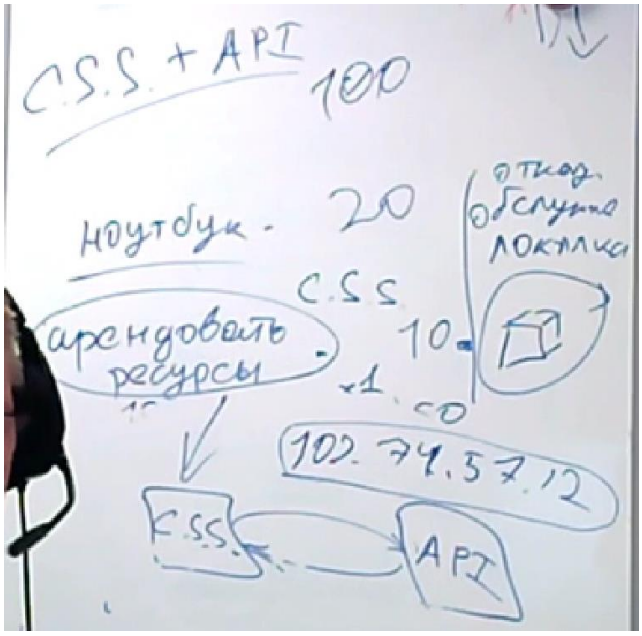
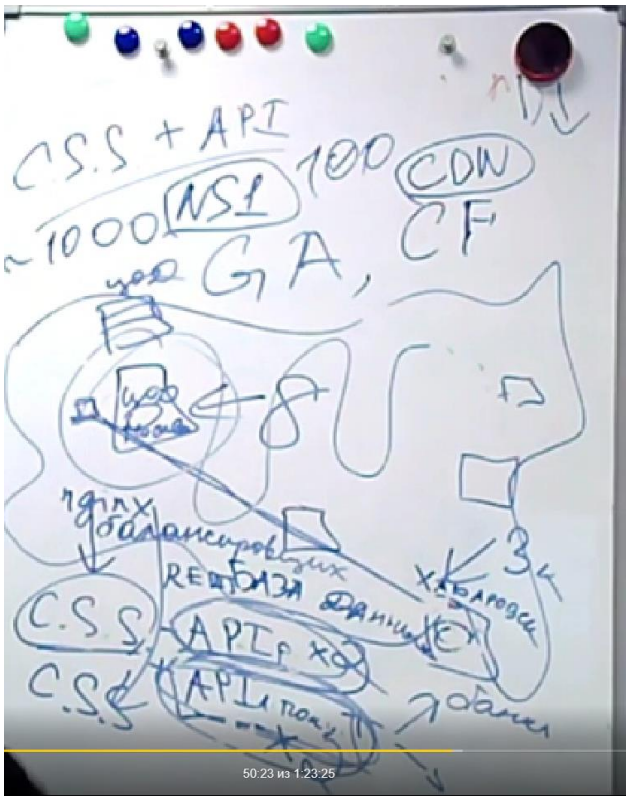
Лекция 11 сентября.

Самый простой пример: вычисления не на одном устройстве, а на несколько.

Основное – компоненты системы отвечают за свой элемент, делают вычисление и получают результат.



Система будет считаться распределённой, если она географически распределена.



Распределенная система – это набор независимых компонент системы, представляющихся их пользователям единой объединенной системой.

- Аппаратно – взаимосвязанные ресурсы.
- Программно – совокупность независимых процессов.

ПК, процессоры, процессы – узлы распределенной системы.

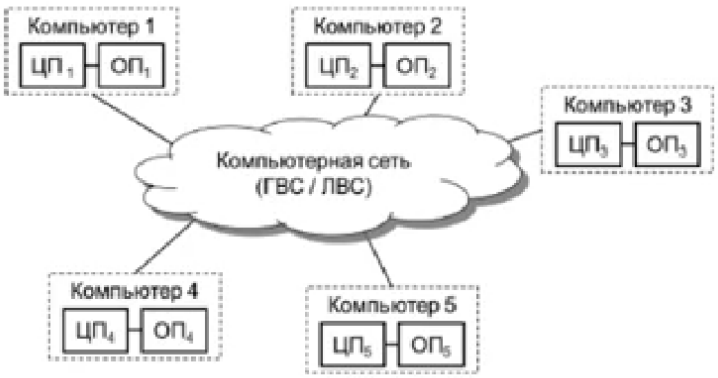
Отличие от параллельных ПК

Схема «одна команда для многих данных».

В РВ каждый процесс обладает набором данных (переменные, регистры и т.д.).

Состояние одно узла полностью закрыто для другого. Обмениваются информацией общаясь по каналу.

Пример РВ



Основные отличительные признаки РВ.

- Отсутствие единого времени (нет синхронности)
- Отсутствие общей памяти
- Географическое распределение
- Независимость и гетерогенность (установка серверов разной мощности и работа их в совокупности)

Цели построения РВ

- Географически РВС

Пример: сеть Интернет, взаимодействие межу банками. Асинхронная.

- Увеличение производительности

Создание решений многопроцессорных или многомашинных комплексов. Легкая масштабируемость системы.

- Совместное использование ресурсов

Обеспечение доступа к удаленным ресурсам и обеспечение их совместного использования.

- Отказоустойчивость

Выход из строя отдельных узлов или их компонентов слабо отражается на функционировании всей системы.

- Масштабирование (увеличение количества ресурсов, потребляемых системой)

Требования к РС

- Transparency (Прозрачность)

Умение системы скрывать свою распределенную природу.

Прозрачность доступа, местоположения, перемещения, смены местоположения клиента, репликации, одновременного доступа, отказов.

- Открытость (Openness)

Открытая система - система, реализующая стандарты на интерфейсы, службы, процессы, достаточные, чтобы обеспечить:

- Кроссплатформенность
- Возможность взаимодействия с другими системами
- Возможность пользователя системе переходить взаимодействия от системы к системе

- Безопасность (Security)

Защита ресурсов от несанкционированных действий. Продолжение предоставления ресурсов в условиях наличия вредоносной активности, направленной на вывод из строя компонентов РВС.

- Масштабируемость (Scalability)

Способность системы расти вместе с ростом нагрузки на нее.

Распространение.

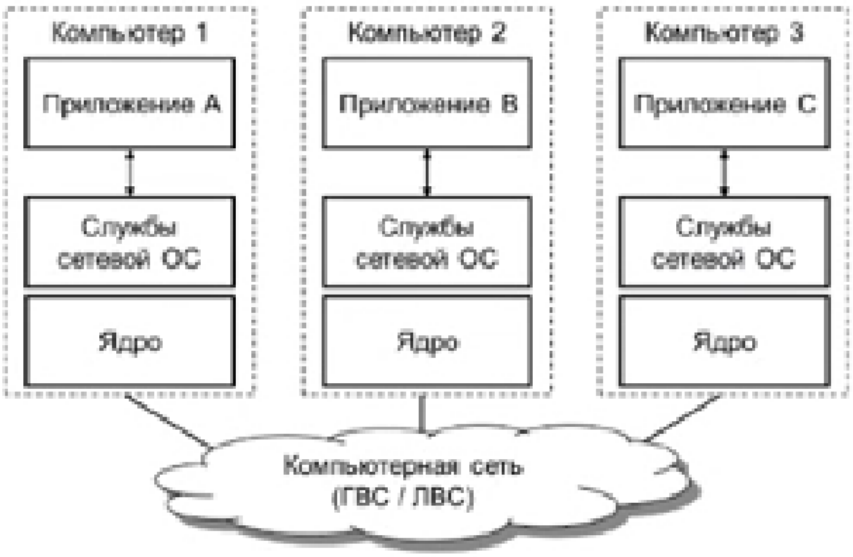
Репликация.

Кэширование, чтобы быстрее отдавать данные

ПО промежуточного уровня

Абстрагирование приложений от базовых платформ

Сетевые ОС



Проблема кроссплатформенности, необходимо реализация взаимодействия

Заблуждения при разработке РВС

1. Сеть надежна

Между серваками есть сеть, которая может выйти из строя.

2. Задержки между узлами равна 0

Серваки: один в Хабаровске другой в Спб, один в Спб другой в Америке и из-за этого задержки.

3. Неограниченная полоса пропускания
4. Неизменность сетевой топологии

Сеть не поменяется, всегда такая же – заблуждение.

