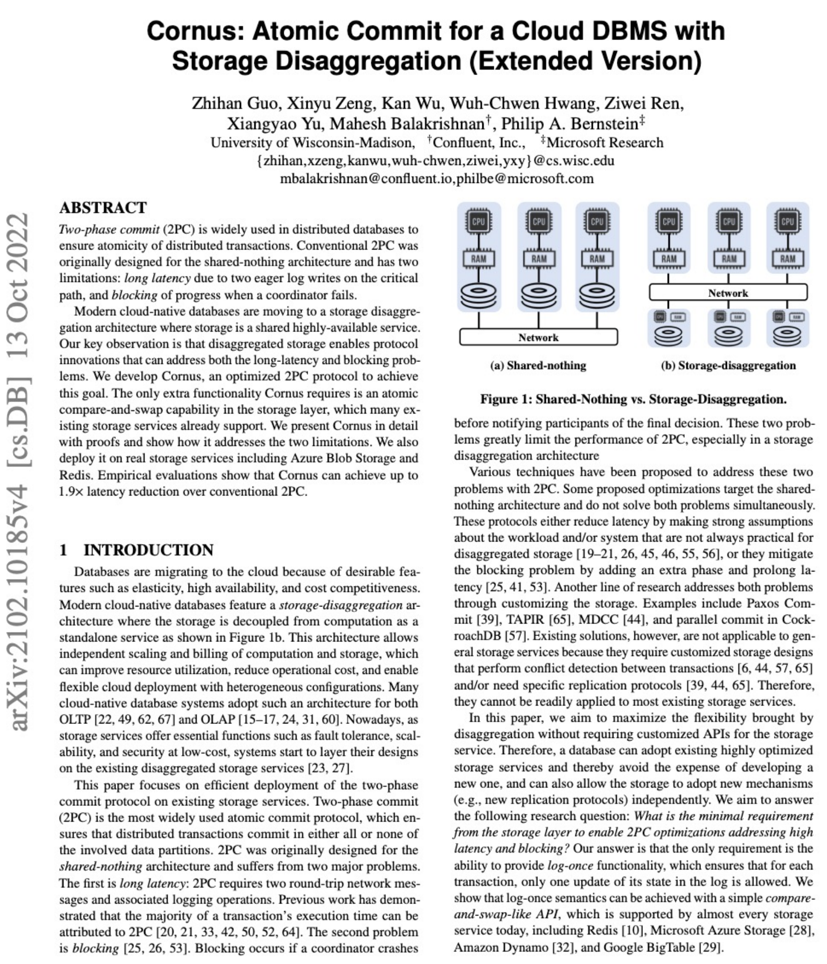
WISCONSIN
UNIVERSITY OF WISCONSIN–MADISON

CS 764: Topics in Database Management Systems

Lecture 21: Cornus

Xiangyao Yu
11/16/2022

Today’s Paper: Cornus



Cornus: Atomic Commit for a Cloud DBMS with Storage Disaggregation (Extended Version)

Zhihan Guo, Xinyu Zeng, Kan Wu, Wuh-Chwen Hwang, Ziwei Ren,
Xiangyao Yu, Mahesh Balakrishnan[†], Philip A. Bernstein[‡]
University of Wisconsin-Madison, [†]Confluent, Inc., [‡]Microsoft Research
{zhihan,xzeng,kanwu,wuh-chwen,ziwei,yxy}@cs.wisc.edu
mbalakrishnan@confluent.io,philbe@microsoft.com

ABSTRACT

Two-phase commit (2PC) is widely used in distributed databases to ensure atomicity of distributed transactions. Conventional 2PC was originally designed for the shared-nothing architecture and has two limitations: *long latency* due to two eager log writes on the critical path, and *blocking* of progress when a coordinator fails.

Modern cloud-native databases are moving to a storage disaggregation architecture where storage is a shared highly-available service. Our key observation is that disaggregated storage enables protocol innovations that can address both the long-latency and blocking problems. We develop Cornus, an optimized 2PC protocol to achieve this goal. The only extra functionality Cornus requires is an atomic compare-and-swap capability in the storage layer, which many existing storage services already support. We present Cornus in detail with proofs and show how it addresses the two limitations. We also deploy it on real storage services including Azure Blob Storage and Redis. Empirical evaluations show that Cornus can achieve up to 1.9× latency reduction over conventional 2PC.

(a) Shared-nothing

(b) Storage-disaggregation

Figure 1: Shared-Nothing vs. Storage-Disaggregation.

1 INTRODUCTION

Databases are migrating to the cloud because of desirable features such as elasticity, high availability, and cost competitiveness. Modern cloud-native databases feature a *storage-disaggregation* architecture where the storage is decoupled from computation as a standalone service as shown in Figure 1b. This architecture allows independent scaling and billing of computation and storage, which can improve resource utilization, reduce operational cost, and enable flexible cloud deployment with heterogeneous configurations. Many cloud-native database systems adopt such an architecture for both OLTP [22, 49, 62, 67] and OLAP [15–17, 24, 31, 60]. Nowadays, as storage services offer essential functions such as fault tolerance, scalability, and security at low-cost, systems start to layer their designs on the existing disaggregated storage services [23, 27].

This paper focuses on efficient deployment of the two-phase commit protocol on existing storage services. Two-phase commit (2PC) is the most widely used atomic commit protocol, which ensures that distributed transactions commit in either all or none of the involved data partitions. 2PC was originally designed for the *shared-nothing* architecture and suffers from two major problems. The first is *long latency*: 2PC requires two round-trip network messages and associated logging operations. Previous work has demonstrated that the majority of a transaction’s execution time can be attributed to 2PC [20, 21, 33, 42, 50, 52, 64]. The second problem is *blocking* [25, 26, 53]. Blocking occurs if a coordinator crashes before notifying participants of the final decision. These two problems greatly limit the performance of 2PC, especially in a storage disaggregation architecture

Various techniques have been proposed to address these two problems with 2PC. Some proposed optimizations target the shared-nothing architecture and do not solve both problems simultaneously. These protocols either reduce latency by making strong assumptions about the workload and/or system that are not always practical for disaggregated storage [19–21, 26, 45, 46, 55, 56], or they mitigate the blocking problem by adding an extra phase and prolong latency [25, 41, 53]. Another line of research addresses both problems through customizing the storage. Examples include Paxos Commit [39], TAPIR [65], MDCC [44], and parallel commit in CockroachDB [57]. Existing solutions, however, are not applicable to general storage services because they require customized storage designs that perform conflict detection between transactions [6, 44, 57, 65] and/or need specific replication protocols [39, 44, 65]. Therefore, they cannot be readily applied to most existing storage services.

In this paper, we aim to maximize the flexibility brought by disaggregation without requiring customized APIs for the storage service. Therefore, a database can adopt existing highly optimized storage services and thereby avoid the expense of developing a new one, and can also allow the storage to adopt new mechanisms (e.g., new replication protocols) independently. We aim to answer the following research question: *What is the minimal requirement from the storage layer to enable 2PC optimizations addressing high latency and blocking?* Our answer is that the only requirement is the ability to provide *log-once* functionality, which ensures that for each transaction, only one update of its state in the log is allowed. We show that log-once semantics can be achieved with a simple *compare-and-swap-like API*, which is supported by almost every storage service today, including Redis [10], Microsoft Azure Storage [28], Amazon Dynamo [32], and Google BigTable [29].

VLDB 2022

Announcement

No lecture on Wednesday next week

Optional 10-min meeting to discuss your project with instructor

Signup sheet (access using your UW account)

- https://docs.google.com/spreadsheets/d/1HatkCJkKUD8ZI0zVe_xZ9OxhfthrY6YAgiI9NX8uS9g/edit?usp=sharing

Meetings over zoom

- https://uwmadison.zoom.us/j/92584913804?pwd=NVdON0VjcWJLOTVwVk9UNzdRSURyZz09

Outline

Cloud database

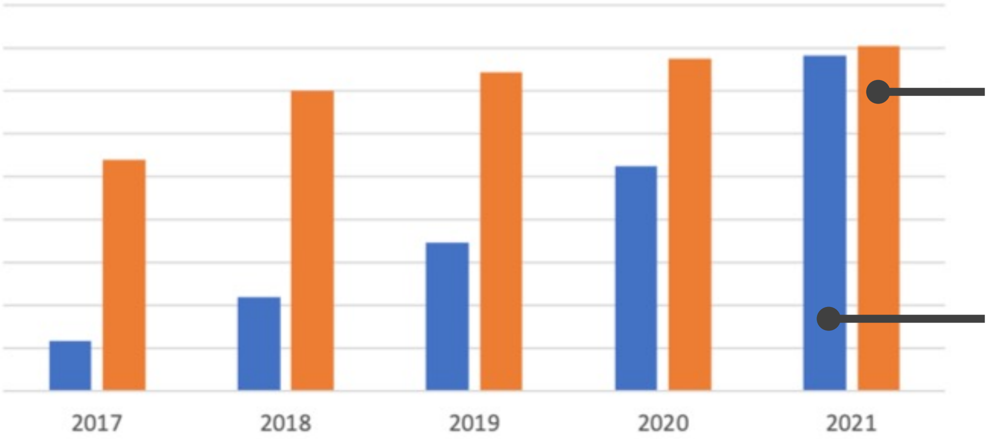
Storage disaggregation

Cornus protocol

Databases Moving to the Cloud

According to Gartner Report [1]

$39.2 billion, 49% of all DBMS revenue from cloud in 2021



Cloud vs. On-premises Revenue

On-prem database
$40.8B in 2021

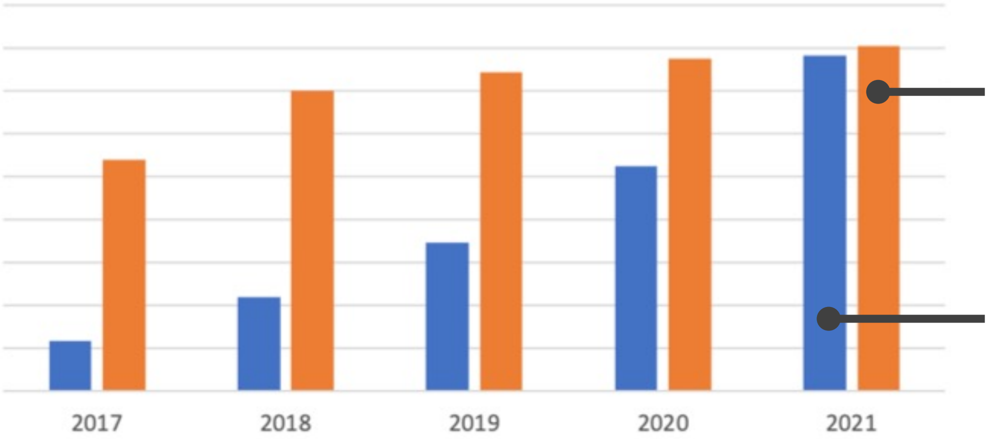
Cloud database
$39.2B in 2021

[1] DBMS Market Transformation 2021: The Big Picture, https://blogs.gartner.com/merv-adrian/2022/04/16/dbms-market-transformation-2021-the-big-picture/

Databases Moving to the Cloud

According to Gartner Report [1]

$39.2 billion, 49% of all DBMS revenue from cloud in 2021



Cloud vs. On-premises Revenue

On-prem database
$40.8B in 2021

Cloud database
$39.2B in 2021

Low Cost

Elasticity

Availability

[1] DBMS Market Transformation 2021: The Big Picture, https://blogs.gartner.com/merv-adrian/2022/04/16/dbms-market-transformation-2021-the-big-picture/

Databases Moving to the Cloud

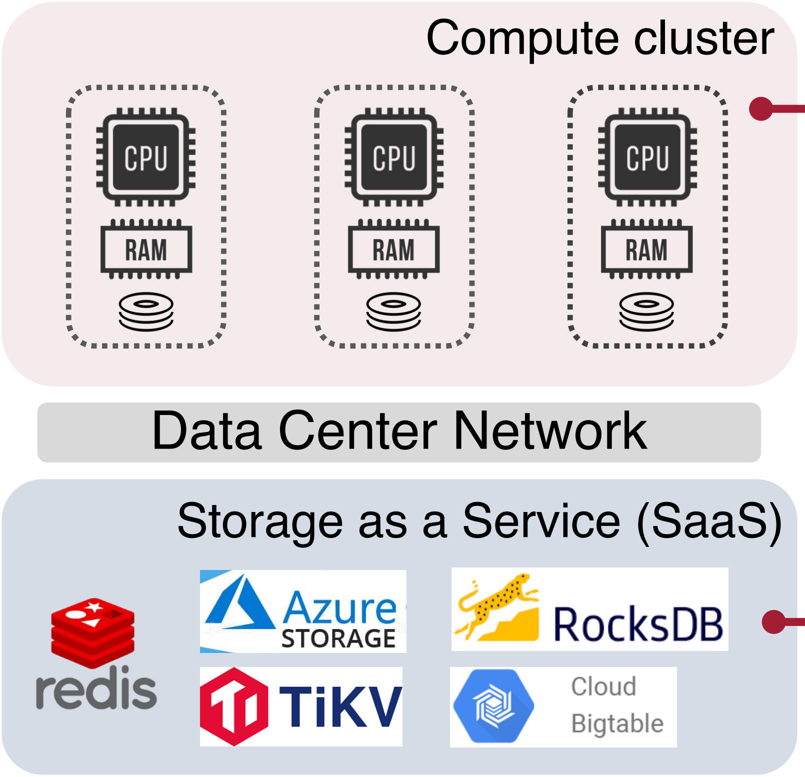
Transactional DB

- Amazon Aurora
- POLARDB
- ORACLE DATABASE CLOUD SERVICE
- Orleans
- Microsoft SQL Server
- GaussDB
- TiDB
- Cockroach DB
- Cloud Spanner
- yugabyteDB
- Azure Cosmos DB

Analytical DB

- Spark SQL
- snowflake
- mongoDB
- HIVE
- VERTICA
- presto
- Amazon Athena
- amazon REDSHIFT
- Google BigQuery
- trino

Cloud DB: Storage-Disaggregation

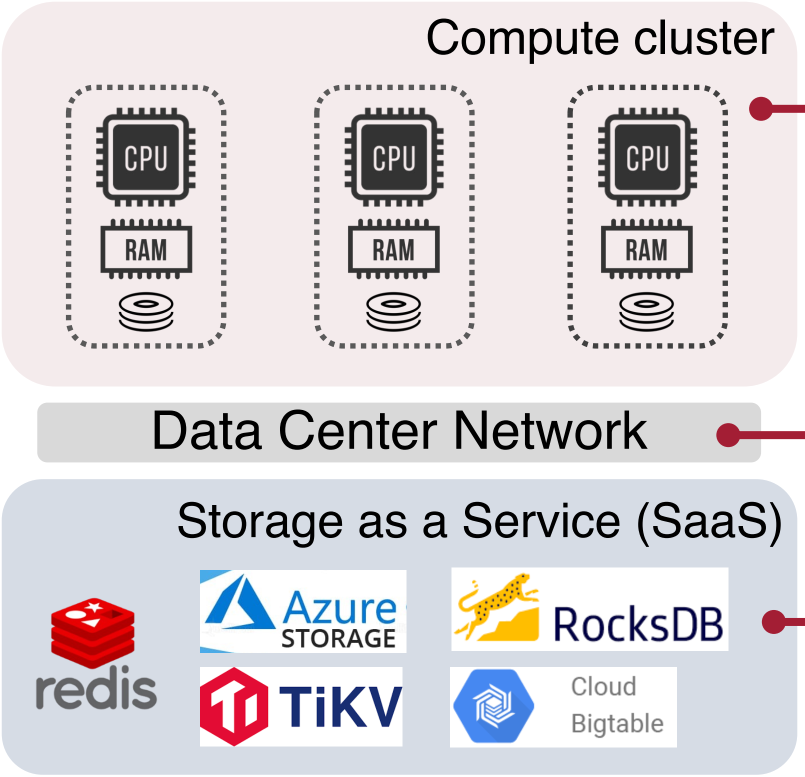


Database logic in computer cluster

Database states (e.g., tables and logs) in cloud storage service

Manage computation and storage as separate services

Cloud DB: Storage-Disaggregation



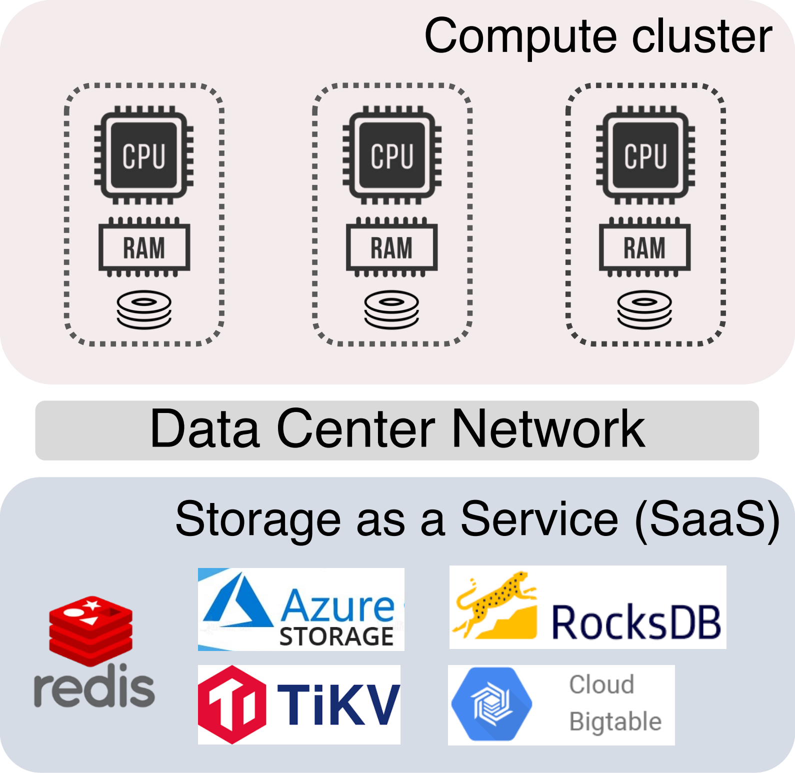
Database logic in computer cluster

Increasing network speed

Database states (e.g., tables and logs) in cloud storage service

Manage computation and storage as separate services

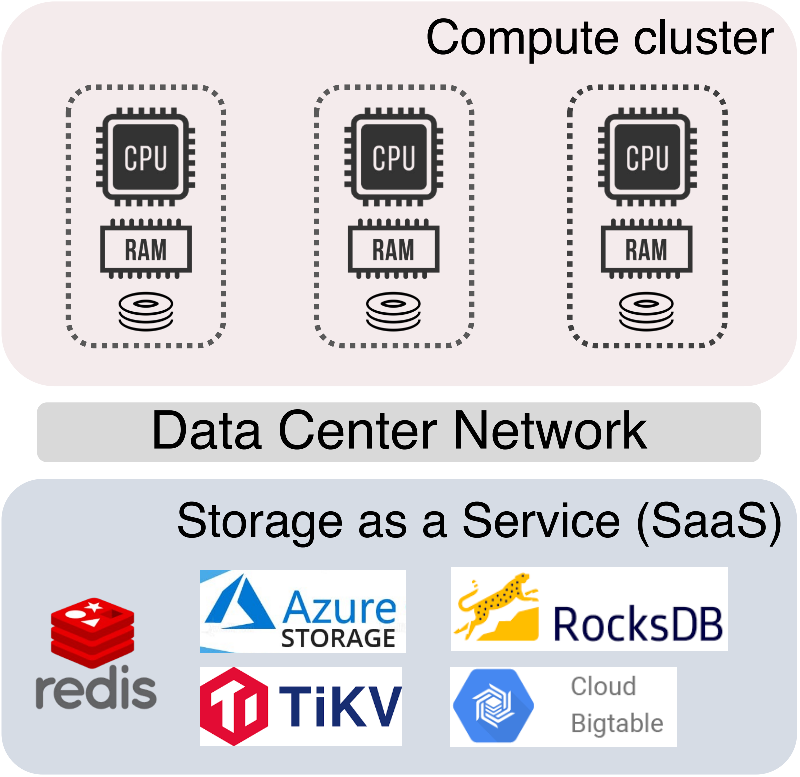
Advantages of Storage-Disaggregation



Advantage #1: Elasticity

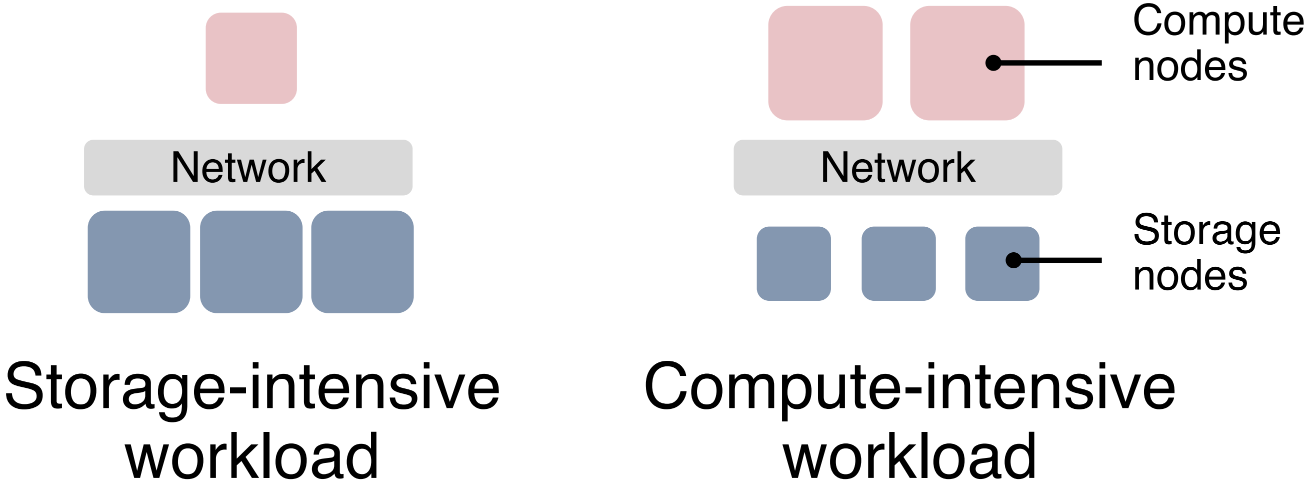
- Compute and storage resources can scale independently

