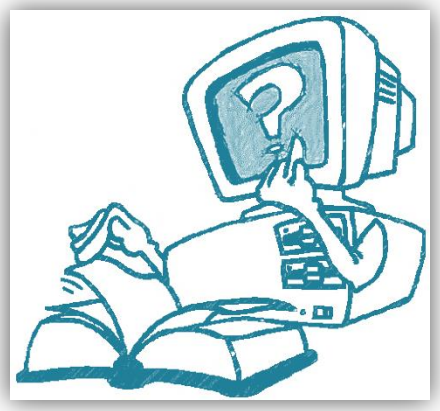
BÁO CÁO THỰC TẬP



Tìm hiểu Hadoop, MapReduce,

và các bài toán ứng dụng

Giáo viên hướng dẫn: Từ Minh Phương

Sinh viên: Vũ Minh Ngọc

Mục lục

Lời nói đầu

Kính chào các thầy cô!

Sau một thời gian thực tập tốt nghiệp, sau đây là bản báo cáo những gì em đã làm được trong thời gian qua. Nội dung chính trong thời gian thực tập vừa qua là Sử dụng Hadoop và framework MapReduce để giải quyết bài toán tinh sinh học BLAST. Theo cảm nghĩ của em thì Hadoop là một ứng dụng mới và cũng không dễ để nắm bắt, và việc làm sao để thuật toán BLAST có thể xử lý song song trên Hadoop cũng khá khó. Nhưng với sự giúp đỡ của thầy hướng dẫn Từ Minh Phương, và các anh chị trong công ti VCCorp thì em cũng phần nào nắm bắt được vấn đề.

Tuy bản báo cáo còn sơ sài, nhưng là tiền đề cho những phần kế tiếp. Em sẽ cố gắng hoàn thiện hơn, và hoàn chỉnh đề tài vào bài cuối khoá.

Một lần nữa em xin cám ơn các thầy cô đã định hướng và hướng dẫn trong suốt thời gian học tập và trong thời gian thực tập vừa qua.

Phần I. Giới thiệu chung

1.1. Hadoop là gì?

Mục đích : Mong muốn của các doanh nghiệp là tận dụng lượng dữ liệu khổng lồ để đưa ra quyết định kinh doanh, Hadoop giúp các công ty xử lý khối lượng cỡ terabyte và thậm chí là petabytes dữ liệu phức tạp tương đối hiệu quả với chi phí thấp hơn.

Các doanh nghiệp đang nỗ lực tìm kiếm thông tin quý giá từ khối lượng lớn dữ liệu phi cấu trúc được tạo ra bởi các web log, công cụ clickstream, các sản phẩm truyền thông xã hội. Chính yếu tố đó dẫn làm tăng sự quan tâm đến công nghệ mã nguồn mở Hadoop.

Hadoop, một dự án phần mềm quản lý dữ liệu Apache với nhân trong khung phần mềm MapReduce của Google, được thiết kế để hỗ trợ các ứng dụng sử dụng được số lượng lớn dữ liệu cấu trúc và phi cấu trúc.

Không giống như các hệ quản trị cơ sở dữ liệu truyền thống, Hadoop được thiết kế để làm việc với nhiều loại dữ liệu và dữ liệu nguồn. Công nghệ HDFS của Hadoop cho phép khối lượng lớn công việc được chia thành các khối dữ liệu nhỏ hơn được nhân rộng và phân phối trên các phần cứng của một cluster để xử lý nhanh hơn. Công nghệ này đã được sử dụng rộng rãi bởi một số trang web lớn nhất thế giới, chẳng hạn như Facebook, eBay, Amazon, Baidu, và Yahoo. Các nhà quan sát nhấn mạnh rằng Yahoo là một trong những nhà đóng góp lớn nhất đối với Hadoop.

1.2. MapReduce là gì?

MapReduce là một "mô hình lập trình" (programming model), lần đầu báo cáo trong bài báo của Jefferey Dean và Sanjay Ghemawat ở hội nghị OSDI 2004. MapReduce chỉ là một ý tưởng, một abstraction. Để hiện thực nó thì cần một implementation cụ thể. Google có một implementation của MapReduce bằng C++. Apache có Hadoop, một implementation mã nguồn mở khác trên Java thì phải (ít nhất người dùng dùng Hadoop qua một Java interface).