5. Однородность сети
6. Один администратор
7. Транспортная инфраструктура бесплатно

На отправку данных мы тратим деньги.

Лекция 25 сентября

Технология виртуализации

10 лет назад средняя утилизация по ЦПУ составляла 3%. Настолько процентов загружен сервис от его возможной мощности.

Виртуализация – это средство для организации на одном стационарном компьютере нескольких виртуальных машин. (это не ПЛОХОЕ определение).

Иными словами, под этим термином понимается абстракция вычислительных ресурсов и предоставление пользователю системы, которая скрывает в себе свою собственную реализацию.

То есть, есть какое-то громадное количество ресурсов, мы делаем их абстракцию и какую-то часть этих ресурсов выделяем пользователю. Пример: отдельная виртуальная машина, ресурсы, которые выдаются, гипервизер и т.д. Пользователь природы ресурсов не знает.

Виды виртуализации

Виртуализация представлений.

Самый яркий пример – это терминальные службы Windows Server. Терминальный сервер предоставляет свои вычислительные ресурсы клиентам, и клиентское приложение выполняется на сервере, клиент же получает только «картинку».

Представляют что-то. Подключение с помощью удаленного рабочего стола. Виртуализация удаленного сервера. Более живой пример: облачный гейминг, Steam. Вычислений происходят в игре на сервере на стороне. Пользователю нужно fullHD и видеокарта.

Виртуализация платформ.

- Полная эмуляция (симуляция)

При данном виде базовая система полностью виртуализирует всё аппаратное обеспечение при сохранении гостевой операционной системы (ОС) в неизменном виде

Гостевая ОС – ОС, которая устанавливается поверх основной ОС. Полностью виртуализируется все программное обеспечение.

Минус: снижение быстродействия

Общение через кучу драйверов к основным ПО ОС.

Возможность полного изолирования.

- Полная виртуализация (родная виртуализация)

В данном случае базовая система виртуализирует только нужное количество аппаратного обеспечения для того, чтобы виртуальная машина могла быть запущена отдельно.

Данный вид виртуализации существенно позволяет увеличить быстродействие гостевых систем (по сравнению с полной эмуляцией) и довольно широко используется в настоящее время.

Виртуализируется только нужное число ПО.

Примеры: XEN, VMware Workstation, VMware Server, Vmware ESXi Server, Virtual PC, VirtualBox и др.

- Паравиртуализация

При применении данного вида виртуализации не нужно симулировать аппаратное обеспечение, однако, вместо этого используется так называемый специальный программный интерфейс (Application Programming Interface) для взаимодействия с гостевой ОС.

Гипервизор и операционная система объединяются при виртуализации, изменяя операционную систему, но зато приводя почти к родной производительности оборудования.

Гипервизор – прослойка между виртуальными машинами и хостовой ОС. Хост – машина, где виртуалки разворачиваются.

Минус: происходит изменение поведения ОС. Проблема функционирования системы во время тестирования и заливания на прод.

Везде с виртуалками проблема с БЕЗОПАСНОСТЬЮ!

- Виртуализация уровня операционной системы

Виртуализирует физический сервер на уровне ОС, позволяя запускать отдельные и безопасные виртуальные серверы на одном физическом сервере.

При такой виртуализации на уровне операционной системы не существует отдельного слоя гипервизора.

Все происходит внутренними сервисами.

Примеры: Linux-VServer, OpenVZ, Solaris Containers, FreeBSD Jail и др.

- Виртуализация уровня приложений

Данный вид виртуализации отличается ото всех остальных: если в вышеперечисленных случаях создаются виртуальные машины, использующиеся для отделения или изоляции приложений, то в этом случае само приложение помещается в специальный контейнер (обозначенный, как правило, в виде папки) с нужными элементами для своей работы

Эмулирование состава приложения. Посмотреть на нескольких версиях.

Примеры: Altiris, Trigerens, Softricity

Виртуализация ресурсов.

- Объединение, агрегация, концентрация компонентов

Организация нескольких физических или логических объектов в группы, представляющие удобные интерфейсы пользователя.

Примеры:

1. Виртуализация систем хранения, используемая при построении сетей хранения данных SAN (Storage Area Network);

2. Виртуальные частные сети (VPN) и трансляция сетевых адресов (NAT), позволяющие создавать виртуальные пространства сетевых адресов и имен.

3. VPN

- Кластеризация компьютеров и распределенные вычисления (grid computing)

Данный вид виртуализации включает в себя технологии (техники), применяемые при объединении множества отдельных компьютеров в глобальные системы, так называемые метакомпьютеры, решающие совместно общую задачу

Как параллельные вычисления

- Разделение ресурсов

Здесь при разделении в процессе виртуализации происходит разделение одного определенного большого ресурса на несколько схожих по типу объектов, удобных для использования. Это называется зонированием ресурсов («zoning»).

- Инкапсуляция

Это процесс создания системы, предоставляющей пользователю удобный интерфейс для работы с системой и скрывающей подробности всей сложности своей реализации.

- Облачная вычислительная среда

ГОСТ ISO/IEC 1778ft-2016

Облачные вычисления - развивающаяся парадигма, в связи с чем не предназначено предписывать или ограничивать какой-либо конкретный метод размещения, предоставления услуг или прикладных операций. В настоящем подразделе определены и описаны ключевые особенности облачных вычислений.

Пример: Яндекс диск, Гугл диск, телефон

Преимущества:

+ **Широкий сетевой доступ:** характеристика, где физические и виртуальные ресурсы доступны по сети и доступ к ним осуществляется через стандартные механизмы, которые продвигают использование разнородных платформ клиентов.

+ **Измеримое обслуживание:** характеристика, где измеренное предоставление служб облачных вычислений таково, что его использование можно отслеживать, управлять им. отчитываться и по нему можно выставить счет. Это важная характеристика, необходимая для оптимизации и проверки корректности предоставления службы облачных вычислений. Данная ключевая особенность подчеркивает тот факт, что потребитель может платить лишь за те ресурсы, которые он использует. Инфраструктура как код.

+ **Мультиаренда**: характеристика, где физические или виртуальные ресурсы распределены таким образом, при котором несколько арендаторов и их вычисления, и данные изолированы друг от друга и недоступны друг другу. Как правило. в контексте мульти аренды группа пользователей службы облачных вычислений, формирующие арендатора, будут все принадлежать одной и той же организации - потребителю службы облачных вычислений. Разные пользователи используют одни и те же мощности.

+ **Самообслуживание по требованию:** характеристика, при которой потребитель службы облачных вычислений может обеспечить требуемые вычислительные способности автоматически или путем минимального взаимодействия с поставщиком службы облачных вычислений.

+ **Быстрая эластичность и масштабируемость:** характеристика, при которой физические или виртуальные ресурсы могут быть быстро и гибко приспособлены, в некоторых случаях автоматически, для быстрого увеличения или уменьшения ресурсов. Для потребителя службы облачных вычислений физические или виртуальные ресурсы, доступные для резервирования, часто, оказываются неограниченными (**ГЛУПОСТЬ**) и могут быть приобретены в любом количестве в любое время автоматически, в соответствии с ограничениями соглашения о предоставлении услуг

+ **Пул ресурсов:** характеристика, при которой физические или виртуальные ресурсы службы облачных вычислений поставщика могут быть объединены для обслуживания одного или более потребителей службы облачных вычислений. Фокус этой ключевой характеристики в том, что поставщики службы облачных вычислений могут поддерживать мульти аренду. В то же время использовать абстракцию, чтобы замаскировать сложность процесса от потребителя. Потребители знают только то, что сервис работает, при этом, как правило, не имея

никакого знания о том, как обеспечиваются ресурсы или где они расположены, и не управляют ресурсами.

Когда что использовать?

Облачно вычислительные среды.

Недостатки:

Наличие стабильного интернета для доступа к ресурсам

Плата много раз в течение использования

Безопасность

Увеличение времени доступа

Виртуализация представлений.

Если у вас есть много пользователей, работающих с одинаковым набором ПО, и система сильно распределена территориально - то стоит подумать об использовании **виртуализации представлений**, речь – терминальных службах.

Достоинства такой системы:

Снижение требований к «железу» на стороне клиентов

Снижение требований к пропускной способности сети

Повышение безопасности

Значительное упрощение администрирования и поддержки

Недостатки:

Повышения требований серверам, как по производительности, так и по надежности

Возможная единая точка отказа

Виртуализация приложений.