Advantages of Storage-Disaggregation

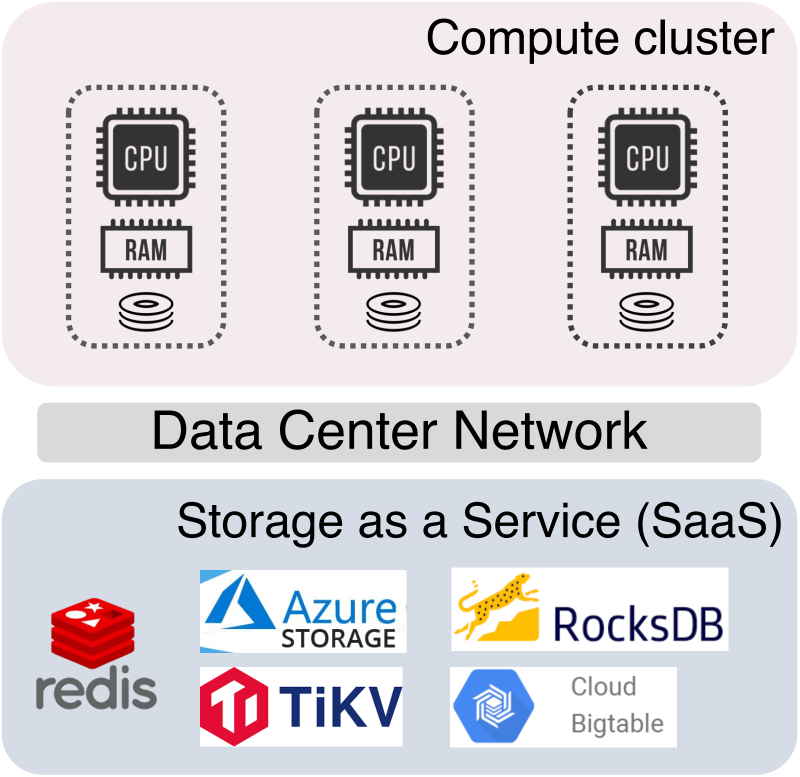


Advantage #1: Elasticity

- Compute and storage resources can scale independently



Advantages of Storage-Disaggregation

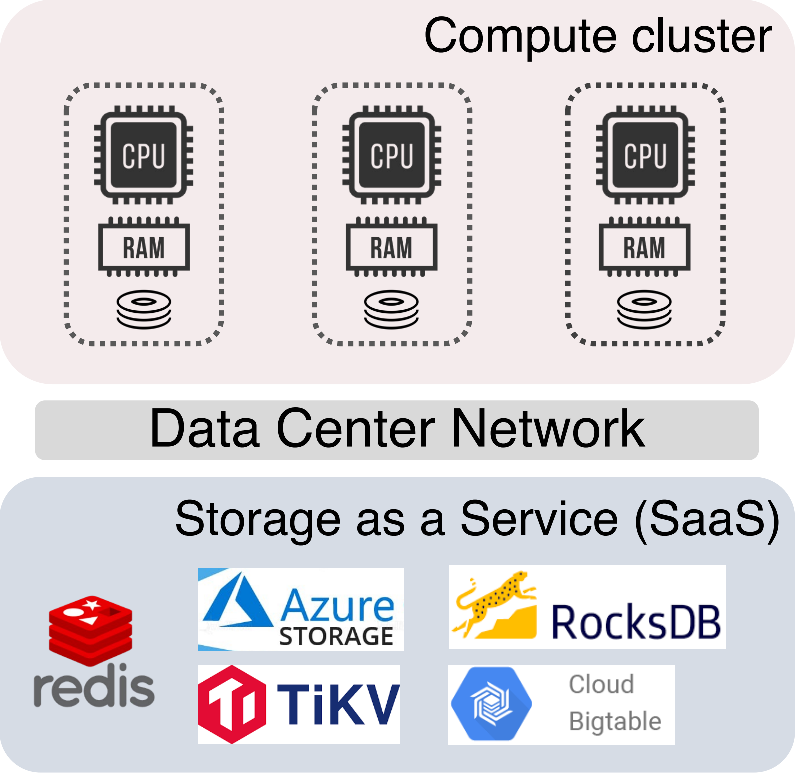


Advantage #2: Low Cost

- Storage service can be much cheaper than compute servers

S3 storage price	**$0.02** per GB per month
16 vCPU Virtual Machine	**$0.5** per hour per VM

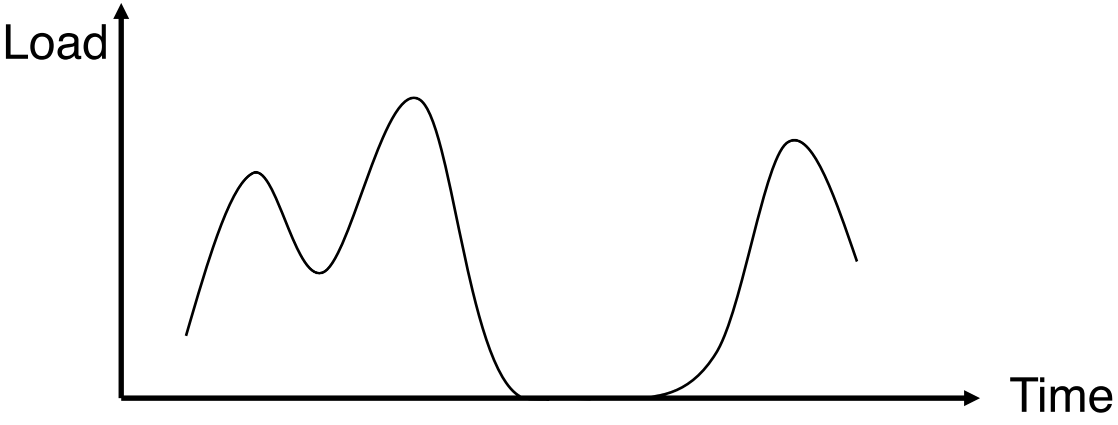
Advantages of Storage-Disaggregation



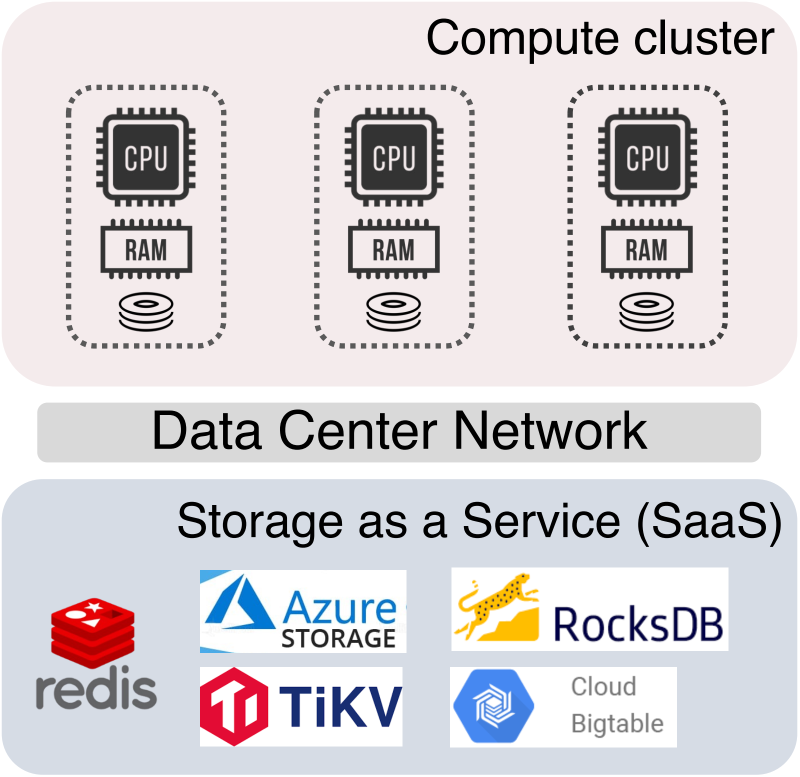
Advantage #2: Low Cost

- Storage service can be much cheaper than compute servers

S3 storage price	**$0.02** per GB per month
16 vCPU Virtual Machine	**$0.5** per hour per VM



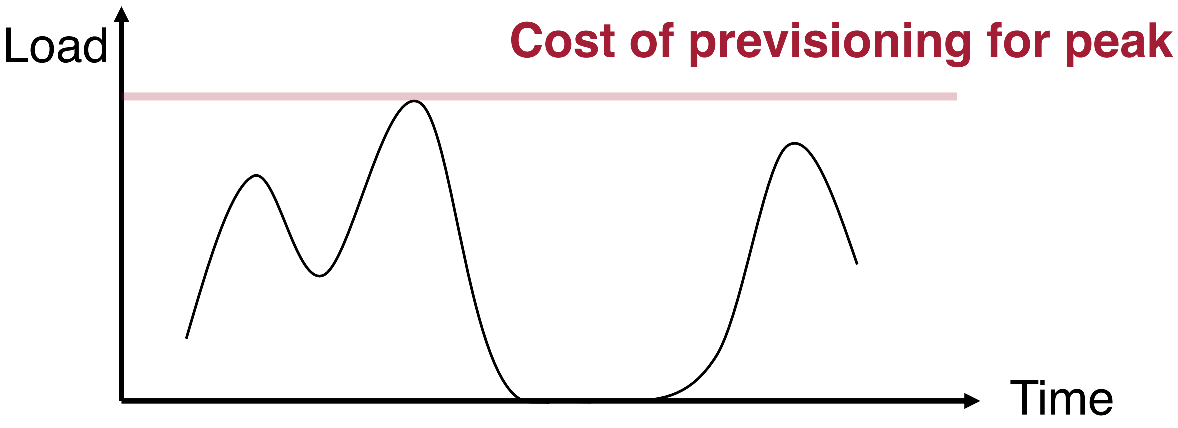
Advantages of Storage-Disaggregation



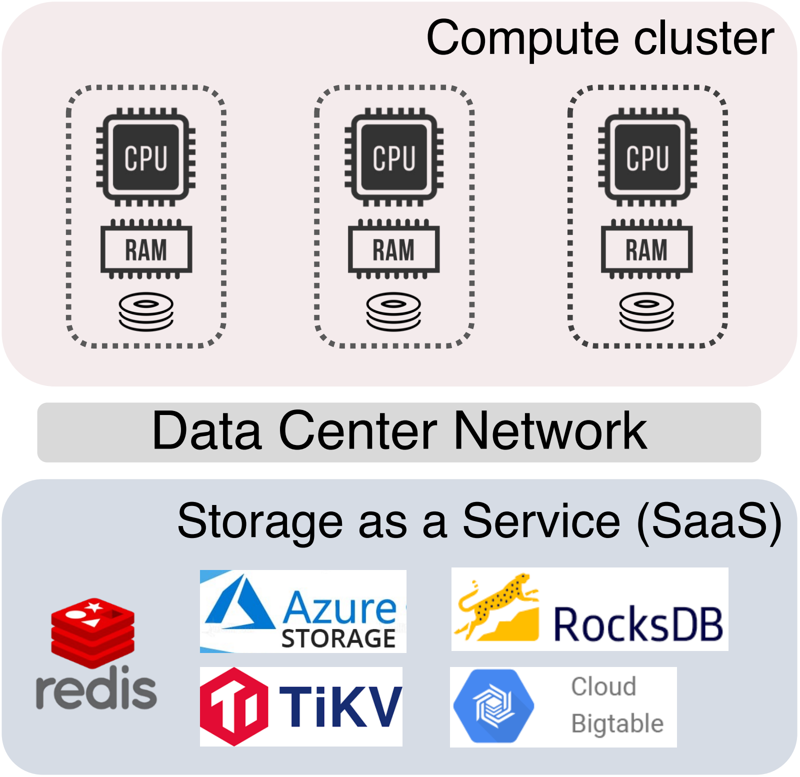
Advantage #2: Low Cost

- Storage service can be much cheaper than compute servers

S3 storage price	**$0.02** per GB per month
16 vCPU Virtual Machine	**$0.5** per hour per VM



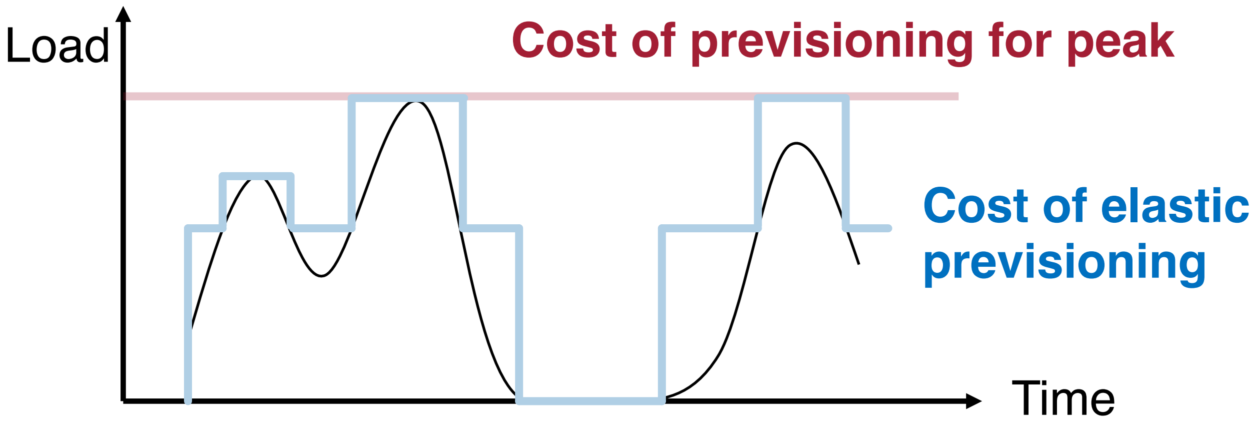
Advantages of Storage-Disaggregation



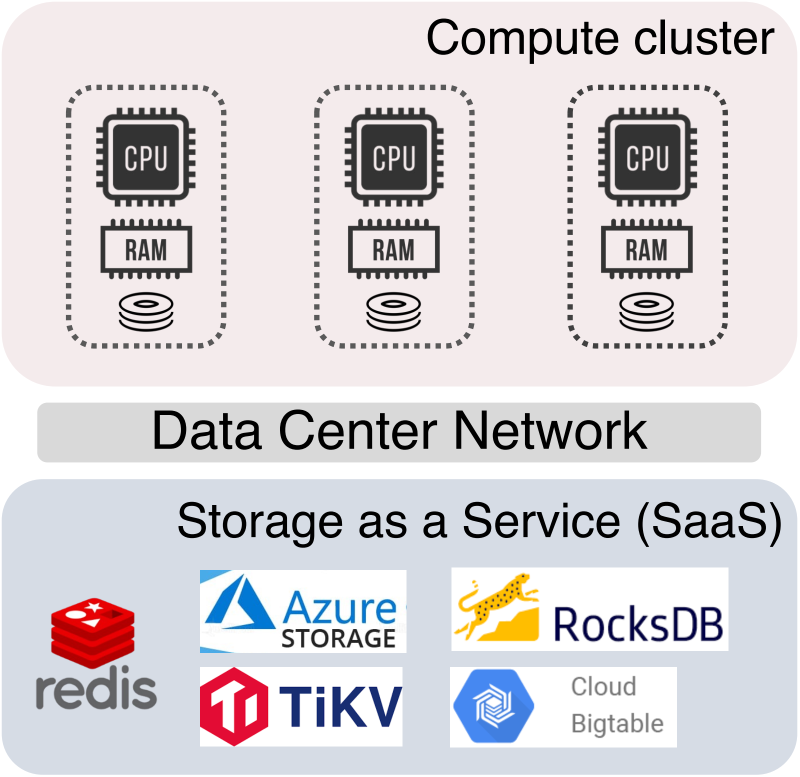
Advantage #2: Low Cost

- Storage service can be much cheaper than compute servers

S3 storage price	**$0.02** per GB per month
16 vCPU Virtual Machine	**$0.5** per hour per VM



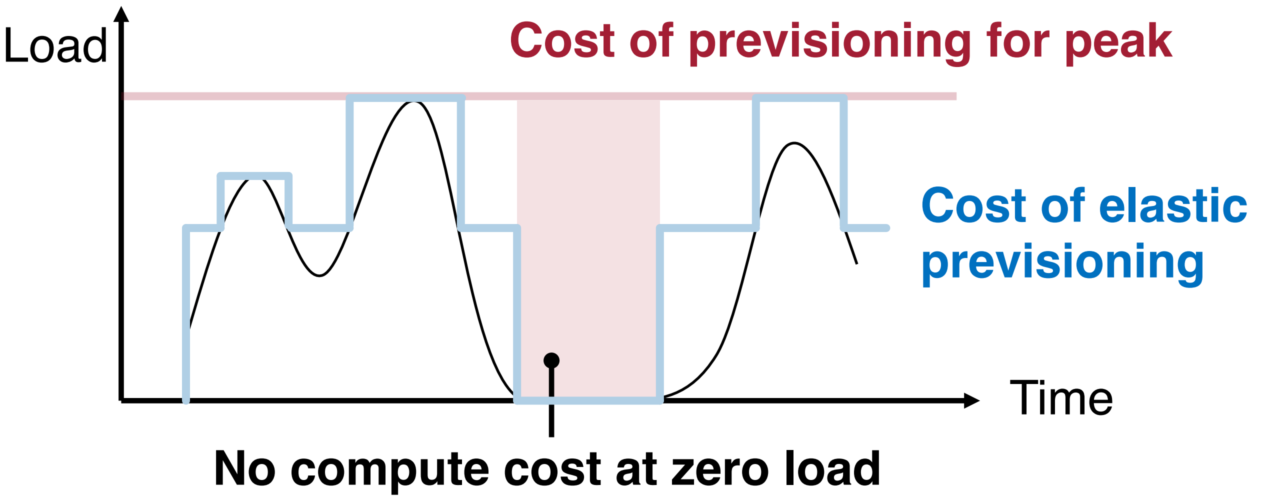
Advantages of Storage-Disaggregation



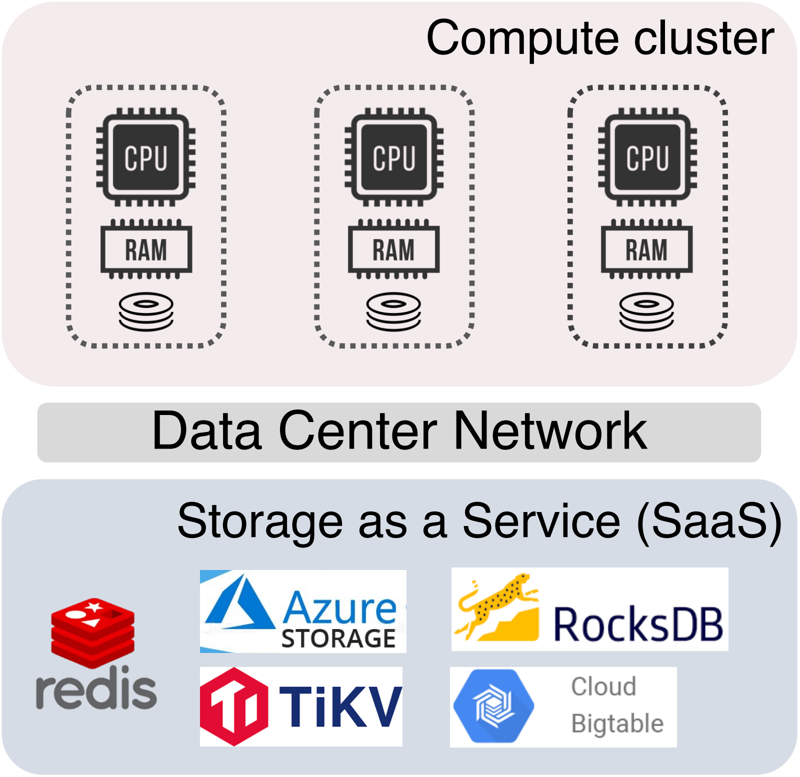
Advantage #2: Low Cost

- Storage service can be much cheaper than compute servers

S3 storage price	**$0.02** per GB per month
16 vCPU Virtual Machine	**$0.5** per hour per VM



