Decentralized Enterprise Search
Charles Cary

Problem

The problem of decentralized enterprise search (DES) should be broken down into two technical sub- problems.

1) The storage and retrieval of an enterprise's information.
2) The searching through and indexing of this information.

Furthermore, these problems should be solved with respect to an important constraint: rather than attempt to build DES systems that scale to an arbitrary number of nodes, this project will focus on solving the problem of DES for a small number of nodes, probably about 100 or less. This is a reasonable constraint because decentralization is not free. First, search and indexing in less costly if it can easily be done in a centralized manner because, with little information about what data each node stores, decentralized search can be inefficient. Second, when data is decentralized, control of it is also decentralized. Systems exist for the distributed access control and encryption of data. However, no system is perfect and beyond this, in a decentralized system that consists of machines that are not a in a controlled area, as in a datacenter, the machines themselves can be physically stolen. Therefore, as organizations grow and acquire the capital necessary to implement centralized storage and search, it makes sense for these organizations to choose centralization over decentralization because centralization becomes both less costly and more secure. The prevalence of the term 'cloud' in contemporary information technology literature is a testament to this; 'cloud' is in many ways about the centralization of data and the centralization of control in a data center where dummy edge node interacts with the services provided by the 'cloud', but do not offer services themselves.

Storage and Retrieval

Peer-to-peer systems offer two advantages that make them a good fit for solving the problem of enterprise search for small businesses. First, peer-to-peer systems can be built that are totally decentralized. This means that a peer-to-peer system requires little or no extra hardware costs from the outset. Second, peer-to-peer systems can be built in a self-organizing way. This reduces or eliminates the expertise necessary to deploy enterprise search systems because the configuration and maintenance of the systems rests on the system itself.

The foundation of DES is a peer-to-peer overlay network that can route requested files to the nodes that the files reside on. Our peer-to-peer system consists of a distributed hash table (DHT) that routes keys associated with a file to the nodes in which the files reside. It is to be implemented as an overlay network on top of any existing networks that a business has deployed. There exist four relatively simple, general-purpose, DHT algorithms each of which could be candidates for our system [1]. Two of these, Chord and Pastry, were evaluated in a more in depth manner because peer-to-peer file storage and retrieval mechanisms exist that have been implemented using these algorithms [1, 4, 5]. Of these two, Pastry, is a good fit for the current system. PAST and Kosha, two different systems for the storage and retrieval of

files in a peer-to-peer manner, have been implemented using Pastry. Furthermore, Pastry offers an effective way for bringing nodes into and system and uses a simple heartbeat mechanism to detect node failures. This is valuable as nodes are likely to be entering and exiting the cluster throughout the business day.

Unlike Kosha, which is a peer-to-peer file system implemented using Pastry that exposes a familiar tree-structured file system to its users, our system will follow Pastry more closely and store files in a flat-file system, simply mapping keys to files [2]. This simplifies our implementation, but introduces extra importance on the text search capabilities of our system because it becomes the only means by which users can find files. Later on, as the project progresses, other mechanisms for file retrieval such as creation time and user may be introduced into our software. However, each of these is a problem requiring thought out solutions when working in a decentralized system.

Pastry will be used not only for its ability to distribute files, but also its ability to easily find replicas of a file when a node exits the network. PAST and Kosha replicate files not only on the node that a file's key hashes to, but on the nodes that are most close to the node in the key-space [2,3]. Pastry allows one to easily route keys to their corresponding nodes. When one of these nodes fails or simply exits the network, Pastry allows for finding the k nodes closest to the missing node. This allows for files to be retrieved even though their main storage node may have gone down. In this way, Pastry will aid in the replicated storage of files leading to fault-tolerance.

The Pastry algorithm itself will be slightly simplified in order to increase the speed of development. Namely, the neighborhood set of the routing table will be replaced with a known-nodes table that stores the IP addresses of nodes known to be part of the search system. As this system is to be implemented in small businesses, multi-site search will not be supported for now. Nodes will be required to reside in the same place and on the same network. This greatly reduces the need to use proximity as a metric to make routing decisions because we have assumed that all nodes are close to each other.

In order to initially define the known-nodes table, we will use a simple, physical system relying on humans and USB sticks. When initially setting up our search system, a user will obtain a USB stick with a copy of our software on it. The stick will be inserted into the first node and software installed. The IP address of this first node will be stored on the stick. Now, for each other node, the same stick will be inserted and same process repeated, storing each node's IP addresses on the stick. However, each time, the node will populate its known-nodes table with all the IP's stored on the stick and then message all the nodes at the known IP's so that they too update their known-nodes table. This process will inform each node of the existence of many of the other nodes. When trying to reestablish connection to a network, nodes simply use the IP's in their known-nodes table in order to find a boot node to join the network. Furthermore, periodic heartbeats throughout the system could alert nodes of IP changes and other new nodes, which were added after the initial set up.