Если у вас существует множество приложений, которые некорректно работают в новой ОС, либо же конфликтуют между собой, или необходимо запускать на одном компьютере несколько версий одной и той же программы – то нужна **виртуализация на уровне приложений.**

Достоинства:

Безопасность

Простота администрирования – централизованное обновление и разграничение прав на доступ к приложениям

Недостатки:

Некоторая сложность в понимании технологий и в практическом внедрении.

VirtualBox

Возможности программы:

•Одновременный запуск нескольких виртуальных машин

•Удобная для пользователя рабочая область

•Управление виртуальными машинами

•Настройка параметров для каждой VM (таких как размер ОЗУ и жесткого диска)

•Клонирование виртуальных машин или их сброс до значений по умолчанию (т. е. переустановка ОС)

•Сохранение текущего состояния VM (т. е. Получение моментального «снэпшота»)

•Запись видеокадров

Настройка сети:

•Трансляция сетевых адресов (NAT), которая является настройкой по умолчанию

•Сетевой мост (Bridged)

•Виртуальный адаптер хоста (Host Only)

•Внутренняя сеть (Internal Network)

NAT:

Протокол NAT позволяет гостевой операционной системе выходить в Интернет, используя при этом частный IP, который недоступен со стороны внешней сети или же для всех машин локальной физической сети. Извне невозможно напрямую соединиться с такой системой, если она использует NAT.

Принцип трансляции сетевых адресов заключается в следующем. Когда гостевая ОС отправляет пакеты на конкретный адрес удаленной машины в сети, сервис NAT, работающий под VirtualBox, перехватывает эти пакеты, извлекает из них сегменты, содержащие в себе адрес пункта отправки (IP-адрес гостевой операционной системы) и производит их замену на IP-адрес машины-хоста. Затем заново упаковывает их и отправляет по указанному адресу.

Сетевой мост:

В соединении типа "Сетевой мост" виртуальная машина работает также, как и все остальные компьютеры в сети. В этом случае адаптер выступает в роли моста между виртуальной и физической сетями. Со стороны внешней сети имеется возможность напрямую соединяться с гостевой операционной системой.

Адаптер в режиме "Сетевой мост" подключается, минуя хост, к устройству, которое распределяет IP-адреса внутри локальной сети для всех физических сетевых карт. VirtualBox соединяется с одной из установленных сетевых карт и передает пакеты через нее напрямую; получается работа моста, по которому передаются данные.

Виртуальный адаптер хоста:

При подключении типа "Виртуальный адаптер хоста" гостевые ОС могут взаимодействовать между собой, а также с хостом. Но все это только внутри самой виртуальной машины VirtualBox. В этом режиме адаптер хоста использует свое собственное, специально для этого предназначенное устройство, которое называется vboxnet0.

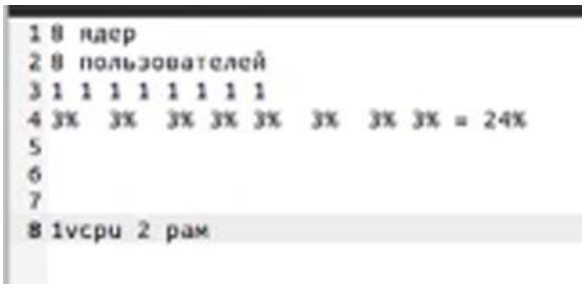
Также им создается подсеть и назначаются IP-адреса сетевым картам гостевых операционных систем. Гостевые ОС не могут взаимодействовать с устройствами, находящимися во внешней сети, так как они не подключены к ней через физический интерфейс.

Внутренняя сеть:

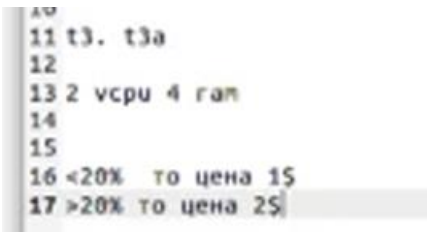
При подключении типа "Виртуальный адаптер хоста" гостевые ОС могут взаимодействовать между собой, а также с хостом. Но все это только внутри самой виртуальной машины VirtualBox. В этом режиме адаптер хоста использует свое собственное, специально для этого предназначенное устройство, которое называется vboxnet0.

Также им создается подсеть и назначаются IP-адреса сетевым картам гостевых операционных систем. Гостевые ОС не могут взаимодействовать устройствами, находящимися во внешней сети, так как они не подключены к ней через физический интерфейс.

Разбор примера с предоставлением ресурсов:



Амазон:



Облачные вычислительные среды

Концепция ОВС

Первые идеи об использовании облачных вычислений (Cloud Computing) как публичной услуги были предложены еще в 1960-х известным ученым в области информационных технологий, изобретателем языка Lisp, профессором MIT и Стэнфордского университета **Джоном Маккарти** (John McCarthy). Понятие облака (cloud) уже давно ассоциируется с метафорическим изображением интернета, с помощью которого доступны некоторые сервисы. Облачные вычисления — это практическая реализация данной идеи. Облачные вычисления основаны на масштабированных и виртуализованных ресурсах (данных и программах), которые доступны

пользователям через Интернет и реализуются на базе мощных центров обработки данных (data centers).

Что-то, чем вы пользуетесь, и это находится в Интернете.

Возникновение облаков

Реализация первого реального проекта приписывается компании Salesforce.com , основанной **в 1999 году.** Именно тогда и появилось первое предложение нового вида b2b продукта "Программное обеспечение как сервис" ("Software as a Service", "SaaS"). Определенный успех Salesforce в этой области возбудил интерес у гигантов ОТ индустрии, которые спешно сообщили о своих исследованиях в области облачных технологий. И вот уже первое бизнес-решение под названием **"Amazon Web Services" было запущено в 2005 году** компанией Amazon.com, которая со времен кризиса доткомов активно занималась модернизацией своих дата центров. Следующим свою технологию постепенно ввела Google, начав с **2006** года b2b предложение SaaS сервисов под названием **"Google Apps".** И, наконец, свое предложение анонсировала компания Microsoft, презентовав ее на **конференции PDC 2008** под названием "Azure Services Platform".

Виды облаков

С понятием облачных вычислений часто связывают такие сервис-предоставляющие технологии (Everything as a service)

- **Инфраструктура как сервис («Infrastructure as a Service»)**

Инфраструктура как сервис (laaS)

laaS - это предоставление компьютерной инфраструктуры как услуги на основе концепции облачных вычислений.

laaS состоит из трех основных компонентов:

1. Аппаратные средства (серверы, системы хранения данных, клиентские системы, сетевое оборудование)

2. Операционные системы и системное ПО (средства виртуализации, автоматизации, основные средства управления ресурсами)

3. Связующее ПО (например, для управления системами)

Примеры: Amazon, OpenStack, Eucalyptus

- **Платформа как сервис («Platform as a Service»)**

PaaS – это предоставление интегрированной платформы для разработки, тестирования, развертывания и поддержки веб-приложений как услуги.

Для разворачивания веб-приложений разработчику не нужно приобретать оборудование и программное обеспечение, нет необходимости организовывать их поддержку. Доступ для клиента может быть организован на условиях аренды. Масштабируемость PaaS предполагает автоматическое выделение и освобождение необходимых ресурсов в зависимости от количества обслуживаемых приложением пользователей.

PaaS как интегрированная платформа для разработки, тестирования, разворачивания и поддержки веб-приложений позволит весь перечень операций по разработке, тестированию и разворачиванию веб-приложений выполнять в одной интегрированной среде, исключая тем самым затраты на поддержку отдельных сред для отдельных этапов. Способность создавать исходный код и предоставлять его в общий доступ внутри команды разработки значительно повышает производительность по созданию приложений на основе PaaS.

Примеры: AppEngine, Hadoop, Windows Azure,

- **Программное обеспечение как сервис («Software as a Service»)**

SaaS - модель развертывания приложения, которая подразумевает предоставление приложения конечному пользователю как услуги по требованию (on demand). Доступ к такому приложению осуществляется посредством сети, а чаще всего посредством Интернет-браузера. В данном случае, основное преимущество модели SaaS для клиента состоит в отсутствии затрат, связанных с установкой, обновлением и поддержкой работоспособности оборудования и программного обеспечения, работающего на нём. Целевая аудитория – конечные потребители.

