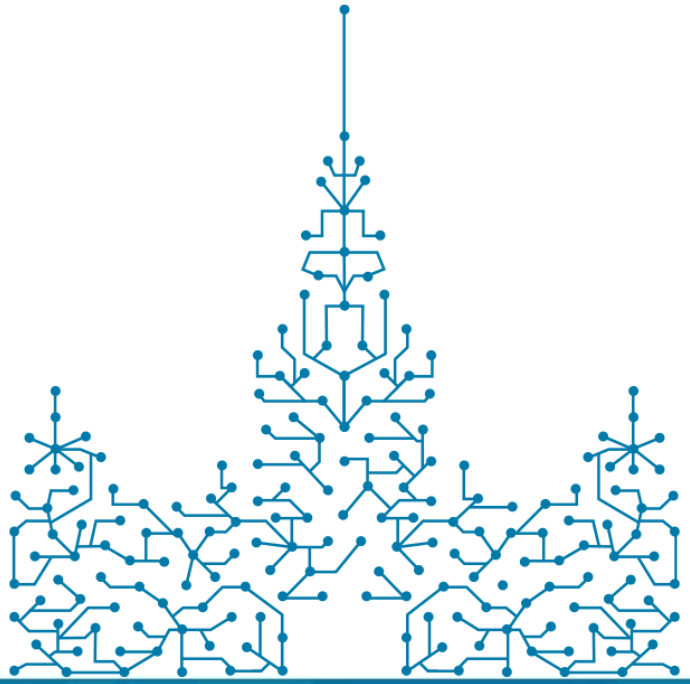
ТЕХНОСФЕРА

# Методы распределенной обработки больших объемов данных в Hadoop

## Лекция 3: Распределенная файловая система HDFS

Москва 2014

# Архитектура HDFS



# HDFS

- *Hadoop Distributed File System*
- Для пользователя как “*один большой диск*”
- Работает поверх обычных файловых систем
  - *Ext3, Ext4, XFS*
- Основывается на архитектуре *Google's Filesystem GFS*
  - *research.google.com/archive/gfs-sosp2003.pdf*
- Fault Tolerant
  - Умеет справляться с выходом из строя дисков, машин и т.д.

# Используется обычное “железо”

- “Дешевое”серверное оборудование
  - **Нет** суперкомпьютерам!
  - **Нет** десктопам!
  - **Да** обычным (~~ненадежным~~) серверам!

# HDFS хорошо подходит для…

- Хранения больших файлов
  - Терабайты, петабайты…
  - Миллионы (но не миллиарды) файлов
  - Файлы размером от 100 Мб
- Стриминг данных
  - Паттерн “write once / read-many times”
  - Оптимизация под последовательное чтение
    - Нет операциям произвольного чтения
  - Операция *append* появилась в Hadoop 0.21
- Обычные сервера
  - Менее надежные, чем суперкомпьютеры

