Journal of Physics: Conference Series **219** (2010) 062060 doi:10.1088/1742-6596/219/6/062060

Dealing with orphans: catalogue synchronisation with SynCat

A Paul Millar[1], Flavia Donno[2], Jens Jensen[3], Shaun De Witt[3], Giuseppe Lo Presti[2]

[1] Deutsches Elektronen-Sychrotron (DESY), Notkestraße 85, 22607 Hamburg, Germany
[2] European Organization for Nuclear Research (CERN), CH-1211, Genève 23, Switzerland
[3] Science and Technology Facilities Council (STFC), Rutherford Appleton Laboratory, Didcot OX11 0QX, United Kingdon

E-mail: paul.millar@desy.de

Abstract. In the gLite grid model a site will typically have a Storage Element (SE) that has no direct mechanism for updating any central or experiment-specific catalogues. This loose coupling was a deliberate decision that simplifies SE design; however, a consequence of this is that the catalogues may provide an incorrect view of what is stored on a SE. In this paper, we present SynCat: a mechanism to allow catalogues to re-synchronise with SEs. The paper describes how catalogues can be sure, within certain tolerance, that files believed to be stored at various SEs are really stored there. SynCat also allows catalogues to be aware of transitory file metadata (such as whether a file normally stored on tape is currently available from disk) with low latency.

1. Introduction

A data grid is a geographically distributed set of computing resources used for processing quantities data currently up to tens of petabytes. The data, computing resources and end-users are all distributed over many countries.

The Large Hadron Collider (LHC) facility is based at CERN and hosts four detectors. Four experimental collaborations have been established, each analysing the data from one of these detectors. To satisfy the computational and storage requirements of these collaborations the World-wide LHC Computational Grid[1] (WLCG) has been established and is currently the largest existing data grid, spanning the globe and utilising resources provided by hundreds of institutes.

To provide a succinct description of the issues and how SynCat addresses them some domain-specific nomenclature is introduced:

Site A grid is composed of many sites. Each site provides a collection of computational and storage resources that are locally managed. Typically, resources provided by a site will be under a common management structure that is usually distinct from the management of other sites.

Storage Element The storage resources provided by a site, the software and hardware, are managed by storage management software. There are different implementations of this software but together they are referred to as storage elements (SEs).

The storage element provides WAN access to allow data movement from site to site. It also provides LAN access to allow locally running computation activity to process data.
For larger sites, the storage element must also manage the flow of data to and from tape storage: archiving incoming data on tape and, when necessary, triggering the restoration of data back from tape.

File replicas Replicas are identical copies of the file. Requests for data stored in the file can be satisfied with any replica of the file. Replicas allow for high availability of data: if one replica is unavailable then applications may use an alternative replica. Also, an application may be able to optimise its data access by selecting a replica that is somehow "best".
A storage element may have multiple internal copies of the same file stored within the hardware it manages. These duplicates are often referred to as internal replicas. Although these replicas can cause issues, we consider such internal replicas to be outside the scope of this paper.

Replica Catalogue A replica catalogue is a service that records in which SEs the different replicas are located. When a client wishes to read some data it may query a replica catalogue to discover the possible locations of some given file. Given this list of available replicas the client may choose the replica that is likely to give the lowest latency. More often, the location of replicas is used to steer user activity to sites that have the required data locally.
The WLCG project provides a centralised catalogue facility: the LFC[2]. This catalogue may be used by end-users to record where replicas are located. It can also be used to identify candidate sites for running jobs that require certain data.
In addition to the LFC, some end-user groups have chosen to implement their own file catalogue[3]. These catalogues operate independently from the LFC service. This allows additional end-user-specific metadata to be stored and tighter integration with their data-flow models.

File-aware Component A file-aware component is a generic term for either an SE or a replica catalogue. It is useful to group these two components together under a common term since catalogue synchronisation can be regarded as a symmetric operation between a storage element and a file catalogue: an inconsistency between a storage element and a file catalogue is also an inconsistency between a file catalogue and a storage element.

