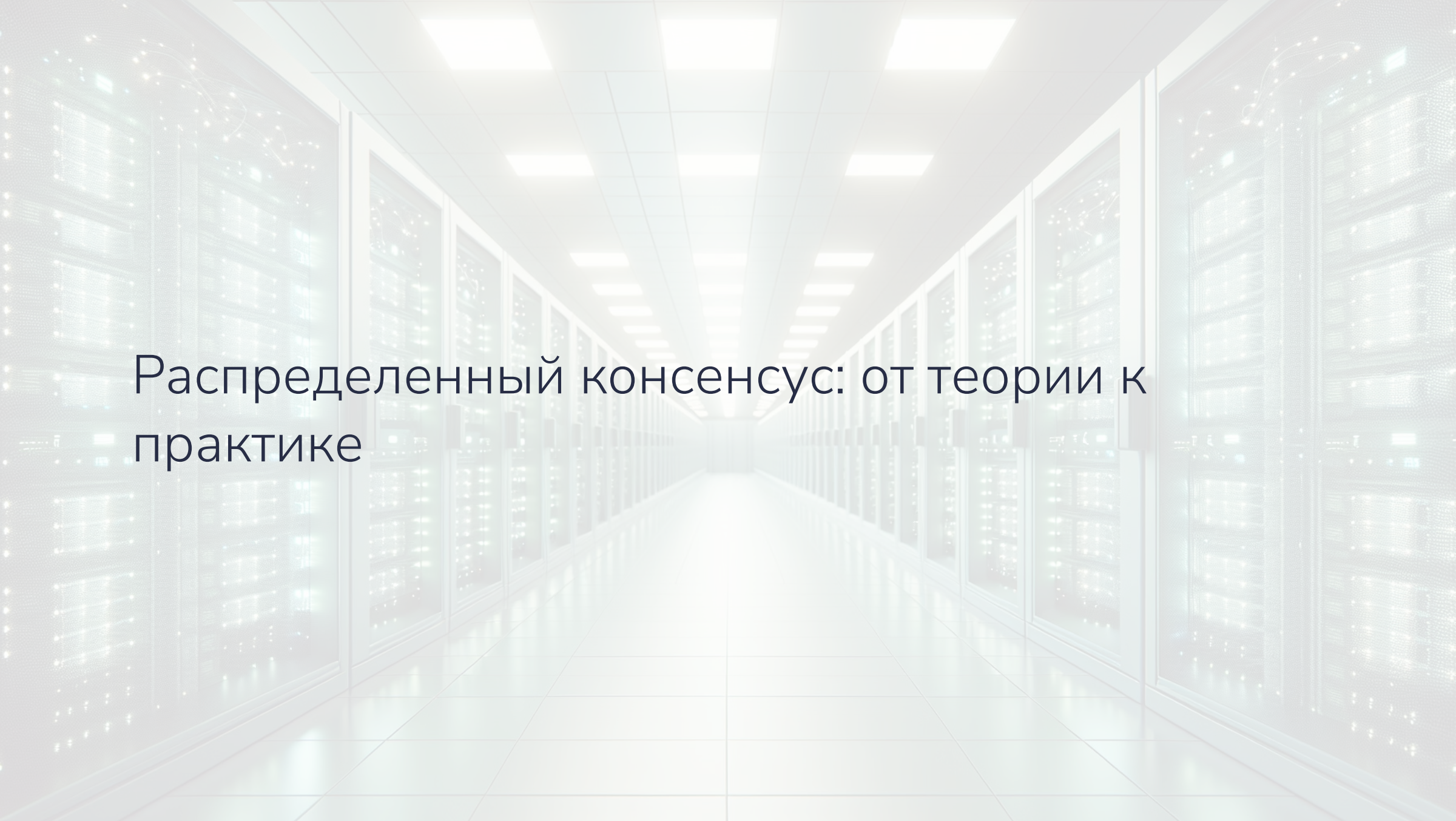


Распределенный консенсус: от теории к практике

Основы распределенного консенсуса

Распределенный консенсус решает фундаментальную проблему достижения согласия между несколькими независимыми компьютерами, даже если некоторые узлы могут выйти из строя или стать недоступными.

Рассмотрим банковскую систему, в которой несколько серверов должны поддерживать согласованные остатки на счетах: если сервер A регистрирует 100 долларов для Алисы, а сервер B показывает 50 долларов из-за пропущенной транзакции, определение правильного остатка для снятия 75 долларов становится критической проблемой консенсуса.

Распределенный консенсус представляет собой одну из самых фундаментальных задач в информатике, лежащую в основе важнейших функций распределенных систем, включая согласованность баз данных, выбор лидера в кластерах, атомарные транзакции между службами и скоординированные изменения конфигурации.

Сложность консенсуса

Задача консенсуса выходит за рамки простой коммуникации и включает в себя принятие решений в условиях неопределенности. Сети могут разбиваться на части, серверы могут выходить из строя в любой момент, а сообщения могут задерживаться или теряться. Системы должны одновременно поддерживать согласованность и продолжать работу, несмотря на эти проблемы.

