

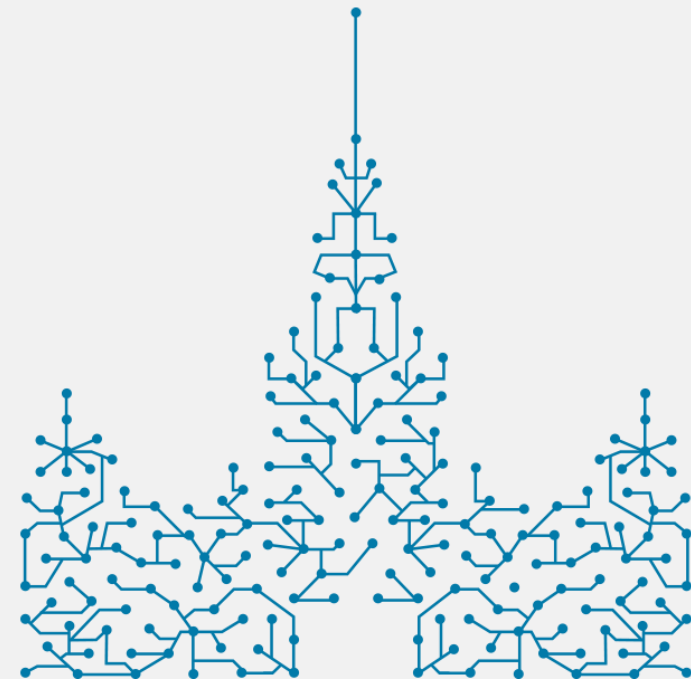
Лекция №2

Распределенная файловая система HDFS

Евгений Чернов



Архитектура HDFS (Hadoop Distributed File System)





Google's Filesystem GFS

research.google.com/archive/gfs-sosp2003.pdf



- Работает на кластере серверов
- Для пользователя как “*один большой диск*”
- Работает поверх обычных файловых систем
(*Ext3, Ext4, XFS*)



Данные не теряются, если выходят из строя диски или сервера



Используется ли “железо”



“Дешевое” серверное оборудование

- **Нет** суперкомпьютеров
- **Нет** десктопов
- **Да** обычным (иногда мощным) серверам!

HDFS хорошо подходит для...



Хранения больших файлов

- Терабайты, петабайты...
- Миллионы (но не миллиарды) файлов
- Файлы размером от 100 Мб

HDFS хорошо подходит для...



Стриминг данных

- Паттерн “write once / read-many times”
- Оптимизация под последовательное чтение
 - Нет операциям произвольного чтения
- Операция *append* появилась в Hadoop 0.21

HDFS хорошо подходит для...



Обычные сервера

- Менее надежные, чем суперкомпьютеры

HDFS не подходит для...



Low-latency reads

- Высокая пропускная способность вместо быстрого доступа к данным
- HBase помогает решать эту задачу

HDFS не подходит для...



Большое количество небольших файлов

- Лучше миллион больших файлов, чем миллиард маленьких

Лучше 1000 по 1 Гб, чем 100 000 по 10 Мб

HDFS не подходит для...



