A Hierarchal Object Storage Environment:
Facilitating Easy Retrieval of Pertinent Data

I. Introduction

Since the advent of modern information storage apparati such as the UNIX file system and later world-wide networks of peer-based data sharing systems like HTTP and Gnutella, there has been a need to organize electronic data in a fashion that facilitates easy retrieval of that information upon demand. However, this goal has been often thwarted by several complicating factors—differing domains of knowledge between the creator and the seeker of information, as well as the creator's own inability to remember much about the data long after the information is stored.

Unfortunately, while many current operating systems' file management subsystems place significant emphasis upon the calculation of file metadata when the user stores information, these same systems generally do not offer much more than brute-force text searching when it comes time to locate information apropos to a user's query. Furthermore, the directory tree structure present in nearly all personal computer operating systems forces users to place files into a hierarchy that may very well be difficult for even the original user to comprehend. For instance, how often have you seen Windows users with 400 unrelated text documents in “C:\My Documents\” ? Searching such a collection would not be easy.

To build a system capable of finding documents pertinent to a query, it seems necessary to create a data storage system that is not only rich in metadata, but also possesses a systematic method for interpreting and categorizing the information contained within the data. Because most computer users have become accustomed to tree-based file system structure, it may very well be undesirable to remove the hierarchy concept. Furthermore, sufficient metadata-rich file systems exist, and hence the mechanics of actual file storage on a disk should be delegated to underlying file system layers. The focus of this proposed Hierarchal Object Storage Environment (HOSE) is to deal with the creation of layered content-aware plug-ins to assist in the query-time interpretation of data in order to retrieve relevant information to the information seeker.

II.a. Approach

The author felt that the following strategies would result in a workable system with the least amount of effort:

- Linux seems like a good choice for an initial implementation—due to the operating system's open nature, tweaking of the kernel's internal operations becomes quite easy. Furthermore, Linux's support for a large number of diverse filesystems (about fifty variations) handily increases the scope over which the HOSE can be used.
- This data retrieval system is intended to be situated between the physical filesystem driver and the computer user. As such, it seems to make the most sense to write a user-space program, in keeping with the Linux design strategy that the OS kernel is primarily for hardware drivers and program management—everything else should remain in user-space.
- With the HOSE thus positioned, Java, with its rich class libraries, plays a significant role in the development of the system. The large software base provided by the platform enables more time to be focused upon development of the actual storage environment, instead of recreating well-known data structures in C. Lastly, the continual integration of XML and web service support into the Java platform means that, in time, the HOSE could very well be extended to work with and as an Internet protocol.

II.b. Design

The first task in developing the HOSE environment was to model the actual storage mechanism behind HOSE. As stated previously, physical data storage tasks are to be left to existing Linux filesystem drivers, and hence it was necessary to create a model generic enough to encompass just about any file system, yet sufficiently extensible to adapt to particular features of a file system (hard linking, ACLs, POSIX extended attributes). To that end, a FileSystemFactory object was designed to create FileSystem objects that are tailored to fit the features of a Linux file system driver and emulate any features that do not exist. With this established, we have a standard method of persisting data objects.

With file system modelling finished, the focus now turns to data processing and interpretation at higher levels inside the system. The original circumspection of this part of the problem indicated the existence of three major categories of data object structures: graphs, tables, and sequences. However, a further inspection revealed that a sequence is a special case of a table, and a table could very well be represented as a lattice-shaped graph, although it may be convenient to preserve the table abstraction for performance reasons. At this point, a fundamental observation was made by the author—the process of conveying information to a computer is converting high-level information into successively lower levels of data—ideas flow from the brain into a structure of binary objects inside the computer, down through several more layers of augmentation and serialization, until finally they are no more than a sequence of bytes. Currently, Unixes and Windows perform file searches at this lowest layer; however, the HOSE system endeavors to provide functionality at many more layers. To do that, above the file system sits a vast collection of data object interpreter modules; each of these plug-ins has the ability, given a graph of input data, to examine the data contained within the graph, and emit a procesed version of the data. The large quantity of plugins, then, enables the user to examine any data at any level with an arbitrary focus. Obviously, each layer should have the ability to attach arbitrary attribute metadata to any graph or part of a graph. Walking upwards from raw byte streams, the HOSE would likely have a lexical interpreter to tokenize raw file data, then atop that a logical interpreter to turn those tokens into a form that might resemble a graph of human-readable data. Continuing upwards, one might find a variety of organizational plugins that link together related or dependent material. Although higher levels allow a broader but shallower presentation of the underlying data, the fact that each plugin depends upon the output of the layer below it implies that a drill-down for detailed data is still quite possible.

An example of HOSE in use might clear things up a bit—imagine a regular file. At the bottom level, it is a collection of bytes. Going up in the path towards specificity, it would be recognized by the environment as, say, an XML document, then later as a tree that represents a chapter of a report. Further up, a “report” organizer plugin might contain links to this chapter and all of its siblings, and continuing beyond that, a “Q1 Status Report” organizer would tie together all fiscal reports enterprise-wide. In this way, we have established an association hierarchy where the data store enforces a structure that encourages only the association of data to relevant data—no more directories of 400 unrelated files! Morever, the richness of the system's ability to interpret data on any level at any time means that a user can query a HOSE, have it examine relevant parts of a data objects, and return (hopefully) more useful results than a simple grep. The author adds that should the interpretation proceed high enough in the hierarchy, on-the-fly document format translation becomes a simple matter of tracking down the appropriate export plug-ins.

III. Status and Future Directions

The planning phase of the project mainly involved talking to friends, roommates, relatives, fellow Linux administrators, and the patrons of the Roma coffee shop on the UCSD campus, in order to gauge the efficacy of the data organization schemes that they had developed for their computerized documents. Very few respondents indicated the presence of a system for organizing data, and so the author proposed to them the above scheme for computer-assisted data retrieval, and the responses were largely enthusiastic. Following that, the author endeavored to sketch out the basic components of the system and how they might be integrated into a Linux machine. This was miraculously finished despite a barrage of school projects and final exams.

On a practical level, an actual implementation of this system would be a good direction for this project; prototyping with a Java client should begin soon. In terms of theory, it seems that once a successful data object retrieval has been created, the notion of a strict hierarchy becomes unnecessary. Instead of a rooted tree, imagine that the information atoms contained within the HOSE are all inter-linked in a graph. Then, searching the HOSE for all information apropos to a query is as simple as finding a handful of extremely relevant nodes and then filtering through surrounding nodes. A couple of benefits come to mind:

- A search on “WebSphere” could return the jar representing WebSphere itself, as well as references to anything else that relates to WebSphere—JVMs, help documents, online IBM tutorials, etc. This helps system administrators audit which software packages are installed on the system, and exactly why those packages need to be present.
- Searching for “Q4 2001 Financial Reports” would give you not only the actual finance report itself, but also links to the weekly operations reports that went into the compilation of that report. Such a drill-down ability could be useful to investors who are researching a company before investing in its stock.

See http://thibs.menloschool.org/~djwong/programs/hose/ for more information.