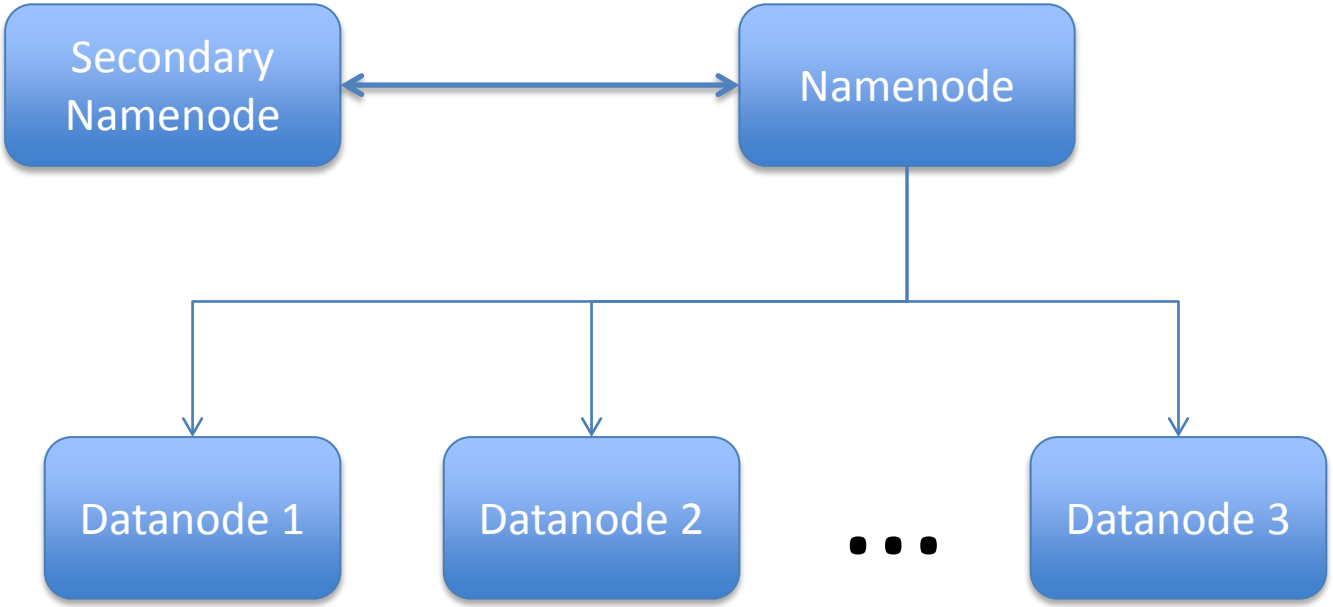
# HDFS не подходит для…

- Low-latency reads
  - Высокая пропускная способность большого объема данных (h*igh-throughput*) вместо небольшого времени доступа к небольшим порциям данных (*low latency*)
  - HBase помогает решать эту задачу
- Большое количество небольших файлов
  - Лучше миллион больших файлов, чем миллиард маленьких
    - Напр. каждый файл размером от 100Мб
- Многопоточная запись
  - Один процесс записи на файл
  - Данные дописываются в конец файла
    - Нет поддержки записи по смещению

# Демоны HDFS

- Для управления файловой системой есть три типа демонов
  - Namenode
    - Отвечает за файловое пространство (namespace), мета-ифномрацию и расположение блоков файлов
    - Запускается на 1й (выделенной) машине
  - Datanode
    - Хранит и отдает блоки данных
    - Отправляет ответы о состоянии на Namenode
    - Запускается обычно на всех машинах кластера
  - Secondary Namenode
    - Периодически обновляет fsimage
    - Требует то же железо, что и Namenode
    - (!) Не используется для high-availability, т.е. это не backup для Namenode

# Демоны HDFS

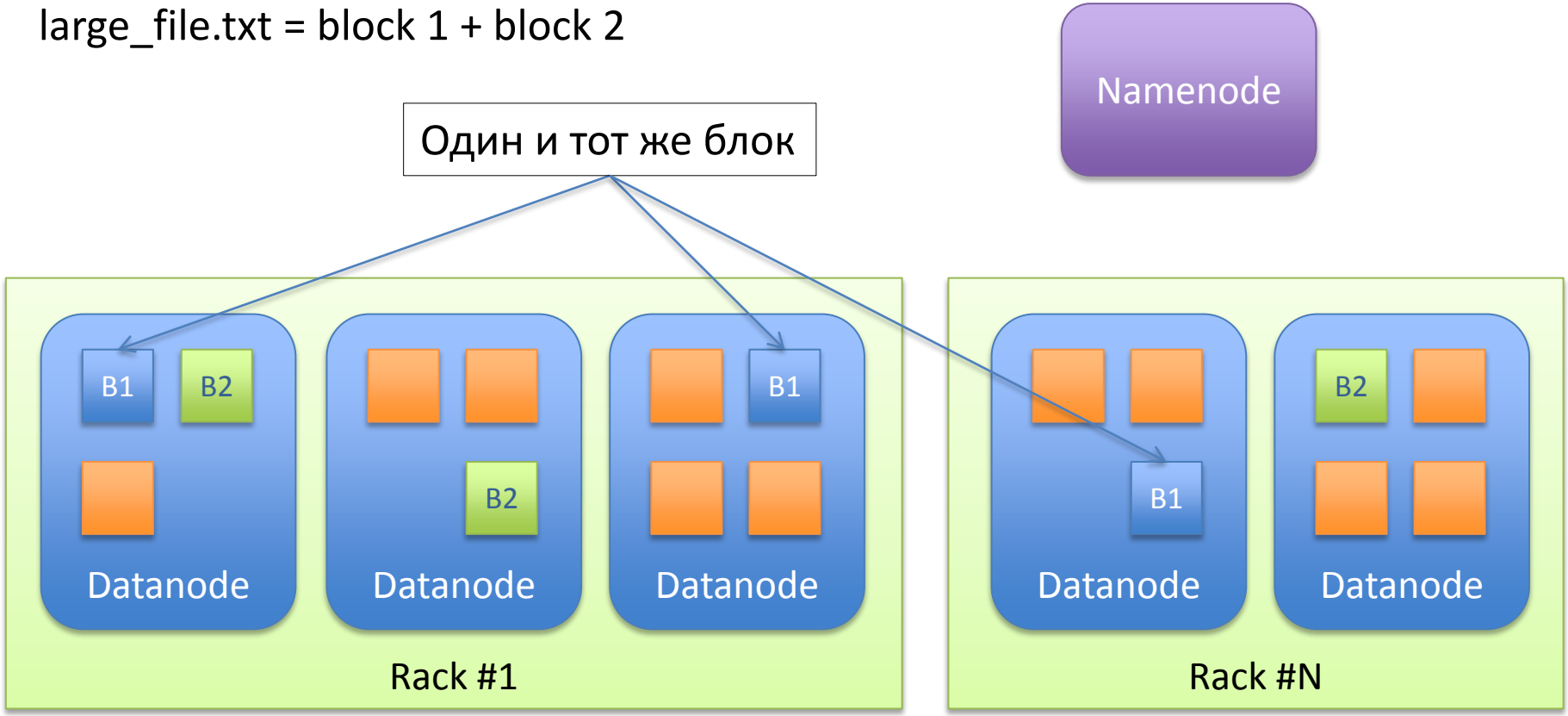

Secondary Namenode
Namenode
Datanode 1
Datanode 2
...
Datanode 3

# Файлы и блоки

- Файлы в HDFS состоят из блоков
  - Блок – единица хранения данных
- Управляется через Namenode
- Хранится на Datanode
- Реплицируются по машинам в процессе записи
  - Один и тот же блок хранится на нескольких Datanode
  - Фактор репликации по умолчанию равен **3**
  - Это нужно для fault-tolerance и упрощения доступа

