1. Эволюция программных систем – последовательные, параллельные и распределённые вычисления. Сравнение подходов, основные преимущества и недостатки. Известные классификации распределённых систем.

Параллельные вычисления — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно).

Существуют различные способы реализации параллельных вычислений. Например, каждый вычислительный процесс может быть реализован в виде процесса операционной системы, либо же вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС.

Основная сложность при проектировании параллельных программ — обеспечить правильную последовательность взаимодействий между различными вычислительными процессами, а также координацию ресурсов, разделяемых между процессами.

Распределённые вычисления — способ решения трудоёмких вычислительных задач с использованием нескольких компьютеров, чаще всего объединённых в параллельную вычислительную систему. Распределённые вычисления применимы также в распределённых системах управления.

Последовательные вычисления — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих последовательно, т.е друг за другом.

Известные классификации распределённых систем:

- По способам распределения данных
- По типу распределения процессоров
- Распределение по функциям

2. Параллельная обработка конкурентных задач. Примитивы параллелизма, проблемы параллельной обработки.

Примитивы синхронизации:

- Семафор (semaphore) – инициализируется максимальным количеством потоков, которым одновременно может быть доступен ресурс.
- Мьютекс (mutex) – частный случай семафора, с максимальным количеством потоков, равным 1. Разблокирован может быть только из того же потока, в котором был заблокирован.
- Барьер (barrier) – создаётся на некоторое количество потоков. Когда один поток заканчивает выполнение, он остаётся ждать «у барьера» окончания работы остальных потоков.
- Условная переменная (condition variable) – обеспечивает блокирование одного или нескольких потоков до момента поступления сигнала от другого потока о выполнении некоторого условия или до истечения максимального промежутка времени.

Проблемы синхронизации:

- Starvation (голодание) – описывает ситуацию, когда поток не может получить доступ к совместно используемому ресурсу, что не позволяет продвинуться ему дальше в своём выполнении. Возникает из-за «жадных» потоков, занимающих совместный ресурс на долгое время.
- Race condition (состояние гонки) – ошибка разработки многопоточного ПО, при которой работа ПО зависит от порядка выполнения частей кода. Возникает, когда несколько потоков пытаются получить доступ к совместно используемым данным, причем хотя бы один поток выполняет запись. Часть кода, которая не должна выполняться на нескольких потоках одновременно, так как может привести к состоянию гонки, называется критической секцией (critical section).
- Deadlock (взаимная блокировка) – описывает ситуацию, когда два или более потока блокируются навсегда, ожидая освобождения ресурсов, занятых самими этими потоками.
- Livelock (активная блокировка) – ситуация взаимной блокировки, когда два и более процессов зацикливаются, непрерывно меняя своё состояние в ответ на изменения в другом процессе, не производя полезной работы.

- Priority inversion (инверсия приоритетов) – . Бывают ограниченная (два потока) и неограниченная (несколько потоков) инверсия приоритетов. Неограниченную инверсию приоритетов можно избежать, используя механизмы наследование приоритета (priority inheritance) или потолок приоритета (priority ceiling).

Наследование приоритета – в момент, когда высокоприоритетный поток хочет захватить ресурс, занятый низкоприоритетным потоком, низкоприоритетному потоку повышается приоритет до приоритета первого потока.

Поток приоритета – ресурсу назначается свойство – максимальный приоритет из всех потока, которые могут его заблокировать. В момент времени, когда поток пытается захватить ресурс, занятый менее приоритетным потоком, приоритет менее приоритетного потока повышается до значения, назначенного ресурсу свойства.

3. Подходы к параллельной обработке задач (Блокировки, CSP, акторы, транзакционная память, immutable data).

Блокировки – см. предыдуший билет.

Communicating Sequential Processes. Является формальным математическим языком, позволяющим описывать взаимодействие параллельных систем. Основан на передачи сообщений по каналам.

Модель акторов. Актор — это некая сущность, обладающая поведением. Актор в данной модели взаимодействует путём передачи сообщений с другими акторами, в ответ на получаемые сообщения может принимать локальные решения, создавать новые акторы, посылать свои сообщения, устанавливать, как следует реагировать на последующие сообщения. Акторы изолированы друг от друга и имеют внутренее состояние которое не может быть изменено извне.

