Understanding Two-Phase and Three-Phase Commit Protocols: Key Differences, Use Cases, and Practical Examples

Damini Bansal Follow

4 min read · Nov 6, 2024

52 6

In distributed systems, coordinating transactions across multiple resources can be complex. Imagine you have multiple systems or databases that need to work in sync, ensuring all either commit or rollback in case of failure. Two well-known protocols for achieving this are the **Two-Phase Commit (2PC)** and **Three-Phase Commit (3PC)**. Both protocols play a crucial role in ensuring data consistency, particularly in distributed environments, but they differ in their approach, functionality, and resilience to system failures.

Let's dive into what these protocols are, how they work, and when to use each.

1. Two-Phase Commit Protocol (2PC)

What is 2PC?

The **Two-Phase Commit Protocol (2PC)** is a widely used approach to ensure atomic transactions in distributed systems. In 2PC, a coordinator oversees multiple participants and ensures that either all participants agree to commit the transaction, or if any participant fails to commit, all participants rollback.

How 2PC Works

2PC operates in two phases: the **Prepare Phase** and the **Commit Phase**.

1. **Preparation Phase:**

- The coordinator sends a **Prepare** request to all participants.
- Each participant checks if it can complete the transaction and responds with either a **Yes** (prepared) or **No** (abort).
- If any participant responds with a No, the process halts, and the transaction is rolled back.

2. **Commit Phase:**

- If all participants respond with Yes, the coordinator sends a **Commit** command, and all participants commit the transaction.
- If any participant responds with No, the coordinator sends an **Abort** command to rollback the transaction across all participants.

Example Scenario for 2PC

Imagine an e-commerce website handling an order transaction:

- When a customer places an order, the e-commerce system needs to check the **inventory** (whether items are in stock), **payment processing** (funds availability), and **delivery system** (delivery partner availability).
- The coordinator (order system) sends a Prepare request to each component.
- If all components confirm they are ready, the coordinator proceeds with Commit. If any component indicates an issue (e.g., insufficient stock), the coordinator aborts the transaction to maintain consistency.

Limitations of 2PC

2PC does not handle network failures very well:

- If the coordinator fails during the Commit Phase, some participants may not receive the final decision.
- It can result in **blocking** (participants waiting indefinitely), especially if the coordinator fails to send a response.

2PC is ideal for systems where high availability is less of a concern, and network failures are rare. It is often used in **financial systems** or **enterprise resource planning (ERP)** systems, where consistency is more critical than availability.

Top highlight

2. Three-Phase Commit Protocol (3PC)

What is 3PC?

The **Three-Phase Commit Protocol (3PC)** is an enhanced version of 2PC designed to address the limitations of 2PC, specifically the risk of blocking. 3PC introduces an additional phase between Prepare and Commit, known as the **Pre-Commit** phase, to ensure participants have enough time to respond, reducing the risk of indefinite blocking.

How 3PC Works

3PC operates in three phases: **Prepare Phase**, **Pre-Commit Phase**, and **Commit Phase**.

1. **Prepare Phase:**

- Similar to 2PC, the coordinator sends a **Prepare** request to participants, asking if they can commit the transaction.
- Each participant responds with Yes or No.

2. Pre-Commit Phase:

- If all participants respond Yes, the coordinator sends a **Pre-Commit** message, indicating the transaction is likely to be committed.
- Participants acknowledge the Pre-Commit, preparing to commit the transaction but not yet completing it.

3. Commit Phase:

- If all participants acknowledge the Pre-Commit, the coordinator sends the **Commit** command.
- If any participant fails to acknowledge the Pre-Commit, the transaction is **Aborted.**

3PC mitigates blocking issues by adding a timeout mechanism, allowing participants to autonomously decide to commit or abort if they don't receive further messages from the coordinator.

Example Scenario for 3PC

Imagine a financial application processing a transaction across multiple bank accounts:

- A transaction coordinator oversees several bank databases to transfer funds between accounts.
- The coordinator sends a Prepare request to all databases.
- If all respond positively, the coordinator sends a Pre-Commit message to ensure all are ready.
- After receiving acknowledgments, the coordinator finally sends the Commit request to finalize the transaction. If any bank fails to acknowledge, the transaction is aborted.

