BIGDATA

Лекция 1

ЧТО ИЗУЧАЕМ

- Работа с большими данными
- Скорее в аналитическом аспекте
- Которые могут какое-то время храниться
- Экосистема Hadoop

ТЕХНИЧЕСКАЯ СОСТАВЛЯЮЩАЯ

- Внутри - Java
- Органичнее всего - работать через JVM-языки
- Но есть адаптации к Python
- Домашки можно сдавать на Python/Java/Scala

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- 2-3 домашки
- Каждая оценивается на 100 баллов
- На балл влияют: полнота решений, качество (по ревью), соблюдение дедлайнов
- Считаем среднюю по домашкам
- В итоговую оценку идет с коэффициентом 0.45

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- В начале семинаров - летучки
- Вопросы на степике
- На выбор или короткие текстовые
- В итоговую оценку идет с коэффициентом 0.2

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- Экзамен - как беседа
- Что-то спрошу по домашкам, попрошу кусочек кода поревьюить
- Что-то спрошу на понимание
- Пять вопросов, по 20 баллов каждый
- В итоговую оценку идет с коэффициентом 0.35

МОЖНО БЕЗ ЭКЗАМЕНА

- Первый способ - получить высокие баллы за домашки
- Тогда 0 за экзамен не помешает зачету
- Но балл будет низкий
- Второй способ - получить право на автомат
- Выдается за высокую (и разумную) активность на семинаре
- На фоне высоких показателей по домашкам и тестам

ПОДРОБНЕЕ ПРО ЛЕТУЧКИ

- Проходят здесь и сейчас
- Досдач без уважительных причин не предусмотрено
- Ответы разбираются сразу
- При уважительной причине возможна индивидуальная пересдача
- Коллективно в конце семестра

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Дедлайн никто не меряет на секунды
- По мере пропущенных дней набегает понижающий коэффициент
- 1 день - 0.95
- 2 дня - 0.9
- 14 дней 0.3
- Понижается в день на 0.05

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Через 14 дней фиксируется
- Остается 0.3 навсегда
- В случае ухода на пересдачу - не восстанавливается !!!
- Мораль: не забрасывайте домашки !!!

ПОДРОБНЕЕ ПРО ДОМАШКИ

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Если тесты пройдены или не предусмотрены, то начинается code review
- По итогам review балл может быть понижен
- Как правило, замечания можно исправлять
- Как правило, со скидкой
- Как правило, одна итерация на исправление

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Возможны бонусы
- За активность на семинаре, доделки заданий с семинаров (будет объявляться)
- За особо красивые решения в тестах
- Бонусные баллы добавляются к релевантным домашкам, но не выше 100 баллов
- Переноса на другие домашки нет

ИНДИВИДУАЛЬНЫЕ ПЕРЕНОСЫ ДЕДЛАЙНОВ

- Каждый имеет право переноса дедлайна
- Лимит - 4 дня
- Гранулярность - 1 день
- Надо сообщить до общего дедлайна

АНТИПАТТЕРНЫ

- Появиться внезапно в конце семестра в расчете на особые условия
- Заявить задним числом, что "ничего не было понятно"
- Для тех, кто переводится - воспринимать перевод как универсальную индульгенцию
- Аппелировать к правилам других курсов
- Молча исчезнуть в середине семестра - лучше дать обратную связь

ПРАВИЛЬНЫЕ ПАТТЕРНЫ

- Вовлеченно участвовать в семинарах
- Задавать вопросы
- Давать конструктивную обратную связь

МОИ ОБЯЗАТЕЛЬСТВА

- Прислушиваться к обратной связи - в рамках реального
- Отвечать на вопросы и обратную связь - в рамках разумного
- Давать намеки и разумные подсказки по мере необходимости
- Незначительно корректировать критерии в сторону смягчения

ПЛАН НА СЕГОДНЯ

- Введение в проблему
- Общие проблемы распределенных систем
- Необходимый ликбез по железу

ОПРЕДЕЛЕНИЕ ПРЕДМЕТА

- Какие данные считаем большими данными ?
- Представим +/- типовую конфигурацию отдельной машины
- 16-32G RAM, 1-2Tb SSD, 8-16 ядер
- То, что в нее не влезает - будем считать большими данными

ОЦЕНИМ ГРУБО И КОНСЕРВАТИВНО

- Сервис на 1 миллион активных пользователей
- В день активный пользователь порождает 50Kb данных: сообщения, логи (покупки, просмотры)
- 50Gb в день
- За пару месяцев диск съедим
- Память и процессоры на запросы - отдельный вопрос