Khối dữ liệu lớn được tổ chức như một tập hợp gồm rất nhiều cặp (key, value) Để xử lý khối dữ liệu này, lập trình viên viết hai hàm map và reduce. Hàm map có input là một cặp **(k1, v1)** và output là một danh sách các cặp **(k2, v2)**. Chú ý rằng các input và output keys và values có thể thuộc về các kiểu dữ liệu khác nhau, tùy hỉ. Như vập hàm map có thể được viết một cách hình thức như sau:

map(k1,v1) -> list(k2,v2)

MR sẽ áp dụng hàm **map** (mà người dùng MR viết) vào từng cặp (key, value) trong khối dữ liệu vào, chạy rất nhiều phiên bản của map song song với nhau trên các máy tính của cluster. Sau giai đoạn này thì chúng ta có một tập hợp rất nhiều cặp (key, value) thuộc kiểu **(k2, v2)** gọi là các cặp (key, value) trung gian. MR cũng sẽ nhóm các cặp này theo từng key, như vậy các cặp (key, value) trung gian có cùng **k2** sẽ nằm cùng một nhóm trung gian.

Giai đoạn hai MR sẽ áp dụng hàm **reduce** (mà người dùng MR viết) vào từng nhóm trung gian. Một cách hình thức, hàm này có thể mô tả như sau:

reduce(k2, list (v2)) -> list(v3)

Trong đó **k2** là key chung của nhóm trung gian, **list(v2)** là tập các values trong nhóm, và list(v3) là một danh sách các giá trị trả về của **reduce** thuộc kiểu dữ liệu **v3**. Do **reduce** được áp dụng vào nhiều nhóm trung gian độc lập nhau, chúng lại một lần nữa có thể được chạy song song với nhau.

Ví dụ cơ bản nhất của MR là bài đếm từ (Tiếng Anh). Rõ ràng đây là một bài toán cơ bản và quan trọng mà một search engine phải làm. Nếu chỉ có vài chục files thì dễ rồi, nhưng nhớ rằng ta có nhiều triệu hay thậm chí nhiều tỉ files phân bố trong một cluster nhiều nghìn máy tính. Ta lập trình MR bằng cách viết 2 hàm cơ bản với pseudo-code như sau:

```
void map(String name, String document):
// name: document name
// document: document contents
for each word w in document:
  EmitIntermediate(w, "1");

void reduce(String word, Iterator partialCounts):
// word: a word
// partialCounts: a list of aggregated partial counts
int result = 0;
for each pc in partialCounts:
  result += ParseInt(pc);
Emit(AsString(result));
```

Chỉ với hai primitives này, lập trình viên có rất nhiều flexibility để phân tích và xử lý các khối dữ liệu khổng lồ. MR đã được dùng để làm rất nhiều việc khác nhau, ví dụ như distributed grep, distributed sort, web link-graph reversal, term-vector per host, web access log stats, inverted index construction, document clustering, machine learning, statistical machine translation, large-scale graph computation …

Phần II. Cài đặt Hadoop

1. Cài đặt máy ảo Ubuntu 10.10 (32 bit) trên VMware

- Sử dụng VMware® Workstation 7.0.0 build-203739 (32-bit)
- Hệ điều hành Ubuntu Desktop Edittion 10.10 (32-bit)
- Tạo user mặc định là hadoop

1. Cài đặt Vmware tools cho Ubuntu

a. Kích hoạt tài khoản root

```
$sudo passwd root
```

- bạn điền pass cho tài khoản hadoop
- điền tiếp 2 lần pass mới cho tài khoản root

b. Cài đặt tools cho Ubuntu

- Đăng nhập lại bằng tài khoản root
- Chọn cài Vmware tools như hình sau
- Vào máy ảo Ubuntu, giải nén file VMwareTools-8.1.3-203739.tar.gz và chạy file vmware-install.pl
- Bấm enter để chọn các tùy chọn mặc định đặt trong dấu móc vuông