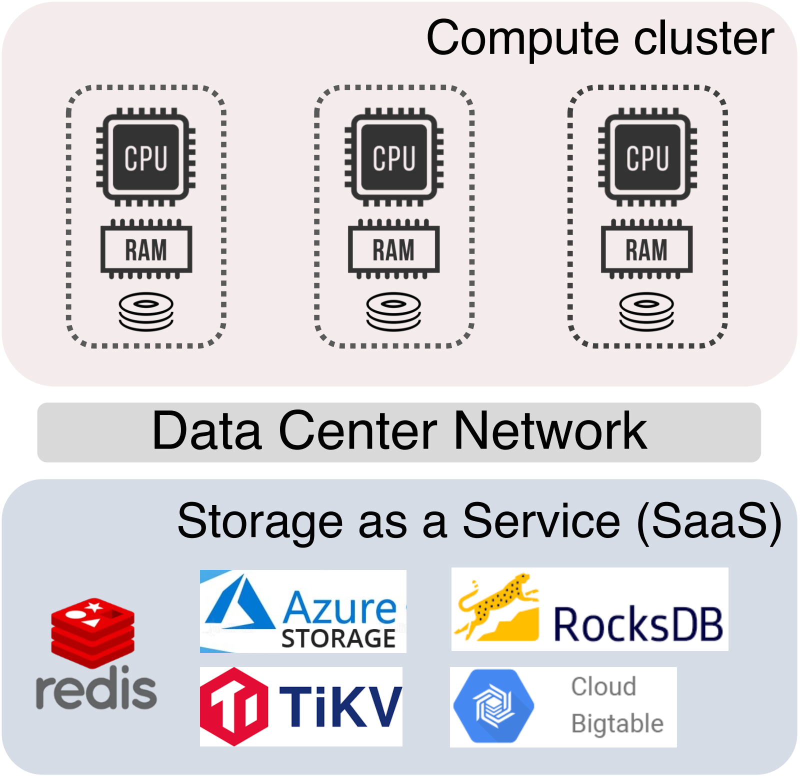
Advantages of Storage-Disaggregation



Advantage #3: Availability

- Storage service provides high availability through geo-replication
- Simplifies fault tolerance in DB

Advantages of Storage-Disaggregation



Advantage #3: Availability

- Storage service provides high availability through geo-replication
- Simplifies fault tolerance in DB

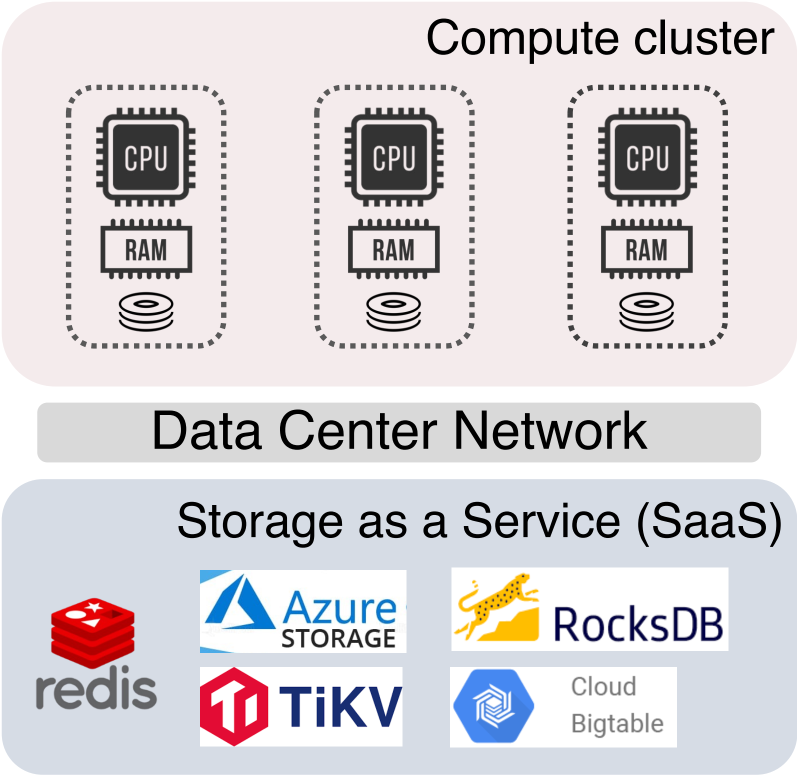
Storage-disaggregation architecture widely deployed in cloud databases

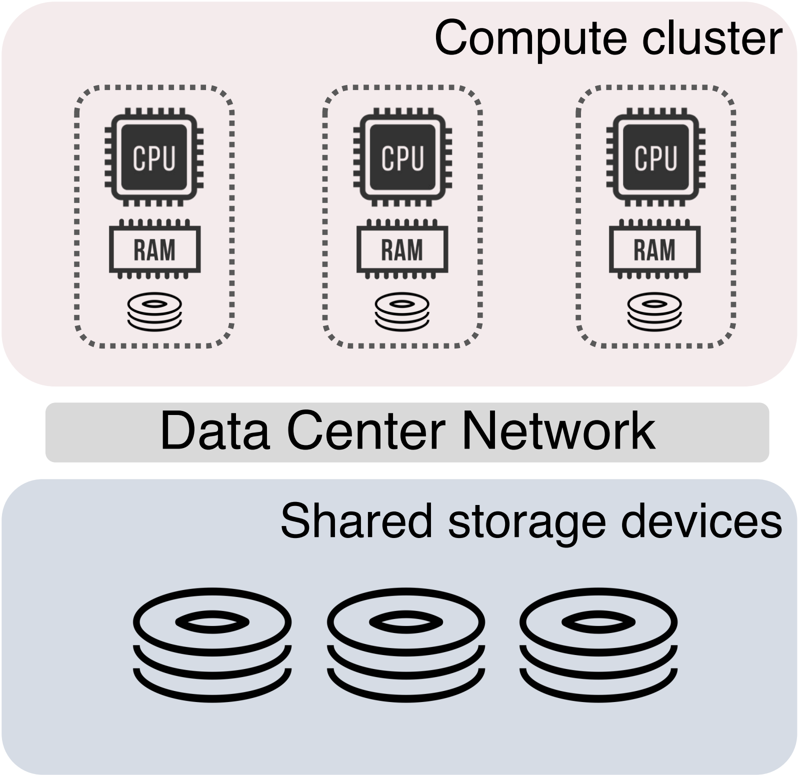
Amazon Aurora POLARDB Microsoft SQL Server GaussDB amazon REDSHIFT

Cloud Spanner TiDB yugabyteDB Cockroach DB Google BigQuery

Azure Cosmos DB Spark SQL snowflake HIVE presto

Storage-Disaggregation vs. Shared Disk

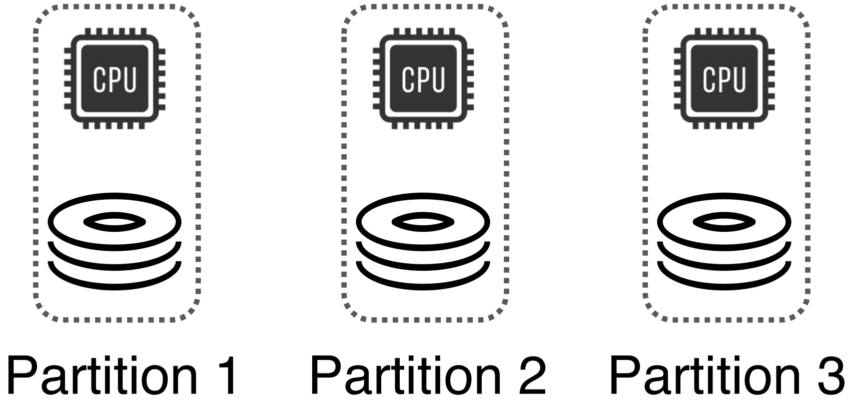




The storage service can **scale horizontally**, has **built-in high availability**, and has **richer APIs**

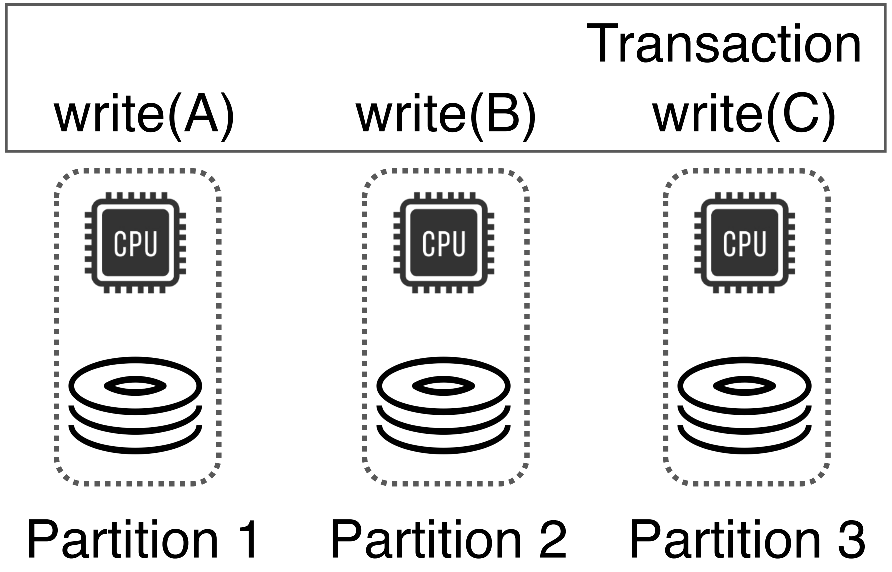
Distributed Atomic Commitment

Data partitioned across machines



Partition 1 Partition 2 Partition 3

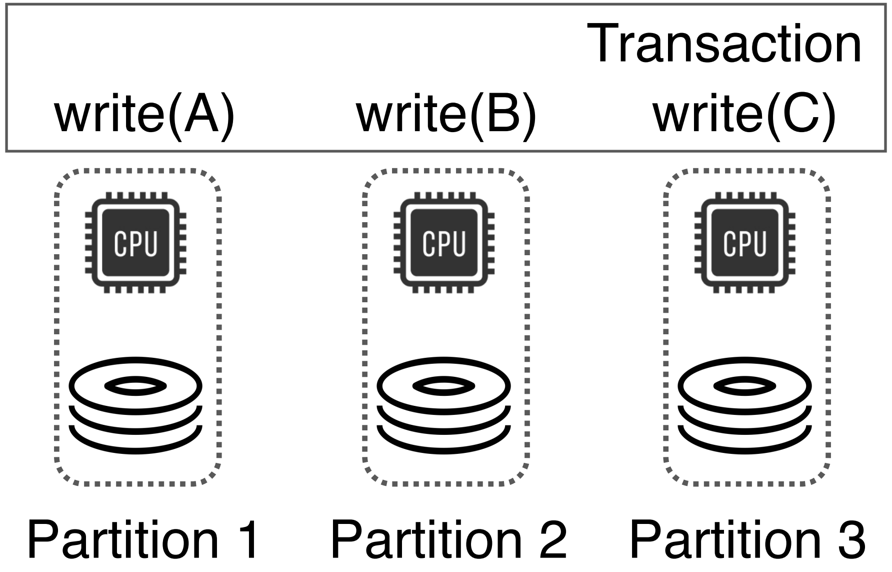
Distributed Atomic Commitment



Data partitioned across machines

A transaction updates data across multiple partitions

Distributed Atomic Commitment

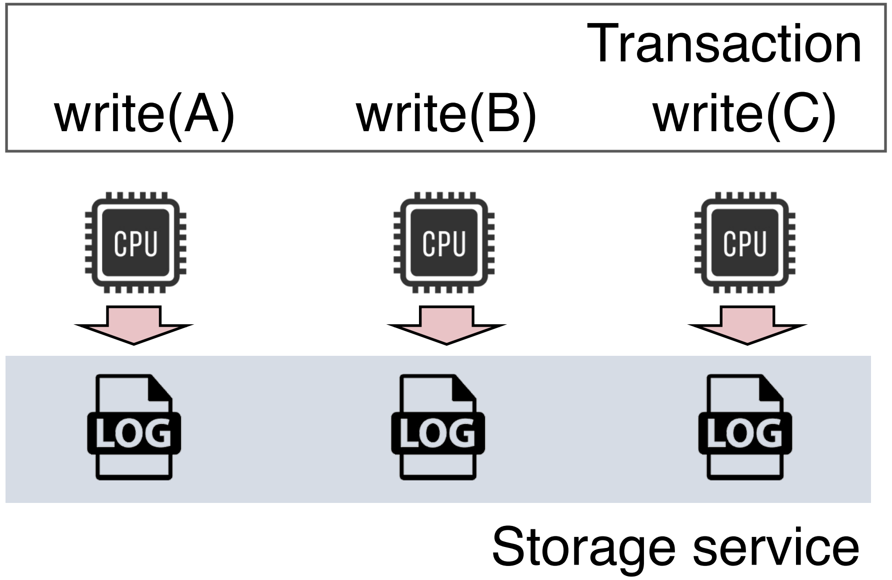


Data partitioned across machines

A transaction updates data across multiple partitions

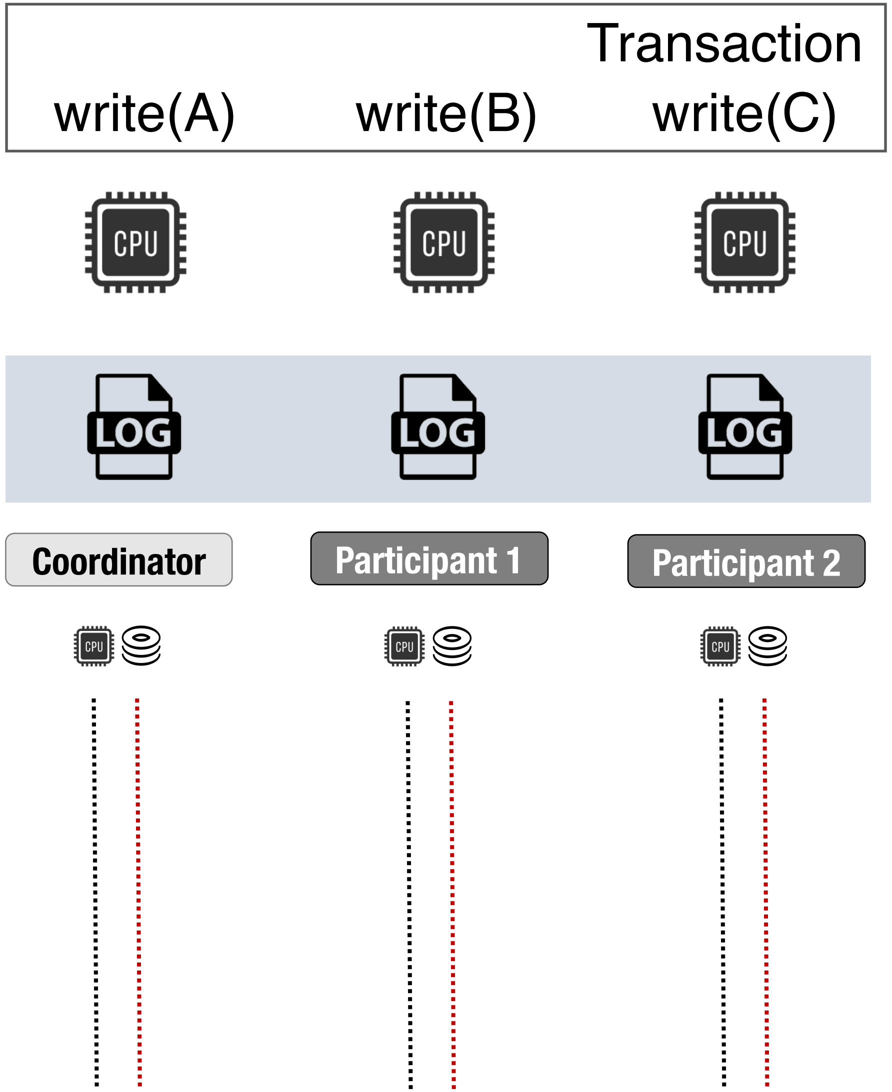
Atomic commitment requires the transaction to commit in **all or none** of the involved partitions

Distributed Atomic Commitment



With storage disaggregation, log files locate in the storage service

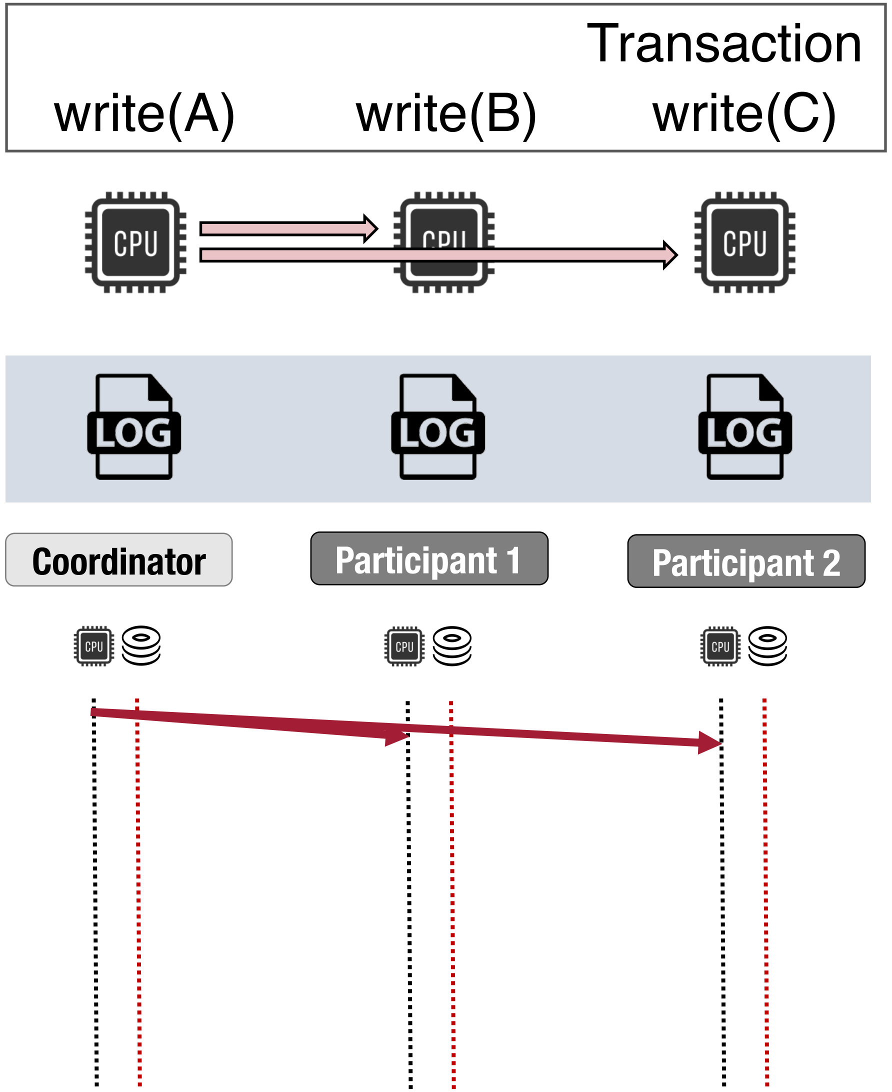
Two-Phase Commit (2PC)



Coordinator initiates the 2PC protocol

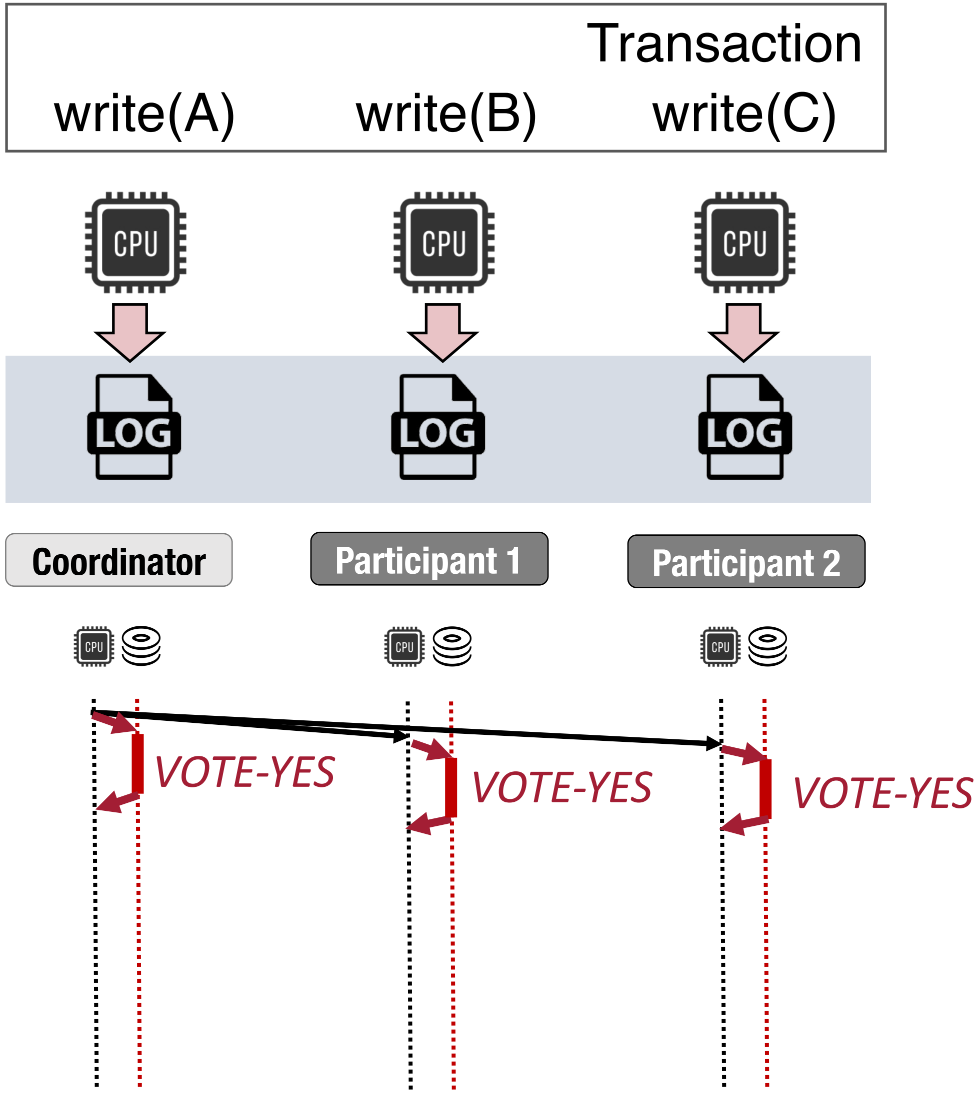
The example assumes a committing transaction

