Proceedings of the Third Conference of the EELA Project
R. Gavela, B. Marechal, R. Barbera et al. (Eds.)
CIEMAT 2007
© 2007 The authors. All rights reserved

Robust and Resilient Services – How to design, build and operate them

Jamie Shiers[1], Gavin McCance[1], Patricia Mendez Lorenzo[1]

[1]CERN, Geneva, Switzerland
{Jamie.Shiers,Gavin.McCance,Patricia.Mendez.Lorenzo}@cern.ch

Abstract

Grid infrastructures require a high degree of fault tolerance and reliability. This can only be achieved by careful planning and detailed implementation. We describe on-going work within the WLCG project to build and run highly reliable services. Following the "a priori" analysis based on the services and service levels listed in the Memorandum of Understanding that sites participating in WLCG have signed[1], this paper provides an "a posteriori" analysis following over 2 years of production service. This work covers not only the services deployed at the Tier0 centre at CERN - which has the most stringent service requirements related to the acquisition of the raw data, the initial processing phase and the distribution of raw and processed data to Tier1 sites, but also a similar analysis for Tier1 and major Tier2 sites. The latter will be covered at a workshop that will take place shortly before the EELA conference and so will be very up-to-date.

1. Introduction

The target service levels that must be reached by sites that are members of the Worldwide LHC Computing Grid (WLCG) [2] – and hence signatories of the WLCG Memorandum of Understanding (MoU) [3] – range from 99% for key Tier0 services to 95% for some services at Tier2s. However, these targets are not for individual services, but for higher level functionality, such as “acceptance of raw data from the Tier0 centre during accelerator operation”. Furthermore, the experiments – in particular CMS – have established lists of so-called “critical services”, the consequences to the experiment in case of downtime or service degradation, and the maximum acceptable delay for problem resolution. Based on these requirements, as well as several years’ experience of running production Grid services, a simple “tool-kit” for deploying highly available services has been established. This is coupled to a “checklist” for new services, as well as recommendations for middleware and database developers, together with operational techniques and procedures. All of these issues will be discussed during a WLCG Service Reliability workshop (to be) held at CERN during the week of November 26 – 30 2007. Whilst up-to-date information will be presented during the EELA 3 conference, the deadlines for paper submission are such that the information given below is based on the preparatory work for the workshop and not the final outcome. However, no significant changes in terms of the basic recommendations are expected – the main deliverable is to develop a concrete timeline for deploying these techniques across the key sites (as dictated by the experiments' requirements) and to understand how the delivered service level can be effectively monitored.

2. Service Intervention Analysis

An analysis of the service interventions that have occurred over the last two years indicates that the dominant interventions are still *unscheduled*. In particular, a significant number of service interruptions are due to power and cooling problems and network interruptions. These give a "background" against which other types of interruption or degradation need to be measured – there is no point in investing heavily to protect against rare cases when a major cause of downtime remains unresolved. Examples of service problems and the frequency that may be expected are given below:

- Network cut between pit & B513: based on experience, ~1 per decade, fixed in ~4 hours (the network cable is largely redundant)
- Oracle cluster-ware "crash": ~1 per year (per RAC?) – recovery in < 1 hour
- Logical data corruption – database level: ~1 per decade, painful recovery (consistency checks can help here, but scripts run directly against the DB have been shown to cause much higher levels of corruption
- Data corruption – file level: being addressed – otherwise a certainty!
- Power & cooling: will we get to (<) ~1 per site per year? Soon?
- Critical service interruption: 1 per year per VO? Most likely higher in 2008

3. The whole is greater than the sum of the parts

The (W)LCG Technical Design Report (TDR) [4] lists two motivations for adopting a Grid solution. These are as follows:

1. *Significant costs of [providing] maintaining and upgrading the necessary resources … more easily handled in a distributed environment, where individual institutes and … organisations can fund local resources … whilst contributing to the global goal*
2. *… no single points of failure. Multiple copies of the data, automatic reassigning of tasks to resources… facilitates access to data for all scientists independent of location. … round the clock monitoring and support.*

For funding reasons, the first argument is clearly extremely important – for the reason stated in addition to the fact that many of the institutes involved are multi-disciplinary. Thus, not only for resource sharing within a site but also to bolster the scientific and intellectual environment in the collaborating countries, such a scenario is much healthier than one where all resources are concentrated at the host laboratory (and acquired locally).

The second argument needs further analysis and is indeed similar to the 3rd criterion in Ian Foster's Grid computing checklist [5]:

"… to deliver nontrivial qualities of service. (A Grid allows its constituent resources to be used in a coordinated fashion to deliver various qualities of service, relating for example to response time, throughput, availability, and security, and/or co-allocation of multiple resource

types to meet complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.)"