Транзакционная память. Группы инструкций выделяются в атомарные транзакции. Конкурентные потоки работают параллельно, пока не начинают модифицировать один и тот же участок памяти. Если происходит конфликт данных, транзакция отменяется. После этого транзакция обычно перезапускается, либо вызывается функция, заранее указанная как «запасной выход», чаще всего откат на использование блокировок.

Immutable data – подход к написанию кода, при котором состояние любого созданного объекта не может быть изменено после создания. Результатом модификации всегда будет создание нового экземпляра объекта (без изменения старого).

4. Распределённые системы - причины появления, типовые решаемые задачи. Обзор базовых технологий, используемых при проектировании распределенных систем.

Распределенная система - система, в которой обработка информации сосредоточена не на одной вычислительной машине, а распределена между несколькими компьютерами. При проектировании распределенных систем, которое имеет много общего с проектированием ПО в общем, все же следует учитывать некоторые специфические особенности. Архитектуру применяют с целью взаимодействия с системой через интернет и другие устройства, независимо от географического положения.

Существует шесть основных характеристик распределенных систем.

1. *Совместное использование ресурсов.*
2. *Открытость.* Это возможность расширения системы путем добавления новых ресурсов.
3. *Параллельность.*
4. *Масштабируемость.* Под масштабируемостью понимается возможность добавления новых свойств и методов.
5. *Отказоустойчивость.* Наличие нескольких компьютеров позволяет дублирование информации и устойчивость к некоторым аппаратным и программным ошибкам.
6. *Прозрачность.* Пользователям предоставляется полный доступ к ресурсам в системе, в то же время от них скрыта информация о распределении ресурсов по системе.

Распределенные системы обладают и рядом недостатков.

1. *Сложность*. Намного труднее понять и оценить свойства распределенных систем в целом, их сложнее проектировать, тестировать и обслуживать. Также производительность системы зависит от скорости работы сети, а не отдельных процессоров. Перераспределение ресурсов может существенно изменить скорость работы системы.

2. *Безопасность*. Обычно доступ к системе можно получить с нескольких разных машин, сообщения в сети могут просматриваться и перехватываться. Поэтому в распределенной системе намного труднее поддерживать безопасность.
3. *Управляемость*. Система может состоять из разнотипных компьютеров, на которых могут быть установлены различные версии операционных систем. Ошибки на одной машине могут распространиться непредсказуемым образом на другие машины.
4. *Непредсказуемость*. Реакция распределенных систем на некоторые события непредсказуема и зависит от полной загрузки системы, ее организации и сетевой нагрузки.

Из этих недостатков можно увидеть, что при проектировании распределенных систем возникает ряд проблем, которые надо учитывать разработчикам.

5. Средства синхронизации и транзакционность, CAP теорема.

Для синхронизации алгоритмов работы и данных в распределенных системах применяют различные алгоритмы.

Алгоритм пекарни Лампорта алгоритм разделения общих ресурсов между несколькими потоками путём взаимного исключения

Лампорт предлагает рассмотреть пекарню с устройством, выдающим номерки у входа. Каждому входящему покупателю выдаётся номерок на единицу больше предыдущего. Общий счётчик показывает номер обслуживаемого в данный момент клиента. Все остальные покупатели ждут, пока не закончат обслуживать текущего клиента и табло покажет следующий номер. После того, как клиент сделает покупку и сдаст свой номерок, служащий увеличивает на единицу допустимые для выдачи устройством у входа номера. Если совершивший покупку клиент захочет снова что-нибудь купить, он должен будет снова взять номерок у входа и встать в общую очередь.

Token Ring. Станции в локальной вычислительной сети Token Ring логически организованы в кольцевую топологию, с данными, передаваемыми последовательно от одной станции в кольце к другой. Token Ring использует специальный трёхбайтовый блок данных, называемый маркером, который так же перемещается по кольцу. Владение маркером предоставляет его обладателю право передавать данные.

Централизованный алгоритм. Все процессы запрашивают у координатора разрешение на вход в критическую секцию и ждут этого разрешения. Координатор обслуживает запросы в порядке поступления. Получив разрешение процесс входит в критическую секцию. При выходе из нее он сообщает об этом координатору.

CAP – теорема подразумевает наложение ограничение на выполнение всех трех видов условий:

- consistency (согласованность данных)
- availability (доступность)
- partition tolerance (устойчивость)

Согласно теореме, невозможно достичь выполнение всех трех условий сразу, максимум – это два из них.

