A Rendezvous Framework for the Automatic Deployment of Services in Cluster Computing

Cristina Rodríguez-Quintana¹, Antonio F. Díaz¹, Julio Ortega¹, Raúl H. Palacios¹ and Andrés Ortiz²

¹ Department of Architecture and Computer Technology, University of Granada
² Department of Communications Engineering, University of Málaga

emails: crodriguez@ugr.es, afdiaz@ugr.es, jortega@ugr.es, raulhp@ugr.es, aortiz@ic.uma.es

Abstract

High-performance computing (HPC) systems are increasing their resources allowing the creation of complex services. Dynamic configuration management tools are needed to carry out the deployments of modern distributed services; however, specific solutions are implemented in every case. In this paper, we present a framework for dynamic configuration and management where servers rendezvous with clients and share their information automatically. The internal mechanisms evaluate overall requirements and undertake actions to maintain the services robustly working. Although we have designed it for general use, it is oriented to distributed filesystems, such as AbFS. Through the evaluation, the proposed model takes low overhead and simplify the creation of new dynamic services.

Key words: Scalability, High-Availability, Filesystems, Consul

1 Introduction

An important issue to create robust distributed services is to management dynamic configurations and define mechanisms which maintain the main function of the global system under fault-tolerant conditions. Some automatic processes are needed to perform some tasks in a cluster computing environment with hundreds of nodes where servers or even network can fail.

Although manual or hardcoded configurations are easy to implement, the final system can provoke numerous problems such us: scalability, resilient to failures, reduced visibility
There are several key issues to manage distributed resources and services: dynamic configuration, fault tolerant capabilities, health checking, and leader election. An ideal service should consider some of these elements. Some algorithms, protocols and tools have been developed to solve specific problems.

Paxos [7] is an algorithm that solves distributed consensus while tolerating host failures, network partitions, and message loss. Paxos allows multiple hosts to reach consensus on a single value by relying on majority decisions. Raft [9] is another consensus algorithm for managing a replicated log. It separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherency to reduce the number of states that must be considered. Chubby [1] is a fault-tolerant lock service. It has been designed to run on a small number of hosts, where each node holds a replica of a simple database. It is used in The Google File System [3] to choose primary nodes which ensure data consistency.

Zookeeper [5] is a high-performance coordination service for distributed applications written in Java. It maintains strong consistent, based on Zab protocol (Paxos-like) and quorum. All dataset must fit in memory. It implements a shared hierarchical namespace, with ephemeral node support and access control list to each node. Consul [4] is a tool for discovering and configuring services. It is written in Go, and based on Gossip protocol for all the nodes and consensus protocol (Raft-based) for servers. It also supports access control list, kev/value storage and multi datacenter capabilities.

Fault tolerant and redundant elements are needed to guarantee reliability systems; however, some studies related to proactive fault tolerance [8] can improve overall functioning. Furthermore, it can be applied to storage systems [6]. Even though these algorithms are intended to cover specific needs, specific solutions are implemented in every case. In this work we present a general framework to simplified the most common elements.

2 RendezVous Approach

We describe a model that allows dynamic system configuration and fault-tolerant based on agents that offer different services in a reliable manner. These services are executed transparently by agents which communicate to each other in a safe way.

Furthermore, the proposed model includes distributed virtual objects that simplify the execution of global and local code. These objects do not have to be associated with a particular host and have properties and methods that can run redundantly. Even though a host fails, another one can continue with the execution.

The framework provides the following resources:

- Dynamic configuration: Once a configuration is created, it must allow modify by
Figure 1: Rendezvous framework interconexión diagram.

adding or removing resources. This is also necessary to provide scalability.

- Runtime changes: Non-stop services require to operate without disruption.
- Consistent information: Despite changes and server failures, all nodes must have the same shared information.
- Health check: It is a critical component that prevent using services that are unhealthy.
- Dynamic Global administration: There are some nodes that must monitor the overall operation of the system.
- Leader election: Some resources can be shared to balance workload.
- Fault tolerant operation: If a resource fails the system can continue working.
- Resynchronization: If there is redundancy and a node fails, there are mechanisms to recover from a failure.
- How the software is updated: Software are continuously evolving with the implementation of new functionality and eliminating bugs, so automatic update mechanism can reduce failures during the process.
• How preventively operation is monitored: Preventing failures with the monitoring of several variables (e.g. S.M.A.R.T data in disks) detecting which devices can fail in a short period of time.

