

A Data Placement Service for Data Staging

Scenarios, Interfaces, and Design

WORKING DRAFT

Introduction

A data placement service is envisioned to support a variety of policy-based placement requests. Such policies may involve N-copy replication to ensure data durability, staging of data sets from Storage Elements to compute clusters, staging out the results from compute clusters back to Storage Elements, hierarchically tiered distribution as in the case of tiered topologies used by high energy physics, and several other possibilities. The scope of the present design document addresses one specific instance of a data placement policy – staging data from Storage Elements to compute clusters in support of workflow execution.

The first section of this document presents a brief overview of the data staging problem. The second section describes the key usage scenarios for data staging. The third section presents abstract interfaces to be exposed by a data placement service. The final section discusses a high-level system design for the data placement service.

Data Staging Overview

Without support from a third party data placement service, workflow planners and other compute cluster clients must manage data staging activities as required by their submitted jobs. In the case of a workflow planner, this may involve embedding data staging tasks into the workflow. For instance, a task may be specified in the workflow with the responsibility of managing a third-party GridFTP transfer from a Storage Element to the compute cluster. Such tasks burden the workflow execution with creating specialized data staging jobs and consuming cycles of the compute cluster's resources in order to execute the data staging jobs.

A data placement service supports workflow planners and workflow execution by offloading the tasks of staging data to compute clusters. Such tasks may include locating originals and replicas of desired data, scheduling transfers between multiple Storage Elements and multiple compute clusters, monitoring data transfers and correcting errors when and where possible, and registering staged data in local replica catalogs.

Workflow planners prepare workflow tasks and identify data sets required for computation. The set of workflow tasks may span more than one compute cluster (though for simplicity a single compute cluster is depicted throughout this document). The planner submits a placement request to a data placement service which performs the staging asynchronously. Some tasks may be ready to execute before the placement service has staged the entire data set. In such cases, the workflow planner or a special job inserted into the workflow contacts the placement service and instructs it to increase the priority for staging the required data items.