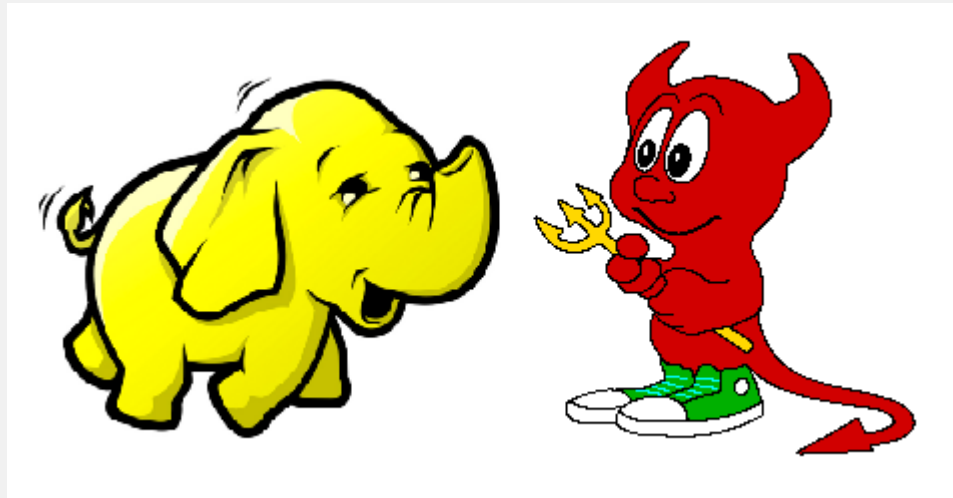
Многопоточная запись

- Один процесс записи на файл
- Данные дописываются в конец файла

Нет поддержки записи по смещению



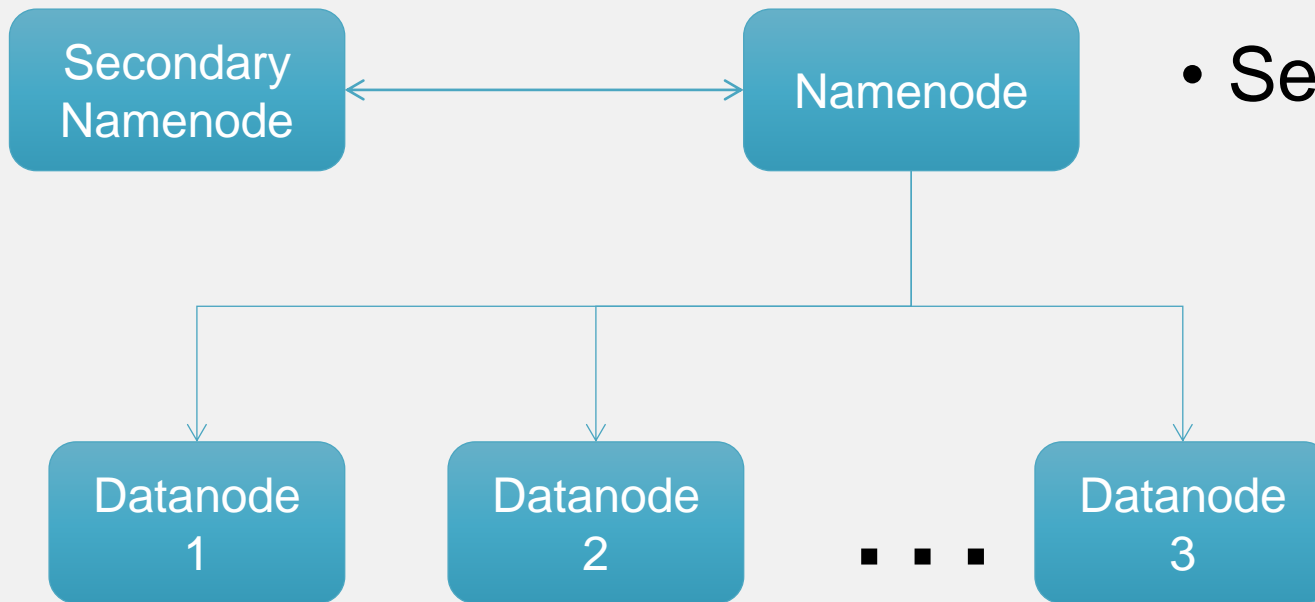
Демоны HDFS



Демоны HDFS



- Namenode
- Datanode
- Secondary node





- Отвечает за:
 - файловое пространство (namespace)
 - мета-информацию
 - расположение блоков файлов
- Запускается на 1й (выделенной) машине



-
- Хранит и отдает блоки данных
 - Отправляет ответы о состоянии на Namenode
 - Запускается на каждой машине кластера

Secondary Namenode

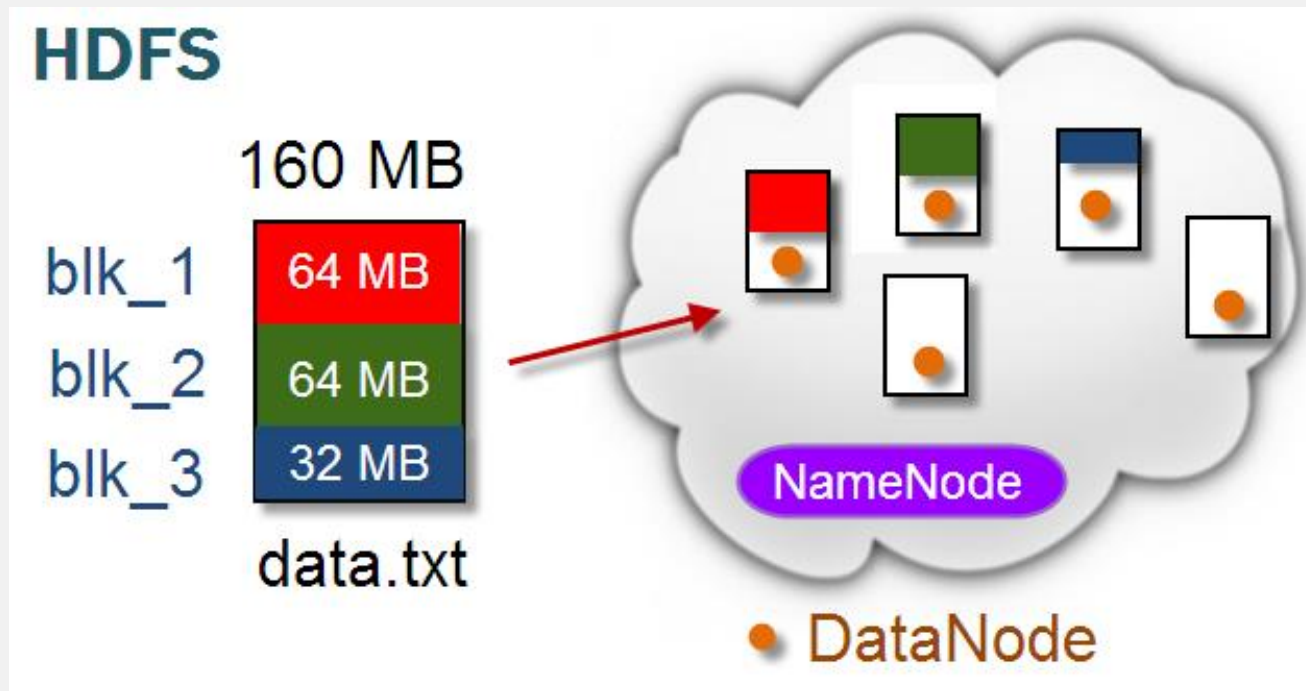


- Периодически обновляет fsimage
- Требуется то же железо, что и Namenode
- (!) Не используется для high-availability, т.е. это не backup для Namenode

Файлы и блоки



- Файлы в HDFS состоят из блоков
 - Блок – единица хранения данных
- Управляется через Namenode
- Хранится на Datanode



