

LIGO Data Replicator (LDR) Administrator's Guide

Contact: Scott Koranda (Librarian) skoranda@gravity.phys.uwm.edu

RCS Revision: 1.81 Date: 2009/07/15 19:56:01 UTC — Compiled: 2009/08/07

Contents

Preface	iii
1 Overview	1
1.1 Synopsis	1
1.2 Features	1
1.3 Design	2
1.3.1 Terminology	2
1.3.2 Basic design	2
1.3.3 LDR components	3
1.3.4 Integration with local storage	4
1.3.5 Anatomy of a file replication	4
1.4 Changes	7
1.4.1 Changes from LDR 0.9.3 to LDR 0.9.4	7
1.4.2 Changes from LDR 0.9.2 to LDR 0.9.3	7
1.4.3 Changes from LDR 0.9.1 to LDR 0.9.2	7
1.4.4 Changes from LDR 0.9.0 to LDR 0.9.1	7
1.4.5 Changes from LDR 0.8.0 to LDR 0.9.0	7
1.4.6 Changes from LDR 0.7.0 to LDR 0.8.0	8
1.4.7 Changes from LDR 0.6.x to LDR 0.7.0	8
2 Installing LDR	10
2.1 Supported Platforms	10
2.2 Hardware requirements	10
2.3 Required software packages	11
2.4 Upgrading from LDR 0.8.x	11
2.4.1 Preparation	11
2.4.2 Dumping database contents	11
2.4.3 Shutting down MySQL	12
2.5 Installing required software	12
2.5.1 CentOS	13
2.5.2 Scientific Linux	13
2.5.3 Debian	13
3 Configuring MySQL, RLS, GridFTP, Apache, and LDR	14
3.1 Configuring, initializing, and restoring state for MySQL	14
3.1.1 Securing MySQL	14
3.1.2 Configuring the location of MySQL table data files	14
3.1.3 Configuring the MySQL InnoDB engine for maximum performance	15
3.1.4 Initializing MySQL for LDR	15
3.1.5 Restoring MySQL state from LDR 0.8.x	16
3.2 Obtaining Credentials or X.509 Certificates	17
3.2.1 Requesting host and datarobot certificates	17
3.2.2 Locating the host and datarobot certificates	18
3.2.3 Working with the datarobot certificate	19
3.3 Configuring and Initializing Globus RLS for LDR	19
3.3.1 Edit the monit configuration file	19
3.3.2 Edit the RLS configuration file	19

3.3.3	Edit the ODBC configuration file	20
3.3.4	Starting the RLS server	20
3.3.5	Final configuration using globus-rls-admin	21
3.4	Configuring globus-gridftp-server	21
3.5	Configuring Apache	21
3.5.1	Securing the default Apache configuration	22
3.5.2	Configuring the LDR web services	22
3.6	Configuring LDR	23
3.6.1	Editing ldr.ini	23
3.6.2	Local Storage Module	29
3.7	Firewalls	31
3.7.1	Static and empheral ports	31
3.7.2	Listening ports needed by individual LDR components	31
3.7.3	Ephemeral ports needed by individual LDR components	32
3.7.4	Summary of all ports needed	33
4	Running LDR	34
4.1	Quick instructions for long-time LDR admins	34
4.2	Overview	34
4.2.1	Starting globus-gridftp-server	35
4.2.2	Starting MySQL	35
4.2.3	Starting RLS	35
4.2.4	Publishing metadata and URLs	35
4.2.5	Obtaining existing metadata	35
4.2.6	Starting LDR	36
4.2.7	Log files	36
4.2.8	Shutting down LDR and its components	36
5	Publishing Data into LDR	37
5.1	LDRPublishLIGOLab	37
5.2	LDRBundle	37
6	Configuring LDRDataFindServer for Private Network Interfaces	39
6.1	Turning off the default virtual host	39
6.2	Configuring virtual host for public network	39
6.2.1	Debian	39
6.2.2	CentOS and SL	41
6.3	Configuring virtual host on private network	42
6.3.1	Debian	42
6.3.2	CentOS and SL	43
6.4	Configure LDRDataFindServer	44
7	LDR As a Service	45
7.0.1	Deploying GridFTP for the LDR service	45

Preface

We continue to actively develop the LIGO Data Replicator (LDR). Please expect LDR to evolve—at times quickly without backward compatibility. We think meeting real-world production needs is our most important goal and sometimes that means abandoning old code.

We intend LDR to be a production tool that administrators can use to get real work done, as opposed to a research project to satisfy our own interests. There are enough real-world challenges without us dreaming up new ones.

Please send us your comments, suggestions, rants...any input you have. We do want to hear it.

Chapter 1

Overview

1.1 Synopsis

LDR is...

- a system for replicating bulk data sets or file sets from one site to another.
- a system for data or file discovery by people and their compute jobs across multiple sites or grid.
- a collection of grid middleware sewn together using the Python scripting language as glue.

It is *not*

- a "data on demand" resource for staging data and managing a storage resource.
- a real-time data replication tool for nearly instantaneous transmission of any single data file from one site to another.
- reliant on any central server or LDR installation—we want you to control your own site's details and data replication strategy, we don't want to do it for you (unless of course you ask us to do it for you with "LDR as a service").

1.2 Features

Our first customer is always the gravitational wave detector community, and so LDR has evolved to meet the needs of specifically the LIGO Scientific Collaboration (LSC) in support of the LIGO and GEO gravitational wave experiments. We especially look forward to collaborating with the Virgo community and helping to evolve a set of data replication tools useful for both LIGO and Virgo. LDR features include

- a distributed metadata catalog system. There is no central metadata catalog or database since we wanted to avoid a single point of failure and having to keep up a 24 by 7 server.

This choice brings with it certain assumptions about how metadata itself flows from site to site, and certain restrictions on how the metadata system as a whole works. More details follow later.

- a tightly integrated and custom GridFTP-savvy client for moving files. The Globus GridFTP API is rich and straightforward to use so there is no reason for us to not leverage it and integrate it tightly within LDR. This allows, among other things, admins to provide for a wide variety of "callback" actions to take place whenever a file is transferred.
- administrator friendly controls. Our admins have demanded it so we have tried to deliver. The LDR daemons can be reconfigured on the fly and then sent a SIGHUP to make the changes go live. We provide for extensive and customizable logging. Still, much more can be done to make LDR more administrator friendly so a lot of new work is happening to make our vision a reality.

- reliability. Our goal has been that LDR would have a mean time between failure of one month or more. That is, we would like LDR to be able to continuously move data and make it available for discovery for three months without an administrator having to do anything. We have for some sites met this goal, though this level of reliability has not come without some pain and is still only achieved for some sites and only after sufficient administrator experience and “tuning”. We hope to make this type of reliability more easily achievable “out of the box”.

For the experts: We don’t use the notion of a global unique file identifier or GUID. If you insist that we have a GUID then we will tell you that each LFN is its own GUID. Hence, we assume that every LFN is unique across a data grid. If two PFNs point to the same “thing” then that “thing” is a LFN (a GUID if you insist).

To put it as simply as we can, our metadata catalog has LFNs and then metadata attributes for the LFNs. That’s it, nothing more.

1.3 Design

1.3.1 Terminology

Before examining the LDR design it is helpful to explain some terminology:

- *LFN*: Logical File Name. Simply the notion of a file without any particular reference to where a file is located on some file system or other storage device. An example of a LFN is [H-RDS_R.L3-894327635-128.gwf](#).
- *PFN*: Physical File Name. A pointer, usually a URL or URN, to a specific instance of an LFN. There may be multiple PFNs associated with any particular LFN. An example of a PFN is [gsiftp://dataserver.phy.syr.edu:15000/data/H-RDS_R.L3-894327635-128.gwf](#).
- *publishing*: Publishing or publication of a file “into” LDR means that an entry is created for the LFN in a LDR metadata catalog (explained below) and at least one PFN for the LFN is entered into at least one site local replica catalog (also explained below). In short publishing a file means to record the existence of a file and metadata about that file such as size, checksum, GPS start and end times, and from which instruments the file holds data.
- *pset*: Publication set. Every LFN must be part of a *single* publication set or pset. The psets to be sets of LFNs that have some intrinsic interest as a set. For example a pset could be all the data files recorded from a particular site during a specific science run of the instrument, for example “LHO_S5”, “LLO_S5”, or “Virgo_VSR1”.
- *collection*: A collection is a set of LFNs to be replicated to a local site and is usually defined by constraints on the metadata attributes for the LFNs (other ways of defining collections are possible, see below for details). A collection may be the same as a pset but it need not be. For example a pset such as “LHO_S5” probably includes all data published at the LIGO Hanford site during the S5 run including raw frames, level 3 RDS frames, and strain or $h(t)$ frames, but a local site may wish to only replicated level 3 reduced (RDS) frames so the collection “S5 RDS level 3 from Hanford” is a subset of pset “LHO_S5”. Note that a collection may cross pset boundaries. For example the collection “S5 RDS level 3 from Handord *and* Livingston” includes LFNs in both the “LHO_S5” and “LLO_S5” psets.

1.3.2 Basic design

The basic LDR design is depicted in figure 1.1. The logical flow of decision making and information gathering to accomplish file replication involves the following steps:

1. A set of constraints on metadata attributes for LFNs is compared to the current LFN metadata catalog to determine the list of LFNs making up a *collection* of LFNs or files that is to be replicated. Typical constraints on metadata attributes include a range for GPS times, a “run tag” such as S5 or VSR1, a frame type for the frame file, and a site type such as “H” for the LIGO Hanford site or “V” for the Virgo site.

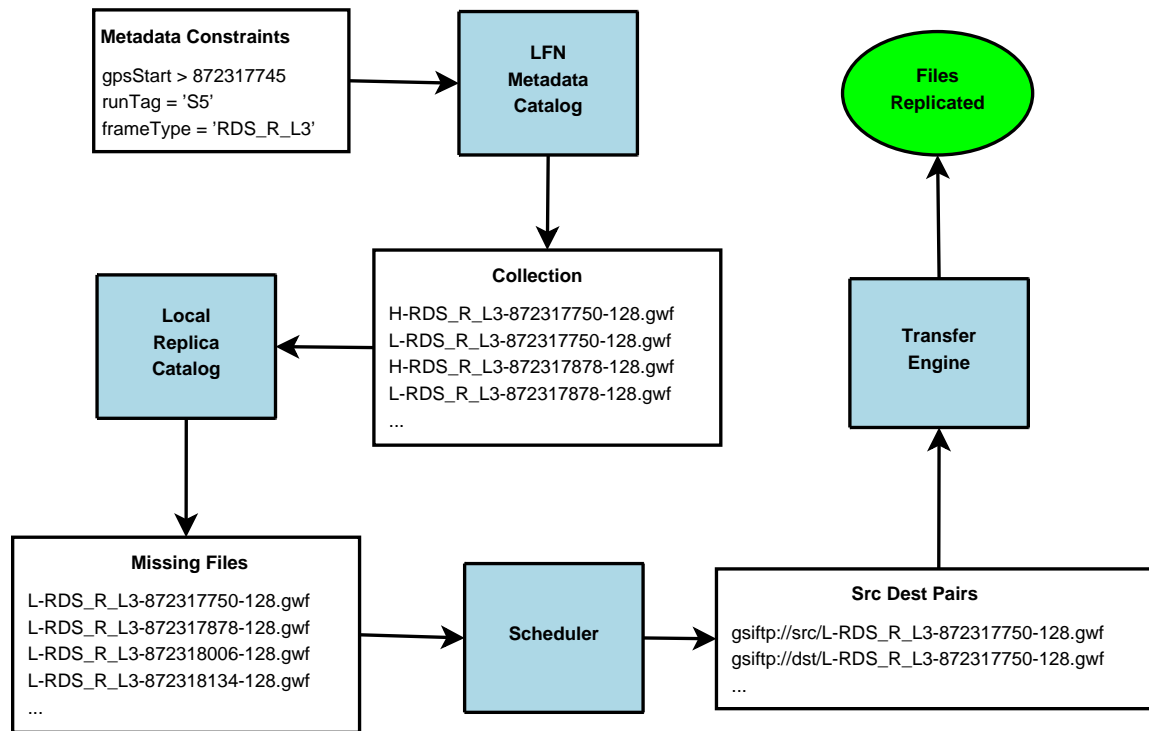


Figure 1.1: The basic LDR design. Metadata constraints define which sets or collections of files are to be replicated. The constraints are reconciled against a metadata catalog to create the list of LFNs in a collection. The list of LFNs in the collection is reconciled against a local replica catalog to determine which of the LFNs are missing and need to be replicated. The list of missing files is acted upon by a scheduler which locates sources for the missing LFNs and schedules them for replication by a transfer engine.

2. The collection of interest is reconciled against a local replica catalog (LRC). The LRC is the authority on which files or LFNs in a collection already exist at the local site. The collection of interest is reconciled with the LRC to generate an ordered list of LFNs that are missing at the local site and which need to be replicated.
3. The list of missing LFNs is input to a scheduler which locates PFNs or URLs available from remote sites from which the LFNs may be obtained. The scheduler output is a list of source-destination pairs which can be input to a transfer engine or service that accomplishes the actual replication of the LFN.
4. The transfer engine or service acts on the source-destination pair and replicates the missing files or LFNs. After a successful replication an entry is made in the LRC so that the LFN is recorded as being locally replicated and no longer in need of replication.

1.3.3 LDR components

The major LDR components are depicted in figure 1.2. Below we briefly describe each component and its role in replicating data files. More details about each component and its configuration is found in later chapters.

The major components of LDR include:

- *MySQL*: The MySQL relational database is used to hold the tables that make up the LDR metadata catalog, the local replica catalog, as well as a number of auxiliary tables used for queuing transfers. A LDR administrator does not directly interact with the MySQL server except for trouble shooting. Note that all communications with the server happen between services or daemons running locally on the same hardware on which the server runs so that MySQL is usually configured to only allow communication over a UNIX socket and not over a network connection.
- *Globus RLS*: Each site runs a Globus Replica Location Service (RLS) instance as part of LDR. RLS provides a local replica catalog (LRC) and a replica location index (RLI). The RLI allows mappings from LFNs to remote LRCs so that to locate a particular LFN for replication one need not query every

remote LRC. LDR administrators may sometimes use client tools to directly query the RLS server when trouble shooting.

