The next generation of distributed systems will be marked by numerous networked units linked through various
technologies such as wired, wireless, and optical. These units will be separated by a range of distances (local
area, wide area, or personal area) and composed of devices from sensors to mobile units to high-end machines.

A variety of applications will run on such networks and will frequently be composed of many different
software services with dynamic execution conditions and resource demands—often across many different
administrative domains. Because these applications can be highly complex, their development and deployment
requires appropriate middleware technology.

In recent years, one of the most popular approaches to dealing with this problem (as well as one of the biggest
buzzwords in distributed systems) is Grid computing, popularized by the work of Ian Foster at Argonne
National Laboratory. Foster characterizes a Grid by three features:

- It coordinates resources that aren't subject to centralized control.
- It uses standard, open, general-purpose protocols and interfaces.
- It delivers nontrivial qualities of service.

This description seems like it could apply to any traditional distributed system. However, a key difference is
that many Grids will provide services that are orchestrations of multiple services, hosted on different sites and
across organizational and administrative boundaries, thus raising a lot of new issues. The Globus Toolkit
—currently at version 4—is the accepted reference implementation for Grid-computing middleware, while
Global Grid Forum members work on Grid-computing standards. But, our concern in this article is the last
point in Foster's checklist. For all the work being done to provide usable Grid middleware, the subject of
dependability has been strangely neglected. The community hasn't launched a large-scale effort to provide tools
to help develop dependable Grid applications.

We'll therefore talk about why dependability is so important in Grid applications and demonstrate some of the
work our team, the Distributed Systems and Services Group at the University of Leeds, is performing.

**Why is dependability so important for Grid applications?**

A key consideration in a Grid-centric, service-based world is that applications can be composed of multiple
autonomous services, each of which might be running on different machines, under the control of different
organizations. We might not be able to make guarantees about a resource's trustworthiness (it might
maliciously alter the returned results, for example) or a service's reliability or performance. Depending on the
domain, this lack of trust makes even obvious dependability mechanisms and measures, such as checkpointing,
potentially ineffective.
We're investigating *multi-version design* (MVD), a classic fault-tolerance technique whereby a system invokes in parallel multiple independent but functionally equivalent services and votes on their results (see figure 1). The voting system can then perform various tasks, such as pass on the consensus result or an averaged result.

![Figure 1. A simple multi-version design system.](image)

**Why multi-version fault tolerance?**

Traditionally, MVD has been quite controversial due to the likelihood of *common-mode failures* (when multiple channels fail in the same way, causing the voter to unknowingly pass on an incorrect result). These failures occur far more frequently than you might expect, simply because errors across channels occur less independently than you might think—people tend to make the same mistakes in the same places. So, the overall dependability gain of the MVD approach might be less than the theoretical gain (especially when assuming errors' independence). Although there are many successful examples of MVD systems in realistic industrial applications, the ultrahigh cost of developing (and maintaining) an MVD system often makes the cost/benefit ratio off-putting.

In the service-oriented Grid world, however, this might not be the case. If we assume that multiple functionally equivalent services exist, developed and maintained by different organizations and sitting on different hardware, then surely we can use these services to create dynamic multi-version systems at runtime? If so, these systems will cost much less than traditional MVD systems, while retaining the same dependability gains. Indeed, we could potentially have even more redundancy—perhaps by using five or seven channels. If this is the case, then we could still only wait for the first three or five results to be returned, thus even masking some late timing faults.
However, the sting in the tail comes in how little we know about a given channel's workflow. In the service-oriented world, we frequently know only little about a given service, such as who owns it, where it's located, and what its interface is. Behind the scenes, the service might itself be composed of numerous other services, all chained together in a workflow. Imagine several of our seemingly independent multi-version channels each having a common service buried somewhere in their workflow. If this were so, then faults in this common service might ripple back and have a large influence on a common-mode failure's probability. (See figure 2 for an example of common services.)