Сетевые разделы

Узлы могут потерять связь друг с другом

Отказы серверов

Узлы могут выйти из строя в любой момент

Потеря сообщений

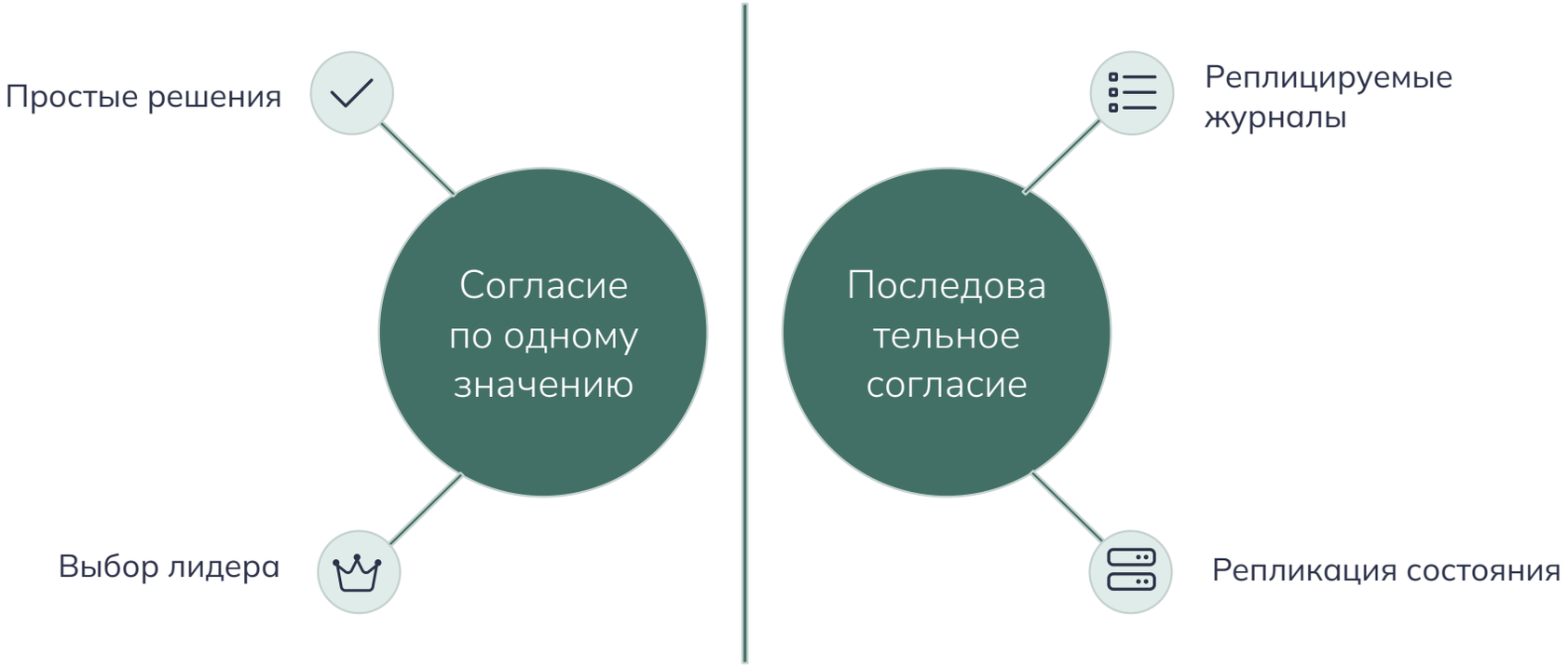
Сообщения могут задерживаться или теряться

Алгоритмы консенсуса решают эту проблему, признавая, что ожидание универсальных ответов чревато бесконечными задержками (неисправные узлы могут никогда не ответить), а односторонние решения чревато несогласованностью (лицо, принимающее решение, может оказаться в сети с разбиением). Решение заключается в требовании согласия большинства узлов, что обеспечивает как безопасность (согласованность), так и работоспособность (прогресс).

Распределенный консенсус тесно связан с атомарной трансляцией — задачей доставки сообщений всем узлам в одинаковом порядке. Консенсус и атомарная трансляция функционально взаимозаменяемы, поскольку каждый из них может быть реализован с помощью другого. В большинстве практических систем для достижения атомарной трансляции используются алгоритмы консенсуса (хотя некоторые реализации, такие как ZAB в ZooKeeper, позиционируют себя как протоколы атомарной трансляции), обеспечивающие обработку всех реплик в одинаковой последовательности.

Для чего нужен консенсус?

Прежде чем рассматривать конкретные алгоритмы, необходимо понять суть требований к соглашению. Несмотря на разнообразные проявления, проблемы консенсуса сводятся к двум основным моделям: согласие по одному значению или согласие по последовательности (достигаемое с помощью нескольких экземпляров консенсуса по одному значению). Большинство практических приложений представляют собой реализации этих основных моделей.



