# Отказоустойчивость

Характерной особенностью **распределенных систем**, которая отличает их от систем с одним компьютером, является **понятие частичного отказа**: часть системы выходит из строя, а оставшаяся часть продолжает работать, и, возможно, правильно. Важной целью при проектировании распределенных систем является создание системы таким образом, чтобы она могла автоматически восстанавливаться после частичных сбоев, не оказывая серьезного влияния на общую производительность. В частности, всякий раз, когда происходит сбой, система должна продолжать работать приемлемым образом во время ремонта. Другими словами, распределенная система должна быть отказоустойчивой.

В этой главе мы обращаем внимание на так называемые **протоколы распределенной фиксации**, по которым группа процессов выполняет **либо совместную фиксацию** своей локальной **работы**, либо **коллективно прерывает работу** и возвращает систему к предыдущему состоянию.

Важной проблемой при использовании групп процессов для допуска ошибок является то, сколько требуется репликаций. Чтобы упростить наше обсуждение, давайте рассмотрим только системы с реплицированной записью. Говорят, что система является **устойчивой к  k неисправностям** (k-fault tolerant), если она может преодолеть неисправности в k компонентах и при этом соответствовать спецификациям. Если компоненты, скажем процессы, молча терпят неудачу, то наличие k + 1 таких компонентов достаточно для обеспечения устойчивости к ошибкам. Если k из них просто останавливаются, то можно использовать ответ от другого.

ПРОСТО ЕСТЬ

Наиболее распространенные консенсусные протоколы, используемые в сетях блокчейнов, такие как **Proof-of-Work, Proof-of-Stake и Proof-of-Authority**, имеют некоторые свойства BFT.

**Некоторые методы решения задачи византийских генералов:**

* **Алгоритм Proof-of-Work (PoW)**. Его реализовал создатель биткоина Сатоши Накамото в 2008 году. PoW-алгоритм позволяет исключить элемент доверия и установить чёткие правила достижения консенсуса между узлами распределённой сети. [1](https://forklog.com/cryptorium/kak-raznye-protokoly-reshayut-zadachu-vizantijskih-generalov)
* **Алгоритм Proof of Stake (PoS)**. Управление сетью отдают владельцам нативной монеты. Главное преимущество такого подхода — незначительное потребление энергии в сравнении с PoW. [1](https://forklog.com/cryptorium/kak-raznye-protokoly-reshayut-zadachu-vizantijskih-generalov)
* **Алгоритм Byzantine Fault Tolerance (BFT)**. [1](https://forklog.com/cryptorium/kak-raznye-protokoly-reshayut-zadachu-vizantijskih-generalov)[4](https://tr-page.yandex.ru/translate?lang=en-ru&url=https%3A%2F%2Fwww.baeldung.com%2Fcs%2Fdistributed-systems-the-byzantine-generals-problem) В нём используется коллективное принятие решений в достижении консенсуса. Узлы отправляют транзакции друг другу, пока не придут к одинаковому результату. [1](https://forklog.com/cryptorium/kak-raznye-protokoly-reshayut-zadachu-vizantijskih-generalov)
* **Алгоритмы консенсуса Raft и Paxos**. Они основаны на сетевом паритете и обрабатывают разделение сети, когда возникают перебои в каналах связи между сторонами. [4](https://tr-page.yandex.ru/translate?lang=en-ru&url=https%3A%2F%2Fwww.baeldung.com%2Fcs%2Fdistributed-systems-the-byzantine-generals-problem)
* **Алгоритмы делегированного доказательства доли (DPoS), подтверждения полномочий (PoA) и HoneyBadger BFT**. [4](https://tr-page.yandex.ru/translate?lang=en-ru&url=https%3A%2F%2Fwww.baeldung.com%2Fcs%2Fdistributed-systems-the-byzantine-generals-problem)

Также существуют и другие варианты, облегчающие решение этой задачи, например, использование цифровых подписей или введение ограничений на связь между одноранговыми узлами. [5](https://habr.com/ru/companies/otus/articles/467053/)



# Лекции Крюкова

7. Обеспечение надежности в распределенных системах

Отказом системы называется поведение системы, не удовлетворяющее ее спецификациям. Последствия отказа могут быть различными.

