

Using Peer-to-Peer Scalable Techniques to Increase Service Availability in SIP Distributed Systems

PhD Thesis

Voichița Almășan
voichita.iancu@cs.pub.ro

Supervisor: **Prof. Dr. Ing. Iosif Ignat** {iosif.ignat}@cs.utcluj.ro

*Technical University of Cluj-Napoca, Automation and Computer Science Faculty, Computer
Science Department
July 2011*

Keywords: highly available service, fault tolerance,
scalability, load balancing, peer-to-peer, SIP.

Abstract

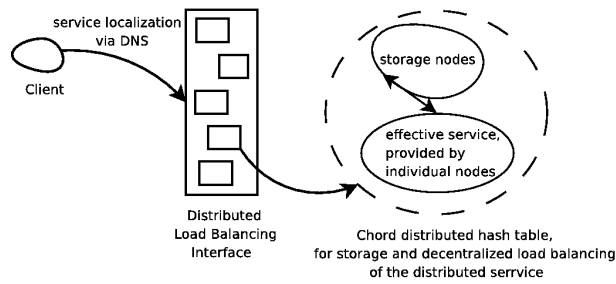


Figure 1: The general method to distribute and decentralize a service, in order to make it highly available.

1 Introduction

The present PhD thesis aims to present the main problems of a service, offered by a distributed system, among which we have identified *availability* as a main issue. After describing and analyzing the aspects that impact availability, in the bibliographical study, this thesis identifies original solutions, based on a logical peer-to-peer infrastructure. Along the extended document we argue why we consider *decentralization* as being the key to high availability within distributed systems. We also suggest an original, generic mechanism for transparency with respect to the external clients regarding decentralization, by providing a Single System Image (SSI). The generic technique was applied and validated by using a SIP (Session Initiation Protocol) server.

2 Increasing flexibility, scalability and load balancing for a distributed service, by transparent decentralization

As can be seen in figure 1, this chapter presents the original method for obtaining a highly available service by assembling independent elements, among which also own contributions presented also in this thesis. This method should be generic enough to be applied to any distributed service, with a minimal amount of changes.

We present, in brief, our original idea, which is based mainly on the advantages of flexibility and dynamicity characteristic to peer-to-peer networks, which have been combined and optimized in our own way.

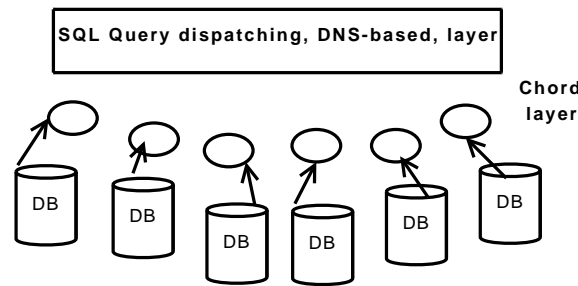


Figure 2: Level architecture of a decentralized distributed database.

3 The distributed database to store reliably and persistently the data belonging to a highly available service

3.1 Building a totally decentralized database, based on the Chord DHT

Data persistence is an important need for any distributed application. The necessity of a *decentralized* database is vital for assuring a successful data storage, vital to any data processing application and most of all by a highly available service. We know that the Chord DHT is a viable solution for data storage, and, by applying to it our original improvements, we have obtained a decentralized distributed database which also provides persistence and has been illustrated schematically in figure 2.

There exist more advantages to our original solution, among which we state: (1) the fact that it introduces a relatively low overhead, compared to traditional, centralized, databases; (2) the fact that, given a very large number of storage peers, interconnected through a high-speed network, the prototype's performances are influenced neither by the network's dimension, nor by peer volatility; (3) the fact that our solution hides the exact location of data and its copies, which increases database security; and (4) the fact that our solution guarantees atomic transactions for any supported SQL queries.

3.2 Improving storage and information security by means of game theory.

For the decentralized database presented in 3.1, we design a mechanism for reliable storage, which maintains the data unaltered in spite of the possible presence of malicious storage nodes. We propose an original voting algorithm, based on Byzantine consensus, which involves very little amount of extra messages exchanged between the

participating storage nodes. Our algorithm is able to identify and exclude malicious nodes from the storage infrastructure by working closely with the Chord network stabilization mechanism, thus eliminating the need of a central authentication authority, which is a major advantage for a highly available service.

