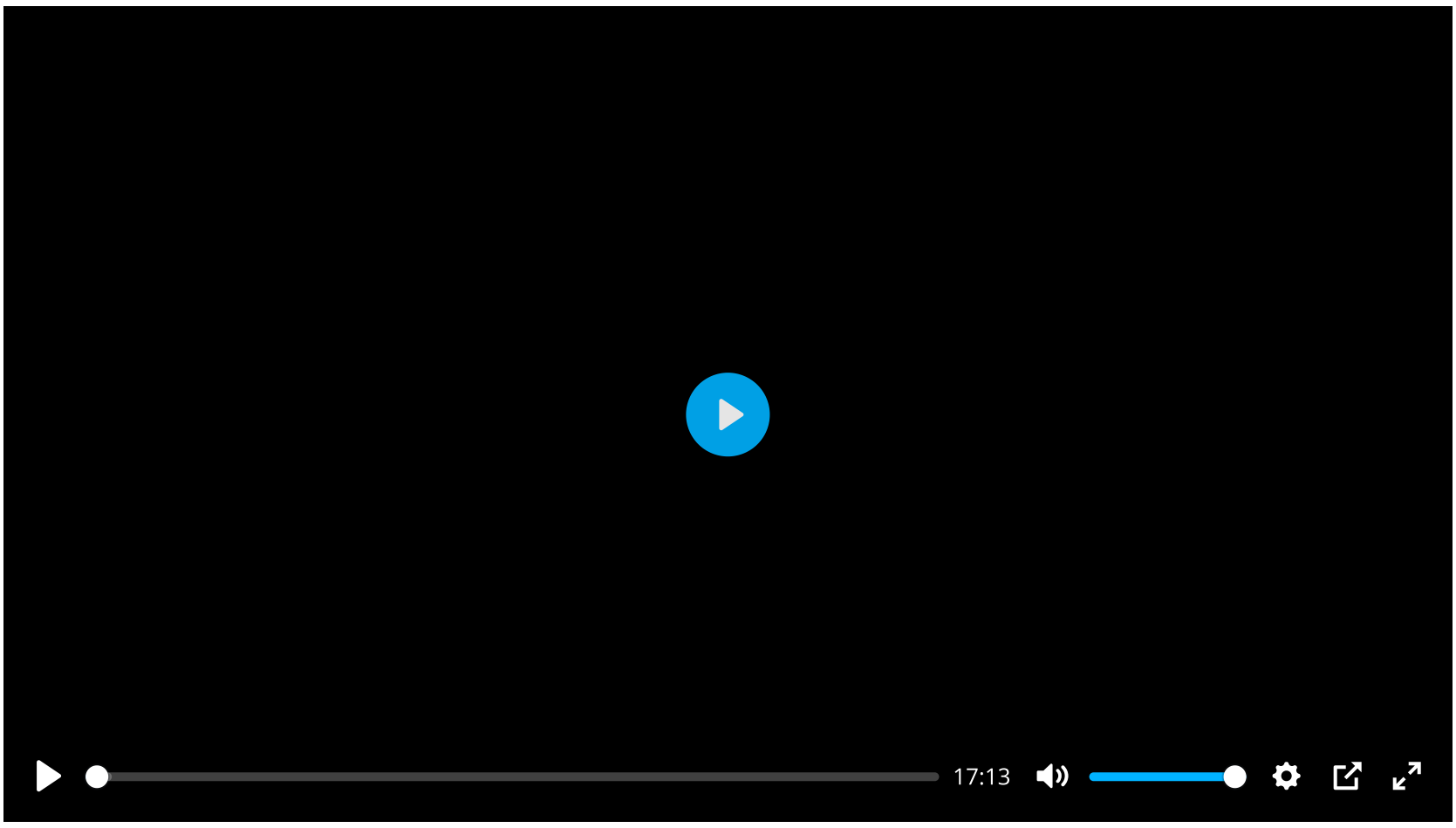
3.3 Технология Map-Reduce



Текстовая расшифровка видео:

ПРОСТЫЕ ЗАДАЧИ

План:

- WordCount на псевдокоде;
- WordCount в стадиях;
- Отладка локально;
- Запускаем все везде и сразу;
- Синяя изолента.

WordCount на псевдокоде

Одна из задач, на которой объясняют MapReduce, – WordCount. Так, например, у нас есть текстовый документ, мы берем из него все слова и считаем сколько раз каждое слово встречается.

Поговорим о методологии MapReduce и о том, как будут выглядеть маппер (map) и редюсер (reduce). В качестве языка используем Python (можно использовать Shell).

