THE FLORIDA STATE UNIVERSITY

COLLEGE OF ARTS AND SCIENCES

A SURVEY ON TECHNIQUES

TO ENRICH FILESYSTEM ACCESS METHODS

By

MICHAEL MITCHELL

A Survey submitted to the
Department of Computer Science
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

The members of the committee approve the area survey of Michael Mitchell defended on June 22, 2012.

_____
An-I Andy Wang
Professor Directing Survey

_____
Gary Tyson
Committee Member

_____
Xiuwen Liu
Committee Member

_____
Zhenghao Zhang
Committee Member

_____
Linda DeBrunner
External Committee Member

Approved:

_____
Robert van Engelen, Chair, Department of Computer Science

_____
Sam Huckaba, Interim Dean, College of Arts and Sciences

The Graduate School has verified and approved the above-named committee members.

TABLE OF CONTENTS

ABSTRACT

The amount of digital data is growing at a staggering rate, and this can stress legacy file-system access methods in many ways. Specifically, locating files within growing hierarchies becomes more daunting, the evolving relationships of files become more difficult to maintain; and adapting to dynamically changing environments becomes increasingly necessary. This survey examines existing techniques for remedying these deficiencies through the use of content clustering, content history, and content context. While these techniques cover a broad range of knowledge bases, this survey demonstrates that an overarching solution remains elusive.

CHAPTER ONE

INTRODUCTION

We are in the midst of a data explosion. The amount of digital data is growing at a staggering rate. During a five-year study from 2000 to 2004, the average number of files increased from 30,000 to 90,000 [Agrawal07]. In addition, these files are becoming larger, with the average size increasing from 110 KB to 190 KB [Agrawal07]. A significant portion of this increase is caused by user-generated content because user documents and settings files have increased from 7% to 15% [Agrawal07].

An increase in the volume of files translates into numerous challenges for filing services. First, locating the desired content becomes more difficult. More files and directories must be searched to locate the desired content. Second, the cost of encoding and tracking the interactions and relationships also becomes prohibitive. This capability is needed for businesses, where the ability to audit is often required due to accounting, intellectual properties, and regulations regarding to privacy issues.

In addition, users are interacting with storage in many new ways. Mobile devices constantly experience a dynamic range of operating environments (e.g., tethered and untethered). File systems with static behaviors no longer can easily accommodate the required fluctuating resource constraints.

1.1 Brief Background on File Systems

On the other hand, the general notion of a file system has been largely unchanged since its introduction. File systems provide a simple container abstraction, the file: general-purpose named byte arrays used to store any type of arbitrary information. To support this requirement, no general structure or formatting constraints are imposed on files. Applications create and manage files and structure data in the way that is most appropriate.

Due to this nature, files are largely semantic free, with exceptions such as definitional and environmental information encoded in file-name extensions (.doc, .pdf, .jpg), and built-in, type-

specific ‘magic’ numbers. This leaves applications the majority of the burden to define file semantics.

A file is generally organized within a folder or a directory, which can also be organized within a directory. From the top-level directory, or the root of the hierarchy, to the file’s immediate parent directory form a path (e.g. /home/user/ etc.) to the file.

1.2 Limitations of Legacy File System

Hierarchical organization becomes limiting as the explosive growth of data continues. As the average number of files within a directory tends to stay the same [Agrawal09], it becomes increasingly unwieldy to remember and locate desired content amidst the growing the number and the depth of paths. Human mental model further shows the ineffectiveness of hierarchical organization [Seltzer09] and difficulty of locating files within [Soules03].

The expressiveness of relationships among files is also limiting with the hierarchical categorization scheme (although it is possible to provide multiple names to a file via shortcuts and symbolic links). Additionally, the relationships between files may deviate substantially from the initial organization and later iterations.

Legacy file systems have limited support to encode metadata (e.g., size, name, descriptions of a file). For example, a user may want to identify contributors to a project to attribute intellectual property or to prosecution of criminal violations. Or, the user may want to understand the history of data derivations and transformations for experimental data sets and simulations. Legacy file systems do not provide this functionality, leaving the burden of lineage tracking to the user or external applications.

There are also times where a user wants to restore a previous version of a file. Legacy file systems generally do not incorporate this type of service, leaving error recovery and rollback to version-control and backup systems.

A user may additionally want to automate software configurations and build environments by tracking associated libraries necessary for a given subset of files. Such dependency-tracking- and history-related information is often not readily available and integrated into the core of file system services.

Another aspect of limitation of legacy file systems is their rigid behavior amidst diversifying computing environments. For example, fluctuations in the availability of resources, such as network bandwidth or power constraints can offer clues for improved user experience. Legacy file systems do not utilize this supplemental information, resulting in a less than optimal user experience.

As these trends continue to progress, it becomes apparent that the legacy file system is not sufficient to handle these constraints.

1.3 Databases and Raw Storage Access

Using databases is one alternative to ease the finding of desired content and to track relationships among objects. Some advantages of using databases include that through predefined schemas, they provide rigorous structuring of data through type-specific field enforcements. That is, given some arbitrary data fields (integer, float, string, etc.) the database is able to ensure a normalized table. Databases also can encode the relationships among data; the relational model links tables and fields of disjoint tables using foreign key constraints. They also use additional recomputed indices to accelerate data retrievals.

Databases, however, are typically unsuited when the user desires fine-grained and dynamic control over their data. Databases updates can be heavyweight, as the re-calculation of indices may be expensive. Further, the restructuring and re-organization of schemas are even more expensive, since they are pre-defined prior to populating the databases. For these collective reasons, databases alone cannot meet the needs for many applications. While some systems utilize databases or database-like constructs to enrich file-system access methods, they will not be the focus of the remainder of the survey.

At the far opposite end of the spectrum, we can interact with raw storage directly using custom constructs. This imposes high complexities for most applications, defying the purpose of having file systems to provide common services and to simplify applications.

1.4 Roadmap

Methods to enhance the file system access to address various limitations mentioned have come from various computing disciplines. They can be generalized into three areas: exploiting

additional knowledge of file names, metadata, and content to ease locating content similar to specified input (Chapter 2), using history to provide backup/rollback, auditing support, and insight into data derivation (Chapter 3), and using contextual information to provide proximate selection, localization, and dynamic application behavior (Chapter 4).

CHAPTER TWO

CONTENT CLUSTERING

2.1 Overview

Content-clustering-based methods exploit the similarity and relationships between files to enrich the user access model. These similarities and relationships can be encoded to facilitate content searches and navigations. Thus, these methods are commonly referred as desktop search application, analogous to search engines for the local file system instead of the web.

Since legacy file systems are not self-indexing or directly searchable, desktop-search applications rely on external indices. During the indexing process, desktop search applications usually collect file/directory names, associated metadata, and keywords from files encoded in supported formats.

When the user searches for content, desktop search applications must consult the index and determine which files are most relevant, ranked based on a similarity distance metric. Since files types are remarkably diverse, it is difficult to create a general-purpose solution to this problem. Even between files of the same type, it is often difficult to create an appropriate distance metric [Cohen08].

