The Journal of Systems and Software 112 (2016) 96–109

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Performance optimization for state machine replication based on application semantics: A review

Wenbing Zhao*

Department of Electrical Engineering and Computer Science, Cleveland State University, 2121 Euclid Avenue, Cleveland, OH 44115, USA

ARTICLE INFO

Article history:
Received 21 July 2015
Revised 2 November 2015
Accepted 3 November 2015
Available online 12 November 2015

Keywords:
Application semantics
State machine replication
Fault tolerance
Byzantine agreement

ABSTRACT

The pervasiveness of cloud-based services has significantly increased the demand for highly dependable systems. State machine replication is a powerful way of constructing highly dependable systems. However, state machine replication requires replicas to run deterministically and to process requests sequentially according to a total order. In this article, we review various techniques that have been used to engineer fault tolerance systems for better performance. Common to most such techniques is the customization of fault tolerance mechanisms based on the application semantics. By incorporating application semantics into fault tolerance design, we could enable concurrent processing of requests, reduce the frequency of distributed agreement operations, and control application nondeterminism. We start this review by making a case for considering application semantics for state machine replication. We then present a classification of various approaches to enhancing the performance of fault tolerance systems. This is followed by the description of various fault tolerance mechanisms. We conclude this article by outlining potential future research in high performance fault tolerance computing.

1. Introduction

The pervasiveness of cloud-based services has significantly increased the dependability expectations of many computer systems not only from business partners, but also from many millions of end-users. Such systems must be made resilient against various hardware failures and possibly against cyber attacks as well. State machine replication has shown to be an effective technique to help achieve this goal (Zhao, 2014c). Highly efficient fault tolerance algorithms have been developed to tolerate both crash faults and Byzantine faults, the most seminal of which are Paxos (Lamport, 1998, 2001) and PBFT (Castro and Liskov, 2002). However, state machine replication requires replicas to run deterministically and to process requests sequentially according to a total order. We argue that these constraints could impede the adoption of fault tolerance techniques in practice, for example

- Practical systems often involve nondeterministic operations when they execute clients' requests, and states of the replicas could diverge if the application nondeterminism is not controlled.
- Executing requests at the replicated server sequentially according to a total order often results in unacceptable low system throughput and high end-to-end latency.

To overcome these issues, the application semantics must be considered, as demonstrated by a large number of fault tolerance systems (Burrows, 2006; Castro and Liskov, 2002; Chandra et al., 2007; Cowling et al., 2006; Kotla et al., 2009; Li et al., 2012; MacCormick et al., 2004; Moraru et al., 2013; Zhao, 2009; Zhao et al., 2009). For example, most of them have focused on building fault tolerant storage systems such as file systems and database systems, where a lighter weight mechanism is designed to handle read-only requests while update requests are totally ordered. The identification of read-only requests requires the knowledge of the application semantics of a particular application. In some systems, such as a networked file system (NFS), some requests would trigger the access of the local physical clock, which constitutes a nondeterministic operation. In Castro and Liskov (2002), the application semantics of NFS was used to identify this nondeterministic operation and to design a control mechanism accordingly.

The need for incorporating application semantics is further showcased by recent works on fault tolerance systems designed to operate in wide-area networks (Li et al., 2012; Moraru et al., 2013). In Li et al. (2012), a hybrid replica consistency model is used to reduce the end-to-end latency and enhance the system throughput for wide-area fault tolerance systems based on state machine replication. In this hybrid model, commutative requests (referred to as Blue operations) are executed locally and the operations are asynchronously disseminated to other replicas using an eventual consistency model. On the other hand, a strong consistency model is used for requests

* Tel.: +12 165237480; fax: +12 166875405.
E-mail address: wenbing@ieee.org

http://dx.doi.org/10.1016/j.jss.2015.11.006
0164-1212/

that have inter-dependencies (referred to as Red operations). Furthermore, some operations that are not directly commutative are converted into commutative operations (the converted operations are referred to as shadow operations) for even better performance. The determination on whether or not two requests are commutative, and the transformation of some pairs of requests into commutative requests, all require the knowledge of application semantics.

In Moraru et al. (2013), a new variation of the Paxos consensus algorithm, referred to as Egalitarian Paxos (or EPaxos), was proposed for state machine replication. EPaxos requires every request to indicate whether or not it is read-only, and to provide the set of requests on which it depends for each update request. Total ordering is needed for conflicting requests only and non-conflicting requests are executed concurrently, thereby, significantly increasing the system throughput. It is apparent that application semantics is essential for EPaxos to work properly.

