Reliable Data Transport: A Critical Service for the Grid

William E. Allcock, Ian Foster, Ravi Madduri

Mathematics and Computer Science Division
Argonne National Laboratory
{allcock, foster, madduri}@mcs.anl.gov

Introduction

Many Grid applications require secure, fast, efficient, and robust data transport mechanisms, for example when moving a large dataset to or from a remote compute server, or when fetching data from several locations for the purpose of integration [1,3,7].

GridFTP [2,3,4,12] has become the de facto standard for moving data in many Grid projects. However, GridFTP does not offer a web service interface. In addition, GridFTP requires that the client driving a remote transfer keep a control channel (i.e., a TCP/IP socket) open to participating servers. Thus, a user that wishes a transfer to run for a week must have a machine that can stay on the net, running this client application, for a week, without any problem such as an NFS failure or network problems. Clearly this can be a challenge in practice, and indeed may run counter to the frequent need for mobile or otherwise disconnected clients. Client failure is problematic for another reason, too: GridFTP is robust with respect to remote failures, but as transfer state is held in memory, there is no way to know where to restart a transfer if the client fails.

Clearly, there is a need for a data transport mechanism that complies with a standard service specification, provides functionality similar to a job submission service for compute jobs, and provides reliability even in the face of local failure.

Within the Globus Alliance [9], we chose to resolve these issues by developing a Reliable File Transfer [5,13] (RFT) service. RFT is available in the Globus Toolkit (GT) V3.0 and V3.2 as an OGSI [8] compliant service, and will be available in GT4.0 as a WSRF [10,11] compliant service. (For simplicity, we structure this presentation in terms of OGSI mechanisms; the changes to exploit WSRF are minor.) This service acts as a proxy for the user, operating as a client on the user's behalf to drive third party transfers. It can maintain open socket connections for weeks. It accepts a SOAP [14] description of a data transfer "job," which may consist of a single transfer or thousands of transfers, and it writes the request and transient state to a database, so that in the event of a service failure, it can resume from the last checkpoint written to the database. The user can check in at any time to determine the status of a transfer, and can also request notifications of completion.

All operations discussed below need appropriate security settings to succeed. The user needs to be authorized and authenticated to perform operations on the RFT service and on service instances created by that service. We provide a command-line client that can

be used to submit transfer requests to the RFT factory, which returns the handle to the transfer service that should be used for all the remaining operations.

In the remainder of this paper we discuss the various XML schema [15,16], the port types defined, and some of our experiences building RFT.

The RFT factory CreateService Operation

The RFT CreateService operation is modeled as a request to create a new *transfer service*, which is then used to represent, monitor, and manage the new request. (The operation creates a new service for the new request, but the transfer does not start until the service or WS-Resource's start() operation is called.) The schema for a transfer request is as follows; it specifies the file(s) to be transferred (transferArray), the default parameters to use for each file transfer in the request (rftOptions), and the concurrency level to use for the transfers (concurrency).

```
<xsd:complexType name="TransferRequestType">
    <xsd:sequence>
        <xsd:element name="transferArray" type="rft-types:TransferType"
                     minOccurs="1" maxOccurs="unbounded"/>
        <xsd:element name="rftOptions" type="rft-types:RFTOptionsType"
                     minOccurs="1" maxOccurs="1"/>
        <xsd:element name="concurrency" type="int" minOccurs="1"
                     maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
```

Each transfer specification comprises a source / destination URL pair, with an optional rftOptions block to override the default parameters for this particular file transfer. The transfer type is defined as follows.

```
<xsd:complexType name="TransferType">
    <xsd:sequence>
        <xsd:element name="transferId" type="int"/>
        <xsd:element name="sourceUrl" type="string"/>
        <xsd:element name="destinationUrl" type="string"/>
        <xsd:element name="rftOptions" type="rft-types:RFTOptionsType"
                     minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
```

