# *Praana*: A Personalized Desktop Filesystem

Amit Roy[1], Sundar Balasubramaniam[2]
[1]Student, Computer Science and Information Systems Group
[2]Associate Professor, Computer Science and Information Systems Group
*Birla Institute of Technology and Science, Pilani*
{h2006039, sundarb}@bits-pilani.ac.in

*Owing to the absence of rich contextual cues, filesystem search has been full of noise, imprecision and very less recall. Contextual information in the form of annotations (tags defined by users or extracted from files) and data/process provenance would improve the personal search experience of the user. A filesystem architecture to support contextual desktop search is proposed. Implementation issues are also discussed.*

## 1. Introduction

Navigation and search are the two ways of locating information. Historically, navigating through a hierarchy has been the primary one, with examples ranging from catalogues of a library to a recipe in a cookbook to a pay bill in a file folder. On the other hand searching has been a relatively new phenomenon. But with data explosion and increased rate of information production combined with the dearth of techniques to store them in a structured manner, search has become an equally (if not the more) important methodology for accessing information.

Advent of various Web Search Engines like Google, Yahoo, Windows Live, Ask etc., has resulted in a revolution in search technologies. In the Web, this has been possible not only because of the advancements in Information Retrieval techniques but also because of a vast number of high performance index servers, running in parallel. When compared in terms of the data that is handled, filesystem search might seem like a distant poor cousin of web search, but it has its own set of challenges, in terms of the resources available (CPU, Memory, Secondary Storage), data security, privacy, provenance, dearth of contextual information and others.

The problem with the present solutions to Filesystem Search (also known as "Desktop Search") is that they have minimal support from the filesystem and have to rely on content indexing only. Indexing has its own share of problems since not all contents can be analysed (e.g. multimedia) and often the "context" based on which the data was organised by the user is not captured. Thus, search in filesystem cannot rely only on syntax, or popularity, both of which have been exploited to the hilt on the Web. For solving this problem existing filesystem architectures needs to be modified to gather more contextual information. These architectures should also support robustness by helping to recover from data corruption, loss of information and other related errors. This paper proposes a design to do the same and a reasonable set of parameters to evaluate the solution. The remainder of the paper is organised as follows: Section 2 provides a brief overview of current research in desktop search, filesystem robustness, and file system architectures. Section 3 outlines the proposed architecture. Section 4 concludes with some implementation details and current focus of the work.

## 2. Related Work

Development of Desktop Search Engines started with content indexing [2, 3], gradually moving to semantic [4, 5] and full text search followed by Context Based Search [6, 7, 8, 9] which is the focus of researchers today. Filesystem Versioning was implemented in [10] to increase robustness. Some of the standard Data Provenance Techniques [11] developed for e-sciences were adopted in [12]. Filesystems that move away from the hierarchical structure for better usability were also developed [7, 13]. These are summarized in Table 1.

The drawbacks of all these efforts (see last column of Table 1) have been either substantial dependency on users or not being able to identify appropriate search parameters.

The proposed architecture models "User Activities" in terms of Provenance, Temporal Locality and Annotations to enable a better search experience. This paradigm helps to achieve maximum impact with minimum input. Although partial use of provenance has been used as a source of contextual information in [9], the parameters included in this proposal have not been explored earlier. Some of the above issues