FTS The File Transfer Service[4] (FTS) provides a service that allows end-user groups to transfer multiple files between sites. It controls its usage of limited resources (storage element connectivity, bandwidth, etc) so storage elements do not suffer from resource starvation and good progress is made.
FTS also provides error detection and retry mechanisms. If an error occurs then the failed transfers are scheduled for retry. In this way, transient errors do not affect the success of a batch of transfers.

In general, the work-flow of the LHC experiments will result in the same data appearing at multiple SEs, located at different sites. These replicas are intentional: they provide assurance against data-loss and improve computer utilisation at the expense of decreased overall storage capacity. The precise distribution of replicas is experiment-specific and may change over time.

In brief, the problem SynCat aims to address is that, files stored within some SE may suffer a state change outside of the end-users' control. This can happen if a file is lost or discovered to be corrupt. Another example is if a file's transient metadata changes; e.g., a file is staged from tape onto disk.

Since this state change is outside the end-users' control, the end-users are unable to update the appropriate file catalogues to match. Due to the decoupled design in WLCG, the SE is also unable to notify any file catalogue. Over time, the file catalogues will have increasingly inaccurate view of where data is located. This will likely decrease efficiency of computer utilisation when

analysing the data: missing data must be fetched from remote sites or the analysis activity rescheduled elsewhere. It will also increase the likelihood of data-loss.

The SynCat proposal is to resolve this issue by introducing a coupling between the SE and the file catalogue via some unspecified external agent. The agent is provided with sufficient information to allow it to detect when inconsistencies have arisen. This agent may be human-driven, fully autonomous or some combination and the actions the agent may take to correct identified problems is outside the scope of this paper.

The paper has the following structure: the next section provides a more detailed description of the synchronisation problem and section 3 outlines what solutions are currently available. Section 4 provides a description of SynCat, both the data structure and the processing model and section 5 gives a description of current status of deployment. Section 6 gives an overview of other work in this area and the paper concludes with section 7.

2. The Problem

A file catalogue records, for each file, the set of storage elements that store a replica of that file's data. There may be several file catalogues with some knowledge of a given storage element. Under the WLCG architecture, a storage element has no knowledge of which file catalogues have registered their files, nor of which files are actually registered in those catalogues. This is deliberate: it decouples the storage element from the catalogues, so simplifying the design.

However, because of the decoupling between SE and file catalogue, any mechanism to reach and maintain an agreement between the different file-aware components must happen outside the storage element and file catalogues.

2.1. Types of inconsistency

When considering potential inconsistencies, it is useful to classify them into the following four main groups:

SE-absent replicas are replicas that are registered in the catalogue but absent from the corresponding storage element. These inconsistencies may have arisen from storage-element intervention; for example, recovering from some hardware failure. These inconsistencies can also occur if a replica is registered in a file catalogue before writing the file, that subsequent data transfer fails. They can also exist when a replica with a finite lifetime is garbage-collected by the storage element.

SE-absent replicas are potentially dangerous to the end-user groups since they give a false impression of the number of existing replica copies. If the end-user group manages the number of replicas, it may delete data elsewhere leading to the number of replicas falling below the desired number, so risking data loss.

catalogue-absent replicas are replicas that are present in the storage element but absent in the file catalogue, also known as 'dark data'. They can occur when a file's data is successfully transferred between sites but the subsequent catalogue registration of the new replica failed. These inconsistencies can also occur from local intervention within the catalogues.

The end-user group is unaware of these replicas since they are not registered in the catalogue, but they consume capacity allocated to the end-user group. In the absense of any catalogue synchronisation, it is impossible to detect the presence of dark data.

inconsistent-info replicas are replicas present in both the storage element and the catalogue; however, the information held by the file catalogue is conflicting with the information from the storage element. These errors can occur if information stored in the file catalogue changes over the lifetime of the replica.

missing-info replicas are replicas that are both registered in the file catalogue and stored in the storage element; however, the file catalogue does not hold all the metadata needed. This

can happen if the information changes over the lifetime of the replica and the catalogue has no means of discovering the replica's current state.

Both inconsistent-info and missing-info replicas can arise due to information changing over the lifetime of a replica. An example of state that may change over time is the occupancy of a replica stored on tape. When the replica's data is recalled from tape or the last disk-based cache copy of the replica is removed then the replica's occupancy will change. This affects the latency Quality of Service (QoS) that the Storage Element will provide: requests to read the replica will take longer if the file is only available from tape.