- *unixODBC and MyODBC*: These two components enable the Globus RLS server to communicate with the MySQL database server. Administrators never need to interact with either component.
- *LDRMetadataServer*: The LDRMetadataServer enables sites where data is published to expose the metadata catalog so that other sites may receive updates of metadata as new LFNs are published. Sites that do not publish data may also run a LDRMetadataServer in order to relay metadata updates to other remote sites but in practice this is not usually done. The LDRMetadataServer is a web service running under the Apache httpd web server.
- *LDRMetadataUpdate*: The LDRMetadataUpdate daemon queries remote LDRMetadataServer instances to acquire metadata catalog information about LFNs. A local LDRMetadataUpdate daemon may query multiple remote server instances in order to gather updates about LFNs being published at multiple remote sites. For example a LDRMetadataUpdate daemon running at Caltech queries the metadata servers at both Hanford and Livingston in order to gather metadata about all LIGO data being published.
- *LDRTransfer*: The LDRTransfer daemon queries the local metadata catalog and both local and remote RLS servers in order to determine which files need to be replicated and from where they can be replicated. LDRTransfer queues up lists of source and destination pairs or URLs for files to be replicated to the local site. and then orchestrates the actual replication or transfer of files from remote sites to the local site. LDRTransfer can be configured to replicate from multiple sites simultaneously using a number of different protocols and configurations for the underlying protocols. The protocol most often used is GridFTP and files are transferred from a remote GridFTP server to a “local” GridFTP server using a so-called “third-party” or server-to-server transfer. The LDRTransfer daemon is a customized GridFTP client built on top of the GridFTP client API and libraries.
- *LDRDataFindServer*: The LDRDataFindServer enables clients to submit queries containing metadata constraints such as GPS time ranges and a frame type and in return receive a list of URLs for the location of files at the local site. The LDRDataFindServer is a web service running under the Apache httpd web server.
- *MySQLdb*: This component is a Python module (built on top of libraries coded in C) that enables the LDR daemons listed above to interact with the MySQL server. Administrators do not need to interact with or configure this component.
- *GridFTP server*: The data files residing on a file system are exposed via a GridFTP server. The GridFTP server enables multiple concurrent file transfers with each one using multiple data streams and tunable TCP windows for high performance data transfers. Striped server configurations are also possible for systems with data on high performance file systems (not NFS) available to multiple nodes.

1.3.4 Integration with local storage

The integration of LDR with the local storage (usually NFS mounted file systems though other storage systems can be supported) is accomplished using a *local storage module*. The local storage module is where local administrators can overload some pre-defined functions to tightly integrate LDR with the local site’s preferences for how files are to be distributed within the local storage, and how files are to be post-processed after replication.

Details on writing a local storage module follow in later chapters. Each LDR instance requires a customized local storage module for any production quality replication of data.

1.3.5 Anatomy of a file replication

Figure 1.3 depicts the anatomy of the replication of a LFN or file. Because of the distributed and networked nature of a multiple-site LDR deployment many scenarios are possible and figure 1.3 only depicts one possible scenario. Note that not every LDR component at each site is shown for clarity.

In figure 1.3 site A is a source site for a set of LFNs, for example the LIGO Hanford site. Site B is a site that wishes to replicate a collection that includes LFNs published at site A, for example the UWM site. Site C is a site that already has replicated some of the data files such as PSU.

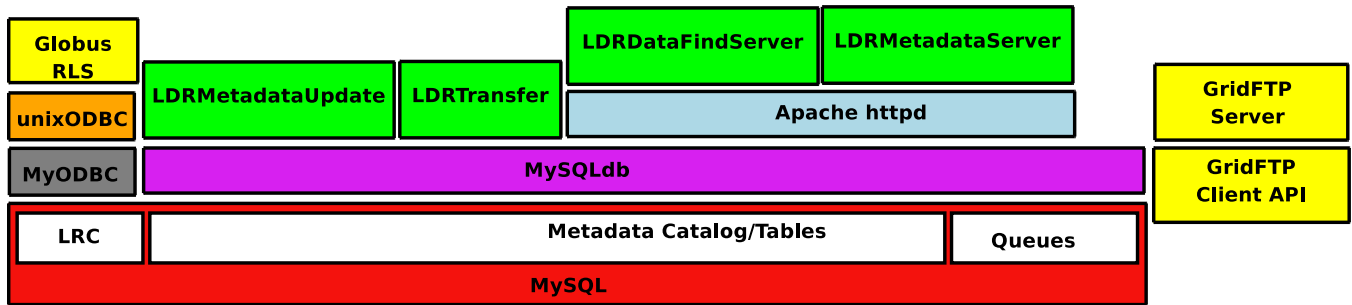


Figure 1.2: The component architecture of an LDR deployment. Items colored in green are written in Python and constitute the “glue” that coordinates the underlying components and technologies. Items not colored green are standard tools and components deployed as part of an LDR deployment but which are not customized in any special way for use with LDR (though they are configured to operate with LDR). Not pictured is the LDRMaster daemon whose sole purpose is to start the other LDR daemons and watch over them, restarting a daemon if it should die for some reason.

The replication of any particular file, say for the LFN [H-H2.RDS_C03_L2-843149014-128.gwf](#), proceeds *schematically* in the following steps as depicted in figure 1.3 (we say schematically since large sets of files are considered and replicated in bulk and many of the steps proceed asynchronously):

1. A set of “publishing” scripts at site A where data is generated creates entries in the local metadata catalog for the file [H-H2.RDS_C03_L2-843149014-128.gwf](#) that include `site = H`, `frame type = H2.RDS_C03_L2`, `GPS start time = 843149014`, `interferometer = H2`, as well as file details including the file size and md5 checksum. The same publishing scripts also publish at least one URL into the local replica catalog via Globus RLS so that the file location is recorded.
2. The local LDRMetadataUpdate daemon at site B periodically queries the LDRMetadataServer at site A and receives an update for the publication of metadata about the LFN [H-H2.RDS_C03_L2-843149014-128.gwf](#). After this step the local metadata catalog at site B has all the metadata for the LFN.
3. The LDRSchedule daemon at site B periodically checks the list of collections defined for replication. If a defined collection includes metadata constraints that include the file [H-H2.RDS_C03_L2-843149014-128.gwf](#) then LDRSchedule determines if the file is already replicated by consulting the local replica catalog. If the file is not already replicated LDRSchedule consults both the local RLS server and remote RLS servers to determine source URLs or PFNs for the file. If the file is available at site C then LDRSchedule will retrieve a URL from the RLS at site C that may look like [gsiftp://sitec/data/H-H2.RDS_C03_L2-843149014-128.gwf](#).
4. This step is depicted using dashed lines to indicate that the deployed RLS servers update each other asynchronously about the contents of their local replica catalogs. Because each local RLS server learns about the contents of remote servers it is able to direct the LDRSchedule daemon to appropriate RLS servers to obtain specific URLs to be used as source URLs for transfers.
5. LDRSchedule also queries the local storage to obtain a destination URL for the file and then it queues the source-destination pair of URLs for transfer. The pair might look like

```
gsiftp://sitec/data/H-H2\_RDS\_C03\_L2-843149014-128.gwf ->
gsiftp://siteb/LH0/strain/H-H2\_RDS\_C03\_L2-843149014-128.gwf
```

6. The LDRTransfer daemon retrieves the source-destination pair of URLs and prepares to orchestrate the transfer.
7. Acting as a client the LDRTransfer daemon requests that the file be transferred or replicated from site C to site B via the GridFTP servers at each site.
8. The GridFTP servers use multiple data streams to replicate the file from site C to site B. Not shown in the figure is that after a successful transfer (usually determined by performing a md5 checksum on the replicated file at site B) the LDRTransfer daemon records the new PFN or URL for the replicated file in the local replica catalog at site B so that the file is officially recorded as being replicated.

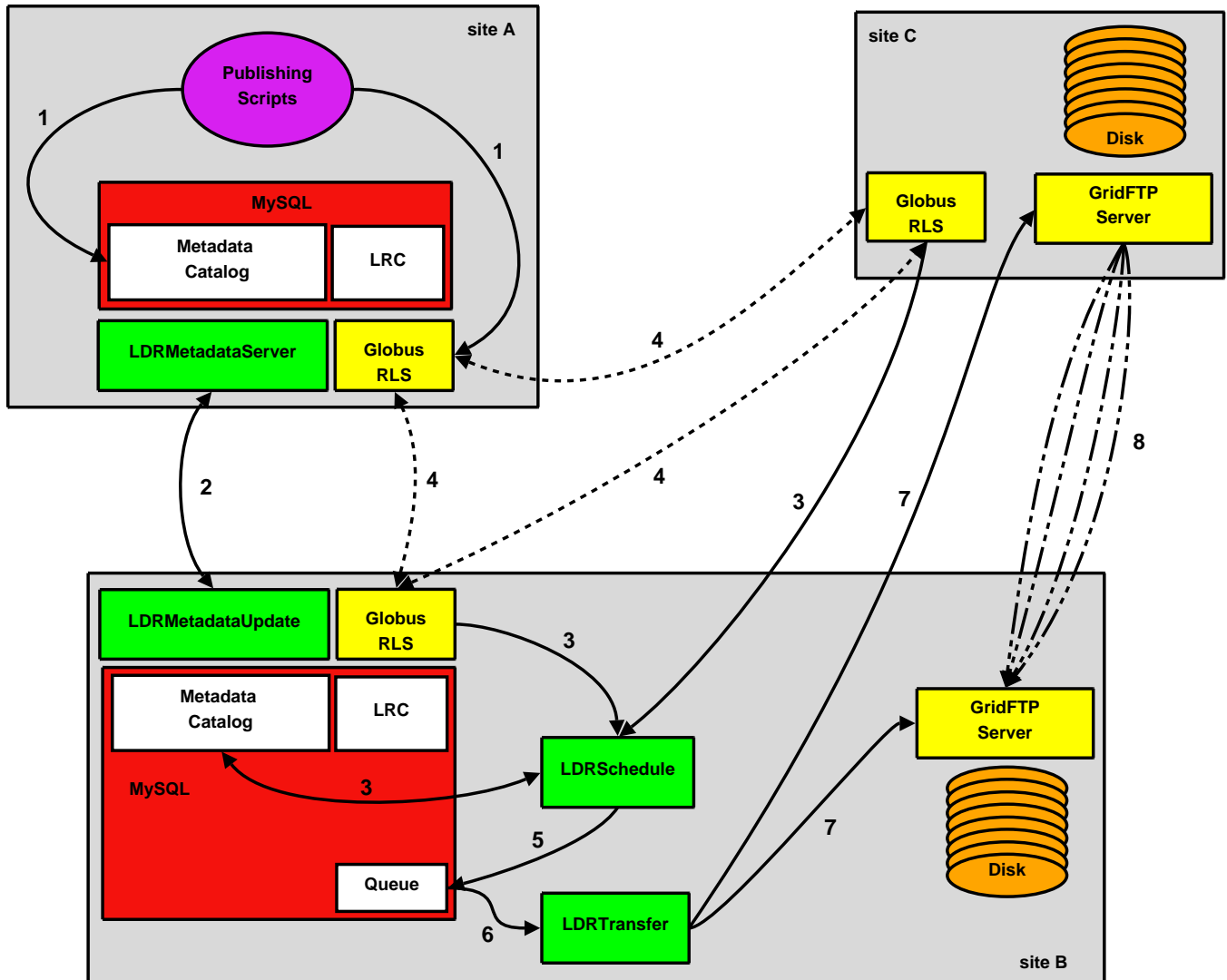


Figure 1.3: Anatomy of a file replication by LDR. Not all LDR components at each site are depicted for clarity: (1) Metadata details about the file are published into the metadata catalog and a URL is recorded in the local replica catalog at site A. (2) The metadata details about the file are pulled from site A to site B when the LDRMetadataUpdate daemon queries the LDRMetadataServer. (3) LDRSchedule queries the local metadata catalog and the local replica catalog at site B to determine lists of files needing replication. (4) The Globus RLS servers asynchronously update each other with the contents of their local replica catalogs to create a distributed cache of file locations. (5) LDRSchedule chooses a source site and URL for the file and queues up a replication. (6) LDRTransfer pops the source-destination pair of URLs from the queue and prepares to orchestrate the transfer. (7) Acting as a client LDRTransfer requests the GridFTP servers at sites C and B to replicate the file. (8) The file is transferred from site C to site B using multiple data streams.

1.4 Changes

1.4.1 Changes from LDR 0.9.3 to LDR 0.9.4

- Fixed bug 24, a race condition that could cause some metadata to no be replicated.
- Fixed a bug in the LDRBundle tool being used at Hannover to rebundle the LIGO strain data into larger frame files. The files, once transferred to Cascina, are now properly deleted.

1.4.2 Changes from LDR 0.9.2 to LDR 0.9.3

- Fixed bug 16 that could cause extraneous error messages regarding a non-existent site.
- Fixed bug 19 to add a missing value to the enum for the status column in the site table.
- Fixed bug 20 so that the publishing script for the LIGO Lab automatically handles failed publication of mappings into RLS if RLS is down.
- Fixed bug 21 that prevented sites marked as non-viable but later marked as viable from being used again.
- Changed RLS deployment to run under the control of monit to restart it if the server crashes.

1.4.3 Changes from LDR 0.9.1 to LDR 0.9.2

- Fixed a race condition that would allow a file to be properly replicated but not published into RLS. The file would not be transferred again because it stayed in the queue in MySQL marked as “publishing in progress”.
- Fixed the log message for when LDRTransfer fails to connect to a remote RLS.
- The LDRPublishLIGOLab script now writes the process ID number (PID) into the PID file instead of the heartbeat time.
- Added logging to indicate the LDRTransfer threads status.
- Fixed a bug preventing the proper identification and handling of transfers that hang and do not update their heartbeat.

1.4.4 Changes from LDR 0.9.0 to LDR 0.9.1

- Fixed bug preventing some methods in LDRDataFindServer from working with authorization type `virtual_host`.
- Removed extraneous `LOCK TABLE` instructions that were affecting throughput during high load with LDRTransfer.
- Fixed bug that import of `LDR.LDRStorage` was left out of `LDRStorageTemplate.py`.