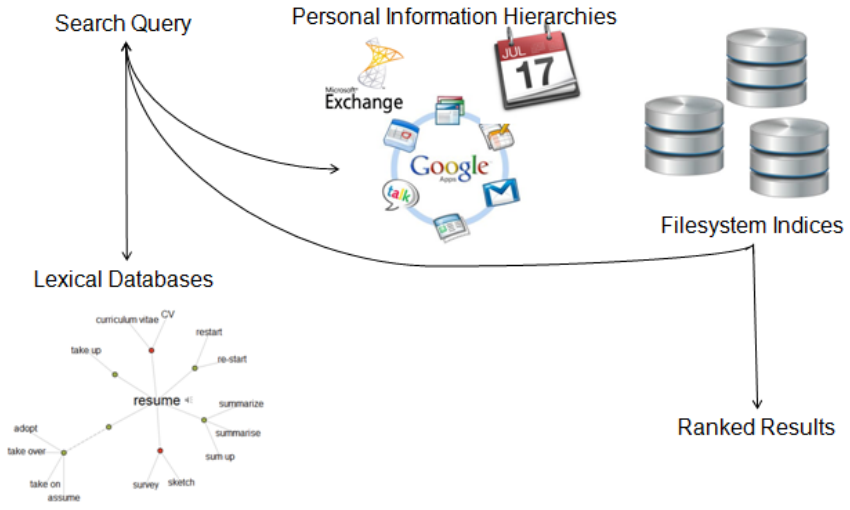

Figure 2.1: Desktop Search Components

Details of ranking mechanisms are largely limited due to the proprietary nature of products from vendors like Google [Google05], Microsoft [Microsoft06], and Apple [Apple08]. However, insight can be gained by analyzing how users search for data. Users tend to search for files they have accessed recently [Cohen08], and almost always, those they have accessed at some point previously [Dumais03]. This makes the modification and access time a useful metric for ranking, and as a result most systems will default to date-based presentation of distance-based results. Beagle++ [Chirita06] uses a variation of the database ObjectRank [Balmin04] mechanism for authority-based ranking.

Some systems, such as Phlat [Cutrell06], do not actually provide any ranking or presentation of ranking by default. The query results can be sorted according to the user's desired metric. Other systems, such as Personal Web [Wolber02], integrate with other information hierarchies such as emails and bookmarks. They utilize these structures for additional indexing and ranking heuristics.

To minimize interference to normal system usage, the index creation process is normally deferred to a time when the system is idle. Depending on the desktop-search environment, updates to the index can be processed similarly during idle system cycles, or can be propagated automatically as files are updated.

Independent of ranking algorithm and heuristics used, the desktop-search application presents the user with a ranked set of files deemed most relevant to the query.

2.2 Capabilities and Benefits of Desktop Search

The primary goal of desktop search is to ease file-system browsing and provide more instinctive information retrieval experience to the user. With correct combinations of keywords, users can locate the desired content directly instead of navigating the file-system hierarchy in the traditional way. They can locate files by name, file system path, or portions of these two descriptors. Desktop search also can provide access based on the content type, or extension of the content. In addition to their search ability, desktop-search indices exist in parallel to the file system proper providing multiple access paths to data [Perugini10].

2.3 Desktop Search vs. Search Engines

While desktop search may seem like a local search engine, there are fundamental differences between the two.

Links and ranking: For web pages, the quantity and quality of inbound links can provide hints to the relevance of the document, and is the fundamental concept behind Google's Page Rank algorithm [Brin98]. Unlike web content, file system contents tend not to be hyperlinked. This is problematic with regard to ranking; algorithms like Page Rank are not effective without links between data content. Compared to search engines, there is not a well-accepted best ranking method for desktop search systems [Cohen08]. However, unlike search engines, desktop search has extra sources for ranking improvement, including file metadata, usage metadata, and folder structures.

Indexing: From text-based content, to media files, to application files, there are many different data types that must be handled. Thus, desktop search must have a diverse understanding of the data layout of files is required to extract relevant content to be indexed.

Presentation: Another distinction between search engines and desktop-search applications is how the results are presented to the user. For file-system access, users prefer to navigate, not to search [Perugini10]. Creation of an indexing structure capable of navigation-based search is difficult since categorizations and organizations evolve over time.

Privacy: Desktop search and search engines have different requirements with regard to privacy as well. Some information, such as credit card, bank account, or social security numbers is highly sensitive, and a user may desire not to have them indexed. For search engines, a web directory can contain a robots.txt text file that directs the indexing spiders to avoid accessing the sensitive data. For a file system, per-directory specification of sensitive information is cumbersome for users, as the presence of such files, with names containing phrases like accounts, statements, or similar can reveal clues to the location of confidential data. Further, as a result of these privacy concerns, research is often hindered by difficulties to acquire relevant datasets [Cohen08].

2.5 User/Desktop Search Interaction

Desktop search applications have taken a number of approaches to index and optimize the results returned. These can be generalized into the categories of direct search, navigational and faceted search, semantic search, and context-aware search (details in Chapter 4).

2.5.1 Direct Search

Direct desktop-search applications allow the user to explicitly search for file system content via relevant words or phrases. The supported queries vary from system to system, but in general, most allow search by filename, path, and type. The origins found in most general search engines such as Google [Google05] and Windows Live Search [Microsoft06].

User interaction with direct desktop-search systems is simple: the user enters a keyword, part of the directory path, or metadata into a search bar, and receives results returned that match the criteria, sorted by the respective ranking algorithm of the search provider. The majority of commercial desktop-search applications are based on this access model, including Google Desktop [Google08], Windows Desktop Search [Microsoft06], and Beagle [Chirita06].

Direct search applications provide basic free-text extraction from basic file types. For example, they can remove formatting and presentation elements from .docs and .pdfs and extract the plaintext representation of these documents. This resulting data content is indexed and included within user queries. More extensive data type handling is covered in the semantic search section (Chapter 2.5.3), as are handling of synonyms and typographical errors.

2.5.2 Navigational and Faceted Search

Navigational-search-based approaches utilize hierarchal classifications to direct user queries. These structures rely on the user to iteratively refine the scope of categorization at each level. Navigational search applications originate from web directories like Yahoo! Directory [Yahoo02], Open Directory/DMOZ [Dmoz02]. In these systems, web pages are indexed, with categories applied to all pages. The categories themselves can then be traversed by successively clicking links to refine the scope of the search. Compared to direct search applications, far fewer navigational-search-based solutions exist.

User interaction with navigational-search applications is similar to traditional file-system interaction. The notion extends from navigation of a directory tree by clicking folder after folder to refine the query. However, unlike file-system browsing, navigational search uses multi-

dimensional indices. Data is accessible through multiple path permutations, conceptually as if they were arranged in overlaying file-system directory structures.

Faceted search is a hybrid of direct- and navigational-search methods. It combines the structured indices of navigational methods with the explicit search functionality of direct search. At any level of the mutli-dimensional search, the user can narrow the scope by subsequent series of sub-searches (direct search), or through categorical refinements (navigational search) of the results set, or facet. Facets may be hard coded [Perugini10], annotated, as in systems like Haystack [Karger04], or learned, in systems like Magnet [Sinha05]. Due to the frequency of a user's search for recently accessed content, sorting the results by date tends to be the most utilizes secondary faceted search feature [Perugini10].

2.5.3 Semantic Search

Semantic search is a logical evolution of the direct search method. Semantic search systems try to humanize the information retrieval experience by providing search options with a better understanding of the data. This insight can take several forms.

Semantic can mean a better understanding of the search term. For example, direct synonyms (e.g., airplane is a jet), hyponyms (e.g., Boeing 747 is a specific airplane), hypernyms (e.g., airplane is a generalization of Boeing 747), meronyms (e.g., wings are part of an airplane) and holonyms (e.g., airplanes contain a fuselagex vs. y) [Chirita05] extracted from lexical databases such as Princeton's WordNet [Princeton10]. The results of these extractions are included as part of the search query allowing the search to return more relevant data files.

Semantic can mean a deeper understanding of the data file internals. Semantic systems can utilize programmable data transducers [Gifford91], essentially plug-ins that tell the semantic system how to interpret the content of the data type. Systems like Beagle++ [Chirita06] go beyond the handling of simple plaintext extraction and provide extensions to handle many file types including non-text-based types and binary data files.