В модели SaaS:

приложение приспособлено для удаленного использования;

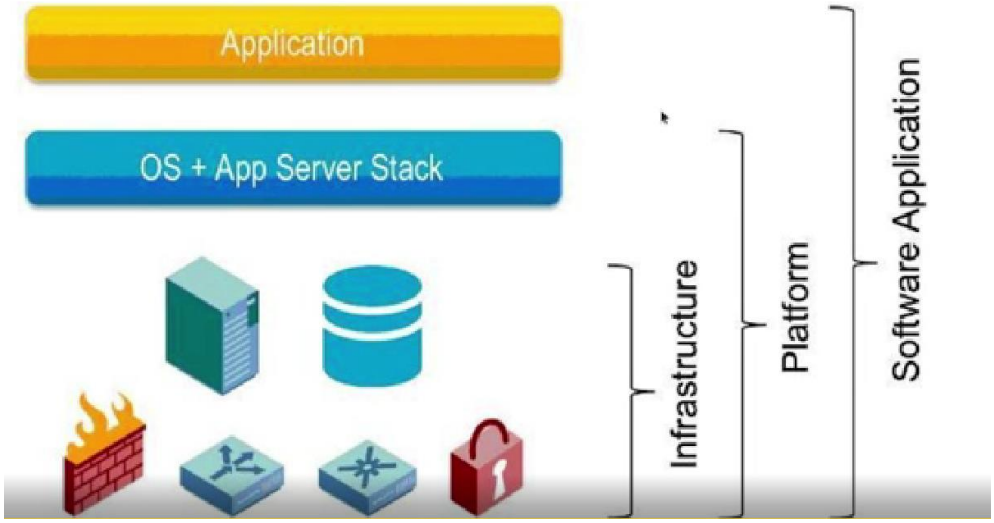
одним приложением могут пользоваться несколько клиентов;

оплата за услугу взимается либо как ежемесячная абонентская плата, либо на основе суммарного объема транзакций;

поддержка приложения входит уже в состав оплаты;

модернизация приложения может производиться обслуживающим персоналом плавно и прозрачно для клиентов.

Отличия … как сервис



Инфраструктура за:

- Коммутацию
- Файервол
- Защиту
- Вычислительные мощности
- Управление
- Отвечает за ВСЕ

Платформа за:

- Инфраструктуру
- ОС
- Библиотеки

ПО за:

- Все, что выше
- Приложения

Варианты развертывания ОВС

Частное облако (private cloud) - используется для предоставления сервисов внутри одной компании, которая является одновременно и заказчиком, и поставщиком услуг. Это вариант реализации "облачной концепции", когда компания создает ее для себя самой, в рамках организации. В первую очередь реализация private cloud снимает один из важных вопросов, который непременно возникает у заказчиков при ознакомлении с этой концепцией - вопрос о защите данных с точки зрения информационной безопасности. Поскольку "облако" ограничено рамками самой компании, этот вопрос решается стандартными существующими методами. Для private cloud характерно снижение стоимости оборудования за счет использования простаивающих или неэффективно используемых ресурсов. А также, снижение затрат на закупки оборудования за счет сокращения логистики (не думаем, какие сервера закупать, в каких конфигурациях, какие производительные мощности, сколько места каждый разрезервировать и т.д.)

В сущности, мощность наращивается пропорционально растущей в целом нагрузке, не в зависимости от каждой возникающей задачи - а, так сказать, в среднем. И становится легче и планировать, и закупать и реализовывать – выпускать новые задачи в производство.

Минусы: самообслуживание, сервера ломаются, экспертные моменты, выключение электричества.

Публичное облако - используется облачными провайдерами для предоставления сервисов внешним заказчикам.

Смешанное (гибридное) облако - совместное использование двух вышеперечисленных моделей развёртывания. Вообще одна из ключевых идей Cloud заключается как раз в том, чтобы с технологической точки зрения разницы между внутренними и внешними облаками не было, и заказчик мог гибко перемещать свои задания между собственной и арендуемой ИТ-инфраструктурой, не задумываясь, где конкретно они выполняются.

Преимущества ОВС

Доступность и отказоустойчивость

•Не надо закупать дорогостоящее оборудование

•Доступ к документам из любого места, подключенного к Интернет

•Устойчивость к потере данных или краже оборудования

•Надежность

Экономичность и эффективность

•Аренда ресурсов

•Аренда ПО

Простота

Гибкость и масштабируемость

Недостатки ОВС

Постоянное соединение с сетью

Безопасность

Функциональность "облачных" приложений

Не все фитчи есть

Зависимость от "облачного" провайдера

Коммуникация как Сервис (Saas)

Коммуникация как Сервис (Saas) - построенное в облаке коммуникационное решение для предприятия. Поставщики этого типа облачного решения отвечают за управление аппаратным и программным обеспечением, требуемым для того, чтобы предоставить:

- систему связи, обеспечивающую передачу речевого сигнала по сети Интернет или по любым другим IP-сетям (VoIP),
- обмен мгновенными сообщениями (IM),
- видеоконференц-связь.

Эта модель начала свой эволюционный процесс в индустрии телекоммуникаций, не сильно отличаясь от модели SaaS, стала результатом сектора служб доставки программного обеспечения. Вендоры Saas ответственные за управление аппаратным и программным обеспечением их пользователей. Вендоры Saas, как правило, предоставляют гарантируемое

качество обслуживания (QoS) в соответствии с соглашением сервисного обслуживания (SLA).

Архитектура OpenStack (общий пример)

Контроллер – узел, управляющий всей системой.

Узлов может быть несколько.

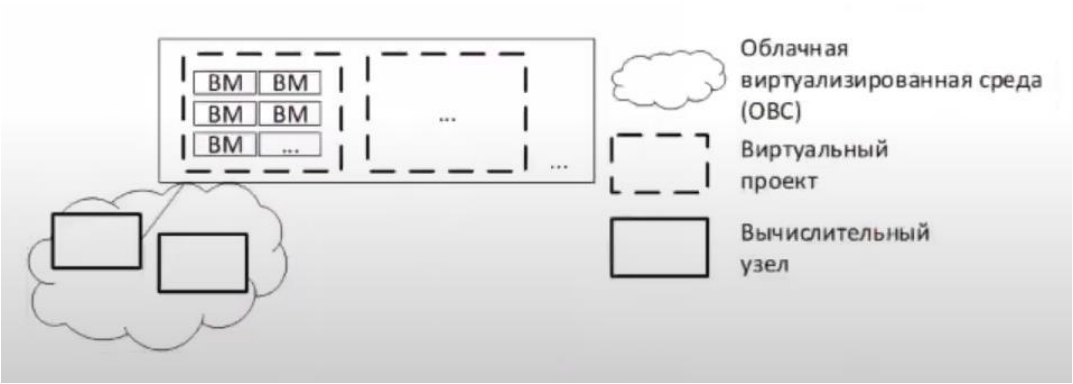
На вычислительных узлах создание виртуалок и их хранение.

Коммутирующий узел – взаимодействие с виртуалками.

Вычислительный узел может заменять коммутирующий узел.