Advantages of 3PC

3PC reduces the likelihood of indefinite blocking, particularly helpful in situations with unreliable networks:

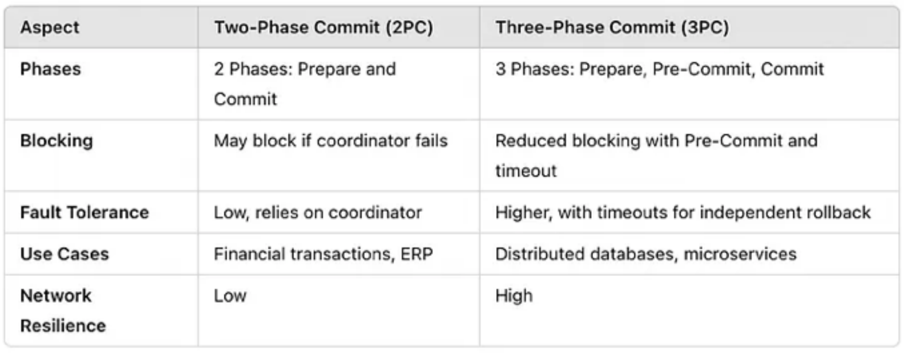
- **Timeout Mechanism**: Allows participants to make autonomous decisions if they don't hear back from the coordinator.
- **Improved Fault Tolerance**: Reduces the risk of hanging transactions.

When to Use 3PC

3PC is ideal for high-availability, fault-tolerant systems where network interruptions are more likely. Use cases include **distributed databases** with **geo-redundant setups** or **microservices architectures** where network stability can vary across services.

. . .

Key Differences Between 2PC and 3PC



Aspect	Two-Phase Commit (2PC)	Three-Phase Commit (3PC)
Phases	2 Phases: Prepare and Commit	3 Phases: Prepare, Pre-Commit, Commit
Blocking	May block if coordinator fails	Reduced blocking with Pre-Commit and timeout
Fault Tolerance	Low, relies on coordinator	Higher, with timeouts for independent rollback
Use Cases	Financial transactions, ERP	Distributed databases, microservices
Network Resilience	Low	High

. . .

Choosing Between 2PC and 3PC

Use 2PC if your system prioritizes **consistency** over **availability** and has reliable network connectivity. Examples include:

- **Banking systems** where each transaction's atomicity is paramount.
- **Inventory management** systems where resource locking ensures stock levels remain accurate.

Use 3PC if your system requires **high availability** and must handle **network unreliability** gracefully. Examples include:

- **Distributed databases** where transactions are spread across regions.
- **Microservices** in cloud environments with potential network interruptions.

. . .

Understanding these protocols allows engineers to choose the right approach to balance data consistency, fault tolerance, and performance in distributed systems. While 2PC is simpler and well-suited to stable networks, 3PC shines in high-availability systems where network stability is less predictable.

Two Phase Commit · Three Phase · 2pc · 3pc · Distributed Systems

Written by Damini Bansal

53 followers · 9 following

Follow

Love to be lazy as lazy find an easiest way to do hard job.

Responses (6)

Ishu

What are your thoughts?

alex west
Sep 1

basically diff between 2PC ND 3PC IS SIMPLE

in 3pc you FIRST ask subsystems about being just available, all of them !

and in 2pc you skip this step assuming all sub systems are already available and ready to pre-commit!

7 Reply

alex west
Sep 1

still not clear!!!

why you cant use timeout in 2PC after pre-commit? you sent back pre-commit w/ timeout!

if you did not get commit event w/in time out, you abort...

what is problem?

2 Reply

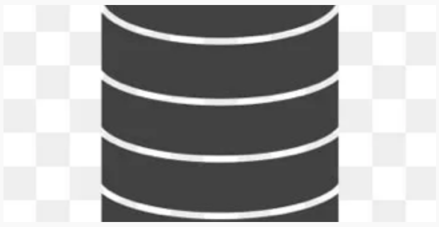
Nitin Agrawal
Jul 7

If I understand it correct, then difference boils down to 'timeout' feature. So can't we add this timeout in 2PC & let the components decide if need to role back?