Semantic can mean a better understanding of data through relationships and association with other data. This area overlaps with several other enrichment methodologies. It can resemble faceted search methods (Chapter 2.5.2) in the integration of other enriching information hierarchies, such as the Semantic Web [Berners-Lee01], and similar classification ontologies with systems like Sile [Schandl09]. Similar to data provenance-based techniques (Chapter 3.1) systems such as Stuff I’ve Seen [Dumais03] track the file access of the user. This provides a

heuristic for ranking, since users are likely to search for files that have previously seen [Cohen08]. Finally, these systems can resemble passive context-aware systems (Chapter 5) by tagging and relating the user interactions with the desktop.

User interaction with semantic search systems is nearly identical to user interaction with direct search applications; the user performs key phrase search, receives enhanced, more intuitive results.

The addition of semantic search methodologies into the file system itself paves way to semantic file systems. Semantic file systems extend the notion of search and access based on data content rather than file-system location or naming convention. In contrast to traditional file systems, semantic file systems organize and access data by content and facilitate associative access and querying of data from within the file system [Gifford91].

Since the functionality is included within the operating system, semantic file systems are significantly more flexible than traditional systems. System benefits can also be achieved through utilization of semantic-file-system techniques [Leung09]. Related files can be pre-fetched or cached to enhance performance [Hua09].

2.6 Pros and Cons of Content Clustering

Content clustering can enrich the user information retrieval experience in a number of ways. (1) It can help locate files without knowledge of path location, improving the productivity of a user by spending less time searching. Desktop-search systems can also mimic traditional file-system interfaces, further extending the user's ability to interact with the desired data content in a more intuitive manner. (2) Content-clustering methods can enhance access through tailored search results by integrating with other exiting information hierarchies, like emails and bookmarks. (3) Content-clustering-based systems can identify semantic relationships across file formats. Users can more intuitively search for the content within the data, independent of its type or container format.

However, content-clustering-based methods have their limitations as well. Many files, such as source code, contain a large amount similar content across data files. Without a way to differentiate the versions and copies, the search results do little help the user locate the desired content amongst many similar files.

Content-clustering methods can also be too aggressive in associating relationships. Unrelated files (e.g., two different files named test.c in different projects that perform different actions) can be identified by the search system as being related. Thus, when the user searches for such content, the returned results little narrow the scope of the search.

In addition, content-clustering-based methods lack of history information. Without knowledge of data lineage, many opportunities to identify relationships between data content are missed. Content-clustering-based methods generally rely on some type of index to facilitate searching. The initial indexing itself can be an expensive operation, as well as the incremental updates to keep it current. Depending on the implementation, this can also cause some consistency issues as well.

CHAPTER THREE

CONTENT HISTORY

3.1 Overview

Data content is far from static; files are constantly being updated and continually evolving. Most file systems disregard this information of how files evolve and only store the most recent version of the file and its metadata. This unfortunately misses many opportunities to enrich the user experience. When this supplemental data is tracked and included in the file system, many improvements can be realized.

This section will discuss the enhancements and capabilities provided to file systems through this content history. They generally fall into three categories: versioning, dependency tracking, and provenance.

3.2 Versioning

Instead of storing only the current copy of the file, versioning systems retain historical versions. These versions, or commits, are a snapshot of a given file or set of files at a certain moment in time. Unique version numbers are assigned to differentiate between versions. Versioning systems provide the user with a diverse array of capabilities.

3.3 Capabilities & Benefits of Versioning

Backup and restore. Versioning systems can trivially offer backup and restore capabilities. With a historical record of commits made to the data, nothing is forgotten. The user can restore a previous version of the data.

Rollback/undo. Versioning systems can improve the safety of a system with efficient rollback and undo capabilities. If an error or a malicious program is discovered, the user can return to the point before the error existed.

Track changes/annotations. Versioning systems can help understand how software evolves. Annotation can encode additional metadata about the evolution of data and

organizational layout. Similar in effect to provenance, changes can be attributed to the corresponding user.

Branching/merging. Versioning systems can support multiple branches. These branches can contain variants of the same files, or different files on different branches and can have independent commit histories from each other.

Synchronization. Versioning systems can help synchronize updates to replicate data across machines. With branching, versioning systems are even able to handle distributed updates with completely different histories.

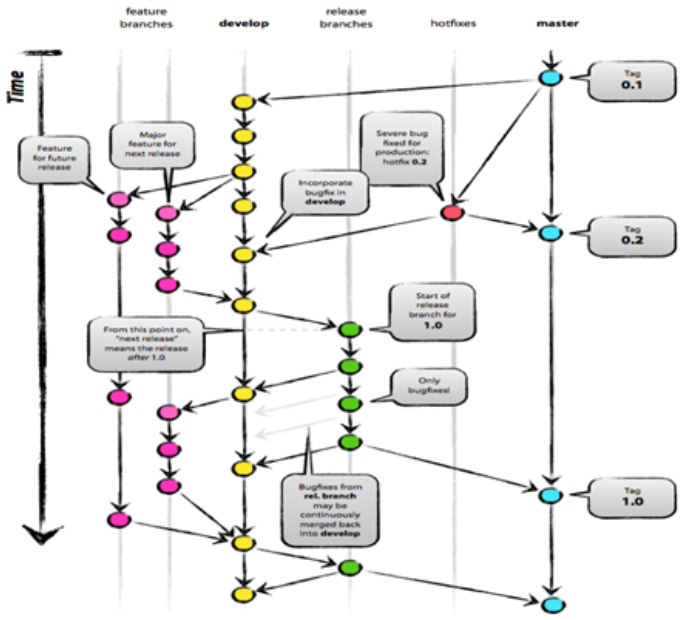

Figure 3.1: Example version history

3.4 Versioning is Not Backup

At first glimpse, versioning systems may seem similar to backup systems. However, there are a few notable differences.

The primary difference is the granularity of the historic copies. Most backup systems operate at the partition- or system-wide granularity. In contrast, most versioning systems store historic copies at the file-level granularity.

They differ in when the historic versions are stored. Non-mission-critical backup systems typically run at periodic intervals. After a specific time period has passed a backup is

triggered. Versioning systems instead store incremental commits at the discretion of the user, or as the files are updated, more commonly used by versioning file systems.

Finally, the systems differ on where the historic versions are stored. Backup systems normally utilize separate or external media for storage. In contrast, most versioning systems store the incremental versions on the same media as the data proper.

3.5 Interactions between Users and Versioning Systems

Versioning systems can store data either explicitly or implicitly. That is, the versioning can be done as the file is updated transparently or explicitly at the request of the user, be it the actual user, or an application or system as the user. Explicit versioning systems have directly evolved from version control systems, with the explicit version representing the developer initiated snapshot of the development tree.

Implicit versioning systems automatically perform the versioning without input from the user. At one extreme, implicit versioning systems can version on each write to the file exemplified by Ext3cow [Peterson05] (ext3 copy on write). At the opposite extreme, versioning can be relaxed to open-to-close versioning. Advanced systems, such as causality-based versioning, utilize algorithms to provide cycle avoidance and handle concurrent execution [Muniswamy-Reddy09].

3.5.1 Versioning File Systems

Versioning file systems such as Elephant [Santry99a], VersionFS [Muniswamy-Reddy04] and Ext3cow [Peterson05] incorporate versioning features directly into the file system. Versioning file systems automatically create commits as changes are made to data files. This offers increased convenience for the user who no longer needs to explicitly commit the updates to the data.

Versioning file systems also offer increased flexibility over application-level solutions. Versioning file systems operate at a lower level of the operating system and allow a finer granularity of control with per-file and per-directory policies [Santry99b]. In addition, applications that utilize versioning file systems benefit transparently without the need to explicitly encode the interaction with the versioning system.

