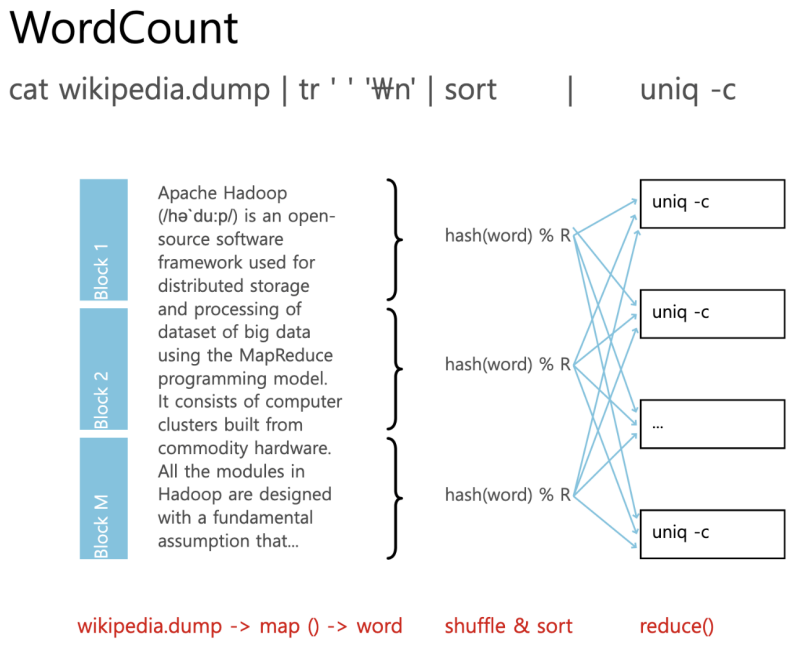
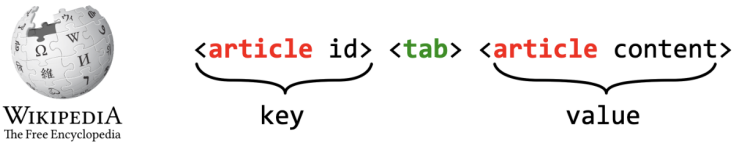
Week2: Hadoop MapReduce Application Tuning: Job Configuration, Comparator, Combiner, Partitioner

Combiner





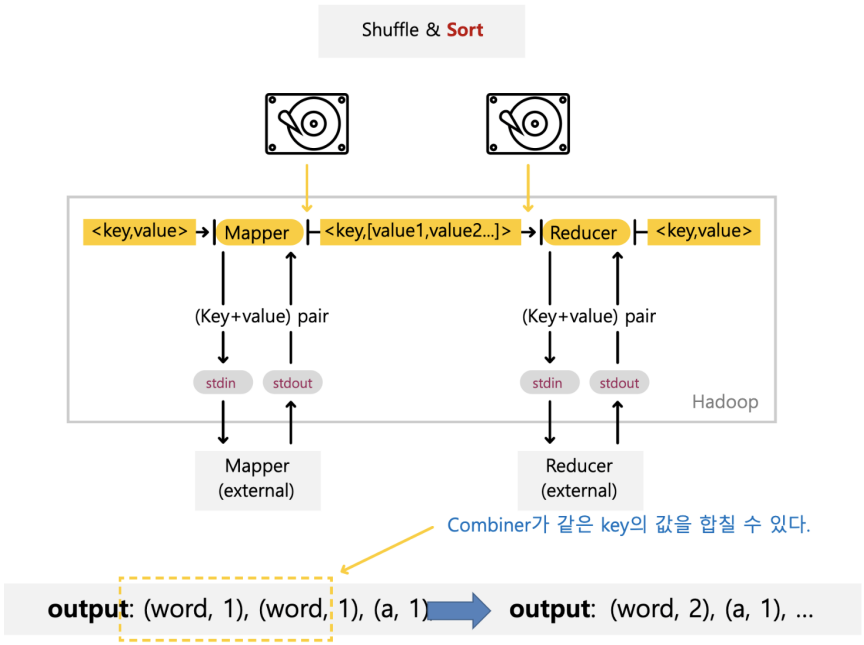
input: word word a word b c d word d e...

Mapper (Python): reporter_mapper.py

```
from __future__ import print_function
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = content.split()
    for word in words:
            print(word, 1, sep="\t")
```

output: (word, 1), (word, 1), (a, 1), ...



input: word word a word b c d word d e...

Mapper (Python): reporter_mapper.py Mapper에서 Combiner역할까지 구현한 예시

```
from __future__ import print_function
from collections import Counter
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = content.split()
    counts = Counter(words)
    for word, word_count in counts.items():
        print(word, word_count, sep="\t")
```

