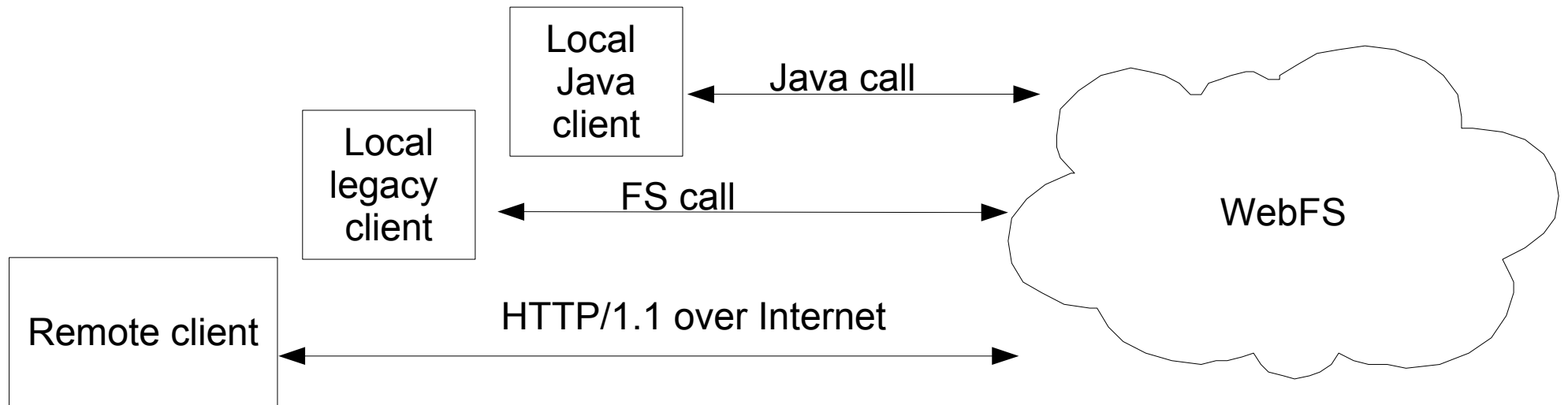




Nicolas Toper's Thesis

General Overview of WebFS
ntoper@gmail.com

What is it? (1)



- WebFS is a Web friendly distributed file system
- WebFS is a repository
- WebFS is redundant: no dataset is ever lost
- WebFS scales out to 100 million objects

What is it? (2)

- A document based distributed file system
- Store object and file in a simple, consistent, secure and redundant way
- Good performance (in average as good as a local FS)
- Low learning curve: HTTP or In/OutputStream
- Type system based on XML Schema
- Configurable search engine
- Heterogeneity of access: from a local Java app to a remote Web service
- Based on HTTP and Web technologies: WAN and LAN « aware »
- Ease of administration

Why build it?

- More data storage need than ever (long tail, personal backup, video web site, mp3 sites,...)
- A lot of framework to ease distribution of applications (both academic and pro), nothing for storage
- Storage and continuous plan are too costly for a whole new class of application to emerge
- Reliability for web services is now a must
- Not that difficult to build so should be open sourced
- I need to write a master's thesis so allow me to write it on a subject useful for everyone

Similar technologies

- Some academic work (Coda for instance)
- Some pro OSS work (mobileFS) but either amateur or used for clustering
- Amazon S3 but SaaS and too minimal to be really useful
- A lot of expensive commercial applications or hardware
- OceanStore for full P2P (Berkeley product) but too academic for now
- Hadoop (from Apache) but based on a cluster of machine and designed for big files: no easy WAN support
- Internal applications (YouTube, DailyMotion)

Goals of my Master's Thesis

- Engineer's way of working (conceptualization)
- Coherent body of work
- **Scientific publication potential**
- 9 to 12 months of work (thesis included)
- Hard and challenging work
- **90 to 150 pages of scientific writing**
- **Build a state of the art or upper product**