Versioning file systems can also be used as a mechanism to ensure integrity. The practice of metadata snapshotting [Soares03] can be a simple and scalable mechanism for consistency.

Versioning systems also can be combined with other techniques to further enhance the user experience. Combining versioning with search can help users identify and distinguish one version of a file from another [Karlson11]. The addition of provenance to versioning can facilitate verifiable audit trails for versioning file systems [Burns05].

3.5.2 Version Control Systems (VCS)

Versioning systems originate with Version Control Systems (VCS). Such source code management applications provide capabilities including reversibility, concurrency, and annotation. VCS have been in existence for a long time with many mature programs like CVS [Berliner90], Subversion [Collins-Sussman11], and Git [Chacon11].

VCS are also use tools to facilitate collaboration. Developers can work in distributed environments and the versioning systems can handle the merges and updates. Most systems even allow parallel histories to exist through branch.

Unlike versioning file systems, VCS requires users to explicit specify when to create a version across files, since the relative versions among files are as important as the versions within a file. Otherwise, a restored source code tree may not be able to compile properly due to inconsistent versions of source files.

3.6 Dependency Tracking

Dependency tracking is yet another way to enrich the user experience with content history, and provide still more methods of enhanced access. These systems track the internal references from one application or library to another.

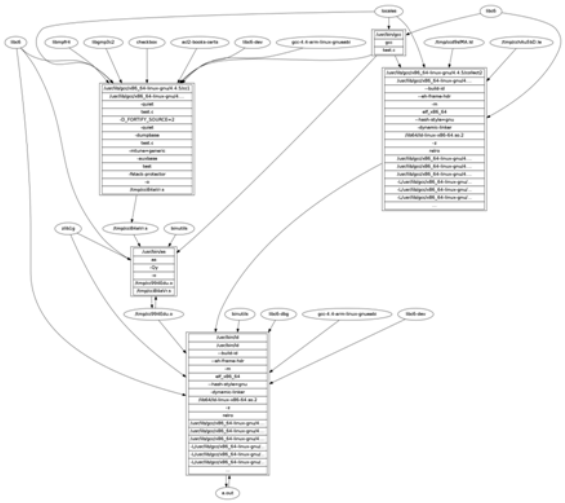

Figure 3.2: Compilation example

3.7 Capabilities & Benefits of Dependency Tracking

There are two main categories of usages for such technologies, the first with software configuration management (SCM) and build automation, and the second with intrusion detection and taint propagation.

3.7.1 Build Automation and SCM

Dependency-tracking systems can benefit software configuration management systems and help automate software builds. Systems such as Vesta [Heydon04] track the necessary files, libraries, etc. needed for compilation or execution. Somewhat related to versioning, such applications automate the arduous task of tracking which version of which library was used to compile which version of software.

Automate reconfiguration is even possible in the case that misconfiguration occurs, eliminating the need for the trial-and-error manual process as demonstrated in AutoBash [Su07].

Dependency tracking can also be used for integrity as well as to provide ACID semantics to the file system [Wright07]. It can also provide functionality similar to provenance, by using dependency analysis to provide explanations to administrators and auditors for policy decisions [Kagal08].

3.7.2 Intrusion and Malware Detection

Dependency-tracking systems are also capable of being used for intrusion detection. The data contents, size, and signature of a given file can be determined and stored by the system. (If the contents are changed by a legitimate user or the operating system proper, these signatures are updated to reflect the most recent version.) At any given time, the integrity of the file can be determined by comparing its integrity information with the referenced version.

Unfortunately, mere detection of an intruder alone is not sufficient. Administrators are interested in the subset of files that may have been affected following the attack. These tainted files can then be analyzed to determine if the integrity of the files has been compromised. Migrations and countermeasures can then be deployed as needed in response to the severity of the taint. Systems such as Dytan [Clause07], Panorama [Yin07], and TaintDroid [Enck10] exemplify this notion of dynamic taint tracking.

For many mission-critical systems, the ability to detect intrusions and track the propagation of taint is not sufficient. For such systems, downtime is simply not an option; recovery from such an event need be swift and transparent. Intrusion recovery systems such as Taser [Goel05] and TaintEraser [Zhu11] offer solutions to this problem. Similar in concept to versioning systems, these systems combine the intrusion detection capabilities of dependency tracking with the rollback functionality of versioning systems. When an intrusion occurs, the affected files can be restored to the last known taint-free state, and normal operation can continue.

3.8 Interactions between Users and Dependency Tracking

The better the dependency-tracking system for SCM and build automation, the less the user will need to interact with it. That is, if the system accurately maintains the correct library versions and connections of which goes with which, the user will be unaware of its existence. Interaction with the system may be required only in the event a misconfiguration occurs.

Similarly, with intrusion detection and build automation systems, the less the user needs interact, the better. In most every case, action will still be required; administrators will need to identify and neutralize the exploit used to gain unauthorized access.

3.9 Provenance

Provenance originates with the French provenir, "to come from," and refers to the history, or lineage of an object. Traditionally, the notion of provenance was used with works of art, or historical documents. The provenance of such pieces is supplemental information about authorship, ownership(s), etc. and helps establish the authenticity of the work.

The historical notion of provenance translates well into the modern computing environment, since provenance is, after all, just metadata. In addition to the authorship and ownership, data provenance can also include other metadata, such as storage location(s), or history of transformations throughout the life of the document.

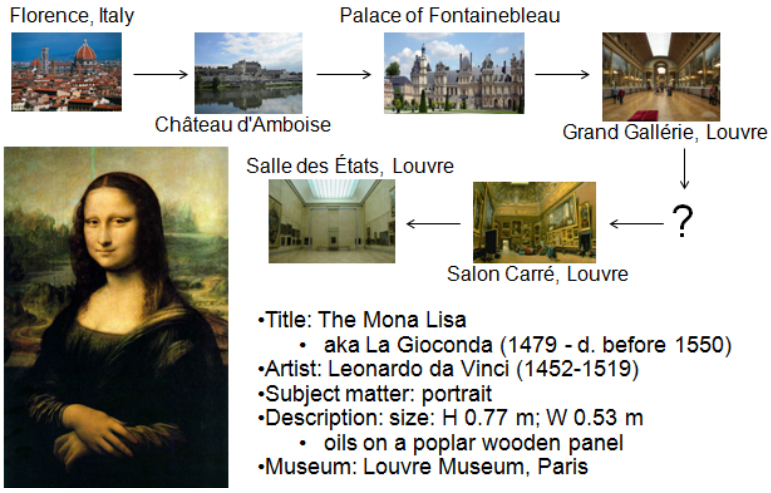

Figure 3.3: Provenance of the Mona Lisa

3.10 Capabilities & Benefits of Provenance

The capabilities offered by provenance systems cover many disciplines. They can be generalized into two primary application domains: scientific and business.

3.10.1 Scientific Domain

Scientific experiments from supercolliders and telescope arrays to climate and genetic simulations all generate massive amounts of data. These projects are largely a community effort [Simmhan05]. The data collected is made freely available to scientists over the world to facilitate scientific discovery.

Often, experiments are run with many configurations, and repeatability is paramount, so it is critical to track such configurations and lineages [Simmhan06]. Provenance addresses this need by providing a means to encode additional metadata such as information flow and workflow descriptions [Muniswamy-Reddy06] as part of the datasets.

As these projects progress and evolve, it is beneficial to understand how results were generated and how data entries were derived. Thus, the collections of data include not only the data itself, but also the relationships between data entities. This scientific provenance also encodes the procedures and derivations from one data set to another.