Higher-level scheduling agents may wish to run jobs on sites that provide files with low latency QoS. This requires the file catalogues to maintain metadata about which files are quickly available, which, in turn, requires synchronisation of this information from the storage element.

Although all four inconsistancies can arise, this paper will focus on SE-absent, catalogue-absent and missing-info replica inconsistencies.

3. Possible approaches with existing software

With the currently available mechanisms, the synchronisation agent can only discover SE-absent replicas by scanning over the complete list of catalogue entries, looking for those replicas that are absent in the corresponding storage element. Likewise, to discover catalogue-absent replicas, the agent must scan over the relevant portion of the storage element namespace, looking for entries that are absent in the file catalogue.

If the client has sufficient resources, it may cache the scans to allow detection of SE-absent replicas and catalogue-absent replicas with a single scan of the file catalogue and storage element respectively. This would allow more efficient processing and reduce the impact on the storage element and catalogue.

Alternatively, the agent may operate so it discovers only one of the two classes of absent replica inconsistencies. This would require the synchronisation agent to scan over only the storage element or catalogue but not both. However, this introduces the risk that some catalogue inconsistencies are missed.

3.1. Problems with currently available options

There are currently two mechanisms for querying a storage element's namespace: using the File Transfer Protocol[5] (FTP) or the SRM[6] `ls` command. FTP has two extension commands[7] `MLst` and `MLsD` (hereafter referred to as `MLsx`) that allow the SE to send arbitrary file metadata in a well-formatted fashion. The SRM `ls` command also provides structured information about files, allowing external synchronisation.

Whilst these two mechanisms for detecting inconsistencies are functional; they both suffer from a set of three problems:

First, the act of collecting information from the file-aware component (storage element or file catalogue) is under the control of the synchronisation agent. There may be several synchronisation agents concurrently attempting to achieve agreement between some storage element and their file catalogue. The site may have no means to schedule these agents' activities to minimise their impact on the storage element.

Second, the process of gathering information uses existing mechanisms (FTP `MLsx` or SRM `ls`) that were not designed to facilitate synchronisation. Although these approaches may provide sufficient data, there may exist more optimal approaches if one wishes to scan the complete namespace. Allowing a storage element to implement a more optimal process would reduce the impact on the storage element, so allow more frequent dumping of the namespace.

Finally, scanning a complete namespace is an expensive operation. This limits how often a file-aware component can sustain a synchronisation agent scan. In contrast, an additional service

that publishes changes to the storage element's namespace would provide a more light-weight approach. This, in turn, would allow much faster propagation of information to synchronisation agents so reducing the latency between storage element and file catalogue.

4. SynCat

The SynCat project[8] aims to overcome the limitations described in section 3. It does this by exposing the namespace of the storage element to allow an external synchronisation agent to discover inconsistencies and take appropriate action.

It is worth emphasising that what actions the synchronisation agent takes is outside the scope of SynCat. The agent may log the inconsistency, alert some operator to fix the inconsistency manually, or it may act autonomously and fix problems as they are discovered. The model employed to achieve this is under end-user group's control and different groups may choose different strategies.

The SynCat model provides two modes of operation: full and incremental. The incremental mode is an enhanced version of the full mode: a file-aware component will support either the full mode or both the full and incremental modes.

The full mode is the easist to implement and also the easiest for the synchronisation agent to process but may lead to relatively long periods where inconsistencies are suffered. With this mode, a synchronisation agent receives a complete dump of the namespace. This information is created by the site and the process of generating the data is under their control: a site can adjust the scheduling of this activity to minimise the impact on their production system. Since dumping the namespace is an expensive operation it cannot be done frequently. Therefore, full mode cannot provide rapidly changing information and full mode is likely limited to solving catalogue-absent and SE-absent replica inconsistencies.

The incremental mode builds on the full mode and aims to reduce the latency. It achieves this by providing a sequence of change-set data, each of which describes the changes within the storage element's namespace within a certain time window. It is anticipated that the overhead in producing this information be far less than the overhead in providing a complete dump; therefore, the change-set data may be provided far more often; allowing synchronisation agents to provide synchronisation checks with much finer granularity.