The RFT service is designed to provide access to the full functionality of GridFTP, and thus the RFT request schema allows for the complete set of options available in the Globus implementation of the GridFTP protocol, including TCP buffer size, parallel streams, and binary/ASCII. The rftOptions type is defined as follows.

```
<xsd:complexType name="RFTOptionsType">
    <xsd:sequence>
        <xsd:element name="binary" type="boolean"/>
        <xsd:element name="blockSize" type="long"/>
```

```
        <xsd:element name="tcpBufferSize" type="int"/>
        <xsd:element name="notpt" type="boolean"/>
        <xsd:element name="parallelStreams" type="int"/>
        <xsd:element name="dcau" type="boolean"/>
        <xsd:element name="subjectName" type="string"/>
        <xsd:element name="destinationSubjectName" type="string"/>
        <xsd:element name="sourceSubjectName" type="string"/>
    </xsd:sequence>
</xsd:complexType>
```

The RFT service instance start() Operation

Start() does not take any parameters and returns a request id. This request ID is not strictly necessary, but is a legacy from the first implementation and was maintained to avoid changing the interface. The user must preserve the Grid service handle (GSH) of the new transfer service, which is returned by the CreateService() call. The GSH can be used to query the status and to destroy the service instance after the transfers are done. Delegation of the user's proxy also occurs in the start operation. The GWSDL for the start() operation and its response are as follows.

```
<xsd:element name="start" type="tns:Start"/>
<xsd:complexType name="Start"/>

<xsd:element name="startResponse" type="tns:StartResponse"/>
<xsd:complexType name="StartResponse">
    <xsd:sequence>
        <xsd:element name="requestId" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>
```

Operation / PortType getStatus():

This operation is used by a client to obtain the status of a *single* transfer (file) in a request, to determine whether or not it has completed. The operation takes the sourceUrl as an argument and returns a FileTransferJobStatusType containing the status of the transfer and the destination URL for the transfer. If there are multiple transfers with the same source URL, it returns the transfer on the top of the transfer stack. The GWSDL for the getStatus() operation and its response are as follows.

```
<xsd:element name="getStatus" type="tns:GetStatus"/>
<xsd:complexType name="GetStatus">
    <xsd:sequence>
        <xsd:element name="sourceUrl"
                     type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="getStatusResponse" type="tns:GetStatusResponse"/>
<xsd:complexType name="GetStatusResponse">
    <xsd:sequence>
        <xsd:element name="rftStatus"
                     type="rft-types:FileTransferJobStatusType"/>
    </xsd:sequence>
</xsd:complexType>
```

The response contains two complex types. The first is FileTransferJobStatus, which as follows.

```
<xsd:complexType name="FileTransferJobStatusType">
    <xsd:sequence>
        <xsd:element name="transferId" type="int"/>
        <xsd:element name="destinationUrl" type="string"/>
        <xsd:element name="status" type="rft-types:TransferStatusType"/>
    </xsd:sequence>
</xsd:complexType>
```

The TransferStatusType is defined as follows.

```
<xsd:simpleType name="TransferStatusType">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="Finished"/>
        <xsd:enumeration value="Retrying"/>
        <xsd:enumeration value="Failed"/>
        <xsd:enumeration value="Active"/>
        <xsd:enumeration value="Pending"/>
        <xsd:enumeration value="Cancelled"/>
    </xsd:restriction>
</xsd:simpleType>
```

Operation / PortType getStatusGroup :
The getStatusGroup operation is similar to getStatus() above, except that (a) it returns status information for *multiple* transfers in a request, and (b) it selects the transfers based on the position (index) in the request rather than by source URLs. Additional options allow a client to specify an initial number and an offset. Thus, a client can get the status of an entire large request in stages, so as to avoid returning a response larger than can be de-serialized.

```
<xsd:element name="getStatusGroup" type="tns:GetStatusGroup"/>
<xsd:complexType name="GetStatusGroup">
    <xsd:sequence>
        <xsd:element name="initial"
                     type="xsd:int"/>
        <xsd:element name="offset"
                     type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="getStatusGroupResponse"
             type="tns:GetStatusGroupResponse"/>
<xsd:complexType name="GetStatusGroupResponse">
    <xsd:sequence>
        <xsd:element name="rftStatusGroup"
                      type="rft-types:FileTransferJobStatusType"
                      maxOccurs="unbounded"
                      minOccurs="0"/>
```

```
    </xsd:sequence>
</xsd:complexType>
```