1.4.5 Changes from LDR 0.8.0 to LDR 0.9.0

The following is a brief overview of the changes from LDR version 0.8.0 to version 0.9.0:

- Change to using native packaging for RedHat platforms (yum and RPM) and Debian platforms (.deb packages).
- Change to using system installations of MySQL, unixODBC, MySQL Connector, and Python.
- Full 64 bit support on all platforms.
- A single configuration file for all of the LDR components, sites, and collections.
- Upgrade to Globus Toolkit 4.2.1.
- LDRSchedule and LDRTransfer are a single executable now.

- The LDRDataFindServer and LDRMetadataServer are now web services running under Apache httpd.
- The local storage module now expects lists or dictionaries as input and output instead of single files or URLs in order to accomodate faster bulk processing.
- The LDR transfer agents automatically use temporary file names when transferring a file and then automatically verify the checksum of the file before the file is renamed and published. Failed transfers and files with bad checksums are automatically deleted.
- The automatic checksums are done in parallel for higher throughput.
- Scheduling and replication of collections is now done in parallel with a separate thread and database connection for each connection and a separate thread to manage data transfer clients for each connection (individual data transfers for each collection might themselves be multi-threaded, but they are managed by a single thread per collection).
- New transfer methods that support multiple concurrent transfers including third-party transfers.
- More efficient scheduling that decreases the time for scheduling a collection by a factor of two for most collections.
- The LDRmetadata and LDRqueue databases have been combined into one database with multiple tables. The database is named “LDR”. Some table definitions have changed but not the main table holding the file metadata.

1.4.6 Changes from LDR 0.7.0 to LDR 0.8.0

The following is a brief overview of the changes from LDR version 0.7.0 to version 0.8.0:

- Support and upgrade to MySQL 5, specifically 5.0.22. We have also switched from IODBC to unixODBC (version 2.2.11) to accomodate this change.
- Upgrade to Globus Toolkit 4.0.2, Python 2.4.3, MySQL Connector/ODBC (MYODBC) 3.51.12, MySQL-python 1.2.1_p2, and pyGlobus 4.0.1-1.15. rlsClient rebuilt with swig 1.3.29.
- LDRSchedule reloads the storage module every loop allowing for dynamic editing of the storage module.
- Sorting of results for dataFind clients from LDRdataFindServer fixed (was previously returned two sorted groups which were not sorted as a whole).
- LDRTransfer now ensures an LFN is completely unpublished if a transfer fails, allowing for re-replication.
- Improved logging in LDRMaster and LDRMetadataUpdate.
- A few smaller bug fixes and API updates.

1.4.7 Changes from LDR 0.6.x to LDR 0.7.0

The following is a brief overview of changes from the LDR 0.6.x release:

- MySQL is being upgraded from binaries of version 4.0.11 that we compiled to standard binaries of version 4.0.25 provided by the MySQL development team.
- The LDR metadata tables in MySQL now use the InnoDB engine for better performance. Our testing has shown that the InnoDB engine allows more memory to be used for keeping table indices in memory, which leads to overall performance increases.
- The LDR metadata table schema has been changed so that the resulting tables are smaller and can be handled more efficiently by MySQL.
- We have upgraded to the new globus-gridftp-server from the Globus Toolkit 4.0.1. The new server has been built “from the ground up” by the Globus team and replaces the patched wuftp server in the Globus Toolkit 3.2.1 and previous preleases.

- We have upgraded to the most recent RLS in the Globus Toolkit 4.0.1. This release of RLS is only change from the RLS used in LDR 0.6.x.
- We have upgraded to Python 2.4.1, IODBC 3.51.2, and Python-MySQL 1.2.
- New configuration options for LDRdataFindServer allow administrators to filter which LFNs and PFNs are returned to clients.
- LDRVerify has been added as a daemon that can be run in order to help keep the state of storage and the RLS catalog in sync (among other things). This is a first release of LDRVerify and we invite feedback on how LDRVerify should evolve.
- Globus GRAM is no longer included and so neither is the Globus gatekeeper. The approach in the future for dealing with lots of transfers of small files will use a customized GridFTP server module and associated LDR custom client, but GRAM will not be a part of that solution.

Chapter 2

Installing LDR

2.1 Supported Platforms

LDR version 0.9.x is supported on the following platforms:

- CentOS 5 on x86_64
- Scientific Linux 5 on x86_64
- Debian 5 (Lenny) on x86_64

In addition the [globus-gridftp-server](#) is supported on Solaris 10 x86_64 to support the LIGO Laboratory infrastructure.

Support for 32 bit versions of those platforms is easily arranged. Please contact send email to ldr-lsc@gravity.phys.uwm.edu.

2.2 Hardware requirements

Since a production installation of LDR usually includes a MySQL database server as well as a GridFTP server for serving (and retrieving) data, we recommend that the hardware meet the following requirements:

- server class CPU (clock speed greater than 2 GHz)
- 2 GB of RAM
- fast disk partition
- 20 GB of free space on the fast partition
- gigabit ethernet connection to a fast network

MySQL performance is greatly enhanced when the database tables are stored on a fast filesystem, and when significant amounts of memory are available for caching frequently requested results from the tables. We have not specified what we mean by a fast disk partition—we will simply say that the faster the better the performance.

Performance is also enhanced when we trade disk space for speed. That is, if we do a lot of indexing for the tables in the MySQL databases then the overall performance is dramatically increased.

So yes, we really do expect 20 GB of free space to be available just for the MySQL tables. If your data grid will use LDR to manage 10^7 files or more then you can definitely expect to use a large fraction of that 50 GB.

Disk space is cheap. We recommend you leave room to grow.

2.3 Required software packages

LDR version 0.9.x requires that the following software packages be available on the machine on which LDR is to run:

- MySQL version 5.0.32 or greater in the 5.0.x series
- Python version 2.4.3 or greater
- MySQL-Python 1.2.2 or greater
- unixODBC 2.2.11 or greater
- MyODBC (MySQL Connector/ODBC) 3.51.x or greater
- Globus Toolkit version 4.2.1 (only some C components)
- Apache httpd 2.2 or greater
- A number of other python modules

On CentOS, Scientific Linux, and Debian platforms the dependencies on those packages are automatically handled during the installation.

2.4 Upgrading from LDR 0.8.x

Due to the change to using native packaging for most platforms there is no simple way to upgrade. We recommend that you make LDR 0.9.0 a fresh installation and restore the state of your current MySQL tables and LDR configuration files.

2.4.1 Preparation

Before beginning the upgrade from LDR 0.8.0 to LDR 0.9.0 please

1. Shutdown LDR 0.8.0 by doing

```
kill -SIGTERM `cat $LDR_LOCATION/ldr/var/LDRMaster.pid`
```

or by just sending a SIGTERM signal to the LDRMaster process. Please wait 60 seconds and then use `ps` to verify that all the LDR daemons have exited.

2. Shutdown Globus RLS by using `ps` to find the RLS server process ID and then send it a SIGTERM signal.
3. Do NOT shutdown the MySQL server at this time.

2.4.2 Dumping database contents

You need to dump the state of your LDR 0.8.0 databases used by both the LDR metadata catalog and Globus RLS.

Dumping the Globus RLS databases

This is the fastest way to dump the Globus RLS databases and prepare them for later uploading into the new MySQL server:

1. Prepare your environment as usual by doing

```
source $LDR_LOCATION/setup.sh
```

2. Change directories to a filesystem where you will have a fair amount of available space to dump the databases.

3. Create the directory `MySQL_dump_RLS_lrc1000` and change directories into it:

```
mkdir MySQL_dump_RLS_lrc1000
cd MySQL_dump_RLS_lrc1000
```

4. Dump the Globus RLS lrc1000 database using mysqldump:

```
mysqldump --quick --tab=./ --user=root --password=PASSWORD lrc1000
```

5. Change directories to go up one level then create the directory `MySQL_dump_RLS_rli1000` and change directories into it:

```
cd ../
mkdir MySQL_dump_RLS_rli1000
cd MySQL_dump_RLS_rli1000
```

6. Dump the Globus RLS rli1000 database using mysqldump:

```
mysqldump --quick --tab=./ --user=root --password=PASSWORD rli1000
```

Dumping the LDR metadata database

1. Change directories to a filesystem where you will have a fair amount of available space to dump the databases.
2. Create the directory `MySQL_dump_LDRmetadata` and change directories into it:

```
mkdir MySQL_dump_LDRmetadata
cd MySQL_dump_LDRmetadata
```

3. Dump the LDRmetadata database using mysqldump:

```
mysqldump --quick --tab=./ --user=root --password=PASSWORD LDRmetadata
```

If disk space is a concern the files dumped into this directory can be tarred and compressed and moved away for sakekeeping, though you will need to keep the `pset` table details available for importing after the upgrade is complete.

2.4.3 Shutting down MySQL

Once all the tables from the MySQL server have been dumped please shut down the server by doing

```
mysqladmin --user=root --password=PASSWORD shutdown
```

2.5 Installing required software

Please skip to the appropriate section below for your operating system.

2.5.1 CentOS

LDR for CentOS is packaged using RPMs. The easiest way to install LDR on CentOS is to configure Yum to access the EPEL repository and the LIGO LSCSOFT repository.

Before configuring Yum and installing LDR using the LSCSOFT repository please note that during the installation a *ldr* user is created automatically if one does not already exist on your system.

To configure your CentOS system and install LDR using Yum please follow these instructions:

1. Login as the root user.
2. Configure Yum to use the EPEL repository. See <http://fedoraproject.org/wiki/EPEL> for details.
3. Create the file `/etc/yum.repos.d/lscsoft.repo` with the following contents:

```
[lscsoft]
name=LSC Data Analysis Software
baseurl=http://www.lsc-group.phys.uwm.edu/daswg/download/software/centos/5/$basearch
enabled=1
gpgcheck=0
```

4. Install LDR by doing

```
yum groupinstall ldr
```

Yum will install all the necessary RPMs and resolve any dependencies not satisfied on your system. After the installation you still need to restore the state of your MySQL server (if you are upgrading from an earlier LDR version) and properly configure LDR and other components.

2.5.2 Scientific Linux

The instructions for installing LDR on Scientific Linux are the same as the instructions for CentOS as detailed above.

2.5.3 Debian

LDR for Debian is packaged using standard Debian “debs”. The easiest way to install LDR on Debian is to add the LSCSOFT Debian repository to your `/etc/apt/sources.list` file by adding the line

```
deb http://www.lsc-group.phys.uwm.edu/daswg/download/software/debian/ lenny contrib
```

Note that aptitude will complain when installing packages from an “untrusted” source. In order to allow aptitude to verify the authenticity of these packages, you need to add the keyring of the LIGO LSCSoft repository. The simplest way to do this is to simply install the package `lscsoft-archive-keyring` via aptitude (insisting only once that you trust this repository):

```
aptitude install lscsoft-archive-keyring
```

Then run

```
aptitude update
```

followed by

```
aptitude install python-ldr
```

Chapter 3

Configuring MySQL, RLS, GridFTP, Apache, and LDR

Before MySQL, RLS, GridFTP, Apache, and LDR itself can be started there is a fair amount of configuration that must be done. If you are upgrading then you must also restore the state of your MySQL that you saved from your previous deployment. Additionally, you must obtain X.509 digital credentials for the LDR components to enable proper authentication and authorization. The details below will help you configure MySQL, RLS, and LDR, and obtain and deploy the necessary credentials.

3.1 Configuring, initializing, and restoring state for MySQL

The MySQL server and its database contents are the foundation for LDR. Before LDR can be configured and used to replicate data you must configure, initialize, and if upgrading restore the state of the MySQL server.

3.1.1 Securing MySQL

LDR does not require any TCP connections to the MySQL server `mysqld`. We strongly suggest you edit the MySQL configuration file `/etc/my.cnf` on CentOS and SL or `/etc/mysql/my.cnf` on Debian and add the option

```
skip-networking
```

in the `[mysqld]` section.

3.1.2 Configuring the location of MySQL table data files

The default location for MySQL to store table data is in the directory `/var/lib/mysql`. If this location is not suitable for your deployment because either there is not enough free disk space (20 GB) or that is not a fast and reliable partition then you must change the location for the table data.

To change the location where MySQL stores its table data:

1. Make sure that the MySQL server is not running by doing `service stop mysqld` on CentOS and SL or `/etc/init.d/mysql stop` on Debian.
2. Create the directory on a fast and reliable partition and make it owned by the `mysql` user with group `mysql` and with permissions `0755`. For example

```
mkdir /opt/mysqldata1
chown mysql /opt/mysqldata1
chgrp mysql /opt/mysqldata1
chmod 0755 /opt/mysqldata1
```

3. Edit `/etc/my.cnf` (CentOS and SL) or `/etc/mysql/my.cnf` (Debian) and in the `[mysqld]` section edit or add *both* the options `datadir` and `innodb_data_home_dir` to point to the directory you just created.

4. Continue to edit `my.cnf` and in the `[mysqld]` section edit or add the option `innodb.data.file_path` as shown:

```
innodb_data_file_path = ibdata1:10G:autoextend
```

3.1.3 Configuring the MySQL InnoDB engine for maximum performance

LDR version 0.9.x uses the InnoDB storage engine from MySQL. The performance of the InnoDB engine (and thus LDR) depends greatly on how much of the system's memory the InnoDB engine is allowed to use. If you only allow the InnoDB engine to use a small amount of memory LDR will still work but the performance will degrade significantly.

We strongly recommend that you configure the MySQL InnoDB engine to use 80% of the physical system memory for the best performance. Further we strongly recommend that your system have at a minimum two (2) gigabytes of physical memory. LDR will work with smaller systems but the performance will be degraded.

To configure the MySQL InnoDB engine to use 80% of the physical memory edit the file `my.cnf` and in the `[mysqld]` section add or change the line

```
innodb_buffer_pool_size = 256M
```

from 256 megabytes of memory to whatever is 80% of your system's memory. For example if your system has 2 gigabytes of RAM then you would set

```
innodb_buffer_pool_size = 1638M
```

Do not set `innodb_buffer_pool_size` to be more than 80% of the physical memory or you will risk your system becoming unstable. See the MySQL documentation for details.