Консенсус по одному значению

Консенсус по значению — это классическая и наиболее фундаментальная форма консенсуса.

Цель: Все участники системы (узлы) должны выбрать **одно-единственное значение** из набора предложенных.

Представьте, что группа серверов должна выбрать нового "лидера" из своего числа. Каждый сервер может предложить себя или другого кандидата. В итоге все **живые** серверы должны согласиться, что "Сервер А" является лидером. Если половина серверов считает лидером "Сервера А", а другая — "Сервера Б", система не сможет работать.

Любой алгоритм, решающий эту задачу, должен гарантировать три вещи:

1. **Согласование (Agreement):** Все не вышедшие из строя (не "упавшие") узлы должны выбрать **одно и то же значение**.
2. **Обоснованность (Validity):** Если все узлы предложили одно и то же значение v, то они должны выбрать именно v. В более общем случае, выбранное значение должно быть одним из тех, что были предложены **хотя бы одним** узлом.
3. **Завершаемость (Termination):** Каждый не вышедший из строя узел в конечном итоге **должен принять решение** (выбрать значение) и остановиться.

Примеры:

- **Выбор лидера**: какой узел (A, B или C) должен быть лидером?
- **Изменения конфигурации**: следует ли добавить узел D в кластер?
- **Простые решения**: следует ли включить режим обслуживания (да/нет)?
- **Членство в кластере**: текущие активные узлы должны быть {A, B, C}
- **Изменения членства**: безопасный переход от {A, B, C} к {A, B, C, D}

> **Алгоритмы:** Здесь превосходит базовый алгоритм Paxos, который был разработан специально для достижения консенсуса по одному значению. Совместный консенсус Raft и варианты Paxos с протоколами членства позволяют обрабатывать более сложные сценарии изменения членства.

Консенсус по последовательности

Консенсус по последовательности, также известный как **проблема упорядочивания** или **репликация журнала (Log Replication)**, является более сложной и практичной задачей, построенной на основе консенсуса по значению.

Цель: Узлы должны договориться не просто об одном значении, а о **едином, упорядоченном списке (журнале)** значений.

В большинстве реальных систем (как базы данных или блокчейны) нам нужно обрабатывать не одно событие, а непрерывный поток команд или транзакций.

> **Пример:** В банковской системе есть два перевода:
>
> 1. Алекс вносит $100.
> 2. Алекс снимает $50.
>
> Договориться о *наборе* этих транзакций недостаточно. Критически важен их **порядок**. Если сначала применить (1), а потом (2), баланс будет $50. Если (случайно) применить сначала (2), а потом (1), то транзакция (2) может быть отклонена из-за недостатка средств (если начальный баланс был $0).

Примеры:

- **Транзакции базы данных**: [INSERT user Alice; UPDATE account balance; DELETE old record]
- **Репликация состояний машины**: [SET x=1, SET y=2, DELETE z]
- **Потоки событий**: [OrderCreated, PaymentProcessed, OrderShipped]
- **Блокчейн**: блоки должны быть в одном порядке на всех узлах
- **Исходные события**: события должны быть упорядочены последовательно для всех потребителей

Алгоритмы: Multi-Paxos и Raft разработаны специально для этой цели — они могут эффективно согласовывать множество значений в последовательности, обеспечивая при этом полную упорядоченность.

Подход на основе конечных автоматов

Консенсус по последовательности — это ключ к **репликации конечного автомата (SMR)**. Это фундаментальная концепция для создания отказоустойчивых систем.

Идея проста:

1. Все узлы в системе начинаются с **одинакового начального состояния**.
2. Все узлы получают **один и тот же список команд** (транзакций).
3. Благодаря алгоритму консенсуса, они договариваются об **одной и той же последовательности** этих команд.
4. Каждый узел применяет команды к своему состоянию в этом едином порядке.

Поскольку все начинают с одного и того же и применяют одни и те же изменения в одном и том же порядке, они **гарантированно придут к одному и тому же конечному состоянию**. Так работают распределенные базы данных: вы отправляете запрос на изменение данных, а алгоритм консенсуса (как Raft) следит, чтобы эта команда попала в общий "журнал" на всех серверах в правильном порядке.

Известные алгоритмы

Алгоритмы, решающие эту задачу, делятся на две большие семьи:

- **Устойчивые к сбоям (Crash-Fault Tolerant - CFT):**
  - **Paxos:** "Дедушка" всех алгоритмов консенсуса. Он математически корректен, но известен своей сложностью в понимании и реализации.
  - **Raft:** Разработан как более понятная и практичная альтернатива Paxos. Он явно разделяет процесс на выборы лидера и репликацию журнала. Raft используется во многих современных системах (например, etcd, Consul).
- **Устойчивые к "византийским" сбоям (Byzantine-Fault Tolerant - BFT):**
  - Эти алгоритмы выдерживают не только "падение" узлов, но и их злонамеренное поведение (ложь, отправку противоречивой информации).
  - **PBFT (Practical Byzantine Fault Tolerance):** Классический BFT-алгоритм для "закрытых" (permissioned) систем.
  - **Proof-of-Work (PoW):** Алгоритм, используемый в Биткойне. Он решает проблему консенсуса по последовательности (цепочке блоков) в открытой, анонимной среде, делая "подделку" истории вычислительно невыгодной.
  - **Proof-of-Stake (PoS):** Альтернатива PoW (используется в Ethereum), где право на добавление нового блока в последовательность определяется владением долей в сети, а не вычислительной мощностью.

