Large-Scale P2P Document Management

Ian Pye | UC Santa Cruz
Storage Systems Research Center

Abstract:

One of the major challenges facing a globally distributed Enterprise is storing and retrieving documents. This task requires features from distributed storage systems, databases and the Internet. However, because of the unique nature of the documents to be stored and the access pattern of the documents, none of these alone suffice to provide truly reliable and searchable distributed document management. We present Ringer – a overlay network which combines elements of peer-to-peer storage, distributed query processing and indexing to create an Internet scale content distribution and retrieval network.

Distributed Document Management

We Need:

- Filesystem Semantics

  Concurrency control and close-to-open consistency are essential to keep things manageable in a distributed system with many readers and writers.

- Database-Style Indexing

  We need the ability to search on arbitrary file attributes and get good results quickly. The rich interconnectedness of hypertext, which enables algorithms like PageRank to work, does not exist for other file types. Information is stored in only a few documents, not 100s!

- Internet-Style Connectivity

  Files need to be available on demand, but only downloaded (an expensive operation on a WAN) as needed. Most files are accessed rarely.

→ No Single System Does All This Yet! ←

We Propose:

- A Hierarchical Peer-to-Peer System
- Tier 1: Desktop clients store and transmit files
- Tier 2: Lightweight MetaData Servers (MDS) index files, manage security and permissions, store metadata
- Tier 3: Central Authority (currently Amazon.com's Simple Storage Service) maps file IDs to metadata servers

Case Study

The United Nations has offices in 192 countries. All of these offices are generating reports. These reports need to be distributed to all of the other offices, as well as being safely archived. However, reports are not all equally useful, and the value of a report changes over time. In addition, the UN does not have a large Information Technobgy budget. With these rules, the problem becomes one of distributed document management: content can be added by any office, content must be findable by any other office and the system must be reliable. Furthermore, the system must be inexpensive yet scalable to support large numbers of documents. It must also function effectively with offices scattered across the globe, connected in a network of varying bandwidth and reliability.

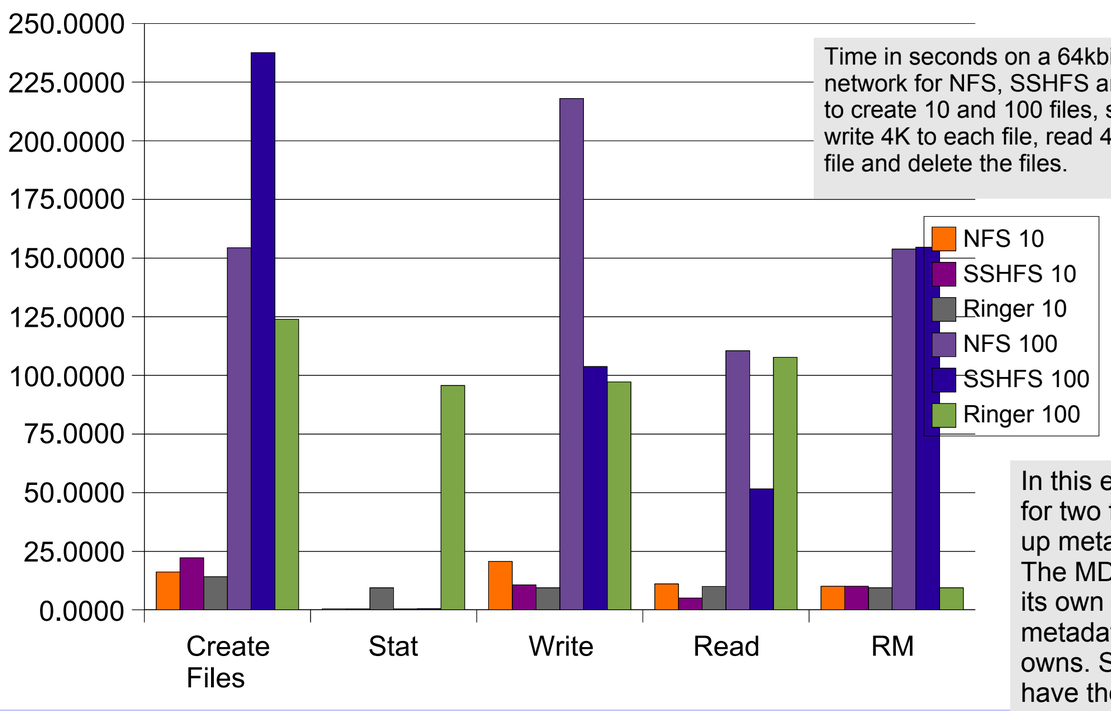
Implementation

Ringer is essentially a three tiered database-driven application with a custom front-end. It consists of a collection of clients, metadata servers (acting as middleware) and a central backing server which ties everything together. XML-RPC is used as the interconnect language. This protocol entails a performance hit, but also facilitates the creation of third-party plugins, and provides access to a large pool of utilities. The client binds with its host operating system via the FUSE (Filesystem in USerspacE) library. FUSE allows Ringer to be mounted and unmounted like any other volume. Metadata servers are simply user processes, and can be started and stopped at will.

Future Work

- **Search**: Files added to the system need to be indexed and made easily retrievable. We are looking at ways to do this involving semantic clustering and routing, probabilistic search and a hierarchy of indices.
- **Security**: There needs to be a system of permissions and access controls, in addition to encryption of data on the wire.
- **Performance**: We are looking to exploit parallelism and asynchronous protocols to improve overall performance.