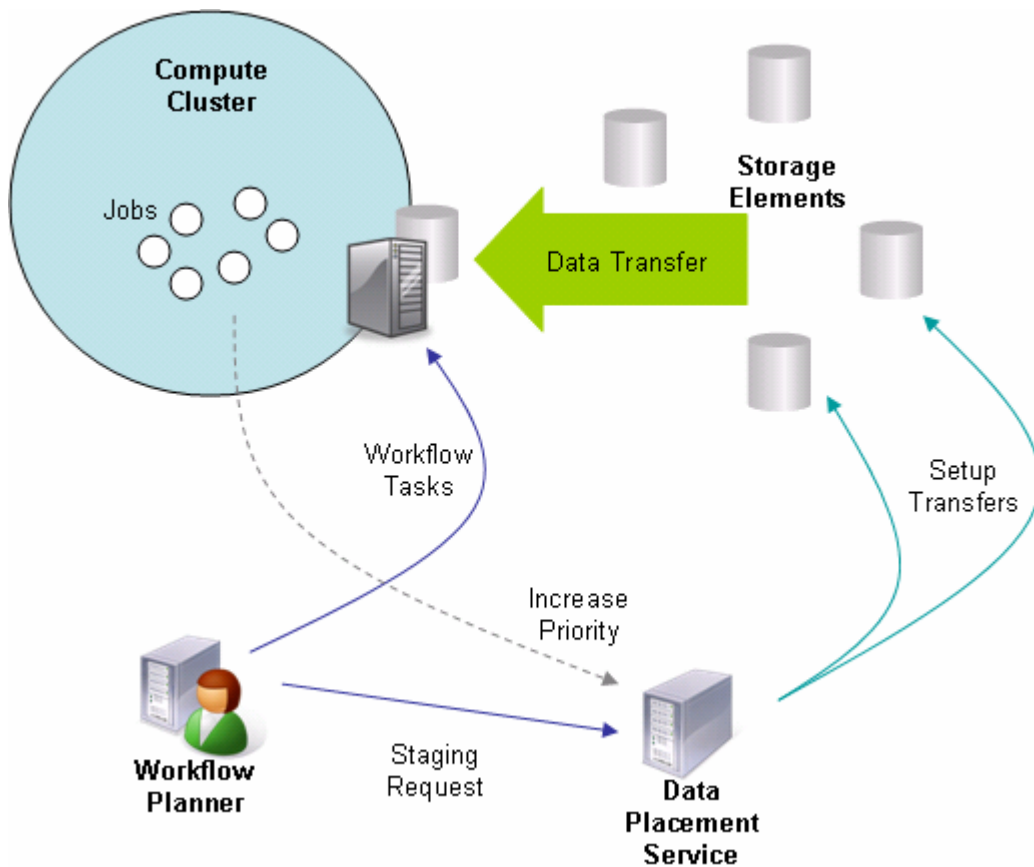


Figure 1. Overview of Data Placement Service

Scenarios

In the present scope of this document, two usage scenarios are presented – staging and prioritization.

Data Staging

In the data staging scenario, the service accepts a staging request from a client. It is anticipated that the client may be some form of a workflow planner. The workflow planner, or other client, will have determined a set of jobs to be executed, the compute cluster with which to execute the workflow, and the data required for each job in the workflow. As depicted in figure 1, the workflow planner submits the workflow tasks to the job manager or DAG manager of the compute cluster while it submits the staging request to the data placement service.

The staging scenario begins for the data placement service upon receipt of the staging request. First, the service must determine how much of the requested data already exists at the designated compute cluster. In a typical configuration, a compute cluster is coupled with a data store (e.g., a “NAS box”) to which job related data is staged. Internally, the data placement service maintains a catalog of data files presently located at each compute cluster. Next, the service locates the remaining files to be staged-in from other storage

elements. After locating the remaining data files, the service may need to select among replicas and then initiate and monitor data transfers between storage elements and the compute cluster's local data store. Once the transfers are complete the service internally records the locations of the newly created replicas.