We start this review by elaborating why considering application semantics for state machine replication is necessary. We then propose a classification of various approaches to building practical fault tolerance systems. This is followed by the description of the actual performance engineering mechanisms under the classification framework. We conclude this article by outlining potential future research in high performance fault tolerance computing.

This article is based on our previous work in attempting to classify existing approaches to designing practical Byzantine fault tolerance systems by incorporating application semantics. We are not aware of other work that aims to provide a systematic review of this subject. In Zhao (2014c), we coined the term "application-aware Byzantine fault tolerance" to refer to this line of work, and compiled known research works roughly based on their complexity. In Zhao (2014b), we proposed our first classification framework. In this article, we have expanded the classification to include both conservative Byzantine fault tolerance and optimistic Byzantine fault tolerance. Furthermore, we have widened the scope of the classification to include state machine replication with both the crash fault and Byzantine fault models.

The ultimate goal of our research is to provide a guideline on designing high performance practical fault tolerance systems by identifying when a distributed agreement is needed, and on what operations. This is analogous to the challenge of building a secure system, where one must know when and where to use security primitives, such as encryption and digital signatures. We believe that it is time to treat distributed agreement algorithms as basic building blocks for dependable systems the same way as security primitives to secure systems. The use of security primitives alone does not warrant a secure system. Similarly, the use of distributed agreement in a naive manner does not necessarily enhance the dependability of a system. It is essential to know exactly when to use the tool and on what operations. This cannot be accomplished without considering application semantics.

The contributions of this review include

- We present a strong argument that it is not practical to treat the sever application as a black box when replicating it for fault tolerance and, that one cannot ignore the application semantics in dependability design. In addition to performance overhead and replica consistency issues, we identify scenarios when the replicated system may deadlock if requests are executed sequentially.
- We propose a classification framework for various performance engineering approaches to fault tolerance, together with the description of the mechanisms used in these approaches. They could serve as a guideline on designing practical dependable systems.
- We outline potential future research directions that could reduce the cost of application semantics discovery and the maintenance of custom fault tolerance implementations.

2. Why application semantics matters

Fault tolerance algorithms designed for state machine replication concern only the total ordering of requests to be delivered to the replicated server replicas. Hence, such algorithms can be used by any application as long as the replicated component acts deterministically as a state machine, *i.e.,*given the same request delivered in the same total order, all replicas would go through the same state transitions (if any), and produce exactly the same reply. However, this does not mean that we should treat each application as a black box and employ a fault tolerance algorithm as it is by totally ordering all requests and executing them sequentially according to the total order. In the following, we present three major motivations for exploiting application semantics in fault tolerance.

2.1. Reduce runtime overhead

There are two main types of runtime overhead introduced by state machine replication based fault tolerance algorithms

1. Communication and processing delays for each remote invocation due to the need for total ordering of requests, which would impact the end-to-end latency.
2. The loss of concurrency degrees at the replicated server due to the sequential execution of requests, which impacts the system throughput (*i.e.,*how many requests can be handled by the replicated server per unit of time).

By exploiting application semantics, we can introduce the following optimizations:

- Reducing the end-to-end latency by not totally ordering all requests. There is no need to totally order read-only requests (*i.e.,*requests that do not modify the server state). In addition, for some stateless session-oriented applications, source ordering of requests may be sufficient (Chai and Zhao, 2013).
- Enabling concurrent processing at the server replicas for some requests. Non-conflicting requests can be executed concurrently (Kotla and Dahlin, 2004; Raykov et al., 2011). Doing source ordering alone for requests also increases the system throughput (Chai and Zhao, 2013).
- Furthermore, deferred agreement for session-oriented applications (using one distributed agreement instance for a group of requests typically at the end of the session), could further increase the system throughput (Zhang et al., 2012).

2.2. Respect causality and avoid deadlocks

General purpose fault tolerance algorithms are designed for simple client-server applications where clients do not directly interact with each other and send requests to the replicated server independently. For multi-tiered applications with sophisticated interaction patterns (Chai and Zhao, 2012b), the basic assumption for these fault tolerance algorithms may no longer hold and hence, if used in a straightforward manner, may lead to two problems: (1) causality violation, when the total order imposed on two or more requests is different from their causal order and, (2) deadlocks, when sequential execution of requests is imposed when concurrent processing of some requests is mandatory according to the application design (Zhao et al., 2005a). In these cases, application semantics must be tapped to discover the causal ordering between requests and identify what requests must be delivered concurrently. Otherwise, the integrity and the availability of the system could be lost.

2.3. Control replica nondeterminism

State machine replication requires that replicas behave deterministically when processing requests. However, many applications are