Two-Phase Commit (2PC)



Coordinator initiates the 2PC protocol

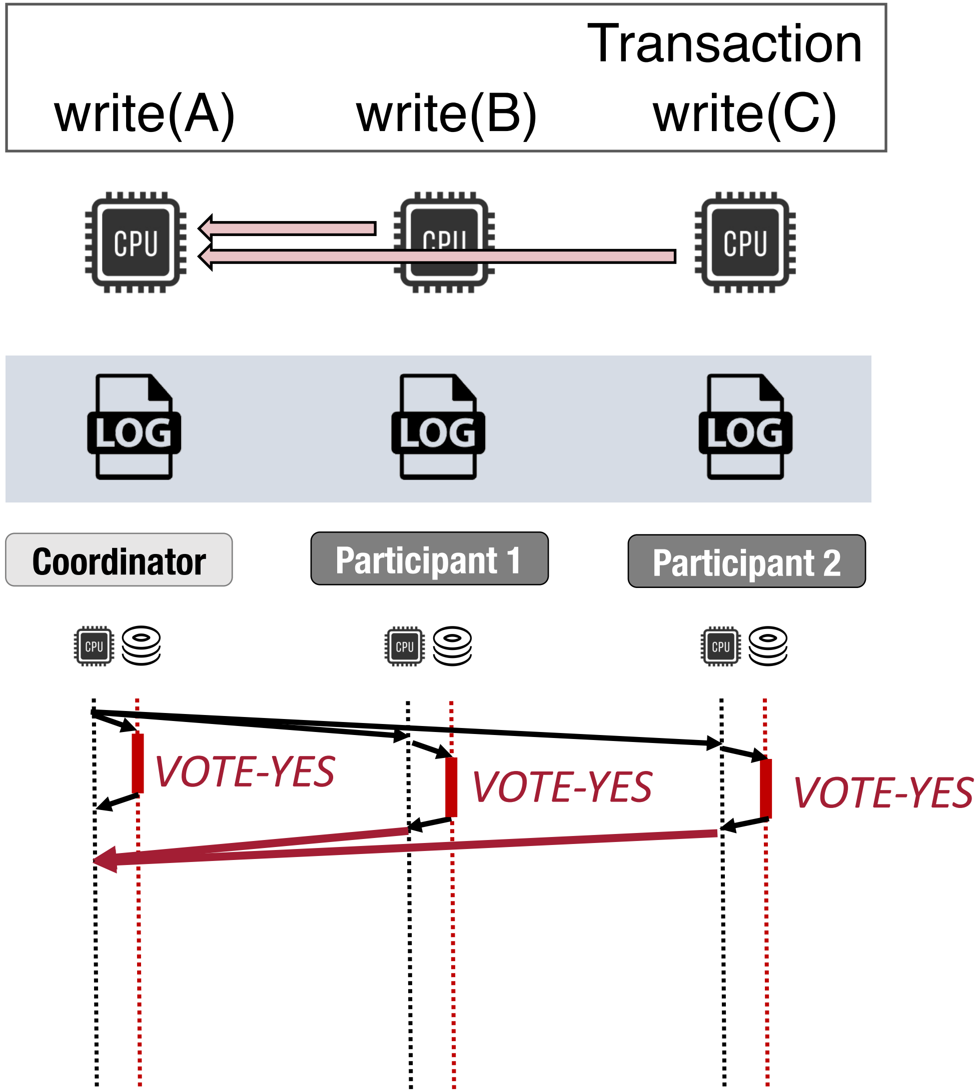
Two-Phase Commit (2PC)



Each participant appends *VOTE-YES* to local log file

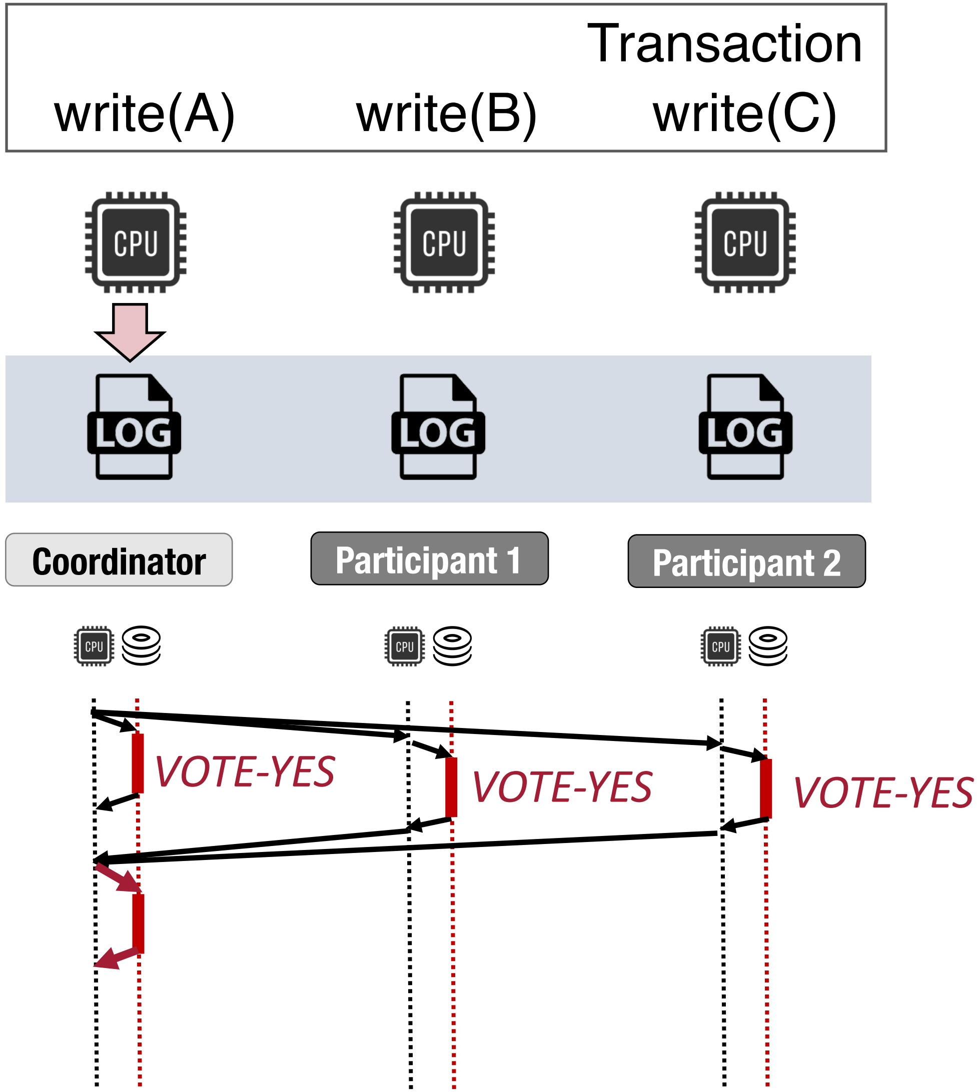
- Promise not to unilaterally abort

Two-Phase Commit (2PC)



Participants reply votes to coordinator

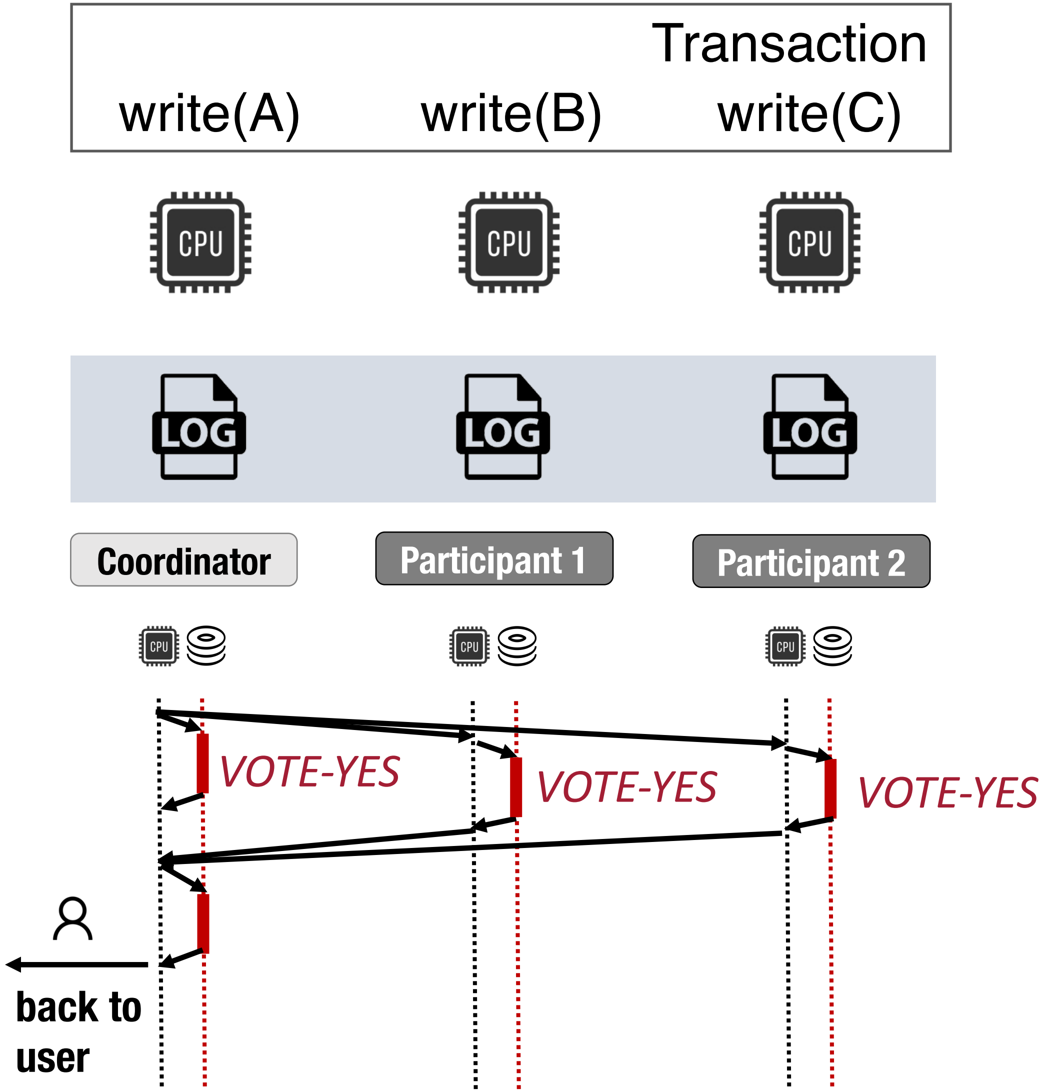
Two-Phase Commit (2PC)



Coordinator logs the final decision (e.g., *COMMIT* or *ABORT*)

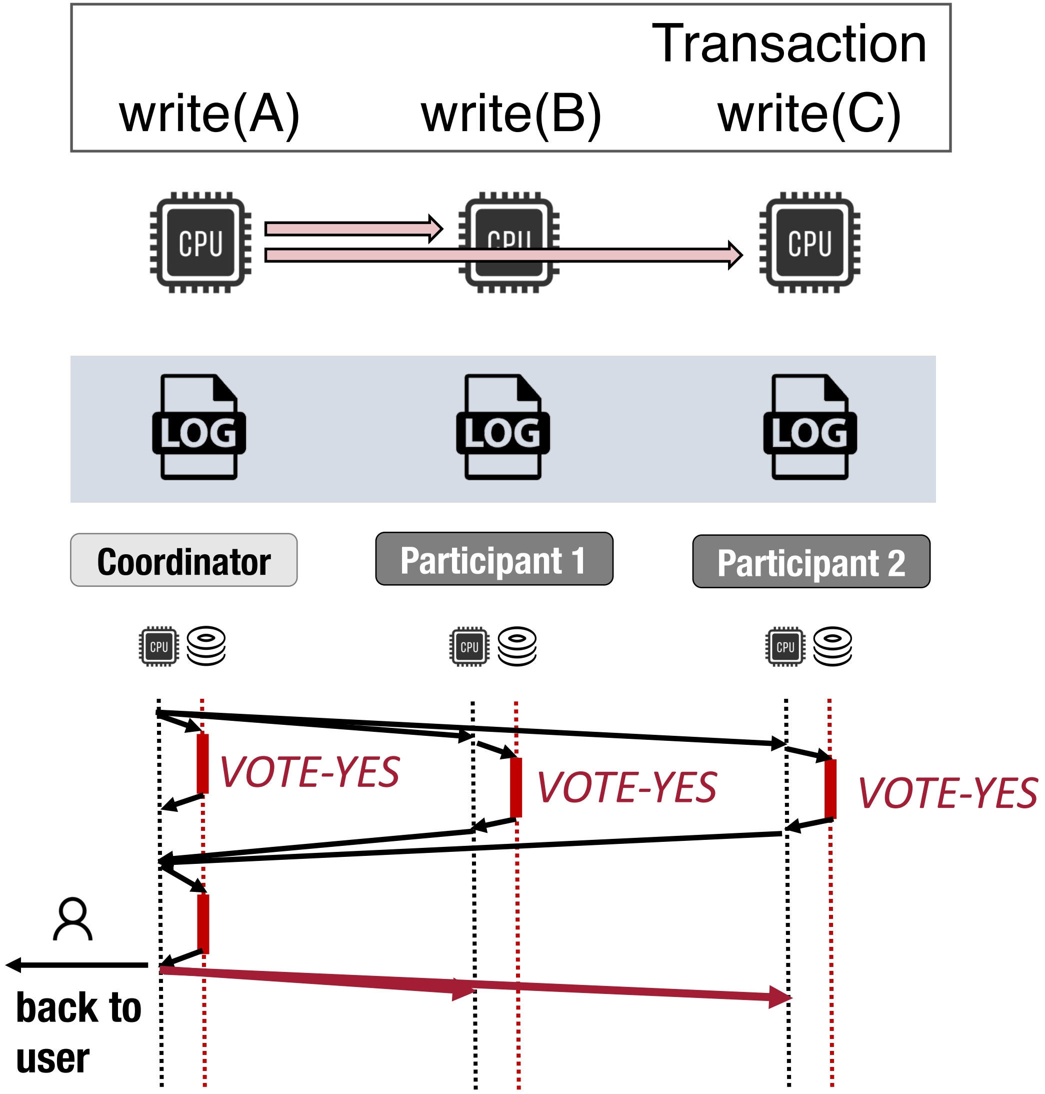
The decision log record is the ground truth of the transaction outcome

Two-Phase Commit (2PC)



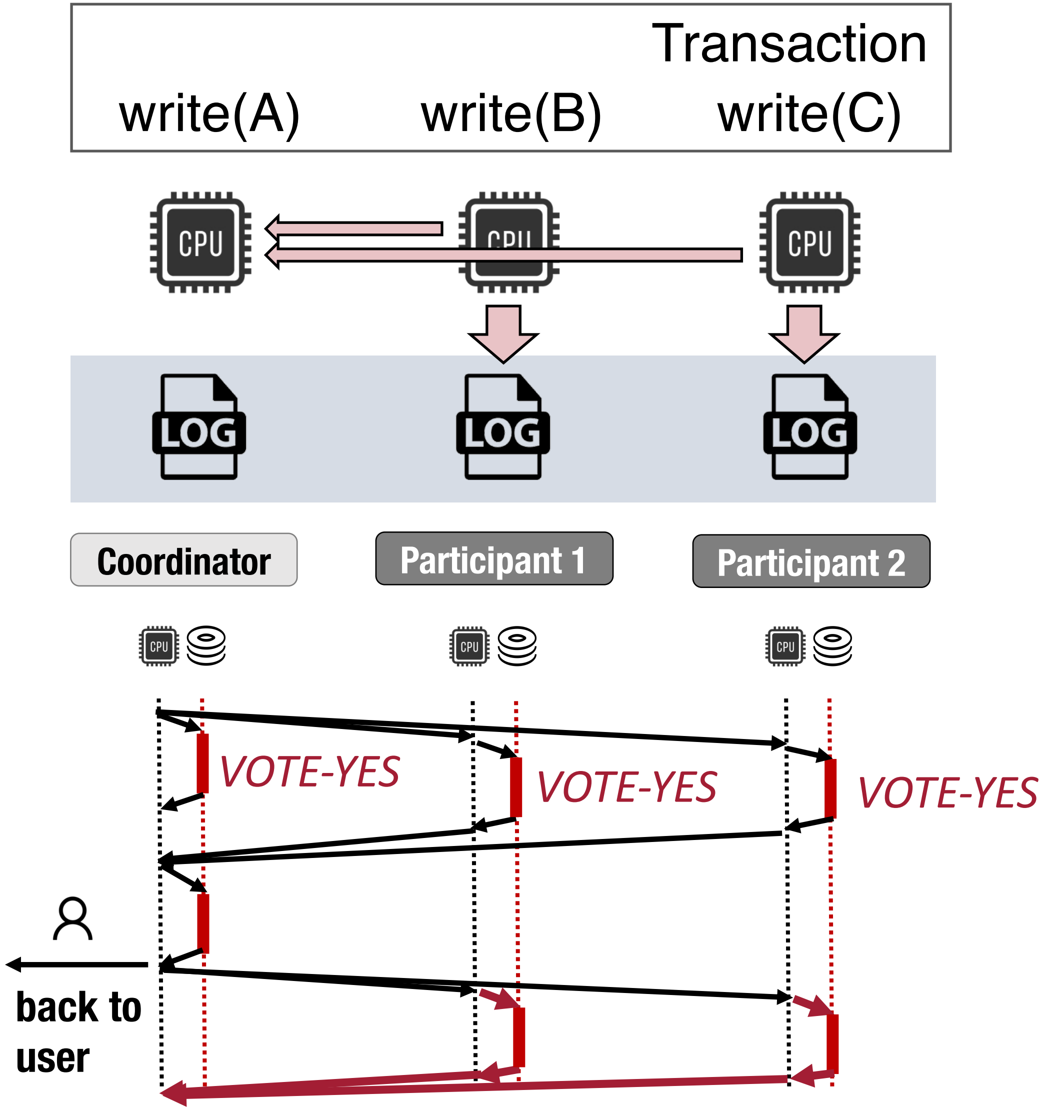
Reply to user after writing the decision log record

Two-Phase Commit (2PC)



Coordinator sends the final decision to all participants

Two-Phase Commit (2PC)

