Clustering the Reliable File Transfer Service

Jim Basney and Patrick Duda

Abstract— As grids move from prototypes to testbeds to production infrastructure, grid resource providers are faced with the challenge of delivering reliable services to enable productive use of available resources. On high performance, distributed grids such as the TeraGrid, moving large data sets to, from, and between supercomputing resources requires reliable data management services. The Reliable File Transfer (RFT) Service in the Globus Toolkit Version 4 (GT4) provides this capability on the TeraGrid and other grids. We present modifications to RFT to support clustering to achieve high availability in the presence of server failures, based on a standard Web service tiered architecture, leveraging the capabilities of modern database systems. Clustering distributes the RFT service across multiple tightly coupled servers so that RFT can continue to provide service even when individual components fail.

Index Terms— Globus Toolkit, Reliable File Transfer Service, High Availability, Cluster, Grid Computing, GridFTP.

1 Introduction

The Reliable File Transfer (RFT) Service [3] in the Globus Toolkit Version 4 (GT4) [4] manages GridFTP [1] operations (file transfers and deletes) on behalf of a client. The client submits a request to RFT for GridFTP operations to be performed, and RFT takes responsibility for the completion of the request, contacting GridFTP servers and restarting operations as needed. RFT removes the requirement for clients to remain online throughout the duration of GridFTP transfers and handles failures on the client's behalf. In addition to handling GridFTP failures, RFT persists its own state to allow it to recover from its own failures. At any time, clients can contact RFT to obtain the status of their requests.

While RFT provides significant improvement in reliability over GridFTP alone, the RFT service is a single point of failure vulnerable to network partition, power loss, system crash, and resource exhaustion under heavy load. The current RFT design does not support clustering or replication for high availability, fail-over, and load balancing. In this paper, we present our modifications to the GT4 RFT service to enable clustering and replication.

This work enables RFT clusters to be combined with GridFTP clusters to provide both high availability and high performance. RFT clusters provide highly available control for GridFTP transfers, while GridFTP clusters enable the high performance data movement capabilities of the GridFTP protocol via parallel TCP streams and data striping across servers [2].

2 RFT Implementation

Before describing our modifications to RFT, we first describe the existing RFT design and implementation in GT4, focusing on those aspects that impact our modifications.

2.1 Delegation Service

RFT depends on the GT4 Delegation Service (DS) for credential management. Before submitting a request, the client delegates credentials to the DS residing in the same Web Services container as RFT. Then, the client includes the WS-Addressing [6] Endpoint Reference (EPR) for the credentials in its request to RFT, which RFT uses to obtain credentials to perform the GridFTP operations on the client's behalf.

The DS stores its credentials in files on disk. The interface for accessing the credentials is internal to the container, so the DS and RFT must be co-located. A client must submit its request to the RFT service in the same container as the DS to which it delegated its credentials. Thus, any replication of the RFT service must either replicate the DS as well or must remove this requirement for co-location (i.e., by providing an external interface for communication between RFT and DS services across containers).

To avoid credential expiration while transfers are in progress, a client can delegate fresh credentials to the DS as needed. RFT registers with the local DS to receive updated credentials when they are renewed. This functionality would also need to be maintained in a replicated scenario.

2.2 RFT Resource Properties

Clients can obtain the status of RFT transfers (Finished/Active/Failed/Restarted/Pending/Canceled state plus bytes transfered and duration of transfer) via WS-ResourceProperties [5] (RPs) and can subscribe to notifications of RP updates. These subscriptions are maintained by the GT4 Core Persistence API, which stores subscription data to files on disk. Replicating RFT must include replication of these RPs, making sure updates are propagated to replicas, so clients can query any replica to obtain status information, and notifications are issued once-and-only-once.

2.3. RFT Database

RFT maintains the state of transfers in a DBMS accessed via JDBC, with documented support for PostgreSQL and MySQL. Multiple RFT instances require separate database instances for two reasons. First, synchronization is performed inside the RFT service rather than via database locks, so database consistency would be a problem if multi-

- *Jim Basney, National Center for Supercomputing Applications, University of Illinois, jbasney@ncsa.uiuc.edu.*
- *Patrick Duda, National Center for Supercomputing Applications, University of Illinois, pduda@ncsa.uiuc.edu.*

ple RFT instances used the same database. Second, on recovery, the RFT service assumes ownership of all requests in the database, which would cause it to take over requests owned by other instances, initiating transfers and modifying the associated transfer data, again resulting in consistency problems.