# Файлы и блоки

large_file.txt = block 1 + block 2



# Блоки в HDFS

- Стандартный размер блоков 64Мб или 128Мб
- Основной мотив этого – снизить стоимость *seek time* по сравнению со скоростью передачи данных (*transfer rate*)
  - 'Time to transfer' > 'Time to seek'
- Например, пусть будет
  - seek time = 10ms
  - transfer rate = 100 MB/s
- Для достижения *seek time* равного 1% от *transfer rate* размер блока должен быть 100Мб

# Репликация блоков

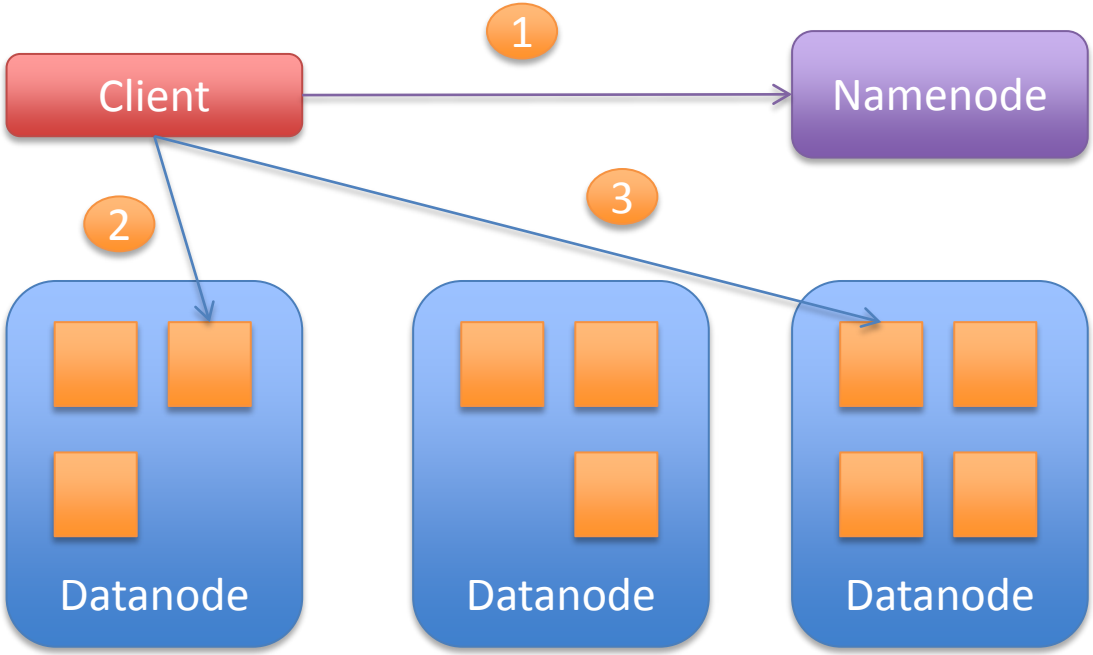
- Namenode определяет, куда копировать реплики блоков
- Размещение блоков зависит от того, в какой стойке стоит сервер (*rack aware*)
  - Баланс между надежностью и производительностью
    - Попытка снизить нагрузку на сеть (*bandwidth*)
    - Попытка улучшить надежность путем размещения реплик в разных стойках
  - Фактор репликации по умолчанию равен 3
    - 1-я реплика на локальную машину
    - 2-я реплика на другую машину из той же стойки
    - 3-я реплика на машину из другой стойки

# Клиенты, Namenode и Datanodes

- Namenode не выполняет непосредственно операций чтения/записи данных
  - Это одна из причин масштабируемости Hadoop
- Клиент обращается к Namenode для
  - обновления неймспейса HDFS
  - для получения информации о размещении блоков для чтения/записи
- Клиент взаимодействует напрямую с Datanode для чтения/записи данных