Although we have presented the two models of operation, the initial focus of the SynCat project has been restricted to how the information is encoded. Other issues, such as the transport, how an synchronisation agent might request SynCat information, authentication of agents, authorisation of different requests, etc, are deferred until experience has been gathered. This is a deliberate policy to allow implementers to focus on the encoding. In the mean time, we anticipate that the other issues will be resolved manually or semi-manually by each site.

4.1. Data Format

The SynCat format allows a file-aware component to express its knowledge in an implementation neutral fashion. It is designed to be extensible, both for future versions of SynCat and to allow the inclusion of information specific to some file-aware component implementation.

SynCat information may be either in 'dump' or 'diff' format. In both cases, the file is an XML file with similar structure between dump and diff format; these formats are described in more detail below.

The format is designed to be extensible. It uses an XML namespace to allow the addition of arbitrary information. In this way the SynCat output may include information that is specific to some file-aware component. Clients that understand this information can process it while clients that lack the understanding can safely ignore the data.

There is a developer's pack that includes an XML Schema for the SynCat file format and a framework for testing SynCat output. This allows anyone to check quickly if some SynCat data

conforms to the specification.

4.1.1. Dump Format In its most minimal form, the dump output provides a simple list of namespace entries or registered replicas for some storage element without any file metadata. However, the dump format also allows the reporting of metadata associated with each replica. This metadata can include implementation-neutral concepts, such as length and checksum, and also arbitrary implementation-specific concepts. It is anticipated that the same set of metadata attributes is reported for each reported replica; however, this is not a requirement.

4.1.2. Diff Format The diff format describes changes to the namespace over a period of time. The time window is anticipated to be short but it may be influenced by the software implementation and the resources available at a site and to the synchronisation agent.

The changes reported by the diff format are grouped into three sections: `created`, `modified` and `deleted`. These sections record new replicas that have been created, those replicas that have had some information change and those that have been removed respectively.

If a replica has changed multiple times over the time window of the diff report then those changes are merged; for example, a file that is created then, within the time window of the report, some information changed would be reported under `created` with information taken from the end of the time window; a file that was created then deleted within the time window is not included.

The same set of replica metadata is reportable under `created` and `modified` sections of the diff format data as is available for reporting replica metadata in the dump format.

4.2. SynCat processing model

As outlined earlier, there are two operational modes in which a file-aware component may provide SynCat information: full or incremental. We can now recast these modes in terms of the interaction between a synchronisation agent and the two available SynCat formats.

With the full mode, the site builds an appropriate portion of the namespace in SynCat dump format. This is done within a schedule controlled by the site so to minimise the impact on their service.

The synchronisation agent receives the SynCat dump information. The site may provide the information directly to the agent, or it may place the data at a standard location. The dump format contains a date reference so the agent can detect if it has already processed the presented data. This allows the agent to poll for new information and process it as the dump data becomes available.

If the file catalogue also exports information as SynCat dump data then the synchronisation agent can search for inconsistencies in a generic fashion; otherwise, the sychronisation agent must use some custom interface to query the catalogue.

The agent can scan over the dump data looking for catalogue-absent, missing-info and inconsistent-info replicas taking the appropriate action when these problems are encountered. It can then scan over the file catalogue entries checking for corresponding entries in the dump format data; this allows the agent to discover SE-absent files.

With the incremental mode a site makes available data in both SynCat dump and diff formats. The dump format data is as before: it is obtained on a schedule controlled by the site and processed by the agent. In addition, the agent receives (typically many) diff format data on a faster timescale, with the diff format reports interleaving the dump data reports.

The simplest model is to update the dump format data based on the received diff format data. This allows the agent to rerun the consistency checks using a single comparator.

As an optimisation, the agent may choose to check for inconsistencies using the diff format data directly. This has the advantage of being much faster since it checks only those areas where

the storage element's namespace has changed. However, it would require different comparators for dump format and diff format data. Also, if the file catalogue does not provide diff format data (or equivalent functionality) then there is a risk that inconsistencies generated within the file catalogue will go unnoticed.