ВАРИАНТЫ РЕШЕНИЯ

- Вертикальное масштабирование - мощнее процессор, больше памяти, больше дисков
- Плохо масштабируется
- Не гибко
- Альтернатива - горизонтальное масштабирование
- Кластер из нескольких вычислителей

РАЗДЕЛЕНИЕ РОЛЕЙ

- Репликация - повторение данных на разных узлах
- На первый взгляд - не решает исходную проблему
- Но обеспечивает непрерывное функционирование кластера
- Часто применяется в дополнение к другим схемам
- Повышает надежность

РАЗДЕЛЕНИЕ РОЛЕЙ

- Шардирование - на каждом узле лежит часть данных
- Все шарды вместе хранят все данные
- Можно сочетать с репликацией
- У каждого шарда - несколько реплик

РАЗДЕЛЕНИЕ РОЛЕЙ

- Есть и другие варианты
- Не за всеми закрепились лаконичные названия
- В HDFS есть репликация
- Есть некое распределение данных
- Но это не классическое шардирование

ПРОБЛЕМЫ В КЛАСТЕРЕ

- Кажущееся простым становится нетривиальным
- Пример: алгоритм отправки сообщения с подтверждением получения
- Как следствие - непросто поддерживать целостное состояния
- Например: чтобы после изменения одной реплики новое состояние читалось с других реплик

ПРОБЛЕМЫ В КЛАСТЕРЕ

- Можно при записи дожидаться переноса данных на все реплики
- Надежно, но медленная запись и ожидания в параллельных чтениях
- Другая крайность - быстро записать на одну реплику и начать переносить на другие
- И не мешать прочитать

ТЕРМИНЫ

- Целостность/consistency/консистентность - система дает одинаковый ответ при запросах через разные узлы
- Доступность/availability - способность системы давать разумный ответ за разумное время
- То есть данные не с потолка
- "Пустой" ответ - только при физической невозможности получить данные

CAP ТЕОРЕМА

- Целостность и доступность в какой-то мере антагонистичны друг другу
- Но их все-таки можно совместить
- Но пострадает третье свойство - устойчивость к сбоям в связности (partition tolerance)
- Способность сохраняться в условиях сетевых сбоев

CAP ТЕОРЕМА

- Утверждается, что система не может одновременно реализовать все три свойства в полной мере
- Утверждение не является теоремой в смысле математики или теоретической информатики
- Нет формального доказательства, понятия определены неформально
- Но есть консенсус экспертов

CAP ТЕОРЕМА

- В первом приближении - есть три типа систем: CA, AP и CP
- Классификация конкретной системы бывает дискуссионной
- Иногда в разных конфигурациях она попадает в разные категории
- Классические SQL-кластеры - CA или CP

CAP ТЕОРЕМА

- Некоторые свойства могут иметь градации
- Есть много разновидностей неполной консистенстности
- Популярный вариант: eventual consistency
- CRDT: conflict-free replicated data type

ПОЛЕЗНОЕ ЧТЕНИЕ

- Википедия: CAP theorem, eventual consistency
- Статьи на хабре

STACKOVERFLOW

- CAP theorem - Availability and Partition Tolerance
- Why isn't RDBMS Partition Tolerant in CAP Theorem and why is it Available?
- Where does mongodb stand in the CAP theorem?

ЛИДЕР

- Управление кластером облегчается наличием узла-лидера
- Он может определять зоны ответственности шардов
- Или отслеживать наличие достаточного числа реплик

ЛИДЕР

- Его участие может требоваться при установлении сеанса с клиентом
- Лидер может быть фиксированным или же меняться

АЛГОРИТМЫ КОНСЕНСУСА

- Фиксированный лидер - угроза устойчивости системы
- Смена лидера - частный случай применения алгоритма консенсуса
- Консенсус - единое понимание всеми узлами какого-либо состояния

АЛГОРИТМЫ КОНСЕНСУСА

- Консенсус критичен применительно к метаданным системы
- Алгоритмы распределенного консенсуса: Paxos, Raft

АРХИТЕКТУРА HADOOP

- Несущая конструкция - HDFS (Hadoop Distributed File System)
- Для клиента - файловый интерфейс
- Базовые операции - открытие, создание, добавление, чтение
- Нет записи в произвольное место существующего файла

АРХИТЕКТУРА HADOOP

- Два типа узлов: NameNode и DataNode
- NameNode - это статически заданный лидер
- (Да, это плохо, но так сложилось)
- DataNode - узел для хранения данных

Блоки HDFS

- Не путаем блоки обычных файлов и блоки HDFS-файлов
- Блок обычного файла - минимальная единица обмена данными с диском
- Обычно - где-то 512 байт