Как алгоритмы соответствуют требованиям

Различные алгоритмы консенсуса оптимизированы для разных сценариев:

Basic Paxos

идеально подходит для решений с одним значением, но неэффективен для последовательностей. Для каждого решения необходимо запускать отдельный экземпляр Paxos.

Multi-Paxos

оптимизирует Paxos для последовательностей, выбирая стабильного лидера, который может предлагать несколько значений без повторения фазы подготовки.

Raft

разработан с нуля для реплицированных журналов. Модель сильного лидера делает консенсус по последовательности естественным и эффективным.

PBFT

расширяет консенсус для обработки вредоносных узлов, что крайне важно для блокчейнов и многоорганизационных систем.

ZAB

разработан специально как протокол атомарной трансляции, обеспечивающий полную упорядоченность доставки сообщений.

Paxos: теоретические основы

Paxos, представленный Лесли Лампортом в 1989 году, был первым практическим решением для распределенного консенсуса. Он математически элегантен, но известен своей сложностью для понимания и правильной реализации. Несмотря на свою сложность, Paxos лежит в основе некоторых крупнейших в мире распределенных систем, включая Google Spanner, службу блокировки Chubby, и используется в облегченных транзакциях Apache Cassandra.

> Paxos — это математически элегантное, но практически сложное решение для распределенного консенсуса

Основная идея Paxos

Paxos работает так, что узлы берут на себя разные роли в тщательно отрепетированном танце предложений и обещаний. Ключевая идея заключается в использовании двухфазного протокола: сначала подготавливают почву, получая обещания от большинства (обещания игнорировать любые будущие предложения с более низкими номерами), а затем предлагают значение, которое соответствует этим обещаниям.

Роли в Paxos

Предлагающий

инициирует процесс консенсуса, предлагая значения

Принимающий

голосует за предложения и обещает игнорировать более старые предложения

Учащийся

узнает выбранное значение после достижения консенсуса

Один узел может играть несколько ролей одновременно.

Рабочий процесс Paxos

Процесс начинается, когда клиент отправляет запрос любому узлу в кластере (например, «установить конфигурацию X» или «выбрать лидера Y»). Этот узел становится предлагающим и инициирует двухфазный протокол консенсуса, чтобы все узлы согласились с значением, предложенным клиентом.

Фаза 1 — Подготовка:

1. Предлагающий генерирует уникальный номер предложения N
2. Отправляет «prepare(N)» большинству принимающих
3. Каждый принимающий отвечает:
    - Обещанием игнорировать предложения < N
    - Предложение с наибольшим номером, которое он ранее принял (если таковое имеется), чтобы предлагающий знал, какое значение следует учитывать

Фаза 2 — Принятие:

1. Если большинство ответило, предлагающий выбирает значение:
    - Если принимающие вернули предыдущие предложения, используется значение из предложения с наибольшим номером
    - В противном случае используется значение, предложенное предлагающим
2. Отправляется «accept(N, value)» большинству принимающих
3. Принимающие принимают, если они не обещали игнорировать N
4. Как только большинство принимает, значение выбирается

Paxos на практике: Multi-Paxos

Базовый Paxos неэффективен для нескольких значений подряд — для каждого решения необходимо запускать полный протокол. Multi-Paxos оптимизирует этот процесс, выбирая стабильного лидера, который может пропустить фазу 1 для последующих предложений, что делает его практичным для реальных систем.

Basic Paxos

- Полный двухфазный протокол для каждого значения
- Высокие накладные расходы на сообщения
- Подходит для разовых решений

Multi-Paxos

- Стабильный лидер пропускает фазу подготовки
- Эффективен для последовательностей
- Основа для реплицированных журналов

Multi-Paxos работает так, что узлы соглашаются не только по отдельным значениям, но и по последовательности значений (как в реплицированном журнале). После установления лидера он может предлагать значения для нескольких позиций в журнале, не проходя каждый раз фазу подготовки. Базовый Paxos может гарантировать порядок, если запустить отдельные экземпляры для каждой позиции в последовательности (экземпляр Paxos 1 для записи журнала 1, экземпляр 2 для записи 2 и т. д.), но это неэффективно, поскольку каждый экземпляр требует полного двухфазного протокола.