Файлы и блоки



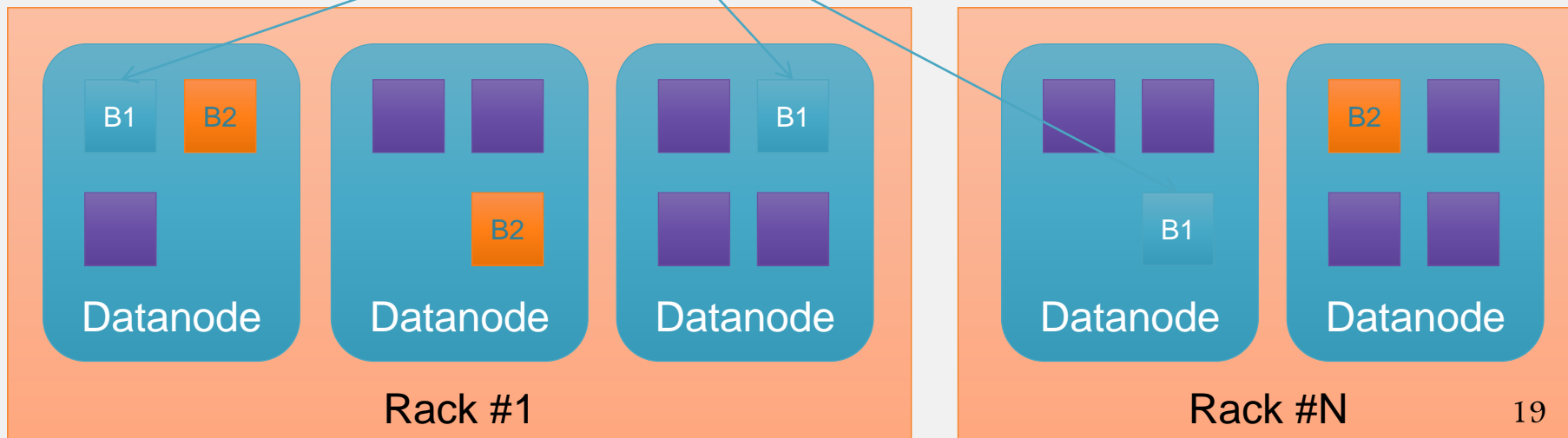
Блок реплицируются по машинам в процессе записи

- Один и тот же блок хранится на нескольких Datanode
- Фактор репликации по умолчанию равен 3

large_file.txt = block 1 + block 2

Один и тот же блок

Namenode





Размер блоков 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

Задача



Какой должен быть размер блока, чтобы seek time составлял 1,5% от transfer time, если:

- seek time = 20 мс
- transfer rate = 48 МБ/с

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

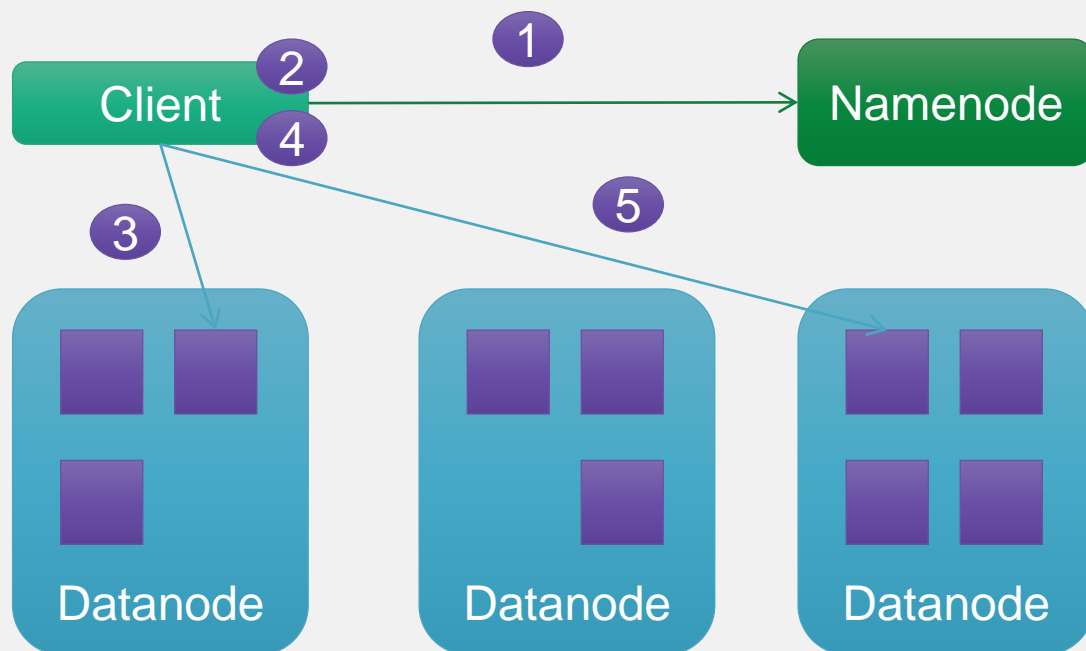


- Namenode не выполняет непосредственно операций чтения/записи данных
 - Это одна из причин масштабируемости Hadoop
- Клиент обращается к Namenode для
 - обновления неймспейса HDFS
 - для получения информации о размещении блоков для чтения/записи
- Клиент взаимодействует напрямую с Datanode для чтения/записи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

HDFS: чтение файла



1. Получить расположение блоков
2. Выбрать “ближайший” Datanode для 1 блока
3. Прочитать 1й блок файла
4. Выбрать “ближайший” Datanode для 2 блока
5. Прочитать 2й блок файла
6.
7. Закрыть соединение