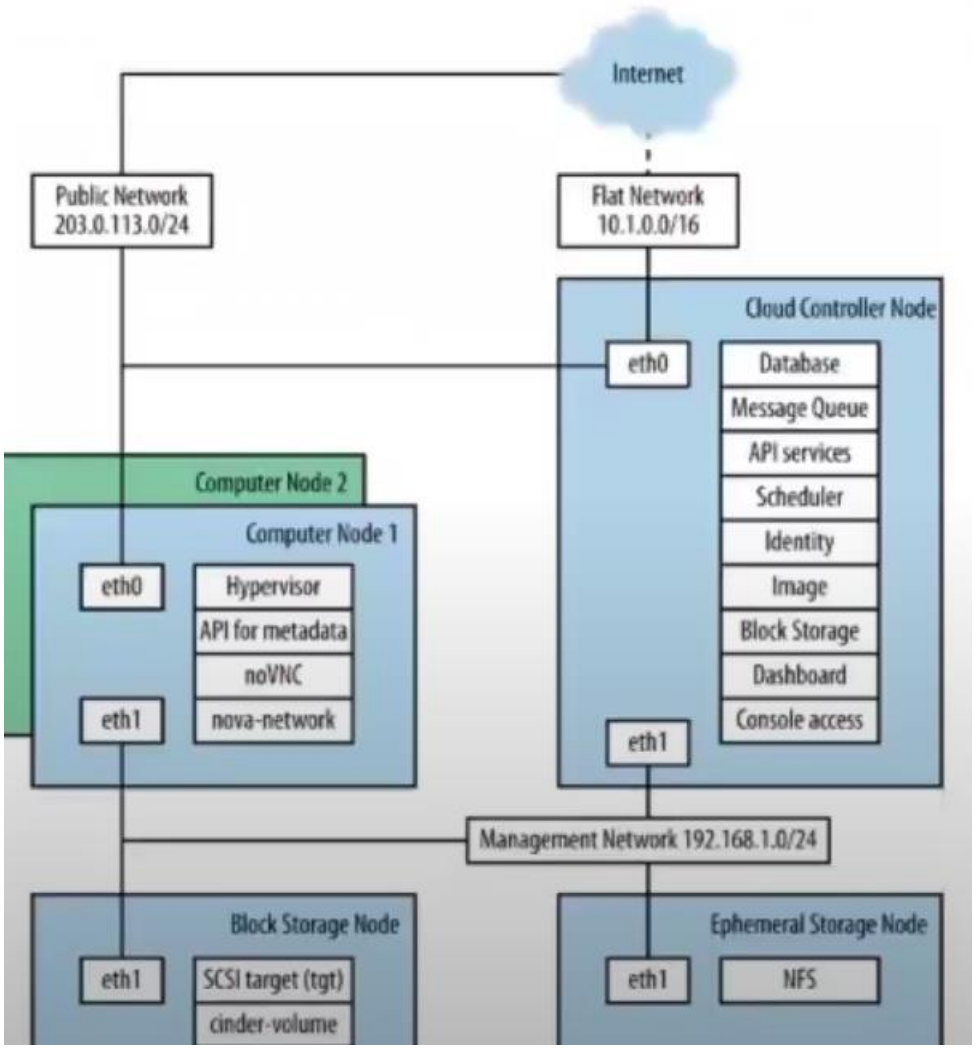
Также бывают сервера хранения данных, которые подключаются к вычислительным узлам.

Структура облачных вычислительных сред.

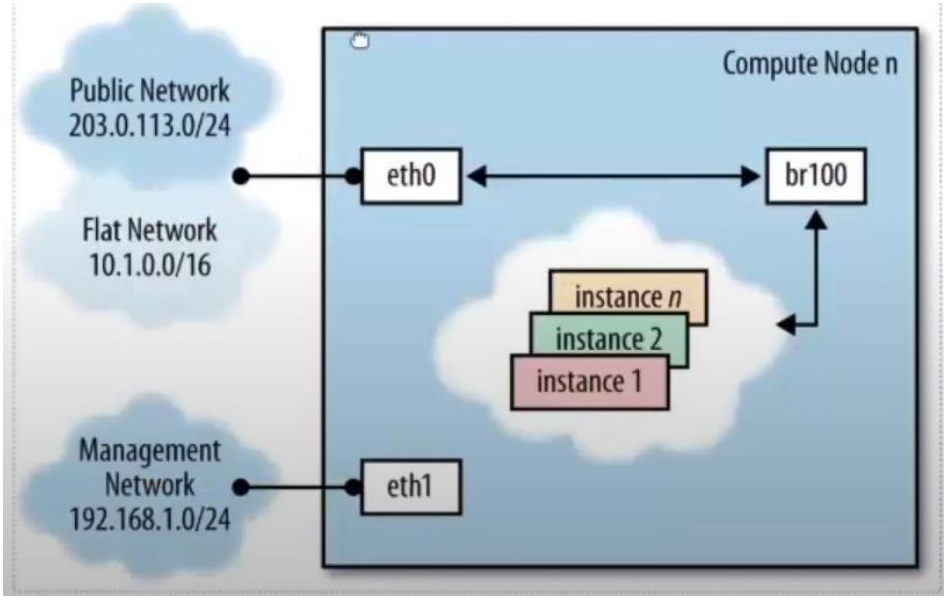


ВМ обособлены от виртуалок на другом виртуальном проекте.

Логическая архитектура



Как проходит сетевой трафик в OpenStack



Контроллер

Узлы контроллера отвечают за выполнение служб управляющего программного обеспечения, необходимого для работы окружения OpenStack. Эти узлы:

- Обеспечивают интерфейс переднего плана для доступа клиентов, а также службы API, которые являются средством общения для всех других компонентов в среде.
- Выполняют ряд служб в режиме высокой доступности с использованием Pacemaker и HAProxy для поддержки виртуальных IP и функций балансировки нагрузки, что обеспечивает использование всех узлов.
- Поддержание служб "инфраструктуры" высокой доступности таких как MySQL и Qpid, которые лежат в основе всех остальных служб.
- Обеспечение того, что называется "постоянным хранением" с помощью служб, также размещенных на этих хостах. Эти системы постоянного хранения являются основой на узлах хранения для обеспечения надежности.

Узел хранения

Узлы хранения содержат все необходимые среде данные, включая образы дисков в библиотеке службы образов и постоянные тома системы хранения, создаваемые службой блочного хранения. Узлы хранения используют технологию GlusterFS для хранения высокодоступных и масштабируемых данных.

Сетевой узел

Сетевые узлы отвечают за обеспечение всех виртуальных сетей для клиентов, для создания общедоступных или частных сетей и выдачи виртуальных машин наружу, во внешние сети. Сетевые узлы:

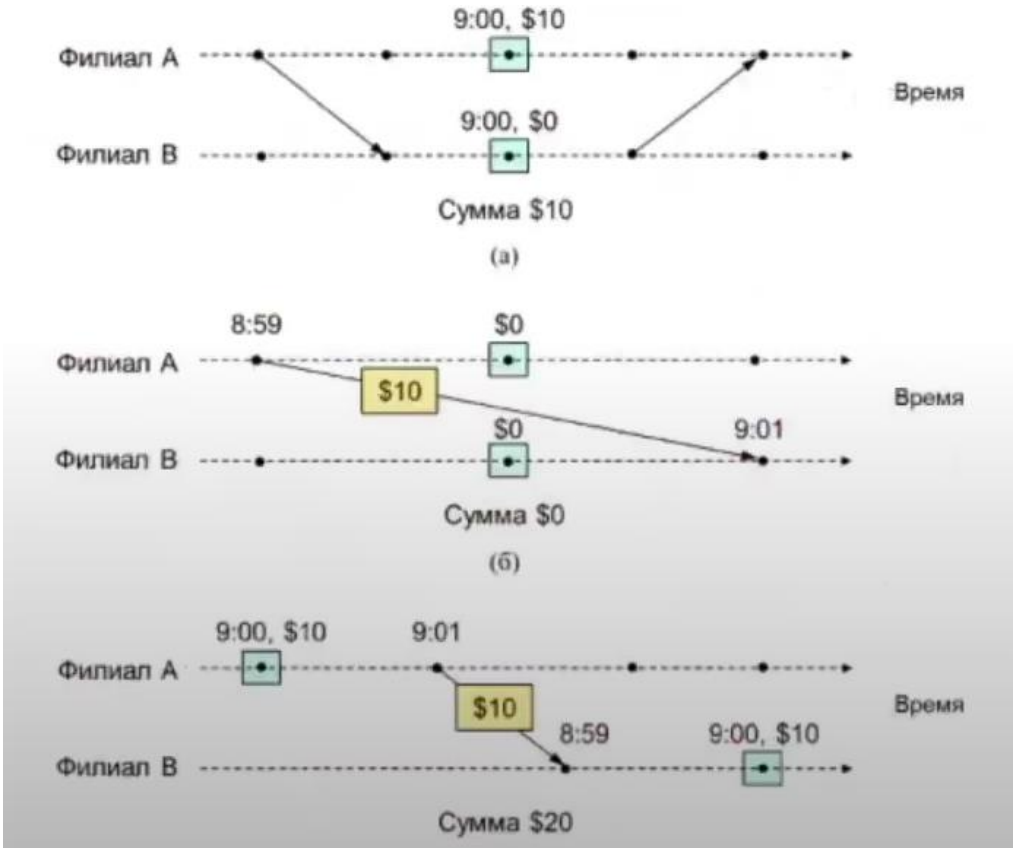
• Формируют только входную и выходную точку для исполняемых поверх OpenStack экземпляров.

• Запускают все службы сетевой среды за исключением службы сетевого API (которая работает на үзле контроллера)

Лекция 9 октября

АРВС

Проблема синхронизации времени на узлах РВС



Синхронные распределенные системы.

Сложнее чем АРВС.