![Figure 2. An example of common services between workflows. In this case, service 1 and service 2 both invoke two common services as part of their workflows.](image)

A way around this scenario is to provide our fault-tolerance mechanism with topological information. We can do this in many ways, but one method that particularly springs to mind is to leverage provenance—the documentation of a process that leads to a result. Several provenance systems for service-oriented architectures are available. By analyzing the provenance data a service generates, we can extract its topology, or workflow, and use that data to either discard or alter the weightings of results going into our voting process.

**The FT-Grid system**

With these ideas, we created our multi-version fault-tolerance tool for service-oriented architectures: FT-Grid (see figure 3). FT-Grid lets developers quickly and easily create dynamic, service-based, multi-version systems by querying UDDI repositories (think of them as search engines for services) and invoking functionally equivalent services in parallel. It has three guises: a GUI client, a Java API, and a service interface. Integrated with the PreServ provenance system developed at the University of Southampton FT-Grid can quickly and easily extract the topology of any provenance-enabled service. FT-Grid can then feed the topology back to the developers, who can use it in whatever voting mechanism they require. We've tested FT-Grid against many experimental systems and gotten some highly encouraging results.
How do you know if a service is dependable?

We've discussed one way to create more dependable Grid and Web services, but what about evaluating how dependable such a system is in the first place? Again, not much work has been done in the Grid and Web service communities regarding tools for evaluating dependability in services. To address the question, our group has created a fault-injection-based tool for Grid and Web services called Grid-FIT (see figure 4).
Figure 4. The Grid-FIT GUI (click on image for larger view).

Grid-FIT is a network-level fault injector capable—via a patched Apache Axis handler—of intercepting SOAP messages sent in and out of a hosting environment before the security layer is applied. This means that we can perturb or modify information such as parameter values and results before the hosting environment signs the SOAP message. In addition, we can simulate late timing faults by intercepting a SOAP message and waiting for a fixed period before releasing the message to be passed on its merry way.

Fault injection is a great technique, because it lets you stress-test services with inputs and conditions that they might not be exposed to during regular testing. In other words, fault injection isn't an alternative to traditional testing methods; rather, it's a complement thereof. By systematically injecting faults and monitoring a service's behavior, Grid-FIT can quickly build up large amounts of metrics about how a service will behave under certain conditions—be it by sending a polite message that the incoming parameters are invalid or by crashing its hosting environment (see figure 5).
As you can see by this last, possibly far-fetched example, Grid-FIT's benefits aren't necessarily just at service level; you can even stress-test the hosting environment under all manner of unusual conditions.

**Conclusion**

The dependability community has an enormous amount work to perform in the Grid and Web services areas. Indeed, some of these issues were discussed at a recent IFIP 10.4 Working Group meeting.

For example, management mechanisms for fault tolerance are a challenge in Grid computing because of the potential for systems to cross organizational boundaries. The community must investigate not only those mechanisms but also the methods of providing fault tolerance to the mechanisms themselves! An almost endless list of issues and topics require investigation and consideration, including fault detection, collaboration, trust, security, licensing, charging models, and so on. As Grids become increasingly mainstream and prevalent, it's vital that the dependability community make a more concerted and coordinated effort to contribute research that will enable the vision of powerful, dependable, and secure Grids to become a reality.

**Figure 5. Metrics collected using Grid-FIT (click on image for larger view).**
Paul Townend is a research fellow in the School of Computing at the University of Leeds. Contact him at pt@comp.leeds.ac.uk.

Jie Xu is a professor in the Department of Computer Science at the University of Leeds. Contact him at jxu@comp.leeds.ac.uk.

Related Links

- DS Online's Dependable Systems Community (http://dsonline.org/dependable)
- DS Online's Grid Computing Community (http://dsonline.computer.org/gc)
- "Emerging Grid Standards" (cms:/dsonline/0507/c4bak.xml)
- "A New Programming Model for Dependable Adaptive Real-Time Applications" (cms:/dsonline/0505/o5001.xml)

Cite this article: Paul Townend and Jie Xu, "Dependability in Grids," IEEE Distributed Systems Online, vol. 6, no. 12, 2005.