

# Bigdata

## Лекция I: распределенные файловые системы

Дмитрий Барашев  
bigdata@barashev.net

Computer Science Center

21 февраля 2013

# Сегодня в программе

Основные концепции

Особенности распределенных ФС

Немного истории

Google File System

Тема для семинара

# Сегодня в программе

Основные концепции

Особенности распределенных ФС

Немного истории

Google File System

Тема для семинара

# Файловая система

- ▶ Модель данных, программные компоненты, персистентные структуры данных и API
- ▶ Предоставляет абстракцию для доступа к данным, находящимся на каком-то физическом носителе
- ▶ Традиционная модель
  - ▶ Файл: объект с именем и бессмысленным для ФС содержанием
  - ▶ Каталог: список файлов и вложенных каталогов
  - ▶ Каталоги и файлы образуют дерево – *пространство имен*
  - ▶ Файл уникально идентифицируется путем:  
`/usr/bin/vim`

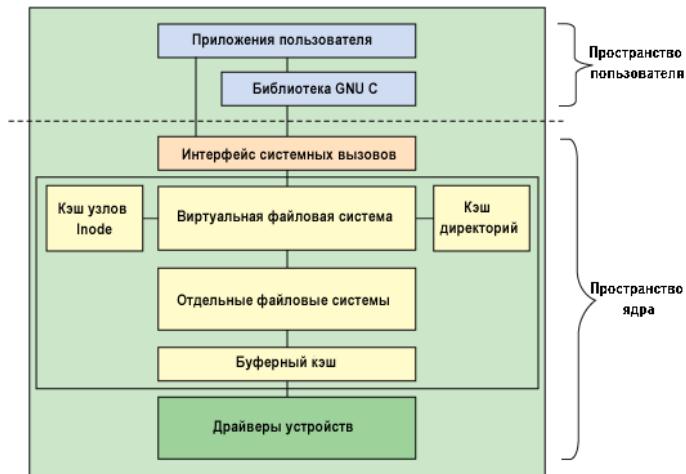
# Метаинформация

- ▶ Для пользователя информация – это содержание файлов
- ▶ Для ФС интереснее метаянформация: название файла, список блоков, время модификации, права доступа

# Локальные файловые системы

- ▶ Более или менее тесно интегрированы с ядром
- ▶ Обычно хранят данные на локальном HDD
- ▶ Блоки размером несколько килобайт
- ▶ Могут кешировать страницы

# Локальные ФС: Linux



картинка с IBM developerWorks

# Сегодня в программе

Основные концепции

Особенности распределенных ФС

Немного истории

Google File System

Тема для семинара



# Распределенная ФС

- ▶ Модель примерно та же
- ▶ Компоненты распределены по разным машинам
- ▶ Распределенность существенно влияет на некоторые решения

# Компоненты РФС

- ▶ *Клиент*: API для прикладных приложений и код для коммуникации с сервером
- ▶ *Серверы файлов*: хранят содержимое файлов
- ▶ *Серверы метаданных*: знают, какой файл где лежит и многое еще

# Аспекты функционирования

- ▶ Прозрачность размещения файлов
- ▶ Совместный доступ
- ▶ Кэширование
- ▶ Репликация
- ▶ Единая точка отказа
- ▶ Наличие состояния
- ▶ Шаблоны доступа
- ▶ Масштабируемость

# Прозрачность размещения файлов

- ▶ Прикладному приложению известен только путь
- ▶ Чем меньше информации о физическом расположении закодировано в путь, тем лучше
- ▶ ...при сохранении здравого смысла

## Пример 1

`/192.168.0.10/sda1/home/alice/kitten.jpg`

тут пожалуй информации многовато

## Пример 2

`/TheEarth/home/alice/kitten.jpg`

а тут ФС придется самостоятельно выбрать континент

# Совместный доступ и кеширование

- ▶ Централизованная система: атомарные чтение и запись; блокировки; журналирование
- ▶ Распределенная ФС: сетевые задержки, репликация усложняют жизнь

# Варианты управления совместным доступом

- ▶ синхронные чтение и запись
- ▶ *write-through cache*: чтение из кеша, синхронная запись
- ▶ *сессионное кеширование*: чтение и запись в кеш, синхронизация при закрытии, политика определения победителя при конкурирующей записи
- ▶ файлы после создания становятся неизменяемыми
- ▶ запись возможна только в конец (append-only)
- ▶ клиенты, открывшие файл уведомляются о записях
- ▶ полноценные транзакции

