J. Parallel Distrib. Comput. 93–94 (2016) 1–9

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Read/write shared memory abstraction on top of asynchronous Byzantine message-passing systems☆

Damien Imbs [a], Sergio Rajsbaum [e], Michel Raynal [b,c,*], Julien Stainer [d]

[a] Department of Mathematics, Bremen University, Germany
[b] Institut Universitaire de France, France
[c] IRISA, Université de Rennes, 35042 Rennes, France
[d] Distributed Programming Lab, EPFL, Lausanne, Switzerland
[e] Instituto de Matemáticas, UNAM, Mexico

HIGHLIGHTS

- First Byzantine-rrsilient read/write memory.
- Introduction of new concurrency objects.
- An impossibility result.

ARTICLE INFO

Article history:
Received 26 August 2015
Received in revised form
11 February 2016
Accepted 28 March 2016
Available online 6 April 2016

Keywords:
Approximate agreement
Asynchronous message-passing system
Atomic read/write register
Broadcast abstraction
Byzantine process
Distributed computing
Message-passing system
Quorum
Reliable broadcast
Reliable shared memory
Single-writer/multi-reader register
t-Resilience

ABSTRACT

This paper is on the construction and use of a shared memory abstraction on top of an asynchronous message-passing system in which up to t processes may commit Byzantine failures. This abstraction consists of arrays of n single-writer/multi-reader atomic registers, where n is the number of processes. These registers enable Byzantine tolerance by recording the whole history of values written to each one of them. A distributed algorithm building such a shared memory abstraction is first presented. This algorithm assumes $t < n/3$, which is shown to be a necessary and sufficient condition for such a construction. Hence, the algorithm is resilient-optimal.

Then the paper presents distributed objects built on top of this read/write shared memory abstraction, which cope with Byzantine processes. As illustrated by these objects, the proposed shared memory abstraction is motivated by the fact that, for a lot of problems, algorithms are simpler to design and prove correct in a shared memory system than in a message-passing system.

1. Introduction

Distributed computing and abstraction: Distributed computing occurs when a set of sequential *processes* (sometimes called nodes, processors, processes, agents, sensors, etc.) cooperate to solve a problem. If processes do not cooperate, the system is no longer a distributed system. Hence, a distributed system has to provide the processes with communication and agreement mechanisms.

Understanding and designing distributed applications is not an easy task [3,15,24–26]. This is because each process is a physically distinct entity that has only a partial knowledge of the ongoing computation. No process can see the global state of the system. More precisely, as processes are geographically localized at distinct places, distributed applications have to cope with the uncertainty created by asynchrony and failures. A main difficulty is that it is impossible to distinguish a crashed process from a very slow process in an asynchronous system prone to process crashes.

☆ A preliminary version of the results presented in this paper appeared in the proceedings of the 18th International Colloquium on Structural Information and Communication Complexity (SIROCCO 2014) (Imbs et al., 2014, [10]).

* Corresponding author at: IRISA, Université de Rennes, 35042 Rennes, France.
E-mail address: raynal@irisa.fr (M. Raynal).

http://dx.doi.org/10.1016/j.jpdc.2016.03.012
0743-7315/

As in sequential computing, a common approach to facilitate the design of distributed applications consists in designing appropriate abstractions. The fundamental paper [1] presents the first asynchronous fault-tolerant abstraction of shared memory on top of a message passing system. While designing distributed applications in either of these models is difficult, it is easier in shared-memory systems, where processors have a more global view of the system. Indeed, this paper substantially impacted both the theory and the practice of distributed systems.

The simulation of [1] shows that there is no fundamental distinction between message passing and shared memory computation. However, this simulation of shared memory in a message passing system is tolerant of crash (unexpected halting) failures only. In recent years it has become more and more important to design systems that tolerate arbitrary, even malicious process failures. The goal of this paper is to show that it is possible to have an emulation of shared memory on top of a message passing system, even in such difficult conditions.

Byzantine behavior: A process has a *Byzantine* behavior when it arbitrarily deviates from its intended behavior; it then commits a Byzantine failure. This bad behavior can be intentional or simply the result of a combination of hardware and/or software errors that altered the behavior of a process in an unpredictable way. Let us notice that process crashes and communication omissions, define a strict subset of Byzantine failures. This failure type was initially studied in the context of synchronous distributed systems (e.g., [14,23,25]), and then investigated in the context of asynchronous ones (e.g., [3,15,24]). Most of the early research considers (synchronous or asynchronous) message-passing systems, and mainly addresses agreement problems, such as consensus and total order broadcast. Only recently general task solvability results have appeared [19,20,27].

