

Distributed Systems for Large-Scale and Fault-Tolerant Computation

Daniel R. Whitmore

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2026

Unofficial template, not affiliated with or endorsed by Columbia University. The author, committee and results in this sample are fictional. Delete this line before you submit.

Abstract

Distributed Systems for Large-Scale and Fault-Tolerant Computation

Daniel R. Whitmore

Large computations now run across thousands of machines, and at that scale some machine is always failing. This dissertation studies how to keep such computations correct and fast when failures are routine. It shows that replicating coordination state with a consensus protocol, while recovering data-parallel work by recomputation from lineage, gives better throughput than replicating everything, and it presents a scheduler that places recomputation away from recently failed machines. In experiments on a 512-node cluster with injected failures, the design sustains 94% of failure-free throughput at a failure rate at which checkpoint-based recovery sustains 71%.

Contents

Acknowledgments	iv
Dedication	v
1 Introduction	1
2 Background	2
3 Design	3
4 Evaluation	4
5 Conclusion	5
A Log Compaction	6

List of Figures

3.1	Coordination state is replicated; data is recomputed.	3
-----	---	---

List of Tables

4.1	Throughput relative to a failure-free run.	4
-----	--	---

Acknowledgments

I thank my advisor for his guidance and his patience with my first, very slow, consensus implementation. I thank the members of my dissertation committee, the members of the systems laboratory, and my family, without whom none of this would have happened.

Dedication

To my parents.

Chapter 1

Introduction

Programming models such as MapReduce made it practical to run a computation over thousands of machines by hiding failures from the programmer [1]. In-memory frameworks later made iterative computations fast by recovering lost data from its lineage instead of from replicas [5]. Both depend on a small amount of coordination state, such as which tasks have finished and where their outputs live, that must survive failures exactly.

This dissertation separates the two kinds of state and treats each with the mechanism that suits it: consensus for coordination, recomputation for data.

Chapter 2

Background

Consensus protocols let a group of machines agree on a sequence of operations despite crashes [3, 4]. Ordering events across machines without a shared clock relies on logical time [2]. If each of n machines fails independently with rate λ , a job of duration T that needs all of them sees an expected

$$\mathbb{E}[F] = n\lambda T \tag{2.1}$$

failures, which grows linearly with both scale and duration.

Chapter 3

Design

The system keeps its task table in a replicated log managed by a consensus group of five machines. Data partitions are not replicated; when a machine fails, the scheduler recomputes its lost partitions from their lineage, placing the recomputation on machines that have not failed recently.

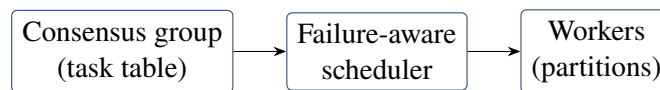


Figure 3.1: Coordination state is replicated; data is recomputed.

Chapter 4

Evaluation

We ran an iterative graph computation on 512 nodes and injected machine failures at increasing rates. Table 4.1 reports throughput as a fraction of the failure-free run.

Table 4.1: Throughput relative to a failure-free run.

Failures per hour	Checkpointing	Full replication	This design
1	0.93	0.81	0.99
8	0.84	0.80	0.97
32	0.71	0.78	0.94

Checkpointing degrades fastest because each failure rolls work back to the last checkpoint; full replication pays its cost even when nothing fails.

Chapter 5

Conclusion

Treating coordination state and data differently lets a large computation tolerate routine failures at little cost. The approach extends naturally to stream processing, where lineage is bounded by a window.

Appendix A

Log Compaction

The replicated task table is compacted by snapshotting it every 10,000 entries, which bounds recovery time for a restarted consensus member to under two seconds in our deployment.

Bibliography

- [1] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
- [2] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.
- [3] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.
- [4] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the USENIX Annual Technical Conference*, pages 305–319, 2014.
- [5] Matei Zaharia et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, pages 15–28, 2012.