In summary, RFT is a stateful web service that depends on the stateful DS. RFT transfer state is persisted to database, while RFT RP subscription state is persisted to disk. DS state is also persisted to disk, and the DS must be able to notify RFT when credentials are refreshed. To deploy an RFT cluster, we must implement mechanisms to replicate this state across RFT instances, handle notifications for changes to DS and RFT resources, and implement transfer restart/recovery across the RFT instances.

3. Clustering Approach

To implement an RFT service cluster, we decided to leverage the clustering and consistency mechanisms provided by modern DBMSs. We modified the existing RFT database tables so they can be shared across multiple RFT instances, and we modified the DS and RFT services to persist all data to a shared database, rather than to disk. Using a single mechanism for all data persistence simplifies software design and system management, and by leveraging the JDBC standard, we have many DBMS options for deployment in different environments. While we have thus far experimented with MySQL, which supports both synchronous and asynchronous replication, replication systems are also available for PostgreSQL, and major commercial DBMSs provide advanced mechanisms for clustering, replication, and high availability.

Clients can submit requests and status queries to any RFT and DS instance in the cluster. Standard load-balancing techniques such as round-robin DNS or HTTP proxy servers can be applied to target requests to the different instances. RFT requests can reference credentials stored on any DS in the cluster, as any DS instance can retrieve the needed credentials from the shared database on behalf of its local RFT instance. The shared DS database table is indexed by EPR, providing a unique namespace for each DS instance based on its IP address.

All RFT instances share a Request, Transfer, and Restart table. The Request table is modified from the existing RFT version, adding a container ID which indicates which RFT instance currently owns a given Request entry and a start time which indicates when the Request was started by the client. A Request entry is initially owned by the RFT instance where it was submitted and will only fail-over to another RFT instance if it is not handled in a timely fashion. The Transfer table contains the individual operations (transfer or delete) that make up a request. It is unmodified. The Restart table holds the restart markers for in-progress transfers, and it is modified to include a timestamp to detect stalled transfers requiring fail-over.

3.1. Fail-Over

Fail-over in our clustering approach is based on timeouts. Periodically (default: 30 seconds), each RFT instance queries the database for Requests that started a while ago (default: over 60 seconds) but have no recent Restart table entries (default: under 60 seconds) and have not completed. Under normal circumstances, this query will return no results, as all RFT instances are properly handling their Requests. However, if any such stalled Requests are found, indicating that one or more RFT instances is overloaded or has failed, this instance obtains a write lock on the RFT tables, runs the query again, claims all resulting Requests for itself, then releases the lock. This algorithm ensures via DBMS locking that each stalled Request will be taken over by at most one RFT instance.

When a Request fails-over to a new RFT instance, that instance then continues the Request's Transfers from where they left off (using the stored Restart markers), and sends notifications on updates to those Requests to any clients that have registered subscriptions. Queries for the status of a request can be satisfied by any RFT instance by simply looking up the information in the shared RFTdatabase.

4. Evaluation

To evaluate our clustering solution, we performed experiments on a dedicated cluster, with a switched Gigabit Ethernet network, running Red Hat Enterprise Linux AS release 3. Each node had 2 GB RAM with dual 2GHz Intel XEON CPUs with 512KB caches. We ran GT v4.0.3 Web Services containers and GridFTP servers, with MySQL Standard v5.0.27.

Our experiments focus on RFT performance. GridFTP provides a variety of mechanisms for high performance data movement, and RFT's role is to provide reliable access to the GridFTP capabilities without itself becoming a bottleneck. We therefore perform our experiments with many small files to maximize the load on RFT. If RFT performs well under this extreme load, we can have confidence that it will perform well under more typical loads consisting of smaller numbers of large files, where GridFTP file transfer time dominates performance.