2. Cài openSSH cho ubuntu

```
$ sudo apt-get install openssh-server openssh-client
```

3. Cài java:

Hadoop yêu cầu java 1.5.x. Tuy nhiên, bản 1.6.x được khuyến khích khi sử dụng cho Hadoop, dưới đây mô tả cách thức cài java :

a. Thêm Canonical Đối tác Repository vào kho apt của bạn

```
$ sudo add-apt-repository "deb http://archive.canonical.com/
lucid partner"
```

b. Cập nhật danh sách nguồn

```
$ sudo apt-get update
```

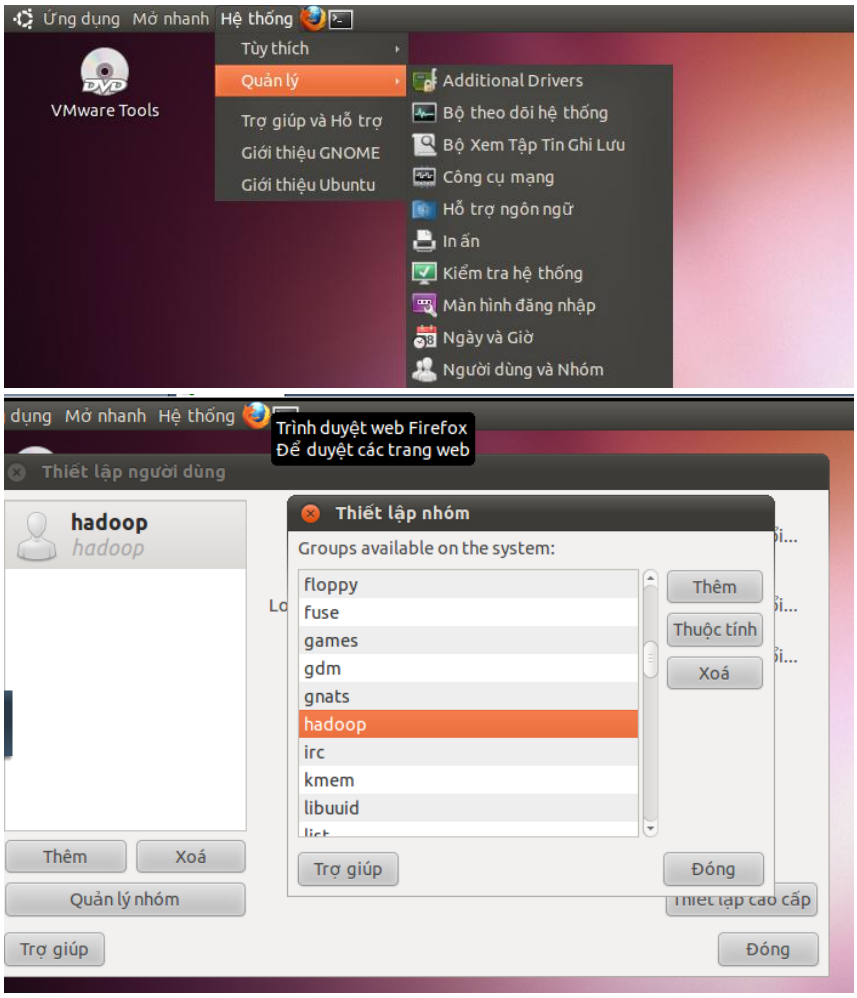
c. Cài đặt sun-java6-jdk

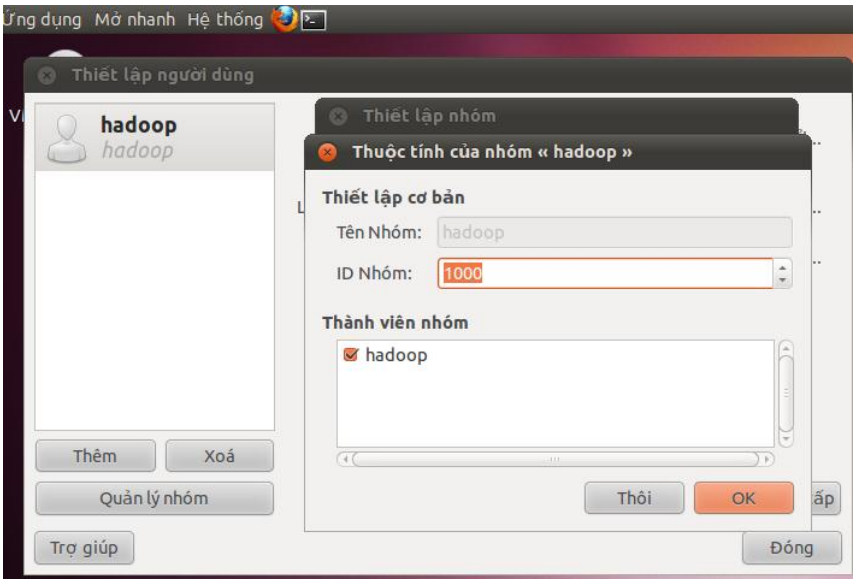
```
$ sudo apt-get install sun-java6-jdk
```

d. Kiểm tra

```
user@ubuntu:~# java -version
      java version "1.6.0_20"
      Java(TM) SE Runtime Environment (build 1.6.0_20-b02)
      Java HotSpot(TM) Client VM (build 16.3-b01, mixed mode,
sharing)
```