Важно: передача данных происходит без участия NameNode

HDFS: ошибки при чтении файла

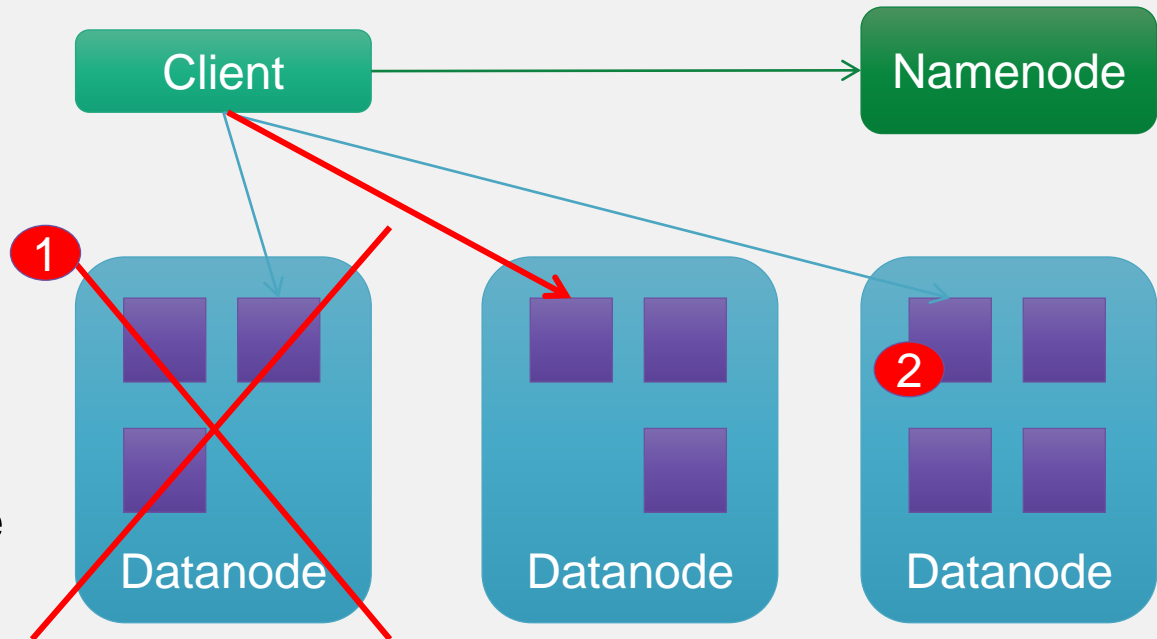


1. Поломка Datanode:

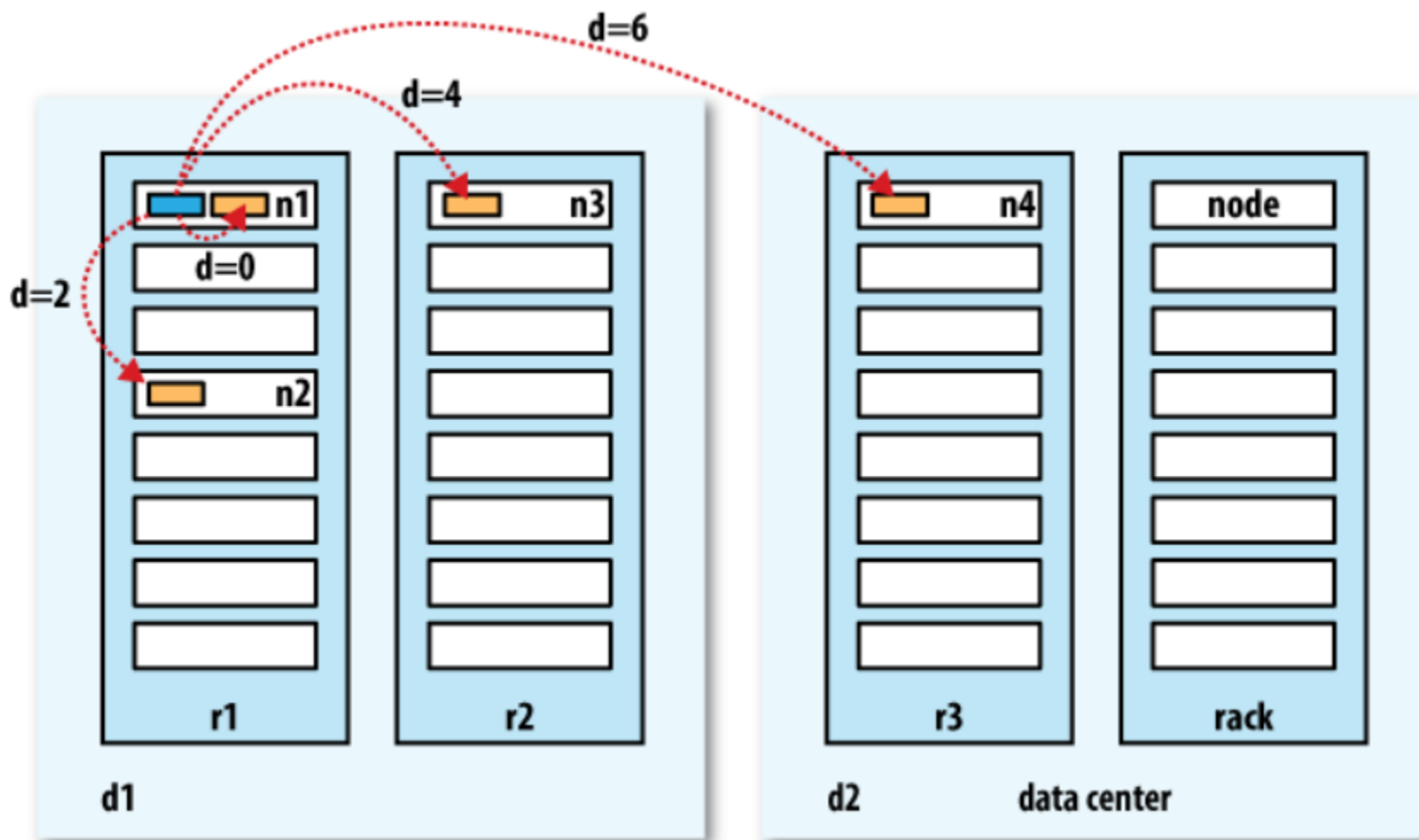
- Читаем блок с другой Datanode
- Запоминаем сломанный