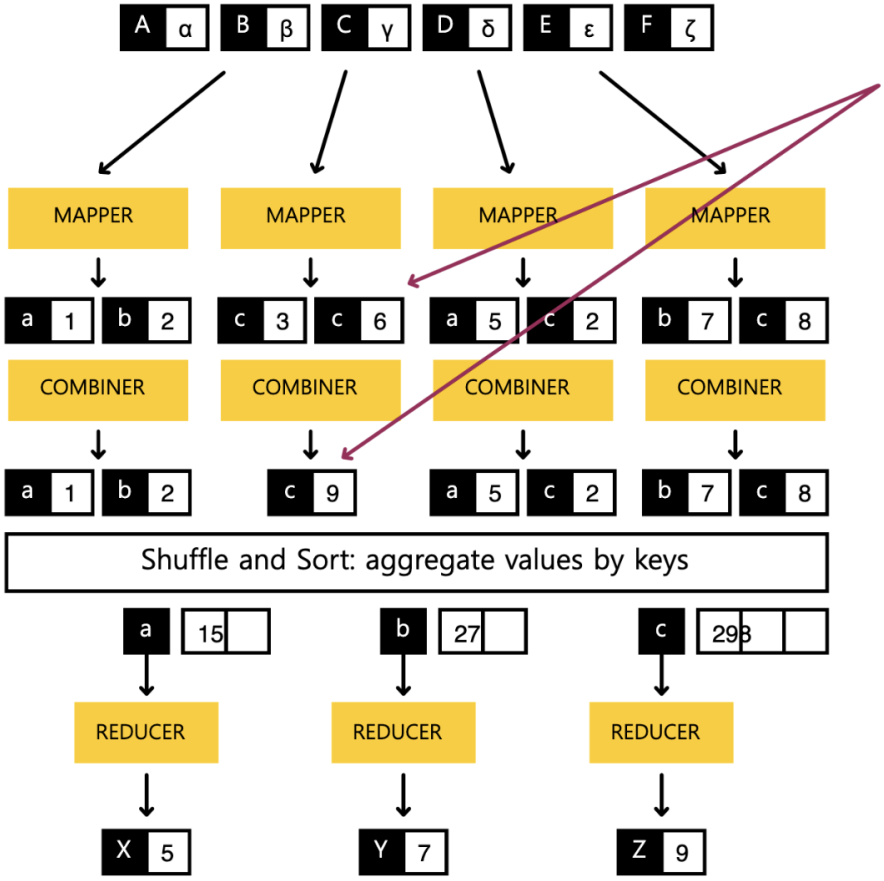
Python의 Counter collections를 활용

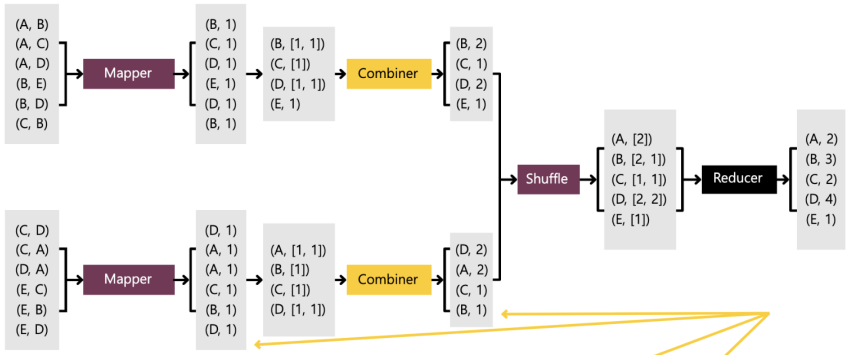
각 단어마다 합계를 자동으로 계산

output: (a, 1), (b, 1), (c, 1), (d, 2), (e, 1), (word, 4), ...

Combiner는 I/O operation을 줄이고 네트워크량을 줄여 전체적인 시간 감소가 가능

	without Combiner	with Combiner
Wall time (sec)	**935**	**528**
CPU time (sec)	**9790**	**6584**
Local FS Read (MB)	**3006**	**1324**
Local FS Write (MB)	**4527**	**1963**
Peek Map phys. memory (MB)	**526**	**606**
Peek Map virt. memory (MB)	**2131**	**2144**
Peek Reduce phys. memory (MB)	**2744**	**631**
Peek Reduce virt. memory (MB)	**3196**	**3194**





- read: [(k_in, v_in), ...]
- map: (k_in, v_in) —> **[(k_interm, v_interm), ...]**
- combiner: **(k_interm, [(v_interm, ...)])** —> **[(k_interm, v_interm), ...]**
- Shuffle & Sort: sort and group by k_interm
- reduce: **(k_interm, [(v_interm, ...)])** —-> [(k_out, v_out), ...]

```
yarn jar $HADOOP_STREAMING_JAR \
                        -files mapper.py,reducer.py \
                        -mapper 'python mapper.py' \
                        -reducer 'python reducer.py' \
                        -combiner 'python reducer.py' \
                        -numReduceTasks 2 \
                        -input griboedov.txt \
                        -output word_count
```

위 예제는 combiner와 reducer의 역할이 같다.
(같은 key값의 value들을 합치는 것)

```
Map-Reduce Framework
        Map input records=2681
        Map output records=182
        Map output bytes=1218
        Map output materialized bytes=955
        Input split bytes=126
        Combine input records=182
        Combine output records=99
        Reduce input groups=99
        Reduce shuffle bytes=955
        Reduce input records=99
        Reduce output records=99
```

As you could have mentioned the input to the MapReduce application is griboedov.txt. who is Griboedov? his comedy "Woe from Wit" has been used as an input to the MapReduce WordCount application.

<article id> <tab> <article content>

key value

input: word word a word b c d word d e...

mean

각 단어의 평균 사용 개수를 구하는 문제

output: (word, 3.5), (a, 0.5), ...

input: word word a word b c d word d e...

Mapper (Python): reporter_mapper.py

Mapper에서 Combiner역할까지 구현한 예시

```
from __future__ import print_function
from collections import Counter
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = content.split()
    counts = Counter(words)
    for word, word_count in counts.items():
        print(word, word_count, sep="\t")
```

Python의 Counter collections를 활용

각 단어마다 합계를 자동으로 계산

output: (a, 1), (b, 1), (c, 1), (d, 2), (e, 1), (word, 4), ...

input: word word a word b c d word d e...