**Отказы** по характеру своего проявления подразделяются на

* **византийские** (система активна и может проявлять себя по-разному, даже злонамеренно)
* **пропажа признаков жизни** (частичная или полная).

Первые распознать гораздо сложнее, чем вторые. Свое название они получили по имени Византийской империи (330-1453 гг.), где расцветали конспирация, интриги и обман

Для обеспечения надежного решения задач в условиях отказов системы применяются два принципиально различающихся подхода - **восстановление** решения **после отказа** системы (или ее компонента) и **предотвращение отказа** системы (отказоустойчивость).

7.1. Восстановление после отказа.

* Прямое восстановление основано на своевременном обнаружении сбоя и ликвидации его последствий путем приведения некорректного состояния системы в корректное. Такое восстановление возможно только для определенного набора заранее предусмотренных сбоев.
* При возвратном восстановлении происходит возврат процесса (или системы) из некорректного состояния в некоторое из предшествующих корректных состояний. При этом возникают следующие проблемы.

7.2. Отказоустойчивость.

Изложенные выше методы **восстановления после отказов** для некоторых систем **непригодны** (управляющие системы, транзакции в on-line режиме) из-за прерывания нормального функционирования.

Чтобы избежать эти неприятности, создают **системы, устойчивые к отказам**. Такие системы

* либо маскируют отказы,
* либо ведут себя в случае отказа заранее определенным образом (пример - изменения, вносимые транзакцией в базу данных, становятся невидимыми при отказе).

Два механизма широко используются при обеспечении отказоустойчивости –

* **протоколы голосования.**

для маскирования отказов (выбирается правильный результат, полученный всеми исправными исполнителями).

* протоколы принятия коллективного решения.
  + **протоколы принятия единого решения**, в которых все исполнители являются исправными и должны либо все принять, либо все не принять заранее предусмотренное решение. Примерами такого решения являются решение о завершении итерационного цикла при достижении всеми необходимой точности, решение о реакции на отказ
  + **протоколы принятия согласованных решений на основе полученных друг от друга данных**. При этом необходимо всем исправным исполнителям получить достоверные данные от остальных исправных исполнителей, а данные от неисправных исполнителей проигнорировать.

**протоколы принятия согласованных решений на основе полученных друг от друга данных**.

Согласие в данном случае заключается в следующем. Каждый генерал знает, сколько воинов находится под его командой. Ставится цель, чтобы все лояльные генералы узнали численности всех лояльных армий, т.е. каждый из них получил один и тот же вектор длины n, в котором i-ый элемент либо содержит численность i-ой армии (если ее командир лоялен) либо не определен (если командир предатель).

Соответствующий рекурсивный алгоритм был предложен в 1982 г. (Lamport).

Lamport доказал, что в системе с m неверно работающими процессорами можно достичь согласия только при наличии 2m+1 верно работающих процессоров (более 2/3).

Другие авторы доказали, что в распределенной системе с асинхронными процессорами и неограниченными коммуникационными задержками согласие невозможно достичь даже при одном неработающем процессоре (даже если он не подает признаков жизни).

Применение алгоритма - надежная синхронизация часов.

**Алгоритм надежных неделимых широковещательных рассылок сообщений.**

# Проблема византийских генералов ЛЕСЛИ ЛАМПОРТ, РОБЕРТ ШОСТАК и МАРШАЛ ПИЗ СRI International

Надежные компьютерные системы должны справляться с неисправными компонентами, которые дают противоречивую информацию различным частям системы.

Эта ситуация может быть абстрактно выражена в терминах **группы генералов византийской армии**, расположившихся со своими войсками вокруг вражеского города. Общаясь только через посланников, генералы должны согласовать общий план битвы. Однако один или несколько из них могут быть предателями, которые попытаются запутать остальных.

**Проблема** заключается в том, чтобы найти алгоритм, который обеспечит достижение согласия среди лояльных генералов.

* Показано, что, используя только **устные сообщения**, эта проблема решаема тогда и только тогда, когда более двух третей генералов лояльны; таким образом, один предатель может сбить с толку двух лояльных генералов.
* С использованием **не подделываемых письменных сообщений** проблема решаема для любого количества генералов и возможных предателей. Затем обсуждаются применения решений к надежным компьютерным системам.