| Filesystem | Indexing/Context | Features | Limitations |
|---|---|---|---|
| ***Content Based Indexing and Search*** | | | |
| Glimpse [2] | Content Based | first full text search engine to use the concepts of Inverted Index and sequential search techniques | - content analysis has limited scope<br>- context is not captured |
| Phlat [3] | | tackles search by bridging the dichotomy of search and browse; allows tagging | |
| ***Full Text Indexing and Semantic Search*** | | | |
| Stuff I have seen [4] | Full Text | implements *faceted search* by creating a unified index of information seen by a user, irrespective of where and how it has been seen | - manual annotation is cumbersome<br>- automatic attribute collection is bounded by content |
| Haystack [5] | Semantic | uses annotations and collections based on ontological agent approach | |
| ***Context Based Search*** | | | |
| MyLifeBits [6] | "Story" (sequence of events using media) | *links* created by user annotates a digital resource (mails, files, multimedia etc) stored in database, with another | manual dependency for story creation |
| Lifestreams [7] | "Substreams" (view of a collection of documents) | - moves away from hierarchical structure; represents filesystem as a time based visualisation (electronic diary)<br>- searching is done by "find" operation that creates "Substreams" which can be stored | |
| Connections [8] | "Successor models" | "Temporal Locality" based relationships are stored in the form of Relation Graphs | capturing context adds a lot of noise |
| [9] | "Provenance" (Strict Causality) | is a modification on Connections and uses partial data from provenance to implement context based search | irrelevant relations between data files |
| ***Other novel implementations*** | | | |
| Comprehensive Versioning Filesystem [10] | - implements metadata versioning through Journaling and Multiversion B-Trees (a variation on standard B-Tree)<br>- inode attributes, file pointer and directory entry data structures were modified | | both PASS and CVFS does not have any mechanism for querying & retrieving the stored data |
| Provenance Aware Storage-System [12] | was the first attempt to capture provenance in filesystem; stores provenance data in a database; | | |
| LiFS [13] | moves away from hierarchical structure; implements filesystem as files which are connected by links that contain rich contextual attributes to represent relationships between files | | requires additional hardware (MRAM: a new type of RAM) |
| ***Commercial Products*** | | | |
| Google Desktop Search | | - per-user index<br>- full-text indexing<br>- low disk area consumption for indexes | - content based index<br>- content preserving changes are not captured |
| Windows Desktop Engine | | | |
| X1 | | | |

**Table 1**: Summary of desktop search and novel filesystem implementations

are addressed by our system (see Table 2).

| Praana |
|---|
| Context based search |
| Automatic provenance collection |
| Captures User activity (Provenance: Data and Process ) |
| Tagging |
| Archiving, Checkpointing |
| Minimal manual dependency |

**Table 2**: Design highlights

## 3. Architecture

The Architecture (Figure 1) introduces three components spread across three layers - a User Interface (UI), a Metadata Repository (that stores provenance data, indexes, archived versions and relation graphs [11]) and a User Activity Aggregator (UAA) module for capturing user activities. The filesystem allows users to enter Annotations. The Metadata Repository consists of two kinds of information – available to the user (annotations and provenance) and not available (indexes, archived versions and relation graphs). When a user performs an operation on a file, the Provenance Collection Engine captures both data and process provenance information. For example if a user runs a spell checker on a document, then the modified spellings are stored as file data and also as part of data provenance. Whereas the information that the spell checker was run is captured as part of process provenance. The process section of the UAA contains an Inference Learning Engine (ILE) that analyses this data for generating contextual cues. The Analyser maintains the contextual information as relation graphs. It takes inputs from the Provenance Collection Engine and modifies the graph accordingly. The annotations are stored as part of the file inode. Also two kinds of indexes are maintained: content based and annotation based**.**

### 3.1 Provenance & Inference

Provenance Information is stored in a database. Every access to the file and the corresponding changes are recorded. In addition, operations that caused these changes are also recorded. Operations are recorded by trapping either