5. Current status

Deployment of SynCat is in its infancy. Perhaps the best support is for dCache[9]: the pnfsDump utility provides the SynCat dump format data for PNFS[10] and there is work underway that has provided support for SynCat format for the Chimera namespace.

The DPM[11], LFC[2], Castor[12] and StoRM[13] projects have agreed to implement SynCat subject to demand from end-users.

6. Alternative approaches

In a sense, the SynCat project aims to ensure a contract between the file catalogue and the storage element: replicas registered in a file catalogue exist and files stored in a storage element exist in a replica catalogue. This can be viewed as introducing a coupling between the two file-aware components, albeit one enacted by a third-party: the synchronisation agent.

There are alternative approaches to achieving this coupling. The storage element could contain explicit knowledge of which file catalogues store metadata against its files and update the catalogue directly (e.g., Smart Storage Element[14]). Namespace synchronisation could be achieved by exporting a Bloom filter, a terse summary of all entries within the storage element, so allowing the discovery of replica locations by a central component redirecting clients to likely SEs (e.g., Giggle[15]/RLS[16]). A more radical approach is to introduce a distributed global namespace (using a technology like Distributed Hash Tables) with storage elements acting as object stores (e.g., Chelonia: self-healing storage element[17]).

Implementing any of these approaches would likely require a substantial investment of effort. Given time constraints imposed by WLCG (ultimately due to the LHC facility) implementing such architectural changes is not feasable. However, the interested reader is directed to these other solutions to the catalogue inconsistency problem as a source of further inspiration.

7. Conclusions

In this paper we describe different types of inconsistencies that can occur between the storage elements and file catalogues within a loosely coupled Grid, such as WLCG. The mechanisms currently available to detect possible inconsistencies are examined and three problems common to them are detailed.

The paper describes the SynCat consistency checking model, detailing the two modes of operation: full and incremental. The data formats for two output modes are explained and the two modes are cast in terms of a synchronisation agent receiving data in these two formats.

Finally we briefly mention alternative approaches that would require architectural changes to the model that, due to time constraints, cannot be explored within the SynCat project.

Acknowledgments

The authors wish to thank Gerd Behrmann for his insightful comments when discussing this paper.

References

[1] Worldwide LHC Computating Grid *http://www.cern.ch/lcg*
[2] LCG File Catalog *https://twiki.cern.ch/twiki/bin/view/LCG/LfcGeneralDescription*
[3] ATLAS Distributed Data Management
https://twiki.cern.ch/twiki/bin/view/Atlas/DistributedDataManagement

[4] Grid File Transfer Service *https://twiki.cern.ch/twiki/bin/view/EGEE/FTS*
[5] RFC 959 - File Transfer Protocol *http://www.ietf.org/rfc/rfc959.txt*
[6] The Storage Resource Manager Interface Specification v2.2 *http://sdm.lbl.gov/srm-wg/doc/SRM.v2.2.html*
[7] RFC 3659 - Extensions to FTP *http://www.ietf.org/rfc/rfc3659.txt*
[8] The SynCat project *http://www.desy.de/syncat*
[9] dCache *http://www.dcache.org/*
[10] PNFS: The Perfectly Normal File System *http://www-pnfs.desy.de/*
[11] Disk Pool Manager *https://twiki.cern.ch/twiki/bin/view/LCG/DpmGeneralDescription*
[12] Castor: CERN Advanced STORage manager *http://castor.web.cern.ch/castor/*
[13] StoRM: Storage Resource Manager *http://storm.forge.cnaf.infn.it/*
[14] Data management services of NorduGrid *http://www.nordugrid.org/documents/chep04-294.pdf*
[15] Chervenak A *et al* 2002 *Proc. Conf. High Performance Networking and Computing (Baltimore)* p 1–17 (IEEE Computer Society Press)
[16] Chervenak A *et al* 2004 *Proc. 13th IEEE International Symposium on High performance Distributed Computing, 2004* p 182–191
[17] Self-Healing Flexible Grid Storage For ARC *http://www.knowarc.eu/documents/Knowarc_D2.1-1_09.pdf*