Где используется Paxos

Google Spanner

использует Paxos для согласованных глобальных транзакций

Apache Cassandra

легкие транзакции используют Paxos для линеаризуемых операций

Google Chubby

служба распределенных блокировок, построенная на Paxos

Amazon DynamoDB

использует варианты Paxos для межрегиональной согласованности

Проблемы Paxos

- **Сложность реализации**: легко ошибиться, сложно отлаживать
- **Производительность при конфликтах**: несколько инициаторов могут привести к лайвлокам
- **Понятность**: даже опытные инженеры сталкиваются с крайними случаями
- **Накладные расходы на передачу сообщений**: двухфазный протокол требует большего количества сетевых циклов

Raft: консенсус для людей

Raft был разработан в 2013 году с конкретной целью: сделать распределенный консенсус понятным. Создатели поняли, что сложность Paxos препятствует внедрению и инновациям в распределенных системах. Raft обеспечивает те же гарантии, что и Paxos, но с дизайном, который инженеры могут действительно понять.

> «Понятность была нашей основной целью при разработке Raft» — создатели алгоритма

Основная идея Raft

Raft упрощает консенсус, разбивая его на три независимые подзадачи:

Выбор лидера

как выбрать координатора

Репликация журнала

как лидер поддерживает согласованность

Безопасность

обеспечение согласованности системы даже во время сбоев

Ключевая идея заключается в наличии сильного лидера, который координирует все изменения, устраняя необходимость в нескольких конкурирующих инициаторах.

Роли и состояния Raft

- **Лидер**: обрабатывает запросы клиентов и координирует репликацию
- **Последователь**: пассивно принимает обновления от лидера
- **Кандидат**: стремится стать лидером во время выборов

Выбор лидера в Raft

Когда лидер выходит из строя или становится недоступным, или когда система запускается, Raft обеспечивает быстрое избрание нового лидера кластером. При запуске или когда последователи обнаруживают сбой лидера по отсутствию сигналов сердцебиения, узлы могут стать кандидатами и запросить голоса у других узлов. Первый кандидат, получивший большинство голосов, становится новым лидером на следующий срок.

> "Номер срока" (Term Number) — это, по сути, **порядковый номер "эпохи" или "правления"** в жизни кластера.

01

Обнаружение сбоя

Последователи не получают сигналы сердцебиения от лидера

02

Становление кандидатом

Узел увеличивает "порядковый номер эпохи" и голосует за себя

03

Запрос голосов

Кандидат отправляет RequestVote другим узлам

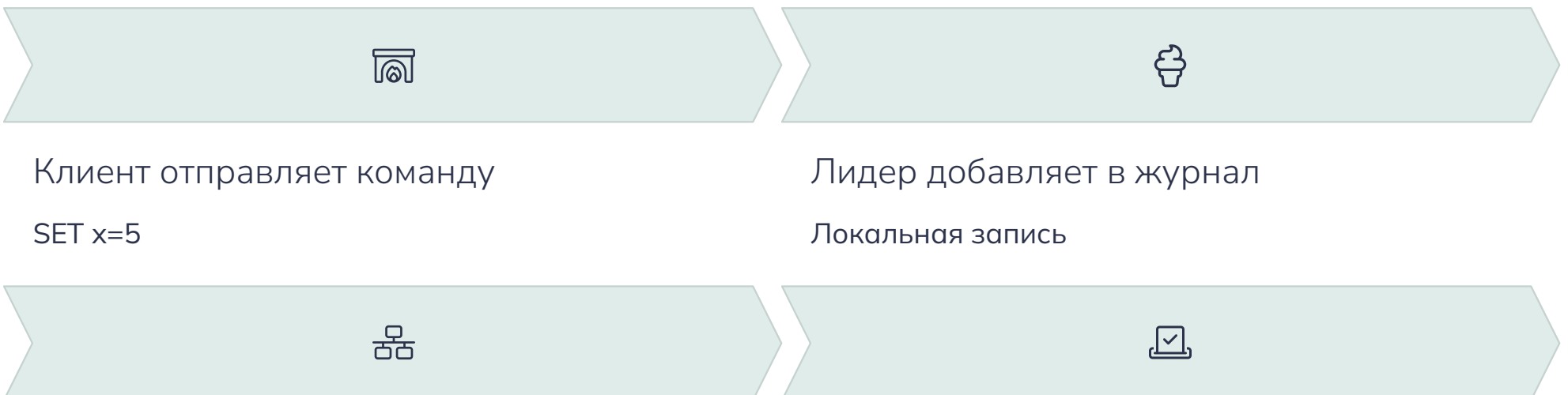
04

Получение большинства

Может случиться так, что почти одновременно "проснутся" **несколько** кандидатов. Они оба начнут свои выборы (например, "раунд №5" и "раунд №6") и оба начнут просить голоса. Кандидат с большинством голосов становится лидером

Репликация журнала в Raft

Лидер получает запросы от клиентов (любой узел может получать запросы, но последователи перенаправляют их лидеру), добавляет их в свой локальный журнал, а затем реплицирует записи в журналы последователей. Только после того, как большинство узлов сохранили запись, лидер фиксирует ее и уведомляет последователей. Это обеспечивает строгую согласованность между всеми узлами.