# Practical Byzantine Fault Tolerance and Proactive Recovery

Наша растущая зависимость от онлайн-сервисов, доступных в Интернете, требует наличия высоконадежных систем, которые обеспечивают корректное обслуживание без сбоев. Ошибки программного обеспечения, ошибки оператора и

вредоносные атаки являются основной причиной сбоев в обслуживании и могут привести к произвольному поведению, то есть к ошибочным действиям. В этой статье описывается новый алгоритм репликации, BFT, который может быть

использован для создания высокодоступных систем, устойчивых к сбоям в работе. BFT можно использовать на практике

для реализации реальных сервисов: он хорошо работает и безопасен в асинхронных средах, таких как

Интернет, он включает в себя механизмы защиты от клиентов, подверженных византийским ошибкам, и

активно восстанавливает копии. Механизм восстановления позволяет алгоритму выдерживать любое количество сбоев

в течение срока службы системы при условии, что менее 1/3 реплик становятся неисправными в течение небольшого

периода уязвимости.

Возможные сбои, такие

как ошибки программного обеспечения, ошибки оператора и вредоносные атаки, являются основной причиной

сбоев в обслуживании. Мы представляем новый алгоритм репликации и методы реализации для создания высокодоступных систем, которые допускают

возможные сбои и могут использоваться на практике.

В этой статье описывается BFT, алгоритм репликации на автомате состояний, который допускает ошибки Byzantine при условии, что менее 1/3 реплик являются ошибочными. BFT

обеспечивает линеаризуемость, которая является важным свойством безопасности, не полагаясь на

любое допущение о синхронности. Кроме того, это гарантирует работоспособность при условии, что задержки сообщений в конечном итоге будут ограничены. BFT обеспечивает безопасность и работоспособность независимо от количества клиентов с ошибками Byzantine.

# Распределенные системы таненбаум

**Маскировка отказов посредством избыточности**

Если система должна быть отказоустойчивой, лучшее, что она может сделать, – это попытаться скрыть возникновение сбоев от других процессов. Ключевой метод маскировки неисправностей заключается в использовании избыточности.

Возможны три вида: информационная избыточность и физическая избыточность (см. также [Джонсон, 1995]).

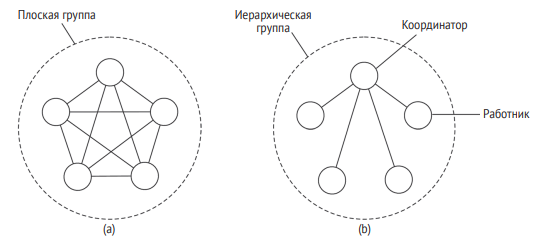
* **При информацинной избыточности** (information redundancy) добавляются дополнительные биты, чтобы обеспечить восстановление искаженных битов. Например, код Хемминга может быть добавлен к  передаваемым данным для восстановления после шума в линии передачи.
* **При временнóй избыточности** (time redundancy) выполняется действие, а затем, если необходимо, его выполнение повторяется. Этот подход используют транзакции. Если транзакция прерывается, она может быть передана без вреда. Другой известный пример – повторная передача запроса на сервер при отсутствии ожидаемого ответа. Временнáя избыточность особенно полезна, когда неисправности являются временными или прерывистыми.
* **При физической избыточности** (резервировании) (physical redundancy) добавляется дополнительное оборудование или процессы, чтобы система в целом могла допускать потерю или неисправность некоторых компонентов. Таким образом, физическое резервирование может быть выполнено либо в аппаратном, либо в программном обеспечении. Например, в систему можно добавить дополнительные процессы, чтобы в случае сбоя небольшого

**Устойчивость групповых процессов**

Ключевой подход к сохранению работоспособности при наличии ошибочного процесса состоит в том, чтобы объединить несколько идентичных процессов в группу

Организация групп

* Каждая из этих организаций имеет свои преимущества и недостатки. Плоская группа симметрична и не имеет единой точки отказа. Если происходит сбой одного из процессов, группа просто становится меньше, но в противном случае может продолжать существовать. Недостатком является то, что принятие решений является более сложным. Например, чтобы что-томрешить, часто приходится проводить голосование, что приводит к некоторым затратам.
* Иерархическая группа обладает противоположными свойствами. Потеря координатора останавливает всю группу, но пока она работает, она может принимать решения, не затрагивая всех остальных. На практике, когда координатор в иерархической группе отказывает, его роль должна быть передана, и один из рабочих избирается новым координатором. Мы обсудили алгоритмы выбора лидера в главе 6.