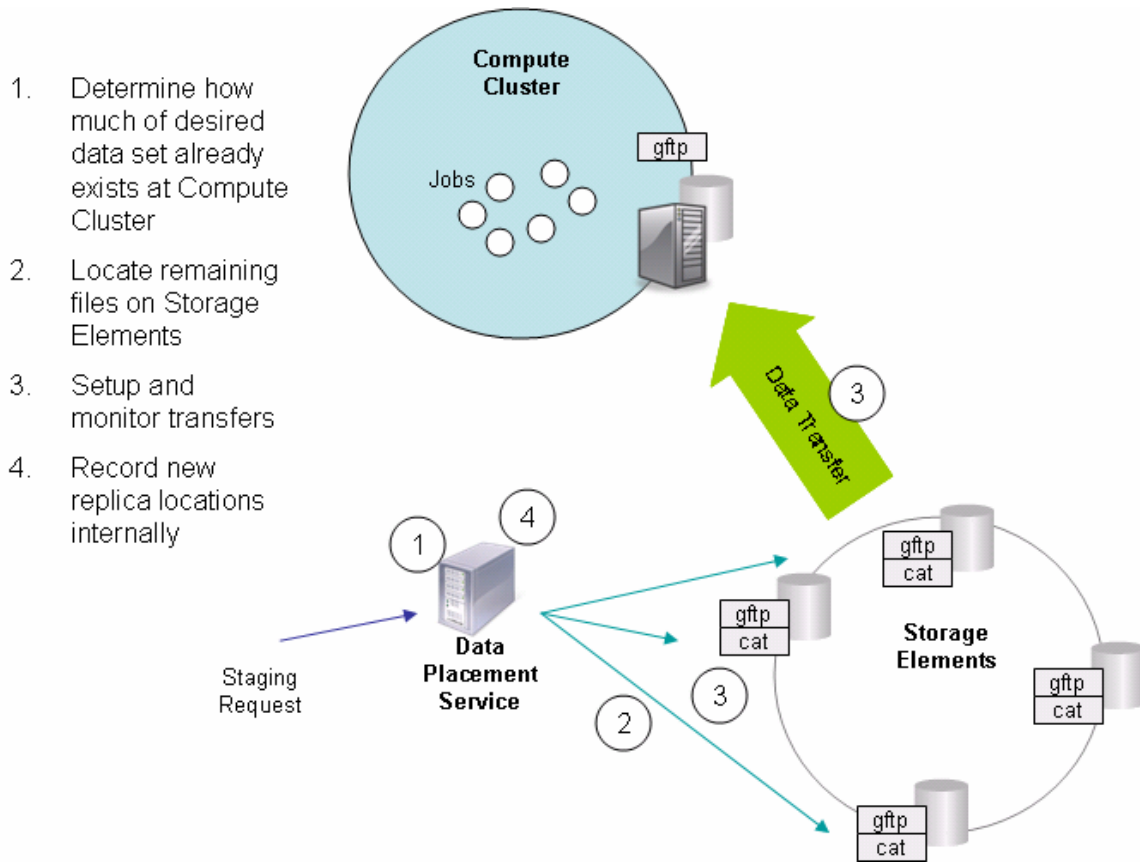


Figure 2. Pre-staging scenario

Prioritization

The compute cluster may be ready to execute some jobs before the data placement service has been able to pre-stage all of the required data. In the Prioritization scenario, a client makes a prioritization request to the data placement service to increase the priority for the staging-in of a data file. The client of such a request may be a special job inserted into the workflow by the workflow planner with the specific purpose of determining whether a data file is properly staged and making a request in the event that the data file is not yet available. Alternatively, the workflow planner or DAG manager could make such a request (not depicted).

The prioritization scenario begins for the data placement service when a client submits a prioritization request. The service identifies the current state of the staging activity for the data file. Then the service increases the priority for the staging request in its priority

queue to move it ahead of normal (non-prioritized) staging requests. In general, the service continues the rest of its ongoing staging activities.