Searching and Indexing

Balakrishnan considers the problem of indexing a decentralized file store an open question [1]. He writes, "While it is expected that distributed indexing and keyword lookup

can be layered on top of the distributed hash model, it is an open question if indexing can be done efficiently". At scale, this may still be an open question. However, in limiting the scope of our project to small networks, the problem becomes far easier to solve. With a small number of nodes, flooding a network with a search request is not a terrible approach. It is simple to implement. Furthermore, it may be the only way to ensure all files are searched when using a DHT algorithm.

The robustness of DHT algorithms comes from their distribution of files in a random way over the nodes, yet, this same randomness poses a huge challenge to indexing. Most indexing schemes rely on the orderly distribution of files to be indexed. Files and indices are often stored across a finite, known, and unchanging number of machines. The machines behave as shards of greater system, indexing files that hash to the same value in a centralized way. When building a file system on top of unknown and changing number of nodes using a DHT algorithm, the standard indexing approach ceases to work. It is unlikely that a subset of the nodes can be guaranteed to contain all the files as nodes join and exit the system.

Instead of constructing an elaborate system to determine subsets of nodes that form a complete image of the stored files, it is better to focus on how to combine duplicate search results from all the nodes after the network has been flooded with a search request. When obtaining different scores for one document in a search result, determining the correct score may be a challenge. Multiple score comparison systems will be devised, implemented, and tested in order to determine which produces the best search results. Simple things like mean, mode, and sum will be considered. More elaborate systems may be devised later.

As an organization grows, flooding will cease to be a viable system. At this stage, maybe a hybrid network structure is necessary. Supernodes could be introduced that provide search and indexing services for the entire system. The nodes themselves could continue to store the files, but the search would be centralized. Implementing this system is beyond the scope of this project; I will attempt it afterwards.

Other Considerations

File updating remains an issue for our system. Many peer-to-peer file systems exist which are write-only; in order to avoid locking, systems do not allow for updating files, only writing new ones. PAST is an example of this approach. It does not support delete or update operations and outlines a different reclamation method for reclaiming storage space [3]. Kosha is a distributed version of NFS. For now, our system will remain write-only. However, this is not necessarily where it will remain; write only is unlikely to capture the many use cases of files in a business [2].

One other major consideration is the issue of security. Our initial version will store files without encryption. This is not acceptable for a production system. Furthermore, network traffic needs to be encrypted and systems developed to verify the authenticity of nodes.

Implementation Details and Logistics

Each node will have a file store directory, cache directory, and running background

process. The file store directory is where the files themselves will reside. The cache directory will include files that the user has most recently requested. Furthermore, the system will be built in such a way that users always copy a file to their cache for modification then add the modified file to the clustered storage once the changes are to be committed. Files are never edited directly. Only modified copies are allowed. The background process consists of a Pastry implementation to route requests along with file processing, indexing, and search systems. Files will be processed using Apache Tika to extract text and indexing and search will be provided using Apache Lucene. This process will run in the background of the user's machine.

The user will not interact with the background process directly; instead, one or more client programs will interact with the background on behalf of the user. A simple search application that takes text input and returns results needs to be implemented. Later on, automatic saving and opening add-ons/plugins for popular software such as Microsoft Office and Open Office need to be developed.

As for the implementation of our system itself, Java will most likely be used as the main language. One goal of the project is operating system independence, a goal which makes Java a good choice. Furthermore, as our search systems are built on top of software that is written in java (Lucene and Tika), selecting the same language will simplify development. The front end systems will be built using python and a wrapper on Nokia's QT project. The development language of the add-ons/plugins will vary based on the target software.

Sources - Cited

[1] Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I. Looking Up Data in P2P Systems. In *Communications of the ACM*, Vol. 46, No. 2, February 2003, pp. 43-48

[2] Butt, A., Johnson, T., Zheng, W., Hu, Y., Kosha: A Peer-to-Peer Enhancement for the Network File System. In *Proceedings of the ACM/IEEE SC2004: High Performance Computing, Networking and Storage Conference* (Pittsburgh, Nov 2004).

[3] P. Druschel and A. Rowstron, "PAST: A large-scale, persistent peer-to-peer storage utility", HotOS VIII, Schoss Elmau, Germany, May 2001.

[4] Rowstron, A., and Druschell, P. Pastry: Scalable, distributed object location and routing for large-s cale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms* (Middleware 2001) (Nov. 2001).

[5] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., and Balakrishnan, H., Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proceedings of ACM SIGCOMM* (San Diego, Aug. 2001).

Sources – Consulted

Apache Tika. Apache Software Foundation. 2012. <http://tika.apache.org/>

Cohen, E. and Shenker, S., Replication Strategies in Unstructured Peer-to-Peer Networks. In

ACM SIGCOMM'02 (Pittsburgh, Aug. 2002).

Lucene. Apache Software Foundation. 2012. <http://lucene.apache.org/core/>

Pastry: A Substrate for Peer-to-Peer Application. Ant Rowstron. 2012. <http://research.microsoft.com/en-us/um/people/antr/pastry/>

Tanenbaum, Andrew and Maarten van Steen. *Distributed Systems: Principles and Paradigms*. 2nd Ed. Upper Saddle River: Pearson, 2007. Print.