Частью решения для построения отказоустойчивых систем являются группы процессов.

В частности, наличие группы идентичных процессов позволяет нам маскировать один или несколько неисправных процессов в этой группе. Cуществует два способа приблизиться к  такой репликации:

* с  помощью первичных протоколов

Первичная репликация в случае отказоустойчивости обычно отображается в форме протокола первичного резервного копирования. в  котором первичный координирует все операции записи

* протоколов репликации и записи.

Эти решения соответствуют организации набора идентичных процессов в плоскую группу. Основным преимуществом является то, что такие группы не имеют единой точки отказа за счет распределенной координации.

Важной проблемой при использовании групп процессов для допуска ошибок является то, **сколько требуется репликаций**

С другой стороны, если процессы демонстрируют произвольные отказы, продолжая работать при отказе и отправляя ошибочные или случайные ответы, необходимо минимум 2k + 1 процессов для достижения устойчивости к отказам.

**Консенсус в неисправных системах со сбоями**

Как уже упоминалось, мы приняли модель относительно клиентов и серверов, в  которой потенциально очень большое количество клиентов теперь отправляют команды группе процессов, которые совместно ведут себя как единый, очень надежный процесс. Чтобы выполнять эту работу, нам необходимо сделать важное предположение:

*В отказоустойчивой группе процессов каждый исправный процесс выполняет те же команды и в том же порядке, что и любой другой исправный процесс.*

Формально это означает, что члены группы должны прийти к **единому мнению**, какую команду выполнять. Если отказы не могут произойти, достичь консенсуса легко. Например, мы можем использовать **полностью упорядоченную многоадресную рассылку Лампорта** (Lamport), как описано в разделе 6.2.

Мы можем достичь консенсуса с  помощью **адаптированного подхода**, используемого в [Cachin et al., 2011] и относящегося к **консенсусу лавинной маршрутизации** (flooding consensus).

* « не очень реалистичен хотя бы потому, что опирается на модель сбоя аварийного останова »

**Пример: Paxos**

Более реалистичным является предположение об отказоустойчивой модели сбоев, в которой процесс в конечном итоге надежно обнаружит сбой другого процесса. Далее мы опишем упрощенную версию широко принятого консенсусного алгоритма, известного как Paxos. Первоначально он был опубликован в 1989 году в качестве технического доклада Лесли Лэмпорта (Leslie Lamport), но прошло около десяти лет, прежде чем было решено, что, может быть, не такая плохая идея – разъяснить и распространять его [Lamport, 1998]. Первоначальную публикацию нелегко понять, примером могут служить другие публикации, нацеленные на ее объяснение [Lampson, 1996; Prisco et.al., 1997; Lamport, 2001; van Renesse and Altinbuken, 2015].

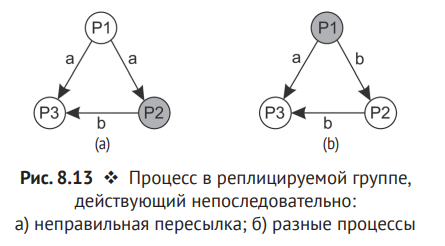
**Основная идея Paxos**

. . .

Обратите внимание, что опора на ведущего заявителя подразумевает, что любая практическая реализация Paxos должна сопровождаться алгоритмом выбора лидера. В принципе, этот алгоритм может работать независимо от Paxos, но обычно является его частью.

До сих пор мы предполагали, что при аварийном отказе затрагивались только реплики, и в этом случае группа процессов должна состоять из 2k + 1 серверов, чтобы выдержать k аварийных участников. Важным предположением в этих случаях является то, что **процесс не вступает в договоренность с другим процессом** или, более конкретно, **является последовательным в  своих сообщениях другим**

**Консенсус в неисправных системах с произвольными отказами**



Чтобы достичь так называемого византийского соглашения (Byzantine agreement, ВА), нам необходимо выполнить следующие два требования:

1) BA1: каждый исправный процесс резервного копирования хранит одно и то же значение;

2) BA2: если первичный процесс не является неисправным, то каждый исправный процесс резервного копирования сохраняет именно то, что отправил основной.