Mapper (Python): mapper.py

```
from __future__ import print_function
from collections import Counter
import sys

for line in sys.stdin:
    article_id, content = line.split("\t", 1)
    words = content.split()
    counts = Counter(words)
    for word, word_count in counts.items():
print(word, word_count, sep="\t")
print(word, 1, word_count, sep="\t")
```

Mapper는 article을 count하기 위한 변수도 출력해야 한다.

~~**output**: (a, 1), (d, 2), (word, 4),~~ ...

output: (a, (1, 1)), (d, (1, 2)), (word, (1, 4)), ...

Mapper (Python): combiner.py

```
from __future__ import print_function
import sys

current_word = None
word_count, article_count = 0, 0

for line in sys.stdin:
        word, articles, counts = line.split("\t", 2)
        articles, counts = int(articles), int(counts)
    if word == current_word:
       word_count += counts
       article_count += articles
    else:
        if current_word:
            assert len(current_word.rstrip()) > 0
            print(current_word, word_count , article_count, sep="\t")
        current_word = word
        word_count = counts
        article_count = articles

if current_word:
    print(current_word, word_count , article_count, sep="\t")
```

Combiner는 article의 합계와 단어의 합계를 구한다.

(a, (1, 1))
(a, (1, 3))
(a, (1, 5))
(d, (1, 2))
(d, (1, 3))
(d, (1, 3))
...

→

(a, (3, 9))
(d, (3, 8))
...

Mapper (Python): reducer.py

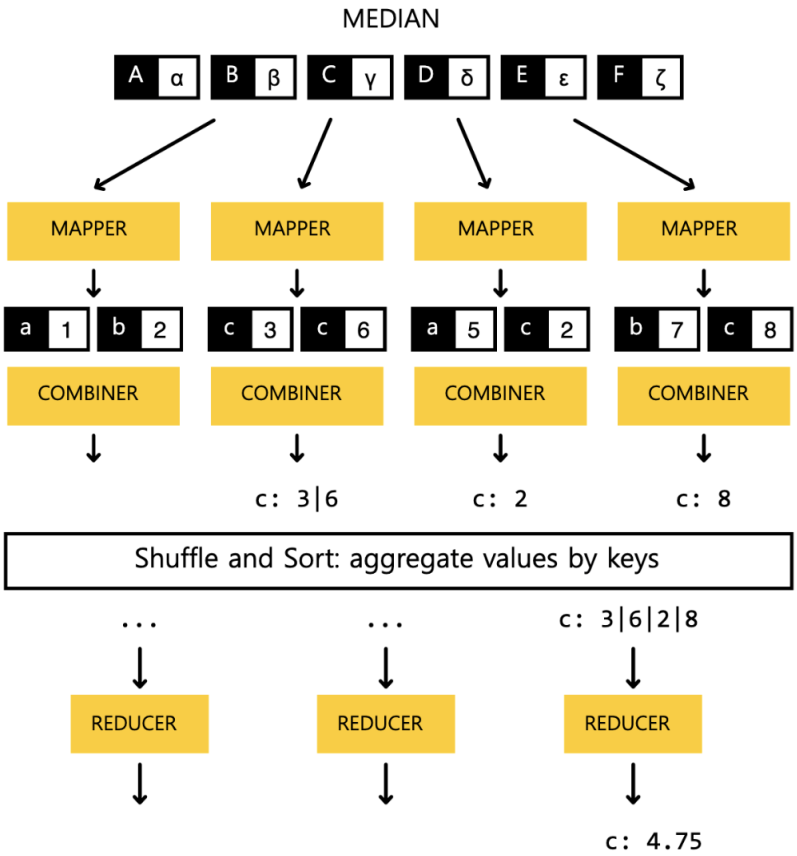
```
from __future__ import print_function
import sys

current_word = None
word_count, article_count = 0, 0

for line in sys.stdin:
        word, articles, counts = line.split("\t", 2)
        articles, counts = int(articles), int(counts)
    if word == current_word:
       word_count += counts
       article_count += articles
    else:
        if current_word:
            assert len(current_word.rstrip()) > 0
            print(current_word, word_count , article_count, sep="\t")
        current_word = word
        word_count = counts
        article_count = articles

if current_word:
    print(current_word, word_count , article_count, sep="\t")
```

Reducer는 Combiner와 코드가 같다.



median의 경우 combiner는 쓸모 없음

Partitioner

Collocations

단어들의 결합이 의미를 갖는 것 => 하나의 단어로 인식해야 한다.

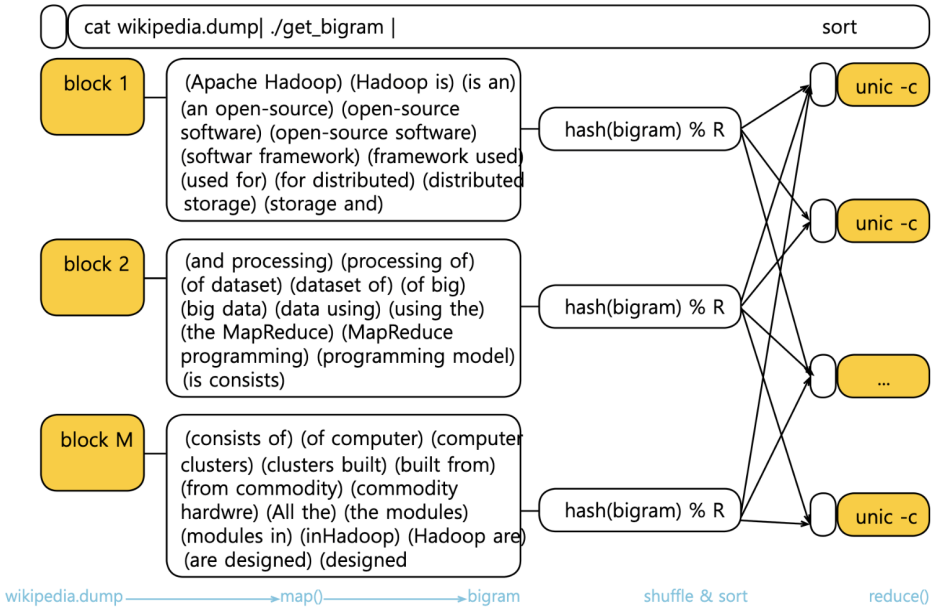
natural English	unnatural English
the fast train	the ~~quick~~ train
fast food	~~quick~~ food
a quick shower	a ~~fast~~ shower
a quick meal	a ~~fast~~ meal