3.1.4 Initializing MySQL for LDR

Initializing MySQL for LDR includes starting the server so that the InnoDB engine can create the 10 GB database file, creating a password for the MySQL root user, creating access rights for the ldr user, and creating the necessary databases and tables.

Before starting the server again Debian users must first run

```
mysql_install_db --user=mysql
```

in order to re-initialize some tables that MySQL needs.

To start the MySQL server please run

1. On CentOS and SL: `service mysqld start`
2. On Debian: `/etc/init.d/mysql start`

After the server starts it will immediately begin to create the 10 GB file for the InnoDB engine to use to store the LDR and RLS tables. Depending on the speed of your system this might take some time, and the init script might report failure.

If the init script reports `[FAILED]` but the `mysql` process is still running then there is no problem and you just need to wait until you see the 10GB file created. After you see the 10GB file created continue with the next steps below. This is only an issue during the first startup.

You should create a password for the root user of this instance of the MySQL server (note that Debian users will most likely have already set the password for the MySQL root password during the package installation). To create a password for the MySQL root user:

```
mysqladmin -u root password YOUR_MYSQL_ROOT_PASSWORD
```

Next you need to create access rights to the LDR and RLS tables in the MySQL database for the ldr user:

```
mysql --user=root --password=YOUR_MYSQL_ROOT_PASSWORD
USE mysql;
GRANT ALL ON lrc1000.* TO ldr@localhost IDENTIFIED BY 'SOME_PASSWORD';
GRANT ALL ON rli1000.* TO ldr@localhost IDENTIFIED BY 'SOME_PASSWORD';
GRANT ALL ON LDR.* TO ldr@localhost IDENTIFIED BY 'SOME_PASSWORD';
QUIT;
```

Here `SOME_PASSWORD` is the password you want to set for the ldr user. Note that the quotes in the grant lines above are necessary. Please be sure to note the password you choose!

Lastly, you need to create the necessary tables in the MySQL database for use with both RLS and LDR. To create the tables use the `mysql` client to initialize the RLS and LDR databases and tables:

```
mysql --user=ldr --password=PASSWORD <
/opt/ldr-globus/setup/globus/globus-rls-lrc-mysql.sql
mysql --user=ldr --password=PASSWORD <
/opt/ldr-globus/setup/globus/globus-rls-rli-mysql.sql
mysql --user=ldr --password=PASSWORD <
/etc/LDR/ldr.sql
```

At this point your MySQL is properly initialized for use with LDR. If you are upgrading from a previous LDR deployment please see the instructions in the next section for details on how to restore the state of your MySQL databases. If this is a new LDR deployment please skip ahead for details on configuring RLS.

3.1.5 Restoring MySQL state from LDR 0.8.x

If you are upgrading LDR from a 0.8.x installation and have saved the state of your LDR 0.8.x databases as detailed above then please follow the instructions below to repopulate the Globus RLS and LDRmetadata database tables.

If you are not upgrading then please skip ahead to the next section.

Repopulating the Globus RLS database tables

The Globus RLS and LDR database tables can be easily created and repopulated with the state of your LDR 0.8.x installation by following these instructions:

1. Change directories to where you dumped the lrc1000 database tables:

```
cd MySQL_dump_RLS_lrc1000
```

Note that the entire directory path to the dump files must be readable and reachable by the mysql user since that is the user that runs the MySQL server.

2. Import the RLS lrc1000 database entries:

```
mysqlimport --user=root --password=PASSWORD lrc1000 'pwd'/*.txt
```

Note that the `'pwd'/*.txt` syntax is required because the `mysqlimport` client requires full path names. Also note that this command may take quite some time if the RLS tables contained a large number of mappings.

3. Change directories to where you dumped the rli1000 database tables:

```
cd MySQL_dump_RLS_rli1000
```

4. Import the RLS rli1000 database entries:

```
mysqlimport --user=root --password=PASSWORD rli1000 'pwd'/*.txt
```

Since most RLS servers use Bloom filter updates rather than full list updates the rli1000 tables are usually small and this command should complete quickly.

5. Change directories to where you dumped the LDR metadata database tables.

```
cd MySQL_dump_LDRmetadata
```

Note that only *two* (2) tables will be imported into the new deployment.

6. Import the `LSCmetadata` table entries:

```
mysqlimport --user=root --password=PASSWORD LDR 'pwd'/LSCmetadata.txt
```

7. Import the `pset` table entries by specifying the particular columns to use:

```
mysqlimport --user=root --password=PASSWORD LDR -c oid,name,ts 'pwd'/pset.txt
```

The RLS and LDR database tables should now be repopulated.

At this point MySQL should be completely configured for use with RLS and LDR. You will want to leave the MySQL server running so that later you may configure and initialize Globus RLS.

3.2 Obtaining Credentials or X.509 Certificates

The Globus RLS and GridFTP servers require X.509 digital certificates for mutual authentication. In addition the LDR clients and servers expect to have X.509 digital credentials.

There are two types of certificates or credentials necessary for a LDR installation to work properly:

- *host certificate* – this is a certificate used by a service running on a machine to identify itself to any clients that might want to authenticate and connect to the service. This type of mutual authentication is important so that clients can be reasonably sure that they are connecting to the correct host.

The Globus RLS and GridFTP servers each require a host certificate to run as a service. The GridFTP server usually runs as `root` and so will require a single copy of the host certificate and its associated private key on the LDR machine where both files are owned by `root` and in group `root`. More details on where to locate the host certificate and key follow later. The RLS server usually runs as `ldr` and so will require a *copy* of the host certificate and private key owned by `ldr` and in group `ldr`.

The Apache httpd server that hosts the `LDRMetadataServer` and `LDRdataFindServer` web services also requires a host certificate for secure communication via the `https` protocol. Since some users may wish to browse the `LDRDataFindServer` using an ordinary web browser you may wish to obtain an X.509 certificate for the web server from a certificate authority (CA) that is by default recognized by most standard web browsers. Such a certificate, however, is not required and there is no technical reason why you cannot use the same host certificate that is used by the GridFTP and RLS servers with the Apache httpd.

- *datarobot certificate* – this is a certificate used by the `ldr` user when LDR client programs such as `LDRMetadataUpdate`, and `LDRTransfer` need to connect to services such as RLS, GridFTP, or the LDR web services mentioned above. This *datarobot* certificate is often a user or non-host service certificate. It is preferable that this certificate not be a person's individual identity X.509 certificate, but if a certificate authority (CA) will not issue a certificate to a "robot" user then an identity certificate is acceptable.

If you have existing host and *datarobot* certificates and the corresponding private keys then you may skip ahead to the section on locating the certificate files.

3.2.1 Requesting host and *datarobot* certificates

Please follow the instructions for your organization for obtaining host and *datarobot* certificates from a trusted CA.

3.2.2 Locating the host and datarobot certificates

While all of the tools are configurable so that the host and datarobot certificates can be located most anywhere on the filesystem, the default installation and configuration of LDR assumes the following locations and naming conventions (you may also use symlinks to link to the following locations):

- Move the files `hostcert.pem` and `hostkey.pem` into the directory `$/etc/ldr-globus` and make sure that both are owned by `root` and in group `root`. This is necessary for the `globus-gridftp-server`.
Also make sure that the file `hostcert.pem` has permissions `0644` and `hostkey.pem` has permissions `0400`.
- Make a *copy* of the `hostcert.pem` and `hostkey.pem` files named `rlscert.pem` and `rlskey.pem` and also locate these in the directory `$/etc/ldr-globus`. These files should be owned by the `ldr` user and should have the same permissions as the host files above. These files will be used by RLS to authenticate its service.
- The convention for CentOS and Scientific Linux is for the certificate and key files that the Apache `httpd` server will use to be located at

```
/etc/pki/tls/certs/localhost.crt  
/etc/pki/tls/private/localhost.key
```

On Debian Lenny the convention is

```
/etc/ssl/certs/localhost.crt  
/etc/ssl/private/localhost.key
```

If you are using the same certificate and key that you are using for the `globus-gridftp-server` and `globus-rls-server` then make a *copy* of the `hostcert.pem` and `hostkey.pem` files named `localhost.crt` and `localhost.key` to be used by the Apache `httpd` server and place them at the locations show above.

For all platforms the files should be owned by `root` and in group `root`. The certificate file should have permissions `0644` and the key file should have permissions `0400`.

You also need to provide a file for the Apache `httpd` that contains the CA certificate(s) for the CA that signed the certificate being used by the server. *If the CA is not a root CA then all certificates in the chain need to be copied into the file.* The file should be in PEM format.

Administrators using certificates signed by the DOEGrids CA can create this file easily:

```
cd /etc/grid-security/certificates  
cat 1c3f2ca8.0 d1b603c3.0 > /tmp/server-chain.crt
```

This file then needs to be located where the Apache `httpd` server can find it. On CentOS and SL the default location is

```
/etc/pki/tls/certs/server-chain.crt
```

On Debian Lenny the default location is

```
/etc/apache2/ssl.crt/server-ca.crt
```

- Move the files `datarobotcert.pem` and `datarobotkey.pem` to the directory `$/etc/LDR`. These files should be owned by the `ldr` user and should have the same permissions as the host files above. These files will be used by the LDR clients to authenticate to remote services.

3.2.3 Working with the datarobot certificate

Often you as an administrator will want to use the datarobot certificate as credentials for testing and debugging your LDR installation. There are a couple of ways to make this convenient for you.

First, you can edit the “dot” files for the LDR user and define the environment variables `X509_USER_CERT` and `X509_USER_KEY` to point to the datarobot certificate and key respectively (note that by default the home directory for the `ldr` user is `/opt/ldr-globus` and so you will need to create the “dot” files). With these environment variables set, and after doing

```
export GLOBUS_LOCATION=/opt/ldr-globus
source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

you will have the datarobot credentials available to you when using command such as `globus-rls-admin`. Note that in this scenario a “proxy” certificate is not necessary.

Second, you can alternatively create a cron job for the `ldr` user to run `grid-proxy-init` on a regular basis to create a proxy certificate using the datarobot credentials. Please run `grid-proxy-init -help` to see what options are necessary so that the datarobot certificate and key can be located when the proxy is created.

3.3 Configuring and Initializing Globus RLS for LDR

Configuring and initializing Globus RLS involves 5 steps:

- configure `monit` to help manage `globus-rls-server`
- edit the RLS configuration file `/etc/ldr-globus/globus-rls-server.conf`
- edit the ODBC configuration file `/etc/odbc.ini`
- testing the server
- final configuration using `globus-rls-admin`

3.3.1 Edit the `monit` configuration file

Edit the `monit` configuration file `/etc/monit/monitrc` on Debian Lenny or `/etc/monit.conf` on CentOS 5 and SL and uncomment or add the line

```
include /etc/monit.d/*
```

Debian users also need to edit `/etc/default/monit` and set

```
startup=1
```

You also need to make sure that `monit` will restart after a machine reboot. Please use your favorite system tool to make sure that `monit` will run at the appropriate runlevel for your system at boot time.

3.3.2 Edit the RLS configuration file

Edit the RLS configuration file `/etc/ldr-globus/globus-rls-server.conf` as follows:

1. For the `db_pwd` fields enter the MySQL database password for the `ldr` user that you granted access to previously. This is necessary so that RLS can connect to the databases.
2. In the line

```
acl <DN of the datarobot certificate>: all
```

replace `<DN of the datarobot certificate>` with the subject or Distinguished Name (DN) from the datarobot certificate. This is necessary so that your `ldr` user using the datarobot certificate can authenticate to the RLS server in order to query it.

If you don't know the subject you can obtain it by doing

```
grid-cert-info -file <path to datarobot cert> -subject
```

3. In the line

```
acl <DN of host certificates from other RLS server>: rli_update
```

replace `<DN of host certificates from other RLS server>` with the subject or DN for a host certificate that another RLS server in your grid will use to identify itself. Repeat as necessary for all the RLS servers in your grid or LDR network.

4. In the line

```
acl <DN of remote datarobot certificates>: lrc_read rli_read stats
```

replace `<DN of remote datarobot certificates>` with the subject or DN for the datarobot certificate from another LDR installation in your grid. Repeat as necessary for all the LDR installations in your grid or LDR network.

3.3.3 Edit the ODBC configuration file

On most systems the default ODBC configuration file `/etc/odbc.ini` is empty. If so you can simply copy the template in `/etc/LDR/odbc.ini` to `/etc/odbc.ini`. If the default file is not empty simply append the definitions in the file `/etc/LDR/odbc.ini` to `/etc/odbc.ini`.

3.3.4 Starting the RLS server

The RLS server will be managed using the monit daemon. Once monit is started it will start the RLS server and make sure that it continues to run.

To start the monit daemon

1. On CentOS or SL please run as root `service monit start`
2. On Debian please run as root `/etc/init.d/monit start`

Note that if your RLS contains a large number of mappings, it can take as long as 10 minutes for the server to compute the necessary bloomfilter (a type of hash) that represents that state of your RLS. During that time your RLS server may not respond to any commands.

You can test the server by contacting it using the `globus-rls-admin` tool. Make sure that you have credentials for the ldr user using the datarobot certificate and then ping the server:

```
bash$ globus-rls-admin -p rls://localhost
ping rls://localhost: 0 seconds
```

You can also use the `-S` option to `globus-rls-admin` and you should see something like this

```
bash$ globus-rls-admin -S rls://localhost
Version:      4.7
Uptime:       00:00:35
LRC stats
  update method: lfnlist
  update method: bloomfilter
  lfnlist update interval: 86400
  bloomfilter update interval: 900
  numlfn: 0
  numpfn: 0
  nummap: 0
RLI stats
  updated via bloomfilters
```

3.3.5 Final configuration using globus-rls-admin

Your RLS server needs to be configured so that it updates other RLS servers in your LDR network about its own contents. You do this using the `-A` option to `globus-rls-admin`. Enter `man globus-rls-admin` for details.

Please note the difference between the `-A` and `-a` options. Due to the number of files that most LDR networks manage, the RLS servers should be configured to use bloomfilter updates, rather than sending entire lists of files. Therefore you want to only use the `-A` option when configuring your RLS server.

3.4 Configuring globus-gridftp-server

Most of the configuration of the globus-gridftp-server was done when you installed LDR and located the host certificate and key.

The only remaining configuration to be done is to populate the grid-mapfile so that when the remote datarobots connect they are mapped to the appropriate local user UNIX accounts. Before detailing the mappings we discuss some of the effects of the mappings and our recommendation.

When a remote datarobot client connects to the globus-gridftp-server daemon running as user `root` it does a type of `setuid` so that the server runs effectively as the UNIX user as determined by the mapping in the grid-mapfile.

This is of particular interest to LDR since it is the only current way of limiting access to particular sites or groups. By mapping a remote site's datarobot credentials to a UNIX user that has (or does not have) certain access permissions to files as determined by the UNIX file permissions a LDR administrator can grant or restrict access to files or data.

For example if the remote site's datarobot has credentials

