Beernet: Building Self-Managing Decentralized Systems with Replicated Transactional Storage

B. Mejías, Université catholique de Louvain, Belgium
P. Van Roy, Université catholique de Louvain, Belgium

ABSTRACT

Distributed systems with a centralized architecture present the well known problems of single point of failure and single point of congestion; therefore, they do not scale. Decentralized systems, especially as peer-to-peer networks, are gaining popularity because they scale well, and do not need a server to work. However, their complexity is higher due to the lack of a single point of control and synchronization, and because consistent decentralized storage is difficult to maintain when data constantly evolves. Self-management is a way of handling this higher complexity. In this paper, the authors present a decentralized system built with a structured overlay network that is self-organized and self-healing, providing a transactional replicated storage for small or large scale systems.

Keywords: Decentralized Systems, Distributed Transactions, Peer-to-Peer, Replicate Storage, Self-Management

INTRODUCTION

There are many technological and social factors that make peer-to-peer systems a popular way of conceiving distributed systems nowadays. From the technological point of view, the increment of network bandwidth and computing power has definitely made an impact on distributed systems which are becoming larger, more complex and therefore, difficult to manage. Although classical client-server architecture provides a simple management scheme with centralized control of the whole system, it does not scale because the server becomes a point of congestion and a single point of failure. If the server fails, the whole system collapses.

The key to deal with the complexity of large-scale distributed systems is to make it decentralized and self-managing. Peer-to-peer networks, and especially in their form of structured overlays, offers a fully decentralized architecture which is self-organizing and self-healing. These properties are very important to build systems that are more complex than file-sharing, which is currently the most common use of peer-to-peer. Despite the nice design of many existing structured overlay networks, many of them present problems when they

DOI: 10.4018/jaras.2010070101
are implemented in real-case scenarios. The problems arise due to basic issues in distributed computing such as partial failure, imperfect failure detection and non-transitive connectivity.

The key issue in distributed programming is partial failure. It is what makes distributed programming different from concurrent programming. This is why we would like to quote Leslie Lamport and his definition of a distributed system:

*A distributed system is one in which the failure of a computer you did not even know it existed can render your own computer unusable*

It does not matter how much transparency can be provided in distributed programming, it will always be broken by partial failure. This is not particularly bad, but it means that we need to take failures very seriously, understanding that perfect failure detection is unfeasible in Internet style networks, and that a failure does not mean only the crash of a process, but also a broken link of communication between two processes, implying non-transitive networks. Because of failures, we cannot trust the stability of the whole system to a single node, or to a reduced set of nodes with some hierarchy. We need to build self-managing decentralized systems, where data storage needs to be replicated and load balanced across the network in order to provide fault tolerance.

The contribution we present in this paper is Beernet, a development framework for decentralized systems that uses our structured overlay network topology, called Relaxed-ring (Mejias & Van Roy, 2008). The relaxed-ring deals with non-transitive connectivity, making it suitable for Internet applications. On top of the relaxed-ring, we implement a replicated storage with the symmetric replication strategy designed by (Ghodsi, 2006). To keep replicas consistent, we developed a transactional support which offers three different protocols: two-phase commit, paxos consensus algorithm (Moser & Haridi, 2007) and paxos with eager locking. The validation of the first two protocols, and the design and implementation of the last one is also part of our contribution. We will explain in detail why we needed eager locking. We also describe decentralized applications developed with Beernet as validation of the programming framework, and to analyze the different scenarios that need different transactional support.

**SELF MANAGEMENT**

The complexity of almost any system is proportional to its size. This rule also holds for distributed systems. As systems grow larger, they become more and more difficult to manage. Therefore, increasing systems’ self-manageability appears as a natural way of dealing with high level complexity. By self management, we mean the ability of a system to modify itself to handle changes in its internal state or its environment without human intervention but according to high-level management policies. This means that human intervention is lifted up to the level where policies are defined.

Typical self-management operations are: tune performance, reconfigure, replicate data, detect failures and recover from them, detect intrusion and attacks, add or remove parts of the system, which can be components within a process, or a whole peer, and others. Each of those actions or a combination of them can be identified as self-configuration, self-organization, self-healing, self-tuning, self-protection and self-optimization, often called in literature self-* properties.

One of the key operations that a system must perform to achieve self-managing behaviour is to monitor itself and its environment. Once relevant information is collected, the system can take decisions over which action to trigger to achieve its goal. Once the action is triggered, the system needs to monitor again to observe the effect of its action, developing a constant feedback loop.

A basic example of a self-managing system working with a feedback loop is air conditioning. A thermometer is constantly monitoring the temperature of the room. The information is given to a thermostat, where the desired tem-
Related Content

A Rough-Sets Approach to Kansei Evaluation Modeling and Design Support
[www.igi-global.com/article/rough-sets-approach-kansei-evaluation/46651?camid=4v1a](www.igi-global.com/article/rough-sets-approach-kansei-evaluation/46651?camid=4v1a)

A New Service Mediator For Human Resource Management
[www.igi-global.com/article/a-new-service-mediator-for-human-resource-management/120577?camid=4v1a](www.igi-global.com/article/a-new-service-mediator-for-human-resource-management/120577?camid=4v1a)

A Simulation-Based Analysis of Electricity Access in Uganda
[www.igi-global.com/article/a-simulation-based-analysis-of-electricity-access-in-uganda/126253?camid=4v1a](www.igi-global.com/article/a-simulation-based-analysis-of-electricity-access-in-uganda/126253?camid=4v1a)

Machines Paying Attention
[www.igi-global.com/chapter/machines-paying-attention/28372?camid=4v1a](www.igi-global.com/chapter/machines-paying-attention/28372?camid=4v1a)