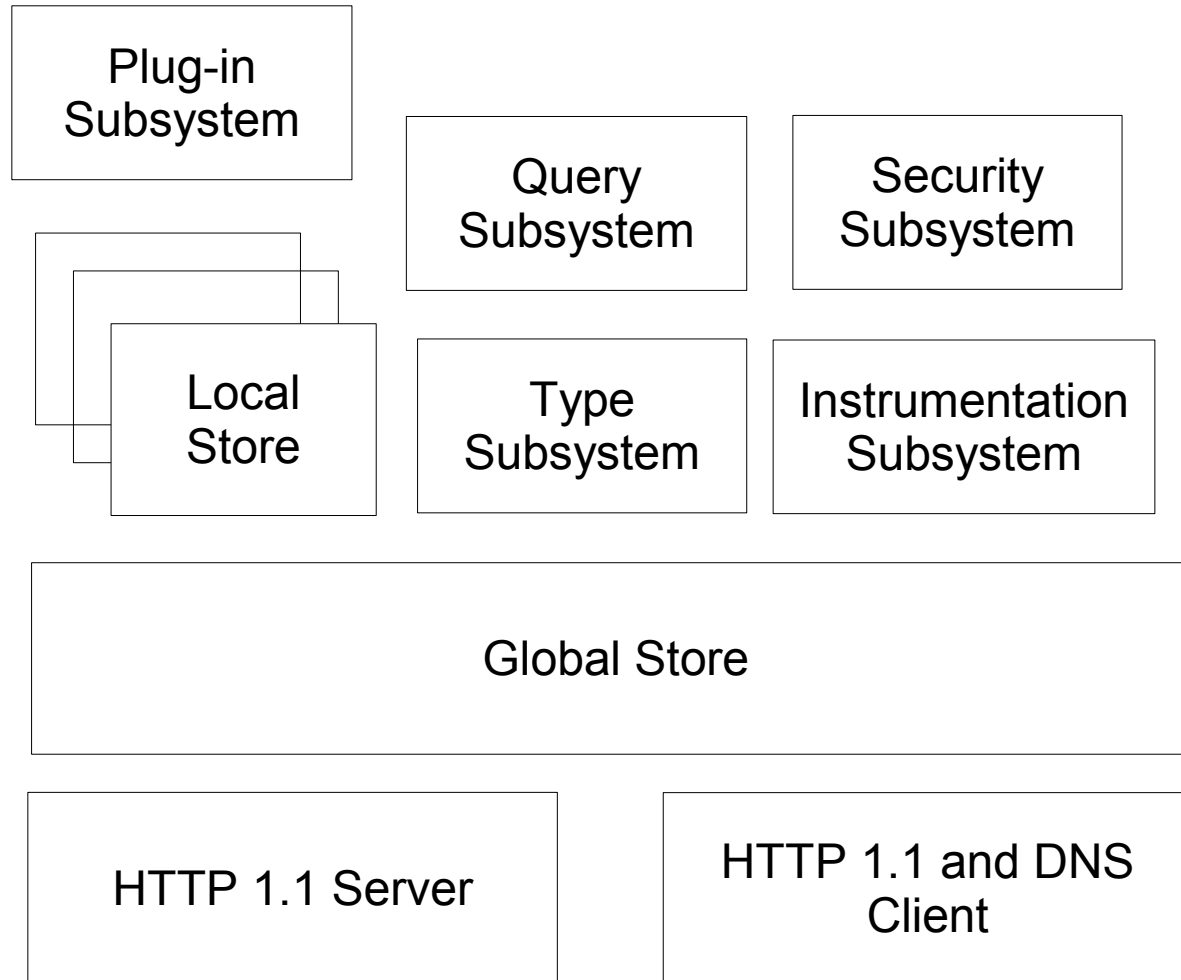
Main Functionalities

- **Content Repository**
 - Sophisticated type system
 - File system paradigm (directories, symlinks)
 - OS interface (through FuseJ)
 - **Scale out** to 100 millions object per workspace
- **Fault tolerance and fail over**
- **Powerful search**: full text, SQL, Xpath
- **ACID properties** with read committed isolation

Properties of CNT 2.0

- Up to 100 millions objects per instance
- Graceful failover of host(s): « P2P » architecture, no single point of failure
- Unpredictable network topology
- Easy add/removal of a machine
- More read than write but concurrent read
- Fast read
- Read committed write
- Write can be slow
- Excellent data integrity through checksum and redundancy of data
- Each object should be replicated twice at an instant t .