Operation / PortType cancel
This operation is used to cancel a transfer. (Currently, one cannot cancel a transfer if it is already in progress due to threading issues within the GridFTP java client libraries.) The transfers to cancel are specified by their position (index) in the request.

```
<xsd:element name="cancel" type="tns:Cancel"/>
<xsd:complexType name="Cancel">
    <xsd:sequence>
        <xsd:element name="requestId" type="xsd:int"/>
        <xsd:element name="fromId" type="xsd:int"/>
        <xsd:element name="toId" type="xsd:int"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:element name="cancelResponse" type="tns:CancelResponse"/>
<xsd:complexType name="CancelResponse"/>
```

Service Data Elements (SDEs)

OGSI improves on previous protocols used in the Globus Toolkit by providing a powerful mechanism for defining, inspecting, and subscribing to notifications concerning changes in internal state. State is defined in terms of Service Data Elements (SDEs). The status operations discussed above represent a pull mechanism for state and are appropriate for a client that has been disconnected and is looking for a “status report.” Subscription to a notification is a push mechanism and works well for an interactive client that remains attached, or perhaps a broker that can take action on each state change. The client may simply update a progress bar, while the broker may use new information to make scheduling decisions.

We put an emphasis in RFT on exposing the maximum possible internal state and state change notifications. We list below the SDEs that we provide, along with a brief description of each. SDEs can be queried using the OGSI-standard findServiceData operation, or by subscribing via the standard subscribe operation.

Version: This SDE provides the version of RFT service that is running.

```
<xsd:complexType name="Version">
    <xsd:sequence>
        <xsd:element name="version" type="string"/>
    </xsd:sequence>
</xsd:complexType>
```

FileTransferProgress: This SDE provides the progress of a given file transfer in terms of the number of bytes successfully transferred as a percentage of the file size.

```
<xsd:complexType name="FileTransferProgressType">
    <xsd:sequence>
        <xsd:element name="percentComplete" type="int"/>
    </xsd:sequence>
</xsd:complexType>
```

FileTransferStatusElement: This SDE is used to represent transfer state. Each time a transfer changes state, such as from pending to running, or running to complete, a notification is generated to any subscribed clients.

```
<xsd:complexType name="FileTransferStatusElement">
    <xsd:sequence>
        <xsd:element name="requestStatus" type="rft-
         types:FileTransferJobStatusT
ype"/>
    </xsd:sequence>
</xsd:complexType>
```

OverallStatus: This SDE provides overall status of a request (transfer job). This includes the number of active transfers, total pending, complete, number of restarts attempted, and the number of transfers that have been cancelled.

```
<xsd:complexType name="OverallStatus">
    <xsd:sequence>
        <xsd:element name="transfersFinished"
                     type="rft-types:TransfersFinished"/>
        <xsd:element name="transfersActive"
                     type="rft-types:TransfersActive"/>
        <xsd:element name="transfersPending"
                     type="rft-types:TransfersPending"/>
        <xsd:element name="transfersFailed"
                     type="rft-types:TransfersFailed"/>
        <xsd:element name="transfersRestarted"
                     type="rft-types:TransfersRestarted"/>
        <xsd:element name="transfersCancelled"
                     type="rft-types:TransfersCancelled"/>
    </xsd:sequence>
</xsd:complexType>
```