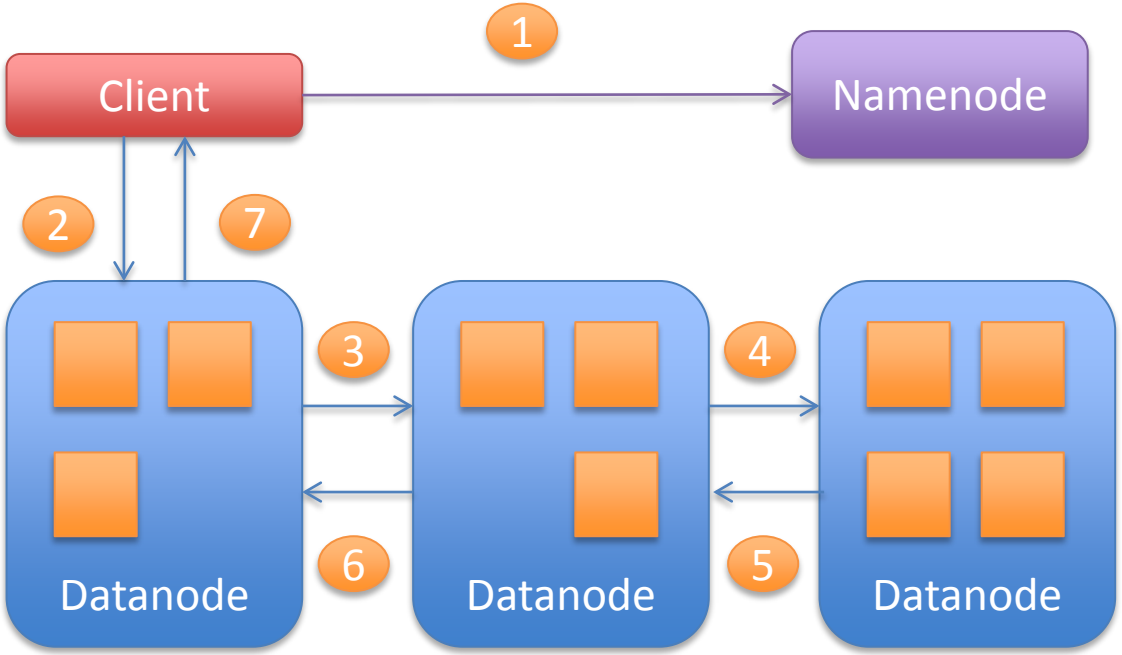
# HDFS: чтение файла

1. Получить расположение блоков
2. Прочитать 1й блок файла
3. Прочитать 2й блок файла



# HDFS: запись файла

1. Создать новый файл в namespace на NN и определить топологию блоков
2. Отправить данные на 1-ю DN
3. Отправить данные на 2-ю DN
4. Отправить данные на 3-ю DN
5. Подтверждение Success/Failure
6. Подтверждение Success/Failure
7. Подтверждение Success/Failure



# Namenode: использование памяти

- Для быстрого доступа вся мета-информация о блоках хранится в ОЗУ Namenode
  - Чем больше кластер, тем больше ОЗУ требуется
    - Лучше миллионы больших файлов (сотни мегабайт), чем миллиарды маленьких
    - Работает на кластерах из сотен машин
- Hadoop 2+
  - Namenode Federation
    - Каждая Namenode управляет частью блоков
    - Горизонтальное масштабирование Namenode
  - Поддержка кластеров из тысячи машин
  - Детали тут: http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/Federation.html

# Namenode: использование памяти

- Изменение размера блока влияет на максимальный размер FS
  - Увеличение размера блока с 64Мб до 128Мб уменьшает число блоков и существенно увеличивает размер места, которое NN может обслуживать
  - Пример:
    - Пусть у нас есть 200Тб = 209,715,200 Мб
    - При размере блока 64Мб это соответствует 3,276,800 блоков
      - 209,715,200Мб / 64Мб = 3,276,800 блоков
    - При размере блока 128Мб это соответствует 1,638,400 блоков
      - 209,715,200Мб / 128Мб = 1,638,400 блоков

# Fault-tolerance в Namenode

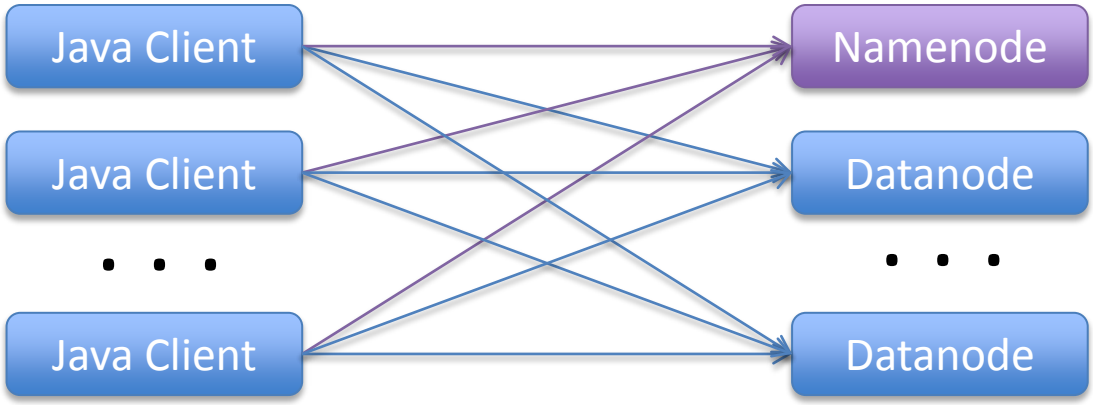
- Процесс демона NN должен быть запущен все время
  - Если демон падает, то HDFS не работает (и обычно весь кластер тоже)
- Namenode – это единая точка отказа (*single point of failure*)
  - Должна работать на отдельной надежной машине
  - Обычно, это не бывает проблемой
- Hadoop 2+
  - High Availability Namenode
    - Процесс Active Standby всегда запущен и берет на себя управления в случае падения NN
    - Все еще в процессе тестирования
  - Более подробно тут:
    - http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailability.html