3.10.2 Business Domain

Business operations are also responsible for generating large volumes of data from financials to trade secrets. The provenance of this data can provide many benefits as well.

Modern business operations are subject to policies such as The Health Insurance Portability and Accountability Act [Hippa96] and the Public Company Accounting Reform and Investor Protection Act [Sarbanes-Oxley02]. These collectively govern the practices and mechanisms that must be enacted by corporations with regard to data privacy of health records and accuracy of financial data reporting. Data provenance can provide such an audit trail. The provenance can encode how standards and practices are executed, as well as parties involved, their respective roles, etc. In addition to legal compliance, the same techniques provide employers internal auditing at their own desired granularity.

Data provenance can also be used to track the creation of intellectual property. With a complete lineage of project development logs with accompanying metadata, it can be easier to attribute discoveries and inventions with the rightful creators. Similarly, data provenance can identify data origins in data warehouse.

In conjunction with data mining techniques, data provenance can facilitate knowledge discovery [Deolalikar09]. Analysis of the relationships between data entities and the associated provenance can give a better understanding of the data as well as determine less obvious relationships and hidden connections.

3.11 User/Provenance Interaction

Provenance collection can be either implicit or explicit. Explicit provenance collection relies on the direct input of the provenance metadata, typically used in the scientific domain. Scientists and researchers encode the data derivations directly into the provenance system.

Provenance can also be collected implicitly. In this operating method, the system will store the provenance automatically as the data is accessed or modified. This is more common in the business domain. Administrators can ensure the audit trail is recorded transparently by the provenance system.

Retrieval of providence metadata is typically provided by the systems in the form of queryable indices, graphs, and visualizations. Provenance queries enable administrators and auditors to locate the desired information quickly [Moreau08]. For scientists, provenance queries offer efficient responses to questions about history of experiments or simulations [Barga08].

3.10.1 Provenance with Other Techniques

Combining provenance with desktop search applications can further enhance the abilities to locate the desired data contents. Hybrid systems such as Provenance Based Retrieval [Yamamoto11] and Provenance for Personal File Search [Shah07] include data provenance as part of their searchable indexes. With the transfers of ownership encoded as metadata, the user is able to perform such queries as documents they received from their professor last week, or files collaborated with a colleague last year [Yamamoto11]. Also retained is the history of the application that created or modified the data file, providing support for searches such as files edited by Microsoft Word last month, or copy and pasted with Firefox yesterday [Jensen10].

This deeper understanding of data can be used to further enhance semantic search techniques. It can provide automatic semantic attribute extraction [Margo10] and enhance the quality of semantic search indices and file systems by providing additional dimensions of metadata for search [Golbeck08].

3.11 Pros and Cons of Content-history-based Approaches

Content history methods can track previous versions of files. These versioned data entries provide recovery and rollback capabilities to the user. These systems can also help facilitate team collaboration, and distributed interaction.

Content history can encode file dependency information. This can help automate development environments and make configurations more portable. They can also help ensure system integrity and provide notification of intrusion, and track taint propagation.

Content-history-based methods can identify and track data lineage and provenance. This provides research scientists the data derivations necessary to perform experiments and simulations. It also provides administrators the ability to audit for regulatory purposes. Further, it can provide rich new methods to analyze datasets.

However, history is not always easy to track. Many operations can cause nondeterministic lineages, and moving and renaming of data can be problematic.

Scalability and performance issues are also major concerns with provenance, versioning and dependency tracking systems alike. The finer the granularity of stored content lineage, the greater the storage requirement will be for the system. Further, it can be difficult to determine the appropriate granularity to store the histories [Muniswamy-Reddy09]. Storage of the derivations at an appropriate level of abstraction useful to researchers can also be a challenge [Simmhan05].

CHAPTER FOUR

CONTENT CONTEXT

4.1 Overview

Context can be defined as that which surrounds us and gives meaning to something else [Howe06]. For CPU executions, context can be associated with available resources for given process or thread, it can also be associated with the state of the process. This includes states such as processor registers and memory allocations, as well as file system state, dependencies etc. An alternative perspective is the minimal information that must he saved to allow for correct executions amidst interruptions and CPU context switches. Such context is largely covered in dependency tracking as a part of the previous section on context history.

Figure 4.1: Examples of context

Orthogonal to traditional computing context is environmental context. This is the context relates to a much higher level of data derivation. It can include data collected from onboard sensors, like GPS, accelerometer, ambient light, to web services, and social media that provide

functionality like geocoding, weather, traffic, and news. This diverse spectrum of inputs translates to equally broad spectrum of collected contextual clues.

Spatial. Context systems can deduce the location, orientation, and speed of the user and the devices given sensory input from accelerometers, magnetometers and GPS.

Temporal. Context systems are aware of the time of day, day of the week, and the season of the year.

Environmental. Context systems can also help determine the local environment through sensory input from temperature, ambient light, and noise as well as related web service content such as weather forecasts.

Social. Context systems can be aware of people nearby, their activities, as well as calendar and scheduling leading to better understanding of the social environment.

Resources. With indices of potentially available resources, context-aware systems can determine which are currently available or nearby.

Physiological. Context-aware systems can even be aware of physiological context including heart rate and blood pressure, respiration, perspiration, even tone of voice.

Context-aware systems exploit these contextual resources to enhance user experience. These systems can interact with the contextual data either indirectly (passively) or directly (actively).

Passive context systems can produce responses dependent upon the context in which the command is issued. Passive systems can also provide contextual annotation. This encodes contextual metadata into the data to help the user organize their data. Passive context systems can also provide proximate selection, emphasizing or otherwise making easier to choose options relevant to the given context.

Active context-aware systems can provide automatic contextual reconfiguration. System and application components can be added, removed, or modified based on triggers from changing context. Active systems can also provide context-triggered actions. Policies can be executed automatically based with context changes.

4.2 Indirect (Passive) Context-aware Systems

Contextual information can be stored for later use by passive context annotation systems. The contextual metadata can provide clues for enhancements and optimizations, such as automatic generation of metadata. Systems such as from context to content [Davis04] utilize spatial and temporal context to generate metadata for media files. This can help save the user the tedious task of manually maintaining such meta-tags and embed situational metadata into multimedia files as they are created. Metadata creation system for mobile images [Sarvas04] interacts with the user periodically to supplement the inferred metadata. Face recognition provides TagSense [Qin11] the ability to senses the people, activity of an image and uses this data to generate tags on-the-fly.

Contextual metadata can also be used to enhance the capability of search applications. iMecho [Chen11] utilizes context from user activity logs and Hidden Markov Model to estimate context during user query. The Connections [Soules05] system identifies temporal relationships between files by tracing file system calls and uses this knowledge to expand and reorder content-based search results. However, file access and trace-based approaches may have limited success due increasingly abstract interactions between application and the filesystem. Confluence [Gyllstrom07] overcomes this obstacle by augmenting the file event stream with window focus events from the UI layer.

4.3 Direct (Active) Context Awareness

In contrast to passive systems, active context can automatically change their behaviors in response to changing context. Context-aware systems that provide active adaptation can be organized into three main models: *laissez faire* adaptation, application-aware adaptation, and application-transparent adaptation.

4.3.1 *Laissez Faire* Adaptation

With the *laissez faire* approach, file systems do not interact with contexts. Instead, applications handle their own adaptations on an ad hoc basis. This method offers flexibility in how and when the context monitoring and adaptation occurs.

Flexibility comes with a price. With everything requiring custom constructs, this method incurs the most development effort. For this reason, general usage in practice is rare with only application-specific examples used in practice.