Почему 3k процессов недостаточно . . .

Почему достаточно 3k + 1 процессов . . .

**Пример: практическая византийская отказоустойчивость**

Примерно в 2000 году Барбаре Лисков (Barbara Liskov) и в то время ее ученице Мигеле Кастро

(Miguel Castro) удалось придумать **практическую реализацию протокола для**

**репликации серверов, способных обрабатывать произвольные с**бои. Давайте кратко рассмотрим их решение, которое было названо практическая византийская отказоустойчивость (Practical Byzantine Fault Tolerance, **PBFT**) [Castro and Liskov, 2002]).

Как и Paxos, PBFT делает только несколько предположений о своей среде. Он не делает никаких предположений о поведении серверов реплики: предполагается, что неисправный сервер демонстрирует произвольное поведение. Кроме того, сообщения могут быть потеряны, задержаны и получены не по порядку. Однако предполагается, что отправитель сообщения может быть идентифицирован

. . . Эти и другие недостатки наряду с присущей византийской отказоустойчивостью и сложностью стали препятствием для его широкого использования.

**Некоторые ограничения по реализации отказоустойчивости**

Существует три требования для достижения консенсуса [Fischer et al., 1985]:

1) процессы выдают одно и то же выходное значение;

2) каждое выходное значение должно быть действительным;

3) каждый процесс должен в конечном итоге обеспечивать вывод.

# Обработка ошибочных ситуаций в больших блокчейн-сетях алгоритмом достижения консенсуса, основанном на решении задачи византийских генералов

Блокчейн-сети, построенные на основе алгоритмов **Роз, DPoS, LPoS, PoE, PoIT, pBFT**, имеют определенные ограничения, связанные со снижением скорости включения новых транзакций в блокчейн при увеличении числа узлов блокчейн-сетей, принимающих участие в создании блока.

Разработанный алгоритм достижения консенсуса sdBFT позволяет по сравнению с существующими алгоритмами BFT увеличить на несколько порядков число узлов сети, участвующих в достижении консенсуса, сохраняя при этом скорость включения транзакций в блокчейн.

sdBFT – 7 ошибочных ситуаций

За последние несколько лет разработаны алгоритмы консенсуса, устраняющие **слабые стороны** алгоритма доказательства работы **PoW** (Proof-of-Work) [1]: PoS, DPoS, LPoS, PoE, PoIT, pBFT [2-4].

Новый подход к решению проблемы производительности в алгоритме pBFT (practical Byzantine Fault Tolerant) реализован в алгоритме консенсуса **sdBFT** (stake distributed ВЕТ),

Сравнение Honey Badger of BFT и sdBFT

# The Honey Badger of BFT Protocols

Таким образом, достижимая пропускная способность PBFT уменьшается с увеличением количества узлов, в то время как Показатель HoneyBadgerBFT остается примерно неизменным.

Мы представили HoneyBadgerBFT, первый эффективный и высокопроизводительный асинхронный протокол BFT. Благодаря нашей реализации и экспериментальным результатам мы демонстрируем, что HoneyBadgerBFT может стать подходящим компонентом для начинающих внедрений отказоустойчивых систем обработки транзакций, основанных на криптовалютах. В целом, мы считаем, что наша работа демонстрирует перспективность создания надежных систем обработки транзакций на основе асинхронного протокола

**PBFT**. Протокол PBFT состоит из двух основных рабочих процессов: “быстрого пути”, который обеспечивает хорошую производительность в оптимистичном случае (когда сеть синхронна и лидер функционирует корректно), и процедуры “изменения представления” для смены лидеров.

# Internet Computer Consensus ICC0 ICC1 ICC2

Мы представляем семейство протоколов Internet Computer Consensus (ICC) для атомарной широковещательной передачи (также известных как консенсус), которые лежат в основе отказоустойчивых автоматов с реплицируемыми состояниями Byzantine Internet Computer. Протоколы ICC основаны на лидерах , которые предполагают частичную синхронизацию и полностью интегрированы с блокчейном. Лидер, вероятно, меняется в каждом раунде. Эти протоколы чрезвычайно просты и надежны: в любом раунде, где лидер коррумпирован (что само по себе происходит с вероятностью менее 1/3), каждый протокол ICC фактически позволит другой стороне взять верх в качестве лидера этого раунда, без особого шума, своевременно продвинуть протокол к следующему раунду. В отличие от многих других протоколов, здесь нет сложных подпротоколов (таких как “изменение вида” в PBFT) или неопределенных подпротоколов (таких как “кардиостимулятор” в HotStuff). Более того, в отличие от многих других протоколов (таких как PBFT и HotStuff), задача надежного распространения блоков среди всех сторон является неотъемлемой частью протокола и не возлагается на какой-либо другой неуказанный подпротокол. Дополнительное свойство, которым обладают протоколы ICC (точно так же, как PBFT и HotStuff, и в отличие от других, например, Tendermint) –