```
/DC=org/DC=doegrids/OU=Services/CN=datarobot/jester.phys.uwm.edu
```

and if the first line of the grid-mapfile for the GridFTP server has

```
"/DC=org/DC=doegrids/OU=Services/CN=datarobot/jester.phys.uwm.edu" jesterbot
```

then when the LDR client from the machine `jester.phys.uwm.edu` connects to the globus-gridftp-server it will do a `setuid` to the UNIX account `jesterbot`. *The account must exist, though it does not need to have login privileges.*

If a data file `/opt/important/data/file1` has permissions that make the file readable by `jesterbot` then that file may be replicated by the LDR running on `jester.phys.uwm.edu`. If not then a transfer of that file will fail.

Often within a grid or VO there will be a number of sites replicating data so a LDR admin will often create a UNIX account for the datarobot at each site. Further the local UNIX accounts are often members of UNIX groups that are used to protect sets of data, such as `ligos1`, `geoS3`, and the like. By manipulating the groups, the permissions on the files, and the grid-mapfiles an LDR administrator can easily make sets of files available to one site but not to another. This is the approach we recommend.

Another approach is to map *all* remote datarobot credentials to the *local LDR user*. This is convenient and will often work just fine but the LDR admin should realize that *in doing so he or she is granting full access to all data to all remote datarobots, and to the LDR installation files themselves most often*. It would be possible for a remote site to upload a different configuration file and alter the behavior of LDR and so this should be considered a serious security risk. We do not recommend this approach.

The grid-mapfile for the globus-gridftp-server is `/etc/ldr-globus/grid-mapfile.gridftp`. The format is simple; each line should contain the subject from a datarobot certificate contained in double quotes (") followed by the local UNIX account, as shown above.

Note that entries can be added on the fly at any given time to either file. The files are parsed during the startup of each service for each connection.

3.5 Configuring Apache

Before considering the details of the Apache configuration for the LDR web services. *you must take steps to secure the Apache configuration as it is installed by default* since the default installation on CentOS, SL, and Debian is relatively insecure.

3.5.1 Securing the default Apache configuration

The most important step to take is to edit the Apache httpd configuration and disable the server from listening on port 80 for standard HTTP traffic. The LDR web services only use port 443 and HTTPS on public network interfaces and so at this time port 80 should be turned off entirely.

On CentOS 5 and SL edit the file `/etc/httpd/conf/httpd.conf` and comment out the line

```
Listen 80
```

On Debian Lenny you should run the command

```
a2dissite 000-default
```

Please start or restart the Apache httpd and try to connect using a standard web browser. Make sure that you are not able to connect to the machine at all on port 80.

3.5.2 Configuring the LDR web services

Much of the configuration necessary for Apache in order to host the LDRMetadataServer and LDRDataFind-Server web services was done during installation of the LDR RPMs or Debian packages. With RPM installations you will find the configuration in

```
/etc/httpd/conf.d/ldr.conf
```

and with Debian installations you will find the configuration in

```
/etc/apache2/mods-enabled/ldr.conf
```

There are, however, two important configuration details that are not done automatically.

First, the environment variable `OPENSSL_ALLOW_PROXY_CERTS` must be set for the Apache httpd server process itself so that the `mod_ssl` code understands how to process RFC 3820 compliant proxy certificates.

On CentOS and SL systems edit the init file `/etc/init.d/httpd` and add the line

```
export OPENSSL_ALLOW_PROXY_CERTS=1
```

just above the line `HTTPD_LANG=${HTTPD_LANG-"C"}`.

On Debian systems edit the init file `/etc/init.d/apache2` and change the line

```
ENV="env -i LANG=C PATH=/usr/local/bin:/usr/bin:/bin"
```

to read instead

```
ENV="env -i LANG=C PATH=/usr/local/bin:/usr/bin:/bin OPENSSL_ALLOW_PROXY_CERTS=1"
```

Second, the Apache module `mod_ssl` needs to be configured to find the CA root or signing certificates so that proxy or X.509 certificates presented by clients (such as the `ligo.data.find` tool) can be properly verified.

CentOS and SL users may find it easiest to install the CA certificates distributed via the Open Science Grid Yum repository. See <http://yum.grid.iu.edu/> for details. At this time Debian users should contact the LDR project manager for suggestions.

After the CA signing and root certificates have been deployed on your system please edit the Apache httpd configuration file that contains the `mod_ssl` directives. For CentOS 5 and SL 5 the file is `/etc/httpd/conf.d/ssl.conf` and for Debian Lenny the file is `/etc/apache2/sites-enabled/default-ssl`. In the configuration file set define `SSLCACertificatePath` to point to the directory where you deployed the CA signing certificates.

For example, if you installed the CA signing certificates using the OSG Yum repository you would then set

```
SSLCACertificatePath /etc/grid-security/certificates
```

3.6 Configuring LDR

Version 0.9.x of LDR has one (1) configuration file that you will find in `/etc/LDR/ldr.ini`. Using this single configuration file you can configure all of the LDR daemons and services and configure information about remote sites and collections of files to replicate.

Just as necessary as the configuration files is the *local storage module*. Since the details of storage at each LDR site are usually vastly different, and in the hope of offering needed flexibility without too much overhead, we require that each site prepare a local storage module (a Python file) that provides an interface between LDR and the local storage. Details on the requirements for the local storage module follow below.

3.6.1 Editing ldr.ini

The `ldr.ini` file contains at a minimum the following sections under which configuration options can be placed:

- `[LDRMaster]`
- `[LDRTransfer]`
- `[LDRStorage]`
- `[LDRMetadataServer]`
- `[LDRDataFindServer]`
- `[Sites]`
- `[Collections]`

Additionally a few configuration options that apply globally, for example the name and password to use for access to the MySQL database, can be placed at the top of the file. These global options will be overridden if the same option appears within a specific section.

For each section there are a number of different configuration options. Strings and path names should be listed without any quotes. If an option requires a list of values the values should be separated using a comma, for example

```
[LDRMaster]
daemons = LDRTransfer, LDRMetadataUpdate
```

In the details that follow an asterisk (*) on a configuration option means that *no attempt* has been made at installation time to try and configure a useable default and you must edit the value for that option.

Of course, you should carefully review the details for all the options!

LDRMaster

The LDRMaster daemon is the main LDR program. It is most likely the only LDR program that you as an administrator will start or run. Its sole purpose is to start the other LDR processes LDRTransfer and LDRMetadataUpdate and watch over them, restarting each as is necessary.

The LDRMaster daemon requires the following configuration options:

- **daemons**: this should be list of the names of the LDR programs that LDRMaster is to start and watch. The order of the names is not important. The default is

```
[LDRMaster]
daemons = LDRTransfer, LDRMetadataUpdate
```

You may configure LDRMaster to set an environment variable for a specific daemon by using a *sub-section* for the daemon. For example, to have LDRMaster start the LDRTransfer daemon with the environment variable `LD_PRELOAD` set to `/usr/lib/libmtmalloc.so.1` you would write

```
[LDRMaster]
daemons = LDRTransfer, LDRMetadataUpdate

[[LDRTransfer]]
LD_PRELOAD = /usr/lib/libmtmalloc.so.1
```

To have LDRMaster *prepend* a value to an environment variable (for example to add something to LD_LIBRARY_PATH) prefix the environment variable with the character '\$'. For example to prepend /ldcg/lib to LD_LIBRARY_PATH for LDRSchedule you would write

```
[LDRMaster]
daemons = LDRTransfer, LDRMetadataUpdate

[[LDRTransfer]]
LD_PRELOAD = /usr/lib/libmtmalloc.so.1
$LD_LIBRARY_PATH = /ldcg/lib
```

- **lockfile**: the location of the lockfile that LDRMaster should use to check for instances of itself that may already be running. The default is best for most sites.
- **pidfile**: the location of a file where LDRMaster will write its pid. The default is best for most sites.
- **logfile**: the location where LDRMaster will write its own logfile. The default is `/var/log/LDR/LDRMaster.log` and is best for most sites.
- **logmaxbytes**: the size in bytes that the log file may grow to before it is rolled over and a new log file started. This is often set as a global option at the top of the file so that it applies to all LDR components. A value may be entered using a simple Python expression, for example

```
logmaxbytes = 1024 * 1024 * 100
```

- **logbackupcount**: the number of backups to keep as the log files roll over as the main log file grows to logmaxbytes in size. This is often set as a global option at the top of the file so that it applies to all LDR components.
- **loglevel**: the level of messages that should be logged. Options are DEBUG, INFO, WARNING, ERROR, CRITICAL. This is often set as a global option at the top of the file so that it applies to all LDR components.

LDRTransfer

The LDRTransfer daemon is responsible for determining what files your site does not have, scheduling those files for replication, and then orchestrating the transfer of those files. It is usually started by LDRMaster and runs as a client using the datarobot certificate to connect to both the local and remote RLS servers in order to find files, as well as both local and remote GridFTP servers.

- **rls**: the URL for the local RLS server.
- **datarobotcert**: the path to the datarobot certificate. This is often set as a global option at the top of the file so that it applies to all LDR components.
- **datarobotkey**: the path to the corresponding datarobot key. This is often set as a global option at the top of the file so that it applies to all LDR components.
- **logfile**: the location where LDRTransfer will write its own logfile. The default is `/var/log/LDR/LDRTransfer.log` and is best for most sites.
- **logmaxbytes**: the same as for LDRMaster.
- **logbackupcount**: the same as for LDRMaster.
- **loglevel**: the same as for LDRMaster.

- **sleeptime**: the number of seconds between scheduling cycles for collections. This can be set on a per-collection basis in the Collections section.

This should be short enough so that there are always files in the queue to be transferred but long enough so that the LDR machine is not burdened by extra MySQL work.

The default works best for most sites.

- **cbsleeptime**: the number of seconds between polls to the database to find files that have successfully transferred and publish their URLs into RLS. The default is 10 seconds and should work well for most sites.
- **maxcleanup**: the maximum number of files that are allowed to end up in the CLEANUP state where intervention is required. The default is 10 and should work well for most sites.
- **rrddir**: the directory into which round robin database files for transfer rate statistics is written. The default should work well for most sites.

LDRStorage

- ***storagemodule**: the name of your local storage module, usually kept in the directory `/etc/LDR`. See below for detailed instructions on how to construct a local storage module.
- **storagelogging**: whether to use a separate file for logging messages from the local storage module. The default is True and should work well for most sites.
- **logfile**: the location where LDRTransfer will write the storage logfile. The default is `/var/log/LDR/LDRStorage.log` and is best for most sites.
- **logmaxbytes**: the same as for LDRMaster.
- **logbackupcount**: the same as for LDRMaster.
- **loglevel**: the same as for LDRMaster.

LDRMetadataUpdate

The LDRMetadataUpdate client is used to keep the local LDR metadata catalog up to date for psets of metadata that are interesting and useful for your site. It is usually started by LDRMaster and uses the datarobot credentials to authenticate to a remote LDRMetadataServer web service.

The LDRMetadataUpdate client has the following configuration options:

- **logfile**: the location where LDRMetadataUpdate will write its own logfile. The default is `/var/log/LDR/LDRMetadataUpdate.log` and is best for most sites.
- **logmaxbytes**: the same as for LDRMaster.
- **logbackupcount**: the same as for LDRMaster.
- **loglevel**: the same as for LDRMaster.

To configure LDRMetadataUpdate to retrieve the metadata for a particular pset you need to configure a subsection for a pset and set the server or host name for the remote server from which to obtain the metadata.

For example, to configure your LDRMetadataUpdate client to retrieve metadata from the LDR deployment at Caltech for the `S6_LHO_RDS` and `S6_LLO_RDS` psets the LDRMetadataUpdate section might look like this:

```
[LDRMetadataUpdate]
```

```
[[S6_LHO_RDS]]
```

```
server = ldr.ligo.caltech.edu
```

```
[[S6_LHO_RDS]]
```

```
server = ldr.ligo.caltech.edu
```

LDRMetadataServer

The LDRMetadataServer web service is responsible for servicing requests from other LDR installations to obtain information about new metadata. Only sites where files are being published need to run the LDRMetadataServer web service. If your site is only replicating data or making data available but is not publishing any *new* metadata then a LDRMetadataServer service is not required.

- **authorization**: the type of authorization to use for clients connecting to obtain metadata updates. The default is `grid-mapfile` and should work well for most sites.
- **grid-mapfile**: the path to the grid-mapfile that will be used for authorization when clients connect to the server to obtain updates. The default location is `/etc/LDR/grid-mapfile.LDRMetadataServer` and should work well for most sites.
- **logfile**: the location where LDRMetadataServer will write the storage logfile. The default is `/var/log/LDR/LDRMetadataServer.log` and is best for most sites.
- **logmaxbytes**: the same as for LDRMaster.
- **logbackupcount**: the same as for LDRMaster.
- **loglevel**: the same as for LDRMaster.

LDRDataFindServer

The LDRDataFindServer web service is responsible for servicing requests by clients (usually clients being run by users) to find data at a site using constraints on metadata attributes. Only sites where data should be exposed at the user level need to run a LDRDataFindServer web service.

- **type**: the type of backend engine to use for the data find service. The current supported types are `lsync` or `rls`.

With a backend of `lsync` the server parses a plain-text or ascii dump from the LIGO Laboratory diskcacheAPI (a product also known at one time as `lsync`) to construct mappings from metadata and file names to paths or URLs. Your site must have deployed and be running the `diskcacheAPI` tool in order to use the LDR `lsync` engine for LDRDataFindServer.

The following configuration options apply only for use with the `lsync` engine:

- **framecachetimeout**: the time between polls to check for and if present parse an updated dump file. The default is 60 seconds and should work well for most sites.
- **cache_site_exclue_exact**: a comma separated list of strings representing site metadata (for example H, L, V, A) that should be *excluded* when the dump file is parsed. Only lines in the dump file where the site metadata exactly matches one or more of the strings in the list will be excluded.
- **cache_site_exclue_pattern**: a comma separated list of string patterns representing site metadata (for example H, L, V, A) that should be *excluded* when the dump file is parsed. Lines in the dump file where the site metadata is matched by one or more of the patterns will be excluded.
- **cache_frametype_exclue_exact**: a comma separated list of strings representing frame type metadata (for example T, R, M, RDS) that should be *excluded* when the dump file is parsed. Only lines in the dump file where the frame type metadata exactly matches one or more of the strings in the list will be excluded.
- **cache_frametype_exclue_pattern**: a comma separated list of string patterns representing frame type metadata (for example T, R, M, RDS) that should be *excluded* when the dump file is parsed. Lines in the dump file where the frame type metadata is matched by one or more of the patterns will be excluded.

With a backend of `rls` the server uses lookups in the LDR and RLS database tables to construct mappings from metadata and file names to paths or URLs. The `rls` engine is the default and will be automatically used at sites that do not specifically configure a LDRDataFindServer engine.

- **authorization**: the type of authorization the server uses to allow or deny access to the services. Currently two types of authorization are allowed:

- **grid-mapfile**: the server expects a client to use a RFC compliant proxy certificate or end-entity X.509 certificate to authenticate to the service, and from that the server will extract the subject or DN from the certificate. The subject will be compared to a grid-mapfile and if the subject is present in the grid-mapfile the client will be authorized to use the service.
- **virtual-host**: with this value for authorization you must configure at least one subsection where the name of the subsection is the IP address of a virtual host. Connections made to the server through that virtual host IP address are then authorized using the value specified for the **authorization** option specific to that subsection. The possible choices for the authorization option for a virtual host subsection are **grid-mapfile** as above and **None**, indicating that no authorization (or authentication) will be required.

For example suppose you wish to configure LDRDataFindServer to listen both on a network interface that is open to the wide internet and on a private network interface only available inside your local computing cluster. On the public network interface the server should require RFC compliant proxy or X.509 certificates and on the private network interface the server should not require any authentication. If the IP address for the public interface is 129.89.61.245 and the IP address for the private interface is 192.168.2.10 then the configuration would look like this:

```
[LDRDataFindServer]
  authorization = virtual_host

  [[129.89.61.245]]
  authorization = grid-mapfile

  [[192.168.2.10]]
  authorization = None
```

- **grid-mapfile**: the location of a grid-mapfile to be used by the server. This option may be specified in the LDRDataFindServer section or in a virtual host subsection. The default is `/etc/LDR/grid-mapfile.LDRDataFindServer` and should work well for most sites.
- **logfile**: the location where LDRDataFindServer will write the storage logfile. The default is `/var/log/LDR/LDRMetadataServer.log` and is best for most sites.
- **logmaxbytes**: the same as for LDRMaster.
- **logbackupcount**: the same as for LDRMaster.
- **loglevel**: the same as for LDRMaster.
- **filter_exclude**: a list of regular expressions to use to filter PRNs or URLs returned by the service. URLs matching the expressions will be excluded and not returned. configuration option.

For example if the option is configured as

```
filter_exclude = cache, GRB
```

then any URL matching the text 'cache' or 'GRB' will not be returned to the client.

- **filter_preference**: a Python dictionary with URL types as keys, and a list of preferences as regular expressions as values. For example

```
filter_preference = { "^file" : ['node', 'slow_cache'] }
```

will return only one PFN, if the user requests "file://" URLs. With those PFNs which match the regex 'node' (no quotes) first. In other words, 'node' PFNs will be preferred over everything, and 'slow_cache' PFNs will be preferred over everything except 'node' PFNs.

***Sites**

In the Sites section you must define as subsections the sites in your LDR network that you will interoperate with and for each site you must specify at least the URL for the site's RLS server. You will most likely also want to specify some default network parameters such as the TCP window size to use during file replication.

For example suppose that your LDR is expected to retrieve data from Caltech and UW-Milwaukee. Also suppose that you have determined in the past that the best performance is obtained for transfers from Caltech when the data connections use 1 MB TCP windows but for UW-Milwaukee the optimal size is 2 MB. Your Sites section then might look like this:

```
[Sites]

[[CIT]]
rls = rls://ldr.ligo.caltech.edu
tcpbs = 1024 * 1024

[[UWM]]
rls = rls://nemo-dataserver.phys.uwm.edu
tcpbs = 1024 * 1024 * 2
```

***Collections**

The Collections section is where you define the collections of data that should be replicated to your local site. There are many options and combinations of options that may appear the various section, subsection, and sub-subsection level.

For each defined collection two threads of control are started. One thread is responsible for scheduling the next set of files to be replicated and replenishing a queue kept in the database. The second thread manages the replication of the data files, using GridFTP, and drains the queue.

In the interest of brevity, rather than detailing each option, please look over these examples:

```
[Collections]

[[LHO_S6_H1_LDAS_C00_L2]]
concurrency = 2           # use two concurrent control channels
numdatastreams = 2       # use two data streams per control channel
type = metadata          # collection is defined by metadata
[[[metadata]]]
  site = site = 'H'
  runTag = runTag = 'S6'
  frameType = frameType = 'H1_LDAS_C00_L2'
  gpsStart = gpsStart >= 928868400 AND gpsStart <= 999999999
  LDROrderAttribute = gpsStart
  LDROrdering = ASC

[[LHO_S6_L1_LDAS_C00_L2]]
concurrency = 2           # use two concurrent control channels
numdatastreams = 2       # use two data streams per control channel
siteselect = ordered     # use a preferred ordering of source sites
siteorder = CIT, UWM     # prefer CIT to UWM as source site
type = metadata          # collection is defined by metadata
[[[metadata]]]
  site = site = 'L'
  runTag = runTag = 'S6'
  frameType = frameType = 'L1_LDAS_C00_L2'
  gpsStart = gpsStart >= 928868400 AND gpsStart <= 999999999
  LDROrderAttribute = gpsStart
  LDROrdering = ASC

[[V_VSR2_HrecOnline]]
```

```

concurrency = 4           # use 4 concurrent control channels
numdatastreams = 2       # use two data streams per control channel
allowedsites = HAN       # only replicate from Hannover
type = metadata          # collection is defined by metadata
[[[metadata]]]
  site = site = 'V'
  runTag = runTag = 'VSR2'
  frameType = frameType = 'HrecOnline'
  gpsStart = gpsStart >= 928868400 AND gpsStart <= 999999999
  LDROrderAttribute = gpsStart
  LDROrdering = ASC

```

3.6.2 Local Storage Module

Since each site most often has local constraints and policies on how data needs to be stored on a filesystem or in some other local storage it is necessary that there be a mechanism so that LDR can be made to adapt to and support the local policies. The current mechanism for LDR is that each site is required to provide a *local storage module*. The local storage module is a Python module (a simple text file) that implements a number of pre-defined LDR functions or methods so that when the LDR tools call the functions actions are taken that respect and implement the local policies with regard to storage.

Note that the methods require that the first argument be “self”. This is a Python idiom for methods of class instances. It must be included by you can ignore its presence.

The necessary methods include:

- **newHoldingFile**: This method is used to return URLs that can be used by the LDR transfer agents as destination URLs for file transfers. This method most often is used to return `gsiftp://` URLs that enable the transfer agents to push the files onto your local storage through the Globus GridFTP server. The URL returned must be a valid URL (for example the directories within the URL must already exist).

Note that a temporary file name or suffix such as `.tmp` should *not* be used here since the LDR agents automatically add a temporary suffix.

The method takes as its only argument a *list* of logical file names (LFNs) that are being scheduled for transfer. Given the list of LFNs the method returns a Python dictionary or hash table. The keys of the dictionary are the LFNs that were input to the method and the values of the dictionary are the URLs (again usually these are `gsiftp://` URLs).

- **newFileCallback**: Perform any necessary steps after a group of files has been transferred and the checksums verified but *before* the mappings of the LFNs to URLs or PFNs has been recorded in the local replica catalog (RLS server). Some typical necessary steps might include changing the user id of a file, setting the group id for a file, or changing the permissions on a file. Note that performing a checksum on a file should not be done within this method since the LDR transfer agents do this automatically.

This method takes as its argument a Python dictionary with LFNs as keys and URLs as values in the same format that was returned by the **newHoldingFile** method above. This method does not return anything, but it can raise an exception if necessary to indicate a failure.

- **enterFile**: Return a Python dictionary with LFNs as keys and a list of URLs as values that should be recorded in the local replica catalog (RLS server) for the successfully replicated files. Usually for each LFN a list of URLs is returned that includes a `gsiftp://` URL so that the file can be retrieved via GridFTP and at least one `file://` URL that can be used to access the file directly from a local file system.

This method takes as arguments a Python dictionary with the LFNs as keys and the URL or list of URLs that was used to transfer or replicate the file.

- **enterAttr**: This method can be used to set string attributes on the PFNs or URLs that are recorded in RLS. LIGO sites often use this method to set the “pool” attribute on PFNs of the form `file://` to indicate to workflows planned with Pegasus that the `file://` URL is to be used for work done locally.

The input is a list of PFNs, some of which may or may not be of the form `file://`. This method should return a Python dictionary with PFNs as keys and tuples as the values. The tuple should be of the form `(string attribute name, string attribute value)`.

- `failedTransferCallback`: Perform any *storage-specific* cleanup necessary if a transfer has failed. This method will be called if during or after a new file has been replicated by the transfer agent it is detected that there was an error in the transfer. For example if the transfer agents tests the checksum for the transferred file and it does not match the reference checksum.

The transfer agent is responsible for removing the remains of failed transfers so this method should not normally attempt to delete failed transfers, especially since the transfer agent may have transferred the file to a temporary URL by appending a prefix (such as `".tmp"`) to the destination URL.

This method should only be used for any special and specific steps storage needs to take after a file has failed to transfer properly.

It should not return anything. If there are errors operating on failed LFN/URL pairs they should be logged and if the errors are critical a `LDRStorageError` exception should be raised.

- `__init__`: This method does not usually need to be modified. It is the method called when LDR first starts up. It is only necessary to modify this method if your particular local storage requires some type of initialization when LDR is started (for example if you have a tape drive that needs to be initialized in some particular way).

Testing a local storage module

To test your local storage module before running LDR to make sure that the methods you have written are returning the results you expect you can follow this recipe:

- Edit your local storage module and towards the bottom add the lines

```
if __name__ == '__main__':
    logging.basicConfig(level=logging.DEBUG)

    import sqlalchemy
    import MySQLdb
    import LDR.LDRConfigParser
    import LDR.LDRMetadata

    configParser = LDR.LDRConfigParser.LDRConfigParser()
    dbuser = configParser.get("dbuser", "LDRTransfer", None)
    dbpasswd = configParser.get("dbpasswd", "LDRTransfer", None)
    url = sqlalchemy.engine.url.URL('mysql',
        username = dbuser,
        password = dbpasswd,
        host = 'localhost',
        database = 'LDR')
    db = sqlalchemy.create_engine(url)

    metadata = LDR.LDRMetadata.LDRMetadata(db)
    myStorage = LDRStorage(configParser, metadata)
```

- Start Python using the `-i` flag and the name of your local storage module:

```
python -i <local storage module>.py
```

- You can now call the methods that you wrote using the `myStorage` instance of the `Storage` class that is created for you. For example to test the `newHoldingFile` method at UWM one might run

```
$ python -i UWM-local-storage.py
>>> myStorage.newHoldingFile(['GHLV-HSG6_S5_R2-851508814-1000.gwf'])
'file://nemo-dataserver.phys.uwm.edu/data/GHLV-HSG6_S5_R2-851508814-1000.gwf'
```

3.7 Firewalls

The components included in a standard LDR deployment communicate on ports often blocked by default when LDR is deployed behind a firewall. Below we discuss in detail each of the ports that a standard LDR deployment needs open and why. Some LDR components require an environment variable to be set to restrict the ports used by the tool, and we include those details as well. Tables 3.1 and 3.2 summarize the required ports for listening (incoming) and sending (outgoing) network communications.

Note that at this time all LDR components only use TCP for network communications. No UDP communications are used at this time.

Also note the distinction between blocking incoming network communications and outgoing network communications. A standard LDR deployment requires significantly fewer adjustments in firewall rules when the firewall only restricts incoming network traffic.

3.7.1 Static and ephemeral ports

First some definitions:

- **Static port:** A deterministic port on which a daemon or server listens for incoming network connections. An example is a standard `sshd` listening on port 22. Administrators often configure firewalls that block incoming network connections to allow connections to certain static ports (sometimes restricting incoming connections to those from certain network domains).
- **Ephemeral port:** A non-deterministic port assigned by the system in the untrusted port range (> 1024). Often a client tool (or a daemon acting as a client) opens a connection using a local ephemeral port to a well known static port on a remote server. For example, when the user invokes `ssh some.server.org` the SSH client requests and receives from the system an ephemeral port (> 1024) to use to connect to a remote server running on the well known port for the `sshd` (by default port 22). That is, the server uses port 22 but the client uses some port above 1024 that is not determined until the user invokes the `ssh` client tool.
- **Controllable ephemeral port:** An ephemeral port selected by the Globus Toolkit libraries to lie within a range of ports specified using the environment variable `GLOBUS_TCP_PORT_RANGE`. One may set the environment variable so that client tools (or daemons and servers acting as clients) built on top of the Globus Toolkit libraries only use ephemeral ports in the specified range. Restricting the range of ephemeral ports facilitates writing firewall rules when the firewall is expected to restrict outgoing network connections. To restrict outgoing network connections the value for `GLOBUS_TCP_PORT_RANGE` should be set to `min,max` where `min` is the lowest port in the allowed range and `max` is the highest port in the allowed range. So to only allow client tools to use ports from 40,000 to 40,500 one might enter