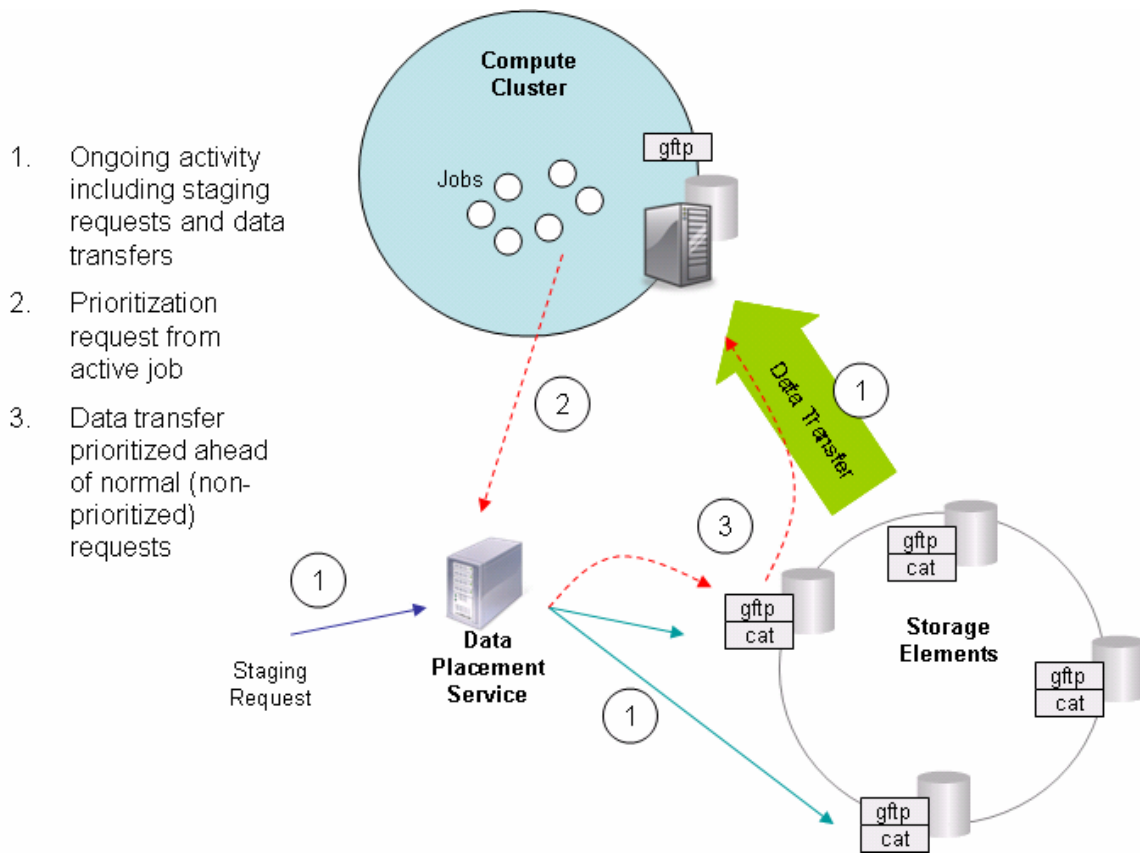


Figure 3. Prioritization scenario

Interfaces

The service supports interfaces to create, modify, and destroy placement requests; to start and cancel placement execution; and to query request state. The following section describes the interfaces without binding the interfaces to a specific protocol or language.

Create Placement Request

The client issues a create placement request command to the data placement service in order to create and specify the details of the request, such as the compute cluster on which to stage the data and the set of files to be staged.

- Command:
 - CreatePlacementRequest
- Precondition(s):
 - Client specifies the placement request, which includes the type of policy (e.g., “data stage-in”) and the details of the request (e.g., “stage XYZ data set to compute clusters A, B, and C”).

- Data placement service is running and initialized
- Data placement service supports the desired policy (e.g., “data stage-in”)
- Postcondition(s):
 - Request accepted or rejected by the data placement service
 - Identifier created for use by client to reference the request
- Parameter(s):
 - `PolicyType`: The type of placement policy. In the broader scope (outside the scope of this document), valid policy types may include n-copies, push-based replication, pull-based replication, and others. For the purposes of this document, only the “data stage-in” policy is addressed.
 - `RequestDetails`: Details of the placement request. For a request using the “data stage-in” policy type, the desired data set (using logical identifiers) is expected along with the destination storage sites (typically attached storage at compute clusters). In addition, the request may also include *hints* provided by the client such as the grouping and/or ordering on the data set items. OPEN ISSUE: We need to determine what types of hints a workflow planner or other client can provide to the placement service.

Modify Placement Request

The client issues a modify placement request command to the data placement service in order to modify some aspect of the request details. For instance, in the case of a data staging policy, the policy may allow *re-prioritization* of individual file transfers in order to support a pending job. Also, in the case of data staging, the policy may allow for *data items to be added* to the request. In this way it is possible for a client to create the request with a subset of high priority data files specified, to start the placement execution, and then to add lower priority data files to the request.

- Command:
 - `ModifyPlacementRequest`
- Precondition(s):
 - Client must reference an existing placement request
 - Client specifies changes to the existing request
- Postcondition(s):
 - Placement service accepts or rejects request modifications
- Parameter(s):
 - `Identifier`: An identifier to an existing placement request
 - `ModificationDetails`: Modifications to the placement request details. In the case of data staging, this may involve *prioritization* of certain data files in the request, or the *addition* of new data files to the request.

Destroy Placement Request

The client issues a destroy command in order to remove a request by implicitly cancelling it if it had been started and by deleting any of its state information from the service.

Destroying a request differs from cancelling a request in that not only are the request's operations terminated but the request's state information is removed from the service permanently and is no longer available to the client in subsequent operations.

- Command:
 - `DestroyPlacementRequest`
- Precondition(s):
 - Client must reference an existing placement request
- Postcondition(s):
 - If the request was previously started but not yet terminated, the request is cancelled as per the cancel command (see Cancel command)
 - State information for the request is deleted from the service and no longer accessible
 - Request identifier is no longer valid
- Parameter(s):
 - `Identifier`: An identifier to an existing placement request

Start

The client issues a start command in order to begin execution of an existing placement request. The start command may also be used to re-start a request that was suspended due to transient failures.