Обратите внимание на примеры:

```
def map(doc):
    for word in doc:
        yield word, 1


def reduce(word, values):
    yield word, sum(values)
```

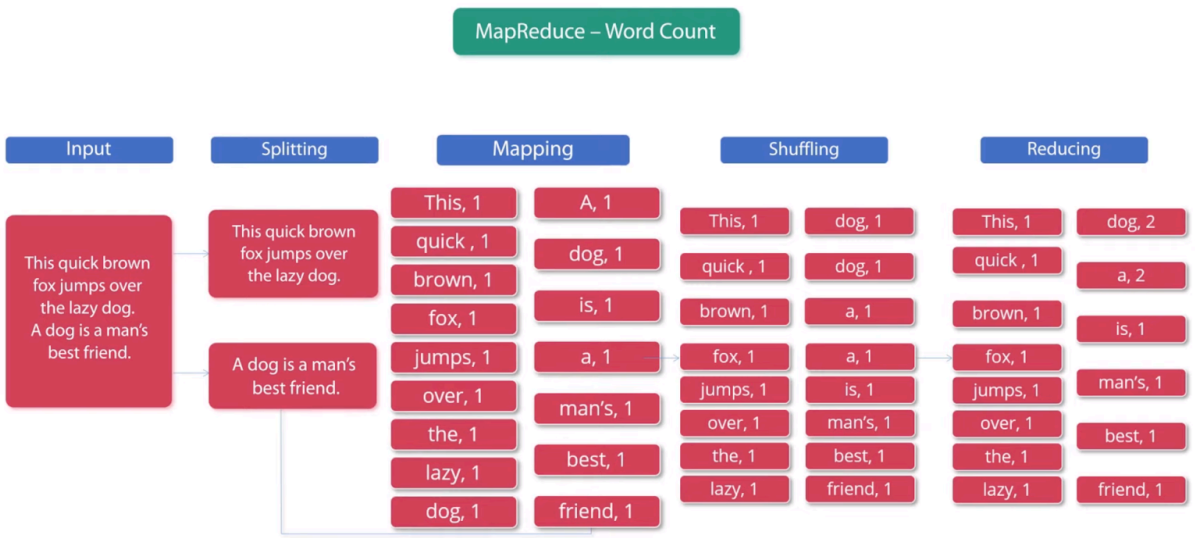
Данный код не является кодом на Python. Это псевдокод, иллюстрирующий идею.

В map для документов и файлов мы перебираем слова и возвращаем пару «ключ-значение», где ключ – слово, значение – единица.

В reduce, поскольку произойдет сортировка и группировка по ключам, для каждого слова все единицы придут в форме набора этих самых единиц. Так, из reduce мы можем возвращать пару «ключ-значение», где ключ – слово, значение – сумма всех единиц.

WordCount в стадиях

Рассмотрим пример того, как описанное выше выглядит по стадиям:



У нас есть текст. Этот текст по умолчанию разбивается на блоки. Каждый маппер приходит в свой блок. Внутри маппера мы для каждого слова генерируем пару. После этого происходит shuffing, и данные по ключам формируются на редюсерах. В рамках редюсера мы все схлопываем и получаем конечный результат.

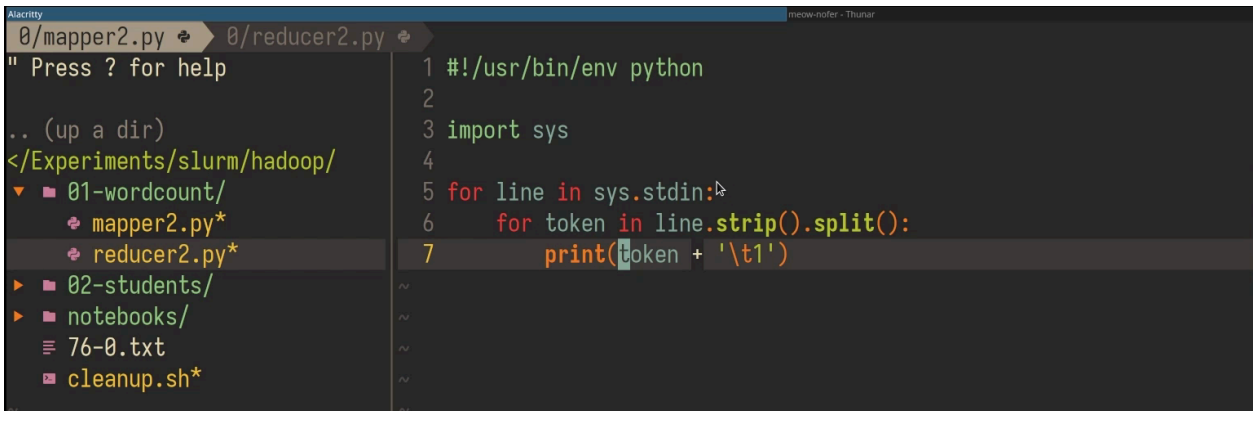
Отладка локально

Чтобы подобное отладить локально нам поможет технология, которая называется «Hadoop Streaming». Это подход, позволяющий писать процессинг **на любом языке**.