Architecture



Main Design Decisions

- **Java based instead of C/C++** easier to set up and easier to build a community around. (Code is simple so it can be ported to C/C++ when needed).
- **Totally open source project** since we need to gain traction and open sourcing it helps (and I gain kudos if the project works fine).
- **DSM principle** to « embed » fault tolerance and scaling in the project design. (DSM is a little bit like P2P design.)
- Fit in Web paradigm
- Use of web standard when possible.
- Use of open source components.

HTTP/1.1 Server And Client

- Web server and client are used to communicate from one host to another
 - WAN and LAN friendly
 - Easy to troubleshoot/secure
 - Performance overhead is small
- Web server is also used to communicate between WebFS and remote clients
 - « Raw » Web server and client component (implementing maybe Servlet API)
 - High concurrent throughput
 - High performance

DNS Client

- Each host has a unique CNAME and is part of an alias.
- CNAME is used to name internally all WebFS machines
- CNAME is also used to differentiate a request between a trusted and untrusted machines (+ login/password of course)
- Alias DNS is used to distribute load evenly between all machines (Round Robin DNS)
- Easy to add/delete a host through this mechanism.

Stores

- Local Stores are responsible to store objects on a specific host.
 - Set up and manage the virtual FS (localisation, naming, size issue)
 - Local transaction management (durability and atomicity)
 - Delegate a lot to the local FS.
- Global Stores are virtualising all local stores in a coherent abstraction
 - Provide a unique naming space
 - Handle concurrency issues
 - Handle cache/replica data (replicate or redirect an internal request; invalidate replicas)
 - Distributed transaction management (isolation, consistency)
 - Update all indexes (localisation table is a special index).

Type subsystem

- Objects are either structured, unstructured or semi-structured
- Some elements in an object could be indexed (searchable) or lockable (instead of locking the whole document)
- Objects can be distributed in two hyperchunk (hyperchunk is the localisation system granularity level)
- Some metadata are at the beginning of the file (ACL, type,...)
- An optional XML schema defining the type and what elements are searchable and/or lockable

Query Subsystem

- Localisation and query/search are the same problem (we need to locate quickly some data)
- Query subsystem allows to dynamically bind an index to a query language
- Distributed indexes: all indexes are partly on all machines
- DSM based
 - No index is totally accurate at an instant t
 - But the subsystem knows when the data is false and go look for it elsewhere
- Localisation is based on hyperchunk (approx. a directory leaf). Xpath search is built on top of the localisation index

Instrumentation Subsystem

- Used by all global components, mainly by the global store
- Create a reflexive layer to update the system dynamically to external events (think thermostat design)
- V1 only minimal set of features but a lot of performance/reliability gain in V2 here
- Crawl WebFS to look for inconsistencies/optimization
- Is called for specific events (or aspect based)
- Ie: Cache pattern detection, knowing whether to replicate an hyperchunk or redirect a request, replicate for « availability » (on a different data centers)

Security Subsystem

- Each object has an associated ACL
- A public user exist: allow anonymous access (ie to turn WebFS into a distributed web servers)
- Same user management system as *Nix's one
- Differentiate WebFS host from external clients
- A system session for each host but authorization might be different according to configuration
- V1: support for 1 000 users account >> we will store all users as an index
- Use of HTTP or HTTPS
- (No session to ease distribution so credentials are passed in each request)

Plug-in Subsystem

- Needs are by essence unpredictable
- The plug in subsystem is based around an event based framework
- Allow to extend WebFS to custom it for specific needs
- Turn this software into a platform where a community can gather around
- Not scheduled for V1 but « hook points » will be already there.

Next Steps

- Define Web API (WebDAV based?)
- Find support in and out of the company
- Detailed requirements and architecture
- State of the art document
- Planning with milestones
- Find a place to host the project
- Open the project now (ML, Wiki, issue tracker,...)