6. Гиперкубы и кластеры, основные различия и типовые примеры использования.

Гиперкуб (*hypercube*) – данная топология представляет собой частный случай структуры решетки (grid), когда по каждой размерности сетки имеется только два процессора (т.е. гиперкуб содержит 2N процессоров при размерности N). Такой вариант организации сети передачи данных достаточно широко распространен на практике и характеризуется следующим рядом отличительных признаков:

- два процессора имеют соединение, если двоичные представления их номеров имеют только одну различающуюся позицию;
- в N-мерном гиперкубе каждый процессор связан ровно с N соседями;
- N-мерный гиперкуб может быть разделен на два (N-1)-мерных гиперкуба (всего возможно N различных таких разбиений);
- кратчайший путь между двумя любыми процессорами имеет длину, совпадающую с количеством различающихся битовых значений в номерах процессоров (данная величина известна как расстояние Хэмминга).

Кластер представляет собой два или более компьютеров (часто называемых узлами), объединяемые при помощи сетевых технологий на базе шинной архитектуры или коммутатора и предстающие перед пользователями в качестве единого информационно-вычислительного ресурса. В качестве узлов кластера могут быть выбраны серверы, рабочие станции и даже обычные персональные компьютеры. Узел характеризуется тем, что на нем работает единственная копия операционной системы.

Наиболее распространенными типами кластеров являются:

- *системы высокой надежности* создаются для обеспечения высокой доступности сервиса, предоставляемого кластером. Избыточное число узлов, входящих в кластер, гарантирует предоставление сервиса в случае отказа одного или нескольких серверов. Типичное число узлов — два, это минимальное количество, приводящее к повышению доступности. обычно применяются для таких служб, как базы данных, системы управления предприятием, сервера почты и мобильных сообщений, ERP и CRM приложения, файловые сервера, сервера веб-приложений, сервера печати
- *системы для высокопроизводительных вычислений* предназначены для параллельных расчетов. Эти кластеры обычно собраны из большого числа

компьютеров. Однако реализовать подобную схему удается далеко не всегда и обычно она применяется лишь для не слишком больших систем.

- *многопоточные системы* используются для обеспечения единого интерфейса к ряду ресурсов, которые могут со временем произвольно наращиваться (или сокращаться). Типичным примером может служить группа web-серверов.

Основные отличия:

1) Узлы в кластере связаны между собой очень тесно(кластер представляет собой группу компьютеров, подключенных локальной вычислительной сетью (LAN)), а гиперкуб имеет более слабые связи и широкий масштаб, могут быть географически распределены.

2) кластеры состоят из машин с аналогичным оборудованием, тогда как облака и сетки состоят из машин с, возможно, очень разными конфигурациями оборудования (Операционные системы, Базы данных и т.д.).

7. Лидеры и консенсус в распределённых системах. Типовые подходы и средства.

Для синхронизации данных и алгоритмов работы в распределенных системах применяются два обобщенных метода – выбор лидера и достижение консенсуса.

1) Выбор лидера

Выбор лидера подразумевает четкое разделение системы на одного лидера и нескольких узлов. В таких системах лидер определяет правила и порядок взаимодействия клиентского узла с критической секцией.

Пример – алгоритм Bully, который определяет лидера путем опроса одного или нескольких соседей с большим либо меньшим номером. Если хотя бы один сообщил об успехе, что следующий узел проводит опрос до тех пор, пока ему никто не ответит.

http://www.cs.colostate.edu/~cs551/CourseNotes/Synchronization/BullyExample.html

2) Консенсус

Алгоритм консенсуса децентрализован и подразумевает синхронизацию данных между узлами.

Пример – Phase King. Алгоритм в определенный момент времени, определяет лидера по порядковому или случайному номеру. Если таковой совпал с текущем узлом, то он становится лидером на определенный промежуток времени, и получив данные со всех узлов, синхронизирует их между собой.

8. Средства обеспечения устойчивости (с) в распределённых системах.

Устойчивость. Под устойчивостью понимается возможность дублирования несколькими компьютерами одних и тех же функций или же возможность автоматического распределения функций внутри системы в случае выхода из строя одного из компьютеров. В идеальном случае это означает полное отсутствие уникальной точки сбоя, то есть *выход* из строя одного любого компьютера не приводит к невозможности обслужить *запрос* пользователя.