1 1 reply Reply

See all responses

More from Damini Bansal

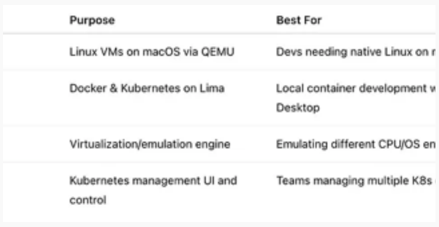


Damini Bansal

Understanding the Key Differences: Unique Index, Index,…

Here's a breakdown of the differences between Unique Index, Index, Unique…

Oct 25, 2024 6



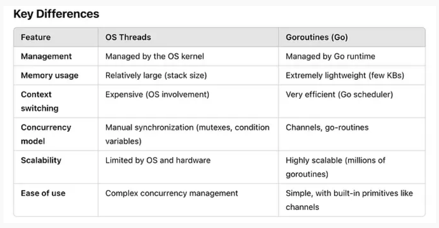
Purpose	Best For
Linux VMs on macOS via QEMU	Devs needing native Linux on
Docker & Kubernetes on Lima	Local container development Desktop
Virtualization/emulation engine	Emulating different CPU/OS en
Kubernetes management UI and control	Teams managing multiple K8s

Damini Bansal

Building a Container-Ready Development Environment on…

The modern software development workflow increasingly revolves around containers,…

Aug 26



Key Differences

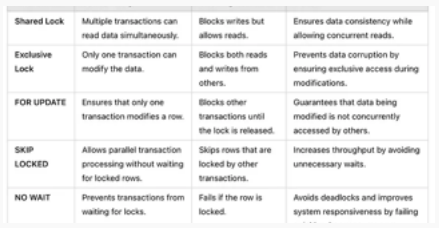
Feature	OS Threads	Goroutines (Go)
Management	Managed by the OS kernel	Managed by Go runtime
Memory usage	Relatively large (stack size)	Extremely lightweight (few KBs)
Context switching	Expensive (OS involvement)	Very efficient (Go scheduler)
Concurrency model	Manual synchronization (mutexes, condition variables)	Channels, go-routines
Scalability	Limited by OS and hardware	Highly scalable (millions of goroutines)
Ease of use	Complex concurrency management	Simple, with built-in primitives like channels

Damini Bansal

OS Threads vs Goroutines: Understanding the Concurrency…

In programming, OS threads and goroutines are both mechanisms for achieving…

Nov 13, 2024 6



Shared Lock	Multiple transactions can read data simultaneously.	Blocks writes but allows reads.	Ensures data consistency while allowing concurrent reads.
Exclusive Lock	Only one transaction can modify the data.	Blocks both reads and writes from others.	Prevents data corruption by ensuring exclusive access during modifications.
FOR UPDATE	Ensures that only one transaction modifies a row.	Blocks other transactions until the lock is released.	Guarantees that data being modified is not concurrently accessed by others.
SKIP LOCKED	Allows parallel transaction processing without waiting for locked rows.	Skips rows that are locked by other transactions.	Increases throughput by avoiding unnecessary waits.
NO WAIT	Prevents transactions from waiting for locks.	Fails if the row is locked.	Avoids deadlocks and improves system responsiveness by failing

Damini Bansal

Understanding Database Locks

Database concurrency is essential to managing multiple transactions that access…

Nov 15, 2024 2

See all from Damini Bansal

Recommended from Medium

In Javarevisited by Code Wiz

Understanding JVM Memory architecture and guidelines and...

Have you ever faced memory issues in your Java applications? Whether it's an...

May 18 23 1

Arvind Kumar

Building a Ticketing System: Concurrency, Locks, and Race...

What happens when 100,000 fans try to book the same concert ticket at exactly 10:00 AM...

Oct 30 238 4

In JavaScript in Plain Engl... by Umesh Kumar Yad...

I Can't Tell the Difference Between Java Generic Symbols: T, E, K, V, ?

Today I want to talk to you about those dazzling symbols in Java generics—T, E, K, ...

Oct 26 212 8

Mayank Yadav

Kafka Consumer Offsets Demystified: What Happens on...

Working with Apache Kafka in real-time applications can be tricky—especially when...

May 14 60

Alpesh Dhamelia

Mastering the Producer-Consumer Pattern: Backpressure,...

In today's era of microservices and distributed systems, data flow management...

Jul 30

Ahmet Temel Kundupoglu

The Most Common @Transactional Errors in Java Spring—And How t...

In the Spring ecosystem, the @Transactional annotation is a powerful tool to manage...

Jun 15 12

See more recommendations