Ограничения:

•Время выполнения ограничено снизу и сверху конкретным значениями (точно знаем, за сколько будет получен результат)

•Задержка доставки сообщения не выше конкретного предела(timeout)

•Скорость отклонения локальных часов не превышает конкретное значение (кварцевые часы, подсчет времени, которое может отличаться)

Преимущества:

Использование таймаутов для определения отказа процесса/узла

Пример: Форд-Боярд

Асинхронные распределенные системы

Ограничения:

•Время выполнения не ограничено сверху (но имеет конечное значение)

•Задержка доставки сообщения – произвольное кол-во времени

•Скорость отклонения локальных часов произвольна

Пример: web-страничка Яндекс.ру

Проблемы асинхронных систем

Порядок наступления событий

Пользователи А, Б, В

1. A send C,B 2. B receive от A, send ответ A, C

2. C receive A, B, send ответ A, B

Есть почтовая пересылка между 4 людьми. Пользователь А отправляет всем пользователям сообщения.

X получает неправильный порядок сообщений.

Важен порядок наступления событий (сообщений). Функционал такой надо добавлять в АРВС соответственно.

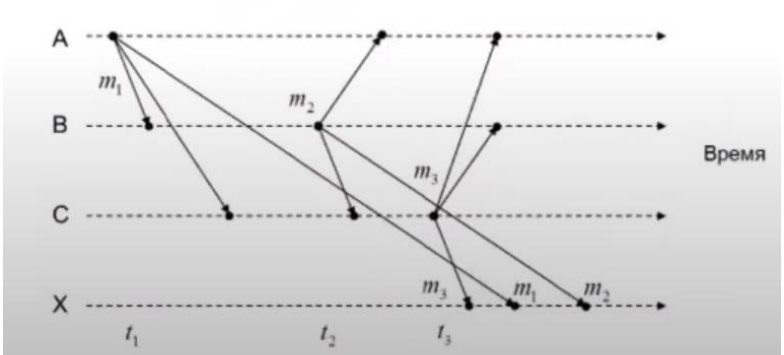


Рис. 1.6. Порядок сообщений, наблюдаемый различными пользователями.

Функции отправки и получения данных

Базовые примитивы – отправить и получить данные.

*send(void *sendbuf, int count, int dest),*где

sendbuf - указывает на буфер, содержащий передаваемые данные,

count - содержит количество передаваемых элементов данных,

dest - идентифицирует процесс-получатель.

*receive(void *recvbuf, int count, int source),* где

recvbuf - указывает на буфер, сохраняющий принимаемые данные,

count - содержит количество принимаемых элементов данных,

source - идентифицирует процесс-отправитель.

Пример

P_1	P_2

a = 100;
send(&a, 1, 2);
a = 0;
	receive(&a, 1, 1);
	printf("%d\n", a);

0? 100? ⟶ P2

Может произойти так, что send отправит0, а не 100. Блокирующие операции отправки и получения – решение.

Решения проблемы (1)

Блокирующие операции отправки/получения без буферизации

Что может быть плохо – см рисунок 2. Оба процесса будут пытаться друг другу что-то отправить, но функцию принятия не будет работать.

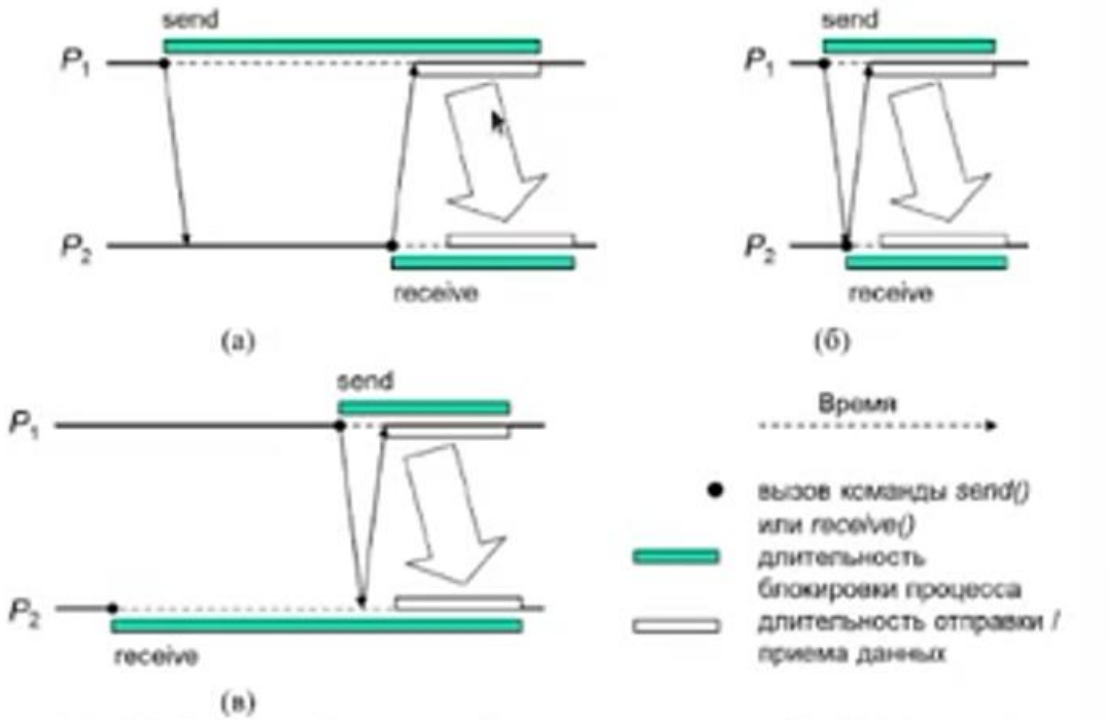


Рис. 1.7. Механизм блокирующей отправки/ получения без буферизации; (а) первым осуществляется вызов *send()*, (б) вызовы *send()* и *receive()* происходят практически одновременно, (в) первым осуществляется вызов *receive()*.

P_1	P_2
send(&a, 1, 2); receive(&b, 1, 2);	send(&a, 1, 1); receive(&b, 1, 1);

Решения проблемы (2)

Блокирующие операции буферизированной отправки/получения

Доп.буфер не меняется!

Минусы такого решения: долгое копирование, использование дополнительной памяти (доп нагрузка на ЦПУ); буфер не бесконечный, возможно переполнение;

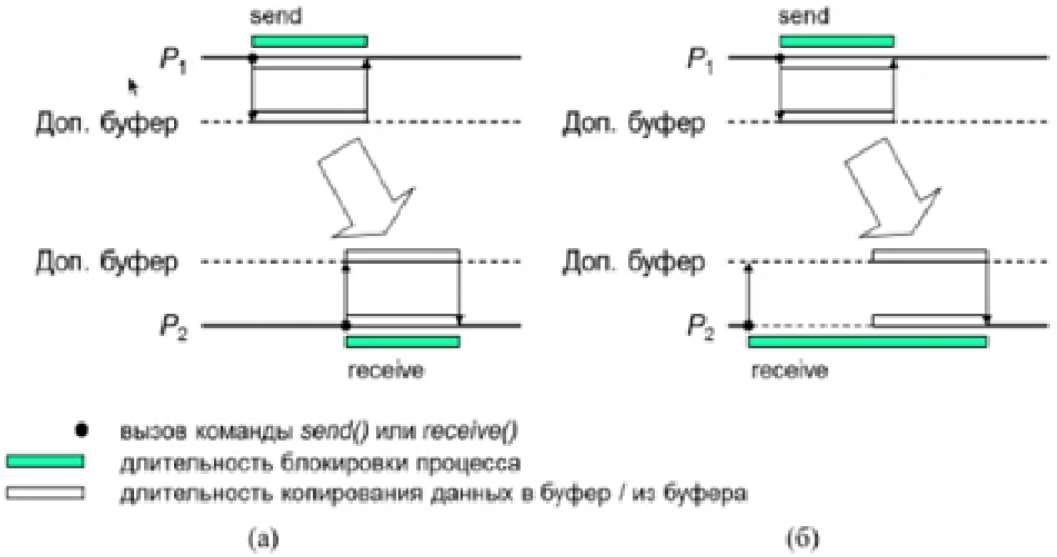


Рис. 1.8. Механизм блокирующей буферизованной отправки/ получения;
(а) первым осуществляется вызов *send()*, (б) первым осуществляется вызов *receive()*.