# Репликация

- ▶ Синхронная или асинхронная
- ▶ Политика согласованности реплик
- ▶ Запись в реплики

# Единая точка отказа

- ▶ Сбой в единой точке отказа (*Single Point of Failure*) делает неработоспособной всю систему
- ▶ Сервер метаданных – кандидат на SPoF
- ▶ ...и еще и на узкое место
- ▶ Два сервера метаданных – кандидаты на несогласованность



# Наличие состояния

- ▶ Сервер с состоянием (*stateful*) знает все про открытые файлы в данный момент
  - ▶ тратит на это память и больно падает
  - ▶ но зато может кое-какие операции оптимизировать
- ▶ Сервер без состояния (*stateless*) возможно будет повторять действия при каждом запросе
  - ▶ но зато быстро восстановится после падения
- ▶ У сервера метаданных всегда есть состояние

# Шаблоны доступа

- ▶ Какого размера типичный файл?
- ▶ Что важнее – надежность или скорость?
- ▶ Что важнее – среднее время одного случайного чтения или суммарная пропускная способность последовательного чтения?

# Масштабируемость

- ▶ Хочется линейную
  - ▶ было  $N$  дисков и  $K$  машин
  - ▶ стало в  $2N$  данных – добавили  $N$  дисков, сохранили пропускную способность
  - ▶ нужна в два раза большая пропускная способность – добавили  $K$  машин, распределили данные
- ▶ На практике у линейной масштабируемости множество препятствий
  - ▶ пропускная способность сети, сетевых интерфейсов файловых серверов, производительность сервера каталогов, блокировки

# Сегодня в программе

Основные концепции

Особенности распределенных ФС

**Немного истории**

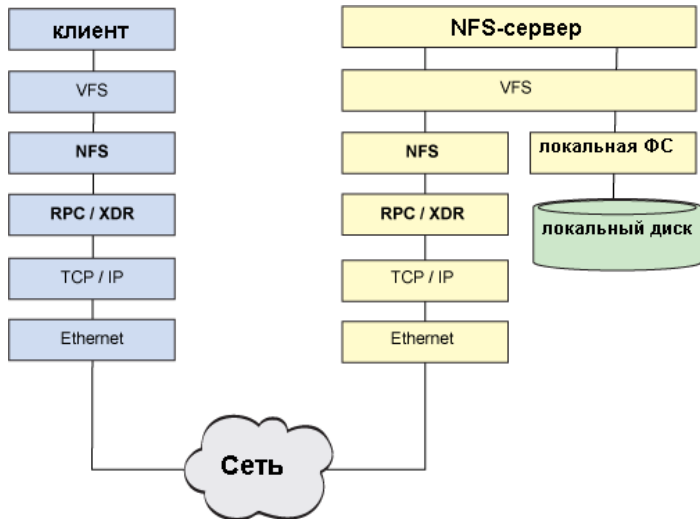
Google File System

Тема для семинара

# NFS

- ▶ Network File System
- ▶ Рождена Sun'ом в начале 1980-х и жива-здорова до сих пор, используется во многих корпорациях
- ▶ POSIX API, клиент монтирует удаленный диск в локальный каталог
- ▶ На сервере NFS демон тоже работает со стандартным интерфейсом файловой системы
- ▶ Поддерживаются блокировки и сессионное кеширование

# NFS



*картинка с IBM developerWorks*

# AFS

- ▶ Andrew File System. Рождена в университете Carnegie Mellon в 1980-х
- ▶ Сессионное кеширование, нотификации об изменениях, блокировки файлов
- ▶ Моментальные read-only снимки томов
- ▶ Не POSIX API

## и другие

- ▶ CIFS, aka Samba, aka Windows Shared Folders
- ▶ Ceph, GlusterFS, Lustre, <add your file system here>



# Сегодня в программе

Основные концепции

Особенности распределенных ФС

Немного истории

**Google File System**

Тема для семинара

# Предпосылки

- ▶ Начало 2000-х, Google завоевывает поиск
- ▶ Большие файлы (порядка N Gb) записываются и читаются пакетными процессами (crawler, indexer)
- ▶ Пропускная способность важнее быстрого случайного доступа
- ▶ Ширпотребные компьютеры, в совокупности часто выходящие из строя