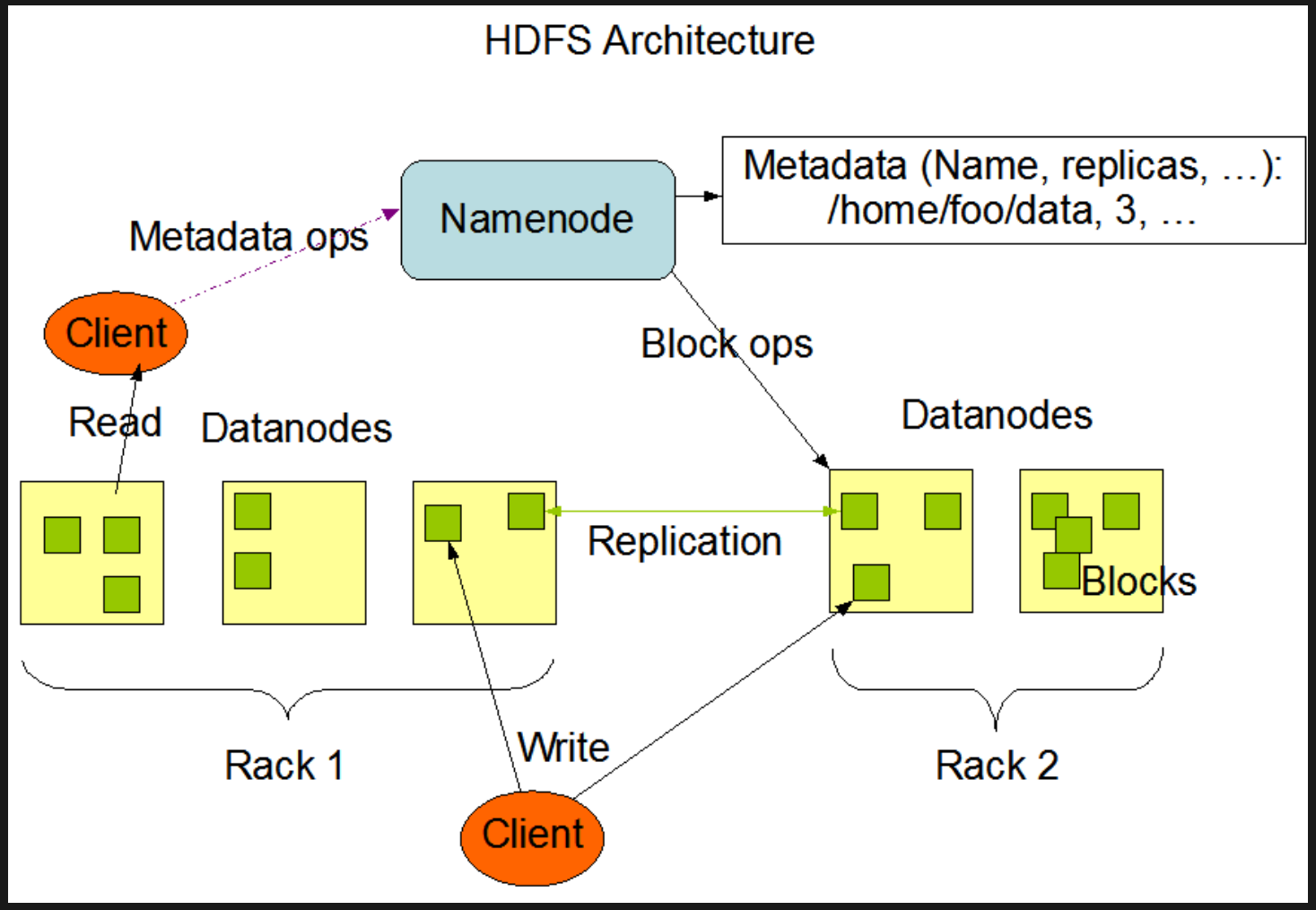
Блоки HDFS

- Иногда еще блоком называют фрагмент данных, оптимальный с точки зрения обмена с диском
- Тогда это может быть порядка мегабайта
- В обоих вариантах это более низкоуровневое понятие по сравнению с файлом

Блоки HDFS

- А блок HDFS - более низкоуровневое по отношению к HDFS-файлу
- Но сам он материализуется в обычном файле
- И в этом смысле - он не "равновелик" блоку в смысле обычной файловой системы
- Размер блока HDFS - 128M
- (Но неполный блок не займет 128M - и в этом тоже отличие от дискового блока)