New and York

United and States

WordCount

bigram : 두 개의 단어가 하나의 의미를 갖는 것



input

```
$ head -c 100 wikipedia_sample.txt
12 Anarchism Anarchism is often defined as a political
philosophy which holds the state to ...
```

Mapper (Python): bigram_mapper.py

```
from __future__ import print_function
import re
import sys
for line in sys.stdin:
 article_id, content = line.split("₩t", 1)
 words = re.split("₩W+", content)
 for index in range(len(words) - 1):
  print(words[index], words[index + 1], 1, sep="₩t")
```

output

```
$ head word_count/part-00000
Anarchism Anarchism 1
Anarchism is 1
is often 1
```

문제점 : 첫번째 단어만 Key로 인식하기 때문에 두번째 단어가 제대로 정렬되는지 보장할 수 없다.

Mapper (Python): inmemroy_bigram_reducer.py

```
from __future__ import print_function
from collections import Counter
import sys

current_word = None
bigram_count = Counter()

for line in sys.stdin:
 first_word, second_word, counts = line.split("₩t", 2)
 counts = Counter({second_word: int(counts)})
 if first_word == current_word:
  bigram_count += counts
 else:
  if current_word:
   for second_word, bigram_count in bigram_count.items():
    print(current_word, second_word,
```

e.g.)

Anarchism Anarchism 1
Anarchism is 1
Anarchism Anarchism 1
Anarchism test 1

문제점 : 첫번째 단어만 Key로 인식하기 때문에 두번째 단어가 제대로 정렬되는지 보장할 수 없다.

따라서, reducer에서 두번째 단어를 메모리에 저장하면서 값을 업데이트 해야 한다.

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
        -files bigram_mapper.py,inmemory_bigram_reducer.py \
        -mapper 'python bigram_mapper.py' \
        -reducer 'python inmemory_bigram_reducer.py' \
        -numReduceTasks 5 \
        -input wikipedia_sample.txt \
        -output word_count \
```

```
$ grep $'^new₩t' word_count/* | sort -nrk3,3 | head -4
word_count/part-00001:new york 112
word_count/part-00001:new world 15
word_count/part-00001:new jewrsey 12
word_count/part-00001:new constitution 12
$ grep $'₩tyork₩t' word_count/* | sort -nrk3,3 | head
word_count/part-00001:new york 112
word_count/part-00004:sergeant york 1
```

key 정렬 문제 해결방법 :

두 개의 단어를 Shuffle&Sort를 위한 Key로 인식하도록 config를 설정할 수 있다.

=> Combiner가 메모리 사용없이 연산을 제대로 수행할 수 있도록 할 수 있다.

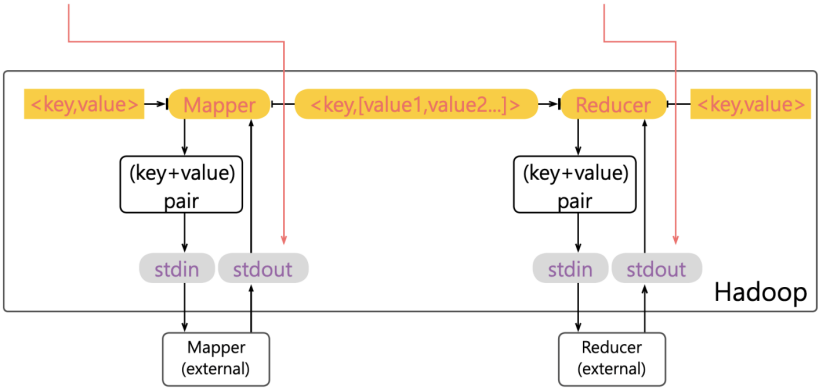
```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
        -D stream.num.map.output.key.fields=2 \
        -files bigram_mapper.py,bigram_reducer.py \
        -mapper 'python bigram_mapper.py' \
        -reducer 'python bigram_reducer.py' \
        -numReduceTasks 5 \
        -input wikipedia_sample.txt \
        -output word_count \
```

```
word_count/part-00001:new york 112
word_count/part-00001:new world 15
word_count/part-00002:new constitution 12
word_count/part-00000:new jersey 12
```

separator는 input data에서 key, value 구분 문자를 의미한다.

-D stream.map.output.field.separator=.
-D stream.num. map.output.key.fields=1

-D stream.reduce.output.field.separator=.
-D stream.num. reduce.output.key.fields=2



IPv4 address (example): 69.89.31.226

input