это оптимистичная скорость отклика, которая означает, что, когда руководитель честен, протокол будет работать со скоростью фактической сетевой задержки, а не с некоторой верхней границей сетевой задержки.

Мы представляем три различных протокола (вместе с различными незначительными вариациями каждого из них). Один из этих протоколов (ICC1) предназначен для интеграции с одноранговым подуровнем gossip, что уменьшает узкое место, созданное в лидере для распространения больших блоков, - проблему, с которой сталкиваются все протоколы, основанные на лидере, такие как PBFT и HotStuff должны устранять, но обычно этого не делают. Наш протокол ICC2 решает ту же проблему, заменяя подпротокол широковещательной передачи с низким уровнем надежности связи (который может представлять самостоятельный интерес) на подуровень gossip.

**PBFT** проходит в несколько раундов. В каждом раунде назначенный руководитель предлагает набор команд, передавая их по радио всем участникам. Затем следует два этапа обмена информацией между всеми участниками для фактического выполнения пакета. При нормальной работе ведущий будет продолжать выполнять свою роль в течение многих раундов. Однако, если достаточное количество участников определит, что протокол не выполняется своевременно, они инициируют операцию по изменению представления, в результате которой будет назначен новый ведущий и устранен любой беспорядок, оставленный старым лидером

Как и в случае с PBFT, **HotStuff** полагается на лидера для распространения блоков (т.е. пакетов), и, как и в случае с PBFT, это может стать узким местом в коммуникации, и не существует явного механизма, гарантирующего надежное распространение блоков в случае повреждения лидера. Кроме того, хотя HotStuff не использует подпротокол “изменение вида”, он по-прежнему использует так называемый подпротокол “кардиостимулятор”

ICC:

Живучесть при интервальной синхронности Как уже говорилось, лемма 6 гарантирует живучесть в течение данного раунда, если сеть является δ-синхронной в течение короткого промежутка времени, начиная с момента, когда первая честная партия вступает в этот раунд. Было бы лучше иметь возможность сказать, что всякий раз, когда сеть остается δ-синхронной в течение достаточно длительного промежутка времени, ее работоспособность будет сохраняться в течение любого цикла, достигнутого протоколом. Мы можем получить такой результат, объединив лемму 6 с Леммой 8 о задержке:

# И. М. Никольский РАСПРЕДЕЛЕННАЯ ОБРАБОТКА ДАННЫХ

2.3 Причины построения распределённых систем

Переход от локальной системы (располагающейся на одном компьютере) к распределённой — трудоёмкий и дорогостоящий процесс. В каких случаях это необходимо? Здесь

возможны разные варианты:

 необходимость в обработке больших объёмов данных;

 обслуживание географически распределённых пользователей;

 распределение нагрузки;

 увеличение производительности;

 повышение отказоустойчивости;

 предоставление совместного доступа к некоторому общему ресурсу.

Кроме того, как показывают исследования, горизонтальное масштабирование

вычислительной системы (переход от системы с одним узлом к кластеру) может быть

экономически выгоднее вертикальной масштабируемости (увеличение мощности

имеющегося сервера).

византийские отказы — непредсказуемое поведение узлов; такую ошибку невозможно воспроизвести — разные узлы получают разные ответы на идентичные запросы;

стр 6 просто зачем все это

В распределенных системах одной из основных проблем является проблема взаимодействия одновременно работающих узлов между собой и принятия единого решения

В распределенных системах проблема взаимодействия одновременно работающих ее элементов между собой является одной

В распределенных системах есть необходимость организации взаимодействия ее одновременно работающих элементов между собой и принятия ими единого решения, то есть достижения консенсуса, даже ситуации, когда некоторые узлы системы начинают работать неверно, злонамеренно или в случае поломки.