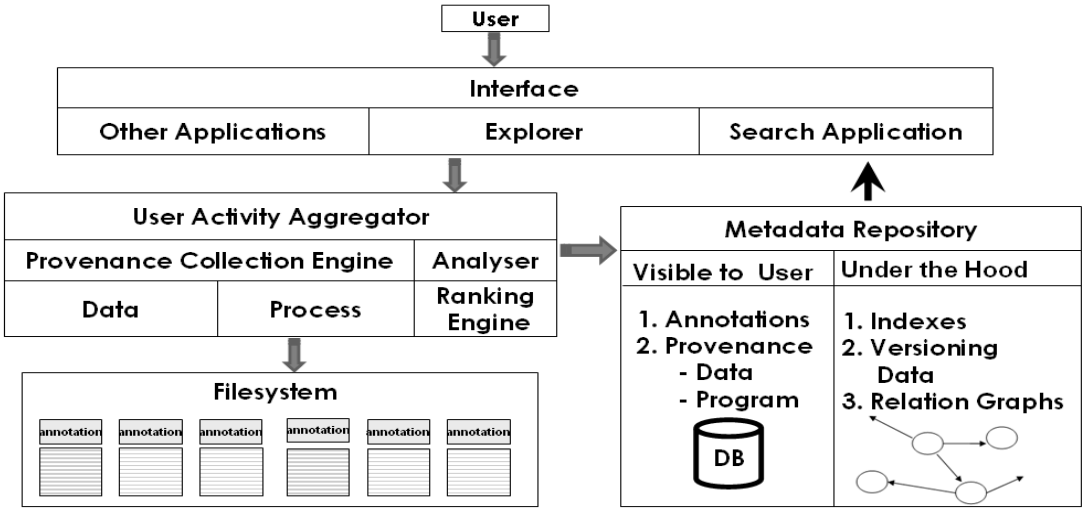


**Figure 1:** System Architecture

functions associated with specific file types or generic tasks defined in the filesystem. The ILE will tune the contextual data based on heuristics generated by the usage of the filesystem. Over a period of time it will learn the effect of the use of a program on a file. The engine learns and Analyser for modifying the relation graphs. When a program is run, a record is entered into the categorizes "Rules" that are passed on to the database about the files that are affected. The first action on the files after the program execution is also recorded. And a corresponding rule is deduced. Consider a scenario, where AutoSummarize (function in Microsoft Word to summarize a document) is used to create the summary S of a file F1 and S is moved to a newly created file F2. File F2 is then dispatched as part of a reply using a Mail client. The data provenance information induces a new context relationship between the F1, F2, and the reply mail. If F2 is never used again or is deleted, the ILE will infer a rule that the source and destination files of an AutoSummarize operation should not be related. However, this rule will initially be assigned a low Confidence Measure which may be reinforced by user feedback or by recurrence of the same operation.

### 3.2 File Clusters & Relation Graph

Context enhanced search solutions that exist today do not address changes in the context of a file. In the relation graphs, edge weights increases depending on the heuristics defined, but they never decrease. When the context of a file weakens its relations over a period of time this is not reflected in the graph. For example, a file on *brokerage firms* might be related to other files associated with *banks*. Accordingly the graphs are formed. However, a few months and a few stock market crashes later the file on *brokerage firms* becomes more related to files on *bankruptcies* rather than those on *banks*. The methodology for capturing such temporal changes in the relationships is described below.

A fresh filesystem will be represented as a forest (on a per user basis); with files inside each directory represented as a graph with edges having a unit weight. The forest is modified every time a file is accessed. The edges as well as edge weights are modified according to a set of rules exemplified below:

a. Edges are added when two files are accessed sequentially (edge is directed or undirected depends on whether the file is an output file or not).

b. If there exists an edge and if the files are accessed sequentially in any order again, weight is increased; otherwise weight is decreased. The weights may not change every time a group of files are accessed but will be changed by recurring access patterns.

c. The provenance information will modify the graph if

i. a file is an output of a program; edges will be added from all the input files.

ii. the ownership changes; then an edge gets added to the other relevant files belonging to the user.

iii. two files are created from the same process/program; then there will be an edge between them (or edge will be strengthened). This will also include cases where two files are downloaded from the same website and by the

same user (this information will be stored as part of provenance).

d. The ILE modifies the graphs every time it deduces an inference. E.g. Feedback from user activities will strengthen some relations and weaken some others; Similarly email replies will have the same effect.; Frequent use of text-to-speech will weaken the links to image files since the ILE will be able to deduce that this program is mostly used by users who prefer an audio interface over a visual interface.

e. The indexes created by annotations (keyword and user entered) will provide cues to the ILE to modify the relationships between nodes.