2. Неправильная чексумма:

- Сообщаем NameNode о сломанном блоке
- Читаем другую реплику блока



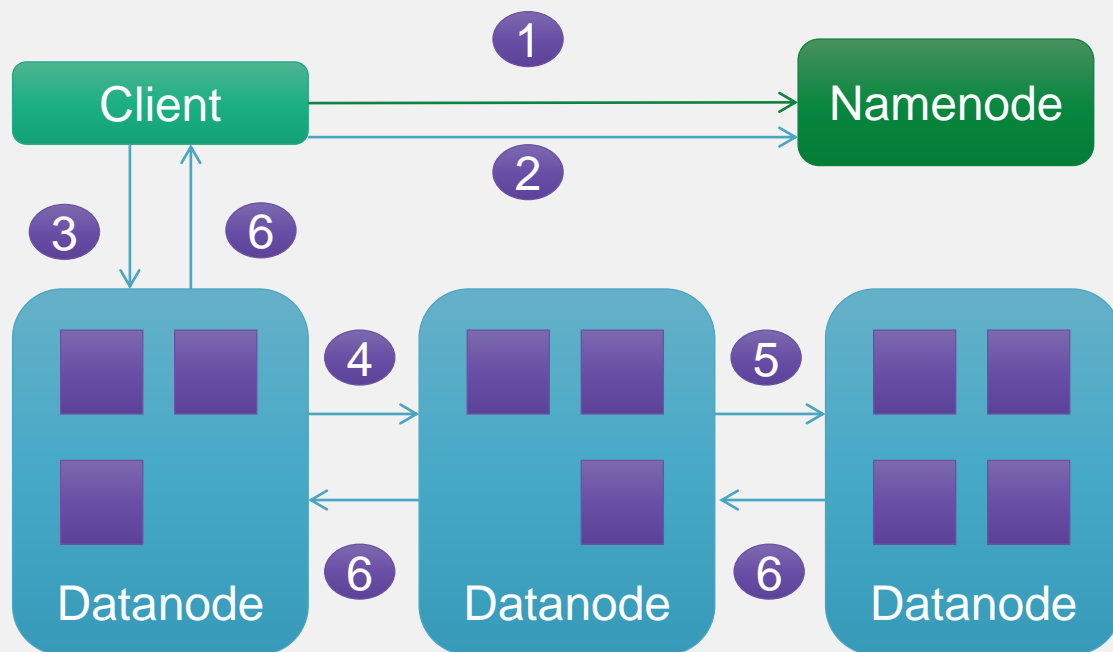
Расстояние до блоков



HDFS: запись файла



1. Сообщаем NN о создании файла:
 - проверка существования
 - проверка прав доступа
 - создание записи о файле
2. Запрашиваем расположение 1-го блока у NN:
 - получаем список DN
 - DN образуя pipeline
3. Начинаем писать данные на 1-ю DN
4. 1-я DN сохраняет данные и отправляет их на 2-ю DN
5. 2-я DN сохраняет данные и отправляет их на 3-ю DN
6. Подтверждение записи



Внимание: данные между DataNode передается пачками по 4 Кб

HDFS: ошибки при записи файла

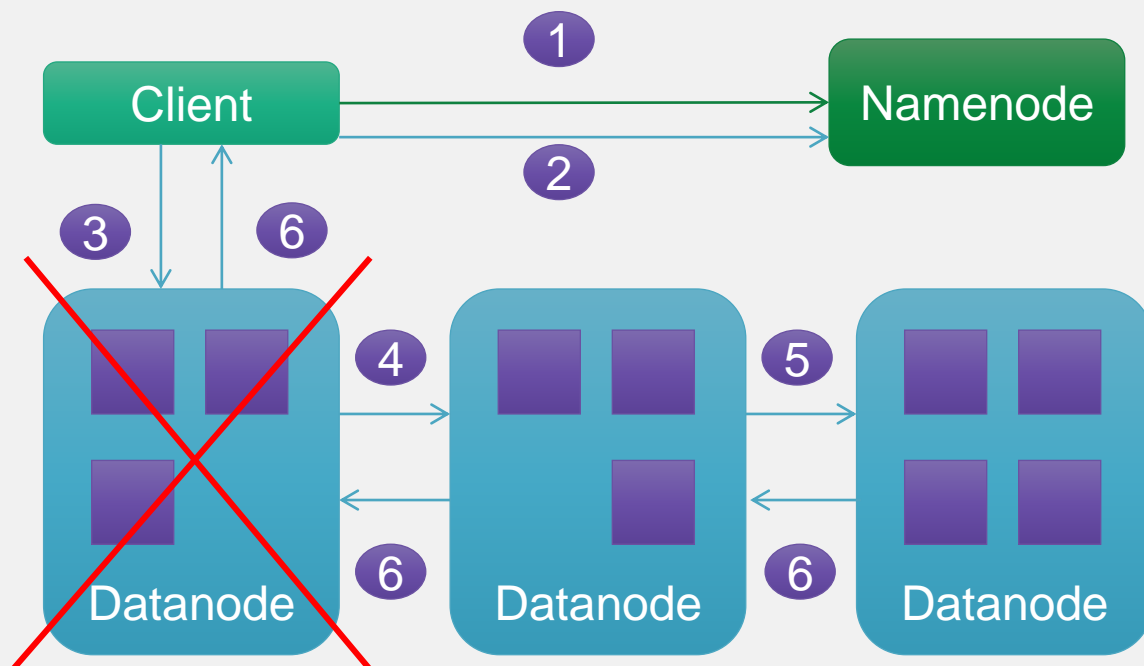


1. Падение DataNode:

- не до конца записанные данные не теряются
- DataNode исключается из pipeline
- данные пишутся на оставшиеся DN
- блок становится недореплицированным
- меняется идентификатор блока

2. Могут упасть несколько DN

- ## 3.
- Если есть хотя бы одна копия, то запись считается успешной



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, чем миллиарды маленьких
- Работает на кластерах из сотен машин

Namenode: использование памяти



Как влияет размер блока на максимальный размер FS?