И возможна ситуация, когда некоторые узлы системы начинают работать неверно, злонамеренно или в случае поломки,

одной из основных проблем является

, которая в может быть сведена к задаче достижения консенсуса, то есть принятия единого решения. Рассматриваемая в данной работе задача византийских генералов описывает ситуацию

# algoritmy-dostizheniya-konsensusa-v-blokcheyn-sisteme.pdf

Разновидности алгоритмов достижения консенсуса Вне зависимости от логики алгоритмов консенсуса все они основаны на реализации основных правил взаимодействия пользователей, таких как

* достижение наибольшей степени согласия всех взаимодействующих сторон,
* соблюдение равноправия участников,
* заинтересованность участников в совместной работе и
* включение в процесс достижения консенсуса максимального числа участников [7].

# Анализ блокчейн-технологии: основы архитектуры, примеры использования, перспективы развития, пр

Протокол доказательства выполнения работы

* Proof of Work (PoW, доказательство выполнения работы)

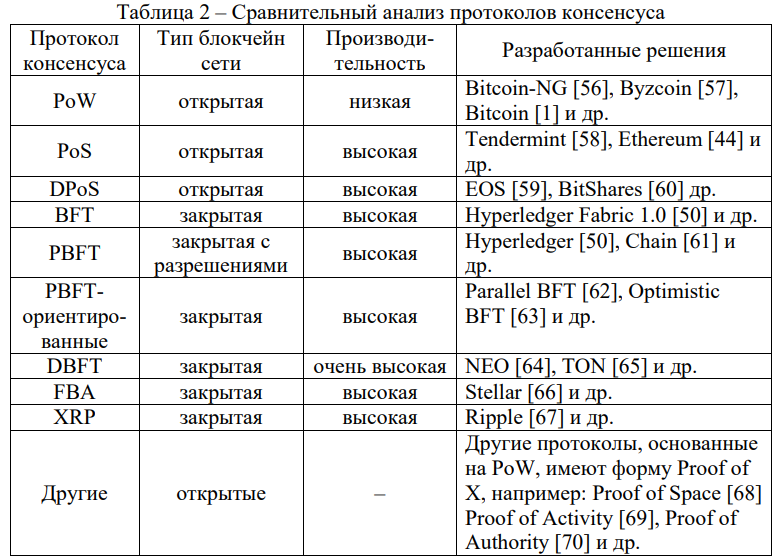
Протоколы доказательства владения долей

* Proof of Stake (PoS) – доказательство владения долей, является более эффективной альтернативой PoW
* Delegated Proof of Stake (DPoS)

они обладают низкой пропускной способностью.

BFT-ориентированные протоколы

* Byzantine Fault Tolerance (PBFT), в которых анонимность не является важным аспектом, то есть узлы знают некую информацию о друг друге (изначально узлы аутентифицированы). Благодаря этому можно оптимизировать алгоритмы консенсуса и достичь намного большей пропускной способности. Фактически скорость может увеличиться в 10 раз, от сотен до тыс
* Practical Byzantine Fault Tolerance (PBFT) Если 2/3 участников проголосуют за эту транзакцию как корректную, валидатор принимает её и передаёт своё решение в блокчейн сеть другим валидаторам.
* Delegated Byzantine Fault Tolerance (DBFT) Но на фоне скорости присутствует вероятность централизации. Это связано с тем, что большая половина узлов может быть сосредоточена в одних руках
* Federated Byzantine Agreement (FBA) FBA не требует разрешения или заранее известного набора участников, в отличие от PBFT и других вариаций BFT



3.1. Типовые проблемы блокчейн-технологии

3.1.1. Уязвимость «51%»

<file:///C:/Users/ck4te/Downloads/978-5-19-011913-8_e-book.pdf>

В рамках данного пособия блокчейн интересует нас поскольку является по сути первой успешной попыткой построения децентрализованной СУБД планетарного масштаба,

Наша растущая зависимость от онлайн-сервисов, доступных в Интернете, требует наличия высоконадежных систем, которые обеспечивают корректное обслуживание без сбоев. Ошибки программного обеспечения, ошибки оператора и вредоносные атаки являются основной причиной сбоев в обслуживании и могут привести к произвольному поведению, то есть к ошибочным действиям. В этой статье описывается новый алгоритм репликации, BFT, который может быть использован для создания высокодоступных систем, устойчивых к сбоям в работе. BFT можно использовать на практике для реализации реальных сервисов: он хорошо работает и безопасен в асинхронных средах, таких как Интернет, он включает в себя механизмы защиты от клиентов, подверженных византийским ошибкам, и активно восстанавливает копии. Механизм восстановления позволяет алгоритму выдерживать любое количество сбоев в течение срока службы системы при условии, что менее 1/3 реплик становятся неисправными в течение небольшого периода уязвимости.