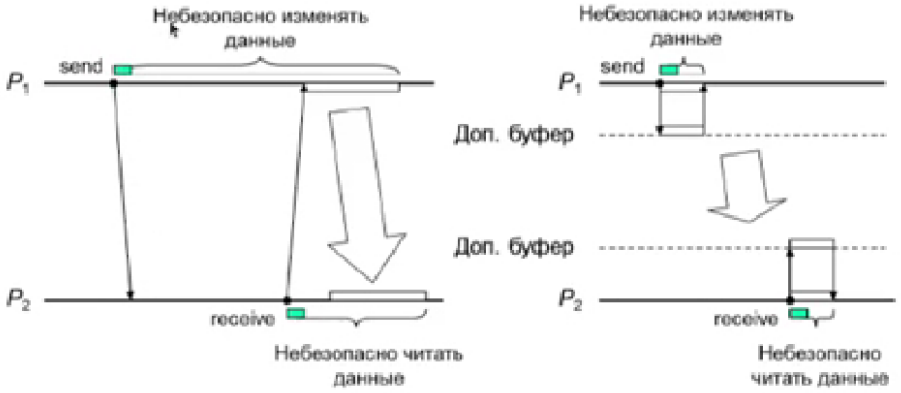
Неблокирующие операции отправки/получения

Есть минимальный простой.

Основной смысл определения блокирующих/неблокирующие отправок – описание локального поведения вызывающего процесса.

Можно делать синхронно.

Асинхронный обмен сообщениями. Преимущества как у АРВС.



(а) (б)

Рис. 1.9. Механизмы неблокирующей отправки/ получения;
(а) без буферизации, (б) с использованием буферов.

Лекция 23.10

Модель распределенных вычислений

Модель

{P1, P2, …, Pn}

Pi – процесс распределенной системы

Cij – канал между процессами i и j

Для удобства будем считать канал однонаправленным.

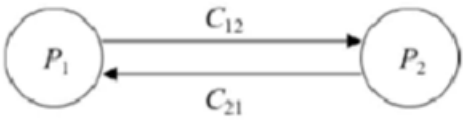
Состояние канала C

Определяется совокупностью всех сообщений, отправленных данному каналу, но еще не полученных каналом

Состояние процесса P

Определяется в контексте поставленных задач, фиксирует все существенные выполненные задачи. Определяет дальнейшее поведение процесса, последующие реакции на внешние события

Пример распределенной системы из двух процессов



Событие e процесса P

e = (P, s, s', m, C)

- P - процесс, в котором оно произошло;
- s - состояние процесса P перед событием e;
- s' - состояние P после события e;
- C - канал, используемый в событии e;
- m - сообщение, отправленное по каналу C;

Работа модели в виде последовательности событий

$R = (e^0, e^1, \dots e^x, e^{x+1}, \dots)$

Глобальные состояния, соответствующие событиям

$(S^0, S^1, \dots S^x, S^{x+1} \dots)$ такая, что $S^{x+1} = e^x(S^x)$ для всех $x >= 0$

Каждое событие строго друг за другом

Глобальные состояния показывают, что события изменили в нашей системе.

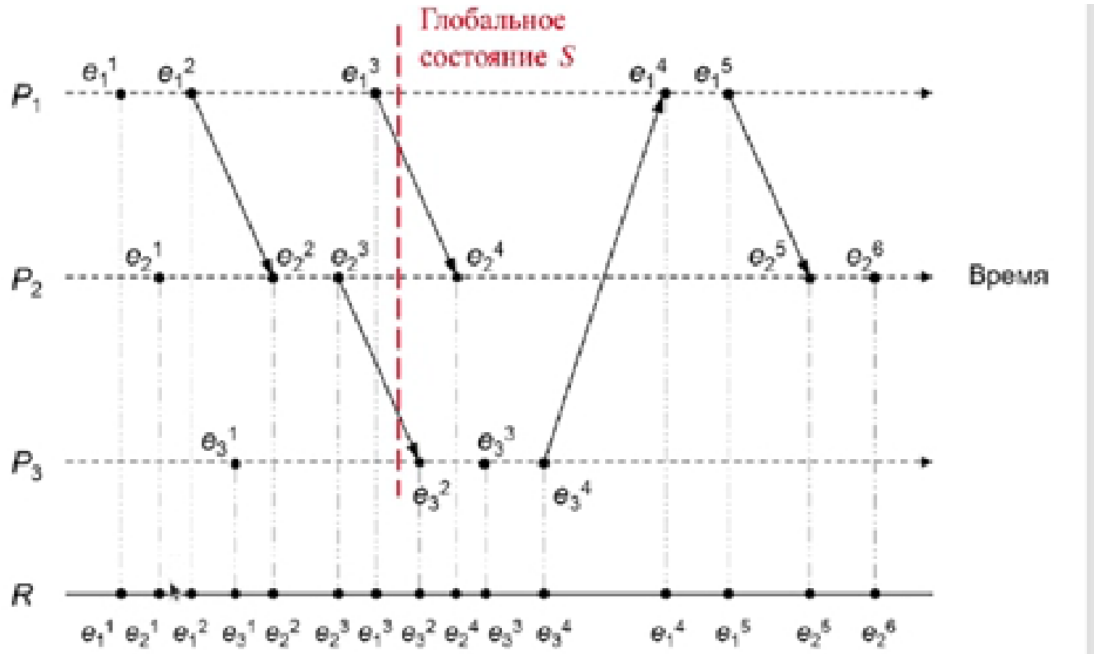
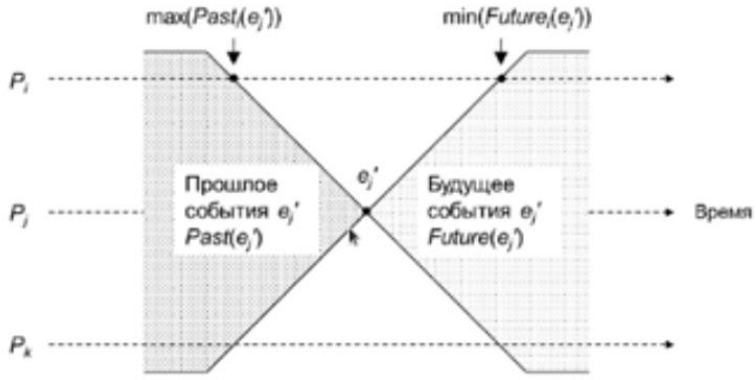


Рис. 2.6. Пример пространственно-временной диаграммы распределенного выполнения.

Конус прошлого и будущего



В распределенном вычислении $\mathbf{C}$ на событие e_j' потенциально влияют только такие события e_i, для которых $e_i \rightarrow e_j'$, и событие e_j' "знает" о наступлении только этих событий e_i. Поэтому множество таких событий e_i формирует "прошлое" события e_j', которое обозначается через $Past(e_j')$:

$$Past(e_j') = \{e_i: \forall e_i \in \mathbf{C}, e_i \rightarrow e_j'\}.$$

Свойства каналов

Очередность

Свойства каналов связи определяются допущениями относительно взаимосвязи событий отправки и получения сообщений.

Свойство очередности. Канал, сохраняющий порядок передаваемых по нему сообщений, называется очередью, или каналом FIFO (англ. First In First

Out). Это означает, что если отправка сообщения m, происходит раньше, чем отправка сообщения по одному и тому же каналу, то получение сообщения m4 также происходит раньше, чем получение сообщения. При отправке сообщения по такому каналу оно помещается в конец очереди, а при получении сообщения оно извлекается из начала очереди.

Канал, не являющийся очередью (поп-FIFO канал), можно представить в виде мультимножества, в которое отправляющий процесс добавляет элементы (сообщения), а принимающий процессе изымает элементы (сообщения) в случайном порядке.

Емкость

Емкость каналов. Емкостью канала называется количество сообщений, которые могут находиться в канале в состоянии пересылки. Канал считается переполненным, если число содержащихся в нем сообщений в точности совпадает с его емкостью.

Для асинхронного механизма обмена сообщениями событие отправки сообщения будет допустимо только в тех состояниях, в которых канал не является переполненным. Стоит отметить, что при описании модели распределенной системы в п. 2.1, мы предполагали, что допустимость события отправки сообщения не зависит от состояния канала, тем самым неявно подразумевая, что каналы имеют неограниченную емкость и, как следствие, никогда не переполняются. В дальнейшем мы будем рассматривать именно такие каналы.