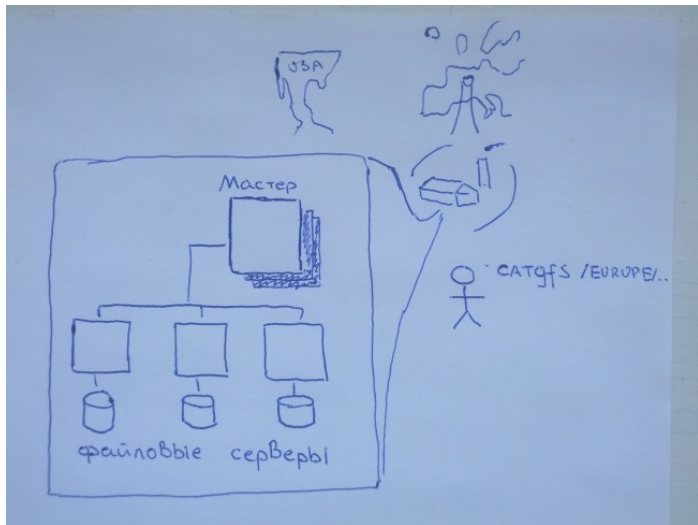
# Основные моменты архитектуры

- ▶ Много файловых серверов, один активный сервер метаданных (мастер)
- ▶ Файлы хранятся фрагментами по 64Мб
- ▶ Каждый фрагмент реплицируется на 3 различных файловых сервера
- ▶ Приоритетные операции с файлом: большое последовательное чтение и конкурентное наращивание
- ▶ Кеширования на клиенте не производится
- ▶ POSIX API не поддерживается

# Развертывание GFS

- ▶ Ячейка - единица развертывания
- ▶ В ячейке один мастер и много файловых серверов
- ▶ Ячейка GFS примерно соответствует физическому датацентру

# Архитектура GFS-ячейки



# Обязанности мастера

- ▶ Поддерживать пространство имен и его отображение во фрагменты
- ▶ Обзвон файловых серверов, «проверка связи», выдача указаний, сбор состояния
- ▶ Размещение фрагментов при их создании, дополнительной репликации или перебалансировке
- ▶ Пересылка данных, однако, осуществляется напрямую между репликами и/или клиентом

# Мутации метаданных

- ▶ Метаданными управляет мастер
- ▶ Теневые серверы дублируют его действия
- ▶ Изменения метаданных атомарны, изолированы и долговечны
  - ▶ в пространстве имен используются иерархические блокировки
  - ▶ мутации метаданных журналируются и журнал реплицируется на теневых серверах

# Взаимодействие клиента и мастера при чтении

- ▶ Приложение собирается прочитать фрагмент
- ▶ GFS библиотека звонит мастеру, тот возвращает адреса *реплик* – файловых серверов, хранящих фрагмент
- ▶ GFS библиотека напрямую звонит одному из файловых серверов с просьбой вернуть нужный диапазон внутри данного фрагмента
- ▶ Дальнейшее общение клиента и файлового сервера идет напрямую



## Взаимодействие при записи

1. Приложение хочет записать данные в фрагмент, хранящийся на нескольких репликах
2. Мастер выбирает главную по записи среди всех реплик
3. Клиент получает адреса всех реплик и передает им данные
4. Когда все реплики получили данные, клиент посылает главной указание произвести запись
5. Главная реплика выбирает порядок применения мутаций, применяет их локально и рассылает репликам указание применить мутации в том же порядке
6. Если все хорошо то клиент счастлив

## Взаимодействие при записи

1. Приложение хочет записать данные в фрагмент, хранящийся на нескольких репликах
2. Мастер выбирает главную по записи среди всех реплик
3. Клиент получает адреса всех реплик и передает им данные
4. Когда все реплики получили данные, клиент посылает главной указание произвести запись
5. Главная реплика выбирает порядок применения мутаций, применяет их локально и рассылает репликам указание применить мутации в том же порядке
6. Если все хорошо то клиент счастлив

А что если не все хорошо?