```
export GLOBUS_TCP_PORT_RANGE=40000,40500
```

before running a client tool.

Many of the LDR components and tools act as servers and listen on static ports. Other LDR components and tools, however, act as clients and require ephemeral ports. Some components act as both servers and clients and require both static and ephemeral ports. LDR components that use ephemeral ports can be configured to use controlled ephemeral ports if necessary. *Controlled ephemeral ports are generally only needed when firewall rules restrict outgoing connections.* A firewall that restricts only incoming network connections will usually not require the use of controlled ephemeral ports (an exception is the `globus-gridftp-server`—details on this exception follow later).

3.7.2 Listening ports needed by individual LDR components

A number of LDR components listen on static (or in the case of GridFTP both static and ephemeral) ports for incoming connections:

- **Globus GridFTP:** By default the `globus-gridftp-server` deployed for use with LDR listens on port 15000 for the FTP *control* channel. This is a different port for the control channel than the default port used for normal Globus Toolkit or Virtual Data Toolkit (VDT) deployments of the server in order to not interfere with servers that might already be deployed at a site and used for other purposes.

The `globus-gridftp-server` also listens for connections to setup *data* channels. The ports used for data channels, however, are not static ports. Instead the server uses ephemeral ports for the data channels, so you must also enable a range of controlled ephemeral ports to receive incoming network packets that carry data.

The number of controlled ephemeral ports required depends on the number of simultaneous file transfers and the number of parallel data streams each transfer is using. Typically in the LIGO collaboration administrators have configured the range 40,000 to 40,500 for a total of 500 ports available to GridFTP data streams.

Before starting the `globus-gridftp-server` you must set the environment variable `GLOBUS_TCP_PORT_RANGE` to the range you have made available for controlled ephemeral ports when adjusting your incoming firewall rules. For example:

```
export GLOBUS_TCP_PORT_RANGE=40000,40500
```

Note that if you use the command line tool `globus-url-copy` distributed with LDR for testing it too must have the environment variable `GLOBUS_TCP_PORT_RANGE` set appropriately to the range of ports enabled in the firewall rules.

- **Globus RLS:** By default the `globus-rls-server` listens on the static port 39281. It needs to receive packets from other RLS servers in the LDR network that are updating the server with their own contents (this is how your local RLS server learns about which files are available at which sites). It also needs to respond to queries from client tools at other sites so that other sites can determine the LFN to PFN mappings available at your local site.
- **LDRTransfer:** The `LDRTransfer` daemon is essentially a GridFTP client. As such it requires the same ephemeral ports as the standard `globus-url-copy` client. To restrict the range of ephemeral ports used the `GLOBUS_TCP_PORT_RANGE` environment variable should be set before the `LDRTransfer` daemon is run.
- **LDRMetadataServer:** The `LDRMetadataServer` web service is served by Apache httpd which by default listens on the static port 443.
- **LDRDataFindServer:** The `LDRDataFindServer` web service is served by Apache httpd which by default listens on the static port 443.

3.7.3 Ephemeral ports needed by individual LDR components

A number of LDR components also require ephemeral ports for network communication:

- **Globus GridFTP:** If your firewall also blocks *outgoing* network connections you must enable a range of ephemeral ports to be used for outgoing connections. Typically in the LIGO collaboration administrators have configured the range 40,501 to 41,000. Before starting the `globus-gridftp-server` you must set the environment variable `GLOBUS_TCP_SOURCE_RANGE` to the range you have made available for outgoing network packets. For example:

```
export GLOBUS_TCP_SOURCE_RANGE=40501,41000
```

Note that if you use the command line tool `globus-url-copy` distributed with LDR for testing it too must have the environment variable `GLOBUS_TCP_SOURCE_RANGE` set appropriately to the range of ports enabled in the firewall rules.

- **Globus RLS:** The `globus-rls-server` must be able to send outgoing network packets to update other RLS servers in the LDR network with its own contents so that other sites may learn about which files are available from your local site. If your firewall blocks outgoing network packets you must enable a (small) range of ephemeral ports for the outgoing packets and then set `GLOBUS_TCP_SOURCE_RANGE` as detailed above. A range of 10 ports should be sufficient for most LDR deployments.
- **LDRMetadataUpdate:** The `LDRMetadataUpdate` daemon is essentially a simple web client that expects to query remote web services using HTTPS. At this time it is not possible to restrict the ephemeral port used by the client to a specific range.

- **LDRTransfer:** The `LDRTransfer` daemon acts as a client and contacts remote `globus-rls-server` daemons using an ephemeral port. If your firewall blocks outgoing network packets you must enable a range of ephemeral ports for the outgoing packets and then set `GLOBUS_TCP_SOURCE_RANGE` as detailed above. A range of 100 ports should be sufficient for most LDR deployments.

3.7.4 Summary of all ports needed

Tables 3.1 and 3.2 summarize the ports needed by a default LDR deployment.

incoming port	tool	comment
15000	globus-gridftp-server	incoming control channel connection
40000-40500	globus-gridftp-server	suggested range for incoming GridFTP data channels
39281	globus-rls-server	default for incoming queries and updates
443	LDRMetadataServer	default for incoming queries
443	LDRDataFindServer	default for incoming queries

Table 3.1: Listening ports required for a standard LDR deployment.

outgoing port	tool	comment
40501-41000	globus-gridftp-server	suggested range for outgoing GridFTP data channels
40100-40110	globus-rls-server	suggested for outgoing updates
40300-40310	LDRTransfer	suggested, 100 ports usually sufficient

Table 3.2: Additional outgoing ports required for a standard LDR deployment. These ranges for controlled ephemeral ports are usually only required for a firewall configured to block outgoing connections.

Chapter 4

Running LDR

4.1 Quick instructions for long-time LDR admins

Please read below if you have not run LDR before or are looking for details. If you are simply looking for the simple instructions on how to get LDR running then follow these:

```
/etc/init.d/mysqld start
/etc/init.d/gridftp start
/etc/init.d/monit start
/etc/init.d/httpd start (on Debian use 'apache2' instead of httpd)
/etc/init.d/ldr start
```

To shut everything down follow these instructions:

```
/etc/init.d/ldr stop
/etc/init.d/httpd stop (on Debian use 'apache2' instead of httpd)
/etc/init.d/monit stop
/etc/init.d/rls stop
/etc/init.d/gridftp stop
/etc/init.d/mysqld stop
```

Note that you must shut down monit first and then RLS. If you do not shut monit down first it will restart RLS as soon as you shut it down.

4.2 Overview

After the basic configuration for LDR is completed as detailed in the previous sections, running LDR and replicating data is straightforward. The sequence of events is

- Data must be “published” at a source site or sites. Publishing means adding metadata about files to the LDR database and entering URLs for the files into the RLS local replica catalog (LRC).
- The published metadata must be replicated to sites that want to replicate the data files themselves. This is accomplished by having the source sites run a [LDRMetadataServer](#) and the destination sites run a [LDRMetadataUpdate](#) client.
- The existence of URLs for data files must be replicated in the RLS servers. RLS does this without any other LDR support, provided the servers have been configured as detailed above. Each source site needs to have the RLS server configured to update each destination site at the very least.

Once these things have happened then a local LDR instance is able to determine (based upon the configuration) what files need to be updated and from where they can be scheduled for replication. The LDRTransfer daemon [LDRSchedule](#) and schedules the files for replication, orchestrates the replication and then records the new locations in the local RLS server.

4.2.1 Starting globus-gridftp-server

The preferred method for starting the server is to use the init script distributed with LDR. The server is usually started by the `root` user, though other configurations are possible. See the Globus Toolkit 4.2 documentation for details.

To start the server:

```
/etc/init.d/gridftp start
```

4.2.2 Starting MySQL

The MySQL server `mysqld` needs to be started before starting either Globus RLS or LDR proper. The preferred method for starting the server is to use the init script.

To start the server:

```
/etc/init.d/mysqld start
```

4.2.3 Starting RLS

The RLS server needs to be started before starting LDR proper. The preferred method for starting the server is to have the monit daemon manage RLS so that if the server crashes it will be quickly restarted by monit. As soon as you start the monit daemon if `globus-rls-server` is not running then monit will start it:

To start the server:

```
/etc/init.d/monit start
```

Note that monit runs the init script `/etc/init.d/rls` and this starts the server with arguments so that it logs to the file `/opt/ldr-globus/var/globus-rls-server.out`—*this file is not automatically rotated*. By default the RLS server logs at level 2. You may edit the init script and change the debugging level by changing the value for the `-L` option to the server. A level of 8 is recommend when debugging.

4.2.4 Publishing metadata and URLs

See chapter 5 for a detailed discussion of publishing of interferometer data into LDR.

4.2.5 Obtaining existing metadata

Before a site can replicate data it first must be updated to obtain the necessary metadata that describes the data you want to replicate. Specifically your local `LDRMetadataUpdate` client must connect to the remote `LDRMetadataServer` that acts as an authority for the metadata publication set (pset) containing the metadata.

Note that your local LDR may only receive metadata for any particular publication set or “pset” from a *single* remote metadata server. For example, if you configure your LDR to receive metadata updates for the `LHO_S6_RDS` pset from Caltech then your LDR must *only* receive updates for that pset from Caltech. You may *not* decide to later switch and begin updating that pset from UW-Milwaukee.

To have the local `LDRMetadataUpdate` contact the server, download the metadata, and insert it into the local database configure your `/etc/LDR/ldr.ini` file as detailed above with subsections for each pset you wish to receive. In each subsection you must specify the hostname of the remote authority server. For example, to have your LDR receive metadata for the psets `LHO_S6_RDS` and `LLO_S6_RDS` from Caltech your configuration would be

```
[LDRMetadataServer]
```

```
[[LHO_S6_RDS]]
```

```
server = ldr.ligo.caltech.edu
```

```
[[LLO_S6_RDS]]
```

```
server = ldr.ligo.caltech.edu
```

If the metadata in the pset is static (no new metadata is being published into the pset then you are finished. There is no need to configure `LDRMetadataUpdate` to regularly update the metadata for the pset. If metadata is still being published into the pset then continue.

4.2.6 Starting LDR

The preferred method for starting the server is to use the init script distributed with LDR. The init script will run the LDR daemons as the ldr user.

To start the server:

```
/etc/init.d/ldr start
```

The LDRMaster daemon will fork and then start the other LDR daemons as child processes. Those processes and the LDRMaster daemon will log output to log files (see below).

4.2.7 Log files

Each of the LDR daemons will create a log file by default in the directory `/var/log/LDR`. The size of the log files, their verbosity, and the rollover count can all be configured. See the documentation above about configuring `ldr.ini` for details.

4.2.8 Shutting down LDR and its components

When it is necessary to shut down LDR and its components (MySQL and RLS) please do the following in order:

1. **shut down LDR:** The preferred method to shut down the LDR daemons is to use the installed init script. You can monitor the progress by watching the LDRMaster log file.
2. **shut down RLS:** You must first shutdown the monit daemon before shutting down RLS or else monit will just restart the globus-rls-server. The preferred method is to use the init script to shutdown monit and then use the init script to shutdown RLS. It may take 30 seconds or so for the server to shutdown as it needs to finish any transactions and then properly close the connections to the MySQL server.
3. **shut down MySQL:** The preferred method to shut down the MySQL server is to use the LDR specific init script. It may take a minute or so for the server to shutdown.
4. **shut down GridFTP:** The preferred method to shut down the GridFTP server is to use the LDR specific init script.

Chapter 5

Publishing Data into LDR

There are currently two tools distributed with LDR for publishing:

1. **LDRPublishLIGOLab**: used primarily at the LIGO Lab sites for steady state publishing of interferometer data frame files.
2. **LDRBundle**: used primarily at the AEI Hannover site for steady state bundling of LIGO strain data files into larger frame files for consumption by the Virgo project.

Below we quickly demonstrate some of the configuration options for each tool.

5.1 LDRPublishLIGOLab

```
dbuser = ldr
dbpasswd = XXXXXXXX

logmaxbytes = 1024 * 1024 * 500
logbackuptcount = 10
loglevel = DEBUG

[LDRPublish]
dry_run = False
logfile = /home/ldr/Publishing/trend/second/publishing.log
frame_cache_file = /ldas_outgoing/diskcacheAPI/frame_cache_dump
frame_cache_refresh = 60
site = H
runtag = trend
pset = LHO_trend_2
interferometer = H1
frame_type = T
extension = gwf
main_path = /archive/frames/trend/second-trend/LHO
min_gps = 928521500
max_gps = 1100000000
```

5.2 LDRBundle

```
[LDRBundle]
dry_run = False
gsiftp_host = ldr.aei.uni-hannover.de:15000
bundle_command = /opt/lscsoft/libframe/bin/FrCopy
verify_command = /opt/lscsoft/libframe/bin/FrCheck -i
directory = /srv/data
remote_rls = rls://dataldr.virgo.infn.it
```

```
[[H1_LDAS_C00_L2]]  
site = H  
frameType = H1_LDAS_C00_L2  
runTag = E14  
interferometer = H1  
bundle_duration = 16384  
pset = BUNDLE_E14  
last_bundle_time = 929127552
```

```
[[L1_LDAS_C00_L2]]  
site = L  
frameType = L1_LDAS_C00_L2  
runTag = E14  
interferometer = L1  
bundle_duration = 16384  
pset = BUNDLE_E14
```

Chapter 6

Configuring LDRDataFindServer for Private Network Interfaces

Please do not follow these instructions during the initial deployment. Begin by making sure that the default LDRDataFindServer deployment works properly on the default virtual host and that the rest of LDR is properly deployed and configured before returning here for this advanced configuration step.

The default LDR installation configures the LDRDataFindServer web service running under Apache to listen on the default virtual host using SSL. In this configuration the server listens on all network interfaces on port 443 and the LDRDataFindServer web service requires a valid proxy or X.509 certificate to connect to the service.

It is often desirable to have the LDRDataFindServer web service not require authentication or authorization for use on *private* network interfaces (for example within a LIGO Data Grid computing cluster internal network). The strategy to enable this mode requires

1. turning off the default virtual host listening on port 443.
2. configuring a virtual host to listen on port 443 on the public network interface.
3. configuring a virtual host to listen on port 80 on the private network interface.
4. editing the LDR configuration file to define the virtual hosts in use and configure the authorization used for each.

6.1 Turning off the default virtual host

To turn off the default virtual host listening on port 443 Debian users can run