Логические часы

Поэтому *логическими часами* (англ. *logical clock*) назовем всякую функцию Θ, отображающую множество событий распределенного вычисления $\mathbf{C}$ в некоторое упорядоченное множество $(T, <)$, где T представляет собой совокупность допустимых значений *логического времени* (англ. *time domain*). В распределенных системах каждый процесс P_i может управлять работой только своих часов Θ_i независимо от часов других процессов. При этом отметка логического времени события e_i, происходящего в процессе P_i, будет определяться показаниями часов этого процесса на момент наступления e_i, т.е. $\Theta(e_i) = \Theta_i(e_i)$ для всех событий e_i процесса P_i. Как было указано выше, показания часов Θ_i должны изменяться *между* наступлением событий процесса P_i, т.е. сам факт изменения показаний Θ_i событием не является.

Множество событий, которые происходят с распределенными вычислениями, становятся упорядоченными друг за другом.

Есть общие логические часы и внутри каждого процесса.

Показания часов изменяются между наступлениями событий.

Логические часы требования

$$\forall e_i, e_j' \in \mathbf{C}: e_i \rightarrow e_j' \Rightarrow \Theta(e_i) < \Theta(e_j').$$

Данное требование монотонного возрастания времени называют условием непротиворечивости логических часов (англ. clock consistency) или условием непротиворечивости отметок времени событий.

Непротиворечивость часов.

Условие 1: если e_i и e_i' - два разных события одного и того же процесса Pi, и событие e_i, наступает в Pi, раньше события e_i', то $\Theta(e_i) < \Theta(e_i')$.

Условие 2: если e_i и e_i' - взаимосвязанные события отправки и получения одного и того же сообщения, передаваемого из процесса Pi, в процесс Pj, то $\Theta(e_i) < \Theta(e_i')$.

Логические часы

В логических часах каждого процесса Pi часто выделяют две составляющие.

- *Логические локальные часы* для измерения собственного хода выполнения процесса. То есть логические локальные часы используются процессом Pi для записи информации о ходе своего собственного выполнения.

- *Логические глобальные часы* для описания локального представления процесса Pi о глобальном времени. То есть логические глобальные часы используются процессом Pi для записи информации о ходе выполнения других процессов. Логические глобальные часы позволяют процессу назначать непротиворечивые отметки времени для собственных событий.

Основная цель – определить, когда какие события происходят, которые важны и отражают состояние системы и при этом зависят от каких-то предыдущих событий.

Основной целью метода продвижения логического времени является обеспечение условия непротиворечивости логических часов. Описание такого метода складывается из определения следующих двух правил.

Правило 1: определяет, как процесс изменяет показания своих логических локальных часов при наступлении в нем любого события.

Правило 2: определяет, как процесс изменяет показания своих логических глобальных часов для отражения своего представления о ходе выполнения других процессов. Точнее говоря, это правило определяет, какая информация о текущем логическом времени процесса включается в каждое отправляемое им сообщение, и как процесс, получающий такое сообщение, использует эту информацию для обновления своего локального представления о глобальном времени.

Скалярное время Лэмпорта

Для линейного упорядочивания событий распределенного вычисления Л. Лэмпорт предложил использовать часовую функцию, отображающую множество событий распределенного вычисления в множество неотрицательных целых чисел. В этом случае логическое локальное время процесса Pi и его представление о логическом глобальном времени выражаются одной скалярной величиной, обозначаемой как Li. Отметку времени произвольного события ei будем обозначать через L(ei).

Правила 1 и 2 продвижения логического времени определяются следующим образом.

Правило 1: перед выполнением любого события процесс Pi увеличивает показания своих часов Li:

Li = Li + d, где d > 0.

В общем случае d может принимать любое значение, каждый раз разное для каждого последующего события. Однако чаще всего значение d полагают всегда равным единице. Логические часы Li инициализируется нулем.

Правило 2: в каждое передаваемое сообщение добавляется значение логического времени Li процесса-отправителя Pi на момент отправки этого сообщения. Когда процесс Pj получает такое сообщение, содержащее отметку времени Lmsg он выполняет следующие шаги:

1. Lj = max(Lj, Lmsg)
2. исполняет Правило 1;
3. доставляет сообщение и приступает к его обработке.

Нетрудно видеть, что это правило обеспечивает выполнение обоих условий 1 и 2 непротиворечивости логических часов для события получения сообщения.

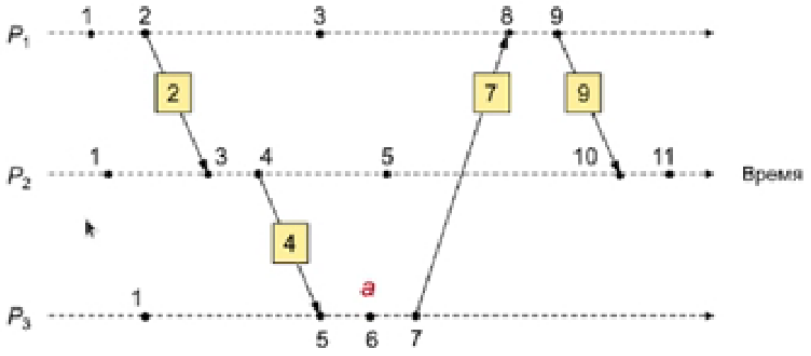
Поэтому такие логические часы являются непротиворечивыми, т.е. для любых двух событий ei и ej':

-> влияние одного события на другое

ei -> ej' => L(ei) < L(ej').

Пример

Пример работы алгоритма скалярных часов Лэмпорта для $d = 1$ приведен на рис. 3.1. Рядом с каждым событием представлена его отметка времени. На стрелках указаны отметки времени, передаваемые с сообщениями.



Безопасность облачных вычислительных сред.

Проблемы связанные с безопасностью

laaS и PaaS могут быть использованы злоумышленниками для:

•Организации взлома паролей;

•DDoS-атак;

•Размещение вредоносного кода;

•Создание ботнет сетей;

•Спам

Рекомендации к решению

•Усовершенствование процедур регистрации

•Верификация кредитных карт, использования платежных средств мониторинг

•Исследование пользователей сервиса сетевой активности

•Исследование blacklist'ов

Небезопасные API

Для управления ресурсами провайдеры предоставляют API пользователю. Безопасность все системы зависит от безопасности API

• Анонимный доступ к интерфейсу

• Перехват аутентификации (паролей, токенов)

• Наличие сервисами неизвестных взаимосвязей между сервисами

Рекомендации к решению

•Анализ и усовершенствование модели безопасности облачного провайдера

•Использование устойчивого шифрования алгоритма

Нельзя расшифровать процесс обмена сообщениями между пользователями

•Надежные методы аутентификации и авторизации

Внутренние нарушители

•Доступ к информации клиента сотрудником провайдера

Решение:

•Внедрение строгих правил закупок оборудование и использование систем обнаружения несанкционированного доступа

•Регламентирование правил найма сотрудников в публичных контрактах пользователей

•Прозрачная система безопасности

•Публикация отчетов об аудите безопасности внутренних систем провайдера

Уязвимости облачных технологий

•Уязвимости гипервизоров (организация доступа виртуальной машины к аппаратным ресурсам)

Решение:

•Внедрение передовых потенц. Методов защиты виртуальных сред

•Использование систем обнаружения несанкционированного доступа

•Применение надежных правил аутентификации и авторизации на проведение административных работ

•Требования к времени применения обновлений

•Сканирование и обнаружение уязвимостей

Потеря или утечка данных

•Неправильно применение аутентификации, авторизации и аудита правил

•Неверное использование правил и методов шифрования

•Поломка оборудования

Решение:

•Использование безопасного API

•Шифрование и защита передаваемых данных

• Надежная система шифрования управления ключами

•Покупка надежных носителей информации

•Резервное копирование данных

Кража персональных данных

•Подбор пароля и логина

•Использование их для получения необходимых данных

Решение:

•Запрет на передачу учетных записей

•Двухфакторный метод аутентификации

•Активный мониторинг

Прочие уязвимости

•Отсутствие описание инфраструктуры систем защиты облака для пользователей. Может быть найдена уязвимость

Пример: Отказ компании Amazon от аудита безопасности E2 Cloud. Уязвимость в ПО датацентра Heartland. Была взломана система безопасности.

Решение:

•Полное или частичное раскрытие данных об архитектуре

•Использование систем мониторинга уязвимостей