Content of the paper: The contribution of this paper is the definition of a shared memory composed of atomic registers in the context of Byzantine processes, and the design of an algorithm that builds such a shared memory on top of an asynchronous message-passing system where up to t processes may be Byzantine. We introduce a technique to achieve this simulation, in terms of *h-registers* (h standing for "history"). These registers differ from classical registers in that each read returns the complete history of writes to the registers. This allows preventing a malicious process from overwriting a previously written value letting correct processes believe it wrote it only once. This t-resilient shared memory is made up of n single-writer/multi-reader (SWMR) atomic h-registers (one per process). The paper also shows that $t < n/3$ is a necessary and sufficient requirement for such a construction.

The threshold of $t < n/3$ for the existence of a construction of a shared memory on top of an asynchronous Byzantine message-passing system quantifies the inherent cost of moving from tolerating crash failures to tolerating arbitrary failures. In the case of crash failures, it is possible to construct an atomic shared memory on top of an asynchronous message-passing system if and only if $t < n/2$, where t is the largest number of processes that may crash [1]. Interestingly, the $t < n/3$ requirement is the same as the one for solving consensus in both Byzantine synchronous systems [14] and Byzantine asynchronous systems (enriched with an appropriate oracle such as a common coin) e.g., [6,21,22].

To show the benefit of the previous construction to obtain easy solutions to distributed computing problems, that run on a Byzantine message passing system, the article presents two simple shared memory algorithms, which are pretty simple, one solving the "one-shot write-snapshot" problem, the other one solving the "correct-only" agreement problem (a weakened version of the consensus problem), respectively. Both illustrate how one can design algorithms that tolerate crash failures and use them almost directly on top of the h-register abstraction to obtain Byzantine-tolerant solutions running of a message-passing system.[1]

As shown by these examples, the important feature of the proposed shared memory abstraction lies in the fact that it prevents Byzantine processes from corrupting synchronization among the correct processes. A Byzantine process can create inconsistency only on the values it writes, but any two correct processes see the same sequence of written values.

Related work: This paper is on the construction of a shared memory (atomic registers) on top of an asynchronous message-passing system where processes may exhibit a Byzantine behavior. A related active research area concerns the construction of Byzantine-tolerant disk storage (e.g., [7,12,16–18]). The focus is on registers built on top of replicated disks (servers), which are accessed by clients, and where disks and clients may exhibit different type of failures (e.g., [2]). In these client/server models, clients communicate only with servers and vice versa.

The reason why different types of failures are addressed in [2,12] comes from the fact that these papers consider "classical" read/write registers, namely, a register may be modified any number of times, and each read must return the last value that was written. In this context, a Byzantine client can overwrite a value it has previously written, and trick a correct client into believing that it wrote only once. To prevent this bad scenario, [12] considers clients that can fail only by crashing, while [2] restricts clients to be "semi-Byzantine": they can issue a bounded number of faulty writes, but otherwise have to respect the code of their algorithm.

Roadmap: The paper is composed of six sections. Section 2 introduces the underlying Byzantine asynchronous message-passing model. Section 3 defines the notion of Byzantine-tolerant atomic read/write registers, and presents an algorithm that builds such registers on top of the basic Byzantine asynchronous message-passing model. This section shows also that $t < n/3$ is a necessary and sufficient requirement for such a construction. Then, Section 4 and presents two simple algorithms that solve distributed computing problems on top of Byzantine-tolerant atomic registers. Finally, the last section concludes the paper.

2. Computation model, reliable broadcast abstraction, and two quorum properties

2.1. Computation model

2.1.1. Computing entities

The system is made up of a set Π of n sequential processes, denoted $p_1, p_2, \ldots, p_n$. These processes are asynchronous in the sense that each process progresses at its own speed, which can be arbitrary and remains always unknown to the other processes.

2.1.2. Communication model

The processes cooperate by sending and receiving messages through bi-directional channels. The communication network is a complete network, which means that each process p_i can directly send a message to any process p_j (including itself). It is assumed that, when a process receives a message, it can unambiguously identify its sender. Each channel is reliable (no loss, corruption, or creation of messages), but asynchronous (while the transit time of each message is finite, there is no upper bound on message transit times). In each channel, messages are not necessarily received in the order in which they were sent. Moreover, Byzantine processes are not prevented from reading all messages and reordering them.

[1] The interested reader can find a more involved example that solves multidimensional approximate agreement on top of the t-resilient shared memory abstraction in [10,11]. This algorithm can be seen as an adaptation (to a Byzantine read/write shared memory system) of the algorithm described in [19], which solves the same problem "directly" on top of an asynchronous Byzantine message-passing system.