Some Notes on Logical File Names and Related Interfaces

David Malon

malon@anl.gov

ATLAS Database Group

LHC Persistence Workshop

5 June 2002

Disclaimers

- Outside the context of the common project, we in ATLAS had been talking about how to wrap testbed-dependent LFN/PFN mapping services behind a common Athena service interface
- I mentioned this to Dirk—mistake!!—now I’m giving a talk
- Consider this “thinking aloud”…

Grid replica services

- The nascent state of grid replica services has led to a somewhat schizophrenic state
  - On the one hand, services have been predicated on the assumption that LFNs are unique with a virtual organization
  - On the other hand, ensuring uniqueness has been left (without tools) to all the world's VOs
- These services are supposed to be the basis for a global file system, but
  - On the one hand, some services or use models require (at least implicitly) that LFNs be immutable;
  - On the other hand, you would never accept a file system in which you had to name everything perfectly the first time—no renaming, no moving to another directory, …
- Good news, though—things are improving
  - See EDG talk

Common project discussions

- Some discussion in Architects Forum about how/whether to write LFNs inside files created by storage manager/streaming layer
  - Pros/cons of aliases, …
  - Vincenzo said it is a requirement that users be allowed to change LFNs
  - I will try to avoid rehashing these discussions
- Do we really need to agree on an approach to LFN determination at file creation time? I wonder….

When to assign names

(At least) three natural(?) times to consider assignment of logical filenames:

- Job submission time
  - We are accustomed to managing (non-grid) productions, with naming conventions and run number assignment and random number seeding all orchestrated to meet uniqueness constraints—why should LFN management be different?
- File creation time
  - This is when local file systems succeed or fail at name assignment
- File registration time

A confession

I have qualms about both

- "Trust me! I am always careful with job configuration. I know this is a safe name choice."

and

- "I cannot return from my create() method until our Uniqueness Assurance Center confirms that this name is okay."

Assumptions

- Jobs using our persistence infrastructure will run “on the grid” and off, interactively or in batch, under control of a job submission service or not.
- Persistence infrastructure will support production and personal use.
- Output may or may not be published to collaboration catalogs and/or to the grid.

I tend to think of this as (at least) a two-stage operation—produce the data, then publish it—more correctly, decide whether to publish it, and under what name(s).

It seems architecturally natural to me to defer final selection of LFNs until the publication stage if we can.

Of course, in production jobs, a single script that runs the job, then publishes output, may in fact have been preconfigured with output LFN choices.

One possible approach

- When file is created, a permanent “globally” unique id is assigned, and written into the file—call this the GUID
  - Many schemes have been proposed (UUID-based, …)—pick one
  - May not care whether “global” is truly global, or within a VO, or …
  - No LFN intended for human interpretation is assigned at this point
- Storage manager writes a {GUID, localName} record into a local file
  - Human-browsable, even editable

One approach…

- At end of job, this helper file contains a list of {GUID, localName} entries—one for each file created
- Ref implementations should rely on GUID, not on logical file name
- If user runs purely locally—no relational layer, no grid—helper file suffices to support interfile navigation
- If/when job output is published—to the grid, cataloged in common project relational layer, …--a logical file name is assigned, and also associated with the GUID
  - In some schemes, the GUID may be an alternative LFN or alias
  - Emerging Replica Location Services support replica metadata—GUID could, alternatively, be metadata associated with the LFN

Extraction for standalone processing

- When a user wishes to copy an event collection or dataset or … for off-net processing, e.g., on her laptop, the (automated) process is something like
  - Translate dataset request into list of LFNs and corresponding list of GUIDs
  - Find nearby replica, copy it to laptop with some localName
  - Write one more file: the {GUID, localName} list
- User can now run disconnected, and navigation works

Interfaces

- Should be easy, right?
  - getLocalName(in string LFN, out string localName)

  or

  - getLocalName(in GUID fileid, out string localName)
- What do you do if the LFN is invalid, if no such LFN is registered, if replicas exist but are not local, …?
- Architects Forum needs to decide how to handle errors and nonstandard situations (Return codes? Status codes? Exceptions?)
  - Should be addressed generally—not unique to this component, or even to the persistence project

Implementation proposals

- Define a simple common service interface like getLocalName
- Deliver one common project local implementation (non-grid)
  - E.g., if we use the {GUID, localName} file approach, a common implementation would be based on this
- Deliver one common project grid-aware solution
  - Proposal: EDG implementation—find the file instance chosen by the Resource Broker, if any, by consulting BrokerInfo interface
- Allow/expect experiment-specific implementations of the common interface, e.g.,
  - Rule-based translation of LFNs to local names (e.g., prefix/suffix rules)
  - Experiment-specific catalog consultation

Proposed implementation choices

- "localName" has a pragmatic meaning—a string one can pass, e.g., to a POSIX open() call, and expect things to work (no special prefixes, no protocols, no host identification, …)
- What if a replica exists, but it is not local? Do we initiate a transfer?
  - This may happen with on-demand traversal of interobject references
  - Propose that "fetch" be a configurable runtime choice
- What if the file could be read remotely via an appropriate protocol, rather than transferred?
  - Propose that we not worry about this in early releases