- Command:
 - `Start`
- Precondition(s):
 - Client must reference an existing placement request
 - Request must be inactive
- Postcondition(s):
 - Request is activated
 - Request status updated to "active"
- Parameter(s):
 - `Identifier`: An identifier to an existing placement request

Cancel

The client issues a cancel command in order to terminate the activities of an active placement request. Cancelling a request differs from destroying a request in that a cancelled request's activities are terminated but the state information is retained and may be referenced by the client in subsequent operations.

- Command:
 - `Cancel`
- Precondition(s):
 - Client must reference an existing placement request
 - Request must be active
- Postcondition(s):

- Request is terminated
- Request status updated to “terminated”
- Parameter(s):
 - Identifier: An identifier to an existing placement request.

Query

The client issues a query command in order to retrieve some aspect of the request state information. This command may be used by clients, such as jobs in a workflow, in order to test whether a file transfer is complete or still in-progress. (In a WS-Resource implementation, the query command would be implemented by the `QueryResourceProperties` interface with standard or possibly custom query dialects.)

- Command:
 - Query
- Precondition(s):
 - Client must reference an existing placement request
- Postcondition(s):
 - No changes
- Parameter(s):
 - Identifier: An identifier to an existing placement request.
 - Expression: A query expression is used in order to specify the request details of interest. For instance, an expression may be used to retrieve:
 - the overall status of a placement request
 - the individual status of one or more data files in a placement request

Design

This section presents a system level design of the Data Placement Service in the context of a potential configuration of a compute cluster and storage elements. The design diagram (figure 4) depicts the Data Placement Service, storage elements, and a compute cluster along with some of the representative relationships between their sub-components.

Data Placement Service

The Data Placement Service may be deployed on a standalone node in a Grid or may be co-deployed with other services so long as resource utilization permits. In its initial release the Data Placement Service (proper) may be developed as a WS-Resource and therefore will depend on the Globus WS Core. In addition to Core, the service may depend on an embedded database to durably maintain state information for staging policies. The service will also depend on a co-deployed RLS server (“embedded” in some sense) and use its RLI and LRC services to track the location of staged data files on compute clusters.

Data Placement Service (WS-Resource)

The Data Placement Service, if initially developed as a WS-RF-compliant web service, will expose a WS-Resource. A WS-Resource would represent each staging policy and its state information. Resource Properties may include the overall status of the policy (i.e., initialized, activated, terminated, etc.), the overall result of the policy (i.e., finished, finishedWithErrors, failed, cancelled, etc.), the location of the destination compute cluster, and detailed status of each file in the staging policy.