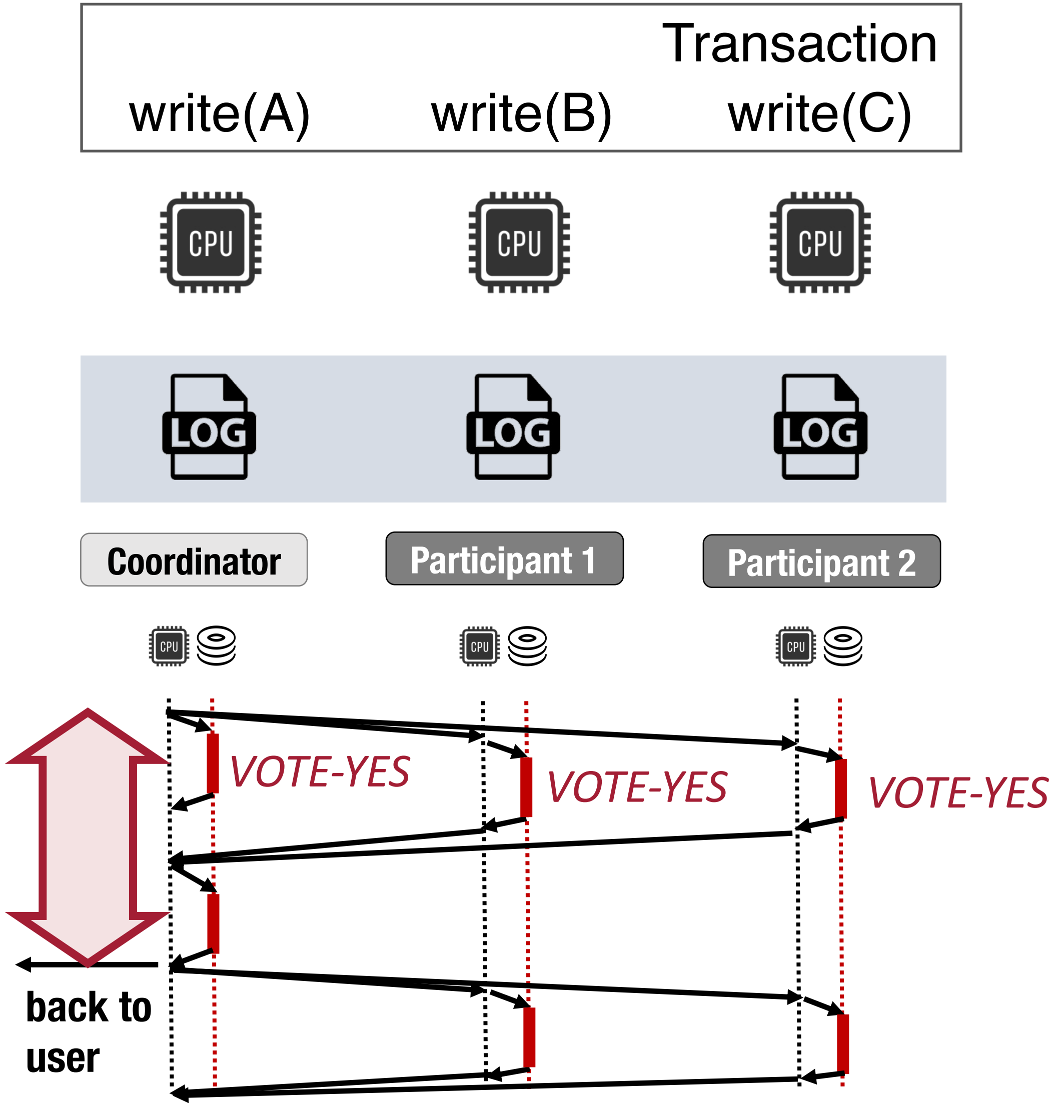


Coordinator sends the final decision to all participants

Participants log the decision

- For independent recovery upon failure

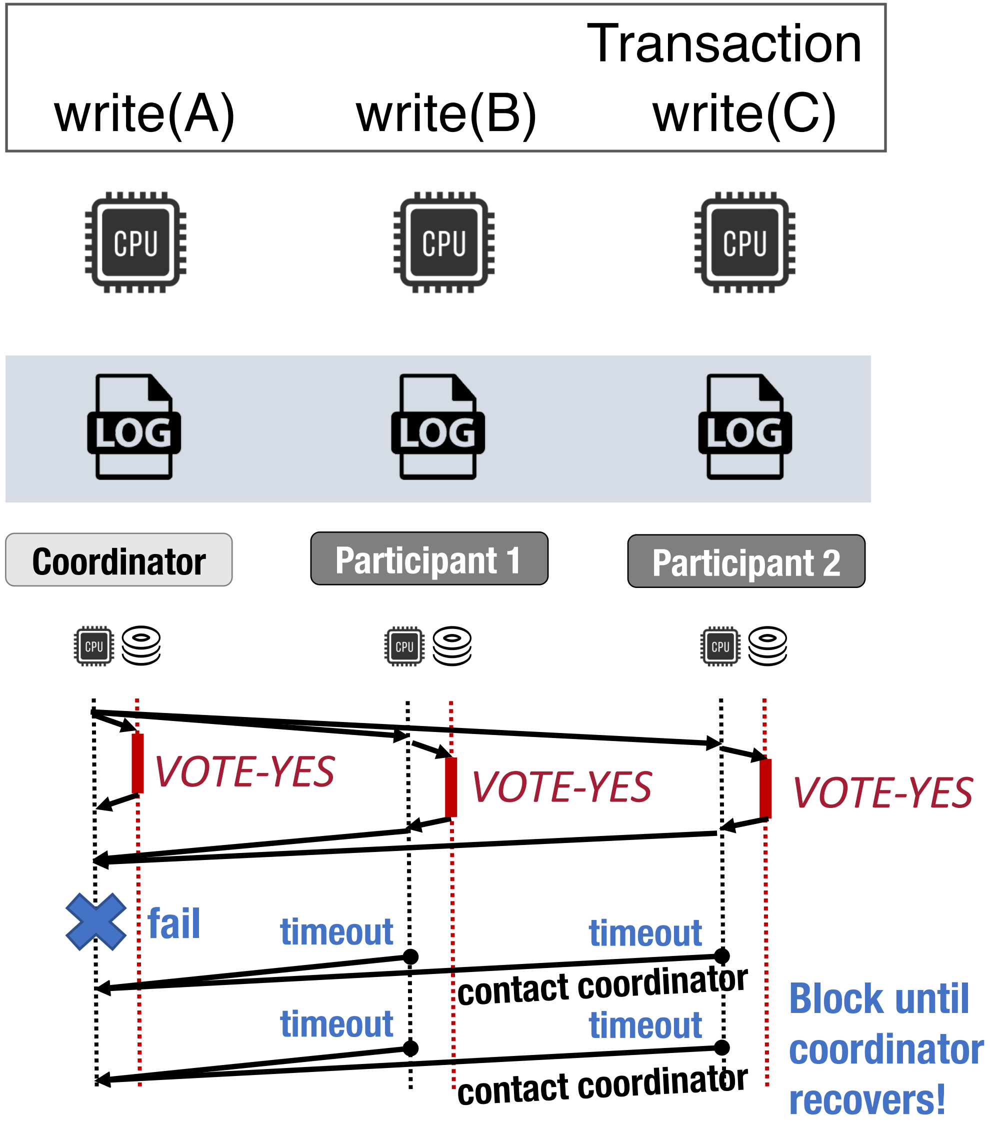
Limitations of 2PC



Limitation #1: **Long latency**

- User experiences latency of two logging operations

Limitations of 2PC



Limitation #1: **Long latency**

- User experiences latency of two logging operations

Limitation #2: **Blocking problem**

- Participants are blocked if the coordinator fails

2PC Limitations – Prior Solutions

Solutions	Example systems	Limitations in prior solutions
Reduce latency	Coordinator log [1] Implicit yes vote [2] Early prepare [3]	• Extra system or workload assumptions • Violate site autonomy

[1] James W Stamos and Flaviu Cristian. *Coordinator log transaction execution protocol*. Distributed and Parallel Databases 1993
[2] Y Al-Houmaily and P Chrysanthis. *Two-phase commit in gigabit-networked distributed databases.* PDCS, 1995
[3] James W Stamos and Flaviu Cristian. *A low-cost atomic commit protocol*. Symposium on Reliable Distributed Systems, 1990

2PC Limitations – Prior Solutions

Solutions	**Example systems**	**Limitations in prior solutions**
Reduce latency	Coordinator log [1] Implicit yes vote [2] Early prepare [3]	• Extra system or workload assumptions • Violate site autonomy
Non-blocking	Three-phase commit (3PC) [4]	• Requires extra latency and/or network messages

[1] James W Stamos and Flaviu Cristian. *Coordinator log transaction execution protocol*. Distributed and Parallel Databases 1993
[2] Y Al-Houmaily and P Chrysanthis. *Two-phase commit in gigabit-networked distributed databases.* PDCS, 1995
[3] James W Stamos and Flaviu Cristian. *A low-cost atomic commit protocol*. Symposium on Reliable Distributed Systems, 1990
[4] Dale Skeen. *Nonblocking commit protocols*. SIGMOD 1981

2PC Limitations – Prior Solutions

Solutions	Example systems	Limitations in prior solutions
Reduce latency	Coordinator log [1] Implicit yes vote [2] Early prepare [3]	• Extra system or workload assumptions • Violate site autonomy
Non-blocking	Three-phase commit (3PC) [4]	• Requires extra latency and/or network messages
Codesign 2PC with replication	Paxos commit [5] MDCC [6] Parallel commit [7] TAPIR [8]	• Extra design complexity • Custom-designed consensus protocol

[1] James W Stamos and Flaviu Cristian. *Coordinator log transaction execution protocol*. Distributed and Parallel Databases 1993
[2] Y Al-Houmaily and P Chrysanthis. *Two-phase commit in gigabit-networked distributed databases.* PDCS, 1995
[3] James W Stamos and Flaviu Cristian. *A low-cost atomic commit protocol*. Symposium on Reliable Distributed Systems, 1990
[4] Dale Skeen. *Nonblocking commit protocols*. SIGMOD 1981
[5] Jim Gray and Leslie Lamport. *Consensus on Transaction Commit*. ACM Trans. Database Syst, 2006
[6] TimKraska, et al. *MDCC: Multi-data center consistency*. European Conference on Computer Systems, 2013
[7] Rebecca Taft, et al. *Cockroachdb: The resilient geo-distributed SQL database*. SIGMOD 2020
[8] Irene Zhang, et al. *Building consistent transactions with inconsistent replication*. TOCS 2018

2PC Limitations – Prior Solutions

Solutions	**Example systems**	**Limitations in prior solutions**
Reduce latency	Coordinator log [1] Implicit yes vote [2] Early prepare [3]	• Extra system or workload assumptions • Violate site autonomy
Non-blocking	Three-phase commit (3PC) [4]	• Requires extra latency and/or network messages
Codesign 2PC with replication	Paxos commit [5] MDCC [6] Parallel commit [7] Tapir [8]	• Extra design complexity • Custom-designed consensus protocol

Research Question: What is the minimal requirement from the storage service to enable 2PC optimizations addressing high latency and blocking?

Cornus Overview

An optimized two-phase commit protocol for a cloud database with storage disaggregation

Cornus Overview

An optimized two-phase commit protocol for a cloud database with storage disaggregation

2PC Limitation 1: **Long latency**

⇒Cornus **reduces 2 logging events to 1 logging event**

2PC Limitation #2: **Blocking problem**

⇒Cornus is **non-blocking**

Cornus Overview

An optimized two-phase commit protocol for a cloud database with storage disaggregation

2PC Limitation 1: **Long latency**

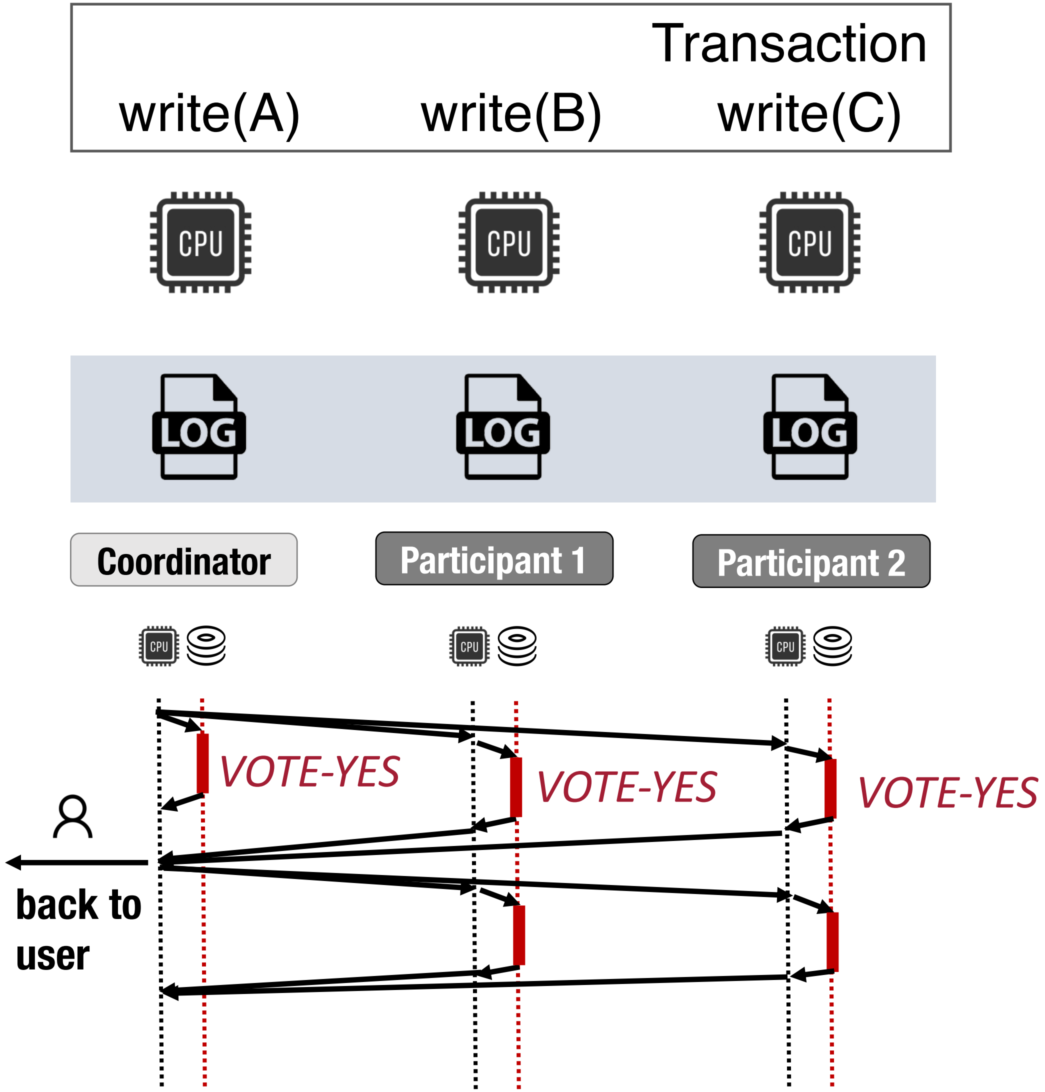
⇒Cornus **reduces 2 logging events to 1 logging events**

2PC Limitation #2: **Blocking problem**

⇒Cornus is **non-blocking**

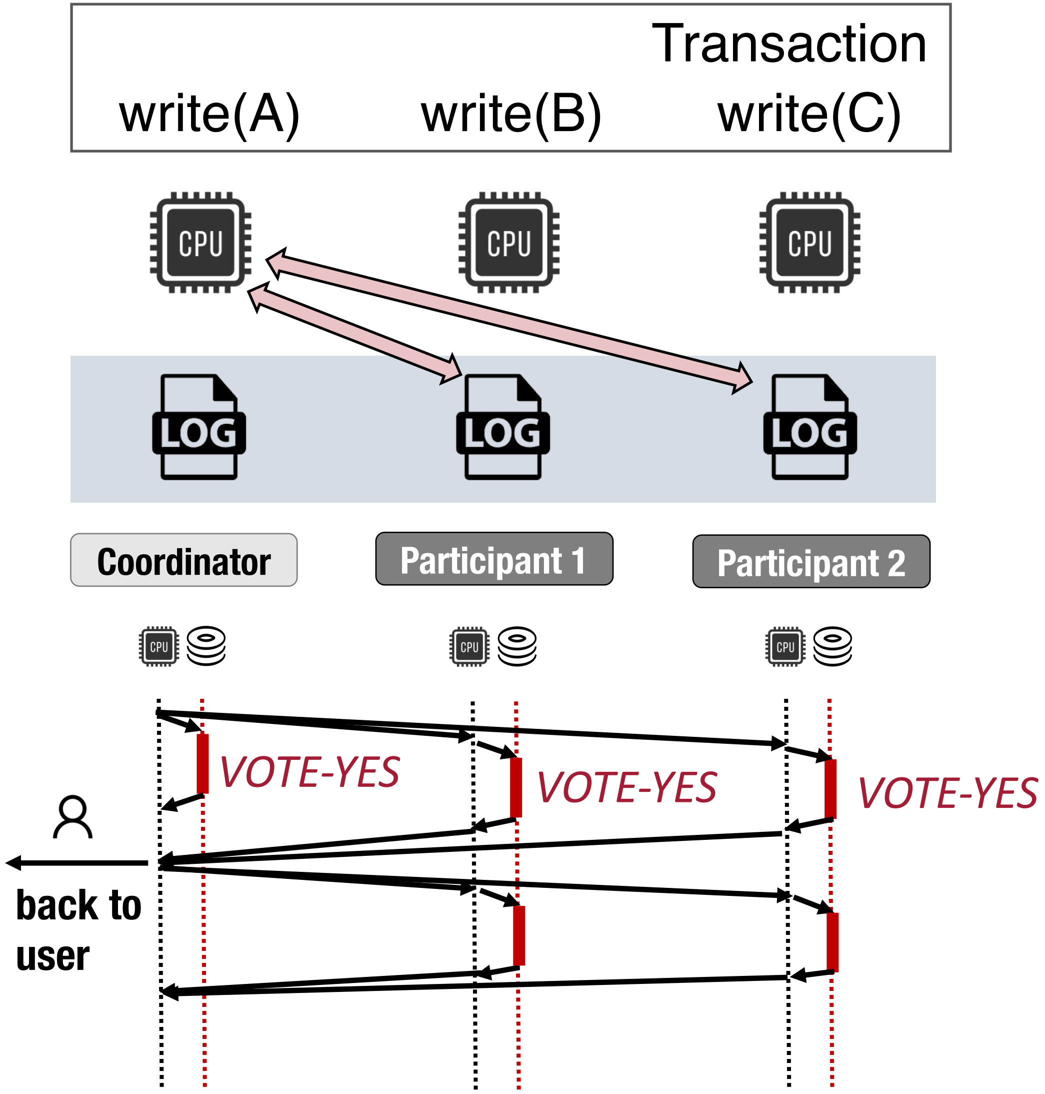
Only new storage-layer function is ***LogOnce()*** *which* can be implemented using compare-and-swap

Cornus Key Ideas



Key idea #1: **Remove decision logging**

Cornus Key Ideas

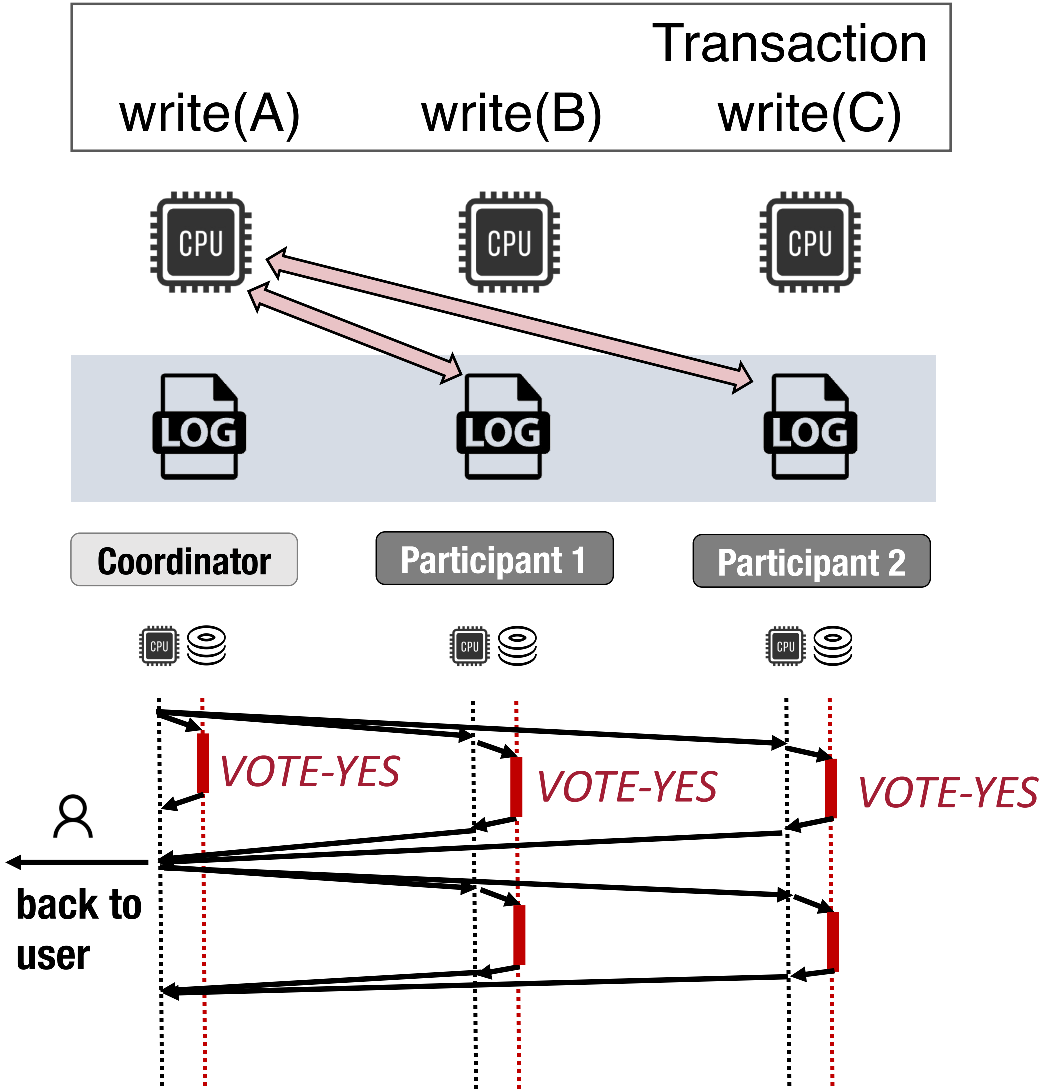


Key idea #1: **Remove decision logging**

Ground truth: collective votes in all participants logs

- Uncertain node can directly read all votes

Cornus Key Ideas



Key idea #1: **Remove decision logging**

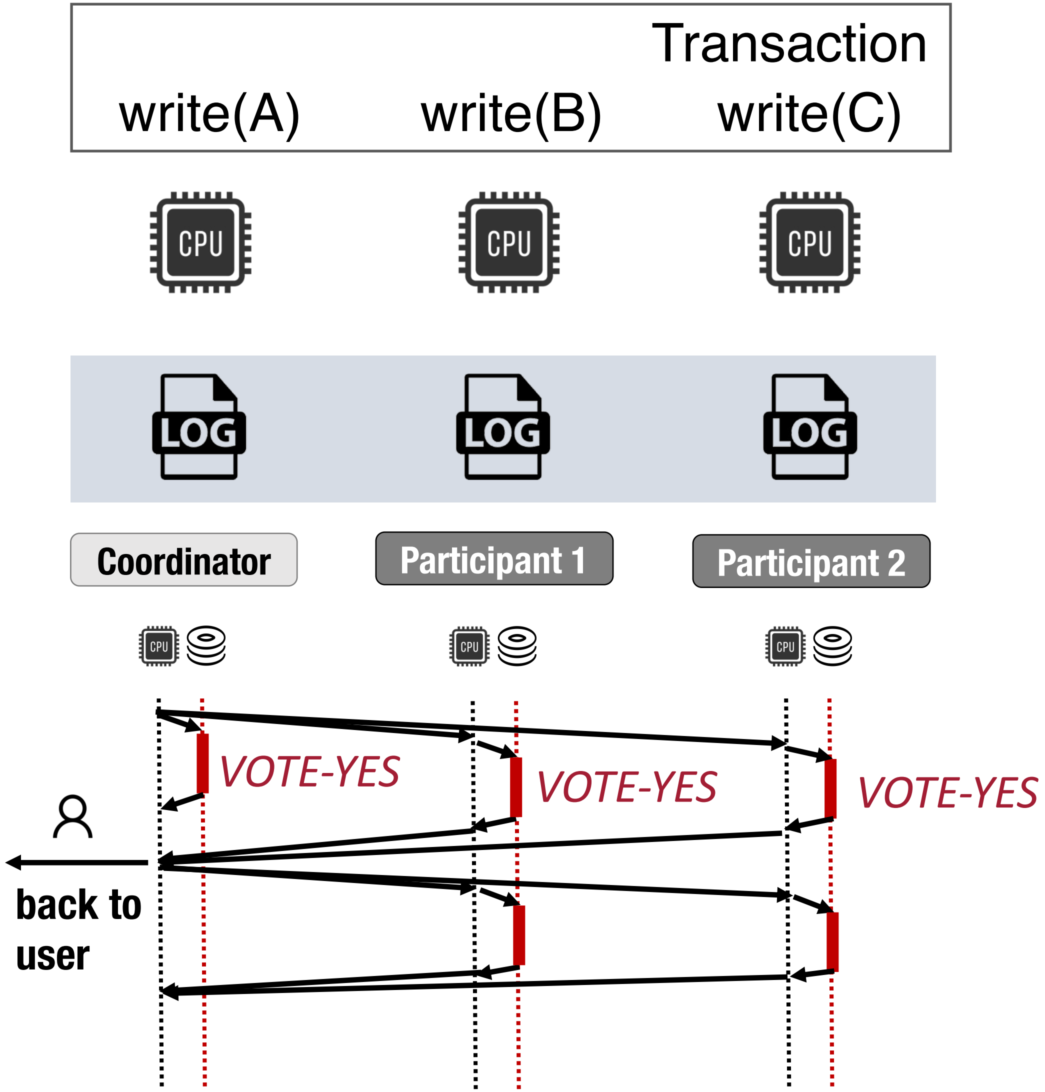
Ground truth: collective votes in all participants logs