```
$ cat subnet.txt
1.2.3.4
2.3.4.5
3.4.5.6
4.5.6.7
```

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
        -D stream.map.output.field.separator=. \
        -D stream.num.map.output.key.fields=1 \
        -D stream.reduce.output.field.separator=. \
        -D stream.num.reduce.output.key.fields=2 \
        -files identity_mr.py \
        -mapper 'python identity_mr.py' \
        -reducer 'python identity_mr.py' \
        -numReduceTasks 2 \
        -input subnet.txt \
        -output subnet_out \
```

output

```
cat subnet_out/*
2   3.4   5
4   5.6   7
1   2.3   4
3   4.5   6
```

input

```
$ cat subnet.txt
1.2.3.4
2.3.4.5
3.4.5.6
4.5.6.7
```

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
        -D stream.map.output.field.separator=. \
        -D stream.num.map.output.key.fields=1 \
        -D stream.reduce.output.field.separator=. \
        -D stream.num.reduce.output.key.fields=2 \
        -files identity_mr.py \
        -mapper 'python identity_mr.py' \
        -reducer 'python identity_mr.py' \
        -numReduceTasks 2 \
        -input subnet.txt \
        -output subnet_out \
```

output

```
cat subnet_out/*
2   3.4   5
4   5.6   7
1   2.3   4
3   4.5   6
```

partition.keypartitioner : Partition을 나누는 기준 설정

-k : key값이라는 의미
1, 2 : 첫번째 단어에서 두번째 단어까지

man sort | grep KEYDEF 방식과 같음

e.g.) 1.2, 1.2 : 첫번째 단어에서 2번째 글자를 정렬key로 선택한다