Шаблонов обеспечения устойчивости / стабильности в сложных распределенных системах:

1) **Pattern[6] = Rate-limiting and Throttling (Ограничение скорости и регулирование)**

Ограничение скорости может применяться к запросам, поступающим в вашу систему (ограничивающий доступ к вашим API), или к запросам,покидающим вашу систему (при обращении к сторонним службам).Мы используем алгоритм Leaky Bucket в большинстве мест для запросов ограничения / регулирования скорости

Используется алгоритм Leaky Bucket в большинстве мест для запросов ограничения / регулирования скорости.

Ограничение скорости / регулирование может помочь защитить системы во многих сценариях:

1) Сценарий атаки на вашу систему

2) Защита вашей интеграции со сторонними внешними сервисами

2) **Bulk-heading**

Название « **Переборка»** происходит от разделенных перегородок на корабле, где, если перегородка повреждена / скомпрометирована, только поврежденная область заполняется водой и предотвращает затопление всего корабля.

Точно так же вы можете предотвратить сбой в одной части вашей распределенной системы, который может повлиять и повредить другие части.

Шаблон перегородки может применяться в системе несколькими способами.

Случай 1: физическая избыточность

Случай 2: Распределение ресурсов по категориям (CRA)

3) **Очередь**

Организация очереди - это буферизация сообщений в очереди для последующей отправки потребителям, что позволяет отделить производителей от потребителей.

Очередь обеспечивает устойчивость благодаря:

- Отделение производителей от потребителей
- Сгладить всплески трафика путем буферизации
- Попытки на случайные сбои

4) **Мониторинг и оповещение**

Мониторинг и оповещения могут помочь вам измерить текущее состояние / состояние вашей системы, чтобы вы как можно быстрее оповещались о сбоях в вашей системе. Это может предотвратить превращение неисправностей в неисправности.

- Он уведомляет о потенциальных сбоях в системе, помогая восстановить его до того, как он вызовет сбой во всех системах (время безотказной работы) (упреждающий)
- Это может помочь диагностировать проблему и точно определить основную причину, чтобы помочь восстановить ваши системы после сбоя (MTTR - среднее время восстановления) (активно)

Ссылка на оригинал статьи
(https://blog.gojekengineering.com/how-to-build-resilience-in-large-scale-distributed-systems-ddd1496b6c)

9. Специфические алгоритмы и типы данных для параллельных систем

Пример 1. Быстрая сортировка массива. Параллельный алгоритм

Пусть, исходный набор данных расположен на первом процессоре, с него начинается работа алгоритма:

1. Процессор выбирает ведущий элемент, сортирует остальные элементы относительно него (большие - в правую сторону, меньшие - в левую);
2. Меньшую часть он отдает другому свободному процессору;
3. После этого процессор продолжает работать с большей частью одновременно, в то время как другой процессор работает с меньшей по принципу начиная с 1го пункта;
4. Когда все процессоры получат свою часть, ускорение алгоритма будет максимальным.

Пример 2. Перемножение матриц. Алгоритм Фокса.

Пусть перемножаемые матрицы A и B имеют порядок n. Количество процессов является полноым квадратом, квадратный корень которого кратен n. В алгоритме Фокса матрицы разделяются среди процессов в виде клеток шахматной доски. При этом процессы рассматриваются как виртуальная двухмерная сетка, и каждому процессу назначена подматрица каждого множителя. В конце работы полученные блоки собираются в единую матрицу.

Алгоритм состоит в следующем:

for(step = 0; step < q; step++) {

* Выбрать подматрицу A в каждой строке для всех процессов

* В каждой строке для всех процессов разослать сетку подматриц, выбранную в этой строке для других процессов в этой строки

* В каждом процессе, перемножить полученную подматрицу A на подматрицу B, находящуюся в процессе

* В каждом процессе, отослать подматрицу B процессу, расположенному выше. (Для процессов первой строки отослать подматрицу в последнюю строку.)

}

Пример 3. Умножение матрицы на вектор.

При разделении элементов матрицы по строкам, каждому процессу распределяется часть строк исходной матрицы. Строки распределяются последовательно, поровну между всеми процессами. В результате каждый процесс скалярно умножит отправленные ему строки матицы A на вектор b и получит часть результирующего вектора c. Останется только собрать все части результата в единый вектор.

10. Специфические алгоритмы и типы данных для распределённых систем. CRDT.