4. Thêm user hadoop vào nhóm hadoop





5. Cấu hình ssh

Hadoop yêu cầu truy cập SSH để quản lý các node của nó, ví dụ như điều kiển một máy tính từ xa cộng với máy cục bộ của bạn nếu như bạn muốn Hadoop làm việc trên đó. Trong thiết lập đơn node cho haddop , chúng ta cấu hình ssh truy cập tới localhost cho user hadoop mà chúng ta tạo ra ở phần trước.

a. Đăng nhập từ tài khoản hadoop
b. Sử dụng dòng lệnh
   // Không nhập gì trong 3 lần hỏi, chỉ ấn xuống dòng Enter

```
hadoop@hadoop:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hadoop/.ssh/id_rsa):
Created directory '/home/hadoop/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hadoop/.ssh/id_rsa.
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub.
The key fingerprint is:
19:f5:c2:b2:19:25:83:25:8f:ec:45:f7:4a:c3:59:25 hadoop@hadoop
The key's randomart image is:
+--[ RSA 2048]----+
|      .o= + E..  |
|     ..= O = .   |
|      o * B o    |
|     . . O +     |
|      . S .      |
|                 |
|                 |
|                 |
|                 |
+-----------------+
```

c. Bạn phải cho phép SSH truy cập tới máy cục bộ của bạn với khóa mới:

```
hadoop@hadoop:~$ cd ~/.ssh
hadoop@hadoop:~/.ssh$ cat id_rsa.pub >> authorized_keys
```

d. Kiểm tra các cài đặt SSH bằng cách kết nối với máy tính cục bộ của bạn với user hadoop. Bước này cũng cần thiết để lưu trữ dấu vân tay của máy bạn trong file know_host. Nếu bạn có bất cứ cấu hình đặc biệt cho SSH giống như một cổng SSH không chuẩn, bạn có thể định nghĩa lại trong $HOME/.ssh/config

```
hadoop@hadoop:~/.ssh$ ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is 0a:3d:86:06:28:82:7f:3a:35:0b:83:d5:35:ee:b8:b1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'localhost' (RSA) to the list of known hosts.
Linux hadoop 2.6.35-22-generic #33-Ubuntu SMP Sun Sep 19 20:34:50 UTC
2010 i686 GNU/Linux
Ubuntu 10.10

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
```

6. Vô hiệu hóa IPv6

Một vấn đề với IPv6 trên Ubuntu là việc sử dụng 0.0.0.0 cho các tùy chọn cấu hình Hadoop cho các mạng có liên quan đến nhau sẽ cho kết quả Hadoop liên kết đến các địa chỉ IPv6 của my Ubuntu box.

a. Để vô hiệu hóa IPv6 trong Ubuntu 10.10, mở /etc/sysctl.conf trong editor bạn thêm dòng sau vào cuối file:

```
#disable ipv6
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
```

b. Khởi động lại máy để thay đổi có hiệu quả.

c. Để kiểm tra lại bạn có thể sử dụng dòng lệnh sau

```
$ cd /
$ cat /proc/sys/net/ipv6/conf/all/disable_ipv6
```

Kết quả trả về là 0 tức là IPv6 vẫn còn được kích hoạt, bằng 1 là đã được vô hiệu hóa.