```
yarn --config $HADOOP_EMPTY_CONFIG jar $HADOOP_STREAMING_JAR \
        -D map.output.key.field.separator=. \
        -D mapreduce.partition.keypartitioner.options=-k1.2,1.2 \
        -files identity_mr.py \
        -mapper 'python identity_mr.py' \
        -reducer 'python identity_mr.py' \
        -numReduceTasks 2 \
        -input subnet.txt \
        -output subnet_out \
        -partitioner org.apache.hadoop.mapred.lib.KeyFieldBasedPartitioner
```

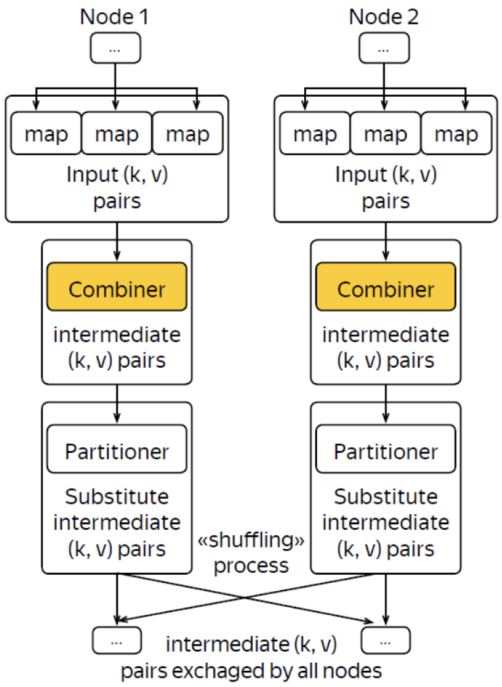
input	output
$ cat subnet.txt	cat subnet_out/*
1a.2.3.4	3b.4.5.6
2a.3.4.5	4b.5.6.7
	1a.2.3.4
	2a.3.4.5

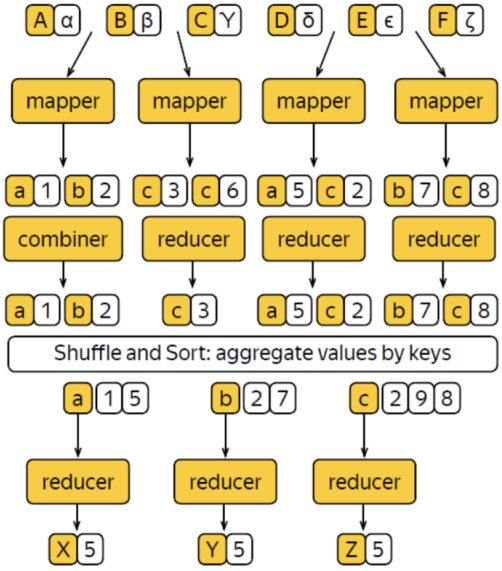
참고 : Partition의 개수 = reducer의 개수(자동)

mapper에서는 각 reducer로 보낼 데이터를 partition으로 구분한다

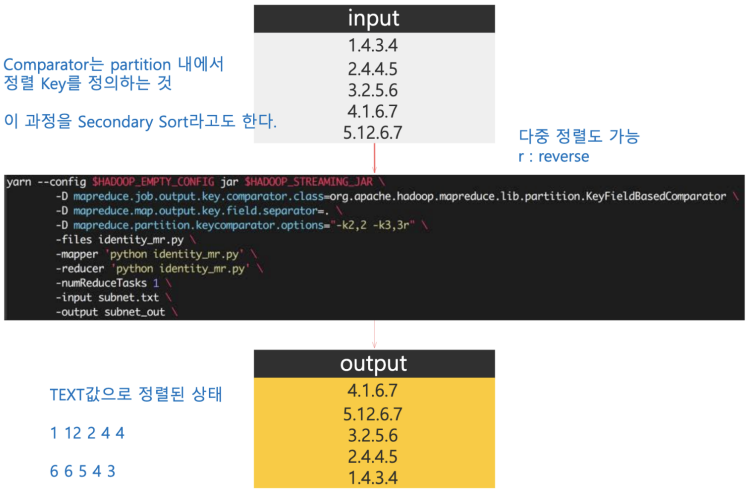
각 Partition에서 Comparator 기준으로 다시 정렬된다.

0 Apr 1 14:59 _SUCCESS
20 Apr 1 14:59 part-00001
20 Apr 1 14:59 part-00000





Comparator

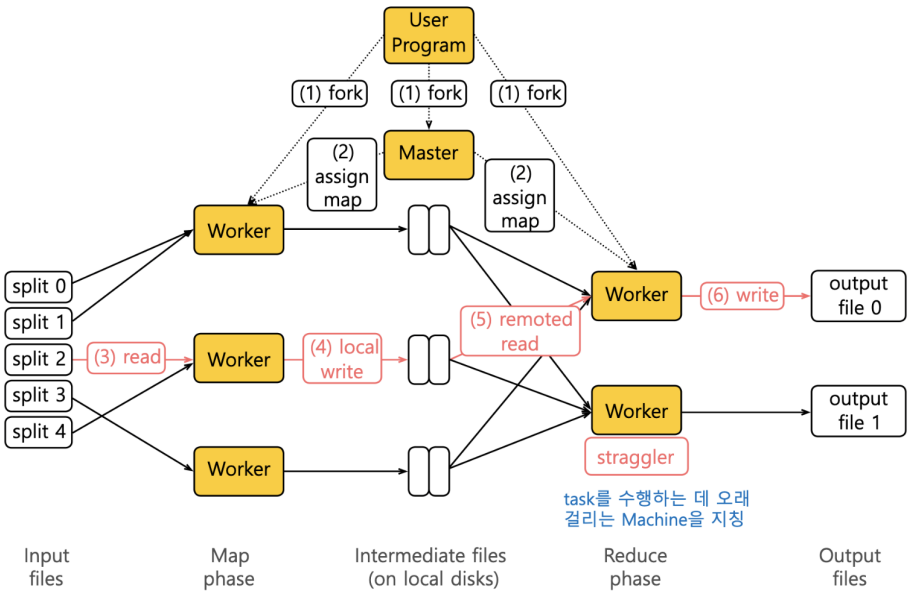


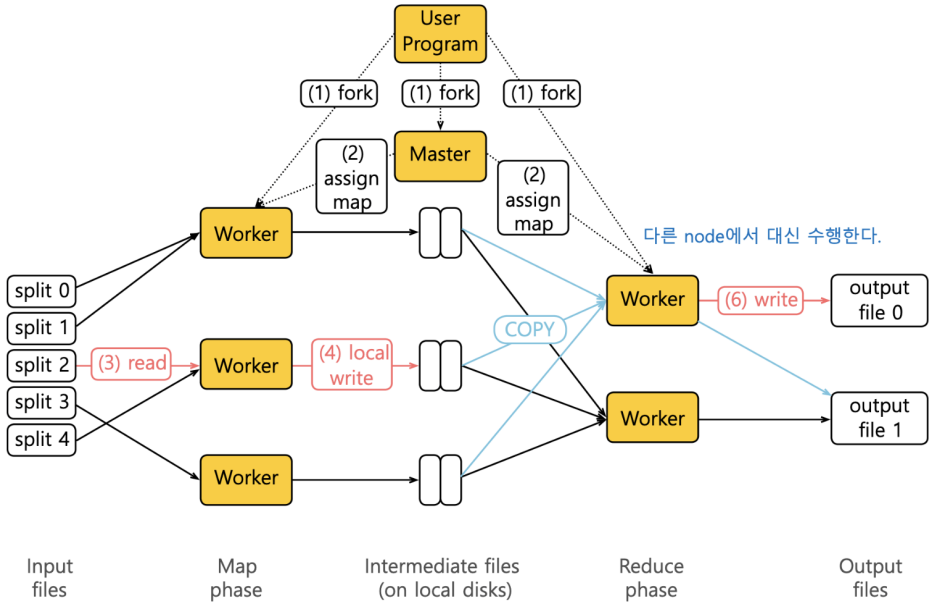