3.3 Decoupled query handling, at client side, of the queries addressed to a database

For the distributed database described here, we identify 2 levels of querying: (1) the SQL level; and (2) the Chord level. In order to be able to handle performantly as much clients as possible, it would be best decouple the SQL service from the Chord service. In our thesis we describe a decoupling layer between the 2 types of services, which is based of the separation of 2 layers: (1) foreground useful service; and (2) background query execution; among which there is an RPC-like communication. There is a simple interface between the 2 layers, which offers functionalities similar to DMA from the hardware level. While evaluating this model, we concluded that the decoupling has good performances, since the overhead introduced is insignificant when compared to the time needed to execute the query. It has also been noticed that our proposed original decoupling model minimizes the risk of some DoS attacks.

4 Taking advantage of the peer-to-peer networks' dynamics, to ensure better data storage

4.1 Improving storage reliability of a Chord network by means of self-extension

We present in this section an original model which can be used when there exists a need for making a peer-to-peer storage infrastructure more fault tolerant, especially when it comes to nodes storing sensitive data. The original replication mechanism is based on a self-extending Chord DHT, which benefits both from the intrinsic Chord stabilization mechanisms and from the behaviour of special newly-launched peers, which ensure extra copies for sensitive data within the system. We obtained a prototype which is closely related to the original DHT, as can be seen from figure 3, having the purpose of efficiently storing sensitive data, by not allowing them to disappear from the system, in most of the functioning situations.

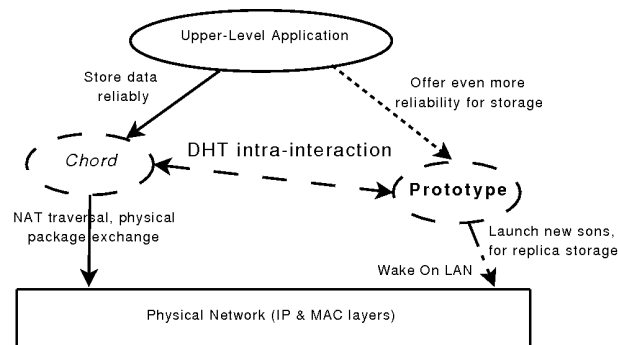


Figure 3: Interactions among the different entities involved in persistent sensitive data storage.

4.2 Self-extension of the Chord DHT, for load balancing purposes

Because the hash function is the one used for deciding which data belongs to which peer, we can see it as a *distributed load balancer* in itself, placed in front of the Chord network. In this chapter we have taken advantage of the benefits brought by the hash function in order to balance the storage load between the peers. We have elaborated and evaluated a load balancing model which allows better adaptability to increasing data storage needs. We have achieved this by using the Chord DHT to which we have added extra storage capacity when it was needed, as can be seen from figure 4. This extra storage capacity is represented by: (1) either newly integrated peers, added through self-extension mechanisms in the best place to help overloaded peers; (2) or extra physical storage, represented by installing supplementary or larger hard disks on the overloaded node. Both of these load balancing methods help the DHT to accept easier and faster to store increasing volumes of data, while keeping the same interface for the external clients.

5 Applying the generic decentralization model for a distributed server to a SIP server, in order to make it highly available

For reasons that have been presented in the extended document, in this thesis we have decided to apply the generic method for obtaining more availability, to a SIP presence server. By sticking to the generic model, we have designed an original way to get more availability for a the SIP presence server, by placing a load balancer in front of the Chord

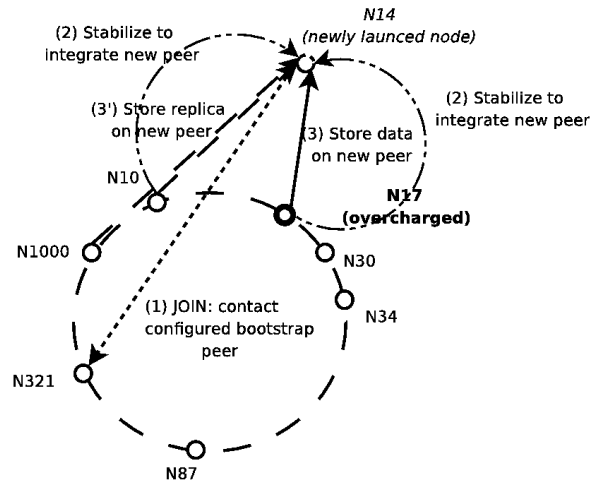


Figure 4: Adding a new Chord peer, to help an overloaded one.

