1. Понятие кластера компьютеров. Вычислительный кластер. Суперкомпьютер. Мультикомпьютер. Мультипроцессор.

Человеку нужно решать большие задачи → хочется больших мощностей → начинаем объединять процессоры → возникают проблемы взаимодействия между процессорами.

Мультипроцессоры и мультикомпьютеры

В любой параллельной компьютерной системе процессоры, выполняющие разные части единого задания, должны как-то взаимодействовать друг с другом, чтобы обмениваться информацией. Как именно должен происходить обмен? Для этого было предложено и реализовано две стратегии: мультипроцессоры и мультикомпьютеры. Ключевое различие между стратегиями состоит в наличии или отсутствии общей памяти.

Мультипроцессоры

Параллельный компьютер, в котором все процессоры совместно используют общую физическую память, называется мультипроцессором, или системой с общей памятью. Все процессы, работающие в мультипроцессоре совместно, могут иметь единое виртуальное адресное пространство, отображенное на общую память. Два процесса имеют возможность легко обмениваться информацией — для этого один из них просто записывает данные в общую память, а другой их считывает.

Мультикомпьютеры

Во втором варианте параллельной архитектуры каждый процессор имеет собственную память, доступную только этому процессору. Такая схема называется мультикомпьютером, или системой с распределенной памятью. Поскольку процессоры в мультикомпьютере не могут взаимодействовать друг с другом простыми обращениями к общей памяти, процессоры обмениваются сообщениями через связывающую их коммуникационную сеть.

Возникает вопрос: зачем вообще создавать мультикомпьютеры, если мультипроцессоры гораздо проще программировать? Ответ прост: создать большой мультикомпьютер проще и дешевле, чем мультипроцессор с таким же количеством процессоров.

Таким образом, мы сталкиваемся с дилеммой: мультипроцессоры сложно разрабатывать, но легко программировать, а мультикомпьютеры легко строить, но трудно программировать.

Кластер — группа компьютеров, объединённых высокоскоростными каналами связи и представляющая с точки зрения пользователя единый аппаратный ресурс.

Вычислительные кластеры

Кластеры используются в вычислительных целях, в частности в научных исследованиях. Для вычислительных кластеров существенными показателями являются высокая производительность процессора в

операциях над числами с плавающей точкой (flops) и низкая латентность объединяющей сети, и менее существенными — скорость операций ввода-вывода, которая в большей степени важна для баз данных и web-сервисов. Вычислительные кластеры позволяют уменьшить время расчетов, по сравнению с одиночным компьютером, разбивая задание на параллельно выполняющиеся ветки, которые обмениваются данными по связывающей сети.

2. Аппаратное обеспечение кластеров и бла бла бла...

- Учёт и орагнизация доступа пользователей
  Например, ssh с авторизацией по ключу. Общие пользователи – копирование конфигов или через общую ФС.
- Общая файловая система
  Необходима для того, чтобы на каждом узле был доступ и исполняемому файлу заадчи, файлам входных данных и для записи результатов. Вместо ОФС могут быть реализованы механизмы копирования соответствующих файлов. Реализовано может быть с помощью NFS или, что более извращённо, SSHFS, FTPFS и т.п... Экспоритировать имеет смысл /home, каталоги с ПО, нужные конфиги.
- Средства программировании
  Vim как редактор :) Компиляторы: gcc, icc и т.д... MPI (open mpi, intel mpi, mpich) Ф топку прикладные библиотеки
- Управление прохождением задач
  OpenPBS, Torque – планировщики задач, управляют очередью
- Мониторинг
  Ganglia – красивые, тыкабельные картинки с историей :)

Тенденции:

- **Сеть**: **Infiniband** — высокоскоростная коммутируемая последовательная шина. Она вместо гигабитного ethernet'a
- **Процессор**: преобладают зиончики, но последнее время появляются кластеры на теслах.
- **ОС**: Linux, Unix.

SL390s G7 + GPU
40 серверовCPU (X5670)
и 120 (61440 яде SL390s G7 с тремя **GPU M2090** каждый:
80 (480 ядер) процессоров p) процессоров GPU (Tesla M 2090).
ОП сервера - 96 Гбайт.
Пиковая производительность – **85 Тфлопс.**

3. Пакет Globus-toolkit. Проектные решения.