Пример


HDFS Architecture
Namenode
Metadata (Name, replicas, …):
/home/foo/data, 3, …
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Почему именно такой размер

- Есть мотив сделать блок побольше
- Потому что система распределенная
- И чтение блока с другого узла требует значимой константной издержки
- И переменная часть издержек должна доминировать над константной

Почему именно такой размер

- И есть мотив сделать блок не совсем большим
- Потому что данные на многих блоках легче параллельно обрабатывать
- Как минимум потому что они лежат на физически разных машинах
- И 128M - это компромисс между двумя желаниями

Процесс создания и наполнения файла

- Клиентская библиотека обращается к NameNode-у
- Если файл только создается, NameNode определяет, куда разместить первый блок
- И сообщает клиентской библиотеке
- Если файл уже есть, NameNode знает, где лежит последний блок
- И тоже сообщает клиенту

Процесс создания и наполнения файла

- "Куда" - необязательно один DataNode
- Скорее - несколько. Зависит от фактора репликации
- Обычно 2-3 DataNode-а
- Клиент устанавливает TCP-сеанс с одним из них
- Передает ему данные "пакетами"
- И список DataNode-ов

Процесс создания и наполнения файла

- DataNode пишет данные на диск
- И передает данные дальше
- Куда - он знает из списка
- И так далее
- С каждой передачей список уменьшается

Процесс создания и наполнения файла

- Последний DataNode в конвейере после записи передает подтверждение предыдущему
- Тот - своему предыдущему
- В итоге подтверждение доходит до клиентской библиотеки

Процесс создания и наполнения файла

- Пакет считается записанным
- Запись и отправка дальше по конвейеру хорошо распараллеливаются
- Отдельно обрабатывается ситуация отказа одного из узлов