With the exception of services and processing that is performed at the Tier0 site, the fact that much of the data is replicated at several or many sites, the partial or even total failure of a site should not stop the associated production or analysis. Similarly, some of the services – such as the reliable File Transfer Service (FTS) – are already designed to cater for service interruptions at source and/or sink site: if the storage element (SE) at a given site is about to enter scheduled maintenance, the corresponding FTS *channels* that source or sink data in that SE can be paused. This still allows new transfer requests to be queued, but they will not be attempted until the channel is re-opened, avoiding wasting bandwidth on transfers that are bound to fail and potentially reducing the background load on support staff (analysing "fake" failures.)

4. Building Robust Services

Robust services can only be delivered through careful planning complemented by a combination of techniques, including the appropriate steps at application design and implementation level, as well as at the deployment and operational stage. We describe below very simple techniques that have proven extremely effective and widely applicable in designing and delivering reliable services with a reasonable level of effort and – importantly – largely avoiding fire-fighting and panic.

Two mindsets that are particularly important in this respect are:

- *Think service* – a service is far more than a middleware release in a 'production' repository;
- *Think Grid* – a Grid is the ultimate distributed computing system (so far). A change to a service deployed at a given site or site(s) may well have an impact far wider than the local community and must be planned and announced accordingly.

Before we list the techniques that are in daily use for deploying and operating the WLCG service, we consider some of the issues related to failures and support calls, together with their associated costs.

Consider, for example, the reliable file transfer service. Given the expected data volumes and rates, a typical LHC experiment will transfer globally of the order of 10^5 1GB files per day – many more if analysis data and calibration datasets are also included. The percentage of such transfers that fail in such a way that human intervention is required must be extremely low, particularly as the problems seen after automatic retries are often complex and time consuming to resolve. Other examples come from user support costs. A ticket that a ticket processing manager spends 1 hour on (and may take much more to solve) has a real and non-negligible cost associated with it. Not all such problems can be avoided purely through good documentation and robust services, but there is clearly very strong motivation to do so. Finally, any operational issues that require human follow up must be reduced to the absolute minimum – anything that can be documented in English (or indeed any other language) can also be programmed as a script or in a higher level language – computers are simply much better and cheaper at doing repetitive tasks rapidly than humans, whose particular analytical skills are best used elsewhere.

5. Check-list for New Services

Before a new service is deployed – be it in a Grid or non-Grid environment – a straightforward checklist has been established that has proven invaluable in ensuring that the resultant services are of the required quality. Ideally, this work starts well prior to deployment – the middleware must be designed and written with reliability in mind. This includes details such as error messages and logging – this must be consistent and in an agreed place to which the necessary support teams have access if required (the latter is non-trivial in the case of cross-site services). The application must be designed to cope with “glitches” – e.g. short-lived problems with services on which they depend and which are simply unavoidable in a distributed environment. Where possible, the ability to share the load across multiple load balanced servers offers numerous advantages, including transparency to many common service interventions and even middleware upgrades. In the case of a database backend, the ability to re-establish a connection and – assuming a database cluster – failover transparently from one node to another are mandatory features. The appropriate hardware must obviously be allocated – avoiding (except in cases such as batch worker nodes) single points of failure through power supplies or feeds, network connections and so forth. Finally, a minimum set of operational procedures – including contact names and addresses – together with a basic set of tests (no contact, high load etc.) is needed. The necessary workflows also needed to be established in the support lines, together with diagnostic tests and procedures for the various levels of support / operations teams. Starting with these essentials, the service manager can readily add more tests and procedures as experience shows are required.