Чтобы локально симулировать то, как это будет запускаться в Hadoop-стриминге, мы можем прочитать файл, скормить его в маппер, потом сделать сортировку по ключу, направить все это в редюсер, и сохранить данные из редюсера в выходной файл:

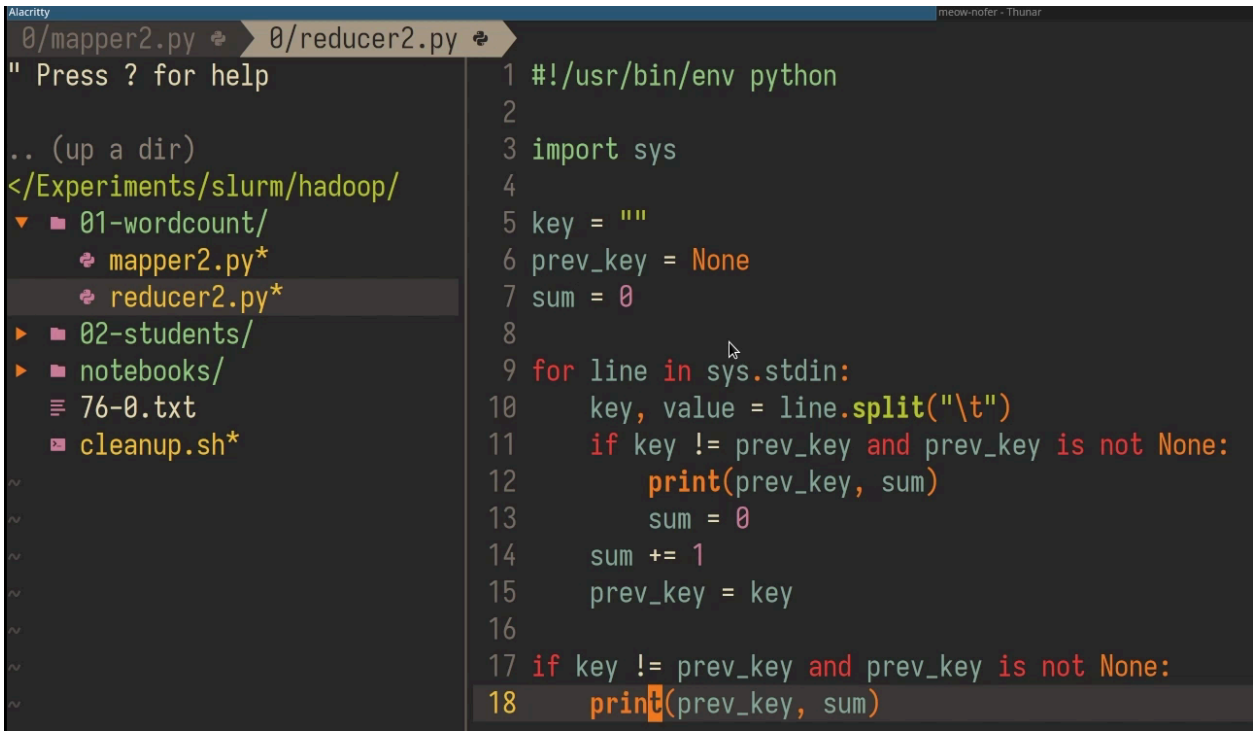
```
~$ cat input_file.txt | python3 mapper.py | sort -k1,1 | python3 reducer.py > output_file.tx
```

Рассмотрим вариант кода и то, как он может выглядеть:



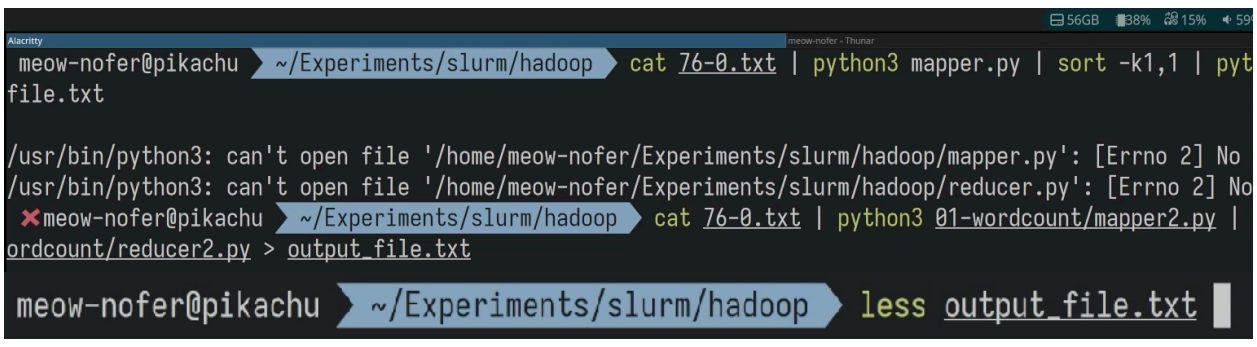
На вход прилетают данные в стандартный ввод. Мы перебираем строчки на стандартном вводе, разбиваем по пробелам и печатаем стандартный вывод (слово, символ табуляции, единичка).

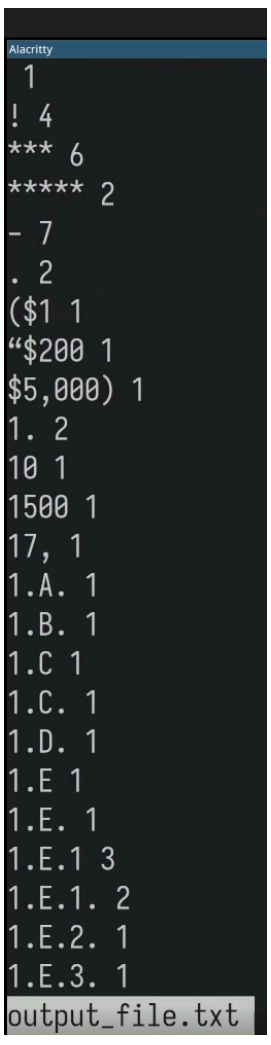
В редюсере все будет несколько сложнее:



Нам прилетают данные, сгруппированные по ключам на стандартный ввод. Нужно понять, когда закончился один ключ и пошел другой. Если ключ поменялся, мы суммируем то, что было до этого и выводим промежуточный результат. Здесь может быть **off-by-one error**.

Протестируем локально:





Данный пример – пример локальной отладки кода. Запустим в Hadoop.

Для того, чтобы все сработало, нам нужно активировать Hadoop-стриминг, а именно указать Hadoop’у код, который мы запускаем, то есть, соответствующий джарник:

```
~$ export HADOOP_STREAMING_JAR=$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.7.3.ja
```

Экспортируем путь в качестве переменного окружения, чтобы не копипастить длинные строчки.

Рассмотрим на примере:

```
meow-nofer@pikachu ~/Experiments/slurm/hadoop export HADOOP_STREAMING_JAR=$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar
```

HADOOP_STREAMING_JAR указывает на Hadoop_Streaming

Запускаем все везде и сразу

Запустим вычисления на Hadoop. Это можно сделать двумя способами:

Через MapReduce:

```
~$ hadoop jar $HADOOP_STREAMING_JAR \
    -input /user/meow-nofer/input \
    -output /user/meow-nofer/output \
    -mapper ./01-wordcount/mapper.py \
    -reducer ./01-wordcount/reducer.py
```

Таким образом мы форсим использование дефолтного шедулера.

Через YARN:

```
~$ yarn jar $HADOOP_STREAMING_JAR \
    -input /user/meow-nofer/input \
    -output /user/meow-nofer/output \
    -mapper mapper2.py \
    -reducer reducer2.py \
    -file 01-wordcount/mapper2.py \
    -file 01-wordcount/reducer2.py
```

Мы указываем запуск джарника для Hadoop-стриминг, далее указываем каталоги для входных и выходных данных. Во входном каталоге уже лежит файл с текстом, при этом выходного каталога существовать не должно. Далее, мы указываем, какая команда должна запускаться в качестве мапера и редюсера.

Чтобы этот код запустился, нам нужно код скопировать дополнительно в облако, на воркеры. **Данная концепция называется «Distributed cache».**

Вопрос: что делать, если Python использует дополнительные зависимости?