7. Download và cài đặt hadoop

a. Download Hadoop 0.20.2 và lưu vào thư mục /usr/local/

```
$ cd /usr/local
$ sudo tar xzf hadoop-0.20.2.tar.gz
$ sudo mv hadoop-0.20.2 hadoop
$ sudo chown -R hadoop:hadoop hadoop
```

b. Cấu hình

i. hadoop-env.sh

Cài đặt JAVA_HOME. Thay đổi

```
# The java implementation to use.  Required.
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun
```

Thành :

```
# The java implementation to use.  Required.
export JAVA_HOME=/usr/lib/jvm/java-6-sun
```

ii. conf/core-site.xml

```
<!-- In: conf/core-site.xml -->
<property>
 <name>hadoop.tmp.dir</name>
 <value>/your/path/to/hadoop/tmp/dir/hadoop-${user.name}</value>
 <description>A base for other temporary directories.</description>
</property>

<property>
 <name>fs.default.name</name>
 <value>hdfs://localhost:54310</value>
 <description>The name of the default file system.  A URI whose
 scheme and authority determine the FileSystem implementation.  The
 uri's scheme determines the config property (fs.SCHEME.impl) naming
 the FileSystem implementation class.  The uri's authority is used to
 determine the host, port, etc. for a filesystem.</description>
</property>
```

iii. conf/mapred-site.xml

```
<!-- In: conf/mapred-site.xml -->
<property>
 <name>mapred.job.tracker</name>
 <value>localhost:54311</value>
 <description>The host and port that the MapReduce job tracker runs
 at.  If "local", then jobs are run in-process as a single map
 and reduce task.
```

```
 </description>
</property>
```

iv. conf/hdfs-site.xml

```
<!-- In: conf/hdfs-site.xml -->
<property>
 <name>dfs.replication</name>
 <value>1</value>
 <description>Default block replication.
 The actual number of replications can be specified when the file is created.
 The default is used if replication is not specified in create time.
 </description>
</property>
```

c. Định dạng các tên node

Đầu tiên để khởi động Hadoop vừa của bạn là định dạng lại hệ thống tệp tin Hadoop mà được thực hiện trên đầu của hệ thống tệp tin của bạn. Bạn cần phải làm việc này trong lần đầu chạy. Bạn chạy lệnh sau:

```
hadoop@ubuntu:~$ /hadoop/bin/hadoop namenode -format
```

Kết quả:

```
01 hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop namenode -format
02 10/05/08 16:59:56 INFO namenode.NameNode: STARTUP_MSG:
03 /************************************************************
04 STARTUP_MSG: Starting NameNode
05 STARTUP_MSG:   host = ubuntu/127.0.1.1
06 STARTUP_MSG:   args = [-format]
07 STARTUP_MSG:   version = 0.20.2
08 STARTUP_MSG:   build = https://svn.apache.org/repos/asf/hadoop/common/branches/branch-0.20 -r
   911707; compiled by 'chrisdo' on Fri Feb 19 08:07:34 UTC 2010
09 ************************************************************/
10 10/05/08 16:59:56 INFO namenode.FSNamesystem: fsOwner=hadoop,hadoop
11 10/05/08 16:59:56 INFO namenode.FSNamesystem: supergroup=supergroup
12 10/05/08 16:59:56 INFO namenode.FSNamesystem: isPermissionEnabled=true
13 10/05/08 16:59:56 INFO common.Storage: Image file of size 96 saved in 0 seconds.
14 10/05/08 16:59:57 INFO common.Storage: Storage directory .../hadoop-hadoop/dfs/name has been
   successfully formatted.
15 10/05/08 16:59:57 INFO namenode.NameNode: SHUTDOWN_MSG:
16 /************************************************************
17 SHUTDOWN_MSG: Shutting down NameNode at ubuntu/127.0.1.1
18 ************************************************************/
19 hadoop@ubuntu:/usr/local/hadoop$
```

d. Chạy hadoop trên cụm một node

Sử dụng câu lệnh : $ /bin/start-all.sh

Kết quả như sau:

```
hadoop@ubuntu:/usr/local/hadoop$ bin/start-all.sh
starting namenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-namenode-ubuntu.out
localhost: starting datanode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-datanode-ubuntu.out
localhost: starting secondarynamenode, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-secondarynamenode-ubuntu.out
starting jobtracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-jobtracker-ubuntu.out
localhost: starting tasktracker, logging to /usr/local/hadoop/bin/../logs/hadoop-hadoop-tasktracker-ubuntu.out
hadoop@ubuntu:/usr/local/hadoop$
```

Một tool khá thuật tiện để kiểm tra xem các tiến trình Hadoop đang chạy là jps:

```
hadoop@ubuntu:/usr/local/hadoop$ jps
2287 TaskTracker
2149 JobTracker
1938 DataNode
2085 SecondaryNameNode
2349 Jps
1788 NameNode
```