В распределённых хранилищах или редакторах каких-либо данных часто бывает нужна поддержка внесения изменений оффлайн, без блокировок и конфликтов. Для этого применяются разные подходы, один из которых — алгоритмы и типы данных conflict-free replicated data type (CRDT).

Сделать readonly-реплику просто и всем понятно, как. Возможность записи в реплику — сложная штука. В CRDT эта задача решается обеспечением strong eventual consistency (SEC) и монотонности состояний.

Eventual consistency называется такая организация взаимодействия и хранения данных, при которой все реплики после прекращения внесения в них изменений когда-нибудь в конечном счёте придут в эквивалентное состояние.

Strong Eventual Consistency накладывает ещё одно ограничение: реплики, получившие одинаковые апдейты (не важно, в каком порядке), приходят в эквивалентное состояние немедленно после получения апдейтов.

В CRDT предполагается, что система обеспечивает SEC и её состояния монотонно прогрессируют, не приводя к конфликтам. Монотонность в этом смысле означает отсутствие откатов: операции нельзя отменить, вернув систему в раннее состояние. Состояния такой системы связаны отношением частичного порядка, в математике такая система с определённой на ней операцией объединения называется полурешёткой.

Полурешётка — частично упорядоченное множество: элементы (не обязательно все) связаны отношением «следует за». Бинарная операция на данном множестве коммутативна и идемпотентна.

Для разных задач удобны разные способы организации взаимодействия и хранения данных. CRDT принято разделять на два класса:

- **Коммутативные** (commutative, CmRDT, op-based): предположим, у вас есть список. При добавлении элемента вы отправляете всем репликам только это измененённое состояние (добавленный элемент). Операции должны быть коммутативными, чтобы состояние реплики не зависело от порядка получения апдейтов.
- **Основанные на хранении состояния** (convergent: CvRDT, state-based): в этом случае отправляются не отдельные апдейты, а вся структура данных

целиком (то есть, весь список, в случае списка). Структура данных должна поддерживать операции:

- **query** — прочитать что-то, не изменяя состояние (например: есть ли элемент в списке?)
- **update** — изменить структуру (например: добавить элемент в список)
- **merge** — замёржить состояние, пришедшее из другой реплики. Эта операция должна быть коммутативной, ассоциативной и идемпонетной: мёрж любых состояний в любых направлениях не «возвращает» систему в более раннее состояние, а монотонно увеличивает состояние системы. Если объединить все реплики, они должны прийти в эквивалентное состояние, объединяющее в себе все изменения, сделанные в репликах.

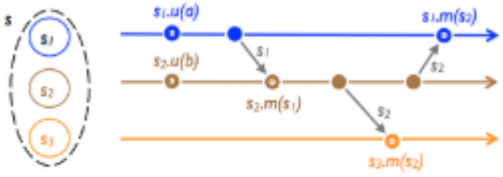


Figure 1: State-based replication

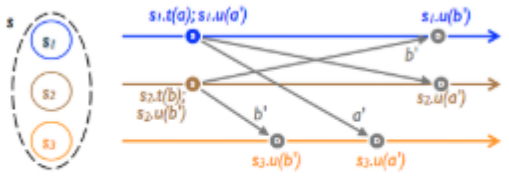


Figure 2: Operation-based replication

11. Сетевое взаимодействие. Многоуровневые модели, распределение нагрузки. Высокая нагруженность и высокая доступность, способы реализации.

Общеприняты следующие **многоуровневые модели сетевого взаимодействия**: OSI и TCP/IP (частный случай OSI)

Сетевая модель OSI (англ. open systems interconnection basic reference model — Базовая Эталонная Модель Взаимодействия Открытых Систем) — сетевая модель стека (магазина) сетевых протоколов OSI/ISO. Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом

Модель OSI				
Уровень (layer)		Тип данных (PDU[1])	Функции	Примеры
Host layers	7. Прикладной (application)	Данные	Доступ к сетевым службам	HTTP, FTP, POP3, WebSocket
	6. Представления (*presentation*)		Представление и шифрование данных	ASCII, EBCDIC
	5. Сеансовый (session)		Управление сеансом связи	RPC, PAP
	4. Транспортный (transport)	Сегменты (segment) /Дейтаграммы (datagram)	Прямая связь между конечными пунктами и надёжность	TCP, UDP, SCTP, PORTS
Media[2] layers	3. Сетевой (network)	Пакеты (packet)	Определение маршрута и логическая адресация	IPv4, IPv6, IPsec, AppleTalk
	2. Канальный (data link)	Биты (bit)/ Кадры (frame)	Физическая адресация	PPP, IEEE 802.22, Ethernet, DSL, ARP, L2TP, сетевая карта.
	1. Физический (physical)	Биты (bit)	Работа со средой передачи, сигналами и двоичными данными	USB, кабель ("витая пара", коаксиальный, оптоволоконный), радиоканал

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. описывается правилом (протоколом передачи).