GridFTPRestartMarkerElement: This SDE exposes the raw GridFTP restart markers, each of which corresponds to a block of data that has been successfully written at the receiving server. This information is stored in the database and used to restart a failed transfer.

```
<xsd:complexType name="GridFTPRestartMarkerType">
    <xsd:sequence>
        <xsd:element name="transferId" type="int"/>
        <xsd:element name="restartMarker" type="string"/>
    </xsd:sequence>
</xsd:complexType>
```

GridFTPPerfMarkerElement: This SDE exposes the raw GridFTP performance

markers. These markers indicate the total number of bytes transferred, the total time the transfer has been running, the instantaneous bandwidth (defined as the bandwidth as calculated between this marker and the previous marker) and the overall average bandwidth (calculated by dividing total bytes moved so far, by total transfer time so far).

```
<xsd:complexType name="GridFTPPerfMarkerType">
    <xsd:sequence>
        <xsd:element name="transferId" type="int"/>
        <xsd:element name="timeStamp" type="double"/>
        <xsd:element name="stripeIndex" type="long"/>
        <xsd:element name="stripeBytesTransferred" type="long"/>
        <xsd:element name="totalStripeCount" type="long"/>
    </xsd:sequence>
</xsd:complexType>
```

Summary

The Reliable File Transfer (RFT) service represents a fundamental need for the Grid: a standard service for data transport. As noted earlier, RFT can be thought of as a scheduler for data transfer "jobs." Thus, we might expect that RFT could share the interface and much of the exposure of state that a job scheduler service might have. We have often discussed this design approach, and it is likely that we will move in that direction once the WS-Agreement work becomes reasonably stable. However, we will always need some kind of extensibility element so that different types of jobs can expose different information. For instance, RFT exposes the number of restarts that have been attempted, the current restart marker, and the current bandwidth. These are important data, but make no sense for a compute job, just as sub-jobs per processor makes no sense for transfers.

Also, though a comment on current technology more than interfaces, we were forced to add the "get status on a range of IDs" operation because, at least on the systems we have used, the XML is always deserialized into a DOM tree. This means the entire blob needs to fit in memory. A transfer request may consist of a million files, and we cannot (with current technology) deserialize a million lines of XML into a DOM tree. A SAX parser would avoid this problem, and would be more like a stream: the receiver takes what it can handle and then the sender must wait until it signals it is ready again. This is a huge scalability issue. Our original implementation was limited to approx 450 files because that is all you could fit into memory specifying each transfer fully one at a time. However, by specifying the default up front, that cut down the repeated data, but the big scalability win was the addition of directory support. Now someone can specify a singe source and destination URL, thus keeping the request small, but have that expand to be millions of files as the directory is traversed.

Acknowledgment

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

References

---

[1] The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke. *Journal of Network and Computer Applications*, 23:187-200, 2001 (based on conference publication from Proceedings of NetStore Conference 1999)

[2] Data Management and Transfer in High Performance Computational Grid Environments. B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. *Parallel Computing Journal*, Vol. 28 (5), May 2002, pp. 749-771.

[3] High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies. B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, A. Shoshani, B. Drach, D. Williams. *SC 2001*, November 2001.

[4] GridFTP Protocol Specification (Global Grid Forum Recommendation GFD.20). W. Allcock, editor. March 2003. http://www.globus.org/research/papers/GFD-R.0201.pdf

[5] Reliable File Transfers in Grid Environments R. Madduri, W. Allcock, C. Hood *Proceedings of the the 27th IEEE Conference on Local Computer Networks*. p737-738

[6] Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke. *IEEE Mass Storage Conference*, 2001, p13-28 .

[7] http://eu-datagrid.web.cern.ch/eu-datagrid/

[8] https://forge.gridforum.org/projects/ogsi-wg

[9] http://www.globus.org

[10] http://www.globus.org/WSRF/

[11] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf

[12] http://www-unix.globus.org/toolkit/docs/3.2/index.html#data

[13] http://www-unix.globus.org/toolkit/reliable_transfer.html

[14] http://www.w3.org/TR/soap/

[15] http://www.w3.org/XML/Schema

[16] http://www.w3.org/XML/