Hadoop MapReduce và chương trình WordCount cơ bản với MapReduce

trannguyenhan on Aug 3, 2021

Updated Jul 16, 2022 • 10 min read

MapReduce là một kỹ thuật xử lý và là một mô hình lập trình cho tính toán phân tán để triển khai và xử lý dữ liệu lớn. Hadoop MapReduce là một khung xử lý dữ liệu của Hadoop xây dựng dựa trên ý tưởng của MapReduce, bây giờ khi nói về MapReduce là chúng ta sẽ nghĩ ngay tới Hadoop MapReduce, nên trong bài viết này một số chỗ mình xin phép nói ngắn gọn Hadoop MapReduce là MapReduce.

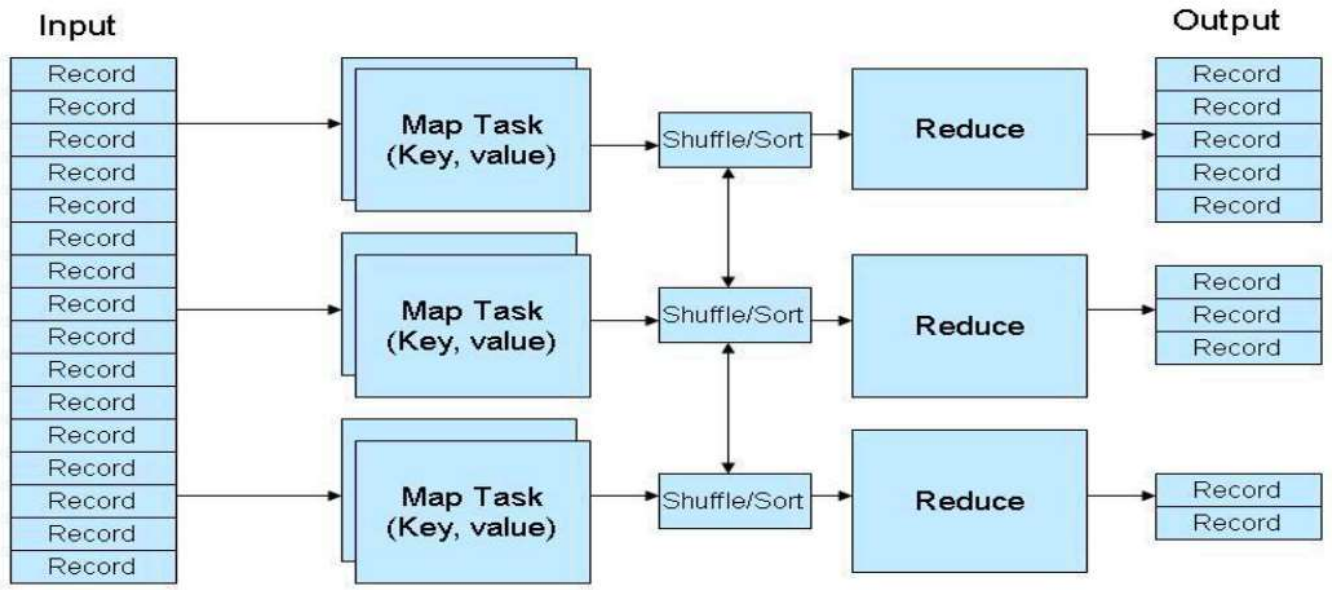
Để hiểu rõ hơn về ý tưởng của MapReduce các bạn có thể xem lại bài viết trước của mình về Mô hình lập trình MapReduce cho Bigdata.

Các thành phần Hadoop MapReduce

Khi lập trình với MapReduce các bạn chỉ cần phải để ý tới 3 lớp sau:

1. Mapper
2. Shuffle and sorting
3. Reducer

Tổng quan thực thi



- Máy master phân phối M `Map` task cho các máy vào theo dõi tiến trình của chúng
- Các `Map` task đọc dữ liệu và tiến hành xử lý, kết quả được lưu trữ tại local buffer
- Pha Shuffle chỉ định các *reducers* tới các vùng nhớ đệm nơi mà chúng được đọc từ xa và xử lý bởi các *reducers*
- `Reducer` xuất kết quả và lưu trữ trên HDFS

Mapper

Đây là pha đầu tiên của chương trình. Có hai bước trong pha này: splitting and mapping. Một tập dữ liệu được chia thành các đơn vị bằng nhau được gọi là chunks trong bước phân tách (splitting). Hadoop bao gồm một RecordReader sử dụng TextInputFormat để chuyển đổi các phân tách đầu vào thành các cặp key-value.