Нужно предоставить доступ к нашему кластеру → делаем одну большую хрень и называем её глобусом → ??? → PROFIT

- Common runtime – хрен пойми зачем надоб, “include a set of C Common libraries needed for building grid infrastructure and XIO”
- Execution management – управляет запуском и выполнением задач, GRAM состоит из разных компонент:
  - gatekeeper отвечает за авторизацию и запуск job manager'a. Создаётся на каждый запрос, но быстро подыхает.
  - Job manager – по одному на пользователя. Обрабатывает запрос за запуск и управляет передачей файлов.
  - Кучка скрптов для работы с LRM (local resource manager, типа PBS)
  - даж что-то для аудитинга есть.
- Data management – инструментарий для передачи и управленями распределёнными данными.
  - GridFTP is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. The GridFTP protocol is based on FTP. Включается в себя сервер и клиент.
  - Есть хрень для репликации данных Replica Location Service (RLS)
- Безопасность. Вся авторизации основана на сертификатах, каждый пользователь/ресурс имеет свой сертификат, который содержит информацию о нём (имя, что он из себя представляет, подпись и т.п.).
  - Авторизация по сертификату:
    1. А подключается В и даёт ему свой сертифика
    2. В у СА, которому принадлежит сертификат А, проверяет, что сертификат действителен. В, кстати, должен доверять этому СА.
    3. В проверяет, что А – это тот, кто за себя выдаёт, с помощью обмена сообщением, зашифрованным ключем из сертификата.
    4. То же, но в обратном порядке, чтобы А убедился с В.

Proxy-сертификаты

Отлично: теперь у нас есть доверенные удостоверяющие центры, их открытые ключи, сертификаты пользователей и их закрытые ключи. Мы можем шифровать сообщения и может создавать цифровые подписи, отказаться от факта производства которых достаточно сложно.

Что же еще? В многокомпонентных системах очень удобно то, что называется Single Sign-On — возможность аутентифицироваться вручную только один раз, а все остальные операции аутентификации будут проводиться автоматически. Обычно это актуально в системах, которые первоначально вас аутентифицируют, а потом система начинает выполнять действия от вашего лица, например, получать данные, запускать задачи, публиковать их результаты и т.д. Это называется делегированием.

Делегирование, основанное на proxy-сертификатах, функционирует следующим образом: после взаимной аутентификации пользователя и

сервиса, который в дальнейшем будет работать от имени пользователя, сервис создает новую ключевую пару и отправляет открытый ключ на подпись пользователю. Пользователь подписывает этот открытый ключ так же, как это делает удостоверяющий центр, но при это используется закрытый ключ пользователя. Появившийся в результате сертификат и называется proxy-сертификатом.

После этого сервис, который выступает от имени пользователя, может аутентифицироваться, используя свой (только что созданный) закрытый ключ и сертификат, подписанный пользователем. Процесс аутентификации проходит примерно так.

1. Проверяется подпись, созданная сервисом. При этом используется открытый ключ, который был передан вместе с подписью.
2. Аутентифицируется открытый ключ, которым была проверена подпись. Сначала проверяется подпись на proxy-сертификате, которая была создана с помощью закрытого ключа пользователя. Это делается с помощью открытого ключа пользователя.
3. Таким же образом аутентифицируется открытый ключ самого пользователя, но здесь уже используются данные об удостоверяющем центре.

В результате этого строится то, что называется цепочкой доверия (chain of trust), которая начинается на какой-то цифровой подписи и заканчивается на цифровой подписи удостоверяющего центра.

С помощью такого же механизма, сервис, которому первоначально был выдан proxy-сертификат, может подписать еще один proxy-сертификат, делегировав полномочия пользователя пользователя по цепочке другому сервису. Именно так и реализуется Single Sign-On.

4. Классификация распределенных вычислительных систем. Примеры распределенных систем. Примеры решения научных задач на распределенных вычислительных системах (BOINC, World community grid)

Классификация по структуре:

- Кластеры
- Грид

Классификация грида по назначению:

- Научный — хорошо распараллеливаемые приложения программируются специальным образом (например, с использованием Globus Toolkit)
- Волонтёрский — гриды на основе использования добровольно предоставляемого свободного ресурса персональных компьютеров
- Коммерческий — обычные коммерческие приложения работают на виртуальном компьютере, который, в свою очередь, состоит из нескольких физических компьютеров, объединённых с помощью грид-технологий

5. Понятие грид-систем. Требования к разработке ПО для организации грид-систем.

Под Grid-вычислениями понимается объединение большого числа разрозненных и разнородных вычислительных ресурсов. Слово "разрозненные" означает, что ресурсы географически удалены и ничего не знают друг о друге. При этом лицу, которому требуется использовать часть этих ресурсов, она предоставляются как единое целое, и он не беспокоится о внутренней организации этого "целого".

Требования:

- Fault-Tolerance
- Предоставление определенного уровня Quality of Service
- Использование открытых стандартов
- Децентрализованное администрирование

6. Программное обеспечение NumGRID для построения вычислительных грид. Проектные решения.