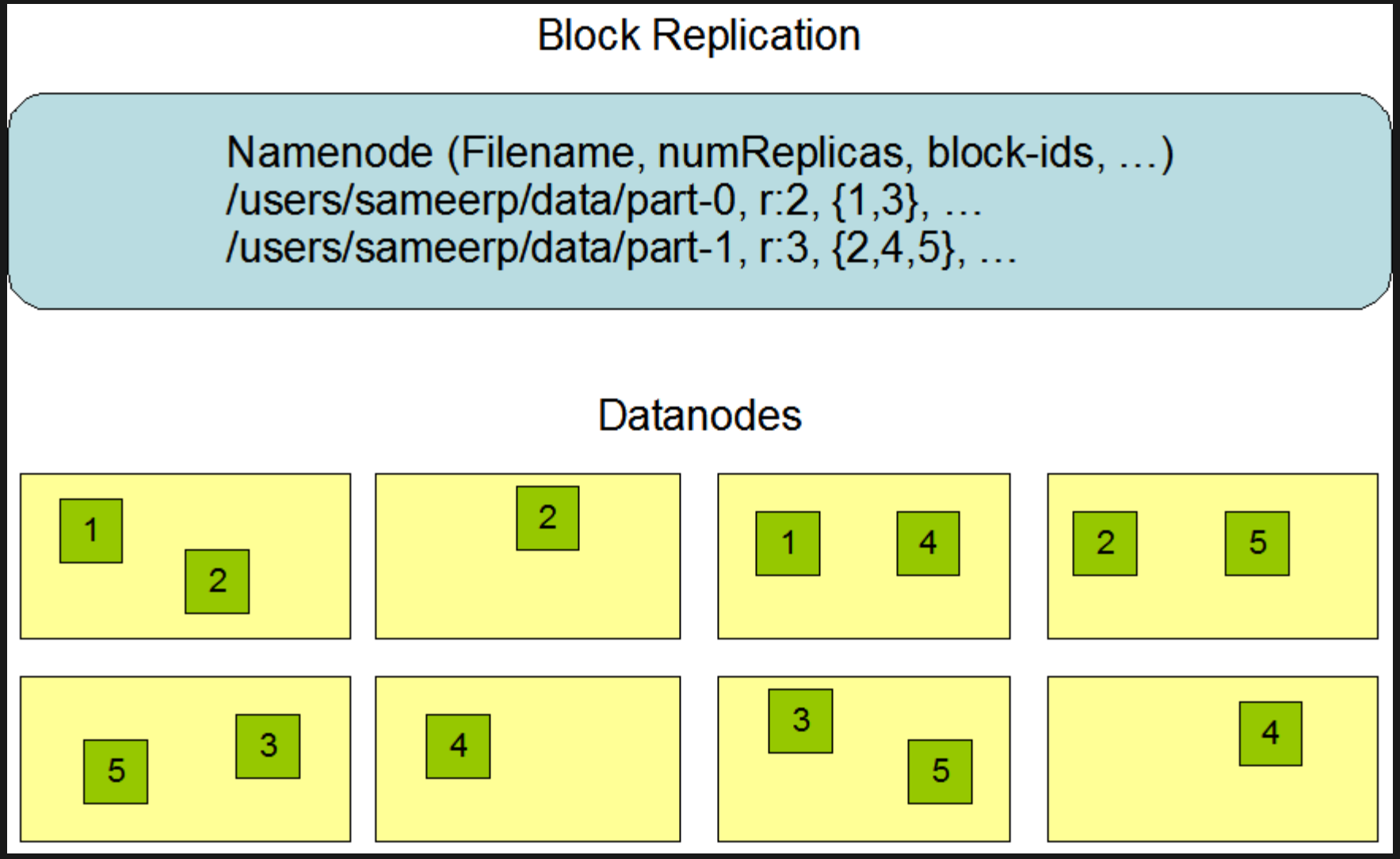
Процесс создания и наполнения файла

- Тут подключается NameNode
- Предоставляет альтернативный DataNode
- И он не только включается в очередь взамен упавшего

Процесс создания и наполнения файла

- На него еще переносится с других узлов то, что уже было записано на упавший
- (Дополнительный фактор для ограничения роста размера блока)
- Когда блок заканчивается - запрашивается NameNode, куда помещать следующий

Пример


Block Replication
Namenode (Filename, numReplicas, block-ids, …)
/users/sameerp/data/part-0, r:2, {1,3}, …
/users/sameerp/data/part-1, r:3, {2,4,5}, …
Datanodes
1
2
2
1
4
2
5
5
3
4
3
5
4

Процесс чтения файла

- Проще, чем запись
- Клиентская библиотека запрашивает у NameNode-а координаты первых блоков
- И читает с одного из DataNode-ов
- Если с ним проблемы, переходит на альтернативный
- По мере необходимости запрашивает у NameNode-а данные про очередную порцию блоков

Глазами клиента

- Пользуется клиентской библиотекой
- Открывает файл
- Читает, пишет - как обычный файл
- С учетом описанных ограничений
- Все подробности - под капотом

NameNode

- Участвует в любом открытии или создании файла
- К нему иногда обращаются за дополнительной информацией
- И он мониторит состояние узлов
- Но непосредственно в обмене данными - он не участвует
- Single Point of Failure - и это серьезная проблема

DataNode

- Напрямую контактирует с клиентской библиотекой
- Сообщает NameNode-у о том, что он жив
- И о проблемах с диском
- Участвует в обработке выпадения узлов
- Участвует в конвейере при записи данных

Клиентская библиотека

- Играет важную роль
- Участвует в протоколе чтения и (в особенности) записи
- Сторонняя реализация - крайне амбициозная и нетривиальная задача
- Есть стандартная JVM-версия
- Есть официальная нативная версия

Расположение новых блоков

- Когда создается файл, надо понять, на каких узлах размещать блоки
- Если несколько вариантов разумных стратегий
- С одной стороны, хотелось бы поближе к клиенту, который собирается записывать
- С другой стороны - нужна распределенность
- И более того - хотелось бы разнести физически подальше

Расположение новых блоков

- Не очень правильно хранить блоки на формально разных виртуалках
- Но работающих на одной физической машине
- И даже на физически разных машинах в одной стойке
- Потому что есть отдельный вариант отказа стойки
- Потому что повредился провод, идущий к ней

Расположение новых блоков

- В идеале - хотелось бы учитывать пропускную способность сети, нагрузку и т.п.
- Реально - это сделать очень тяжело
- Тем более это зависит от реального состояния сети
- А правильно мерить скорость передачи - сложная задача
- В итоге - придумана была эвристика

Эвристика для новых блоков

- Если клиент находится на узле кластера
  - Первая реплика на том же узле
- Если нет - на случайно выбранном
- Избегая слишком заполненных и слишком загруженных
- Вторая реплика - случайно выбранная из другой стойки

Эвристика для новых блоков

- Третья реплика - в той же стойке, что и вторая
- Но на другом узле
- Остальные реплики - на случайных узлах
- Если нужно - трех реплик часто бывает достаточно
- Избегая скоплений в одной стойке

Эвристика для новых блоков

- Дает баланс между следующими факторами
  - Надежностью (данные лежат как минимум на двух стойках)
  - Пропускной способностью (для трех реплик только один раз идем в соседнюю сеть)
  - Параллелизмом при чтении (можем читать разные реплики с разных стоек)
- Равномерно распределяет нагрузку по кластеру