Shuffle

Đây là pha thứ hai diễn ra sau khi hoàn thành Mapper. Nó bao gồm hai bước chính: sắp xếp và hợp nhất (sorting and merging). Trong pha này, các cặp key-value được sắp xếp bằng cách sử dụng các key. Việc hợp nhất đảm bảo rằng các cặp key-value được kết hợp.

Reduce

Trong pha Reduce, đầu ra của pha Shuffle được sử dụng làm đầu vào. Reducer xử lý đầu vào này hơn nữa để giảm các giá trị trung gian thành các giá trị nhỏ hơn. Nó cung cấp một bản tóm tắt của toàn bộ tập dữ liệu, ví dụ như là tính tổng, tìm max, min,… . Đầu ra của pha này được lưu trữ trong HDFS.

Ưu điểm của Hadoop MapReduce

Hỗ trợ xử lý và tính toán song song (Parallel Processing)

Trong MapReduce, công việc được phân chia giữa nhiều node và mỗi node hoạt động đồng thời với một phần công việc. Mô hình MapReduce cho phép công việc được phân chia ra thành các công việc nhỏ hơn hoàn toàn riêng biệt.

Data Locality

Mặc dù các máy đã kết hợp với nhau thành một cụm, tuy nhiên khi dữ liệu càng lớn lên thì việc di chuyển dữ liệu giữa các máy là rất mất thời gian và có thể gây ra các vấn đề như tắc nghẽn đường truyền.

Hadoop khắc phục vấn đề trên bằng cách phân phối dữ liệu ở nhiều node và mỗi node xử lý các phần dữ liệu nằm trên chính nó.

Khả năng mở rộng (Scalability)

Hadoop có khả năng mở rộng cao, có thể lên tới hàng nghìn node mà không ảnh hưởng tới hiệu năng cũng như phát sinh lỗi.

Ví dụ để scan 1000TB dữ liệu trên 1 node với tốc độ 100MB/s thì sẽ mất 24 ngày, và khi mở rộng cụm lên 1000 node chúng ta cũng mất tương đương 35 phút để scan xong 1000TB dữ liệu này (hiệu năng hoàn toàn không bị giảm sút và không có phát sinh lỗi trong quá trình mở rộng)

Tính sẵn có và khả năng chịu lỗi (Availability & Fault Tolerance)

Hadoop lưu trữ các bản sao của dữ liệu trên các node khác nhau, vì thế trong trường hợp bị lỗi bản sao dữ liệu luôn sẵn sàng sử dụng bất cứ khi nào được yêu cầu để đảm bảo tính sẵn có của dữ liệu.

Nhờ tính năng sẵn có của dữ liệu mà Hadoop có khả năng chịu lỗi cao, khi một Task bị kill hay là một node bị mất kết nối dẫn tới Task đó không được hoàn thành thì Hadoop sẽ nhanh chóng phát hiện và chỉ định một node mới có chứa bản sao dữ liệu thực hiện Task đó (đảm bảo tính *locality*)

Chi phí thấp (Cost-effective)

Như đã được đề cập tới trong bài Giới thiệu tổng quan về Hadoop, Hadoop chạy trên các máy có phần cứng phổ thông (commodity hardware), là các máy rẻ, băng thông không cao. Hadoop có khả năng chịu lỗi cao vì vậy cần ít các quản trị viên hơn. Hadoop là dễ học, dễ sử dụng nên cũng tốn ít chi phí trong việc đào tạo cũng như thuê nhân công.

Bảo mật và xác thực (Security & Authentication)

Mô hình lập trình MapReduce giải quyết rủi ro về bảo mật bằng cách làm việc với HDFS và HBase có tính mật cao chỉ cho phép người dùng được phê duyệt mới có thể thao tác trên dữ liệu được lưu trữ trong hệ thống.

Mô hình lập trình đơn giản

Các bạn có thể thấy mô hình lập trình MapReduce là cực kì đơn giản, ngoài ra thì Hadoop MapReduce sử dụng ngôn ngữ Java là một ngôn ngữ phổ biến và dễ học.

Chương trình WordCount với MapReduce

Chương trình WordCount là chương trình kinh điển minh họa cho MapReduce và được lấy làm ví dụ cho hầu hết các bài giới thiệu về MapReduce.