```
a2dissite default-ssl
```

CentOS and SL users should edit the file `/etc/httpd/conf.d/ssl.conf` and comment out the lines including and between

```
<VirtualHost _default_:443>
...
</VirtualHost>
```

6.2 Configuring virtual host for public network

6.2.1 Debian

The recommend approach for Debian users to configure a virtual host for the public network that will listen on port 443 and require proxy or X.509 certificate authentication is to create the file `/etc/apache2/sites-available/ldr-external` with the following contents:

```
<IfModule mod_ssl.c>

Listen YOUR_PUBLIC_IP_ADDRESS_HERE:443

<VirtualHost YOUR_PUBLIC_IP_ADDRESS_HERE:443>
    ServerAdmin webmaster@localhost
    ServerName YOUR_PUBLIC_FQDN_HERE:443

    DocumentRoot /var/www/
    <Directory /var/www/>
        Options -Indexes
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>

    ErrorLog /var/log/apache2/lbr_external_error.log

    LogLevel warn

    SSLEngine on
    SSLCertificateFile /etc/ssl/certs/localhost.crt
    SSLCertificateKeyFile /etc/ssl/private/localhost.key
    SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt
    SSLCACertificatePath /etc/grid-security/certificates

    BrowserMatch ".*MSIE.*" \
        nokeepalive ssl-unclean-shutdown \
        downgrade-1.0 force-response-1.0

    <Directory /var/www/LDR>
        Options -Indexes
    </Directory>

    <Directory /var/www/LDR/LDRMetadataServer>

        SSLVerifyClient require
        SSLVerifyDepth 10
        SSLOptions +StdEnvVars +ExportCertData

        WSGIProcessGroup LDRMetadataServer
        WSGIApplicationGroup %{GLOBAL}

        Options -Indexes
    </Directory>

    <Directory /var/www/LDR/LDRDataFindServer>

        SSLVerifyClient require
        SSLVerifyDepth 10
        SSLOptions +StdEnvVars +ExportCertData

        WSGIProcessGroup LDRDataFindServer
        WSGIApplicationGroup %{GLOBAL}

        Options -Indexes
```

```

        </Directory>
</VirtualHost>
</IfModule>

```

Replace `YOUR_PUBLIC_IP_ADDRESS_HERE` with the IP address for the public network interface. *Do not use the FQDN for this option. You must use the numerical IP address.* Also replace `YOUR_PUBLIC_FQDN_HERE` with the FQDN for the public IP address. Adjust any of the other SSL options appropriately for your local configuration.

The “ldr module” must also be edited. Edit the file `/etc/apache2/mods-available/ldr.conf` and comment out the lines starting from

```
<Directory /var/www/LDR>
```

until the end. This is necessary because the directory configurations are now in the virtual host file you just created above.

Next Debian users should run

```
a2ensite ldr-external
```

to enable the virtual host configuration. Start or stop Apache httpd and test to make sure that LDR-DataFindServer is again responding appropriately.

6.2.2 CentOS and SL

CentOS and SL users should edit the file `/etc/httpd/conf.d/ldr.conf` so that it now reads as

```

# location of WSGI sockets
WSGISocketPrefix run/

# one process with 15 threads for serving metadata
WSGIDaemonProcess LDRMetadataServer threads=15 user=ldr group=ldr python-path=/opt/lscsoft/glue/lib64/p

# one process with 15 threads for serving data
WSGIDaemonProcess LDRDataFindServer threads=15 user=ldr group=ldr python-path=/opt/lscsoft/glue/lib64/p

# matches for LDRMetadataServer
WSGIScriptAliasMatch /LDR/services/data/v1/pset.* /var/www/html/LDR/LDRMetadataServer/LDRMetadataServer

# matches for LDRDataFindServer
WSGIScriptAliasMatch /LDR/services/data/v1/gwf.* /var/www/html/LDR/LDRDataFindServer/LDRDataFindServer.
WSGIScriptAliasMatch /LDR/services/data/v1/sft.* /var/www/html/LDR/LDRDataFindServer/LDRDataFindServer.

# load LDRDataFindServer on startup
WSGIImportScript /var/www/html/LDR/LDRDataFindServer/LDRDataFindServer.wsgi process-group=LDRDataFindSer

Listen YOUR_PUBLIC_IP_ADDRESS_HERE:443
<VirtualHost YOUR_PUBLIC_IP_ADDRESS_HERE:443>

ServerName YOUR_PUBLIC_FQDN_HERE:443

ErrorLog logs/ssl_error_log
TransferLog logs/ssl_access_log
LogLevel debug

SSLEngine on
SSLProtocol all -SSLv2
SSLCipherSuite ALL:!ADH:!EXPORT:!SSLv2:RC4+RSA:+HIGH:+MEDIUM:+LOW
SSLCertificateFile /etc/pki/tls/certs/localhost.crt
SSLCertificateKeyFile /etc/pki/tls/private/localhost.key
SSLCertificateChainFile /etc/pki/tls/certs/server-chain.crt

```

```

SSLCertificatePath /etc/grid-security/certificates
SetEnvIf User-Agent ".*MSIE.*" \
    nokeepalive ssl-unclean-shutdown \
    downgrade-1.0 force-response-1.0

<Directory /var/www/html/LDR>
Options -Indexes
</Directory>

<Directory /var/www/html/LDR/LDRMetadataServer>

SSLVerifyClient require
SSLVerifyDepth 10
SSLOptions +StdEnvVars +ExportCertData

WSGIProcessGroup LDRMetadataServer
WSGIApplicationGroup %{GLOBAL}

Options -Indexes

</Directory>

<Directory /var/www/html/LDR/LDRDataFindServer>

SSLVerifyClient require
SSLVerifyDepth 10
SSLOptions +StdEnvVars +ExportCertData

WSGIProcessGroup LDRDataFindServer
WSGIApplicationGroup %{GLOBAL}

Options -Indexes

</Directory>

</VirtualHost>

```

Restart or start Apache httpd and verify that the LDRDataFindServer is able to respond to data find requests on port 443 on the public network interface.

6.3 Configuring virtual host on private network

6.3.1 Debian

The recommend approach for Debian users to configure a virtual host for the private network that will listen on port 443 and not require authentication is to create the file `/etc/apache2/sites-available/ldr-internal` with the following contents:

```

Listen YOUR_PRIVATE_IP_ADDRESS_HERE:80

<VirtualHost YOUR_PRIVATE_IP_ADDRESS_HERE:443>
    ServerAdmin webmaster@localhost
    ServerName YOUR_PRIVATE_ALIAS_HERE

    DocumentRoot /var/www/
    <Directory /var/www/>
        Options -Indexes
        AllowOverride None
        Order allow,deny

```

```

        allow from all
    </Directory>

    ErrorLog /var/log/apache2/ldr_internal_error.log

    LogLevel warn

    <Directory /var/www/LDR>
    Options -Indexes
    </Directory>

    <Directory /var/www/LDR/LDRMetadataServer>

    WSGIProcessGroup LDRMetadataServer
    WSGIApplicationGroup %{GLOBAL}

    Options -Indexes
    </Directory>

    <Directory /var/www/LDR/LDRDataFindServer>

    WSGIProcessGroup LDRDataFindServer
    WSGIApplicationGroup %{GLOBAL}

    Options -Indexes
    </Directory>
</VirtualHost>

```

Replace `YOUR_PRIVATE_IP_ADDRESS_HERE` with the IP address for the private network interface. *Do not use an alias for this option. You must use the numerical IP address.* Also replace `YOUR_PRIVATE_ALIAS_HERE` with the alias for the private IP address.

Next Debian users should run

```
a2ensite ldr-internal
```

to enable the virtual host configuration.

6.3.2 CentOS and SL

CentOS and SL users should again edit the file `/etc/httpd/conf.d/ldr.conf` and *append* to the end of the file the following:

```

Listen YOUR_PRIVATE_IP_ADDRESS_HERE:80
<VirtualHost YOUR_PRIVATE_IP_ADDRESS_HERE:80>

    ServerName YOUR_PRIVATE_ALIAS_HERE:80

    ErrorLog logs/internal_error_log
    TransferLog logs/internal_access_log
    LogLevel debug

    <Directory /var/www/html/LDR>
    Options -Indexes
    </Directory>

    <Directory /var/www/html/LDR/LDRMetadataServer>

    WSGIProcessGroup LDRMetadataServer
    WSGIApplicationGroup %{GLOBAL}

```

```
Options -Indexes
</Directory>

<Directory /var/www/html/LDR/LDRDataFindServer>

WSGIProcessGroup LDRDataFindServer
WSGIApplicationGroup %{GLOBAL}

Options -Indexes
</Directory>
</VirtualHost>
```

Replace `YOUR_PRIVATE_IP_ADDRESS_HERE` with the IP address for the private network interface. *Do not use an alias for this option. You must use the numerical IP address.* Also replace `YOUR_PRIVATE_ALIAS_HERE` with the alias for the private IP address.

6.4 Configure LDRDataFindServer

With Apache httpd configured to listen on multiple virtual hosts the next step is to configure LDRDataFindServer to use the appropriate authorization for each virtual host.

Edit the file `/etc/LDR/ldr.ini` and change the `LDRDataFindServer` section to read

```
[LDRDataFindServer]
<other options here>

authorization = virtual_host

[[YOUR_PUBLIC_IP_ADDRESS]]
authorization = grid-mapfile

[[YOUR_PRIVATE_IP_ADDRESS]]
authorization = None
```

You must substitute the actual public and private IP addresses for your deployment. *Do not use FQDN or aliases—you must use numerical IP addresses.*

Start or stop Apache httpd and test to make sure that LDRDataFindServer is again responding appropriately, now on port 443 (requiring proxy or X.509 certificate authentication) and on port 80 on the private network address without any authentication.

Chapter 7

LDR As a Service

Not every site that wishes to replicate data locally is able to provide enough administrative effort for a full LDR deployment. As an alternative a site may choose to consume “LDR as a service”. In this configuration a site only needs to deploy a GridFTP server to expose the file system that will host the replicated data. No other parts of LDR are deployed locally. Rather the other parts of LDR (including Globus RLS, the Apache web services, and the LDR daemons) are deployed and managed by the LDR team at the University of Wisconsin-Milwaukee.

If you are interested in replicating interferometer data using the LDR service model please contact the LDR project manager to discuss the possibilities and requirements on the local site.

7.0.1 Deploying GridFTP for the LDR service

If your site is consuming LDR as a service and you have been asked by the LDR project manager to deploy the GridFTP server please follow the instructions below.

Installing the GridFTP server

For CentOS or Scientific Linux please follow these instructions:

1. Login as the root user.
2. Create the file `/etc/yum.repos.d/lscsoft.repo` with the following contents:

```
[lscsoft]
name=LSC Data Analysis Software
baseurl=http://www.lsc-group.phys.uwm.edu/daswg/download/software/centos/5/$basearch
enabled=1
gpgcheck=0
```

3. Install the GridFTP server package for use by LDR by doing

```
yum install ldr-globus-gridftp
```

Debian users can use `aptitude` to download and install the GridFTP server package for use with LDR. Note that aptitude will complain when installing packages from an “untrusted” source. In order to allow aptitude to verify the authenticity of these packages, you need to add the keyring of the LIGO LSCSoft repository. The simplest way to do this is to simply install the package `lscsoft-archive-keyring` via aptitude (insisting only once that you trust this repository):

```
aptitude install lscsoft-archive-keyring
```

Next add the LSCSOFT Debian repository to the `/etc/apt/sources.list` file by adding the line
`deb http://www.lsc-group.phys.uwm.edu/daswg/download/software/debian/ lenny contrib`

Then run

```
aptitude update
```

followed by

```
aptitude install ldr-globus-gridftp
```

Configuring the GridFTP server

Most configuration details for the GridFTP server are set as defaults during the installation. You can review them in the file `/etc/ldr-globus-gridftp/gridftp.conf`. To complete the deployment, however, you must

1. obtain and properly deploy credentials for the server
2. create at least two accounts, one for writing and one for reading
3. create a file to authorize connections by the LDR tools to the server

Host credentials for the GridFTP server

The Globus GridFTP server requires a X.509 digital certificate for mutual authentication with clients. The GridFTP server usually runs as `root` and so will require a single copy of the host certificate and its associated private key on the machine where both files are owned by `root` and in group `root`.

Please follow the instructions for your organization on how to obtain host or server X.509 certificates. Typically the host certificate is put into a file named `hostcert.pem` and the associated private key is put into a file name `hostkey.pem`.

Move the file `hostcert.pem` and `hostkey.pem` into the directory `/etc/ldr-globus-gridftp` and make sure that both are owned by `root` and in group `root`. Also make sure that the file `hostcert.pem` has permissions `0644` and `hostkey.pem` has permissions `0400`.

Create writing and reading accounts

When the LDR client tools connect to your GridFTP server they are mapped to a UNIX local account and access rights are handled using standard UNIX file permissions. You should therefore create two UNIX accounts (neither need have login permissions but they should have a reasonable home directory):

1. an account to be used for writing data onto the filesystem. The convention is that this account is named `ldr` and is in the group `ldr`, but any account name is acceptable. The account should have write access to the directories where data is to be stored on the file system but nothing more.

Please do not use the root account for this purpose.

2. an account to be used for reading data so that other sites may replicate data from your site. While this is not strictly necessary it is suggested. The account should have read-only access to the directories where data is to be stored and nothing more. It should not be in the same group as the account with write access.

The convention is that this account is named `readrobot` and is in the group `readrobot`.

Grid-mapfile for the GridFTP server

The GridFTP server checks the credentials for clients that connect to it and maps the clients to a local UNIX account using a grid-mapfile. The LDR project manager should provide you the subject name from a certificate that the LDR client tools will use to contact your GridFTP server. Create the file `/etc/ldr-globus-gridftp/grid-mapfile.gridftp` and add the subject name provided by the LDR project manager, in double quotes, followed by the local UNIX account for writing.

For example:

```
"/DC=org/DC=doegrids/OU=Services/CN=datarobot/ldr-birmingham.phys.uwm.edu" ldr
```