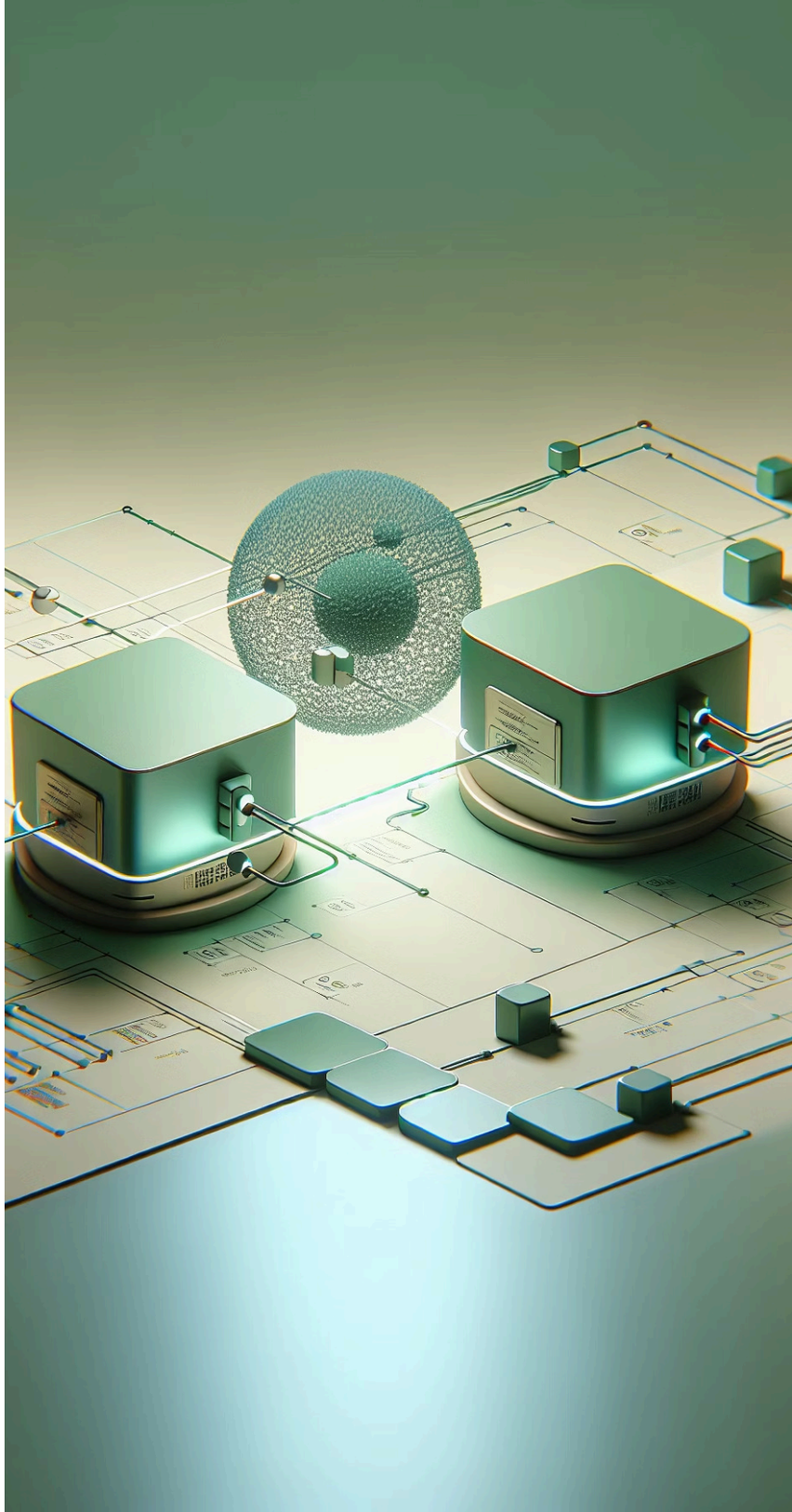


Этот процесс гарантирует, что все узлы в кластере имеют идентичные журналы и применяют команды в одинаковом порядке, обеспечивая строгую согласованность распределенной системы.

Безопасность и восстановление в Raft

Механизмы безопасности Raft предотвращают повреждение данных при повторном присоединении узлов после разделения. Каждая эпоха имеет не более одного лидера, а узлы с устаревшей информацией обновляются посредством проверки согласованности журналов. Более высокие номера эпох всегда переопределяют более низкие, обеспечивая сходимость кластера к единому согласованному состоянию.

1. **Разделение сети**
   Узел A изолирован от кластера
2. **Новый лидер**
   Узел B избран лидером эпохи 4
3. **Воссоединение**
   Узел A пытается отправить старые данные
4. **Обновление**
   A обнаруживает более высокий срок и синхронизируется



Свойства безопасности Raft

Raft гарантирует несколько ключевых свойств безопасности:

Безопасность выбора

не более одного лидера на эпоху

Лидер только для добавления

лидеры никогда не перезаписывают свои журналы

Соответствие журналов

идентичные записи в одном индексе во всех журналах

Полнота лидера

зафиксированные записи появляются в журналах будущих лидеров

Безопасность состояния машины

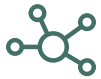
один и тот же индекс журнала = одна и та же команда

> Эти свойства математически доказуемы и обеспечивают корректность алгоритма даже в самых сложных сценариях сбоев.