	TCP/IP	OSI
7	**Прикладной**	**Прикладной**
6		**Представления**
5		**Сеансовый**
4	***Транспортный***	**Транспортный**
3	***Сетевой***	**Сетевой**
2	***Канальный***	**Канальный**
1		**Физический**

Распределение нагрузки

Как правило необходимо обслуживать одновременно огромное количество клиентов.

Горизонтальное масштабирование заключается в добавлении большего количества машин (или узлов) в систему для увеличения пропускной способности (*capacity*).

Вертикальное масштабирование — это по сути «купить машину побольше/посильнее» —

При горизонтальном масштабировании необходимо быстрое распределение запросов между узлами.

Высокая нагруженность и высокая доступность, способы реализации.

Высоконая нагруженность

Высокая нагруженность означает, что система должна сохранять стабильную работку при одновременной работе с ней большого количества клиентов.

Например есть сайт: БД+вебсервер(nginx)+бэкенд(php)

- **отделение базы данных** (вынесение бд на другой сервер)
- **отделение веб сервера**
- **введение нескольких копий бэкэнда**
- **подключение серверов кэширования** (memcache самостоятельно распределит нагрузку между используемыми серверами.)
- **введение очереди задач**. Сервер очереди принимает задачи от приложения.
- **отделение веб сервера**
- **введение нескольких копий бэкэнда**
- **подключение серверов кэширования** (memcache самостоятельно распределит нагрузку между используемыми серверами.)
- **введение очереди задач**. Сервер очереди принимает задачи от приложения.
- **файловые хранилища**

Другой вариант: http://dotrunet.ru/hiload.html

Высокая доступность

Доступность любой из систем важна.

Согласованность — это ключевая проблема в высокодоступных системах. Система

согласована, если все узлы видят и возвращают одни и те же данные в одно и то же время.

Слабая согласованность: когда важнее скорость доступа

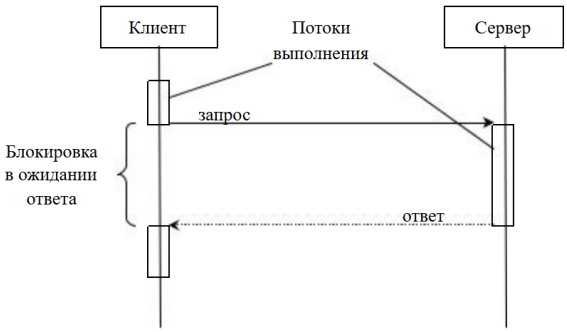
- обновить некоторые реплики, например наиближайшие
- обновленная реплика шлет сообщение обновления другим репликам
- различные реплики могут возвращать разные значения для запрашиваемого атрибута объекта должно быть возвращено либо значение, либо “не известно” с отметкой времени
- в долгосрочной перспективе обновления должны распространяться на все реплики учитывая отказы, перезапуски, порядок прибытия, конфликтующие обновления.

Сильная согласованность: гарантирует полную согласованность

Все реплики возвращают теже значения запрашиваемых атрибутов объекта.

Это достигается за счет высокой задержки.

12. Синхронное и асинхронное взаимодействие, взаимодействие в реальном времени.



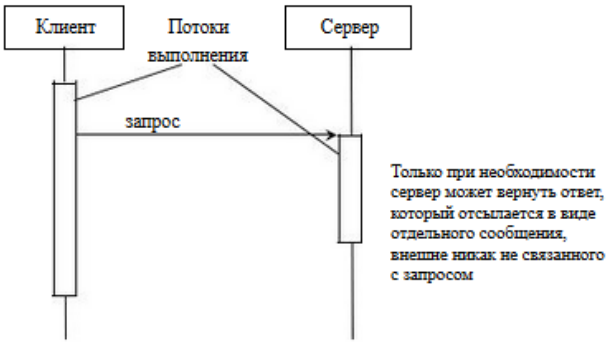
Синхронное взаимодействие.