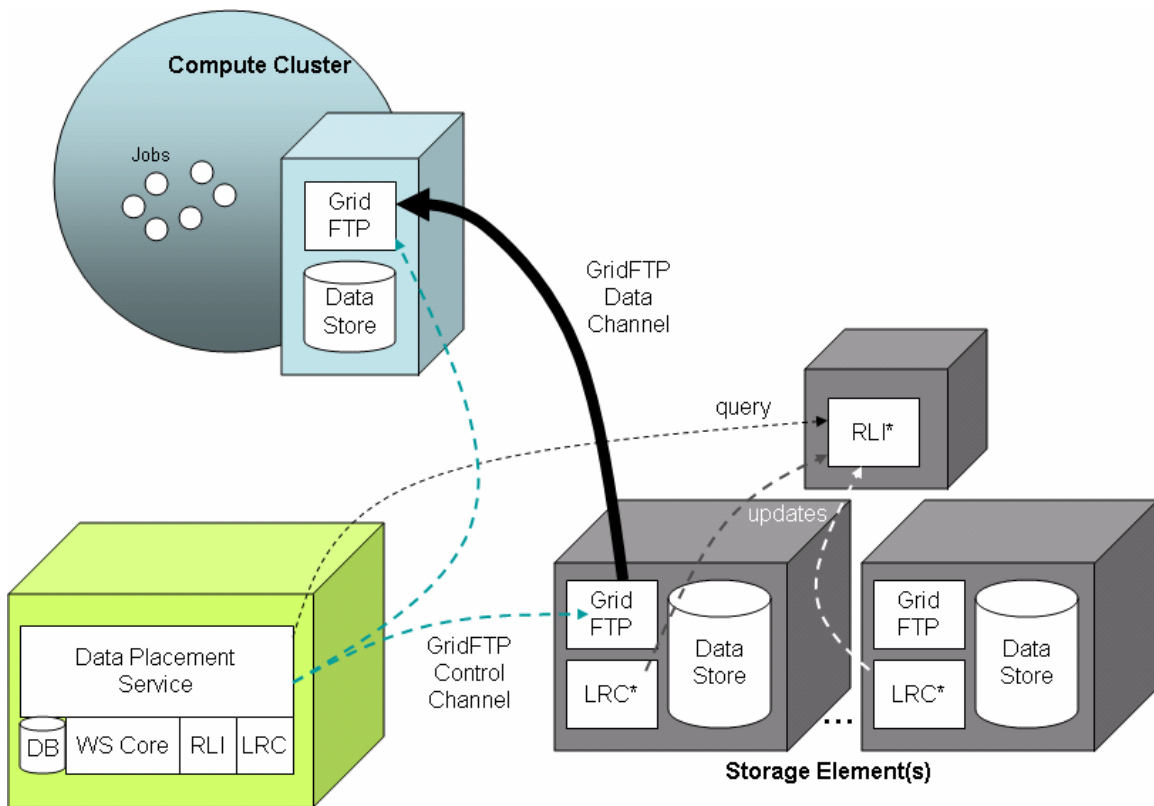


Figure 4. System-level design

WS Core

The Globus WS Core, presumably the Java version, will provide the basic infrastructure for supporting web service deployment and generic operation support such as basic state management, query operations, endpoint references, and so on.

Embedded Database

An embedded database will be used to maintain internal state information persistently and durably (if required). One of the existing Globus database projects may be used for this purpose, such as the Globus package for the Derby (Java) database or the SQLite (C/C++) database. These databases store data in as little as a single file which can be backed up easily for additional robustness.

Replica Location Service (RLI + LRC)

The service will depend on a co-deployed (almost “embedded”) RLS with its RLI and LRC services. The Data Placement Service will use its internal RLS to persistently store the locations of staged-in files at various compute clusters. The internal RLS will *not* be used to maintain the location of files at the storage elements. If necessary, “power users” may deploy RLS-LRC services at each compute cluster to catalog the contents of each cluster’s attached storage device and then send updates to the RLS-RLI that is co-deployed with the Data Placement Service. In this way, users may scale up their cluster-related catalogs without changes to the Data Placement Service.

Storage Elements

Storage Elements are expected to maintain the majority (the entirety?) of data used by an application. Storage elements may be configured in a number of ways and may rely on multiple services, such as storage resource management services (not depicted), outside the scope of the present design. The two key services required for interaction with the Data Placement Service include a transfer service and a file or replica catalog.

Transfer Service

A transfer service is required to move data from storage elements to compute clusters’ attached storage devices. Support for 3rd party transfers as provided by GridFTP services would be an essential capability. While the Globus GridFTP Server is one such instance of a transfer service, the Data Placement Service will provide callouts (“hooks”) to support alternative transfer services. The default deployment will include support for a TBD set of transfer services (for instance, SRM, RFT, Stork, GridFTP Server, ...).

File or Replica Catalog

The locations of data stored on Storage Elements in the Grid must be exposed in some form of a file or replica catalog. Depicted in the diagram (figure 4), a combination of LRC and RLI services provide a distributed replica location service. While the Globus RLS is one such instance of a file or replica catalog, the Data Placement Service will provide callouts (“hooks”) to support alternative catalogs. The default deployment will include support for Globus RLS.

Compute Cluster

Compute Clusters execute computational jobs and provide resource management services to clients that submit jobs. They are typically configured with an attached storage device to which data are staged prior to execution of a client’s job.

Transfer Service

A transfer service is required to receive data transferred from storage elements.