# Доступ к HDFS

- Способы доступа
  - Direct Access
    - Взаимодействует с HDFS с помощью нативного клиента
    - Java, C++
  - Через Proxy Server
    - Досутп к HDFS через Proxy Server – middle man
    - Серверы REST, Thrift и Avro

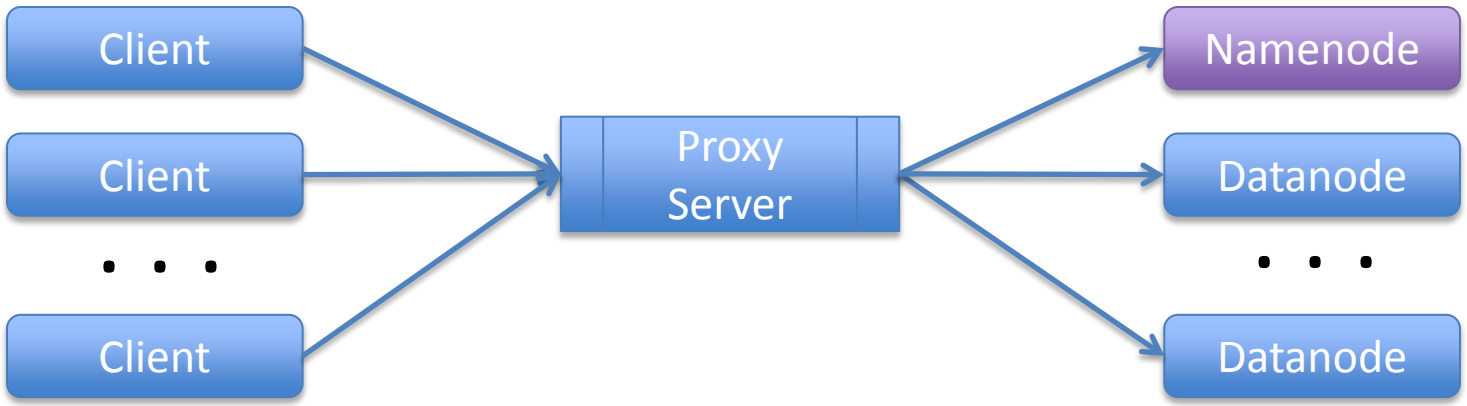
# Direct Access

- API для Java и C++
- Клиент запрашивает метаданные (такие, как расположение блоков) от NN
- Клиент напрямую запрашивает данные от DN
- Java API
  - Наиболее часто используется
- Используется для MapReduce

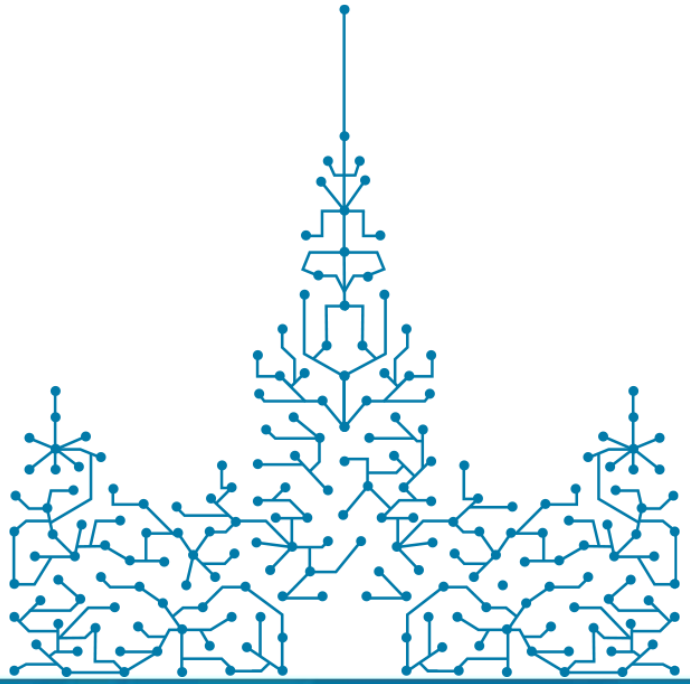


# Доступ через Proxy Server

- Клиент работает через внешний Proxy Server
  - Т.о. должна быть независимость от языка
- Существует несколько серверов в поставке с Hadoop
  - Thrift – язык определения интерфейса
  - WebHDFS REST – ответы в формате JSON, XML или Protocol Buffers
  - Avro – механизм сериализации