- Больше размер блока → меньше блоков
- Меньше блоков → больше файлов в FS

Namenode: использование памяти



- Пусть у нас есть 200Тб = 209,715,200 Мб
- При размере блока 64Мб:
$$209,715,200\text{Мб} / 64\text{Мб} = 3,276,800 \text{ блоков}$$
- При размере блока 128Мб:
$$209,715,200\text{Мб} / 128\text{Мб} = 1,638,400 \text{ блоков}$$

Namenode: использование памяти



Hadoop 2+

- Namenode Federation
 - Каждая Namenode управляет частью блоков
 - Горизонтальное масштабирование Namenode
- Поддержка кластеров из тысячи машин
- Детали: <http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/Federation.html>

Fault-tolerance в Namenode



- Если Namenode падает, то HDFS не работает
- Namenode – это единая точка отказа (*single point of failure*)
 - Должна работать на отдельной надежной машине
 - Обычно, это не бывает проблемой

Fault-tolerance в Namenode



Hadoop 2+

- High Availability Namenode
 - Процесс Active Standby всегда запущен и берет на себя управления в случае падения Namenode
 - Все еще в процессе тестирования
- Более подробно тут:
 - <http://hadoop.apache.org/docs/r2.0.2-alpha/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailability.html>

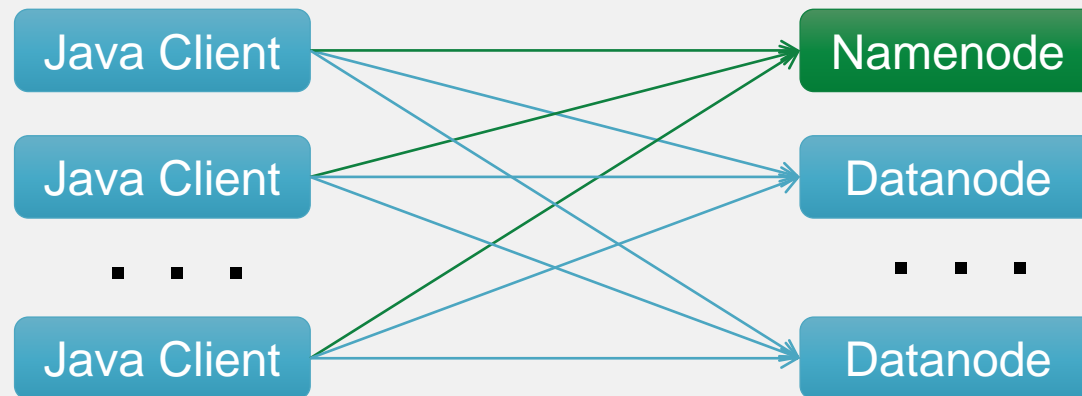


- Способы доступа
 - Direct Access
 - Взаимодействует с HDFS с помощью нативного клиента
 - Java, C++
 - Через Proxy Server
 - Доступ к HDFS через Proxy Server – middle man
 - Серверы REST, Thrift и Avro

Direct Access



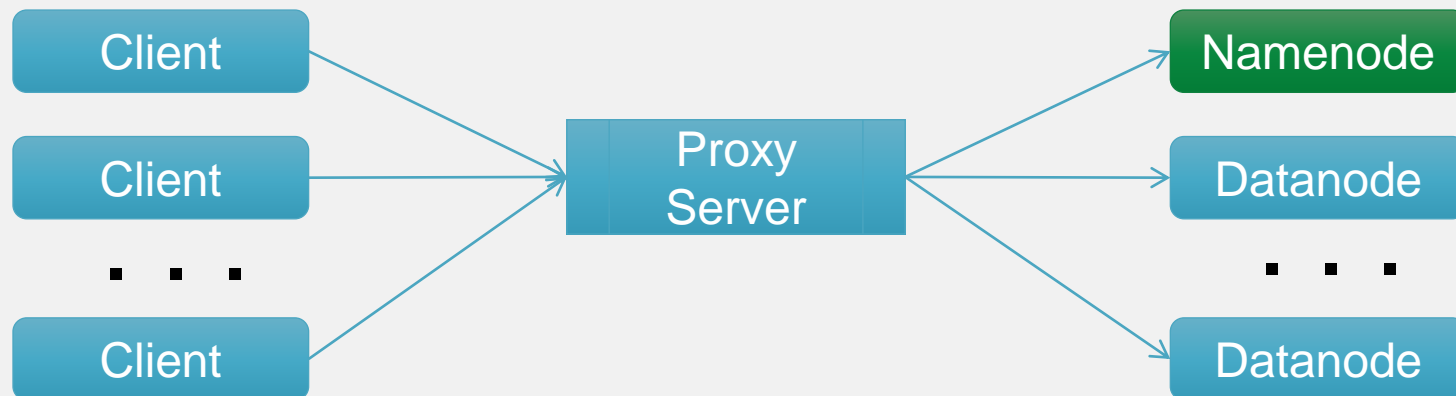
- API для Java и C++
- Клиент запрашивает метаданные от NN
- Клиент напрямую запрашивает данные от DN
- Используется для MapReduce



