The Cost of Consensus on Synchronization of Distributed Systems

***Abstract*—**
***Index Terms*—**

I. Introduction

Coordination is essential for large-scale distributed applications. Handling the simplest operations in a distributed manner gave rise to unprecedented challenges. Different forms of coordination are used to handle a variety of tasks. Leader election and group membership is one way. Worrying about aspects of synchronization, concurrency, and distributed management is a huge burden on application developers. This is why many distributed coordinators were designed to be leveraged by those developers, such as ZooKeeper and Chubby. Other packages focus on one primitive, or aspect, of distributed coordination such as Amazon Simple Queue Service that focuses on queueing.

Locking is a powerful coordination primitive. It guarantees mutual exclusion when accessing critical sources. However, it is also widely used to provide a mean of synchronization between distributed applications. The choice of synchronization primitive is not an easy decision. Different applications have different characteristics. The amount of contention for example is cruicial on the choice of synchronization primitive and is highly dependent on the application type. The computing environment is of importance too. The latency of coordination and consensus have an effect on the performance of different primitives. The topic of synchronization protocols' pros and cons and comparison of both are widely studied in the literature of multiprocessors.

here is a need to reinvestigate synchronization protocols for large-scale distributed systems. In these systems communication latency can reach hundreds of milliseconds. This dramatic difference to the conventional multiprocessor environment might carry with it new revelations on the community's prejudice on synchronization protocols. General distributed coordination packages delivers basic coordination primitives to end users. ZooKeeper provides a simple API to manipulate hierarchically organized wait-free data objects, resembling a file system. These manipulations are guaranteed to be FIFO ordered and writes are linearizable. Using these primitives allow users creating more complex coordination primitives (*e.g.*, synchronization primitives). Chubby, on the other hand, provides locking with strong guarantees.

In this paper, we carry the first steps into realizing the question of synchronization protocols in distributed systems. We leverage ZooKeeper to coordinate between different machines. Synchronization protocols are then implemented using ZooKeeper's primitives. In our study we will display protocols shortcomings in different operation conditions. The protocols we will be focusing on are test-and-set and queues. After we map each one of these two to a favorable operation condition, we lay the ground for a reactive mechanism that, according to current operation, choose the better protocol to manage synchronization.

II. Framework

A. testbed

describe machines and topology

B. consensus protocol

talk about consensus

C. synchronization primitives

D. Zookeeper

III. Analysis of consensus cost

A. Round-trip time latency

display data we got from experiments (using ping for example) to get the distribution of RTTs. Maybe we should check with inter- and intra-datacenter communications. we discuss insights from the behavior of RTTs regarding their cost on consensus then we develop a model to describe RTTs, and develop a model to expect latency of getting all responses.

B. Consensus latency

developa model to describe the latency of requests (such as the ones used for our baseline case, like zookeeper's paper). this will develop over previous section in addition to accounting the effect of service times in clients.

C. Synchronization primitives latency

barriers, test-and-set, queues

IV. Experimental evaluation

A. Request latency

baseline experiment. effect of adding machines, effect of adding zookeeper servers.

B. synchronization primitives

test test-and-set and queues (as in paper: reactive synchronization

C. application performance

map reduce. effect of adding machines, effect of adding zookeeper servers.

V. Conclusions and Future Work

Lets cite [1].

References

[1] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proc. VLDB Endow.*, 1(2):1277–1288, August 2008.