Ответ: есть два варианта: либо мы договариваемся с людьми, поддерживающими кластер, и ставим зависимости на все ноды, на которых будут запускаться все job’ы, либо мы можем использовать разные трюки. Например, архивировать зависимости кода и отправить вместе с job’ой в distributed cache. Чтобы запустить job’у сначала нужно установить окружение, а потом – маппер и редюсер.

Рассмотрим пример:

```
Alacritty                                                                    meow-nofer - Thunar
 meow-nofer@pikachu ~/Experiments/slurm/hadoop hdfs dfs -l /user/meow-nofer
-l: Unknown command
 ✖meow-nofer@pikachu ~/Experiments/slurm/hadoop hdfs dfs -ls /user/meow-nofer
Found 1 items
drwxr-xr-x   - meow-nofer supergroup          0 2023-06-10 15:13 /user/meow-nofer/input
 meow-nofer@pikachu ~/Experiments/slurm/hadoop hdfs dfs -ls /user/meow-nofer/input
Found 1 items
-rw-r--r--   1 meow-nofer supergroup     627169 2023-06-10 15:13 /user/meow-nofer/input/76-0.txt
 meow-nofer@pikachu ~/Experiments/slurm/hadoop yarn jar $HADOOP_STREAMING_JAR \
```

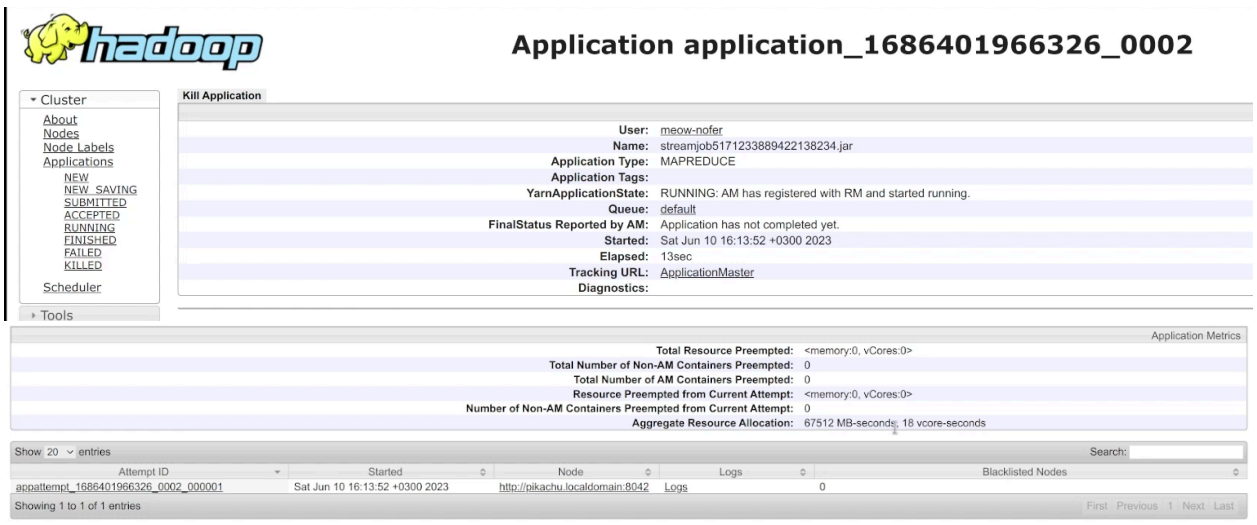
Как мы уже говорили, есть каталог input, а каталога output у нас специально нет. В каталоге input лежит наш файл. Запустим MapReduce job’у:

```
 meow-nofer@pikachu ~/Experiments/slurm/hadoop hdfs dfs -l /user/meow-nofer
-l: Unknown command
 meow-nofer@pikachu ~/Experiments/slurm/hadoop hdfs dfs -ls /user/meow-nofer
Found 1 items
drwxr-xr-x   - meow-nofer supergroup          0 2023-06-10 15:13 /user/meow-nofer/input
 meow-nofer@pikachu ~/Experiments/slurm/hadoop hdfs dfs -ls /user/meow-nofer/input
Found 1 items
-rw-r--r--   1 meow-nofer supergroup     627169 2023-06-10 15:13 /user/meow-nofer/input/76-0.txt
 meow-nofer@pikachu ~/Experiments/slurm/hadoop yarn jar $HADOOP_STREAMING_JAR \
    -input /user/meow-nofer/input \
    -output /user/meow-nofer/output \
    -mapper "python mapper2.py" \
    -reducer "python reducer2.py" \
    -file 01-wordcount/mapper2.py \
    -file 01-wordcount/reducer2.py
```

```
23/06/10 16:13:51 WARN streaming.StreamJob: -file option is deprecated, please use generic option -fil
23/06/10 16:13:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java clas
ses where applicable
packageJobJar: [01-wordcount/mapper2.py, 01-wordcount/reducer2.py, /tmp/hadoop-unjar2937110002587779142/] [] /tmp/streamjob5171
233889422138234.jar tmpDir=null
23/06/10 16:13:51 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/06/10 16:13:51 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/06/10 16:13:52 INFO mapred.FileInputFormat: Total input paths to process : 1
23/06/10 16:13:52 INFO mapreduce.JobSubmitter: number of splits:2
23/06/10 16:13:52 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1686401966326_0002
23/06/10 16:13:52 INFO impl.YarnClientImpl: Submitted application application_1686401966326_0002
23/06/10 16:13:52 INFO mapreduce.Job: The url to track the job: http://pikachu.localdomain:8088/proxy/application_1686401966326
_0002/
23/06/10 16:13:52 INFO mapreduce.Job: Running job: job_1686401966326_0002
23/06/10 16:13:57 INFO mapreduce.Job: Job job_1686401966326_0002 running in uber mode : false
23/06/10 16:13:57 INFO mapreduce.Job:  map 0% reduce 0%
```