Доступ через Proxy Server

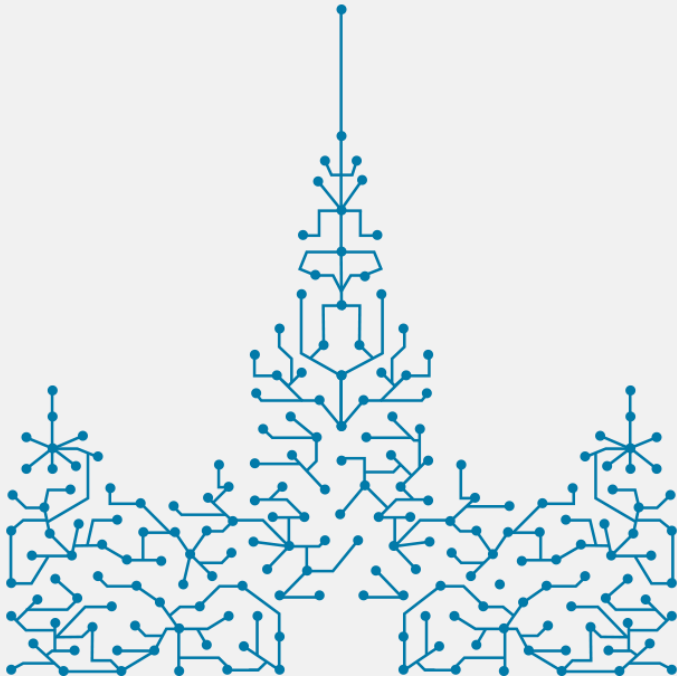


- Клиент работает через внешний Proxy Server
- Существует несколько серверов в поставке с Hadoop
 - Thrift – язык определения интерфейса
 - WebHDFS REST – ответы в формате JSON, XML или Protocol Buffers
 - Avro – механизм сериализации





HDFS Shell





\$hdfs dfs -<command> -<option> <URI>

\$hdfs dfs -ls /

URI (*Uniform Resource Identifier*)



hdfs://localhost:8020/user/home

scheme

authority

HDFS path

Примеры URI



- Local:

\$hdfs dfs -ls file:///to/path/file3

- HDFS:

\$hdfs dfs -ls hdfs://localhost/to/path/dir

- fs.default.name=hdfs://localhost

\$hdfs dfs -ls /to/path/dir

Команды в shell



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

Основные команды в shell



ls – листинг директории и статистика файлов

```
[cloudera@localhost ~]$ hdfs dfs -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
```

Основные команды в shell



mkdir – создать директорию

\$hdfs dfs -mkdir /data/new_path

Чтение файлов 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`

Чтение файлов в shell



text – аналог **cat**, но разархивирует архивы:

*\$hdfs dfs -**cat** /dir/file.gz – непонятный текст*

*\$hdfs dfs -**text** /dir/file.gz – понятный текст*

*\$hdfs dfs -**cat** /dir/file.txt = \$hdfs dfs -**text** /dir/file.txt*

tail – выводит последние строки файла

*\$hdfs dfs -**cat** /dir/file.txt | tail – плохо*

*\$hdfs dfs -**tail** /dir/file.txt – хорошо*

Копирование данных в shell



cp – скопировать файлы из одного места в другое

```
$hdfs dfs -cp /dir/file1 /otherDir/file2
```

! годится только для небольших файлов

distcp – копирует большие файлы или много файлов

```
$hadoop distcp /dir/file1 /otherDir/file2
```

Копирование данных в shell



- ***mv*** – перемещение файла из одного места в другое
 - `$hdfs dfs -mv /dir/file1 /dir2`
- ***put (copyFromLocal)*** – копирование файла из локальной FS в HDFS
 - `$hdfs dfs -put localfile /dir/file1`
 - `copyFromLocal`
- ***get (copyToLocal)*** – 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 – размер файла или директории в байтах

```
$hdfs dfs -du /dir/
```

du -h – размер файла или директории в удобно-читаемом формате

```
$hdfs dfs -du -h /dir/  
65M /dir
```

Остальные команды в shell



- Другие команды
 - *chmod, chown, count, test 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>**
 - *\$ hdfs fsck /*
- Выводит информацию о файлах и блоках:
 - *\$ hdfs fsck / -files -blocks*



- 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>

Права в HDFS



```
[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
- Балансер – это утилита, которая автоматически анализирует расположение блоков в HDFS и старается его сбалансировать
 - `$ hdfs balancer`

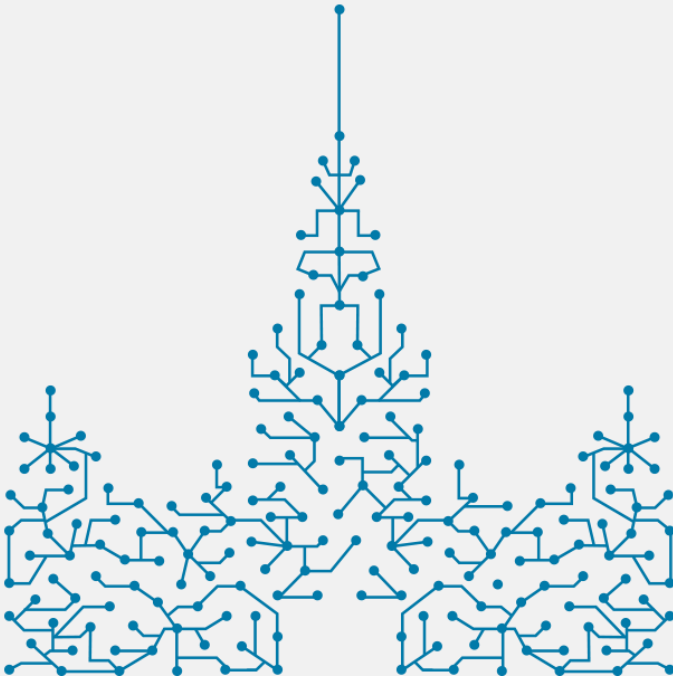
Задание



1. Создать локально файл `test.txt` размером 100Мб
2. Создать `hdfs`-директории `temp` и `logs`
3. Записать файл `test.txt` в директорию `temp`
4. Посмотреть свойства записанного файла
5. Переместить файл `test.txt` в директорию `logs`
6. Установить фактор репликации для файла равным 2
7. Скопировать `test.txt` в `test2.txt`
8. Скопировать директорию `logs` в `logs2` с помощью `distcp`
9. Посмотреть размер всех директорий в `./`
10. Удалить директорию `logs2`
11. Запустить `fscck` на директории `logs`
12. Посмотреть отчет о HDFS через `dfsadmin`



HDFS 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

Пример SimpleLocals.java