Although the framework can be used by any application, it is oriented to distributed filesystem, such as AbFS [2] with fault tolerant and scalability needs.

3 Implementation

The rendezvous framework was implemented in Python using some tool for discovering and configuring services such as Consul. It provides several key features: service discovery, health checking, key/value store and multi datacenter capabilities. The framework is composed of the following components:

• Admin servers: Responsible of the global operation.
• Servers: Offer resources and services to clients and can implement redundant elements.
• Clients: Access to servers.
• Services: Components that are shared.
• Virtual objects: can reside in any node and execute code without interruption.

Figure 1 shows the main components and how different services interact automatically sharing vital information to maintain the effective performance of the system.

Figure 2: Some internal resources from Rendezvous Framework.

As shown in Figure 2.a, the system deploys two agents. A primary agent is the responsible of software updates and execute code remotely. Another global agent is in
charge of the rest of services: health checking, config information, service discovery (with leader election), virtual object environment and the possibility of setting up custom services.

Virtual objects can interact with each other and provide an hierarchy of objects. These objects can send and receive events asynchronously. There is a global event queue: This queue is redundant so when an event is generated it is replicated in the global queue so the event is not lost by an unexpected failure. If the node goes down, another node is responsible for continuing execution so the overall process is always alive. 3 to 5 servers are recommended to avoid failure scenarios. Virtual objects can include a state machine with a default set of states, as shown in Figure 2.b. These states can be extended as required.

4 Results

The framework has been tested in a cluster with 16 server nodes equipped with: 2.27GHz Intel(R) Xeon(R) E5520 CPUs, 16GB RAM memory, 1TB Local Disk and OS CentOS 6.6. Our scenario has 3 servers and 13 clients. Some times have been obtained:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Max</th>
<th>Average</th>
<th>Min</th>
<th>Std.Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server write</td>
<td>0.072</td>
<td>0.042</td>
<td>0.037</td>
<td>0.01</td>
</tr>
<tr>
<td>Client access</td>
<td>0.026</td>
<td>0.005</td>
<td>0.002</td>
<td>0.007</td>
</tr>
<tr>
<td>Event response</td>
<td>0.05</td>
<td>0.047</td>
<td>0.045</td>
<td>0.0015</td>
</tr>
<tr>
<td>Health checking</td>
<td>0.0028</td>
<td>0.0027</td>
<td>0.0026</td>
<td>0.000082</td>
</tr>
<tr>
<td>Leader election</td>
<td>0.026</td>
<td>0.005</td>
<td>0.02</td>
<td>0.007</td>
</tr>
<tr>
<td>Update configuration</td>
<td>0.13</td>
<td>0.09</td>
<td>0.07</td>
<td>0.017</td>
</tr>
<tr>
<td>Virtual object creation</td>
<td>0.07</td>
<td>0.013</td>
<td>0.007</td>
<td>0.019</td>
</tr>
</tbody>
</table>

The system provides optimal response time considering that management processes do not require high speed operations.

5 Summary and future work

We presented a rendezvous framework that performs automatic resource management in HPC systems. It provides fault tolerant based on agents that offer different services in a reliable manner. These services are executed transparently by servers and clients agents to communicate each other in a safe way. The proposed model includes distributed virtual objects that simplify the execution of global and local code. This framework was developed in Python combining some tools for coordinate elements in a robust and dynamic way. It offers fast response time and simplifies dynamic configuration in distributed systems. We want to improve the framework with new features and define a robust test bench.
A RendezVous system ...

Acknowledgements

This work has been partially supported by European Union FEDER and the Spanish Ministry of Economy and Competitiveness TIN2015-67020-P, FPA2015-65150-C3-3-P, and PROMEP/103.5/13/6475 UAEH-146.

References