- Uncertain node can directly read all votes

Enabled by storage disaggregation through

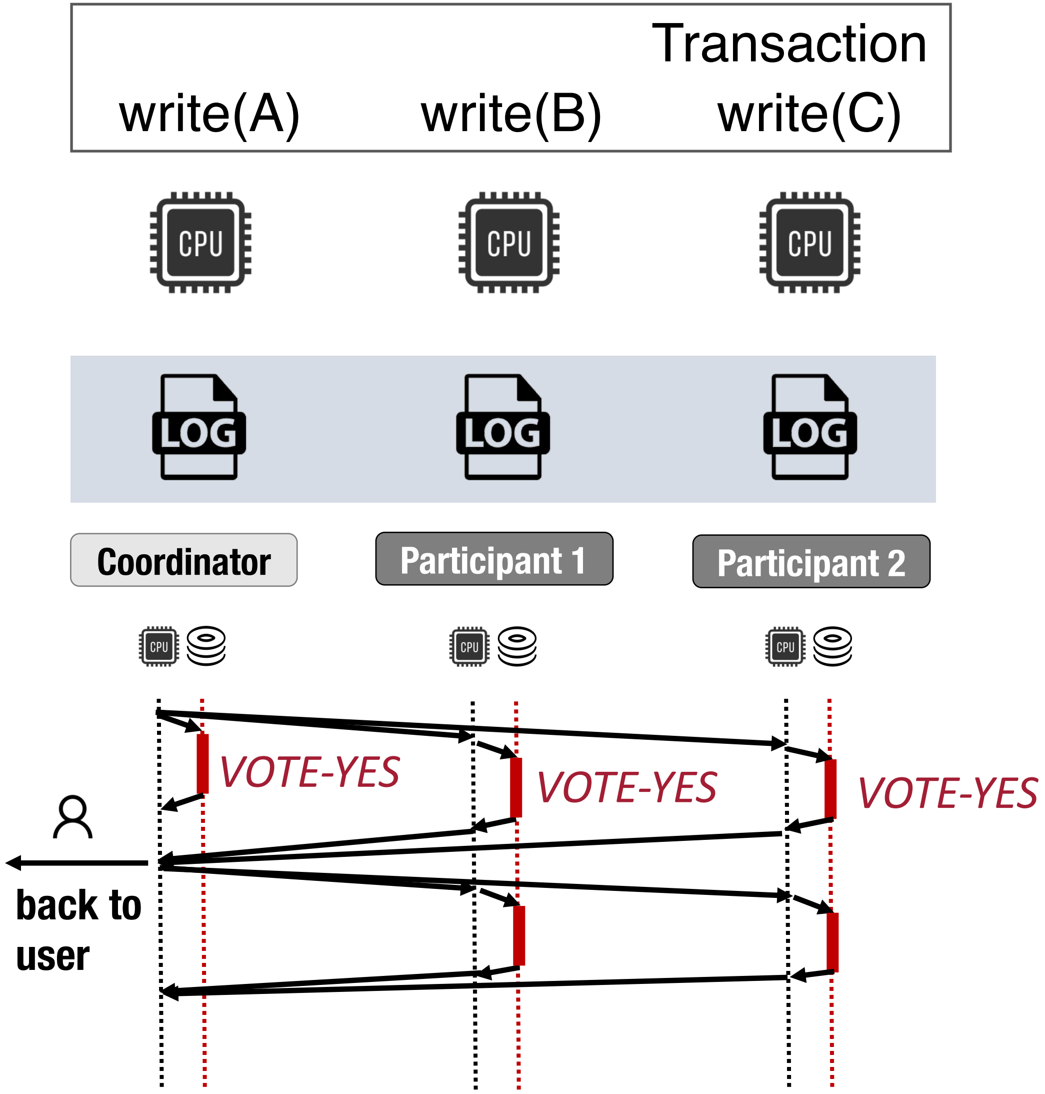
- Highly available storage service
- Shared across compute nodes

Cornus Key Ideas



Key idea #2: **LogOnce() storage API**

Cornus Key Ideas

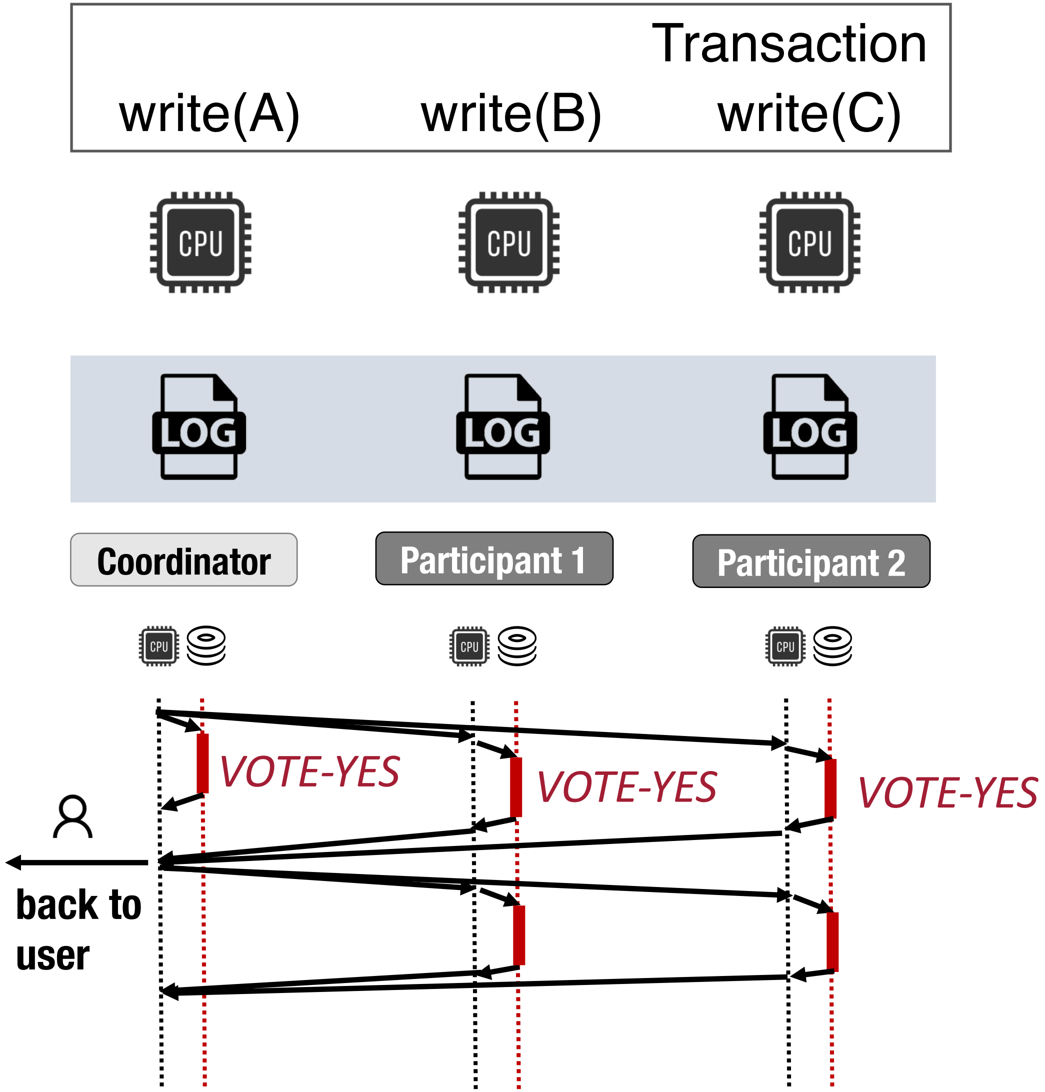


Key idea #2: **LogOnce() storage API**

Avoid blocking by directly updating log files of unresponsive nodes

- Only first LogOnce() request can succeed

Cornus Key Ideas



Key idea #2: **LogOnce() storage API**

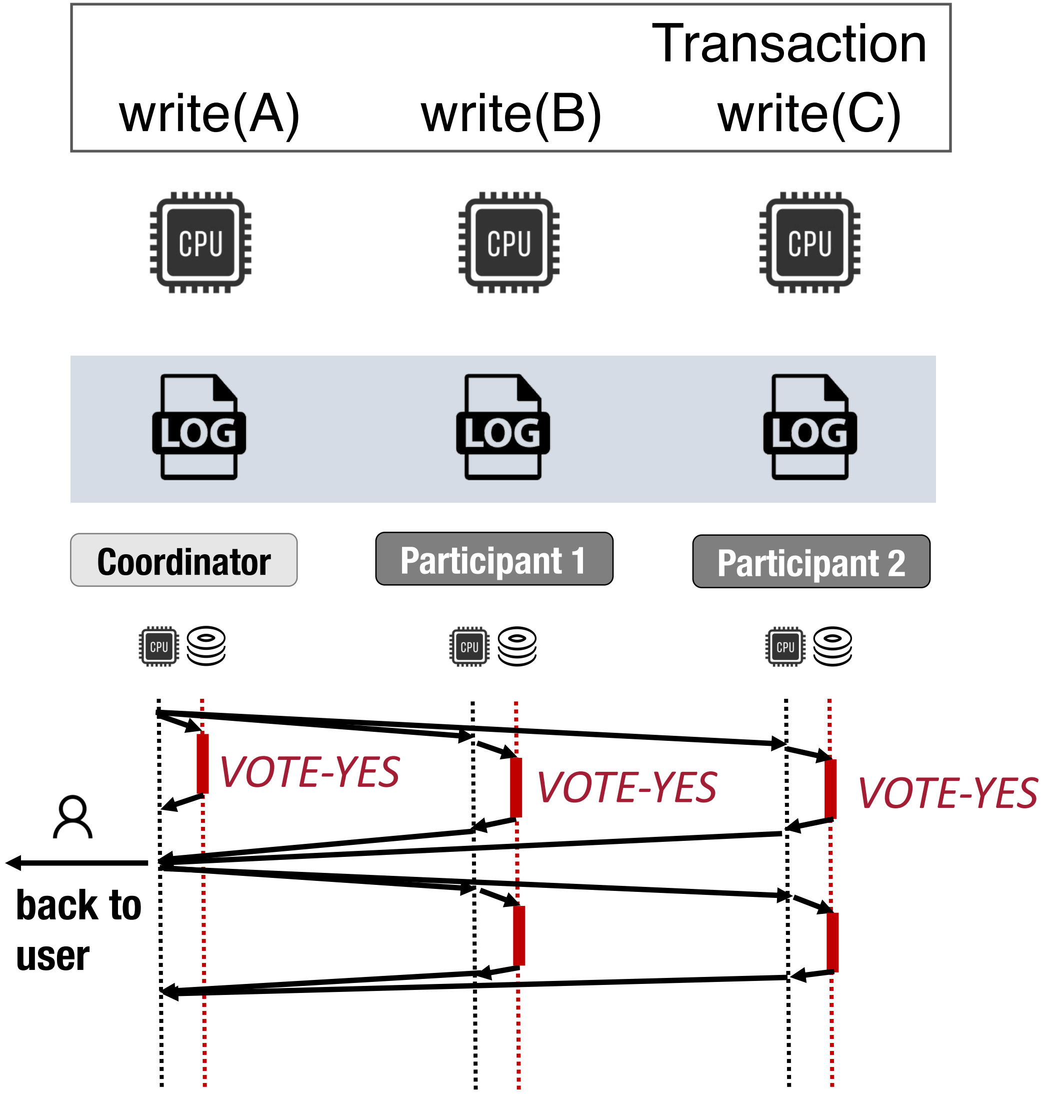
Avoid blocking by directly updating log files of unresponsive nodes

- Only first LogOnce() request can succeed

LogOnce() can be implemented using CAS-like APIs (e.g., Etags)

redis Azure STORAGE Cloud Bigtable amazon DynamoDB

Cornus Key Ideas



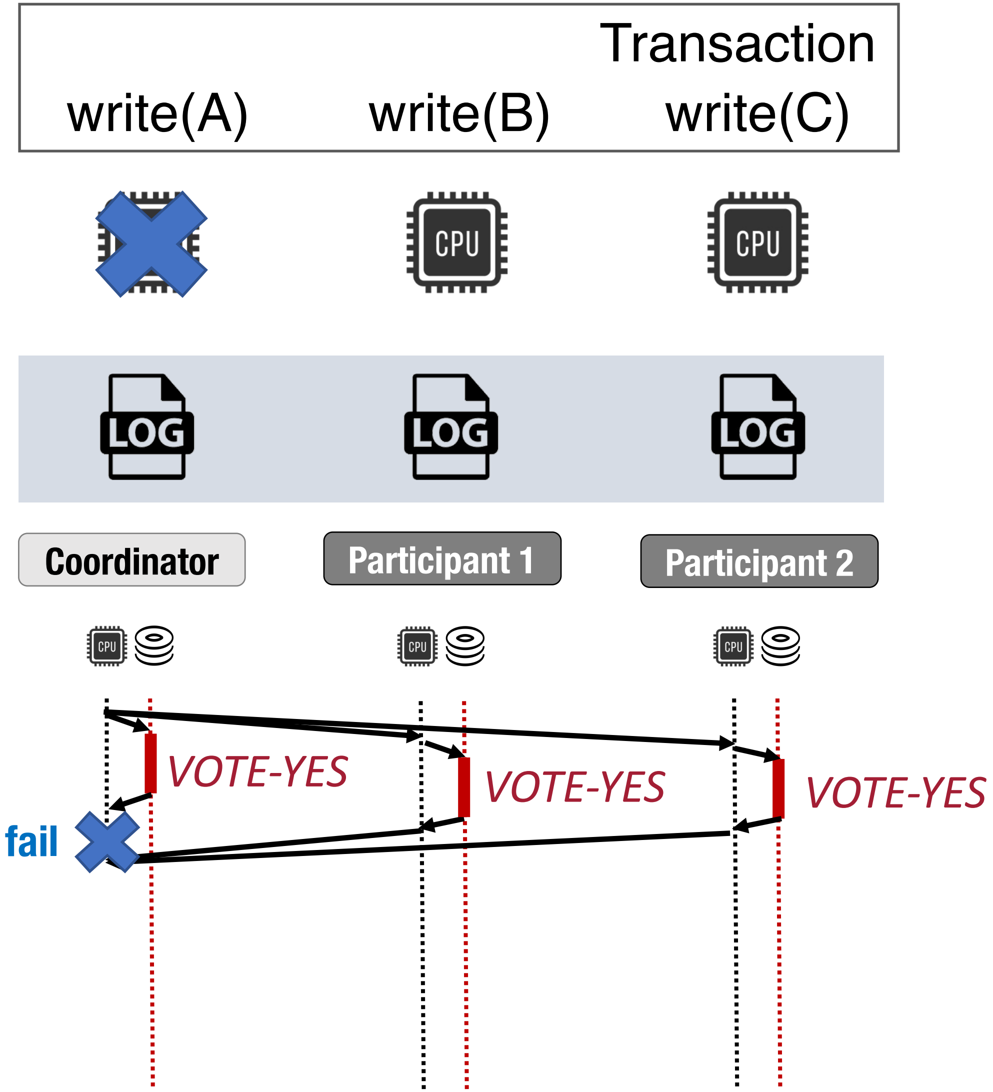
Key idea #2: **LogOnce() storage API**

Enabled by storage disaggregation through

– Rich APIs of storage service

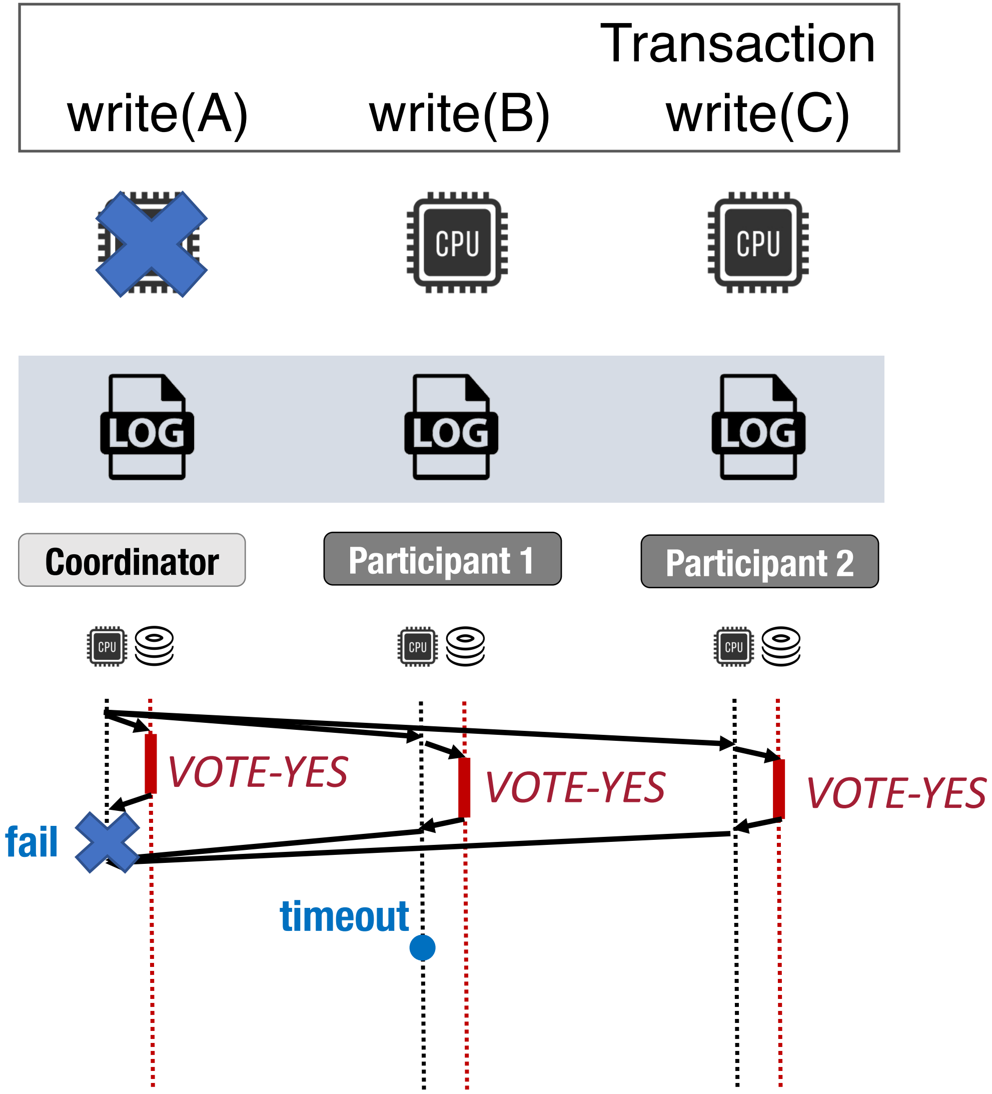
redis Azure STORAGE Cloud Bigtable amazon DynamoDB