The rule set illustrated above is only a partial list. The rule c-(i) is adopted from [9].

### 3.3 Ranking Engine and User Interface

The Ranking Engine (RE) (part of the UAA) ranks the results based on a set of parameters that can be selected through the UI. The UI (Figure 2) is a parallel Interface for accessing files along with the explorer. It displays the results as well as

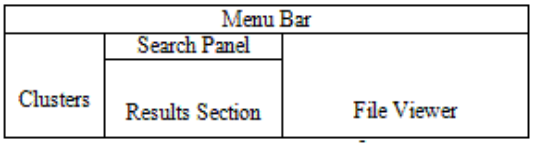


**Figure 2:** User Interface

gives feedback to the UAA. The UI consists of four sections (adapted from [16]), namely the Menu Bar, Search Panel (includes query entry as well as results), the File Viewer and the Clusters. File Viewer panel will be a scaled down version of KDE (K Desktop Environment) that can open any type of file. Clusters will display recently accessed files or a timeline or clusters based on known heuristics.

### 3.4 Database & Inode Modifications

A combination of Database and file inodes is used for storing metadata. Database support helps in delegating storage issues and faster retrieval. Inode storage leverages OS level caching, thus improving performance. The database (Figure 3) consists of tables storing Provenance, Inference, Versions, Modification Types, Files and Operations. The **Version** table holds all the modifications that are done on the files. *delta* is a composite field that stores the incremental changes of a file and its metadata.

The *uId* stores the user id of the user who is responsible for change. The *modId* points to an unique key in the **Modtype** table. The **Modtype** table is used for storing the different kind of modifications that can be done on a file – create, change content, move, rename, delete, append etc. The **Inference** table stores all the tuples might share the same *infId* if both of them are part of the same operation. The details of the changes are

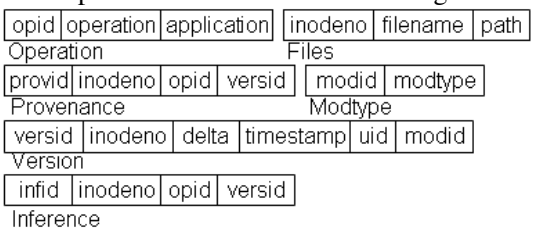


**Figure 3:** Tables to store the metadata

stored in the **Version** table. The **Operation** table stores all the unique operations (both atomic and composite) available in the system. The table **Provenance** holds the information pertaining to data provenance with the details of the changes in the file being stored in the **Version** table.

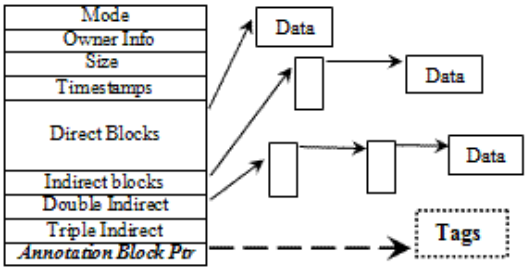


**Figure 4:** Modified Inode Structure

The Inode data structure (Figure 4) is a variant of the traditional data structure with an additional sequence of blocks that stores annotations associated with the file. The Annotation block consists of four parts: common, keyword, annotation and provenance. The common section stores the common information about all the annotations. Users can add any text in the annotation section. Keywords are extracted automatically from a file using standard Information Retrieval techniques (term frequency-inverse document frequency, stemming, parts of speech tagger, proper noun extraction) [1] and stored in the keyword section. The provenance section acts as a cache for the provenance information. This entry is a collection of ordered values that will be stored in the database. Throughout the duration of the file's usage, all the provenance data are noted down in this block. When the file is closed, this data is flushed into the database. In case if the file is not closed after a definite quantum of time the provenance block is automatically flushed. The common and the keyword sections are common to all the users

whereas the annotation and the provenance sections are divided on a per user basis.

