Transparent Byzantine Fault-Tolerant Directory Service using COTS components

Francisco Vieira, Paulo Sousa, Alysson Neves Bessani
University of Lisboa, Faculdade de Ciências, LaSIGE - Portugal
fvieira@lasige.di.fc.ul.pt, {pjsousa,bessani}@di.fc.ul.pt

Abstract

A directory service is a critical component of any distributed computing infrastructure given that its failure may lead to the inaccessibility of many network services. In this work, we propose an architecture that allows to add Byzantine fault tolerance to existing directory services in a transparent way: neither directory services nor client applications need to be modified. Moreover, this architecture can be implemented using COTS components.

1. Introduction

A directory service is a network service that associates names (e.g., users', servers', printers') with arbitrary attributes (e.g., password, telephone number, IP address, features) and supports queries based on names and/or combinations of attributes (e.g., what is the IP address of server bob.mycompany.com?, which printers have a duplex module installed?). Directory services perform thus a fundamental role and their failure (due to, e.g., bugs, hardware faults) may provoke the unavailability/unacessibility of other services.

Directory services are deployed in different contexts, ranging from companies local networks to the Internet. While in the former different solutions can be implemented (e.g., based on LDAP[1]), in the latter a single service has been in use since the debut of the Internet: DNS[2]. DNS is in fact a name service given that it associates names with a fixed (not arbitrary) set of attributes, the most important being IP addresses. Previous works have shown how to make DNS more secure [1], namely on how to make it tolerate arbitrary (Byzantine) faults.

In this paper we study how to increase the dependability of *generic* directory services through Byzantine-fault-tolerant state machine replication [2], [3]. We propose an architecture for Byzantine-fault-tolerant (BFT) directory services that uses COTS[3] components and maintains transparency with regard to client applications (i.e., client applications do not need to be changed in order to access a BFT directory service). Moreover, Byzantine fault tolerance is added as a layer external to the directory service itself, which means that the directory service does not require modifications.

Starting from a baseline configuration where distributed client applications access a non fault-tolerant directory service (e.g., OpenLDAP) deployed in a single server, the basic idea is to replicate the service in a set of servers and to use a BFT replication library for the communication between clients and servers. Transparency is achieved by taking advantage of the design of an already existing standard interface that is widely deployed nowadays.

2. Architecture

JNDI[4] [4] is the standard interface used by Java applications to access directory services. It is designed to be independent of any specific directory service implementation. The JNDI architecture consists of an API (Application Programming Interface) and an SPI (Service Provider Interface) [5]. Java applications use the JNDI API to access directory services. The JNDI SPI enables a variety of directory services to be plugged in transparently, virtually allowing a Java application to access any directory service for which there is an SPI implementation. Figure 1 depicts the typical usage scenario of JNDI. Client applications use the JNDI API to send standard commands (e.g., queries), these commands are converted to the directory service (possibly proprietary) protocol by the the corresponding SPI (a kind of driver), which communicates with the remote directory service server through TCP.

We took advantage of the modularity that the JNDI design offers to introduce Byzantine fault tolerance in a transparent way. A BFT replication library is typically composed of client- and server-side components. Transparency is typically broken when Byzantine fault tolerance is added to a non fault-tolerant system because the client-side component needs to be integrated with the normal software that runs on the clients. With JNDI, this problem can be circumvented by implementing a specific SPI driver that acts as a bridge between the JNDI API and the BFT replication library client-side component. From the point of view of the client

1. Lightweight Directory Access Protocol.
2. Domain Name System.
3. Commercial, Off-The-Shelf.
4. Java Naming and Directory Interface.

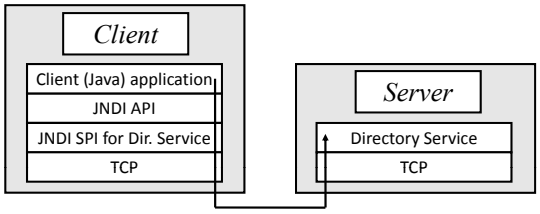

Figure 1. Architecture of a non fault-tolerant directory service accessed through JNDI.

