Designing and Evaluating Fault-tolerant Leader Election Algorithms *

Christian Fernández-Campusano, Roberto Cortiñas, Mikel Larrea
Computer Architecture and Technology Department
University of the Basque Country UPV/EHU
20018 San Sebastián, Spain
{*christianrobert.fernandez, roberto.cortinas, mikel.larrea*}*@ehu.es*

Jian Tang
Distributed Systems Laboratory (LSD)
Technical University of Madrid
28031 Madrid, Spain
tjapply@gmail.com

Abstract

Fault-tolerant leader election is a basic building block for dependable distributed computing, since it allows coordinating decisions among replicas such that they remain consistent. Indeed, several fault-tolerant agreement protocols rely on an eventual leader election service. This problem has been initially studied in crash-prone systems, and more recently in other failure scenarios, e.g., crash-recovery, omission and Byzantine. Most of the previous works have considered those failure models individually. This paper presents our ongoing work on the design and evaluation of distributed leader election algorithms in systems prone to both crash-recoveries and message omissions.

1 Introduction

A common technique to provide highly available and fault-tolerant services in current distributed systems consists in replicating their critical components, such that if a component fails then one of the replicas takes over and the service continues uninterruptedly. Replication has clear benefits, but also requires adequate coordination protocols in order to keep replicas consistent.

The consensus problem [11] is a paradigm for many coordination problems in fault-tolerant distributed computing. Informally, in consensus processes propose an initial value and, despite failures, have to decide on one of the proposed values. A fundamental result by Fischer et al. [6] showed that consensus cannot be solved deterministically in asynchronous systems where at least one process may crash. Several approaches have been proposed to circumvent this impossibility, usually assuming some form of weak (partial) synchrony in the system, either explicitly [4] or encapsulated in some abstract mechanism, e.g., unreliable failure detection [2] or eventual leader election [1]. Paxos [8], and more recently Raft [10], are two examples of consensus protocols relying on an eventual leader election mechanism. These protocols are indulgent [7], i.e., they always preserve safety and guarantee liveness as soon as a unique leader remains for sufficiently long in the system.

In this work we address the design and evaluation of fault-tolerant leader election algorithms in partially synchronous systems prone to process crash-recoveries and message omissions, in order to support leader-based consensus protocols.

2 System Model and Definitions

We consider a system model composed of a finite and totally ordered set of processes that communicate only by sending and receiving messages through reliable links that cannot create or alter messages. We assume that the system is partially synchronous, i.e., there exist upper bounds on processing time and on message communication delay, although those bounds are not known a priori by processes [4].

*Research supported by the Spanish Research Council (grants TIN2013-41123-P and TIN2013-46883-P), the Basque Government (grant IT395-10), the University of the Basque Country UPV/EHU (grant UFI11/45), and the Community of Madrid (grant CLOUD4BIGDATA, S2013/ICE-2894).

Processes can fail by crashing. Crashes are not permanent, i.e., crashed processes can recover. In every execution of the system, we have the following three types of processes: (1) eventually up, i.e., processes that eventually remain up forever, (2) eventually down, i.e., processes that eventually remain crashed forever, and (3) unstable, i.e., processes that crash and recover an infinite number of times. We assume that processes have access to stable storage that maintains its content despite crashes. Processes can also fail by omission at sending and/or receiving messages. Omissions can be selective, i.e., with respect to some given process, or general. Also, omissions can be transient or permanent.

Indulgent consensus protocols require a majority of correct processes. Hence, we will assume a majority of eventually up processes in the system. Note that these processes may omit some messages. However, we assume that there is a time after which a majority of eventually up processes stop omitting messages.

Our focus on the crash-recovery and omission failure models makes our results more general and of practical application. Indeed, in real systems failed processes usually recover, and messages can get omitted due to collisions, buffer overflow or network saturation. Moreover, we have shown recently that Byzantine failures can be reduced to omission failures by relying on a distributed security mechanism, e.g., by equipping nodes with a tamper-proof smartcard [3].

Finally, the eventual leader election problem is defined by the following property [9]: there is a time after which (1) every correct process always trusts the same correct process ℓ, and (2) every incorrect process always trusts either $\perp$ (i.e., it does not trust any process) or ℓ.

3 Work in Progress

We have recently designed a distributed leader election algorithm according to the system model and definitions presented in the previous section [5]. In that algorithm, processes exchange messages periodically, detecting omissions through sequence numbers. Messages also carry the incarnation value of the sender, which allows detecting crash-recoveries. The incarnation value is stored in stable storage, and incremented upon recovery. A process is a candidate to become the leader only if it communicates without omissions with a majority of processes. Among the candidates, the process with lowest incarnation value is selected as leader.

We are currently evaluating the performance of this algorithm in order to get initial references. We are interested in studying the scalability of the algorithm for different scenarios and failure patterns, measured in terms of the number of messages exchanged among nodes. Also, we want to study the responsiveness of the algorithm, measured in terms of the latency until a new leader is elected after the current leader fails.

Other lines of research are the following:

- From a practical perspective, we are interested in designing leader election algorithms where processes do not have access to stable storage, as well as more efficient algorithms, in which among correct processes eventually only the elected leader keeps on sending messages. These algorithms will be of application in ubiquitous computing scenarios (sensors, wearable devices...) as those envisioned by the Internet of Things.
- From a more theoretical perspective, we are interested in determining the weakest system model for implementing a leader election service.

References

[1] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, 1996.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[3] R. Cortiñas, F. C. Freiling, M. Ghajar-Azadanlou, A. Lafuente, M. Larrea, L. D. Penso, and I. Soraluze. Secure failure detection and consensus in trustedpals. *IEEE Trans. Dependable Sec. Comput.*, 9(4):610–625, 2012.

[4] C. Dwork, N. A. Lynch, and L. J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[5] C. Fernández-Campusano, M. Larrea, and R. Cortiñas. A distributed leader election algorithm in crash-recovery and omissive systems. Technical report, 2014. EHU-KAT-IK-05-14, University of the Basque Country UPV/EHU. Available at http://www.sc.ehu.es/acwlaalm/.

[6] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[7] R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Trans. Computers*, 53(4):453–466, 2004.

[8] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[9] M. Larrea, C. Martín, and I. Soraluze. Communication-efficient leader election in crash-recovery systems. *Journal of Systems and Software*, 84(12):2186–2195, 2011.

[10] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014*, pages 305–319, 2014.

[11] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.