Видим, что что-то уже засабмитилось.

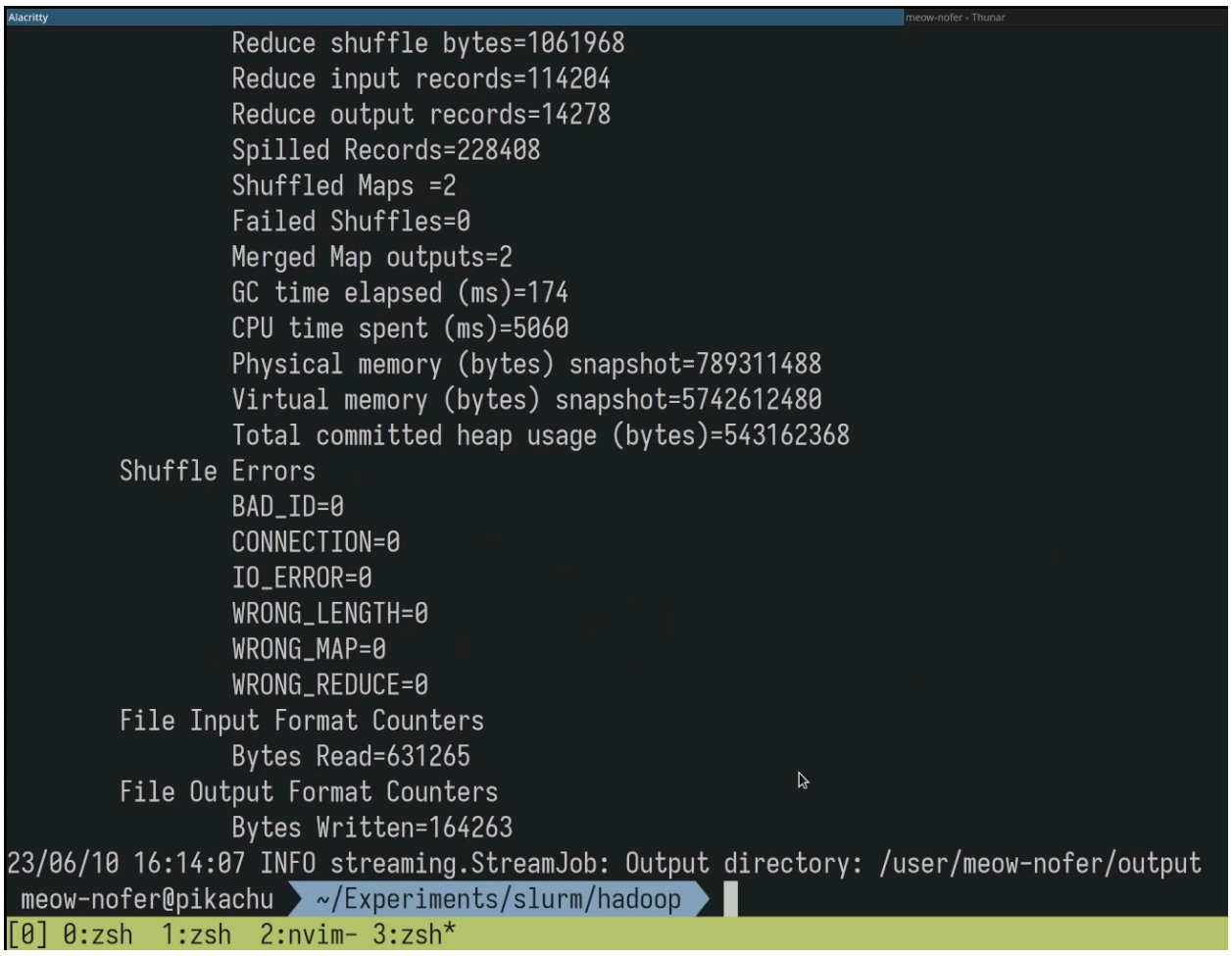
Если мы зайдем в UI YARN, то увидим, что здесь есть job’a:



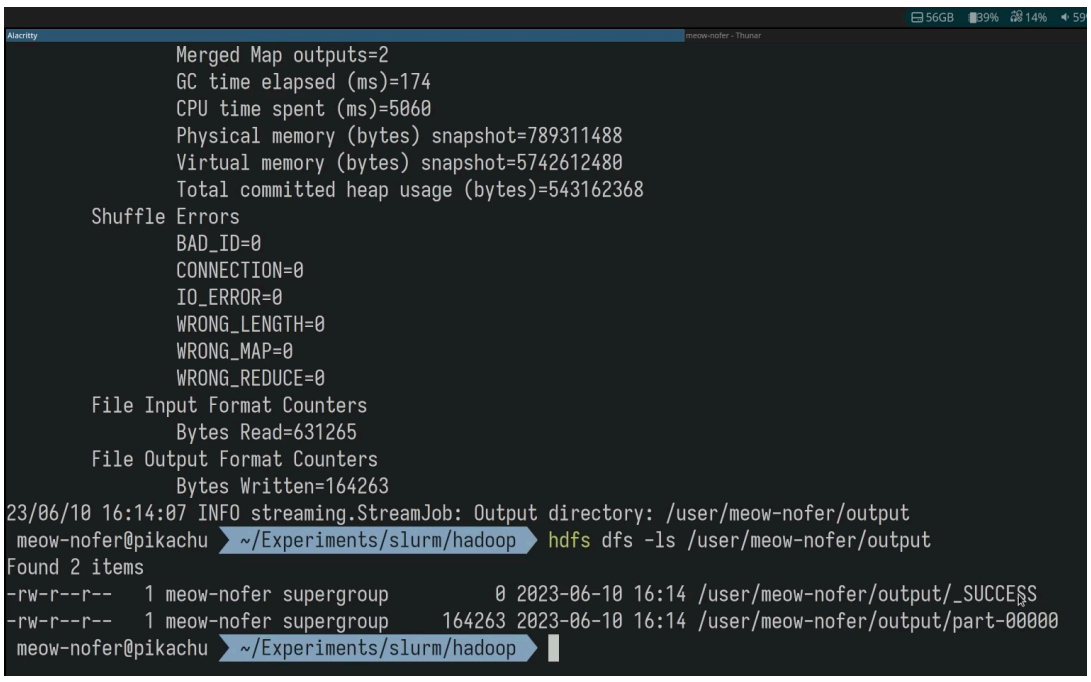
Она уже отработала, писала по стадиям:

```
    -file 01-wordcount/mapper2.py \
    -file 01-wordcount/reducer2.py
23/06/10 16:13:51 WARN streaming.StreamJob: -file option is deprecated, please use generic option -fi
23/06/10 16:13:51 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform
ses where applicable
packageJobJar: [01-wordcount/mapper2.py, 01-wordcount/reducer2.py, /tmp/hadoop-unjar2937110002587779
233889422138234.jar tmpDir=null
23/06/10 16:13:51 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/06/10 16:13:51 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
23/06/10 16:13:52 INFO mapred.FileInputFormat: Total input paths to process : 1
23/06/10 16:13:52 INFO mapreduce.JobSubmitter: number of splits:2
23/06/10 16:13:52 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1686401966326_0002
23/06/10 16:13:52 INFO impl.YarnClientImpl: Submitted application application_1686401966326_0002
23/06/10 16:13:52 INFO mapred.FileInputFormat: Total input paths to process : 1
23/06/10 16:13:52 INFO mapreduce.JobSubmitter: number of splits:2
23/06/10 16:13:52 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1686401966326_0002
23/06/10 16:13:52 INFO impl.YarnClientImpl: Submitted application application_1686401966326_0002
23/06/10 16:13:52 INFO mapreduce.Job: The url to track the job: http://pikachu.localdomain:8088/proxy/application_1686401966326
_0002/
23/06/10 16:13:52 INFO mapreduce.Job: Running job: job_1686401966326_0002
23/06/10 16:13:57 INFO mapreduce.Job: Job job_1686401966326_0002 running in uber mode : false
23/06/10 16:13:57 INFO mapreduce.Job:  map 0% reduce 0%
23/06/10 16:14:01 INFO mapreduce.Job:  map 50% reduce 0%
23/06/10 16:14:04 INFO mapreduce.Job:  map 100% reduce 0%
23/06/10 16:14:07 INFO mapreduce.Job:  map 100% reduce 100%
23/06/10 16:14:07 INFO mapreduce.Job: Job job_1686401966326_0002 completed successfully
23/06/10 16:14:07 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=1061962
[0] 0:zsh  1:zsh  2:nvim- 3:[tmux]*                                                    "pikachu" 16:14 10-июн-23
```