Cornus Failure Example



Coordinator fails

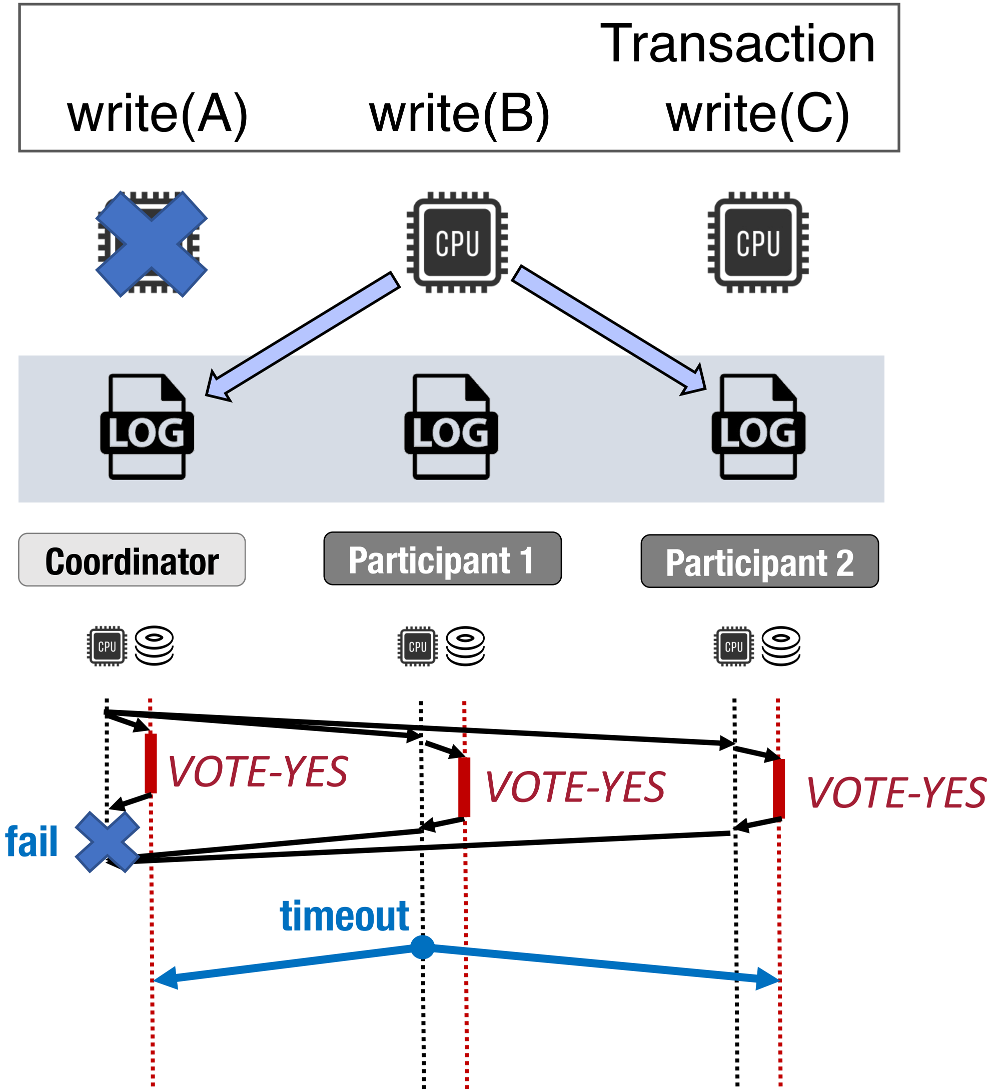
Cornus Failure Example



Coordinator fails

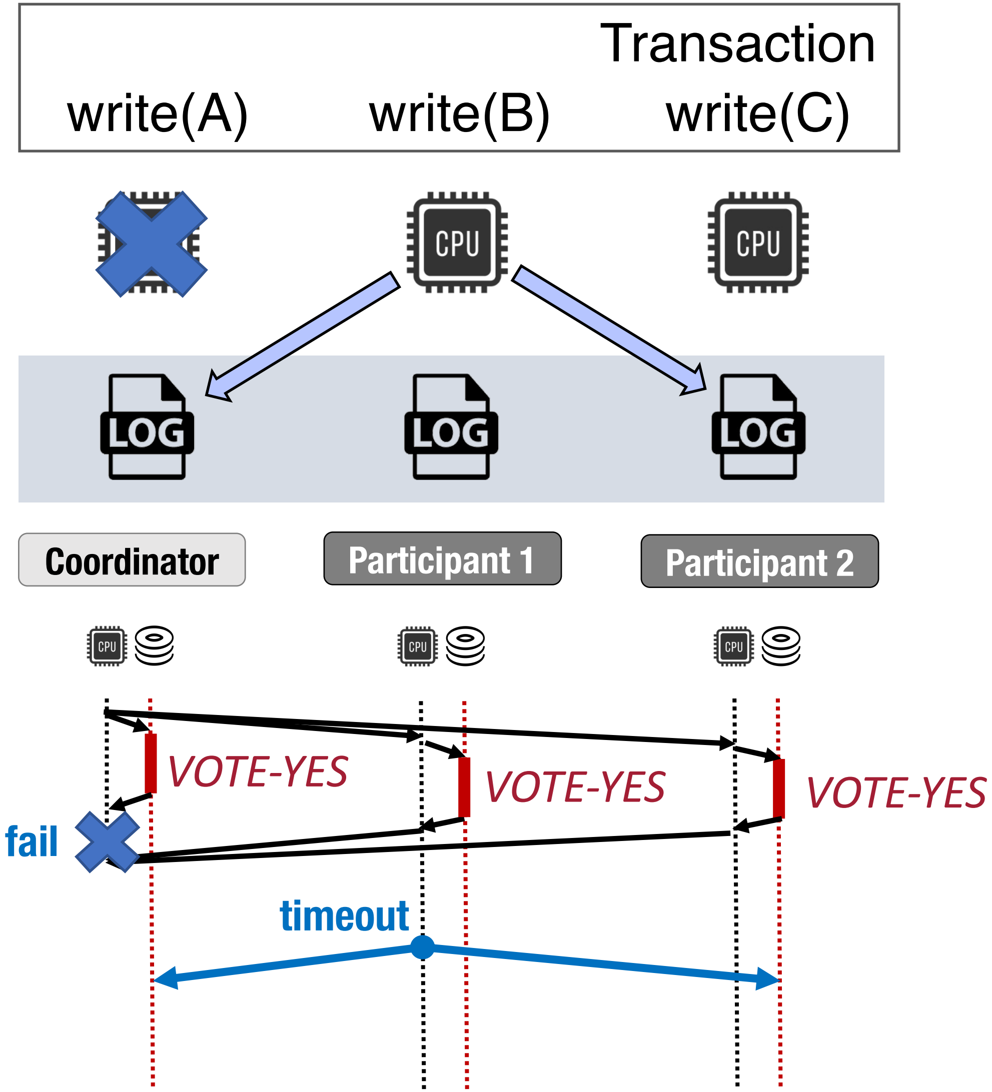
Timeout in participant 1 waiting for coordinator’s message

Cornus Failure Example



Use LogOnce() to write ABORT to other nodes' log files

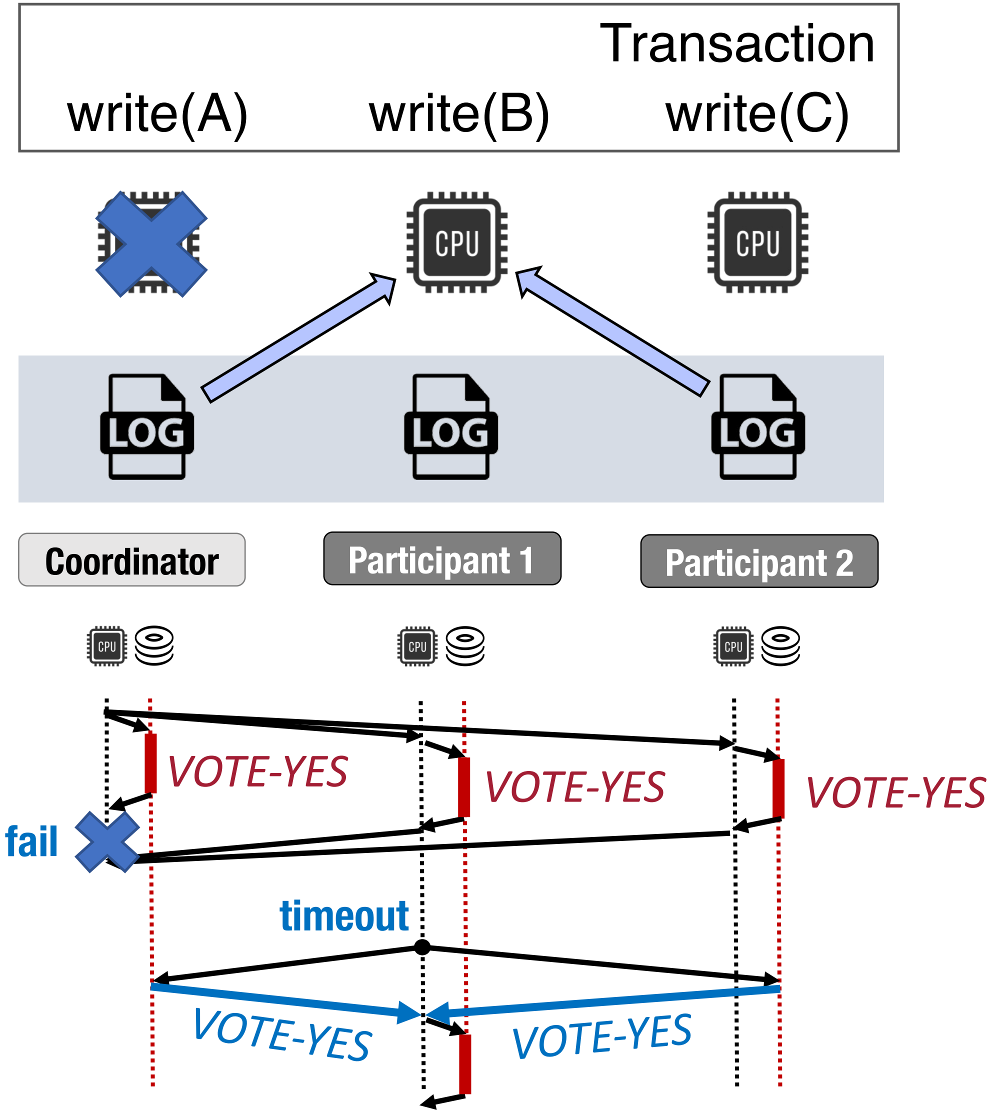
Cornus Failure Example



Use LogOnce() to write ABORT to other nodes’ log files

VOTE-YES already exists, LogOnce() does not modify log content

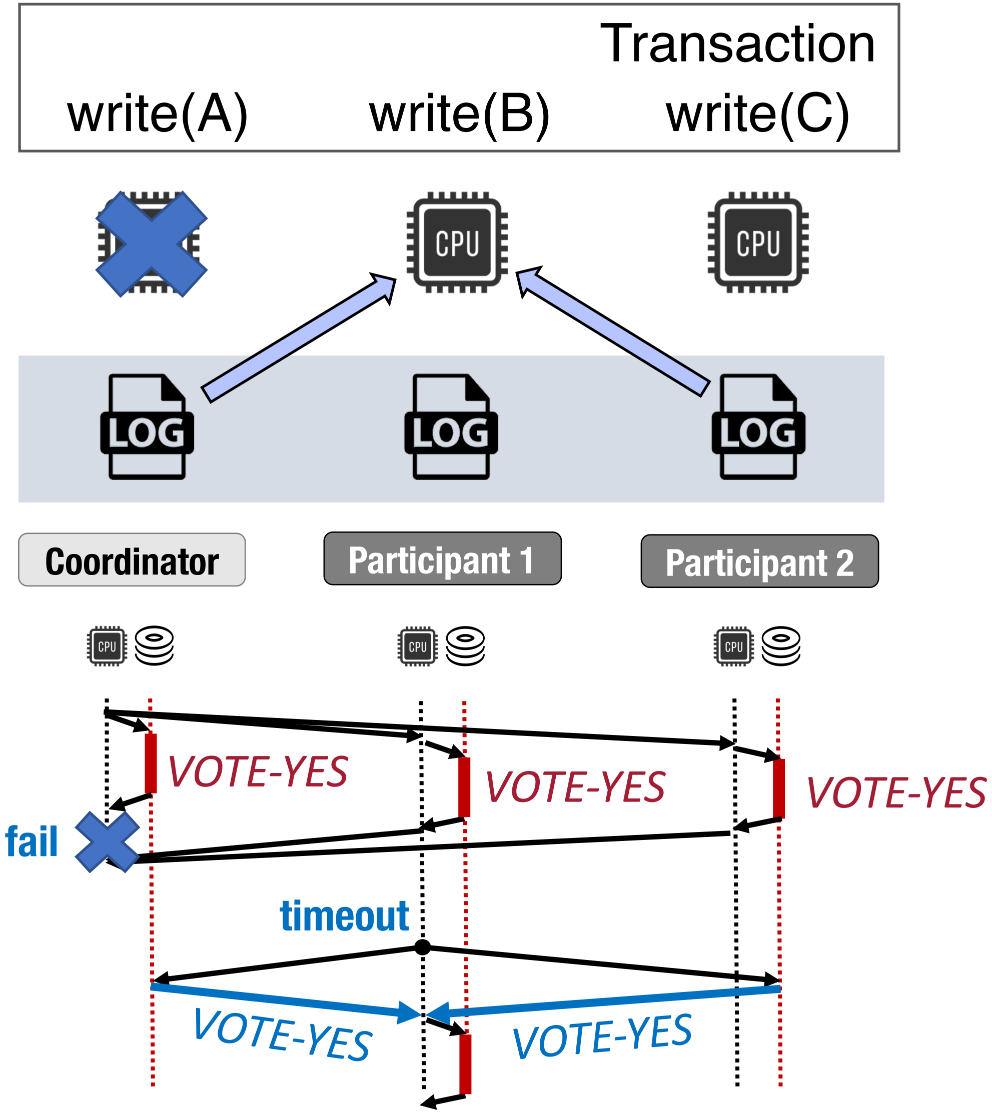
Cornus Failure Example



Storage service returns *VOTE-YES* without updating the logs

Participant 1 logs the *COMMIT* decision

Cornus Failure Example



Storage service returns *VOTE-YES* without updating the logs

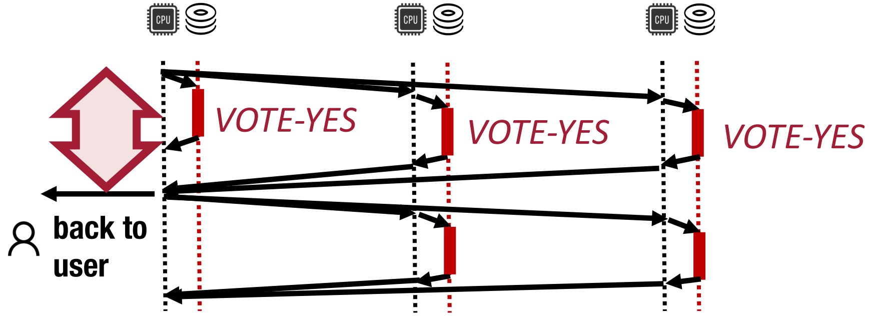
Participant 1 logs the *COMMIT* decision

Same process can happen for other participants (e.g., Participant 2)