4.3.2 Application-aware-adaptation-based Systems

In application-aware-adaptation-based systems, the context collection and adaptation is centralized and coordinated. For convenience and efficiency, system support or middleware(s) are required. These systems collect data from the various sensory inputs and use it to make contextual derivations. This information is made available for application to use through APIs.

Thus, to utilize the context data, applications must be modified to interact with these systems, the specific adaptive behavior needs to be specifically encoded. Odyssey [Noble99] exemplifies this behavior with fidelity—a quantifiable, type-specific notion of quality, adaptive in differing contexts. Applications define their specific data fidelities and parameters for contextual adaptation. For example, a video player switching to a lower video resolution when less network bandwidth is available [Noble99], or degrading application quality to conserve battery life [Flinn99].

Even though these systems utilize centralized constructs, they still require significant development effort for efficient operation. In addition, programmers must conform to the requirements of the context APIs, which can be restrictive. Finally, opportunities for automated operation are missed since context adaptation must be explicitly encoded per application, in advance.

4.3.3 Application-transparent Adaptation

Application-transparent, context-aware adaptation systems handle all aspects of context collection and adaptation internally. Unlike application-aware systems, contextual behaviors need not be encoded explicitly. The system is capable of handling the adaptation with the program to be specifically coded to interact with it. In fact, the application need not even know the existence of the context adaption and can still reap the benefits from the system transparently.

Of the three methods, this is the most desirable environment for application developers. Developers can create applications using standard environments and tools without the need to

conform to the context system. Applications can still transparently receive the benefits of context adaptive behavior.

With all collection and behavior managed below application environments, transparent adaptation provides support for rich policies. quFiles [Veeraraghavan10], for instance, bundles arbitrary data types transparently within the file system. Then, dependent upon contextual situations such as platform, external devices, connectivity, or battery power, differing results can be returned from the file system. Examples usage of such context policies include tailoring images and video content based on the screen dimensions of a mobile device [Fox96], or context-aware data redaction [Veeraraghavan10].

4.4 Pros and Cons of Context-based Approaches

System awareness of the surroundings can translate into many capabilities provide to the user. The clues provided by context can help trim the search space for data retrieval. The contextual meta-data can return results that are more intuitive to the desired query.

The availability of content context can also lead to dynamic file contents. Depending on the current conditions surround the file system access, the system can provide different versions for different contexts.

Content context also provides system support for rich policies. Application behavior can be tailored to contextual situations, providing much greater level of control.

Context awareness can also be applied to other techniques. The addition of context can provide more explanation to modifications and policy decisions. It can also lend clues to dependency tracking and intrusion detection systems.

However, many aspects of context-aware system remain challenging. It is difficult to find a balance between manual specifications and automated behavior.

It is also difficult to quantify the effectiveness of context-aware systems. With such subjective metrics as effectiveness of adaptation or search optimizations, traditional evaluation methods are not sufficient. These systems require an inherent human component, and this is problematic with regard to privacy issues with human subjects and difficulty in location of useful data sets for testing.

Migration to context-aware systems can also be problematic. Non-traditional interfaces can require potential learning curves for uses and developers alike. Many context-aware adaptations need to be manually specified, and this can require extensive effort by the user or application developer. Additionally, users need to understand the limitations and non-deterministic application execution.

Finally, context-aware systems also must wrestle with the same performance and overhead issues as content-history-based approaches. With so much additional information available, it can be problematic to determine the appropriate storage granularity. Without careful considerations, context-aware systems may incur unnecessary storage overheads by maintaining irrelevant or non-useful contextual data.

CHAPTER FIVE

CONCLUSION

5.1 A Wealth of Capabilities

This survey has demonstrated how the file system can be extended to provide the user with a diverse spectrum of capabilities. Systems can utilize the similarity of contents to provide searchable indices that can be extended through tailored and localized results. Systems can utilize the semantic relationships between data to provide results with a deeper understanding of the data and their relationships

File-system enhancements can provide regulatory compliance for auditors and administrators, and rich explanations of data derivations for scientists and researchers. They can offer backup, rollback, and help facilitate distributed operation. They can provide automated build environments and software configuration management, as well as intrusion detection and integrity assurance through the use of dependency tracking methods.

The addition of context can provide even more extensibility. Systems can offer context-based modules and configurations, as well as provide better localized and customized results. Systems can also provide adaptive behavior and execution transparently for given contextual triggers.

5.2 No Silver Bullet

While it is clear that collectively these systems have much to offer, this survey also demonstrates that an overarching solution does not yet exist. Content-similarity-based enhancement methods lack the lineage of the data.

Content-history-based enrichment methods are able to overcome this deficiency through provenance, versioning, and dependency tracking. However, these systems can be tricky to implement and often have prohibitive overhead.

Content-context-based enhancements can provide intuitive and adaptive behavior through environmental and situational awareness. However, the effectiveness of such systems is difficult to evaluate.

5.3 Conclusion

This survey explores the capabilities that can be provided by the file system. It outlined methods that exploit content similarity, content history, and content context to enhance data retrieval. The approach focused on the interactions from the perspective of the end users, and showcased benefits across a broad scope of disciplines. Finally, it outlined the shortcomings of each method and demonstrated the lack of an overarching solution.

REFERENCES

[Agrawal07] Nitin Agrawal, William J. Bolosky, John R. Douceur, and Jacob R. Lorch. 2007. A five-year study of file-system metadata. *ACM Transactions on Storage* (TOS). 3(3), Article 9 (October 2007).

[Agrawal09] Nitin Agrawal, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2009. Generating realistic impressions for file-system benchmarking. *ACM Transactions on Storage* (TOS). 5(4), Article 16 (December 2009), 30 pages.

[Soules03] Craig A. N. Soules and Gregory R. Ganger. 2003. Why can't I find my files? new methods for automating attribute assignment. *In Proceedings of the 9th Conference on Hot Topics in Operating Systems* – 9(9):20-20. USENIX Association, Berkeley, CA, USA.

[Seltzer09] Margo Seltzer and Nicholas Murphy. 2009. Hierarchical file systems are dead. In *Proceedings of the 12th Conference on Hot Topics in Operating Systems (HotOS'09).* USENIX Association, Berkeley, CA, USA, 1-1.

[Brin98] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*. 30, April 1998, 107-117.

[Berners-Lee01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[Wolber02] David Wolber, Michael Kepe, and Igor Ranitovic. 2002. Exposing document context in the personal web. *In Proceedings of the 7th International Conference on Intelligent User interfaces (IUI '02).* ACM, New York, NY, USA, 151-158.

[Google05] Google, Inc. 2005. Google search engine. http://www.google.com.

[Microsoft06] Microsoft, Inc. 2006. Windows Live Search. http://windows.microsoft.com/en-US/windows7/products/features/windows-search

[Yahoo02] Yahoo!. 2002. Yahoo! Directory. http://dir.yahoo.com/.

[Dmoz02] The Open Directory Project. 2002. About the Open Directory Project. http://www.dmoz.org/.

[Google08] Google, Inc. 2008. Google desktop. http://desktop.google.com/features.html#search.

[Microsoft06] Microsoft, Inc. 2006. Windows Desktop Search. http://windows.microsoft.com/en-US/windows7/products/features/windows-search

[Copernic07] Copernic, Inc. 2007. Copernic desktop search. www.copernic.com/.

[Apple08] Apple, Inc. 2008. Spotlight. http://www.apple.com/macosx/features/spotlight/.

[Schandl09] Bernhard Schandl and Bernhard Haslhofer. 2009. The Sile Model -- A Semantic File System Infrastructure for the Desktop. *In Proceedings of the 6th European Semantic Web Conference on The Semantic Web: Research and Applications (ESWC 2009 Heraklion)*, Lora Aroyo, Paolo Traverso, Fabio Ciravegna, Philipp Cimiano, Tom Heath, Eero Hyvönen, Riichiro Mizoguchi, Eyal Oren, Marta Sabou, and Elena Simperl (Eds.). Springer-Verlag, Berlin, Heidelberg, 51-65.