### 3.5 Dataflow

The automated extraction of contextual information is done by two orthogonal processes: data provenance and inference collection from process provenance. The search workflow (after queried with a set of keywords) is divided into two parts: a content based search, the results of which are fed into a context enhancer engine. For the first part an existing search engine (Glimpse) is integrated into the kernel which indexes the files and then does a search on them. The results produced in the previous step are then fed into the Relations graph that produces more context rich results. This list acts as input to the RE which supplies the ranked results to the UI.

### 3.6 Versioning

Along with the versions in database, Checkpointing is used to store snapshots of the filesystem. Also beyond a particular checkpoint (which will be decided based on heuristics), data will be archived in compressed format to reduce storage area consumed.

## 4. Conclusion & Current Focus

Modifications to an existing filesystem (ext3) as well as the system call layer for incorporating annotations has already been implemented. The Annotation structure consists of *userid*, *groupid*, *annotation title* and *text*. System Calls for creating, reading, deleting and editing annotations and keywords have already been implemented. Experimental validation is also under progress. Along with the design parameters mentioned in this paper, security at the filesystem level needs to be addressed. Overhead in terms of both performance and storage in the system needs to be optimised. Trapping a process while it is running will generate more dynamic information for the ILE to learn from. Increasing the Confidence Measure of the ILE is very crucial for better search results. Based on the type of queries, a combination of Hash Table and several other data structures (e.g. graphs) may be considered as an alternative to a database for improved performance.

## 5. References


1. *Modern Information Retrieval*, Ricardo Baeza Yates & Berther Ribeiro Neto, Addison Wesley, 1999
2. Udi Manber, **GLIMPSE: A Tool to Search Through Entire File Systems,** Winter USENIX Technical Conference, 1994
3. Edward Cutrell, Daniel C. Robbins, Susan T. Dumais, and Raman Sarin. **Fast, flexible filtering with *Phlat*-personal search and organization made easy**, In *CHI 2006*, Montréal, Québec, Canada.
4. Susan T. Dumais, Edward Cutrell, J. J. Cadiz, Gavin Jancke, Raman Sarin, and Daniel C. Robbins. **Stuff I've Seen: A system for personal information retrieval and re-use**, In *SIGIR 2003*, Toronto, Ontario, Canada.
5. D. Quan, D. Huynh, and D. R. Karger. **Haystack: a platform for authoring end User semantic web applications,** International Semantic Web Conference, 2003
6. J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. **MyLifeBits: fulfilling the Memex vision,** ACM Multimedia, 2002.
7. S. Fertig, E. Freeman, and D. Gelernter. **Lifestreams: an alternative to the desktop metaphor**, ACM SIGCHI Conference, 1996.
8. CAN Soules, GR. Ganger, Carnegie Mellon University, **Connections: Using Context to Enhance File Search**, ACM SIGOPS Operating Systems Review, 2005
9. Sam Shah, Brian D. Noble, University of Michigan, Craig A. N. Soules, HP Labs, Gregory R. Ganger Carnegie Mellon University, **Using Provenance to Aid in Personal File Search**, USENIX '07 Annual Technical Conference, Santa Clara, CA
10. CAN Soules, GR Goodson, JD Strunk, GR Ganger, Carnegie Mellon University. **Metadata Efficiency in Versioning File Systems**. Proceedings of the 2nd USENIX Conference on File and Storage, 2003
11. Yogesh L. Simmhan, Beth Plale, Dennis Gannon, Computer Science Department, Indiana University, Bloomington. **A Survey of Data Provenance Techniques**. In SIGMOD RECORD, 2005
12. Kiran-Kumar Muniswamy-Reddy, David A. Holland, Uri Braun, andMargo Seltzer. **Provenace-aware storage systems**. In *USENIX 2006* Boston, MA, USA,
13. S Ames, N Bobb, KM Greenan, OS Hofmann, MW Storer. **LiFS: An attribute-rich file system for storage class memories**. 23rd IEEE/14th NASA Goddard Conference on Mass Storage, 2006