Преимущества и применение Raft

etcd

хранилище конфигурации Kubernetes использует Raft для обеспечения согласованности



HashiCorp Consul

координация сервисной сетки с помощью консенсуса Raft

CockroachDB

использует Raft для обеспечения согласованности SQL в разных регионах

TiKV

распределенное хранилище ключей-значений с репликацией на основе Raft

Модификации Raft

- **Предварительное голосование**: предотвращает ненужные выборы из разбитых на разделы узлов
- **Совместный консенсус**: безопасные изменения членства без потери доступности
- **KRaft (Kafka)**: Apache Kafka заменяет ZooKeeper на Raft для метаданных
- **Multi-Raft**: запуск нескольких независимых групп Raft для горизонтального масштабирования

Paxos vs. Raft: выбор правильного алгоритма

Оба алгоритма решают одну и ту же фундаментальную проблему, но с разными философиями и компромиссами:

Аспект	Paxos	Raft
Обработка запросов	любой узел может принимать запросы клиентов и выступать в качестве инициатора	только лидер принимает запросы клиентов; последователи перенаправляют
Сложность реализации	Высокая, легко ошибиться	Умеренная, понятная структура
Производительность	Может страдать от конфликтов	Стабильная при нормальной работе
Теоретическая элегантность	Математически строгий	Практически ориентированный

Выбирайте Paxos, если

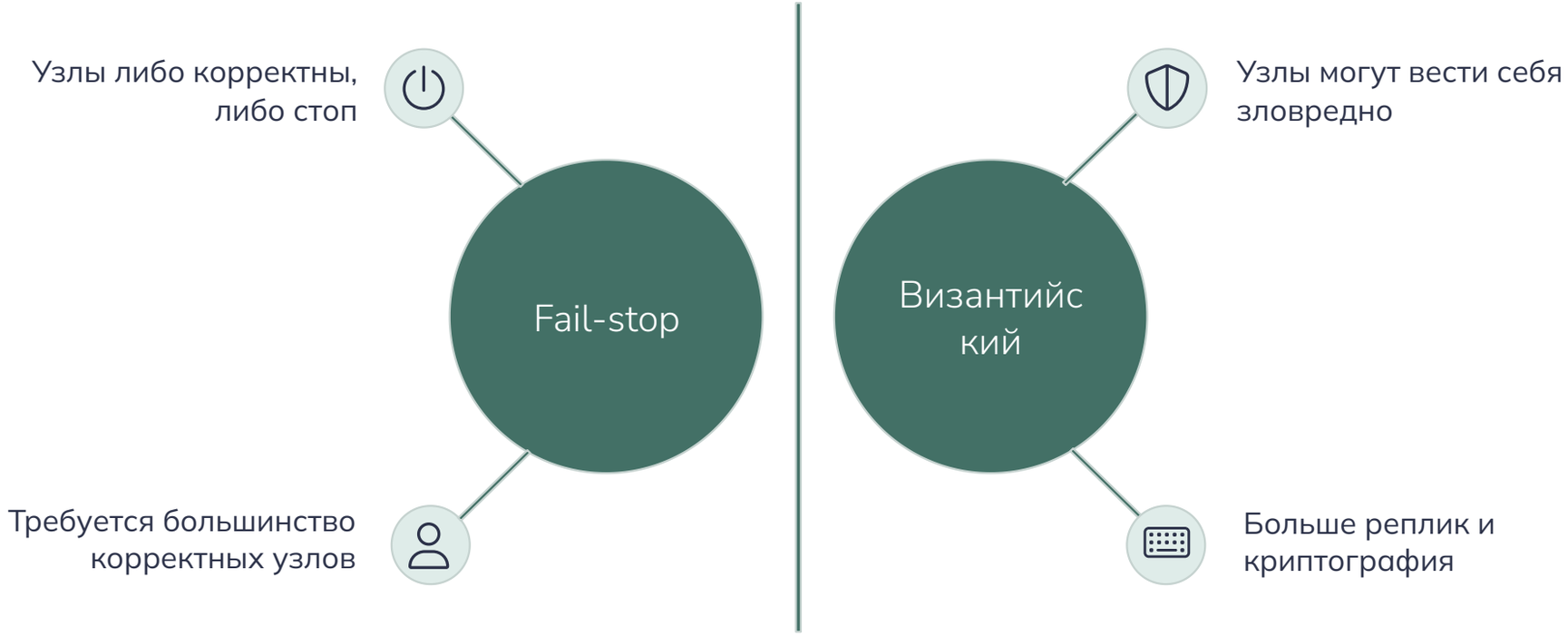
- Нужно наиболее общее решение для достижения консенсуса
- Есть сценарии с высокой степенью конфликтов
- Расширяете существующие системы на основе Paxos
- Требуется лучшее распределение нагрузки