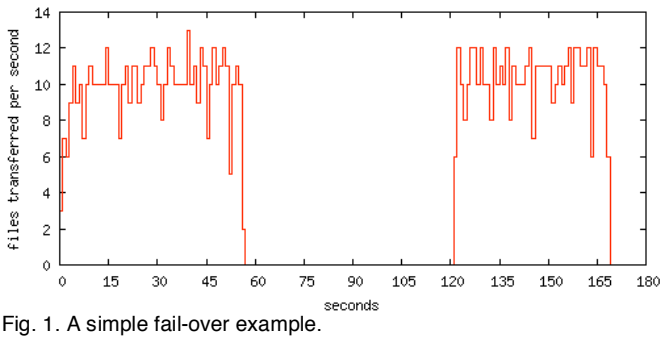

Fig. 1. A simple fail-over example.

Our first experiments were to verify the basic functionality of the clustering and fail-over implementation of our modified RFT service running across 12 nodes of the clus-

ter. In Figure 1, we see an example of fail-over in action. We submitted an RFT Request to transfer 1000 1MB files, up to 5 at a time. 55 seconds later, we killed the Web Services container to which we submitted the request, and the GridFTP transfers ceased. 65 seconds after that, one of the 11 remaining RFT instances noticed that no activity had occurred for this Request for over 60 seconds and took it over, completing the remaining GridFTP transfers. During this experiment, the RFT client received notifications for all 1000 transfers, which succeeded successfully. We have tested the fail-over capability with up to 12 instances, with multiple requests of different sizes, to verify correct functionality.

We also performed experiments comparing the performance of our clustered RFT with the current GT4 RFT service. We ran GT4 RFT instances on 10 nodes with a local MySQL server to provide a performance baseline, then ran our modified RFT instances on 10 nodes, connected to a MySQL server on another node. To each RFT instance, we submitted a single RFT Request to transfer 1000 1MB files, up to 5 at a time. We submit a large number of small transfers to create a heavy database load, since the database is the critical component for our comparison. Since the load on the shared MySQL server instance increases as the number of active containers increases, we performed individual experiments submitting simultaneous requests to 1–10 RFT instances.

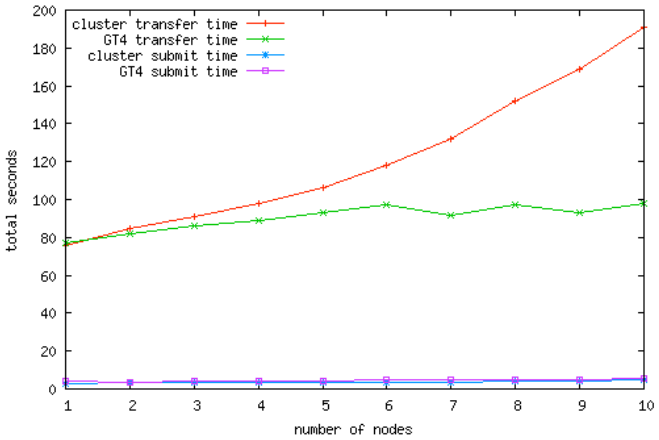

Fig. 2. A performance comparison.

As we see in Figure 2, the performance of the GT4 RFT service did not change significantly as we submitted requests to more containers simultaneously. This result is expected, since the GT4 RFT instances run independently, with their own local GridFTP servers and MySQL databases. The small increase, as the number of containers is increased, in the time to submit the requests and complete the transfers, can be explained by the overhead of submitting all requests from the cluster head node and the use of a shared NFS filesystem for container logs. It took 4 seconds to submit the requests, and 77–93 seconds to transfer the 1000 files on each node (11–13 GridFTP transfers per second).

As expected, the MySQL database becomes a bottleneck for the RFT cluster as the number of active RFT instances increases. For a single node, performance for GT4 is the same as for the RFT cluster (77 seconds for 1000 files, 13 transfers per second on average). For 2 nodes, the transfers took 85 seconds rather than 82 (4% overhead), and 91 versus 86 seconds (6% overhead) for 3 nodes. For 10 nodes, the transfers take approximately twice as long in the cluster than they do for the independent RFT instances, but the performance is still relatively good, at 52 transfers per second for the 10 node cluster. The time to process each transfer is insignificant in comparison with the transfer time of large files in typical GridFTP workloads.

Finally, in our experiments, we found that RFT did not handle DBMS errors well. If the RFT instance started up before MySQL, or MySQL restarted, or the MySQL network connection was otherwise severed, the RFT instance would cease operation. We modified RFT to reconnect to the database in these cases.

5. Related Work

We believe that ours is the first effort to cluster GT4 web services for high availability. We are aware of two related projects.

