Short paper: Materializing Highly Available Grids

Mark Silberstein, Gabriel Kliot,
Artyom Sharov and Assaf Schuster
Technion, Haifa, Israel
{marks, gabik, sharov, assaf}@cs.technion.ac.il

Miron Livny
University of Wisconsin, Madison, USA
miron@cs.wisc.edu

Abstract

Grids are becoming a mission-critical component in research and industry. The services they provide are thus required to be highly available, contributing to the vision of the Grid as a dependable virtual computer of infinite power. However, building highly available services in Grid is particularly difficult due to the unique characteristics of the Grid environment. We believe that high availability functionality should itself be provided as a service, which can be used by transparently decorating*, but not changing, the original services, thus making them highly available. In this work we highlight the major challenges and describe our initial experience in building such a generic high availability service in the context of the Condor system.*

1 The Problem of High Availability in Grids

The ever-growing computational and storage needs of contemporary science and industry led to broad adoption of computational Grids, making them an essential tool in everyday practice. Naturally, their popularity resulted in a growing dependence upon the services which they provide, and they often become a critical link in production and research chains. However, numerous factors contribute to a decreased availability of Grid-supplied services. Such factors include frequent network interruptions common in large scale deployments over WANs, uncoordinated maintenance tasks due to multiple administrative domains, and commodity hardware and software components, susceptible to frequent failures.

It could be expected that the problem of building highly available (HA) services, which has been studied extensively over the last 30 years, and successfully solved in small-scale cluster environments, should not require any additional effort in the context of Grid.

However, the characteristics of large-scale Grid environments differ significantly from those of tightly-coupled clusters, making the previous solutions inadequate in a new setup. We identify the following major challenges:

Service replication over WAN Imminent network failures, common in large scale deployments over WAN, make it desirable to set backup replicas of a service at multiple sites, on different subnets. Thus, IP fail-over techniques, such as [4, 1] are inapplicable in these cases.

Lightweight protocols Large scale deployment of HA services makes it impossible to use tightly coupled, heavily synchronized protocols (such as traditional Group Communication systems [2]). The inherent dynamicity of the Grid environment may result in frequent state changes, making them a common scenario and requiring optimized protocols.

Autonomous partitions Multiple service replicas in large scale deployment can be partitioned in the case of network failure. It is often desirable that a service be provided in each partition independently, falling back to a single replica when the network is reconnected.

Network anomalies Trivial assumptions of symmetry, transitivity and bounded network delays, which significantly simplify implementation of HA in LANs, are no longer valid for WANs.

Handling transient failures Transient short network outages may be quite frequent in WANs, challenging the robustness of failure detection protocols. For example, in the context of primary/backup scenario, improper handling of such failures may result in undesirable "fibrillation" of the primary server between backup instances, only decreasing the quality of service.

We propose to encapsulate the handling of these and other issues related to HA in Grids in a single service, *HADecorator*, which can be used to augment any existing service with HA functionality. There are two main requirements for designing the *HADecorator*. First, it

should allow transparent addition of HA via decoration of existing and already deployed services, requiring no or minimal change, and freeing the service developer from the need to deal with HA issues. Second, it should be capable of providing different HA semantics as required by various services, such as single/multiple active replicas, state replication consistency schemes, and others. Ultimately, our goal is to turn HA into an easy-to-use commodity, which can be adjusted to the specific requirements of different services.

Our design separates *HADecorator* into two loosely coupled building blocks: *HAInvocator*, which maintains a set of active and backup replicas of a given service according to the required semantics, and *HAReplicator*, which ensures a consistent service state in all replicas, in accordance with the required consistency scheme. This modular design, which decouples the state replication from service availability, achieves the degree of flexibility required to accommodate a wide variety of HA semantics via combination of different implementations of both components.

In the next section we describe our experience in building *HADecorator* and applying it in a production system.

2 Work in Progress: Adding High Availability to the Condor Central Manager

Condor [5] is a high throughput batch system, widely used in many production Grid deployments. Condor's core capability to find and allocate computational resources that match pending execution requests is encapsulated in a single node, called the Central Manager (CM). The CM is comprised of two components: *Collector* and *Negotiator*. *Collector* serves as a global pool metadata repository, that stores the state of the pool, namely, the information about all resources and execution requests, sent periodically by all Condor components. *Negotiator* uses the data accumulated in *Collector* for the actual resource allocation.

CM failure paralyzes Condor's ability to match newly submitted jobs, thus requiring this service to be highly available. While CM was not originally designed for HA, adding this functionality by changing the service's internals is not a viable option, a decoration approach is thus required.

We separate the issue of providing HA to *Collector* from that of providing it to *Negotiator*. *Collector*'s HA is accomplished by redundancy, as *Collector* is essentially a passive data repository and permits multiple active instances to coexist in a pool. Periodic state updates from all Condor components are sent to multiple active *Collector* instances, replicating the pool state in all of them.

Adding HA to *Negotiator* is much more challenging. First, only a single active instance of Negotiator should be available in a connected network partition. Second, *Negotiator* maintains its own state, which comprises pool accounting and user priority information, and should be made available in all replicas to preserve the correctness of resource allocation protocol in case of failover.