ring on whose nodes SIP presence servers are run. Each load balancing node, which has been added to the DNS entry regarding the SIP service for the current domain, performs the following steps: (1) takes new SIP presence messages from clients, by using a minimal SIP presence server implementation; (2) for any SIP client, it parses the SIP message header, to identify the current SIP UAC's URI and also of the UAC whose state we wish to find out, plus the SIP presence method: PUBLISH or SUBSCRIBE; (3) if the method is PUBLISH, the URI that will be furtherly used is the one of the source and if the method is SUBSCRIBE, the URI that will be furtherly used is the other URI that appears in the message; (4) based on the SIP URI that should be used, the Chord key will be computed by means of the DHT's hash function, to identify the node that is responsible for this URI; (5) the current dispatcher node sends, via Chord, the whole SIP message to the responsible Chord peer, which also runs a SIP presence server; (6) the receiving presence server will answer to the message according to the SIP protocol, as follows: (6.a) if the client is behind a NAT, the presence server will save this information for it, and the answer will be sent via the dispatcher node; or (6.b) otherwise, the answer will be sent directly to the client, and the conversation will continue directly, excluding the dispatcher node from the conversation.

6 Contributions and conclusions

In the previous chapters we have identified, analyzed and handled problems with impact upon the availability of a distributed service. The handled aspects are: (1) data persistence; (2) fault tolerance; (3) scalability; (4) load balancing; and (5) system secu-

rity. To all these, along this PhD thesis, we offer original, generic solutions, based on a decentralized peer-to-peer-like architecture. We would like to stress out that all of the original contribution entailed by this thesis are orthogonal to one another and can, thus, be also used independently.

We shall present briefly the main original contributions of the present work:

1. In section 3.3: an original solution to decouple query execution, so that a server that needs to contact another server to be able not to block while waiting for the answer. The requesting server can, thus, handle other clients until it gets the needed answer from the distant server. This original contribution is useful because no matter what service we would like to provide, the designed server will always need to interact with other servers, providing different services. This is why it is important to decouple our server from any distant servers, whose behaviours we cannot control.
2. In section 3.1: an original solution for persistent data storage, by using a decentralized distributed database, based on the peer-to-peer Chord architecture. The Chord peers control independent nodes, which run single-node ordinary database servers. This original solution is useful and necessary because any service, and especially highly available services, need to store its data as long and as reliable as possible.
3. In section 3.2: an original solution to insure a degree of safety and confidence as large as possible, for data storage, even given the presence of possibly malicious nodes. This original solution applies to a Chord architecture and defines a distributed consensus protocol, made suitable for this architecture. This solution is useful and necessary because any service, and especially highly available services, need a degree of confidence as large as possible for storing its data, no matter what storage nodes are involved.
4. In section 4.1: an original solution to ensure fault tolerance of individual system nodes and also to increase system scalability. Fault tolerance is especially increased for storage of the system's critical data. This original solution is based on the Chord DHT architecture, to which we apply a self-extension mechanism, which is invoked in order to increase the degree of redundancy for storing critical data. The solution is useful and necessary since any highly available service should assure fault tolerance regarding especially data storage.
5. In section 4.2: an original solution to ensure load balancing among nodes, thus increasing system scalability. Our solution is based on the Chord DHT, by exploiting as much as we can the distributed hashing advantages. Thanks to the hash function, our solution manages to integrate, when needed, through auto-extension, new resources, especially at the place where they are mostly needed with respect

to the node load. This original solution is necessary and useful because any highly available distributed service should handle node load and, thus, system scalability with respect to the number of clients it can handle.

6. In chapter 2: an original solution to combine the advantages of the other original contributions presented throughout the thesis, which handle: (1) data persistence; (2) trust; (3) fault tolerance; (4) load balancing; and (5) scalability. We introduce a generic method, based on DNS and improved DHTs, to have a single point of access to the service, thus making its decentralization transparent to its clients, which, in turn, will benefit from the service's increased availability. This original solution is necessary and useful because it offers a generic "recipe" for increasing availability, which can apply to any distributed service. Moreover, this solution is validated not only by experimental results, but also by effectively applying it to a SIP presence server.
7. In chapter 5: an original solution to obtain a highly available SIP service, by applying the generic decentralization method, which increases availability, to it. This is the final objective of this thesis and it is necessary and useful because of the emergency of this protocol in solutions that try to take the place of phone carriers, at a comparable degree of availability, while having significantly lower implementation and usage costs.

Based on the results we obtained from experimental evaluation, we can conclude that the main objective of the thesis has been fulfilled. The observed performance impact, noticed for increasing availability, is generally comparable to the one observed for a regular Chord DHT, which is usually considered to be acceptable for a fully decentralized distributed system.