# Модель согласованности

- ▶ Характеристики байтового региона: согласованный и определенный
- ▶ Регион *согласован*, если у него одинаковое значение во всех репликах
- ▶ Регион *определен* после записи, если он согласован и клиенты видят ровно те данные, которые были записаны
- ▶ Успешная запись при отсутствии конкурирующих записей производит определённый регион
- ▶ Конкурирующие успешные записи могут произвести согласованные, но неопределённые данные

# Атомарная операция наращивания

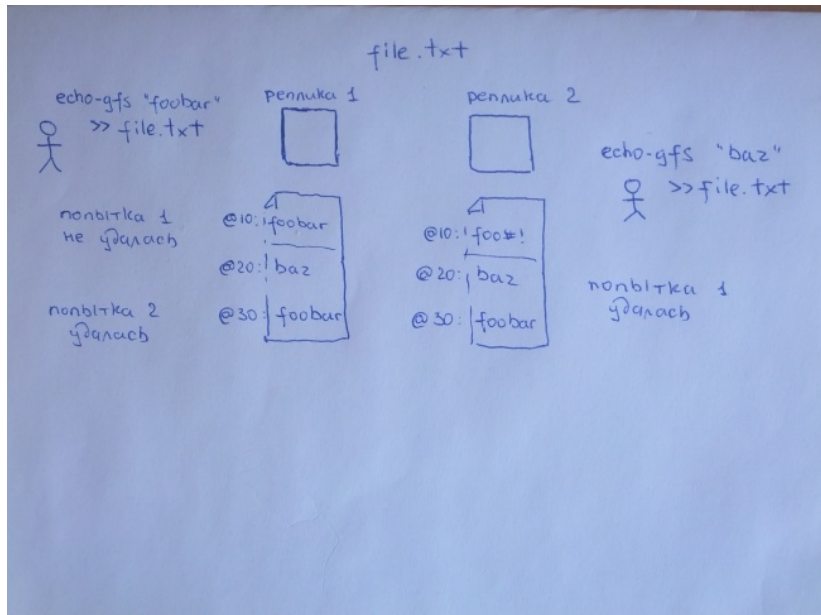
- ▶ Гарантия: успешная операция наращивания производит согласованный определённый регион, смещение которого определяет GFS
- ▶ В «хорошем» случае все почти как в операции произвольной записи
  - ▶ Главная реплика назначает смещение и может попросить произвести запись в следующий фрагмент, если в текущем нет места
- ▶ В «плохом» случае клиент повторяет операцию, и главная реплика назначает новое смещение
- ▶ Результат: в некоторых репликах могут быть дубликаты, полные или частичные добавляемых данных, но рано или поздно появится согласованный определённый регион

# Атомарная операция наращивания

- ▶ Гарантия: успешная операция наращивания производит согласованный определённый регион, смещение которого определяет GFS
- ▶ В «хорошем» случае все почти как в операции произвольной записи
  - ▶ Главная реплика назначает смещение и может попросить произвести запись в следующий фрагмент, если в текущем нет места
- ▶ В «плохом» случае клиент повторяет операцию, и главная реплика назначает новое смещение
- ▶ Результат: в некоторых репликах могут быть дубликаты, полные или частичные добавляемых данных, но рано или поздно появится согласованный определённый регион

Реплики фрагмента не являются бинарно идентичными!

# Схема наращивания



# Неопределённые данные

- ▶ И произвольная запись и наращивание могут произвести неопределённые и несогласованные регионы
- ▶ Выяснение осмысленности хранящихся в регионе данных становится заботой приложения
  - ▶ контрольные суммы записи
  - ▶ контрольные точки в файле

# Время жизни главной реплики

- ▶ Главная реплика назначается мастером и получает билет (*lease*) на 60 секунд
- ▶ Если билет просрочен, главная реплика не имеет права осуществлять операции записи
- ▶ Билет можно продлевать
- ▶ Каждый новый билет, выданный главной реплике, увеличивает номер версии фрагмента
- ▶ Мастер может отобрать билет раньше срока



# Операция фиксации состояния

- ▶ Фиксация состояния (*snapshot*) делает почти мгновенную копию файла или каталога методом *copy-on-write*
  - ▶ отбираются все выданные билеты
  - ▶ делается копия метаданных дерева
  - ▶ новые файлы ассоциируются с оригинальными фрагментами
- ▶ Когда кто-то захочет изменить данные, он запросит адрес главной реплики фрагмента
- ▶ В этот момент мастер распорядится создать фрагмент-копию