использование симметричной криптографии для аутентификации сообщений.

BFT - это первый византийский отказоустойчивый алгоритм **репликации на автомате состояний**, который безопасен **в асинхронных системах, таких как Интернет**: он не полагается на какие-либо предположения о синхронности для обеспечения безопасности. В частности, он никогда не возвращает неверные ответы даже при наличии атак типа "отказ в обслуживании". Кроме того, он гарантирует работоспособность при условии, что задержки сообщений в конечном итоге будут ограничены. Служба может быть не в состоянии возвращать ответы, когда активна атака типа "отказ в обслуживании ", но клиенты гарантированно получат ответы по окончании атаки

BFT включает в себя ряд важных **оптимизаций**, которые позволяют алгоритму работать эффективно, чтобы его можно было использовать на практике. Наиболее важной оптимизацией является **использование симметричной криптографии для аутентификации сообщений**. Криптография с открытым ключом, которая была названа основной причиной задержки [Reiter 1994] и пропускной способности [Malkhi and Reiter 1996a] в предыдущих системах, использовалась только для обмена симметричными ключами. Другие оптимизации снижают затраты на передачу данных: алгоритм **использует только один цикл передачи сообщений в оба конца для выполнения операций только для чтения и два для выполнения операций чтения-записи, а также он использует пакетную обработку данных под нагрузкой, чтобы снизить нагрузку на протокол при выполнении операций чтения-записи по множеству запросов.** Алгоритм также использует оптимизацию для снижения нагрузки на протокол по мере увеличения размера аргумента операции и результата. Кроме того, в статье описываются эффективные методы сбора протокольной информации и передачи состояния для обновления реплик; это необходимо для создания практических сервисов, которые допускают византийские ошибки

Описани елагоритма

Сервис моделируется как конечный автомат, который реплицируется на разных узлах распределенной системы. Каждая реплика поддерживает состояние сервиса и реализует сервисные операции. Клиенты отправляют запросы на выполнение операций репликам, а BFT гарантирует, что все исправные реплики выполняют одни и те же операции в одинаковом порядке. Поскольку реплики являются детерминированными и запускаются в одном и том же состоянии, все исправные реплики отправляют ответы с идентичными результаты для каждой операции. Клиент ожидает ответов f + 1 от разных реплик с одинаковым результатом. Поскольку по крайней мере одна из этих реплик исправна, это правильный результат операции

**Задержка**. Мы измерили задержку при запуске операции, когда к службе обращается один клиент.

**Пропускная способность**. В этом разделе представлены результаты экспериментов по измерению пропускной способности BFT и NO-REP в зависимости от количества клиентов , обращающихся к сервису simple. Клиентские процессы были равномерно распределены по пяти клиентским компьютерам.2 Мы измерили пропускную способность для операций с различными размерами аргумента и результата. Каждый тип операции обозначается как a/b, где a и b - размеры аргумента, а результат - в килобайтах.

Configurations with More Replicas

Результаты показывают, что замедление, вызванное увеличением числа реплик до семи, незначительно. Максимальное замедление составляет 30% для операций чтения-записи и 26% для операций только для чтения. Кроме того, замедление быстро уменьшается по мере увеличения размера аргумента или результата. Например, при размере аргумента 8 КБАЙТ замедление составляет всего 7% при выполнении операций чтения-записи и 2% при оптимизации только для чтения. Замедление уменьшается по мере увеличения значения аргумента. размер увеличивается, поскольку накладные расходы, связанные с добавлением реплик, не зависят от этого размера. Оптимизация дайджеста ответов делает накладные расходы, связанные с добавлением реплик, независимыми от размера результата, что объясняет, почему замедление также уменьшается с увеличением размера результата