При описании взаимодействия между элементами программных систем инициатор взаимодействия, т.е. компонент, посылающий запрос на обработку, обычно называется клиентом, а отвечающий компонент, тот, что обрабатывает запрос — сервером. «Клиент» и «сервер» в этом контексте обозначают роли в рамках данного взаимодействия. В большинстве случаев один и тот же компонент может выступать в разных ролях — то клиента, то сервера — в различных взаимодействиях.

Синхронным (synchronous) называется такое взаимодействие между компонентами, при котором клиент, отослав запрос, блокируется и может продолжать работу только после получения ответа от сервера. По этой причине такой вид взаимодействия называют иногда блокирующим (blocking).

Обычное обращение к функции или методу объекта с помощью передачи управления по стеку вызовов является примером синхронного взаимодействия.

Синхронное взаимодействие ведет к значительным затратам времени на ожидание ответа. Это время часто можно использовать более полезным образом: ожидая ответа на один запрос, клиент мог бы заняться другой работой, выполнить другие запросы, которые не зависят от еще не пришедшего результата. Поскольку все распределенные системы состоят из достаточно большого числа уровней, через которые проходят практически все взаимодействия, суммарное падение производительности, связанное с синхронностью взаимодействий, оказывается очень большим.



Асинхронное взаимодействие.

В рамках асинхронного (asynchronous)или неблокирующего (non blocking) взаимодействия клиент после отправки запроса серверу может продолжать работу, даже если ответ на запрос еще не пришел.

Асинхронное взаимодействие позволяет получить более высокую производительность системы за счет использования времени между отправкой запроса

и получением ответа на него для выполнения других задач. Другое важное преимущество асинхронного взаимодействия — меньшая зависимость клиента от сервера, возможность продолжать работу, даже если машина, на которой находится сервер, стала недоступной.

В то же время асинхронные взаимодействия более сложно использовать. Поскольку при таком взаимодействии нужно писать специфический код для получения и обработки результатов запросов, системы, основанные на асинхронных взаимодействиях между своими компонентами, значительно труднее разрабатывать и сопровождать.

Для выполнения нескольких потоков одновременно в многозадачной операционной системе реального времени необходимо, чтобы ОС имела развитой механизм взаимодействия потоков между собой. Потоки отправляют и получают сообщения, а также отвечают на них. При этом изменяются состояния потоков. Определяя состояния потоков, ОС осуществляет их оптимальное планирование с максимально эффективным использованием ресурсов. Следовательно, механизм обмена сообщениями является основополагающим и постоянно действующим на всех уровнях ОС. Для приложений, работающих в режиме реального времени, требуется, чтобы механизм межзадачного взаимодействия имел высокую степень надежности, поскольку процессы, на основе которых работают такие приложения, тесно связаны между собой.

Причинами изменения состояния потока могут быть:

- блокировка потока;
- изменение приоритета потока;
- собственное «желание» потока освободить процессор.

Среди потоков, готовых к исполнению, всегда исполняется тот поток, который имеет наибольший приоритет. Если все потоки, готовые к исполнению, имеют одинаковый приоритет, то они устанавливаются в очередь. Таким образом, можно считать, что для каждого уровня приоритета существует своя очередь.

13. Анонимные распределённые системы, отличия от классических систем и средства реализации. Технология блокчейн как вид распределённых систем.

Анонимные распределенные системы – распределенные системы, ключевым аспектом которой является сохранение анонимности и конфиденциальности для узлов сети. Анонимность в сети достигается многоуровневым шифрованием, а отсутствие единой точки отказа делает перехват трафика или даже взлом узла сети не фатальным событием. За анонимность пользователь расплачивается увеличением отклика, снижением скорости и большим объемом сетевого трафика.

Особенности системы:

- нет гарантии постоянного соединения;
- акцент на анонимность и конфиденциальность узла сети;
- распределенность, отсутствие единой точки отказа.

Некоторые системы строят таким образом, что получив пакет, не сможет определить его источник. На принципе работы анонимных распределенных системах основана работа блокчейна.

Блокчейн – выстроенная по определенным правилам непрерывная последовательность цепочки блоков, содержащих информацию. Чаще всего, копии цепочек блоков хранятся на множестве разных компьютеров независимо друг от друга.