6. Daily and Weekly Operations Meetings

One of the key secrets to running smooth services is a regular operations meeting. This has been in place at CERN since decades before the Grid and used to be performed by vendors (CERN having a number of large mainframes / clusters at that time). In recent years, these meetings have been extended to cover the Grid world, with a clear impact on the state of the Grid services. On occasion, people have expressed ‘disappointment’ that there is not an atmosphere of mad panic / firefighting at these meetings – but this is precisely the point – these meetings are to ensure a smooth service, exactly the opposite of firefighting. Instead of being an overhead, these meetings act as an excellent point of information exchange, and in fact significantly reduce the amount of time spent on identifying and debugging problems. The meetings typically take around 10 minutes – slightly longer on Mondays – and quickly run through alarms and problems seen since the last meeting. If not immediately solved, the problems are assigned to a system administrator or technical expert as appropriate. More importantly, they allow weaknesses in the services – such as lack of adequate monitoring or alarms – to be exposed and peer pressure proves a very effective mechanism for ensuring that these holes are rapidly plugged. Once a week, any outstanding tickets against the CERN Regional Operations Centre are reviewed, again ensuring that problems are not left unaddressed for prolonged periods. Another important topic that is reviewed daily is any interventions scheduled for that day, or any foreseen in the coming days. It cannot be stressed too highly how important adequate preparation for interventions has repeatedly been proved to be – it is not just a question of informing fellow service providers and users, but also ensuring that the intervention proceeds smoothly. All too often, a well debugged procedure runs into problems (often because it is not strictly followed, or as the availability of needed colleagues for a given step has not been checked), turning a smooth or even “transparent” intervention into a prolonged downtime that may even need a further intervention to adequately complete. In the worst cases, unannounced “transparent” interventions have resulted in severe service degradation that have led to extreme user dissatisfaction and have been extremely costly in terms of manpower to resolve. We have therefore agreed simple procedures for announcing scheduled interventions of various lengths, as well as unscheduled interventions. Equally importantly, an announcement through the agreed channels is required when the service is fully restored (or periodic announcements in case of prolonged problems), as well as an open post-mortem, recording any unforeseen problems, their resolutions and lessons for the future. These daily – primarily site-oriented (see caveat above) meetings are complemented by weekly joint operations meetings with all the main sites that have a similar agenda but also include VO-specific issues. Finally, less frequent meetings are held to ensure that the operations tools adequately address the needs of the community. These meetings are typically held bi-annually.

7. Key Techniques

The main techniques that are used in conjunction with the standard operations procedures are both simple and well-understood:

- Understanding the impact of downtime or degradation to service. In some cases, it may even be acceptable for a problem only to be resolved the next working day whereas in others this would clearly be unacceptable: resources being limited, the effort (and money) needs to be focused in the right places;
- The use of database clusters for middleware components that have persistent state (together with the appropriate deployment and application development strategies);
- Load-balanced servers for the middle tier.

These techniques not only allow services to be resilient to single (or even multiple) component failure, but permit many of the common interventions to be performed with zero user-visible downtime. These include operating system, database or middleware upgrade or security patches as well as the addition of new hardware / replacement of old or failure nodes. In the case of the best behaving applications, these techniques have been fully supported for a number of years. Further work is required to make all of the main WLCG services sufficiently resilient – this is currently underway, being driven by the priorities of the experiments.

8. Middleware and Database Development Techniques

CVS: ‘ça va saigner’
Subversion: Destroying someone's (or some group's) honesty or loyalty; undermining moral integrity

The key point about designing middleware for robustness and resilience is to incorporate these aspects into the initial design. This is because many of the deployment and operational features already discussed have an impact on the basic architecture and design of the software; it is typically much more expensive to retrofit high-availability features onto a software product after the design and implementation (although it is possible). In general, the service scaling and high-availability needs typically mandate a more decoupled architecture.

Decoupling the different sub-components of a single service from each other is extremely desirable – quite apart from the long-term maintainability of the code, debugging the running service in production is much easier if the architecture is cleanly defined and the responsibilities of the components are clear. It is also advantageous from the point of view of basic operations of the service. For example, the FTS allows the agent daemons (that actually do the real work of processing the file transfers) to be taken down for intervention, moved or redistributed while maintaining the front-end of the service up and running, so that the users can continue to interact with the system. The individual channels in the service may be taken down separately for intervention without affecting the other channels in the service.

The internal architecture and design of a service must assume that some sub-components of the service will be unavailable at some times (either due to scheduled maintenance or unscheduled problems) – individual components should be resilient to failures of the other components, or at least the failures modes should be understood and documented, together with the impact on the overall service. Experience has shown time and again that this is the major cause for ‘mysterious’ service failures and lock-ups - and these sorts of problems can absorb considerable debugging effort. What happens to a service component, for example, when the central logging server all the components rely on goes down? Does it stop? Is it acceptable (from an audit point of view) for the service to continue without the central logger? Another example of over-coupled architectures is the often excessive use of remote procedure call between sub-components. Sometimes the use of RPC desirable, but generally, passing critical information in this way should be avoided – favouring instead the use of a transactional system such as a database.

Limiting the state maintained in the middle-tier of a service is also important – for example, the LFC and FTS services commit frequently to the database and the user operations on the service are designed to be transactional, such that even in the case of immediate power loss, they will come back up without any loss of data that the user has committed into the service (or any corrupted state). Sessions, if required, should be migratable, or at least, individual instances of the load-balanced service should be ‘drainable’. The use of industry-standard components (e.g. Apache) to build up a service helps with this since many of these service-oriented features come ‘for free’.