If you want to sort a collection of IPv4 addresses by the second and third octets numerically and in the reverse order, then you need to provide the following arguments for KeyFieldBasedComparator:

- ○ -n2,3r
- ○ -n1,2r
- ◉ -k2,3nr
- ○ -k2,4nr

Speculative Execution, Backup Tasks (추측에 근거한)

- task가 너무 오래 걸린다고 판단 → 다른 node에서 task를 대신 수행하도록 하는 것
- MapReduce Application이 끝나갈 때쯤 speculation 실행





- Backup Tasks

due to the deterministic behavior of the mapper and reducer, you can easily re-execute straggler bofdy of work on other node.

in this case, the worker which processes data, they first outputs data to a distributed file system.

all the other concurrent executions will be killed.

of course, the mapreduce framework is not going to have a copy for each running task.

⇒ it is only used when a mapreducer application is close to completion

30-40% faster

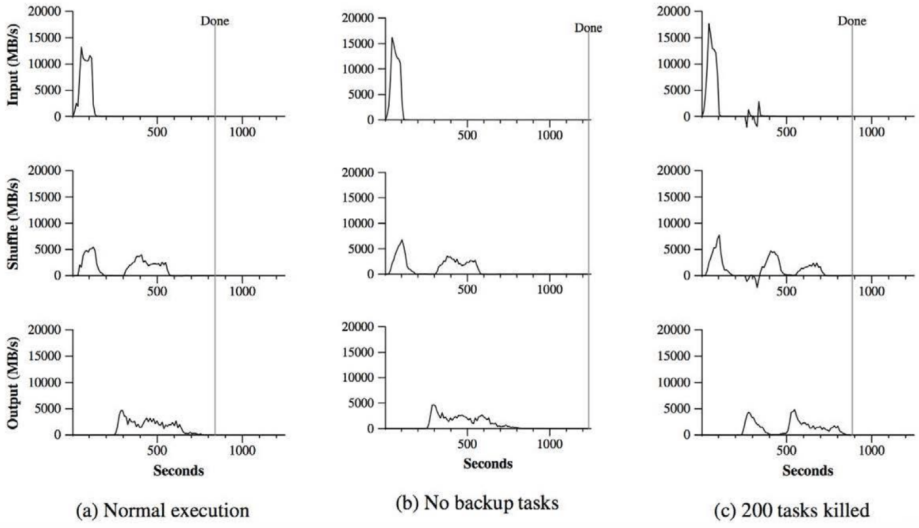


Figure 3: Data transfer rates over time for different executions of the sort program