Блок транзакций — специальная структура для записи группы транзакций в системе Биткойн и аналогичных ей. Транзакция считается завершённой и достоверной («подтверждённой»), когда проверены её формат и подписи, и когда сама транзакция объединена в группу с несколькими другими и записана в специальную структуру — *блок*. Содержимое блоков может быть проверено, так как каждый блок содержит информацию о предыдущем блоке. Все блоки выстроены в одну цепочку, которая содержит информацию обо всех совершённых когда-либо операциях в базе. Самый первый блок в цепочке — *первичный блок*— рассматривается как отдельный случай, так как у него отсутствует родительский блок.

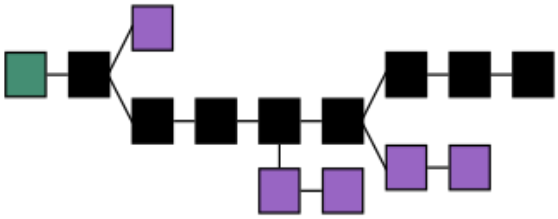
Блок состоит из заголовка и списка транзакций. Заголовок блока включает в себя свой хеш, хеш предыдущего блока, хеши транзакций и дополнительную служебную информацию. В системе Биткойн первой транзакцией в блоке всегда указывается получение комиссии, которая станет наградой майнеру за созданный блок. Далее идёт список транзакций, сформированный из очереди транзакций, ещё не записанных в

предыдущие блоки. Критерий отбора из очереди задаёт майнер самостоятельно. Это не обязательно должна быть хронология по времени. Например, могут включаться только операции с высокой комиссией или с участием заданного списка адресов. Для транзакций в блоке используется древовидное хеширование, аналогичное формированию хеш-суммы для файла в протоколе BitTorrent.

Созданный блок будет принят остальными пользователями, если числовое значение хеша заголовка равно или меньше определённого целевого числа, величина которого периодически корректируется.

Попадание транзакции в блок является подтверждением её достоверности вне зависимости от наличия других транзакций с теми же биткойнами. Каждый новый блок считается дополнительным «подтверждением» транзакций из предыдущих блоков.

Основная последовательность блоков (чёрные) является самой длинной от начального (зелёный) до текущего. Побочные ветви (фиолетовые) отсекаются.



Помимо майнинга криптовалют, технологию блокчейн используют для удостоверения личности, платежа, банковского сектора и земельного реестра.

14. Типичные сложности в реализации распределенных систем и способы их решения.

Приведу википедийную оригинальную статью.

https://en.wikipedia.org/wiki/Fallacies_of_distributed_computing

В 1994 году Питер Дейч сформулировал семь заблуждений о распределённых вычислениях. Позже 1997 году Джеймс Гослинг добавил восьмое. Это типичные ошибки, которые допускают специалисты – новички в разработки распределенных систем.

1. Сеть надежна.

   Есть вероятность что некоторые из наиболее надежных ресурсов выходят из строя, к примеру электричество, сетевые провода обрываются, софт глючит (привет VS).

2. Нулевая латентность.

   Если пропускная способность вашей сети большая, то ей нужно какое-то время чтобы заработать в полную мощность. Поэтому ваше плавное AJAX приложение надо тестировать не только в корпоративной сетке

3. Пропускная способность бесконечна.

   Несмотря на то, что пропускная способность растет очень быстро, мы всегда найдём, чем ей загрузить: большие интерфейсы, IP-телефония, видео высокого качества.

4. Сеть безопасна.

   В вашу сеть могут и скорее всего в бедующем будут пытаться влезть ребята, которые хотят стырить всё что возможно.

5. Топология никогда не меняется.

   Появляются новые интерфейсы, старые устройства меняются на старые. В случае мобильных устройств топология меняется постоянно. Так что не стоит исключать тот момент, что передаваемые данные всегда будут ходить одним и тем же путем.

6. Администратор всегда только один.

   В наиболее распространенном случае, когда задействованы более чем одна компания, проблема где-то на стыке, решить её бывает не так просто, как на первый взгляд.

7. Цена передачи данных нулевая.

   Компьютеры, роутеры, время людей всё стоит денег.

8. Сеть однородна.

   В сети работают устройства с разными операционными системами, все время появляются новые, в частности мобильные устройства.