 - Recovering residual forensic data from smartphone interactions with cloud storage providers," in *The Cloud Security Ecosystem*, R. K.-K. R., Boston: Syngress, 2015, pp. 347–382.

[14] G. Grispos, W. B. Glisson, and T. Storer, "Using smartphones as a proxy for forensic evidence contained in cloud storage services," *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, pp. 4910–4919, 2013, doi: 10.1109/HICSS.2013.592.

[15] E. Harshany, R. Benton, D. Bourrie, and W. Glisson, "Forensic Science International : Digital Investigation," *Forensic Sci. Int. Digit. Investig.*, vol. 32, p. 300909, 2020, doi: 10.1016/j.fsidi.2020.300909.

[16] B. D. Carrier and E. H. Spafford, "Categories of digital investigation analysis techniques based on the computer history model," *Digit. Investig.*, vol. 3, no. SUPPL., pp. 121–130, 2006, doi: 10.1016/j.diin.2006.06.011.

[17] E. T. Mueller, "The Event Calculus," *Commonsense Reason.*, pp. 19–52, 2006, doi: 10.1016/b978-012369388-4/50059-x.

[18] K. Khazanchi, A. Kanwar, and L. Saluja, "An Overview of Distributed File System," *Ijmer*, vol. 2, no. 10, pp. 2958–2965, 2013.

[19] H. G. Sanjay Ghemawat and Shun-Tak Leung, "The google filesystem," 2003.

[20] S. Weil, S. Brandt, E. Miller, and D. Long, "Ceph: A Scalable, High-Performance Distributed File System," in *USENIX Symposium on Operating Systems Design and Implementation*, 2006, vol. 30, no. 2, pp. 267–291, doi: 10.2307/624306.

[21] B. Carrier, *File System Forensic Analysis*. Addison-Wesley Professional ©2005, 2005.

[22] B. Carrier and E. Spafford, "An event-based digital forensic investigation framework," *Digit. forensic Res. Work.*, pp. 1–12, 2004, doi: 10.1145/1667053.1667059.

[23] B. D. Carrier and E. H. Spafford, "Defining Event Reconstruction of Digital Crime Scenes," *J. Forensic Sci.*, vol. 49, no. 6, pp. 1–8, 2004, doi: 10.1520/jfs2004127.

[24] W. Minnaard, "The Linux FAT32 allocator and file creation order reconstruction," *Digit. Investig.*, vol. 11, no. 3, pp. 224–233, 2014, doi: 10.1016/j.diin.2014.06.008.

[25] C. Hargreaves and J. Patterson, "An automated timeline reconstruction approach for digital forensic investigations," *Proc. Digit. Forensic Res. Conf. DFRWS 2012 USA*, vol. 9, pp. S69–S79, 2012, doi: 10.1016/j.diin.2012.05.006.

[26] S. Soltani and S. A. H. Seno, "A formal model for event reconstruction in digital forensic investigation," *Digit. Investig.*, vol. 30, pp. 148–160, 2019, doi: 10.1016/j.diin.2019.07.006.

[27] K. Chen, A. Clark, O. De Vel, G. Mohay, and Q. Brisbane, "Ecf – Event Correlation for Forensics Introduction : Ecf System Requirements and Design," *Network*, no. November, pp. 1–10, 2003, [Online]. Available: http://scissec.scis.ecu.edu.au/secauconfs/proceedings/2003/forensics/pdf/11_final.pdf.

[28] B. Schatz, G. Mohay, and A. Clark, "Rich Event Representation for Computer Forensics," *Asia Pacific Ind. Eng. Manag. Syst. APIEMS 2004*, no. April 2016, pp. 1–16, 2004, [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:RICH+EVENT+REPRESENTATION+FOR+COMPUTER+FORENSICS#0.

[29] P. Gladyshev and A. Patel, "Finite state machine approach to digital event reconstruction," *Digit. Investig.*, vol. 1, no. 2, pp. 130–149, Jun. 2004, doi: 10.1016/J.DIIN.2004.03.001.

[30] J. James, P. Gladyshev, M. T. Abdullah, and Y. Zhu, “Analysis of evidence using formal event reconstruction,” *Lect. Notes Inst. Comput. Sci. Soc. Telecommun. Eng.*, vol. 31 LNICST, pp. 85–98, 2010, doi: 10.1007/978-3-642-11534-9_9.

[31] M. N. Khan, E. Mnakhansussexacuk, and I. Wakeman, “Machine Learning for Post-Event Timeline Reconstruction,” *PGnet*, pp. 1–4, 2006.

[32] S. Y. Willassen, “Using simplified event calculus in digital investigation,” *Proc. ACM Symp. Appl. Comput.*, pp. 1438–1442, 2008, doi: 10.1145/1363686.1364020.

[33] E. M. Clarke and W. Klieber, “Model Checking and the State Explosion Problem,” vol. 1041377, no. 2005, pp. 1–30, 2012.

[34] “Primitive Data Types (Learning the Java Language Basics),” 2018. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html (accessed Aug. 07, 2019).

[35] M. Shanahan, “The Event Calculus Explained,” *Wooldridge M.J., Veloso M. Artif. Intell. Today*, pp. 19–52, 1999, doi: 10.1016/b978-012369388-4/50059-x.