Выбирайте Raft, если

- Придаете приоритет простоте реализации
- Важно понимание командой
- Нужна производительность разработки
- Создаете базы данных или хранилища конфигураций

Византийская отказоустойчивость: когда узлам нельзя доверять

И Paxos, и Raft предполагают, что узлы выходят из строя в результате сбоя (модель fail-stop) — они либо работают правильно, либо полностью перестают работать. Но что, если узлы могут вести себя злонамеренно, отправляя конфликтующие сообщения или повреждая данные? Это проблема византийской неисправности, названная в честь проблемы византийских генералов, в которой генералы могут быть предателями.



Когда важна византийская отказоустойчивость

Византийская отказоустойчивость становится критически важной в:

Сетях блокчейнов

где участники могут быть враждебными

Мультиорганизационных системах

Когда нельзя доверять всем участникам

Критически важная инфраструктура

где скомпрометированные узлы могут привести к катастрофическим сбоям

Распределенные реестры

где финансовые стимулы могут мотивировать злонамеренное поведение

В отличие от обычных сбоев, византийские узлы могут координировать атаки, отправлять противоречивые сообщения разным узлам и пытаться нарушить консенсус.

Алгоритмы византийского консенсуса

Алгоритмы византийского консенсуса разработаны для сред, где некоторые узлы могут действовать злонамеренно или быть скомпрометированы. Они требуют большего количества узлов и более сложных сообщений, чем алгоритмы с отказоустойчивостью при сбоях.

Традиционный византийский консенсус (сети с разрешенным доступом)

PBFT

Practical Byzantine Fault Tolerance был первым практическим алгоритмом византийского консенсуса. Он требует 3f+1 узлов для толерантности к f византийским сбоям (по сравнению с 2f+1 для сбоев при сбое) и использует трехфазный протокол с криптографической верификацией. Используется в Hyperledger Fabric, библиотеке BFT-SMaRt и академических исследовательских системах.

HotStuff

обеспечивает линейную сложность сообщений и лучшую производительность, чем PBFT. Используется в проекте Facebook Diem и некоторых корпоративных платформах.

Tendermint

основан на PBFT с немедленной окончательностью, оптимизирован для использования в блокчейне. Используется в Cosmos Hub, Terra, Binance Chain и многих блокчейнах Cosmos SDK.

Istanbul BFT

вариант, совместимый с Ethereum, используемый в частных сетях Ethereum и некоторых корпоративных блокчейн-платформах.

Консенсус в блокчейне (сети без разрешений)

Сети без разрешений сталкиваются с дополнительными проблемами, поскольку к ним может присоединиться любой, что требует разных подходов:

- **Правило самой длинной цепочки (Bitcoin)**: простой консенсус, при котором побеждает самая длинная действительная цепочка, с использованием Proof of Work для защиты от атак Сибилла и выбора лидера
- **Gasper (Ethereum 2.0)**: сочетает протоколы Casper FFG и LMD GHOST, используя Proof of Stake для выбора валидатора и экономической безопасности

Стоимость византийской толерантности

Византийские алгоритмы сопровождаются значительными накладными расходами:

- **Требуется больше узлов**: 3f+1 против 2f+1 для отказоустойчивости при сбоях
- **Более высокая сложность сообщений**: узлы должны проверять поведение друг друга
- **Криптографические накладные расходы**: цифровые подписи и проверка увеличивают задержку
- **Сложность реализации**: гораздо сложнее реализовать правильно, чем алгоритмы отказоустойчивости при сбоях

Для большинства традиционных распределенных систем стоимость византийской толерантности превышает преимущества, поскольку узлы обычно находятся в пределах одной зоны доверия (одна компания, один центр обработки данных и т. д.).

Будущее консенсуса

Современные распределенные системы продвигают алгоритмы консенсуса в новых направлениях:

Гибкий консенсус

протоколы, которые могут адаптировать свои гарантии согласованности в зависимости от потребностей приложения (EPaxos, варианты PBFT).

Интеграция блокчейна

адаптация классического консенсуса для платформ криптовалют и смарт-контрактов.

Геораспределенные системы

обработка консенсуса между несколькими центрами обработки данных с различными сетевыми условиями.

Фундаментальная проблема распределенного консенсуса остается актуальной как никогда. По мере того, как мы создаем все более сложные распределенные системы — от архитектур микросервисов до баз данных планетарного масштаба — понимание этих алгоритмов становится необходимым для любого инженера, работающего с распределенными системами.

> Выбор между Paxos и Raft часто зависит от приоритетов вашей команды: теоретическая строгость против практической реализации, гибкость против простоты, академическая элегантность против инженерной производительности.

Оба алгоритма имеют свое место в наборе инструментов распределенных систем, и понимание обоих сделает вас лучшим инженером распределенных систем.