# HDFS Shell



# Команды Shell

- Взаимодействие с FS через стандартный unix-shell
- Использование:
  **$hdfs dfs -<command> -<option> <URI>**
  - Example *$hdfs dfs -ls /*
- **URI usage:**
  - HDFS: *$hdfs dfs -ls hdfs://localhost/to/path/dir*
  - Local: *$hdfs dfs -ls file:///to/path/file3*
  - Схема и имя хоста NN опционально, по-умолчанию используется параметр из конфигурации
    - В core-site.xml - fs.default.name property

# Hadoop URI

**scheme://autority/path**

**hdfs://localhost:8020/user/home**

scheme authority HDFS path

# Команды в shell

- Большинство команд ведет себя схожим образом, что и команды в Unix
  - *cat, rm, ls, du …*
- Поддержка специфичных для HDFS операций
  - Напр., смена фактора репликации
- Вывод списка команд
  - *$ hdfs dfs –help*
- Показать детальную информацию по команде
  - *$ hdfs dfs -help <command_name>*

# Основные команды в shell

- ***cat*** – вывод источника в stdout
  - Весь файл: *$hdfs dfs -cat /dir/file.txt*
  - Полезно вывод перенаправить через pipe в *less, head, tail* и т.д.
  - Получить первые 100 строк из файла
    - *$hdfs dfs -cat /dir/file.txt | head -n 100*
- ***ls*** – отобразить файловую статистику, для директории отобразить вложенные директории и файлы
  - *$hdfs dfs -ls /dir/*
- ***mkdir*** – создать директорию
  - *$hdfs dfs -mkdir /dir*

# Копирование данных в shell

- ***cp –*** скопировать файлы из одного места в другое
  - *$hdfs dfs -cp /dir/file1 /otherDir/file2*
- ***mv*** – перемещение файла из одного места в другое
  - *$hdfs dfs -mv /dir/file1 /dir2*
- ***put*** – копирование файла из локальной FS в HDFS
  - *$hdfs dfs -put localfile /dir/file1*
  - *copyFromLocal*
- ***get*** – копирование файла из HDFS в локальную FS
  - *$hdfs dfs -get /dir/file1 localfile*
  - *copyToLocal*

# Удаление и статистика в shell

- ***rm –*** удалить файл (в корзину)
  - *$hdfs dfs -rm /dir/file*
- ***rm -r –*** удалить рекурсивно директорию
  - *$hdfs dfs -rm -r /dir*
- ***du –*** отобразить размер файла или директории в байтах
  - *$hdfs dfs -du /dir/*
- ***du -h –*** отобразить размер файла или директории в удобно-читаемом формате
  - *$hdfs dfs -du –h /dir/*
    *65M  /dir*

# Остальные команды в shell

- Другие команды
  - *chmod, count, test, tail и т.д.*
- Чтобы узнать больше
  - *$hdfs dfs* ***-help***
  - *$hdfs dfs* ***-help <command>***

# Команда **fsck**

- Проверка неконсистентности файловой системы
- Показывает проблемы
  - Отсутствующие блоки
  - Недореплицированные блоки
- Не устраняет проблем, только информация
  - Namenode попытается автоматически исправить проблемы
- **$ hdfs fsck <path>**
  - Напр., *$ hdfs fsck /*

# Права в HDFS

- Ограничения на уровне файла/директории
  - Сходство с моделью прав в POSIX
  - Read (r), Write (w) и Execute (x)
  - Разделяется на пользователя, группу и всех остальных
- Права пользователя определяются исходя из прав той ОС, где он запускает клиентское приложение
- Авторизация через Kerberos
  - Hadoop 0.20.20+
  - http://hadoop.apache.org/common/docs/r0.23.0/hadoop-yarn/hadoop-yarnsite/ClusterSetup.html

```
[cloudera@localhost ~]$ hadoop fs -ls /user/cloudera/wordcount/output
Found 3 items
-rw-r--r--   3 cloudera cloudera    0 2014-03-15 11:56 /user/cloudera/wordcount/output/_SUCCESS
drwxr-xr-x   - cloudera cloudera    0 2014-03-15 11:56 /user/cloudera/wordcount/output/_logs
-rw-r--r--   3 cloudera cloudera   31 2014-03-15 11:56 /user/cloudera/wordcount/output/part-00000
```

