Queriable File Systems for Metadata Management

Sasha Ames
University of California, Santa Cruz

Maya B. Gokhale
Lawrence Livermore National Laboratory

Carlos Maltzahn
University of California, Santa Cruz

Ethan L. Miller
University of California, Santa Cruz

Abstract:

Problem:

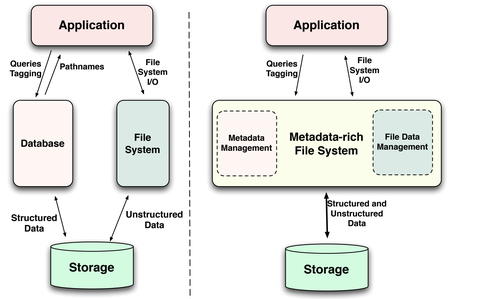
- Approaches to access and manage metadata has changed little
- Dichotomy: (1) files store raw data, (2) databases store metadata
- This hybrid approach is inflexible and slow

Quasar File System (QFS):

- First class objects: *files*, *file attributes*, *relational links* between files
- Reduces complexity and improves performance by orders of magnitude
- Evaluation based on Livermore Entity Extraction

Introduction:

- Traditional Architecture - file systems store data, relational databases store metadata
- Separation reduces performance and consistency:
  1. **Expensive 2-step query evaluation**: first database query, second file name resolution
  2. **Brittle metadata/data association**: costly maintenance of path names in database while moving data
  3. **Global scope only**: as opposed to scope of file directories (often aligned to semantics and locality)
- **Metadata-rich file system**: conv. file system I/O services + metadata storage and query infrastructure

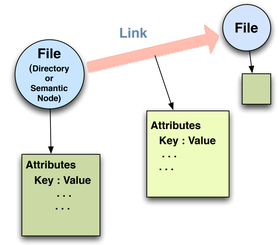


Our Approach:

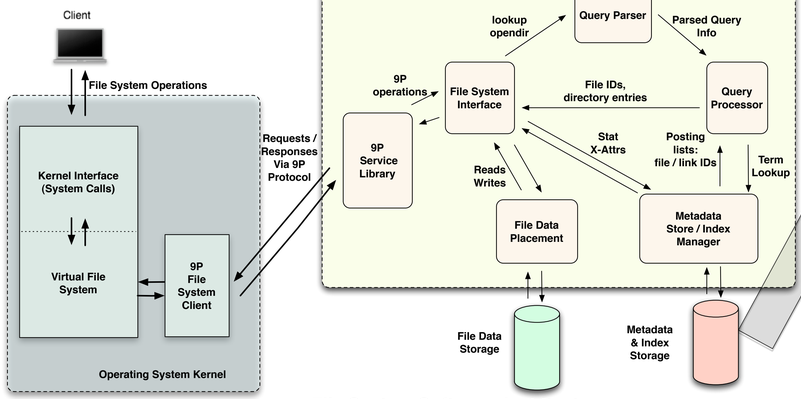
We propose extending file systems to manage interfile relationships using links and provide a common interface for querying utilizing conventional file metadata, extended attributes and our new links. As with files, our links contain key-value pair attributes. We use links to implement POSIX directories, in which zero-byte files link to other files. We have developed the QUASAR query/naming language specifically for posing queries to our file system data model, based in part on XPATH.

Our prototype file system implementation uses the 9P protocol for transport. Pathnames are expressed in QUASAR and are processed by the file system's "lookup" pathway. In the prototype, file data is stored in corresponding "backing files" on the local disk. Metadata is organized in data structures suited for byte-addressable memories. Indices are incrementally updated when new attributes are written.

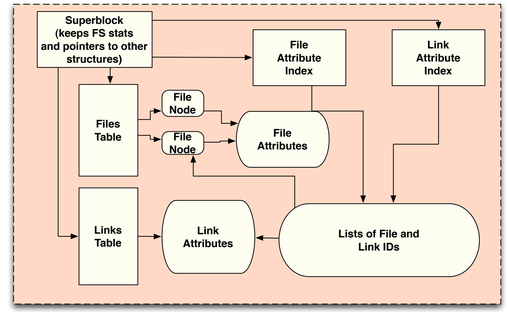
The prototype system is operational and we now have directed our efforts to system performance optimization. Our optimizations either implemented or in progress include: simple caching of query results; batch transfer of multiple metadata "write-oriented" operations; improved query planning/optimization.