```
public class SimpleLocals {  
    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 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>

```
new Path("/test/file1.txt");  
new Path("hdfs://localhost:9000/test/");
```

Объект Configuration



- Объект Configuration хранит конфигурацию сервера и клиента
- Использует простую парадигму key-value
- Получения значения параметра:

```
String name = conf.get("fs.default.name");
```

```
String name = conf.get("fs.default.name", "hdfs://localhost:9000");
```

```
float size = conf.getFloat("file.size");
```

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  
            // Copy from Input to Output Stream  
            IOUtils.copyBytes(input, System.out, 4096);  
        } finally {  
            IOUtils.closeStream(input); // Close stream  
        }  
    }  
}
```

Запись данных в файл



- Создать объект ***FileSystem***
- Открыть ***OutputStream***
 - Указывает на ***Path*** из ***FileSystem***
 - Используем ***FSDDataOutputStream***
 - 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();  
        // Create FileSystem  
        FileSystem fs = FileSystem.get(conf);  
        // Open OutputStream  
        FSDataOutputStream out = fs.create(file);  
        IOUtils.copyBytes(in, out, conf); // Copy Data  
    }  
}
```

FileSystem: запись данных

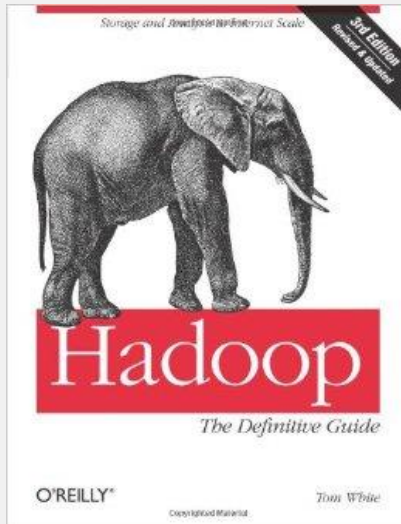


- ***fs.append(path)*** – дописать к существующему файлу
 - Поддержка для HDFS
- Нельзя записать в середину файла
- ***FileSystem.create(Path)*** создает все промежуточные директории для заданного каталога (по умолчанию)
 - Если это не нужно, то надо использовать
 - *public FSDataOutputStream create(Path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



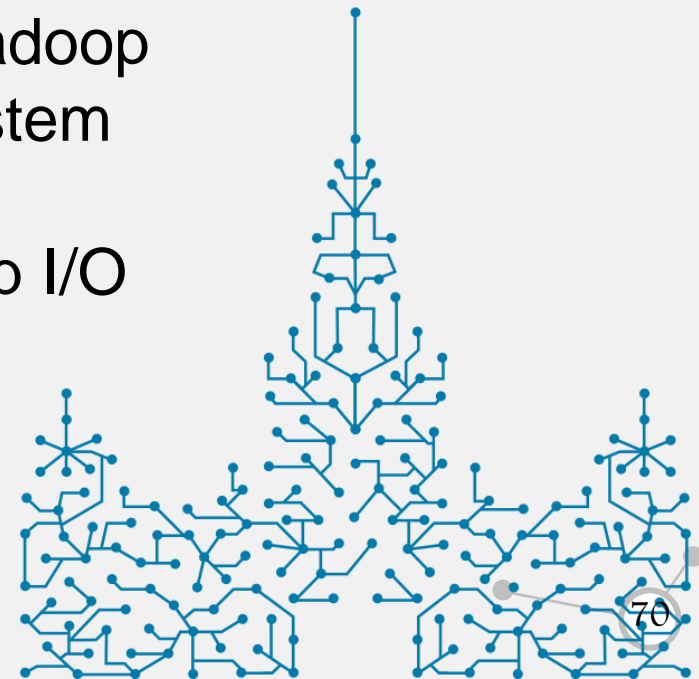
Hadoop: The Definitive Guide

Tom White (Author)

O'Reilly Media; 3rd Edition

Chapter 3: The Hadoop Distributed Filesystem

Chapter 4: Hadoop I/O



Отмечайтесь и оставляйте отзыв

**Спасибо за
внимание!**

Чернов Евгений

e.chernov@corp.mail.ru

Задание



- Написать Java-класс `CpFile`, который осуществляет копирование файла. При этом и источник и место назначения могут располагаться как локально, так и в HDFS

Тестирование:

- Локальный файл **test.txt** копируем в локальный файл **test2.txt** (аналог **cp**):
\$ `hadoop jar cpfile.jar org.sfera.CpFile file:///home/cloudera/test.txt file:///home/cloudera/test2.txt`
- Локальный **test2.txt** копируем в hdfs файл **test3.txt** (аналог **-put**):
\$ `hadoop jar cpfile.jar org.sfera.CpFile file:///home/cloudera/test2.txt test3.txt`
- Hdfs файл **test3.txt** копируем в рваы файл **test4.txt** (аналог **-cp**):
\$ `hadoop jar cpfile.jar org.sfera.CpFile test3.txt test4.txt`
- Hdfs файл **test4.txt** копируем в локальный файл **test5.txt** (аналог **-get**):
\$ `hadoop jar cpfile.jar org.sfera.CpFile test4.txt file:///home/cloudera/test5.txt`