```
yarn jar $MR_STREAMING_JAR \
        -D mapreduce.map.speculative="true" \
        -D mapreduce.job.speculative.speculative-cap-running-tasks=0.99 \
        -D mapreduce.job.speculative.speculative-cap-total-tasks=0.99 \
        -D mapreduce.job.speculative.retry-after-no-speculate=10 \
        -files flaky_mapper.py \
        -mapper 'python flaky_mapper.py' \
        -output flaky_mr \
        -numReduceTasks 0 \
        -input /data/wiki/en_articles_part
```

›› mapreduce.map.speculative (default:"true")

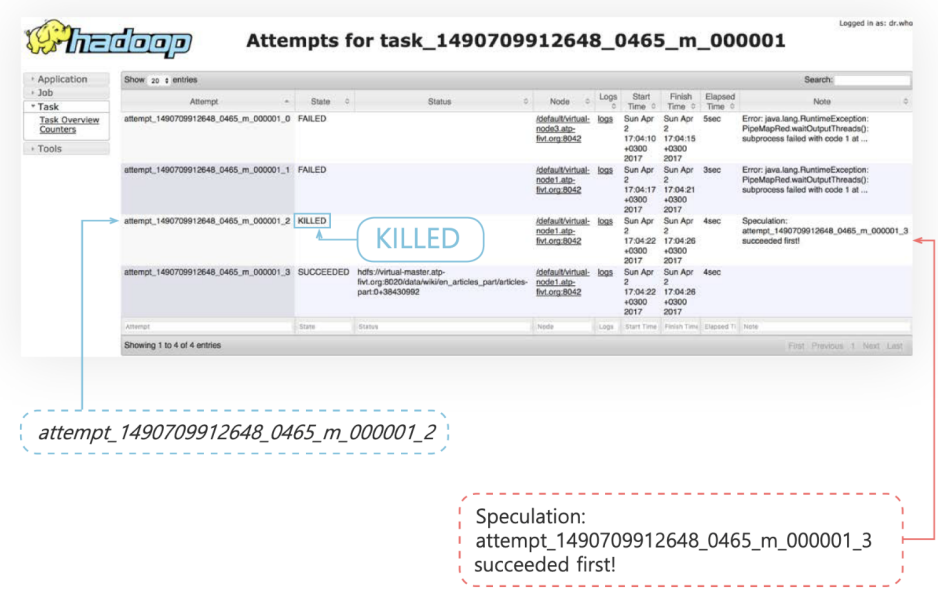
›› mapreduce.reduce.speculative (default:"true")

›› mapreduce.job.speculative.speculative-cap-running-tasks
  ››(default: 0.1) 퍼센트

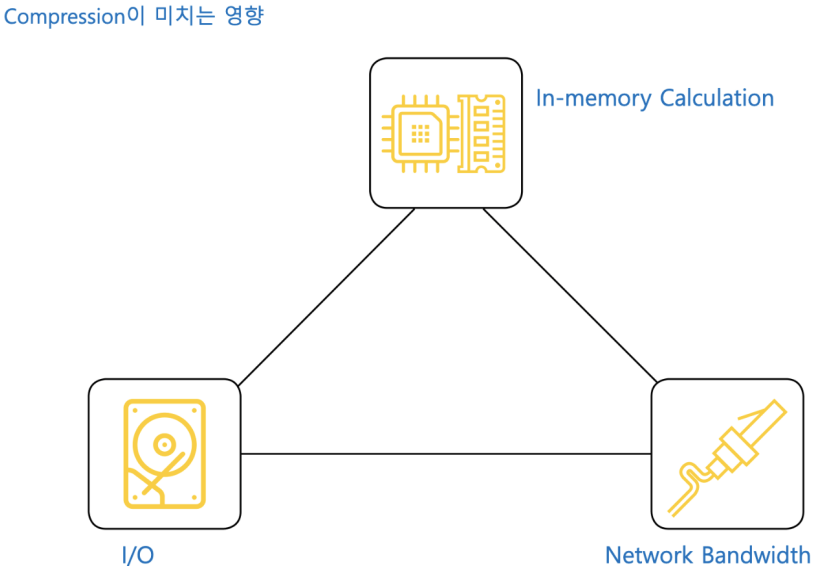
›› mapreduce.job.speculative.speculative-cap-total-tasks
  ››(default: 0.01) 퍼센트

›› mapreduce.job.speculative.retry-after-no-speculate
  ››(default: 1000, ms)

›› mapreduce.job.speculative.retry-after-speculate
  ››(default: 15000, ms)



Compression



Splittable : 압축파일을 여러 mapper에게 나눠 풀 수 있다. <-> 전체 파일 압축을 하나의 mapper에서 풀어야 한다.
(임의의 위치에서 파일을 읽을 수 있음) (처음부터 파일을 읽어야 한다.)

Compression Format	Splittable	Comments
.deflate .gz (gzip)	NO	Uses DEFLATE algorithm
.bz2 (bzip)	YES	more effective than gzip, but slower compression
.lzo 일반적인 하둡 시나리오에 적합	YES* (Write once read many)	decompres way faster than gzip, compres less efficient
.snappy	NO	faster than LZO for decompression

flags: -1(optimised for speed)...-9(optimised for space)

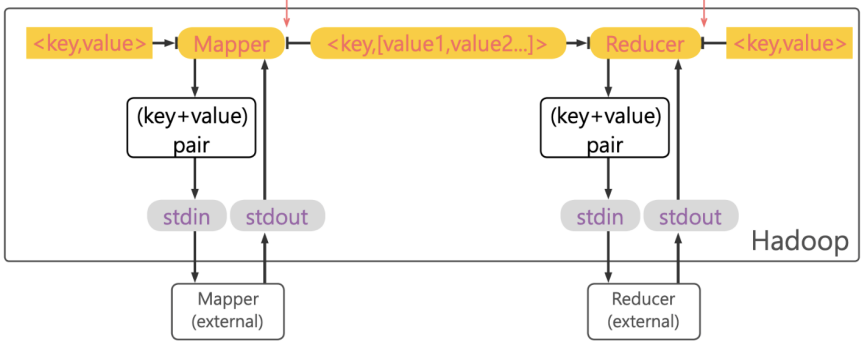
Comptession format	Hadoop CompressionCodec
DEFLATE	org.apache.hadoop.io.compress.DefaultCodec
gzip	org.apache.hadoop.io.compress.GzipCodec
bzip2	org.apache.hadoop.io.compress.BZip2Codec
LZO	com.hadoop.compression.lzo.LzopCodec
LZ4	org.apache.hadoop.io.compress.Lz4Codec
Snappy	org.apache.hadoop.io.compress.SnappyCodec

Kinds of compression

- Block-level compression
  - used in SequenceFiles, RCFiles, Parquet
  - applied within a block of data
- File-level compression
  - applied to the file as a whole
  - hinders an ability to navigate through file

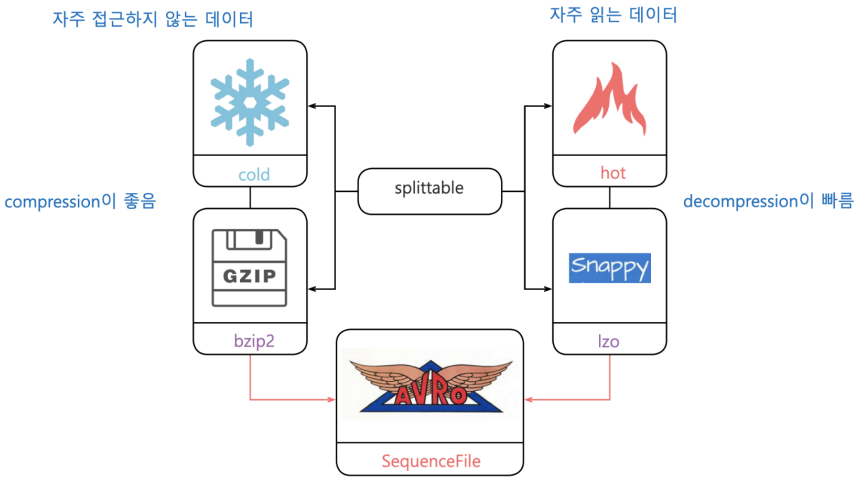
-D mapreduce.compress.map.output=true
-D mapreduce.map.output.compression.codec=...

-D mapreduce.output.compress=true
-D mapreduce.output.compression.codec=...



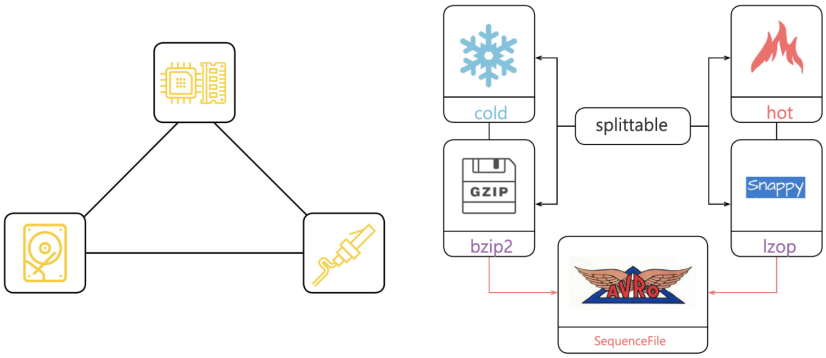
intermediate data 또는 output data 둘 다 적용 가능

Summary



gzip과 snappy는 file-level compression이 불가능 / block-level compression만 가능하다. (splittable 문제)

›› you know where and how to use compression to
››optimise MapReduce Application



›› you know how to use speculative execution to speed-up MapReduce application

›› see more configuration options in mapred-default.xml

출처: https://github.com/yahwang/Learn-Big-Data-Essentials-Yandex/blob/master/Week2/Week2_5_MapReduce_Optimization.pdf

Hadoop Streaming Final (Quiz)