These requirements are satisfied as follows. The *HAInvocator* module implements primary/backup semantics, making only one active instance of the service available at any moment. Multiple *HAInvocator* daemons run on different machines in the pool, each one controlling a single *Negotiator*. Only one *HAInvocator* instance acts as an active leader at any given moment, maintaining an active *Negotiator*. When failure is detected, backup *HAInvocator*s trigger a leader election protocol, eventually selecting a new leader, which starts the *Negotiator* instance it controls. In the case of network failure, a leader is elected in each connected partition, acting autonomously until network connectivity is restored. Merging of partitions may result in multiple active leaders. When this is detected by the algorithm, the process of conflict resolution is triggered, again yielding only a single active instance of *Negotiator*. Our leader election algorithm is a modified version of the Bully algorithm [3].

Negotiator's state is encapsulated in a set of files, which are reliably and periodically replicated by *HAReplicator*, collocated with every replica. *HAReplicator* maintains a consistent state among connected replicas, accommodating for conflicts that arise when multiple instances running autonomously in different network partitions are reconnected.

Figure 1 depicts the main components of our system. Every Central Manager machine runs an instance of *Collector* daemon and an instance of *HADecorator*, comprising of *HAInvocator* and *HAReplicator* daemons. Notice that only a single instance of *Negotiator* exists in the pool in any instant. One *HAInvocator* acts as a leader and maintains a running instance of *Negotiator* on its machine, while backup *HAInvocator*s monitor leader's activity and are ready to trigger a leader election process once leader's failure is recognized. *Negotiator*'s state is being periodically replicated by a leader *HAReplicator* to all backup *HAReplicator*s.

We emphasize that neither *Collector* nor *Negotiator* are aware of being made highly available, so that not a
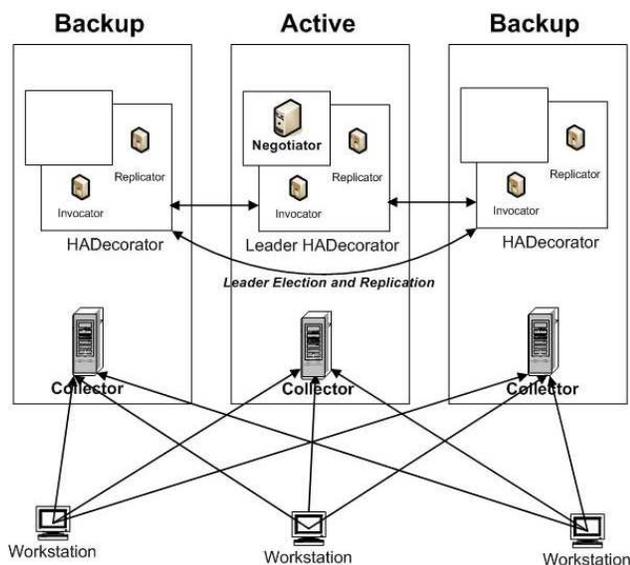
Figure 1. HA Architecture in Condor

single line of code needs to be changed or added. Furthermore, the lightweight asynchronous protocols behave as self stabilizing in the case of transient network failures, which makes them suitable for both LAN and WAN environments. *HADecorator* has been already integrated into the mainstream Condor release and is being adapted for use by additional Condor components.

HADecorator is a critical component in the system, and any potential correctness flaws, inevitable in a development cycle, should be eliminated. To ensure the correctness of the implementation, we have developed a distributed testing system which allows to simulate a steady state, crashes, network disconnections and network partitions. The testing system automatically creates test cases from the set of possible scenarios. Each test case is run distributively over a set of computers, allowing to validate the correctness of the implementation in a real setup, as opposed to single-machine testing systems. The system monitors the activities of all relevant components (*Negotiator*, *Collector*, *HAInvocator* and *HAReplicator*) and verifies in an online manner that a set of distributed invariants defined for a given scenario is not violated. Examples of such invariants include "exactly one *Negotiator* in every network partition" (except for short stabilization periods), consistency of *Negotiator*'s replicated state, consistency between multiple *Collector* instances and others.

3   Conclusions and future work

This paper draws the general design guidelines and emphasizes the main challenges for building a generic service for high availability. Such service hides the complexities of providing HA functionality in the Grid and allows the developer to concentrate on the business logic. We believe that the separation into two loosely coupled building blocks, each of which (and many variants of each) can be built using existing algorithms, is what allows to accomplish the two goals we strive for, namely decorability and flexibility. The Condor CM is our initial attempt to apply *HADecorator* in a real system, which will be extended for a broad set of Grid services.

There are still several challenging issues which are the subject of active research. They include adding support for various consistency schemes and different HA semantics, as well as coping with lack of network transitivity and symmetry in WANs, dynamic addition and removal of replicas, and dynamic registration of additional services that require HA functionality.

References

[1] http://www.linux-ha.org.

[2] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469, 2001.

[3] H. Garcia-Molina. Elections in a Distributed Computing System. *IEEE Trans. on Computers*, C-31(1):48–59, Jan 1982.

[4] Kshitij Limaye, Box Leangsuksun, Venkata K. Munganuru, Zeno Greenwood, Stephen L. Scott, Richard Libby, and Kasidit Chanchio. Grid-Aware HA-OSCAR. In *Proc. of 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS)*, pages 333–339, 2005.

[5] D. Thain and M. Livny. Building reliable clients and servers. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San-Francisco, 2003.