GT4 Component Fact Sheet

Replica Location Service (RLS)

Last updated: 19 July 2004

General Component Description

The *replica location service (RLS)* allows the registration and discovery of replicas. An RLS maintains and provides access to mapping information from logical names for data items to target names. These target names may represent physical locations of data items or an entry in the RLS may map to another level of logical naming for the data item.

The Replica Location Service design consists of two components. Local Replica Catalogs (LRCs) maintain consistent information about logical-to-physical mappings on a site or storage system. The Replica Location Indexes (RLIs) aggregate state information contained in one or more LRCs and build a global, hierarchical distributed index to support discovery of replicas at multiple sites. LRCs send summaries of their state to RLIs using soft state update protocols. Information in RLIs times out and must be periodically refreshed. To reduce the network traffic of soft state updates and RLI storage overheads, the RLS also implements an optional Bloom filter compression scheme. In this scheme, each LRC only sends a bit map that summarizes its mappings to the RLIs. The bit map is constructed by performing a series of hash functions on logical names that are registered in an LRC and setting the corresponding bits in the bit map. Bloom filter compression greatly reduces the overhead of soft state updates. However, the bloom filter is a lossy compression scheme. Using bloom filters, the RLIs lose information about specific logical names registered in the LRCs. There is also a small probability that the Bloom filter will provide a false positive, an incorrect indication that a mapping exists in the corresponding LRC when it does not. The current implementation uses a static configuration for the distributed RLS; for each server, configuration files specify which LRCs are send updates to which RLIs.

The RLS is implemented in C and uses the *XIO libraries* from the Globus Toolkit. The server consists of a multi-threaded front end server and a back-end relational database, such as MySQL or PostgreSQL. The front end server can be configured to act as an LRC server and/or an RLI server. Clients access the server via a simple string-based RPC protocol. The client APIs support C, Java and Python. The implementation supports two types of soft state updates from LRCs to RLIs: (1) a complete list of logical names registered in the LRC and (2) Bloom filter summaries of the contents of an LRC. The implementation also supports partitioning of the soft state updates based on pattern matching of logical names.

The distributed RLI index can provide redundancy and/or partitioning of the index space among RLI index nodes. LRCs can be configured to send soft state updates summarizing their contents to one or more RLIs. When these updates are sent to multiple RLIs, we

avoid having performance bottlenecks or single points of failure in the index space. RLS also supports the capability of limiting the size of soft state updates based on a partitioning of the logical namespace. With partitioning, we perform pattern matching of logical names and send only matching updates to a specified RLI index.

What's New in GT4?

The RLS for GT4 will be very similar to the GT3 RLS. We will include a range of bug fixes that address problems that have been encountered at high scale (tens of millions of mappings per catalog and/or thousands of connections to the RLS server). The RLS released in GT4 will be more robust and higher-performance than earlier versions.

Backward Compatibility Statement

The RLS released in GT4 will be compatible with previous versions.

Technology Dependencies

- RLS requires a relational database back end (MySQL, PostgreSQL or Oracle) and an associated IODBC layer
- The GT XIO Component