Trong phần này, mình sẽ hướng dẫn mọi người chạy Job này với Hadoop MapReduce. Toàn bộ mã nguồn của chương trình WordCount bạn có thể tải về TẠI ĐÂY.

Bước 1: Tải về Project

Các bạn hãy clone project của mình đã được chuẩn bị sẵn về máy TẠI ĐÂY.

Bước 2: Cài đặt mvn

Nếu máy bạn chưa có `mvn`, hãy tải về `mvn` nha, việc cài đặt rất đơn giản nên mình sẽ không đề cập ở đây.

Để kiểm tra xem đã cài đặt thành công `mvn` thì bạn sử dụng câu lệnh `mvn --version`:

Bước 3: Build file jar

Di chuyển tới thư mục chứa project vừa clone về, chạy lệnh `mvn clean package` để build project thành một file `jar` (khi build lần đầu, mvn sẽ phải tải xuống các phụ thuộc nên sẽ lâu, các lần sau sẽ nhanh hơn).

Để ý nếu bạn gặp lỗi như dưới đây nếu build thì hãy sửa lại cho đúng phiên bản jdk mà bạn đang dùng trên máy, ví dụ trong file `pom` được cấu hình là jdk 15, tuy nhiên trên máy mình chỉ có jdk 11 thì sau khi build nó sẽ hiện lỗi:

Tại file `pom.xml` trong thẻ `<plugins>` sửa lại `<release>` là 11, sau đó lưu lại và thực hiện lại câu lệnh `mvn clean package`:

Nếu hiện dòng chữ “BUILD SUCCESS” là các bạn đã build thành công file `jar` rồi nha, file `jar` sinh ra nằm trong thư mục `target` của project. Các bạn có thể thấy là lần build lại thứ 2 này rất nhanh do các phụ thuộc đã được tải về để trên máy local.

Bước 4: Chuẩn bị dữ liệu

Sau khi có file `jar` tiếp theo là chúng ta sẽ chuẩn bị dữ liệu đầu vào, các bạn có thể lấy 1 file text bất kì để test, bạn có thể chạy Hadoop với input đầu vào nằm ở local, tuy nhiên điều này chỉ thành công nếu bạn chạy single node còn nếu bạn có 1 cụm Hadoop và muốn chạy job trên đó thì file đầu vào của bạn phải được đẩy lên HDFS.

Copy file của bạn từ local lên HDFS thông qua dòng lệnh:

```bash
hdfs dfs -copyFromLocal input.txt /
```

Xem lại các câu lệnh thao tác với file và thư mục trên HDFS TẠI ĐÂY

Kiểm tra xem file `input.txt` của bạn đã có trên HDFS chưa bằng lệnh:

```bash
hdfs dfs -ls /
```

Bước 4: Chạy chương trình

Mọi thứ đã xong bây giờ là submit job của bạn lên Hadoop và chờ đợi kết quả, chạy dòng lệnh sau:

```bash
hadoop jar target/wordcount-V1.jar com.hadoop.mapreduce.WordCount hdfs://localhost:9000/input.txt
hdfs://localhost:9000/output
```

- Với tham số đầu tiên `hdfs://localhost:9000/input.txt` là đường dẫn tới file input đầu vào, do file input được đặt trên HDFS nên phải thêm vào đường dẫn `hdfs://` nếu không chương trình sẽ hiểu đó là một đường dẫn local
- Tham số thứ 2 `hdfs://localhost:9001/output` là đường dẫn đặt output của chương trình, chúng ta cũng sẽ lưu output trên HDFS
- Nếu bạn không cài HDFS tại cổng `9000` hãy sửa lại cho đúng
- Nếu bạn submit job lên cụm nhiều node hãy thay đổi `localhost` thành tên tương ứng với máy master