The basic rule of ‘Think Service’ should be applied to the middleware design – this is often overlooked in the rush to get user-facing features ‘out of the door’. Can a service be cleanly paused without affecting the user’s ability to interact with it? Can a node / channel / part of the service be cleanly ‘drained’ such that its removal does not affect the running service? The robustness and resilience of the overall service are critically dependent on providing to the service manager facilities to allow common scheduled interventions (e.g. node replacement, kernel upgrade) to take place with minimum impact to the service; these facilities can also help ameliorate the impact of unscheduled interventions.

For a distributed Grid where the often the overall service that the user ‘cares about’ depends on multiple services, potentially in multiple administrative domains (e.g. file transfer) the same rule applies – services should be designed assuming that there will be occasional outages of their dependent services, outages of the wide-area network and other such glitches [Rules of Distributed Computing]. Ideally, the middleware should seek to ‘add value’ in this regard making the overall service more resilient to these glitches (“The whole is greater than the sum of its parts”). These features should then be fed back into the grid operational procedures (or automated systems) to make sure that they are used where appropriate. The other point is to think how the overall service will be debugged - which leads to requirements such as the adoption of a reasonably uniform logging format (e.g. use UTC in log timestamps, or at worst standard TZ-stamped formats)

Often the most critical part of grid services is the database. Apart from the HA deployment of the database itself, there are a number of simple techniques to improve the reliability of middleware with regard to its interaction with the database:

- Connection retries. The middleware should retry to connect if a database connection becomes unusable. There are a variety of standard connection-pooling implementations available that do this, many coming for ‘free’ if you build the application using an industry-standard tool such as Apache or J2EE containers. For multi-threaded applications, connection pooling is also critical for performance since databases suffer rather badly under constant connection/reconnection loads. Making sure the application and its deployment scripts are written to make use of the DB’s High Availability features also help considerably (e.g. Oracle’s Transparent Application Failover);
- Using the database to enforce all known integrity constraints is good design and helps considerably for the robustness of a service. It helps catch application logic errors which can otherwise be very hard to debug. In an environment (such as HEP) where the application requirements are evolving, ad-hoc ‘scripts’ tend to appear, either bug workarounds or one-off tools providing functionality not in the software (for example, “please add this extra ACL to all 30 million files in this file catalog with no online performance impact”). Having the database enforce constraints is advisable since not all of these ‘scripts’ are of production quality and experience has shown that the cost of logical schema corruption on a production system is extremely high;
- Testing of the application with new versions of the (database) software prior to its deployment is also critical to smooth service operations – it is not unknown for high performance grid applications to expose bugs and glitches in new versions of the database software;
- Testing the application at an appropriate scale on a reasonably sized validation cluster is also important, since many issues only appear at close to production scale;
- Maintain a good relationship with your database administrator (DBA), and don’t treat the database as a ‘black box’. There are a number of simple techniques your DBA can advise you on – the use of bind variables (your DBA can show you where you forgot to use them), appropriate schema design, appropriate use of indices (your DBA can show you where you need to define a new one) and appropriate use of more advanced DB-specific features such as table partitioning. These features become more and more important as the amount of data stored by your service grows (and, if neglected, will typically begin to bite just as your main production phase begins, when you have the least time to deal with it).

Many more suggestions for good database / software interactions (focussed on Oracle) can be found in [6].

9. Critical services

The services that an LHC experiment relies on to run its production include a number of important VO-specific components over and above the standard Grid middleware-based ones. If one of these services is down or impaired, the experiment is impacted at least as severely as if one of the key Grid services was down. It is therefore essential to address the reliability of these components. A proposal that has yet to be put in place is to treat these services in the same manner as the standard Grid services, including techniques for writing robust services as well as their deployment and operation. Particularly in the early months and years of data taking, it is likely that there will still be some residual instability in some of these services and it is proposed that in the key areas of storage, Grid data management and databases for physics that an on-call service is established. This would allow technical specialists to be contacted 24x7 in case of problems that cannot be resolved using the standard documented procedures. It will clearly provide significant motivation to further improve the robustness of the services and it is foreseen that the need for such on-call be reviewed annually. For other important services, for which a permanent on-call rota is not justified, named contacts within each of the experiments will be able to call out an expert by passing through the console operators. Such interventions need to be relatively rare and will also be regularly monitored to understand both the need and sustainability of the system.