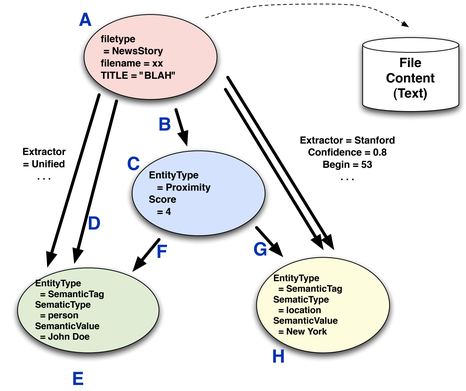
Data Model for File System Metadata



File System Software Architecture



Metadata Storage Data Structure Layout



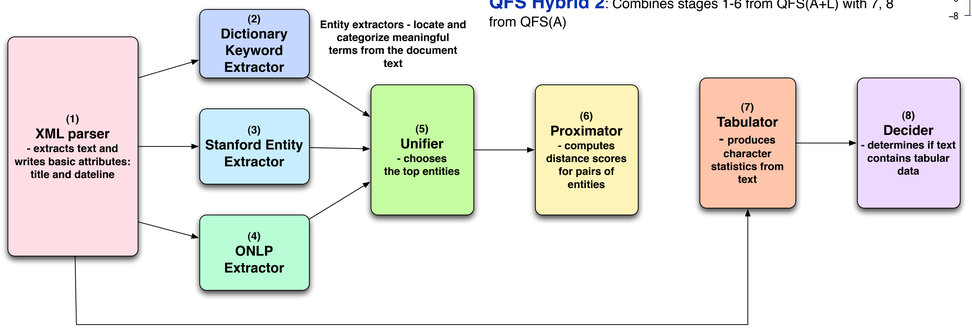
Metadata Ontology

Case Study:

Livermore Entity Extraction (LexTrac)

- Purpose to automate analysis of text documents
- Each entity found by extractors contains its own metadata
- We show the "Ingest" phase. Once all tags have been generated we follow with a "Query" phase.

Original pipeline implementation (FS+DB): writes metadata at each pipeline stage to flat files, and subsequent stages read and parse the files. The drawback to this approach is that the metadata is not searchable and must be written to a database for that purpose.



New pipeline implementation (QFS):

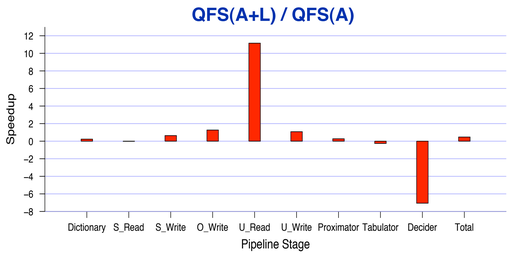
The application uses our queriable file system metadata storage capabilities and queries for required metadata at each stage. To retrieve previously written metadata, the application uses the file system query interface. Our versions:

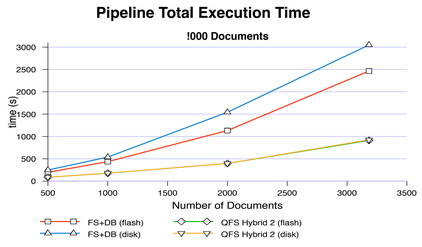
QFS(A): Only utilizes extended attributes - to make the metadata useful between pipeline stages, we had to devise a naming scheme for groups of attributes, as the metadata for each entity.

QFS(A+L): Utilizes extended attributes, links and zero-byte files as semantic nodes for a rich semantic model

QFS Hybrid 2: Combines stages 1-6 from QFS(A+L) with 7, 8 from QFS(A)

Results:





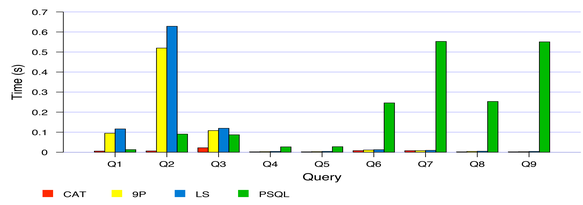
- Database system used: PostgreSQL.

Pipeline stage execution times of queriable file system with optimizations
Pipeline version: QFS(A+L) over 1000 documents (times in ms)

RUN	XML PARSE	DICTIONARY READ	DICTIONARY WRITE	STANFORD READ/PROC	STANFORD WRITE	ONLP READ	ONLP PROC
No Optimizations	1458	1219	19281	27848	18115	263	2044
Caching	1314	1253	18084	27723	17095	264	2072
Batching	1459	1403	2633	28539	2747	264	2190
Caching + Batching	1330	1395	2580	28338	2850	249	2013
	ONLP PROC 2	OLNP WRITE	UNIFY READ	UNIFY WRITE	PROXIMATOR	TABULATOR	DECIDER
No Optimizations	58544	13085	5685	31888	380484	20658	129481
Caching	58370	11757	5324	29611	329850	20560	43840
Batching	59070	2381	5928	3520	38468	2647	24873
Caching + Batching	57962	2331	5709	3527	38372	2456	9254

Query	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
Joins	1	0	0	2	2	3	5	3	5
Results	1500	10000	1000	4	4	10	2	5	2
Elements	A	C	C	A D E	A D E	ABCEF	ABCEFGH	ABCEF	ABCEFGH
Big Table?	N	Y	Y	N	N	Y	Y	Y	Y

Individual Query Performance



This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-407377