Sau khi chạy xong terminal sẽ hiện một số thông tin về job đã chạy của bạn:

```
        File System Counters
                FILE: Number of bytes read=517
                FILE: Number of bytes written=469865
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=418
                HDFS: Number of bytes written=339
                HDFS: Number of read operations=8
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
                HDFS: Number of bytes read erasure-coded=0
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=1840
                Total time spent by all reduces in occupied slots (ms)=1779
                Total time spent by all map tasks (ms)=1840
                Total time spent by all reduce tasks (ms)=1779
                Total vcore-milliseconds taken by all map tasks=1840
                Total vcore-milliseconds taken by all reduce tasks=1779
                Total megabyte-milliseconds taken by all map tasks=1884160
                Total megabyte-milliseconds taken by all reduce tasks=1821696
        Map-Reduce Framework
                Map input records=4
                Map output records=58
                Map output bytes=554
                Map output materialized bytes=517
                Input split bytes=96
                Combine input records=58
                Combine output records=43
                Reduce input groups=43
                Reduce shuffle bytes=517
                Reduce input records=43
                Reduce output records=43
                Spilled Records=86
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=21
                CPU time spent (ms)=950
                Physical memory (bytes) snapshot=517545984
                Virtual memory (bytes) snapshot=5479784448
                Total committed heap usage (bytes)=520093696
                Peak Map Physical memory (bytes)=302477312
                Peak Map Virtual memory (bytes)=2736652288
                Peak Reduce Physical memory (bytes)=215068672
                Peak Reduce Virtual memory (bytes)=2743132160
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
```

Ví dụ trong job của mình có 1 `Map` task và 1 `Reduce` task, File input đầu vào là 332 bytes và file output là 330 bytes.

Kiểm tra file output đầu ra trên HDFS thông qua lệnh `hdfs dfs -cat /output/part-r-00000`:

```
                Bytes Written=339
trannguyenhan@PC0628:~/CodeFolder/java-workspace/word-count-hadoop$ hdfs dfs -cat /output/part-r-00000
Hadoop  1
The     2
This    2
above   1
all     1
alphabets.      1
also    1
article 1
as      1
brown   1
code    1
contains        1
count   1
dog.    1
ecosystem.      1
english 1
example 4
examples        1
famous  1
file    1
for     2
fox     1
geek    1
hello   1
is      3
java    1
jumps   1
knows   1
language        1
lazy    1
line    1
lines   1
most    1
of      3
one     1
over    1
quick   1
text    1
the     6
which   1
word    1
world   1
written 1
trannguyenhan@PC0628:~/CodeFolder/java-workspace/word-count-hadoop$
```

Chạy chương trình với 2 hoặc nhiều Mapper

Chúng ta không thể set được số Mapper chạy trong 1 job mà chỉ có thể set được số Mapper tối đa, Hadoop sẽ tự chỉ định số Mapper dựa vào số blocks.

Để Hadoop chạy với 2 Mapper thì dữ liệu đầu vào phải được HDFS lưu trên 2 blocks vậy thì có 2 cách:

- Một là file input đầu vào của bạn nặng hơn block size khi đó HDFS sẽ lưu trữ file đó dưới dạng 2 blocks.
- Hai là input đầu vào của bạn thay bằng 1 folder và tron folder đó bạn đặt vào 2 file text bất kì, Hadoop cho phép đầu vào dữ liệu là 1 folder và nó sẽ tự động quét tất cả các file có trong thư mục đó.

Bây giờ mình sẽ tạo ra 1 bản sao của file `input.txt` vừa rồi và đặt tên là `input-1.txt`:

Sau đó tạo ra 1 thư mục `input` trên HDFS:

```bash
hdfs dfs -mkdir /input
```

Di chuyển lần lượt từng file `input.txt` và `input-1.txt` vào trong thư mục `input` :

```bash
hdfs dfs -mv /input.txt /input
hdfs dfs -mv /input-1.txt /input
```

Kiểm tra xem dữ liệu đã nằm trong thư mục `input` hay chưa:

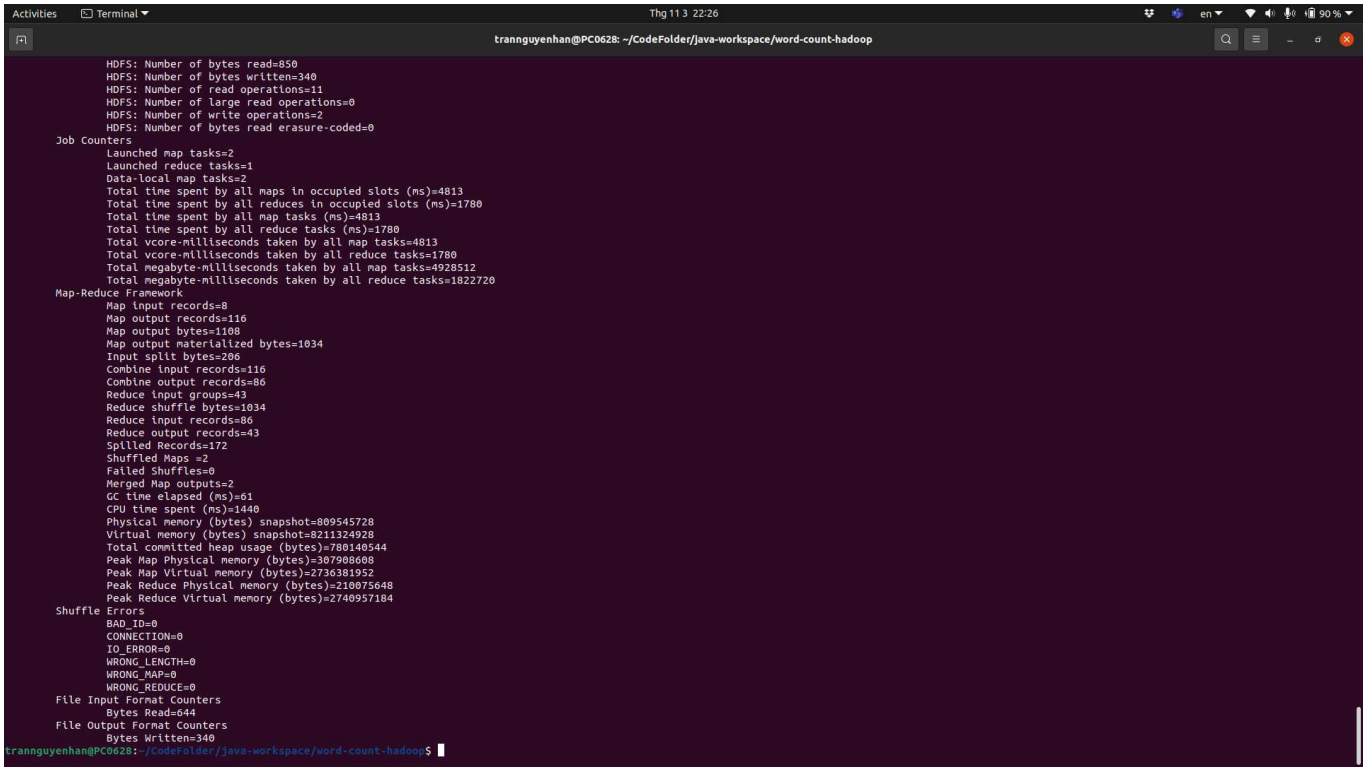
```bash
hdfs dfs -ls /input
```

Vậy là đã xong dữ liệu, bây giờ chúng ta run lại chương trình:

```bash
hadoop jar target/wordcount-V1.jar com.hadoop.mapreduce.WordCount hdfs://localhost:9000/input hdfs://localhost:9000/output
```

- Lúc này input của chúng ta không phải là file `input.txt` nữa mà là cả thư mục `input` nên hãy đổi lại tham số input

Sau khi chương trình kết thúc chúng ta có thể thấy ngay các thông tin về chương trình đã thay đôi như số `Map` task đã chạy là 2:



Kiểm tra thông tin job đã chạy tại cổng UI của YARN

Để sinh động hơn, các bạn có thể kiểm tra thông tin 1 job đã chạy tại cổng UI của YARN là cổng 8088:

- Nếu các bạn chạy job trên một cụm nhiều node, hãy thay `localhost` tên tương ứng với máy master

    - Trên đây hiện thông tin về 2 job WordCount mà mình vừa chạy
    - Bấm vào từng task một để xem thông tin chi tiết về task đó

Tham khảo: https://hadoop.apache.org, https://www.educba.com/, https://www.edureka.co/

Hadoop & Spark , Hadoop

Hadoop Apache Hadoop Bigdata HDFS Hadoop MapReduce MapReduce

Share:

Further Reading

Jun 29, 2021

Giới thiệu tổng quan Hadoop

Hadoop là framework dựa trên 1 giải pháp tới từ Google để lưu trữ và xử lý dữ liệu lớn,…

Jul 1, 2021

Cài đặt và triển khai Hadoop single node

Mỗi ngành công nghiệp lớn đang triển khai Apache Hadoop là khung tiêu chuẩn để xử…

Jul 4, 2021

HDFS

Hadoop Distributed File System (HDFS) là hệ thống lưu trữ phân tán được thiết kế để ch…

OLDER
Docker cơ bản và thực hành

NEWER
Tổng hợp các câu hỏi về Apache Hadoop