[Sinha05] Vineet Sinha and David R. Karger. 2005. Magnet: supporting navigation in semistructured data environments. *In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD '05)*. ACM, New York, NY, USA, 97-106.

[Cutrell06] Edward Cutrell, Daniel Robbins, Susan Dumais, and Raman Sarin. 2006. Fast, flexible filtering with phlat. *In Proceedings of the SIGCHI conference on Human Factors in Computing Systems (CHI '06)*, Rebecca Grinter, Thomas Rodden, Paul Aoki, Ed Cutrell, Robin Jeffries, and Gary Olson (Eds.). ACM, New York, NY, USA, 261-270.

[Perugini10] Saverio Perugini. 2010. Supporting multiple paths to objects in information hierarchies: Faceted classification, faceted search, and symbolic links. *Information Processing and Management*. 46(1):22-43, January 2010.

[Dumais03] Susan Dumais, Edward Cutrell, JJ Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. 2003. Stuff I've seen: a system for personal information retrieval and re-use. *In Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR '03)*. ACM, New York, NY, USA, 72-79.

[Karger04] David R. Karger and Dennis Quan. 2004. Haystack: a user interface for creating, browsing, and organizing arbitrary semistructured information. *In Proceedings of CHI '04 Extended Abstracts on Human Factors in Computing Systems (CHI EA '04)*. ACM, New York, NY, USA, 777-778.

[Gifford91] David K. Gifford, Pierre Jouvelot, Mark A. Sheldon, and James W. O'Toole, Jr.. 1991. Semantic file systems. *In Proceedings of the Thirteenth ACM Symposium on Operating Systems Principles (SOSP '91)*. ACM, New York, NY, USA, 16-25.

[Hua09] Yu Hua, Hong Jiang, Yifeng Zhu, Dan Feng, and Lei Tian. 2009. SmartStore: a new metadata organization paradigm with semantic-awareness for next-generation file systems. *In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*. ACM, New York, NY, USA, , Article 10 , 12 pages.

[Leung09] Andrew W. Leung, Minglong Shao, Timothy Bisson, Shankar Pasupathy, and Ethan L. Miller. 2009. Spyglass: fast, scalable metadata search for large-scale storage systems. *In Proceedings of the 7th Conference on File and Storage Technologies (FAST '09)*, Margo Seltzer and Ric Wheeler (Eds.). USENIX Association, Berkeley, CA, USA, 153-166.

[Chirita06] Paul-Alexandru Chirita, Stefania Costache, Wolfgang Nejdl, and Raluca Paiu. 2006. Beagle++: semantically enhanced searching and ranking on the desktop. *In Proceedings of the 3rd European Conference on The Semantic Web: Research and Applications (ESWC'06)*, York Sure and John Domingue (Eds.). Springer-Verlag, Berlin, Heidelberg, 348-362.

[Cohen08] Sara Cohen, Carmel Domshlak, and Naama Zwerdling. 2008. On ranking techniques for desktop search. *ACM Transactions on Information Systems (TOIS)*. 26(2), Article 11, April 2008, 24 pages.

[Balmin04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. 2004. Objectrank: authority-based keyword search in databases. *In Proceedings of the Thirtieth International Conference on Very Large Data Bases*, 2004.

[Princeton10] Princeton University "About WordNet." WordNet. Princeton University. 2010. http://wordnet.princeton.edu

[Berliner90] Berliner, B., CVS II: parallelizing software development. *In Proceedings of the Winter 1990 USENIX Conference*, (Washington DC, USA,), USENIX Assoc., 1990, 341--352.

[Collins-Sussman11] Collins-Sussman, B., Fitzpatrick, B. W., and Pilato, C. M. 2011. Version Control with Subversion. O'Reilly Media. http://svnbook.red-bean.com/

[Chacon11] Chacon, S. 2011. Git Documentation. http://git- scm.com/documentation

[Santry99a] Douglas J. Santry, Michael J. Feeley, Norman C. Hutchinson, and Alistair C. Veitch. 1999. Elephant: The File System that Never Forgets. *In Proceedings of the Seventh Workshop on Hot Topics in Operating Systems (HOTOS '99)*. IEEE Computer Society, Washington, DC, USA, 2-8.

[Santry99b] Douglas S. Santry, Michael J. Feeley, Norman C. Hutchinson, Alistair C. Veitch, Ross W. Carton, and Jacob Ofir. 1999. Deciding when to forget in the Elephant file system. *ACM SIGOPS Operating Systems Review*. 33(5):110-123, December 1999.

[Muniswamy-Reddy04] Kiran-Kumar Muniswamy-Reddy, Charles P. Wright, Andrew Himmer, and Erez Zadok. 2004. A Versatile and User-Oriented Versioning File System. *In Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST '04)*. USENIX Association, Berkeley, CA, USA, 115-128.

[Soares03] Livio B Soares, Orran Y Krieger, and Dilma Da Silva. 2003. Meta-data snapshotting: a simple mechanism for file system consistency. *In Proceedings of the International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI '03)*. ACM, New York, NY, USA, 41-52.

[Peterson05] Zachary Peterson and Randal Burns. 2005. Ext3cow: a time-shifting file system for regulatory compliance. *Transactions on Storage*, 1(2):190-212, May 2005.

[Burns05] Randal Burns, Zachary Peterson, Giuseppe Ateniese, and Stephen Bono. 2005. Verifiable audit trails for a versioning file system. *In Proceedings of the 2005 ACM Workshop on Storage Security and Survivability (StorageSS '05)*. ACM, New York, NY, USA, 44-50.

[Karlson11] Amy K. Karlson, Greg Smith, and Bongshin Lee. 2011. Which version is this?: improving the desktop experience within a copy-aware computing ecosystem. *In Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 2669-2678.

[Muniswamy-Reddy09] Kiran-Kumar Muniswamy-Reddy and David A. Holland. 2009. Causality-based versioning. *ACM Transactions on Storage (TOS)*. 5(4), Article 13, December 2009, 28 pages.

[Heydon04] Allan Heydon, Roy Levin, Timothy Mann, and Yuan Yu. 2004. Software Configuration Management System Using Vesta (Monographs in Computer Science). Telos Pr.

[Wright07] Charles P. Wright, Richard Spillane, Gopalan Sivathanu, and Erez Zadok. 2007. Extending ACID semantics to the file system. *ACM Transactions on Storage (TOS)*. 3(2), Article 4 (June 2007).

[Su07] Ya-Yunn Su, Mona Attariyan, and Jason Flinn. 2007. AutoBash: improving configuration management with operating system causality analysis. *In Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles (SOSP '07)*. ACM, New York, NY, USA, 237-250.

[Kagal08] Lalana Kagal, Chris Hanson, and Daniel Weitzner. 2008. Using Dependency Tracking to Provide Explanations for Policy Management. *In Proceedings of the 2008 IEEE Workshop on Policies for Distributed Systems and Networks (POLICY '08)*. IEEE Computer Society, Washington, DC, USA, 54-61.

[Chirita05] Paul Alexandru Chirita, Rita Gavriloaie, Stefania Ghita, Wolfgang Nejdl, and Raluca Paiu. 2005. Activity based metadata for semantic desktop search. *In Proceedings of the Second European Conference on The Semantic Web: Research and Applications (ESWC'05)*, Asunción Gómez-Pérez and Jérôme Euzenat (Eds.). Springer-Verlag, Berlin, Heidelberg, 439-454.