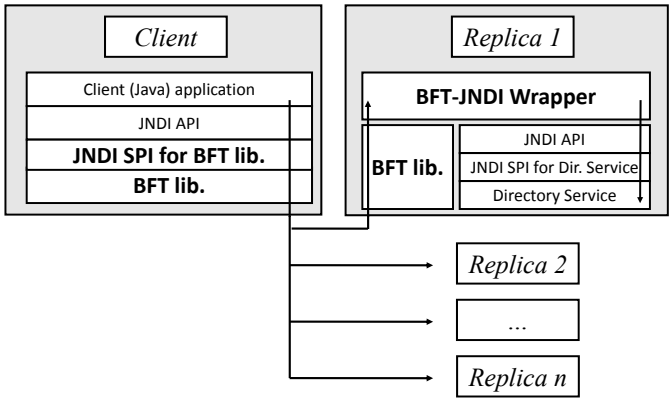

Figure 2. Architecture of a (transparent) Byzantine fault-tolerant directory service (new components in bold).

application, the interface continues the same: the JNDI API. Figure 2 presents the proposed architecture for a transparent Byzantine fault-tolerant directory service.

On the client slide, applications access the JNDI API as before, but we replace the directory service specific SPI driver by another one that simply forwards JNDI requests to the client component of the BFT library. This component multicasts each request to all server replicas and the BFT library ensures that all replicas receive all the requests by the same order even if a subset of them are faulty. Typically, one needs $n \geq 3f + 1$ replicas to tolerate f faulty replicas [6]. These requests are processed by a BFT-JNDI wrapper that uses the normal JNDI API to finally send them to the directory service.

One interesting characteristic of this architecture is that it allows the inclusion of any directory service (for which a SPI driver exists) in a transparent way. This means that different replicas may use diverse directory services, reducing the probability of common mode failures in all replicas at the same time.

3. Prototype

We have implemented a prototype of the architecture described in Section 2 using COTS components. Our prototype uses JBP[5] (Java Byzantine Paxos) [7] as the BFT replication library and OpenLDAP[6] as the directory service. We have done a preliminary comparison of the performance of OpenLDAP with and without Byzantine fault tolerance, and the overhead is minimal. We are currently conducting a more comprehensive evaluation.

4. Conclusions and Future Work

We have presented an architecture that allows to enhance a directory service with Byzantine fault tolerance capabilities in a transparent way. This architecture has interesting characteristics, not only because it does not require any changes on existing client applications, but also because it is fully implementable with COTS components and different directory services may be used in the various replicas without too much additional effort.

As future work, we will enrich the described prototype with other COTS directory service products, and conduct an extensive experimental evaluation using simulated and real-world data.

Acknowledgment

This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs, and by the University of Lisboa - Fundação Amadeu Dias scholarship program.

References

[1] C. Cachin and A. Samar, "Secure distributed DNS," in *34th IEEE International Conference on Dependable Systems and Networks (DSN 2004)*, Jun. 2004, pp. 423–432.

[2] F. B. Schneider, "Implementing faul-tolerant services using the state machine approach: A tutorial," *ACM Computing Surveys*, vol. 22, no. 4, pp. 299–319, Dec. 1990.

[3] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[4] Sun Microsystems Inc, "The JNDI tutorial," 2008, http://java.sun.com/products/jndi/tutorial/.

[5] Sun Microsystems Inc., "Java Naming and Directory Interface - Service Provider Interface (JNDI SPI)," Jul. 1999.

[6] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[7] A. N. Bessani, E. P. Alchieri, M. Correia, and J. da Silva Fraga, "Depspace: A Byzantine fault-tolerant coordination service," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Systems Conference - EuroSys 2008*, Apr. 2008.

5. Available at http://www.navigators.di.fc.ul.pt/software/jitt/jbp.html.
6. Available at http://www.openldap.org/.