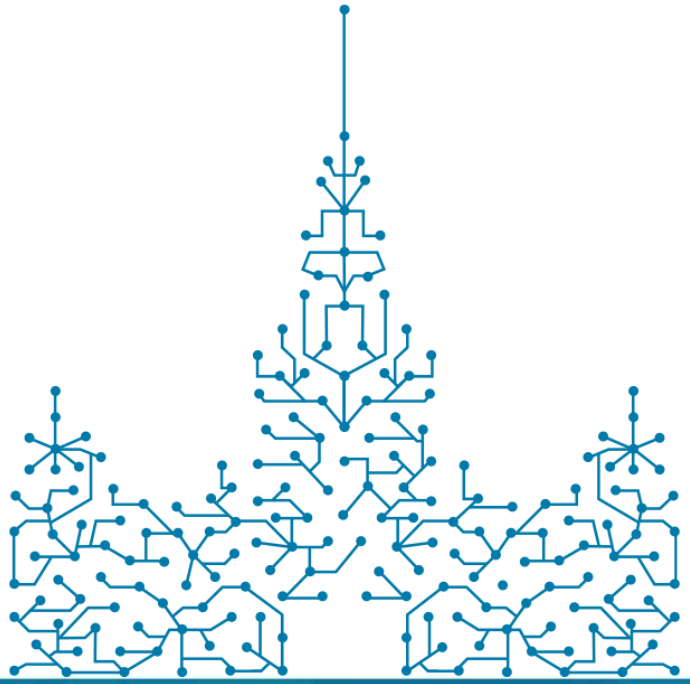
# Команда **DFSAdmin**

- Команды для администрирования HDFS
  - *$hdfs dfsadmin <command>*
  - Напр.: *$hdfs dfsadmin –report*
- ***report*** – отображает статистику по HDFS
  - Часть из этого также доступна в веб-интерфейсе
- ***safemode*** – переключения между режимом safemode для проведения административных работ
  - Upgrade, backup и т.д.

# Балансер HDFS

- Блоки в HDFS могут быть неравномерно распределены по всем Datanode’ам кластера
  - К примеру, при добавлении новых машин они будут какое-то время почти пустыми, т.к. новые данные будут записываться исходи из топологии
- Балансер – это утилита, которая автоматически анализирует расположение блоков в HDFS и старается его сбалансировать
  - *$ hdfs balancer*

# HDFS API



# File System Java API

- org.apache.hadoop.fs.FileSystem
  - Абстрактный класс, которые представляет абстрактную файловую систему
  - (!) Это именно класс, а не интерфейс
- Реализуется в различных вариантах
  - Напр., локальная или распределенная

# Реализации *FileSystem*

- Hadoop предоставляет несколько конкретных реализаций
  - *org.apache.hadoop.fs.LocalFileSystem*
    - Подходит для нативных FS, использующих локальные диски
  - *org.apache.hadoop.hdfs.DistributedFileSystem*
    - Hadoop Distributed File System (HDFS)
  - *org.apache.hadoop.hdfs.HftpFileSystem*
    - Доступ к HDFS в read-only режиме через HTTP
  - *org.apache.hadoop.fs.ftp.FTPFileSystem*
    - Файловая система поверх FTP-сервера
- Различные реализации для разных задач
- Для работы с HDFS обычно используется
  - org.apache.hadoop.hdfs.DistributedFileSystem

# Пример SimpleLocalLs.java

```
public class SimpleLocalLs {
   public static void main(String[] args) throws Exception{

      Path path = new Path("/");
      if ( args.length == 1 ){
         path = new Path(args[0]);
      }

      Configuration conf = new Configuration();
      FileSystem fs = FileSystem.get(conf);

      FileStatus [] files = fs.listStatus(path);
      for (FileStatus file : files ){
         System.out.println(file.getPath().getName());
      }
   }
}
```

# FileSystem API: Path

- Объект Path в Hadoop представляет файл или директорию
  - *java.io.File* сильно завязан на локальную FS
- Path – это на самом деле URI в FS
  - HDFS: hdfs://localhost/user/file1
  - Local: file:///user/file1
- Пример:
  - *new Path("/test/file1.txt");*
  - *new Path("hdfs://localhost:9000/test/file1.txt");*

# Объект Configuration

- Объект Configuration хранит конфигурацию сервера и клиента
  - Довольно много где используется в Hadoop
    - HDFS, MapReduce, Hbase,..
- Использует простую парадигму key-value
  - Является враппером надо java.util.Properties,
  - Который в свою очередь враппер над java.util.Hashtable
- Получения значения параметра
  - *String name = conf.get("fs.default.name");*
    - returns null если свойства не существует
- Получения значения параметра и вернуть значение по-умолчанию, если не существует
  - *String name = conf.get("fs.default.name“, “hdfs://localhost:9000”);*
- Также есть типизированные варианты
  - getBoolean, getInt, getFloat и т.п.
    - float size = conf.getFloat("file.size");

# Объект Configuration

- Обычно инициализируется значениями через конфигурационный файлы из CLASSPATH (напр., conf/coresite.xml и conf/hdfs-site.xml)
  - *Configuration conf = new Configuration();*
    *conf.addResource(*
    *new Path(HADOOP_HOME + "/conf/coresite.xml"));*
- *conf.addResource()* может принимать как String, так и Path
  - *conf.addResource("hdfs-site.xml")*
  - *conf.addResource(new Path("/my/location/site.xml")*
- По-умолчанию загружает
  - core-default.xml
    - Расположен в hadoop-common-X.X.X.jar/core-default.xml
  - core-site.xml