The HAND [7] infrastructure implements a dynamic redeployment capability for services in the Globus Toolkit, providing the ability to migrate services between containers, to maintain availability in the event of scheduled server outages, but it does not address the management of persistent service state or fail-over in the case of unplanned outages.

The myGrid [8] project developed services based on Apache WSRF to support DBMS persistence of WS-ResourceProperties. In contrast, our DBMS persistence support is specific to the RFT and DS services in the Globus Toolkit using the GT4 Java WS Core. Investigating a general-purpose DBMS-based persistence solution for GT4 is an important area of future work.

6. Conclusion and Future Work

We have presented our modifications to the GT4 RFT service to enable clustering for load-balancing and fail-over. Our initial experiments demonstrate the effectiveness of our approach, with acceptable performance overheads for small service clusters. We believe that clustering is a promising approach for application to other grid services.

We have submitted our modifications to the RFT developers, and we plan the following future work:

- **Correctly handle replay of FTP deletes.** The current implementation assumes all operations are idempotent and thus can be replayed on fail-over. This is sufficient for transfer operations, but replayed deletes will currently fail with a ``no such file or directory'' error. We plan to investigate the desirability of a two-phase commit solution, compared to the optimistic approach of simply ignoring this specific error.
- **Implement credentialRefreshListener.** The credential-

RefreshListener interface notifies RFT when a DS credential has been updated (for example, before it expires). It currently works only in the container to which the credential was delegated, but other RFT instances in the cluster may also need to receive this notification to update their credentials for ongoing transfers. We plan to use database mechanisms to detect credential updates across the cluster to generate the needed notifications.

- **Evaluate use of different DBMS solutions.** To-date we have experimented primarily with standalone MySQL servers. Additional experiments using clustered databases are needed to validate our design. Candidates include MySQL Cluster; PGCluster, Slony-I, and Postgres-R for PostgreSQL; Oracle Real Application Clusters (RAC); and Microsoft SQL Server High Availability.
- **Investigate GT4 DBMS persistence in general.** Rather than developing DBMS interfaces separately for each GT4 service, a general-purpose implementation of the GT4 Core Persistence API that supports DBMS storage and synchronization of WS-Resources would enable clustering of multiple GT4 services.
- **Investigate use of WS-Naming.** While network transport layer services such as DNS and HTTP proxy servers provide effective mechanisms for locating clustered services, a Web Services approach based on WS-Addressing can provide configurable and dynamic grid-level or application-level resolvers for locating replicated grid services. The OGSA Naming working group in the Open Grid Forum is working to provide standards for WS-Addressing-based name resolution.

Please see http://grid.ncsa.uiuc.edu/dependable for the source code modifications and additional supporting materials.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 0426972.

Performance experiments were conducted on computers at the Technology Research, Education, and Commercialization Center (TRECC), a program of the University of Illinois at Urbana-Champaign, funded by the Office of Naval Research and administered by the National Center for Supercomputing Applications. We thank Tom Roney for his assistance with the TRECC cluster.

We also thank Ravi Madduri from the Globus project for answering our questions about RFT.

REFERENCES

[1] W. Allcock, "GridFTP Protocol Specification", Global Grid Forum Recommendation GDF.20, March 2003.

[2] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, I. Foster, "The Globus Striped GridFTP Framework and Server", Proceedings of Super Computing 2005 (SC05), November 2005.

[3] W. Allcock, I. Foster, and R. Madduri, "Reliable Data Transport: A Critical Service for the Grid", In Global Grid Forum Building Service Based Grids Workshop, June 2004.

[4] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems", In IFIP International Conference on Network and Parallel Computing, pages 2–13. Springer-Verlag LNCS 3779, 2006.

[5] S. Graham and J. Treadwell, editors, "Web Services Resource Properties 1.2", W3C, April 2006.

[6] M. Gudgin, M. Hadley, and T. Rogers, editors, "Web Services Addressing 1.0 – Core", W3C, May 2006.

[7] L. Qi, H. Jin, I. Foster, and J. Gawor, "HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4", 2006.

[8] B. Wietrzyk and M. Radenkovic, "Supporting Semantic Life Science Middleware with Web Service Resource Framework", In Proceedings of the UK e-Science All Hands Meeting, September 2005.