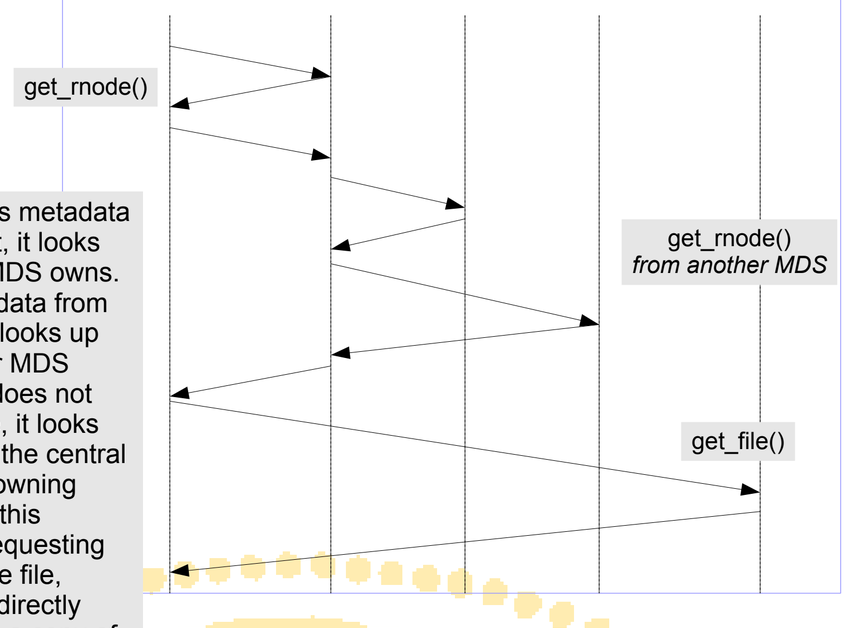
Comparative Microbenchmarks



Time in seconds on a 64kbit/sec network for NFS, SSHFS and Ringer to create 10 and 100 files, stat the files, write 4K to each file, read 4K from each file and delete the files.

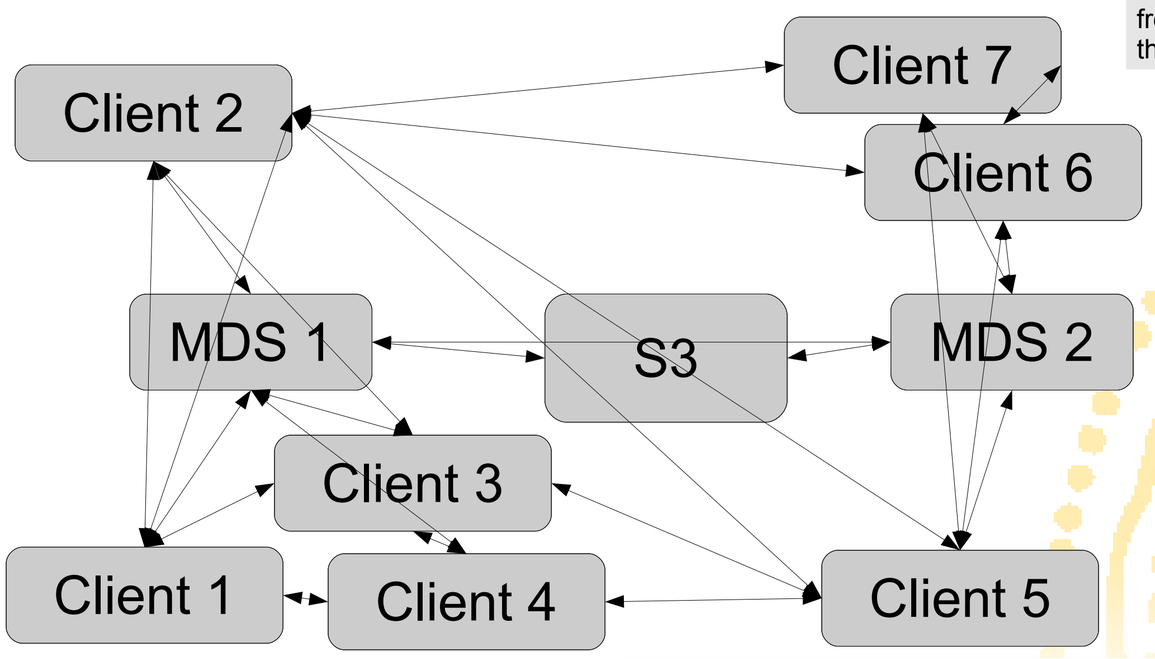
Retrieval Protocol

Client 1 | MDS 1 | S3 | MDS 2 | Client 2



In this example, a client fetches metadata for two files from its MDS. First, it looks up metadata for a file that its MDS owns. The MDS is able to return the data from its own cache. Next, the client looks up metadata for a file that another MDS owns. Since the client's MDS does not have the metadata in its cache, it looks up the requested file's MDS in the central authority. Then, it queries the owning MDS for the metadata. Lastly, this metadata is sent back to the requesting client. The client now opens the file, which involves downloading it directly from another client possessing a copy of the file.

Architecture



Contact

Ian Pye | ipye@cs.ucsc.edu

Scott Brandt | sbrandt@cs.ucsc.edu

Carlos Maltzahn | carlosm@cs.ucsc.edu

http://code.google.com/p/metaring