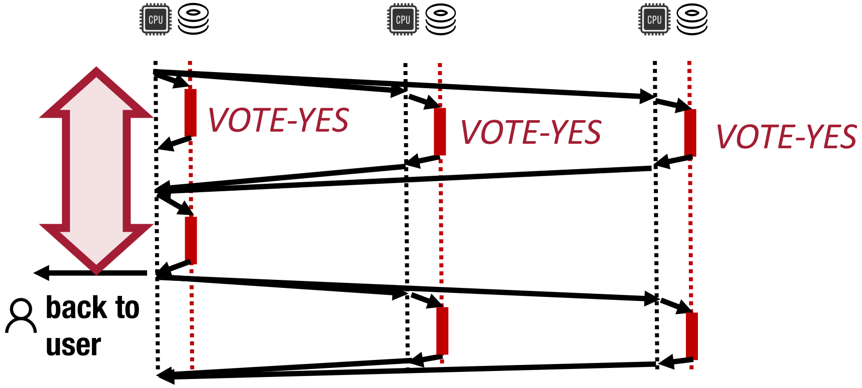
Cornus vs. 2PC Summary

Commit Case

Cornus



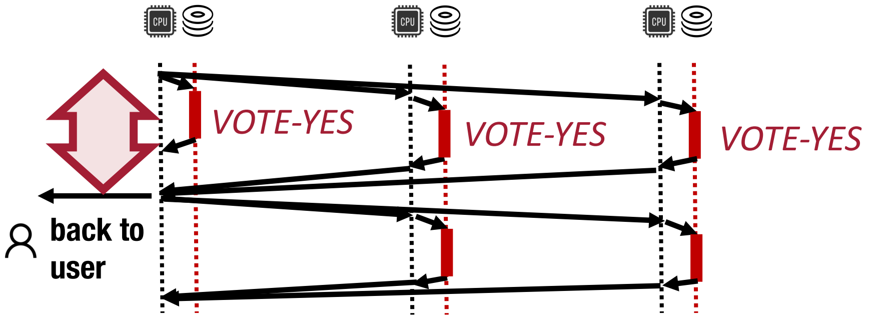
Two-Phase Commit

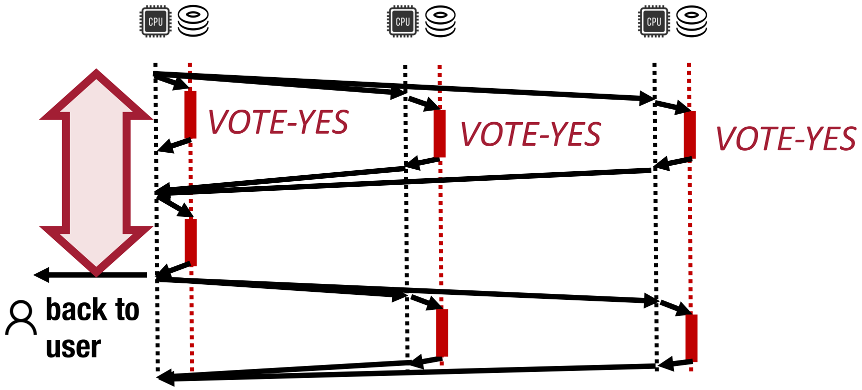


Cornus vs. 2PC Summary

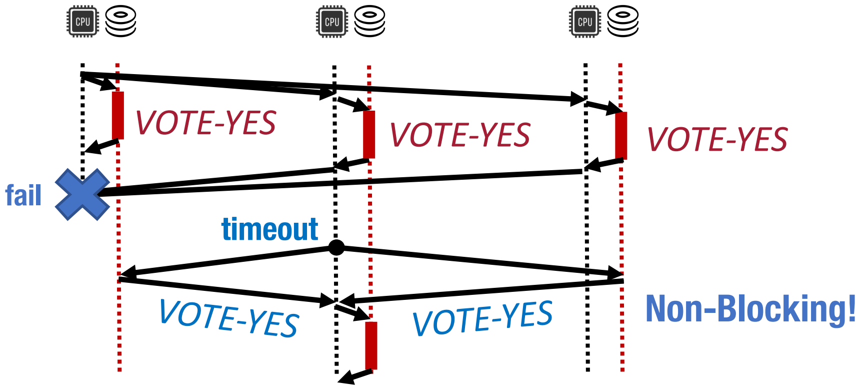
Cornus | **Two-Phase Commit**

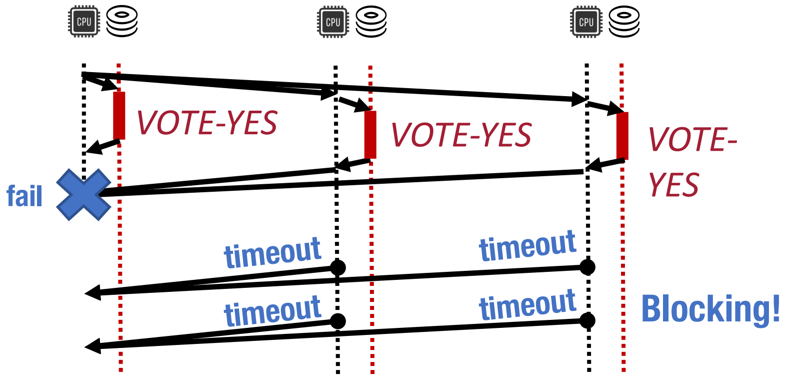
Commit Case





Failure Case



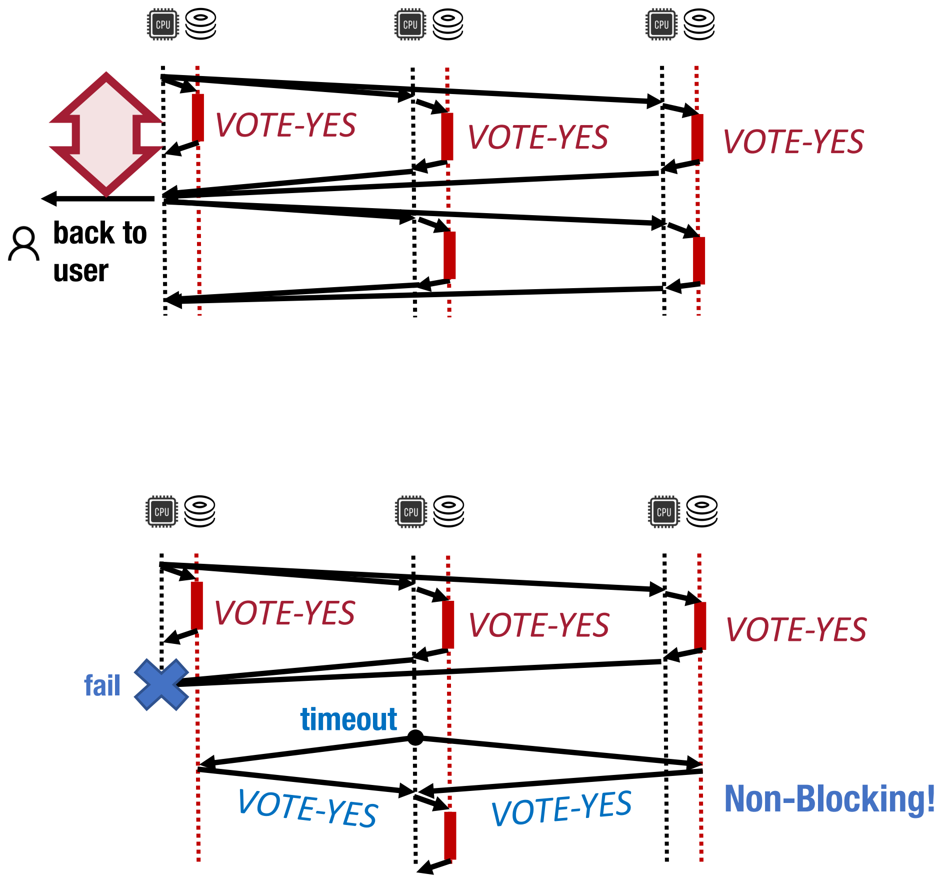


Cornus vs. 2PC Summary

Cornus

Commit Case

Failure Case



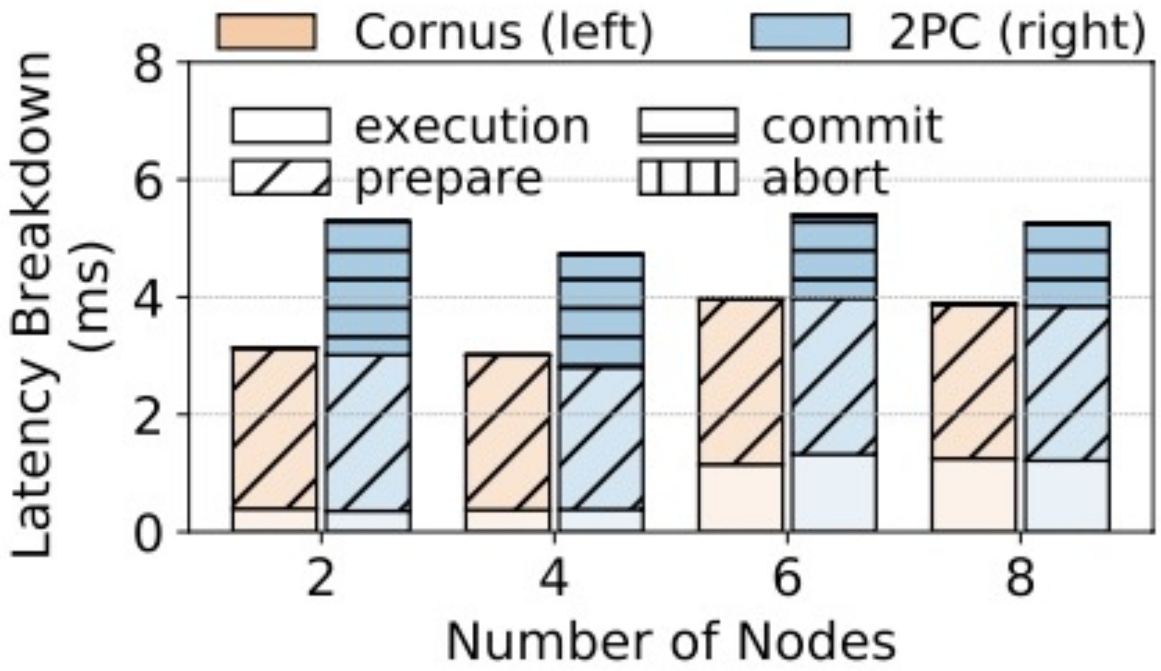
Key idea #1: **No decision logging**

Key idea #2: **LogOnce() storage API**

Enabled by storage disaggregation through

- Highly available storage service
- Shared across compute nodes
- Rich APIs of storage service

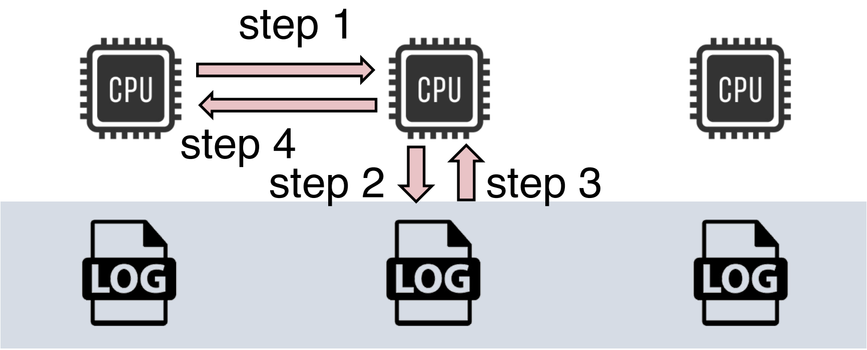
Performance Evaluation (on Redis)



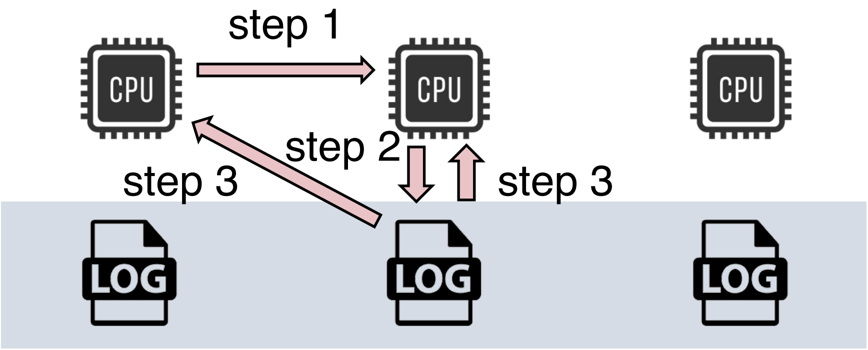
Cornus reduces latency by up to **1.9×** compared to 2PC

Hardware: 8 core (Intel Xeon 8272CL × 8), 64 GB DRAM
Workload: 10GB YCSB data set, 16 accesses per txn, reads/updates = 50/50, no skew
Storage service: Premium P4 Redis instance on Azure. One master node + one slave node.

Further Optimizations



Prepare in Cornus

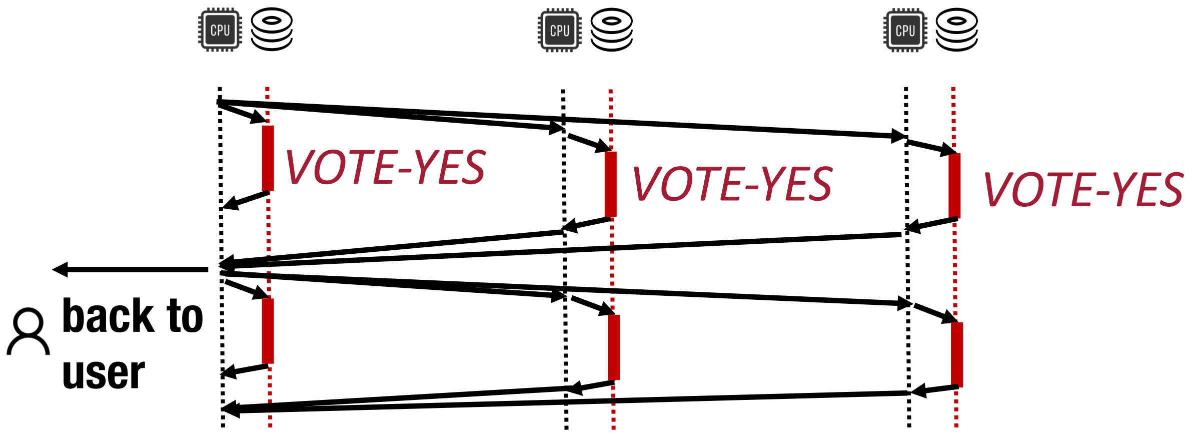


Optimization #1

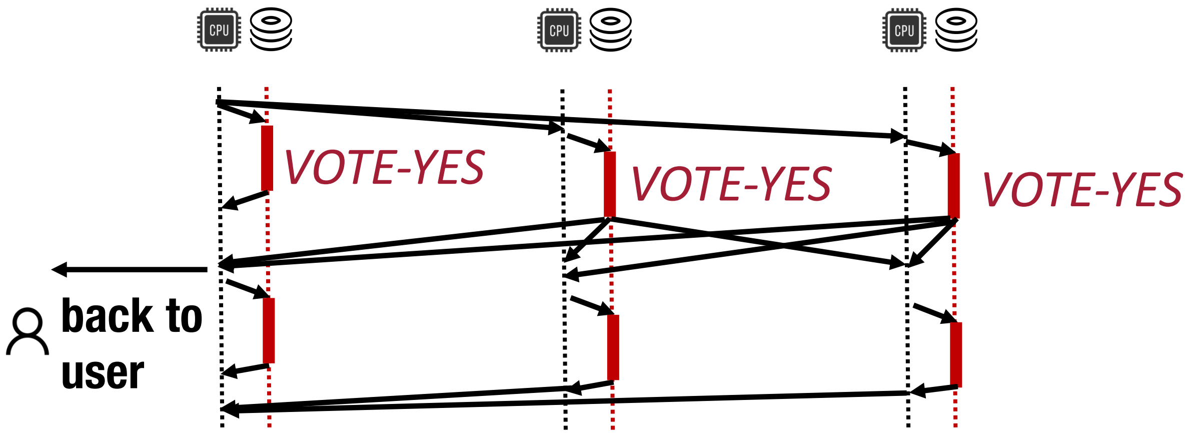
Optimization #1: Storage service responds to both the requesting participant and coordinator

- Save one network hop
- Requires changes in storage API

Further Optimizations



Baseline Cornus

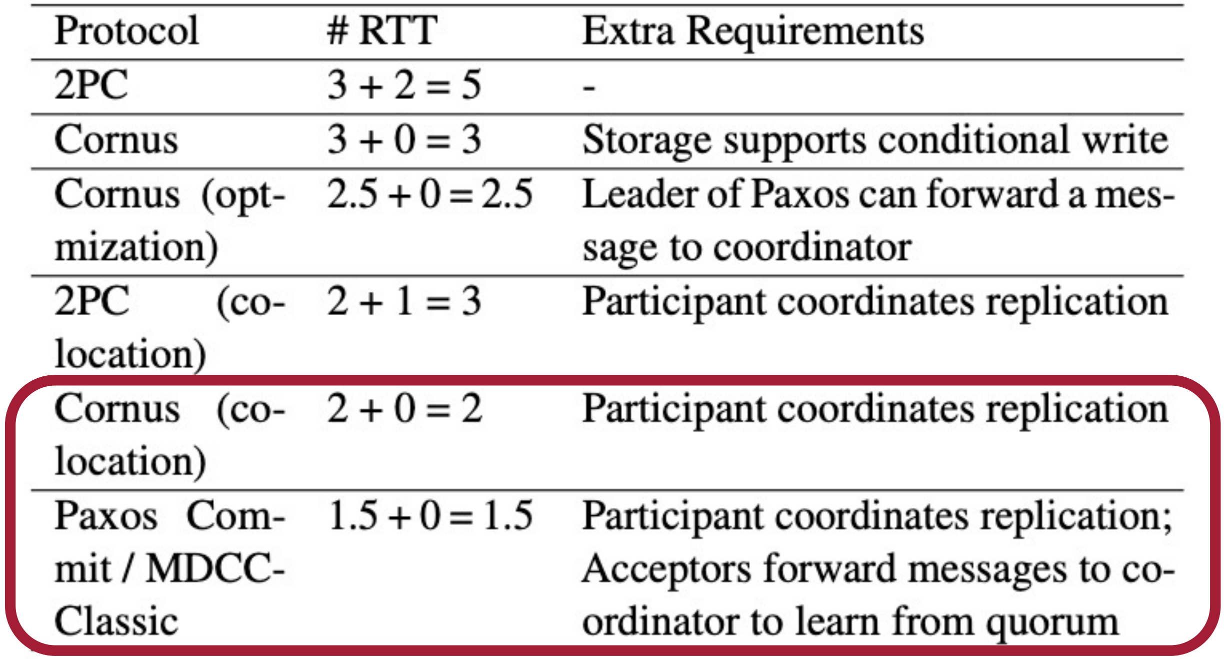


Optimization #2

Optimization #2: Storage service responds to coordinator and all participants

- Save one more network hot
- Incurs more network traffic
- Requires changes in storage API

Further Optimizations

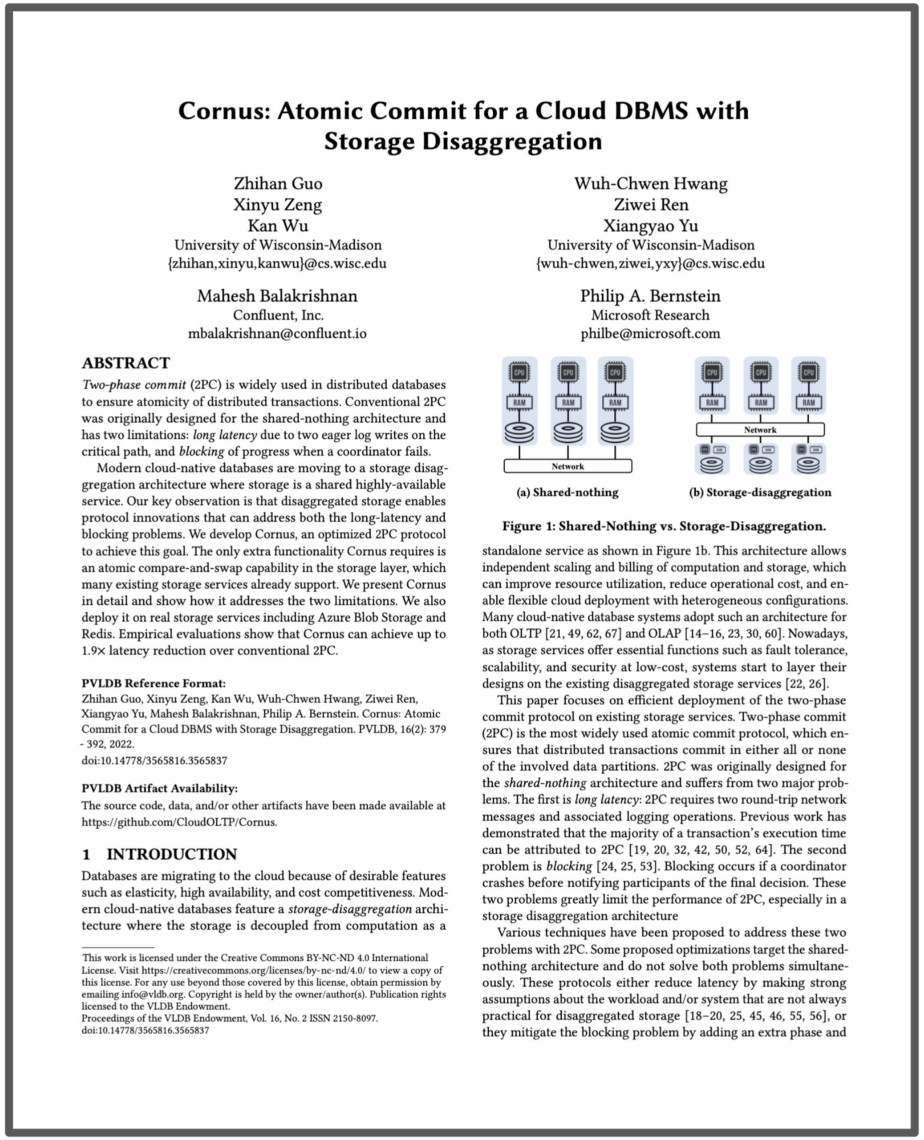


Protocol	# RTT	Extra Requirements
2PC	$3 + 2 = 5$	-
Cornus	$3 + 0 = 3$	Storage supports conditional write
Cornus (optimization)	$2.5 + 0 = 2.5$	Leader of Paxos can forward a message to coordinator
2PC (co-location)	$2 + 1 = 3$	Participant coordinates replication
Cornus (co-location)	$2 + 0 = 2$	Participant coordinates replication
Paxos Commit / MDCC-Classic	$1.5 + 0 = 1.5$	Participant coordinates replication; Acceptors forward messages to coordinator to learn from quorum

Table 3: Time complexity for protocols integrating with Paxos or its variations

Further optimizations require the codesign of 2PC and consensus

Check out Our VLDB’22 Paper



Cornus: Atomic Commit for a Cloud DBMS with Storage Disaggregation

Zhihan Guo
Xinyu Zeng
Kan Wu
University of Wisconsin-Madison
{zhihan,xinyu,kanwu}@cs.wisc.edu

Wuh-Chwen Hwang
Ziwei Ren
Xiangyao Yu
University of Wisconsin-Madison
{wuh-chwen,ziwei,yxy}@cs.wisc.edu

Mahesh Balakrishnan
Confluent, Inc.
mbalakrishnan@confluent.io

Philip A. Bernstein
Microsoft Research
philbe@microsoft.com

ABSTRACT

Two-phase commit (2PC) is widely used in distributed databases to ensure atomicity of distributed transactions. Conventional 2PC was originally designed for the shared-nothing architecture and has two limitations: *long latency* due to two eager log writes on the critical path, and *blocking* of progress when a coordinator fails.

Modern cloud-native databases are moving to a storage disaggregation architecture where storage is a shared highly-available service. Our key observation is that disaggregated storage enables protocol innovations that can address both the long-latency and blocking problems. We develop Cornus, an optimized 2PC protocol to achieve this goal. The only extra functionality Cornus requires is an atomic compare-and-swap capability in the storage layer, which many existing storage services already support. We present Cornus in detail and show how it addresses the two limitations. We also deploy it on real storage services including Azure Blob Storage and Redis. Empirical evaluations show that Cornus can achieve up to 1.9× latency reduction over conventional 2PC.

PVLDB Reference Format:
Zhihan Guo, Xinyu Zeng, Kan Wu, Wuh-Chwen Hwang, Ziwei Ren, Xiangyao Yu, Mahesh Balakrishnan, Philip A. Bernstein. Cornus: Atomic Commit for a Cloud DBMS with Storage Disaggregation. PVLDB, 16(2): 379 - 392, 2022.
doi:10.14778/3565816.3565837

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/CloudOLTP/Cornus.

1 INTRODUCTION

Databases are migrating to the cloud because of desirable features such as elasticity, high availability, and cost competitiveness. Modern cloud-native databases feature a *storage-disaggregation* architecture where the storage is decoupled from computation as a standalone service as shown in Figure 1b. This architecture allows independent scaling and billing of computation and storage, which can improve resource utilization, reduce operational cost, and enable flexible cloud deployment with heterogeneous configurations. Many cloud-native database systems adopt such an architecture for both OLTP [21, 49, 62, 67] and OLAP [14–16, 23, 30, 60]. Nowadays, as storage services offer essential functions such as fault tolerance, scalability, and security at low-cost, systems start to layer their designs on the existing disaggregated storage services [22, 26].

(a) Shared-nothing

(b) Storage-disaggregation

Figure 1: Shared-Nothing vs. Storage-Disaggregation.

This paper focuses on efficient deployment of the two-phase commit protocol on existing storage services. Two-phase commit (2PC) is the most widely used atomic commit protocol, which ensures that distributed transactions commit in either all or none of the involved data partitions. 2PC was originally designed for the *shared-nothing* architecture and suffers from two major problems. The first is *long latency*: 2PC requires two round-trip network messages and associated logging operations. Previous work has demonstrated that the majority of a transaction’s execution time can be attributed to 2PC [19, 20, 32, 42, 50, 52, 64]. The second problem is *blocking* [24, 25, 53]. Blocking occurs if a coordinator crashes before notifying participants of the final decision. These two problems greatly limit the performance of 2PC, especially in a storage disaggregation architecture

Various techniques have been proposed to address these two problems with 2PC. Some proposed optimizations target the shared-nothing architecture and do not solve both problems simultaneously. These protocols either reduce latency by making strong assumptions about the workload and/or system that are not always practical for disaggregated storage [18–20, 25, 45, 46, 55, 56], or they mitigate the blocking problem by adding an extra phase and

Proceedings of the VLDB Endowment, Vol. 16, No. 2 ISSN 2150-8097.
doi:10.14778/3565816.3565837

- Pseudo-code of Cornus
- Analysis of failure and recovery
- Proof of correctness
- Deployment over Redis and Azure blob store
- More performance evaluation

Q/A – Cornus

What implementation of 2PC used for comparison?

Cornus on a shared-nothing architecture?

Consensus algorithm like Paxos or Raft used for replication?

Completely decouple compute sharding from storage sharding?

In storage disaggregation, any strength to partition keys? Why not to run one transaction only in one node?

Consistency required from underlying storage service?

How does storage implement compare-and-swap?

Next Lecture

Yi Lu, et al., Aria: A Fast and Practical Deterministic OLTP Database. VLDB, 2020