# Чтение данных из файла

- Создать объект ***FileSystem***
- Открыть ***InputStream,*** указывающий на ***Path***
- Скопировать данные по байтам используя ***IOUtils***
- Закрыть ***InputStream***

# Пример ReadFile.java

```
public class ReadFile {
   public static void main(String[] args) throws IOException {
      Path file = new Path("/path/to/file.txt");
      FileSystem fs = FileSystem.get(new Configuration()); // Open FileSystem

      InputStream input = null;
      try {
         input = fs.open(file); // Open InputStream
         IOUtils.copyBytes(input, System.out, 4096); // Copy from Input to Output
Stream
      } finally {
         IOUtils.closeStream(input); // Close stream
      }
   }
}
```

# Запись данных в файл

- Создать объект ***FileSystem***
- Открыть ***OutputStream***
  - Указывает на ***Path*** из ***FileSystem***
  - Используем ***FSDataOutputStream***
  - Автоматически создаются все директори в пути, если не существуют
- Копируем данные по байтам используя ***IOUtils***

# Пример WriteToFile.java

```
public class WriteToFile {
   public static void main(String[] args) throws IOException {
      String text = "Hello world in HDFS!\n";
      InputStream in = new BufferedInputStream(
           new ByteArrayInputStream(text.getBytes()));

      Path file = new Path("/path/to/file.txt");
      Configuration conf = new Configuration();
      FileSystem fs = FileSystem.get(conf); // Create FileSystem
      FSDataOutputStream out = fs.create(file); // Open OutputStream
      IOUtils.copyBytes(in, out, conf); // Copy Data
   }
}
```

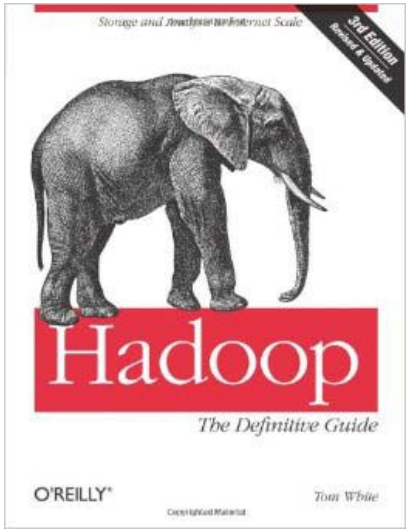
# **FileSystem:** запись данных

- ***fs.append(path)*** – дописать к существующему файлу
  - Поддержка для HDFS
- Нельзя записать в середину файла
- ***FileSystem.create(Path)*** создает все промежуточные директории для заданного каталога (по умолчанию)
  - Если это не нужно, то надо использовать
    - *public FSDataOutputStream create(Path f, boolean overwrite)*
    - *overwrite = false*

# **FileSystem:** подстановки (globbing)

- ***FileSystem*** имеет поддержку матчинга имени файла по заданному паттерну используя метод ***globStatus()***
  - *FileStatus [] files = fs.globStatus(glob);*
- Используется для выбора списка файлов по шаблону
- Примеры шаблонов
  - **?** – любой один символ
  - ***** - любые 0 и больше символов
  - **[abc]** – любой символ из набора в скобках
    - [a-z]
  - [^a] – любой символ, кроме указанного
  - {ab,cd} – любая строка из указанных в скобках

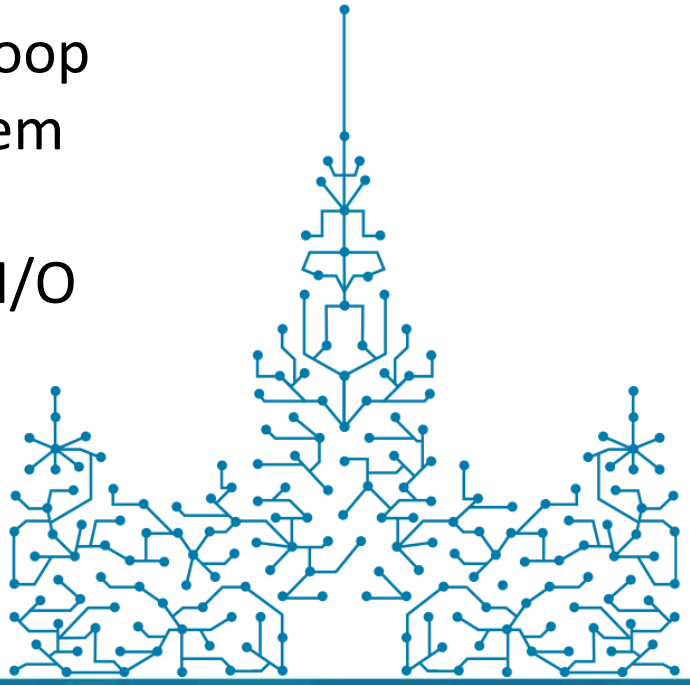
# Книги




**Hadoop: The Definitive Guide**
Tom White (Author)
O'Reilly Media; 3rd Edition

Chapter 3: The Hadoop Distributed Filesystem

Chapter 4: Hadoop I/O



# Вопросы?