В некоторых ситуациях редюсер может запуститься, чем отработали все мапперы:



Поскольку job’а отработала, у нас должен появиться каталог «output», в котором должен лежать результат:

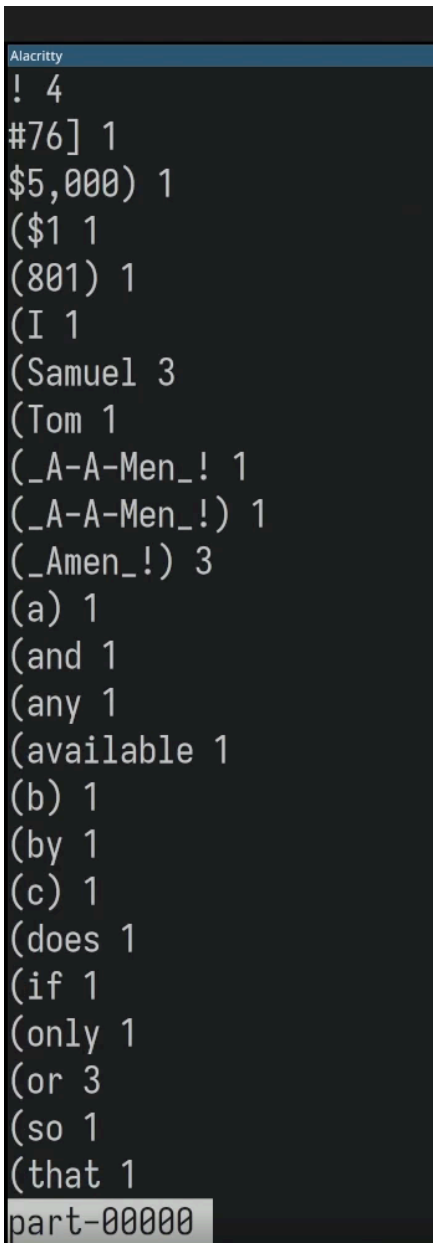


Лежит там следующее:

- **Файл «SUCCESS»**. Данный файл создается, чтобы сообщить о завершении работы job’ы. Если данного файла нет, значит какой-то процесс все еще работает. Это важный механизм синхронизации!
- **Файл «part-00000».**

Наличие лишь одного файла говорит о работе всего лишь одного воркера.

Скачаем файл «part-00000» и посмотрим на его содержимое:



Мы видим результат того, какое слово и в каком количестве оно нам встречалось в исходном файле.

Синяя изолента

Решим задачу:

У нас есть табель с оценками:

Holmes	4
Lopez	3
Holmes	2
Wilson	5
Robinson	5
Weber	3
Sanders	4
Ball	2
Butler	5
Wilson	5

Отличником считается студент, у которого средний балл выше 4.5. Как найти всех отличников?

Решение:

Если прикинуть на MapReduce, то на входе мы можем выдавать пары для студентов (студент-оценка). При этом данные сами сгруппируются по ключам в Reducer. В Reducer мы берем все значения, которые для данного ключа прилетели, считаем для них среднее. Если среднее полученное значение выше 4.5, то мы выводим результат для этого студента:

```
# mapper
map(x, score) →
      emit (x, score)


# reducer
reduce(x, scores) →
avg = average(scores)
if avg >= 4.5:
      emit(x, avg)
```

Почему «синяя изолента»?

Есть шутка, что все можно починить синей изолентой, а если починить не получается, значит вы просто мало взяли изоленты. Это аллегория отлично иллюстрирует MapReduce. Им можно решить любую задачу, а если задача не решается, то следует взять еще больше MapReduce.

Единственный минус: все будет записываться на диск.

Как вам урок?

Изучил, далее