Bạn cũng có thể kiểm tra với *netstart* nếu Hadoop đang nghe trên các cổng đã được cấu hình:

```
hadoop@ubuntu:~$ sudo netstat -plten | grep java
tcp   0  0 0.0.0.0:50070   0.0.0.0:*  LISTEN  1001  9236  2471/java
tcp   0  0 0.0.0.0:50010   0.0.0.0:*  LISTEN  1001  9998  2628/java
tcp   0  0 0.0.0.0:48159   0.0.0.0:*  LISTEN  1001  8496  2628/java
tcp   0  0 0.0.0.0:53121   0.0.0.0:*  LISTEN  1001  9228  2857/java
tcp   0  0 127.0.0.1:54310 0.0.0.0:*  LISTEN  1001  8143  2471/java
tcp   0  0 127.0.0.1:54311 0.0.0.0:*  LISTEN  1001  9230  2857/java
tcp   0  0 0.0.0.0:59305   0.0.0.0:*  LISTEN  1001  8141  2471/java
tcp   0  0 0.0.0.0:50060   0.0.0.0:*  LISTEN  1001  9857  3005/java
tcp   0  0 0.0.0.0:49900   0.0.0.0:*  LISTEN  1001  9037  2785/java
tcp   0  0 0.0.0.0:50030   0.0.0.0:*  LISTEN  1001  9773  2857/java
hadoop@ubuntu:~$
```

e. Dừng hadoop trên cụm một node

Sử dụng lệnh : /bin/stop-all.sh

8. Chạy một ví dụ MapReduce

Chúng ta chạy ví dụ WordCount có sẵn trong phần ví dụ của Hadoop. Nó xẽ đếm các từ trong file và số lần xuất hiện. file đầu vào và đầu ra đề là dạng text, mỗi dòng trong file đầu ra chứa từ và số lần xuất hiện, phân cách với nhau bởi dấu TAB.

a. Download dữ liệu đầu vào

Download 3 cuốn sách từ Project Gutenberg:

The Outline of Science, Vol. 1 (of 4) by J. Arthur Thomson

The Notebooks of Leonardo Da Vinci

Ulysses by James Joyce

Chọn file trong Plain Text UTF-8, sau đó copy vào thư mục tmp của Hadoop: /tmp/gutenberg , kiểm tra lại như sau:

```
hadoop@ubuntu:~$ ls -l /tmp/gutenberg/
total 3604
-rw-r--r-- 1 hadoop hadoop   674566 Feb  3 10:17 pg20417.txt
-rw-r--r-- 1 hadoop hadoop 1573112 Feb  3 10:18 pg4300.txt
-rw-r--r-- 1 hadoop hadoop 1423801 Feb  3 10:18 pg5000.txt
hadoop@ubuntu:~$
```

Restart lại hadoop cluster: hadoop@ubuntu:~$ /bin/start-all.sh

b. Copy dữ liệu vào HDFS

```
01    hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -copyFromLocal
      /tmp/gutenberg gutenberg
02    hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls
03    Found 1 items
04    drwxr-xr-x   - hadoop supergroup          0 2010-05-08 17:40
      /user/hadoop/gutenberg
05    hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls gutenberg
06    Found 3 items
07    -rw-r--r--   3 hadoop supergroup     674566 2011-03-10 11:38
      /user/hadoop/gutenberg/pg20417.txt
08    -rw-r--r--   3 hadoop supergroup    1573112 2011-03-10 11:38
      /user/hadoop/gutenberg/pg4300.txt
09    -rw-r--r--   3 hadoop supergroup    1423801 2011-03-10 11:38
      /user/hadoop/gutenberg/pg5000.txt
10    hadoop@ubuntu:/usr/local/hadoop$
```

c. Chạy MapReduce job

Sử dụng câu lệnh sau:

```
hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop jar hadoop-<version>-
examples.jar wordcount gutenberg gutenberg-output
```