NumGRID – программный комплекс, предназначенный для объединения разнородных вычислительных кластеров в единый вычислительный ресурс на основе частичной реализации стандарта MPI-2.2

Профит от объединения кластеров:

1. больше процессоров, больше памяти
2. гибкость планирования
3. возможность учитывать специализированное оборудование или ПО на отдельных кластерах
4. мона вовлекать в счёт устаревшее оборудование
5. несложно наращивать мощности

Проектные решения:

1. **Внутрекластерные коммуникации**: сети с высокой пропускной способность и низкой латентностью для обмена данных между процессами вычислительных приложений. Для управления узлами и распределённой ФС используется всякое УГ.
2. **Коммуникации между кластерами**: можно делать VPN, но это не учитывает топологию сетей. В проекте PACX-MPI предполагается, что узел трансляции имеет доступ как в высокоскоростную сеть внутри кластера, так и в сеть между кластерами. Для организаци передачи сообщений между кластерами на узле трансляции запускается дополнительный процесс MPI, через этот процесс все остальные процессы внутри данного кластера общаются с внешним миром. Преимущества такого подхода – PACX-MPI осведомлен о топологии сети и на этой основе может оптимизировать межкластерные взаимодействия, эффективно используется высокоскоростная сеть.
3. Межкластерные коммуникации можно оптимизировать, за счёт параллельной доставки сообщений, если доступно более одного физического пути. Пакетирование, чтобы не блокировать передающие узлы.
4. Коллективные коммуникации – бинарное(?) дерево.
5. Автоматическое перераспределение загрузки: между узлами передаются процессы MPI. Так нагрузка становится распределённой равномерно, коммуникации можно производить на фоне счёта.
6. **Безопасность**. Пользователь сам выбирает кластеры, которым он доверяет, за счёт чего никакие сертификаты и т.п. не нужны. Шифрование коммуникаций могут реализованы за счёт внешних средств (ssh-туннелирование, напр.).
7. **Устойчиваость к сбоям**: если я правильно понял, то никакая xD

Программный комплекс NumGRID:

- Шлюз
- Библиотека NumGRID-MPI
- гуйня для мониторнига выполнения и управления.

7. Понятие облачных вычислений. Примеры систем.

Облачные вычисления – это модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу конфигурируемых вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам — как вместе, так и по отдельности), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и/или обращениями к провайдеру.

Характеристики:

- *Самообслуживание по требованию* (англ. *self service on demand*), потребитель самостоятельно определяет и изменяет вычислительные потребности, такие как серверное время, скорости доступа и обработки данных, объём хранимых данных без взаимодействия с представителем поставщика услуг;
- *Универсальный доступ по сети*, услуги доступны потребителям по сети передачи данных вне зависимости от используемого терминального устройства;
- *Объединение ресурсов* (англ. *resource pooling*), поставщик услуг объединяет ресурсы для обслуживания большого числа потребителей в единый *пул* для динамического перераспределения мощностей между потребителями в условиях постоянного изменения спроса на мощности; при этом потребители контролируют только основные параметры услуги (например, объём данных, скорость доступа), но фактическое распределение ресурсов, предоставляемых потребителю, осуществляет поставщик (в некоторых случаях потребители всё-таки могут управлять некоторыми физическими параметрами перераспределения, например, указывать желаемый центр обработки данных из соображений географической близости);
- *Эластичность*, услуги могут быть предоставлены, расширены, сужены в любой момент времени, без дополнительных издержек на взаимодействие с поставщиком, как правило, в автоматическом режиме;
- *Учёт потребления*, поставщик услуг автоматически исчисляет потреблённые ресурсы на определённом уровне абстракции (например, объём хранимых данных, пропускная способность, количество пользователей, количество транзакций), и на основе этих данных оценивает объём предоставленных потребителям услуг.

Модели развертывания:

- Частное облако
- Публичное облако
- Гибридное облако
- Общественное облако

Модели обслуживания:

- ПО как услуга
- Платформа как услуга
- Инфраструктура как услуга

Профит:

- Гибко реагировать на изменения вычислительных потребностей,

используя свойства вычислительной эластичности облачных услуг.
- Объединение ресурсов: много и разных ресурсов объединены, задачи между ними динамически распределяются.

Технологии: виртуализация.

То, что когда-то говорил Максим Александрович:

Облачные вычисления - сервис для вычислений, который предоставляет некий QoS для пользователя, при этом структура и внутренняя организация этого сервиса значения не имеет.

Т.е. определение дается чисто с точки зрения пользователя. Внутри себя облачный сервис может быть и гридом, и кластером и чем угодно еще. Хоть нетбуком.