[Goel05] Ashvin Goel, Kenneth Po, Kamran Farhadi, Zheng Li, and Eyal de Lara. 2005. The taser intrusion recovery system. *In Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05)*. ACM, New York, NY, USA, 163-176.

[Clause07] James Clause, Wanchun Li, and Alessandro Orso. 2007. Dytan: a generic dynamic taint analysis framework. *In Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07)*. ACM, New York, NY, USA, 196-206.

[Yin07] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. 2007. Panorama: capturing system-wide information flow for malware detection and analysis. *In*

Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07). ACM, New York, NY, USA, 116-127.

[Zhu11] David (Yu) Zhu, Jaeyeon Jung, Dawn Song, Tadayoshi Kohno, and David Wetherall. 2011. TaintEraser: protecting sensitive data leaks using application-level taint tracking. *ACM SIGOPS Operating Systems Review*. 45(1):142-154, February 2011.

[Enck10] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 1-6.

[Hippa96] Centers for Medicare & Medicaid Services. The Health Insurance Portability and Accountability Act of 1996 (HIPAA). http://www.cms.hhs.gov/hipaa/, 1996.

[Sarbanes-Oxley02] U.S. Public Law No. 107-204, 116 Stat. 745. The Public Company Accounting Reform and Investor Protection Act, 2002.

[Simmhan05] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. 2005. A survey of data provenance in e-science. *SIGMOD* Rec. 34(3):31-36, September 2005.

[Simmhan06] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. 2006. A Framework for Collecting Provenance in Data-Centric Scientific Workflows. *In Proceedings of the IEEE International Conference on Web Services (ICWS '06)*. IEEE Computer Society, Washington, DC, USA, 427-436.

[Barga08] Roger S. Barga and Luciano A. Digiampietri. 2008. Automatic capture and efficient storage of e-Science experiment provenance. *Concurrency and Computation: Practice & Experience - The First Provenance Challenge*. 20(5):419-429, April 2008.

[Moreau08] Luc Moreau, Paul Groth, Simon Miles, Javier Vazquez-Salceda, John Ibbotson, Sheng Jiang, Steve Munroe, Omer Rana, Andreas Schreiber, Victor Tan, and Laszlo Varga. 2008. The provenance of electronic data. *Communications of the ACM*. 51(4):52-58, April 2008.

[Muniswamy-Reddy06] Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, and Margo Seltzer. 2006. Provenance-aware storage systems. *In Proceedings of the Annual Conference on USENIX '06 Annual Technical Conference (ATEC '06)*. USENIX Association, Berkeley, CA, USA, 4-4.

[Golbeck08] Jennifer Golbeck and James Hendler. 2008. A Semantic Web approach to the provenance challenge. *Concurrency and Computation: Practice & Experience - The First Provenance Challeng*. 20(5):431-439, April 2008.

[Deolalikar09] Vinay Deolalikar and Hernan Laffitte. 2009. Provenance as data mining: combining file system metadata with content analysis. *In Proceedings of the First Workshop on*

Theory and Practice of Provenance (TAPP'09). USENIX Association, Berkeley, CA, USA, Article 10, 10 pages.

[Muniswamy-Reddy09] Kiran-Kumar Muniswamy-Reddy, Uri Braun, David A. Holland, Peter Macko, Diana Maclean, Daniel Margo, Margo Seltzer, and Robin Smogor. 2009. Layering in provenance systems. *In Proceedings of the 2009 Conference on USENIX Annual technical conference (USENIX'09)*. USENIX Association, Berkeley, CA, USA, 10-10.

[Jensen10] Carlos Jensen, Heather Lonsdale, Eleanor Wynn, Jill Cao, Michael Slater, and Thomas G. Dietterich. 2010. The life and times of files and information: a study of desktop provenance. *In Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 767-776.

[Margo10] Daniel Margo and Robin Smogor. 2010. Using provenance to extract semantic file attributes. *In Proceedings of the 2nd Conference on Theory and Practice of Provenance (TAPP'10)*. USENIX Association, Berkeley, CA, USA, 7-7.

[Yamamoto11] Keiko Yamamoto, Taku Kuriyama, Haruki Shigemori, Itaru Kuramoto, Yoshihiro Tsujino, and Mitsuru Minakuchi. 2011. Provenance Based Retrieval: File Retrieval System Using History of Moving and Editing in User Experience. *In Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference (COMPSAC '11)*. IEEE Computer Society, Washington, DC, USA, 618-625.

[Shah07] Sam Shah, Craig A. N. Soules, Gregory R. Ganger, and Brian D. Noble. 2007. Using provenance to aid in personal file search. *In Proceedings of the 2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference (ATC'07)*, Jeff Chase and Srinivasan Seshan (Eds.). USENIX Association, Berkeley, CA, USA, Article 13 , 14 pages.

[Fox96] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. 1996. Adapting to network and client variability via on-demand dynamic distillation. *In Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*. ACM, New York, NY, USA, 160-170.

[Noble99] B. D. Noble and M. Satyanarayanan. 1999. Experience with adaptive mobile applications in Odyssey. *Mobile Networks and Applications*. 4(4):245-254, December 1999.

[Howe06] Howe, D., Free on-line dictionary of computing. 2006, Imperial College Department of Computing London, UK.

[Veeraraghavan10] Kaushik Veeraraghavan, Jason Flinn, Edmund B. Nightingale, and Brian Noble. 2010. quFiles: The right file at the right time. *ACM Transactions on Storage (TOS)*. 6(3), Article 12 (September 2010), 28 pages.

[Qin11] Chuan Qin, Xuan Bao, Romit Roy Choudhury, and Srihari Nelakuditi. 2011. TagSense: a smartphone-based approach to automatic image tagging. *In Proceedings of the 9th*

International Conference on Mobile Systems, Applications, and Services (MobiSys '11). ACM, New York, NY, USA, 1-14.

[Chen11] Jidong Chen, Hang Guo, Wentao Wu, and Wei Wang. 2011. iMecho: a context-aware desktop search system. *In Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11)*. ACM, New York, NY, USA, 1269-1270.

[Gyllstrom07] Karl Anders Gyllstrom, Craig Soules, and Alistair Veitch. 2007. Confluence: enhancing contextual desktop search. *In Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '07)*. ACM, New York, NY, USA, 717-718.

[Soules05] Craig A. N. Soules and Gregory R. Ganger. 2005. Connections: using context to enhance file search. *ACM SIGOPS Operating Systems Review*. 39(5):119-132, October 2005.

[Azizyan09] Martin Azizyan, Ionut Constandache, and Romit Roy Choudhury. 2009. SurroundSense: mobile phone localization via ambience fingerprinting. *In Proceedings of the 15th Annual International Conference on Mobile Computing and Networking (MobiCom '09)*. ACM, New York, NY, USA, 261-272.

[Beach10] Aaron Beach, Mike Gartrell, Xinyu Xing, Richard Han, Qin Lv, Shivakant Mishra, and Karim Seada. 2010. Fusing mobile, sensor, and social data to fully enable context-aware computing. *In Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications (HotMobile '10)*. ACM, New York, NY, USA, 60-65.

[Sarvas04] Risto Sarvas, Erick Herrarte, Anita Wilhelm, and Marc Davis. 2004. Metadata creation system for mobile images. *In Proceedings of the 2nd international conference on Mobile systems, applications, and services (MobiSys '04)*. ACM, New York, NY, USA, 36-48.

[Davis04] Marc Davis, Simon King, Nathan Good, and Risto Sarvas. 2004. From context to content: leveraging context to infer media metadata. *In Proceedings of the 12th annual ACM international conference on Multimedia (MULTIMEDIA '04)*. ACM, New York, NY, USA, 188-195.