Trong câu lệnh này bạn sửa <version> thành phiên bản mà bạn đang sử dụng. Bạn có thể kiểm tra trong thư mục cài Hadoop có chứa file *.jar này. Câu lệnh này sẽ đọc tất cả các file trong thư mục butenberg từ HDFS, xử lý và lưu kết quả vào gutenberg-output. Kết quả đầu ra như sau:

```
hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop jar hadoop-*-examples.jar wordcount gutenberg gutenberg-output
10/05/08 17:43:00 INFO input.FileInputFormat: Total input paths to process : 3
10/05/08 17:43:01 INFO mapred.JobClient: Running job: job_201005081732_0001
10/05/08 17:43:02 INFO mapred.JobClient:  map 0% reduce 0%
10/05/08 17:43:14 INFO mapred.JobClient:  map 66% reduce 0%
10/05/08 17:43:17 INFO mapred.JobClient:  map 100% reduce 0%
10/05/08 17:43:26 INFO mapred.JobClient:  map 100% reduce 100%
10/05/08 17:43:28 INFO mapred.JobClient: Job complete: job_201005081732_0001
10/05/08 17:43:28 INFO mapred.JobClient: Counters: 17
10/05/08 17:43:28 INFO mapred.JobClient:   Job Counters
10/05/08 17:43:28 INFO mapred.JobClient:     Launched reduce tasks=1
10/05/08 17:43:28 INFO mapred.JobClient:     Launched map tasks=3
10/05/08 17:43:28 INFO mapred.JobClient:     Data-local map tasks=3
10/05/08 17:43:28 INFO mapred.JobClient:   FileSystemCounters
10/05/08 17:43:28 INFO mapred.JobClient:     FILE_BYTES_READ=2214026
10/05/08 17:43:28 INFO mapred.JobClient:     HDFS_BYTES_READ=3639512
10/05/08 17:43:28 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=3687918
10/05/08 17:43:28 INFO mapred.JobClient:     HDFS_BYTES_WRITTEN=880330
10/05/08 17:43:28 INFO mapred.JobClient:   Map-Reduce Framework
10/05/08 17:43:28 INFO mapred.JobClient:     Reduce input groups=82290
10/05/08 17:43:28 INFO mapred.JobClient:     Combine output records=102286
10/05/08 17:43:28 INFO mapred.JobClient:     Map input records=77934
10/05/08 17:43:28 INFO mapred.JobClient:     Reduce shuffle bytes=1473796
10/05/08 17:43:28 INFO mapred.JobClient:     Reduce output records=82290
10/05/08 17:43:28 INFO mapred.JobClient:     Spilled Records=255874
10/05/08 17:43:28 INFO mapred.JobClient:     Map output bytes=6076267
10/05/08 17:43:28 INFO mapred.JobClient:     Combine input records=629187
10/05/08 17:43:28 INFO mapred.JobClient:     Map output records=629187
10/05/08 17:43:28 INFO mapred.JobClient:     Reduce input records=102286
```

Kiểm tra kết quả nếu lưu thành công:

```
hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls
Found 2 items
drwxr-xr-x   - hadoop supergroup          0 2010-05-08 17:40 /user/hadoop/gutenberg
drwxr-xr-x   - hadoop supergroup          0 2010-05-08 17:43 /user/hadoop/gutenberg-output
hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -ls gutenberg-output
Found 2 items
drwxr-xr-x   - hadoop supergroup          0 2010-05-08 17:43 /user/hadoop/gutenberg-output/_logs
-rw-r--r--   1 hadoop supergroup     880802 2010-05-08 17:43 /user/hadoop/gutenberg-output/part-r-00000
hadoop@ubuntu:/usr/local/hadoop$
```

Nếu bạn muốn sửa đổi các thiết lập của Hadoop giống như tăng số task Reduce lên, bạn có thể sử dụng tùy chọn "-D" như sau:

```
hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop jar hadoop-0.20.2-
examples.jar wordcount -D mapred.reduce.tasks=16 gutenberg
```

d. Lấy kết quả từ HDFS

Để kiểm tra các file, bạn có thể copy nó từ HDFS đến hệ thống file địa phương.

Ngoài ra, bạn có thể sử dụng lệnh sau:

```
hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -cat gutenberg-
output/part-r-00000gutenberg-output
```

Trong ví dụ này, chúng ta có thể copy như sau:

```
01 hadoop@ubuntu:/usr/local/hadoop$ mkdir /tmp/gutenberg-output
02 hadoop@ubuntu:/usr/local/hadoop$ bin/hadoop dfs -getmerge gutenberg-
   output /tmp/gutenberg-output
03 hadoop@ubuntu:/usr/local/hadoop$ head /tmp/gutenberg-output/gutenberg-
   output
04 "(Lo)cra"       1
05 "1490   1
06 "1498," 1
07 "35"    1
08 "40,"   1
09 "A      2
10 "AS-IS".        1
11 "A_     1
12 "Absoluti       1
13 "Alack! 1
14 hadoop@ubuntu:/usr/local/hadoop$
```