10. Monitoring, Logging and Reporting

These related but distinct aspects of running a service are often confused. Furthermore, the specific information that should be presented depends very much on the target audience. A common trap is to try and build everything into a single tool, which continues to grow until it is un-maintainable and even unusable. For example, a funding agency may be concerned with how well the resources provided being used. A VO manager may wish to see how well their production is proceeding. A site administrator on the other hand may simply want to see if his or her services up and running and meeting the agreed MoU targets. The on-duty operations team will typically want to know if there any outstanding alarms. Finally, an LHCC referee may want to see how the overall preparation progressing with any areas of concern highlighted. Nevertheless, much of the information that would need to be collected is common and so it is important to separate the collection from presentation (views…), as well as the discussion on metrics. It is precisely this approach that is being adopted – with some success – by the LCG monitoring working groups that were created one year ago. In addition, the issue of improved and consistent logging is being actively pursued by the middleware developers – the status of both of these issues being presented at the EGEE'07 conference in Budapest.

11. Mind The Gap

During the above conference, a number of service problems came up that highlighted the need for well documented – and followed – procedures, as well as excellent communication. To be explicit, three significant service problems came up in a single week – all of which were easily avoidable. These were as follows:

- A bug in Oracle client libraries – both documented and already fixed in production releases – caused a number of daemons to go into an infinite loop (after 248 days of uptime – the maximum number of clock-ticks (1/10s) that can be represented in a 32bit integer). An analysis of the problem revealed that not only are there numerous methods for deploying Oracle client releases but also there was no consistent agreement for which of these to use, nor for moving to new versions;

- A change to the service availability algorithm – improved in principle at the various management boards – was released in production without being scheduled or even announced via the regular operations meetings. This caused significant knock-on effects in other service monitoring tools;
- A database house-keeping exercise resulted in an index being de-selected, with following service overload and meltdown.

Whilst it is unlikely that all such problems can be avoided in the future, we cannot afford to tolerate such a high rate of completely avoidable issues. Hopefully, the experience from these events will reinforce the widespread adoption of the simple and lightweight procedures that have been shown to work in exactly these situations.

12. Caveat Emptor

A final piece of cautionary advice concerns coupling between services and the sometimes unexpected consequences. Two concrete examples in this area relate to the choice of database synchronization technology that we have deployed. This is used for the file catalog middleware component and for detector and calibration alignment information – in both cases the corresponding information is kept in sync between the main sites with minimal delays. However, this has meant on one occasion that a key database feature had to be disabled – with significant consequences on the ability to recover the service in case of accidental loss of data – and on another led to silent data corruption. On balance, the benefits certainly outweigh the drawbacks, but underline the need for openness and transparency – the reasons for choosing a specific release and the consequences must be clear to all, particularly in the case of complex, layered services.

13. Conclusions on Robust Services

Taken together, these techniques and procedures have been demonstrated to be sufficient to offer robust and resilient services, but are unfortunately often overlooked. We know how to run reliable services – this is not to say that no user support issues remain! The issue of support for large and diverse user communities of a system with the complexity of the Grid is certainly one of the challenges that will need to be addressed by future e-infrastructures. In particular, it is essential that we neither design nor use Grids in such a way that the unavailability of a single service renders a site – or worse the entire Grid – down. Such problems should, in the worst case, result in a small inefficiency of the overall Grid resources, rather than a downtime.

14. Summary and conclusions

We have described a set of basic techniques for the design, implementation, deployment and operation of robust and resilient Grid services. These techniques are now being extended beyond the basic set of WLCG services to cover also experiment-specific services that are critical to their production activities. These techniques are sufficiently general as to be applicable to many other application domains and indeed other Grid projects. May the force be with you.

References

[1] A Worldwide Production Grid Service Built on EGEE and OSG Infrastructures – Lessons Learnt and Long-term Requirements: presented at this conference.
[2] The Worldwide LHC Computing Grid (WLCG), http://lcg.web.cern.ch/LCG/.
[3] Memorandum of Understanding for Collaboration in the Deployment and Exploitation of the Worldwide LHC Computing Grid, available at http://lcg.web.cern.ch/LCG/C-RRB/MoU/WLCGMoU.pdf.
[4] LCG Technical Design Report, CERN-LHCC-2005-024, available at http://lcg.web.cern.ch/LCG/tdr/.
[5] I. Foster, Argonne National Laboratory and University of Chicago, What is the Grid? A Three Point Checklist, 2002.
[6] Effective Oracle by Design (Osborne ORACLE Press Series), Thomas Kyte.