# Проблемы файлового сервера

- ▶ Данные на файловом сервере могут повредиться из-за аппаратного или программного сбоя
- ▶ ФС может пропустить мутацию фрагмента и его фрагмент устареет (номер версии меньше чем на мастере)

# Целостность данных

- ▶ Фрагменты разбиваются на блоки размером 64Kb и для каждого блока считается контрольная сумма
- ▶ Контрольные суммы хранятся в памяти и журналируются отдельно от данных
- ▶ При чтении файловый сервер сравнивает КС блока с записанной и в случае противоречия бьёт в набат

# Устаревшие фрагменты, удаление файлов и сборка мусора

- ▶ Фрагмент может устареть
- ▶ Стратегия удаления
  - ▶ отметить как удалённый и переместить в Trash, записав момент удаления
  - ▶ через некоторое время удалить из метаданных совсем
- ▶ Стратегия сбора мусора
  - ▶ Файловые серверы сообщают мастеру ID хранимых фрагментов
  - ▶ Мастер смотрит в свои метаданные: отображение файла во фрагменты
  - ▶ Первое множество без второго = мусор
  - ▶ Файловый сервер удаляет мусор по своему усмотрению

# Требования меняются

- ▶ Google 2010-х годов – это интерактивные приложения
- ▶ Файлы меньше в размерах и больше в количестве
- ▶ Речь уже о пета и эксабайтах
- ▶ Требования по времени произвольного доступа жестче

# Требования меняются

- ▶ Google 2010-х годов – это интерактивные приложения
- ▶ Файлы меньше в размерах и больше в количестве
- ▶ Речь уже о пета и эксабайтах
- ▶ Требования по времени произвольного доступа жестче

GFS в Google больше не используется, на смену пришел Colossus

# Реализации, похожие на GFS

- ▶ Реализация Google File System закрыта
- ▶ Открытые проекты, с GFS-like архитектурой:
  - ▶ Apache HDFS: реализация на Java из проекта Hadoop
  - ▶ QFS: реализация на C++

# ФС без сервера метаданных

- ▶ Применяем хеш к полному имени файла и получаем номер(а) файловых серверов
- ▶ Если файл переименовывается, он оставляет свой новый адрес на старом месте
- ▶ Если нужно увеличить число файловых серверов, используем схему, похожую на extensible hashing
- ▶ Реализация: GlusterFS



# Сегодня в программе

Основные концепции

Особенности распределенных ФС

Немного истории

Google File System

Тема для семинара

# Недостатки репликации

- ▶ N реплик – это конечно хорошо, но ...
- ▶ Диска тратится в N раз больше
- ▶ Выхода из строя всего N машин достаточно чтобы потерять 1 фрагмент

# Алгоритмы коррекции ошибок

- ▶ Уменьшение избыточных данных
- ▶ Повышение надежности: чтоб потерять данные нужно вывести из строя существенно больше машин

Reed-Solomon encoding:  $n$  исходных дисков +  $m$  контролирующих гарантируют восстановление при выходе из строя любых  $k \leq m$  дисков

# Тема для семинара

- ▶ Рассказать о применении кодировки Рида-Соломона для реализации хранилища, устойчивого к сбоям
- ▶ +5 баллов в копилку!

Эта презентация сверстана в

**PAPERIA**

L<sup>A</sup>T<sub>E</sub>X в вашем браузере  
[alpha.paperia.com](http://alpha.paperia.com)

# Литература I



Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung.

The Google file system.

In *ACM SIGOPS Operating Systems Review*,  
volume 37, pages 29–43. ACM, 2003.



M. Tim Jones.

NFS: удобная и перспективная сетевая файловая  
система.

[http://www.ibm.com/developerworks/ru/library/  
l-network-fileystems/index.html/](http://www.ibm.com/developerworks/ru/library/l-network-fileystems/index.html/).

Accessed: 23.01.2013.





M. Tim Jones.

Анатомия файловой системы Linux.

[http://www.ibm.com/developerworks/ru/library/  
l-linux-fileystem/](http://www.ibm.com/developerworks/ru/library/l-linux-fileystem/).

Accessed: 23.01.2013.

# Литература II

-  James S. Plank et al.  
A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems.  
*Software Practice and Experience*, 27(9):995-1012, 1997.
-  В.Г. Олифер and Н.А. Олифер.  
*Сетевые операционные системы*.  
Питер, 2002.