8. Синхронизация в распределенных системах(http://parallel.ru/krukov/lec4.html)

1. Алгоритм Лэмпарта для синхронизации часов.
2. Алгоритм для выбора координатора
   - Алгоритм “задиры”
   - Круговой алгоритм
3. Алгоритм организации взаимного исключения
   - Централизованный алгоритм
   - Алгоритм с круговым маркером
   - Алгоритм маркерный древовидный
   - Децентрализованный на основе временных меток
   - Широковещательный маркерный (маркер содержит очередь запросов и массивчик с номерами последних запросов для каждого процесса)

9. Отказоустойчивость(http://parallel.ru/krukov/lec7.html)

1. Восстановление после отказа
Проблемы:

- Сообщения сироты и эффект домино
- Потеря сообщений
- Проблема бесконечного восстановления

Алгоритмы:

- Синхронная фиксация чекпоинтов и восстановление (процесс говорит “давайте сохранимся”, если все согласны - сохраняемся, нет - значит нет)
- Откат (процесс говорит “давайте откатимся”, если все согласны - откатываемся, нет - значит нет)
- Асинхронная фиксация чекпоинтов (все процессы время от времени независимо делают чекпоинты, при откате производится поиск консистентного множества)

2. Отказоустойчивость

- Горячий резерв
- Активное размножение
- Принятие единого решения (невозможно при ненадежных коммуникациях)
- Принятие согласованного решения
- Надежные неделимые широковещательные сообщения

10. Достоинства многопроцессорных систем. Достоинства распределенных систем. Виды операционных систем.(http://parallel.ru/krukov/lec1.html)

Распределенная система - совокупность независимых компьютеров, которая представляется пользователю как единый компьютер.

Почему создаются распределенные системы? В чем их преимущества перед централизованными ЭВМ? Причины:
1. Экономическая.
2. Можно достичь производительности недостижимой в централизованных компьютерах.
3. Естественная распределенность (напр., банк).
4. Надежность.
5. Наращиваемость производительности.

Недостатки распределенных систем:
1. Проблемы ПО (приложения, языки, ОС).
2. Проблемы коммуникационной сети (напр., потери).
3. Секретность.

	Сетевая ОС	Распределенная ОС	ОС мультипроцессора
Компьютерная система выглядит как виртуальная однопроцессорная ЭВМ	НЕТ	ДА	ДА
Одна и та же ОС выполняется на всех процессорах	НЕТ	ДА	ДА
Сколько копий ОС имеется в памяти	N	N	1
Как осуществляются коммуникации	Разделяемые файлы	Сообщения	Разделяемая память
Требуется ли согласованный сетевой протокол	ДА	ДА	НЕТ
Имеется ли единая очередь выполняющихся процессов	НЕТ	НЕТ	ДА
Имеется хорошо определенная семантика разделения файлов	Обычно НЕТ	ДА	ДА

Принципы построения распределенных ОС: прозрачность, гибкость, надежность, производительность, масштабируемость.

11. Понятие веб-сервиса. Понятие удаленного вызова процедур, проблемы реализации удаленных вызовов. Концепция RESTful сервисов. Примеры RESTful-сервисов в IT-практике.

Remote Procedure Call (RPC) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве (как правило, на удалённых компьютерах).

Характерными чертами вызова локальных процедур являются:

- Асимметричность, то есть одна из взаимодействующих сторон является инициатором;
- Синхронность, то есть выполнение вызывающей процедуры при останавливается с момента выдачи запроса и возобновляется только после возврата из вызываемой процедуры.

Идея, положенная в основу RPC, состоит в том, чтобы сделать вызов удаленной процедуры выглядящим по возможности также, как и вызов локальной процедуры. Другими словами - сделать RPC прозрачным: вызывающей процедуре не требуется знать, что вызываемая процедура находится на другой машине, и наоборот.

Проблемы:

- Реализация этой самой “прозрачности”: через заглушки.
- Передача аргументов. Нет передачи “по ссылке”, сериализация.
- Обработка отказов сервера и клиента.

RESTful-сервисы (*Representational State Transfer*, «передача состояния представления»)

- **Client-server**. Клиент и сервер имеют общий протокол общение и больше их ничего не связывает, они могут жить, изменяться и разрабатываться независимо. Профит: масштабируемость.
- **Stateless**. Никакая информация на сервере между вызовами не сохраняется. Надёжность, масштабируемость.
- **Cacheable**. Клиент может кэшировать ответы на запросы. Производительность.
- **Layered system**. (ась?). Клиент может связываться с сервером напрямую или же не напрямую. Расширяемость за счёт балансировки нагрузки, безопасность при желании.
- **Unifrom interface**. Вроде в первом пункте описано.
- **Code on demand**. Сервер может временно расширять возможности клиента, отправляя ему исполняемый код.

A **Web service** is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

Интероперабельность (англ. interoperability — способность к взаимодействию) — это способность продукта или системы, интерфейсы которых полностью открыты, взаимодействовать и функционировать с другими продуктами или системами без каких-либо ограничений доступа и реализации.