9. Cài đặt và sử dụng Hadoop trên Eclipse

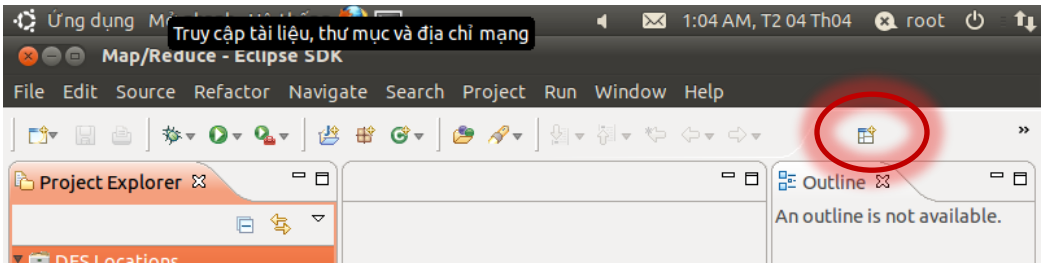
a. Download và cài đặt plug-in

Các bạn download:

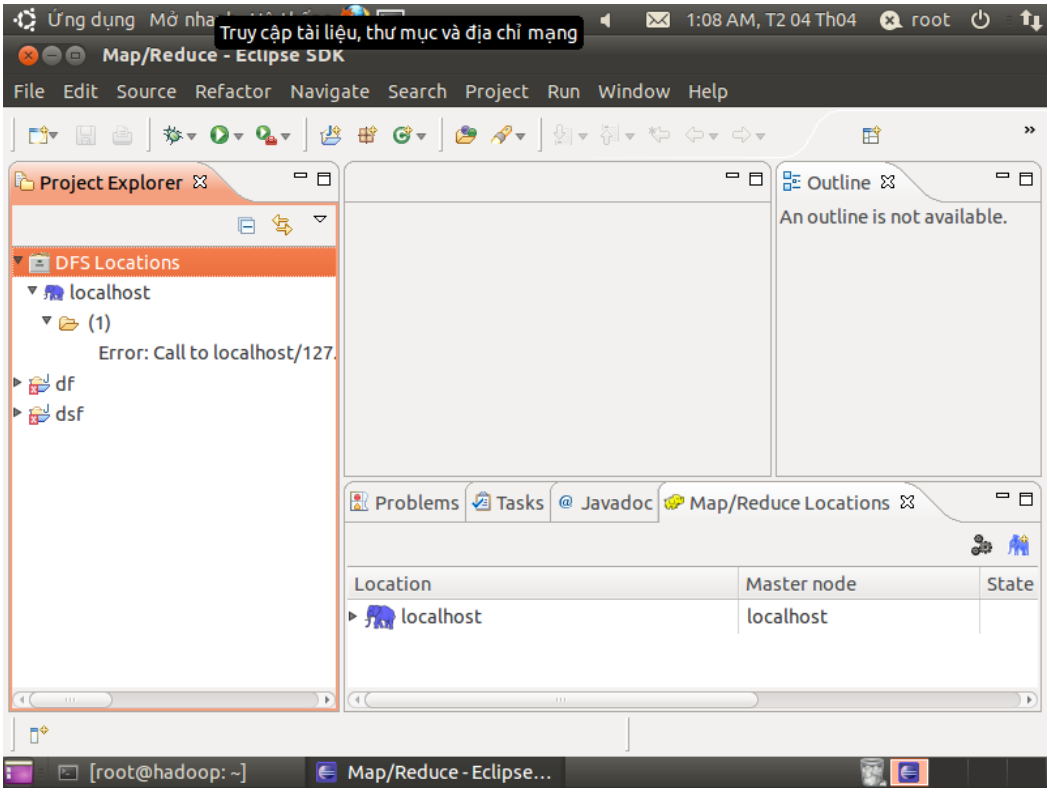
- Eclipse SDK Version: 3.5.2
- Hadoop plug-in cho Eclipse: `hadoop-0.20.1-eclipse-plugin.jar`
- Copy hadoop-0.20.1-eclipse-plugin.jar vào trong thư mục plug-ins của Eclipse

b. Cài đặt MapReduce location

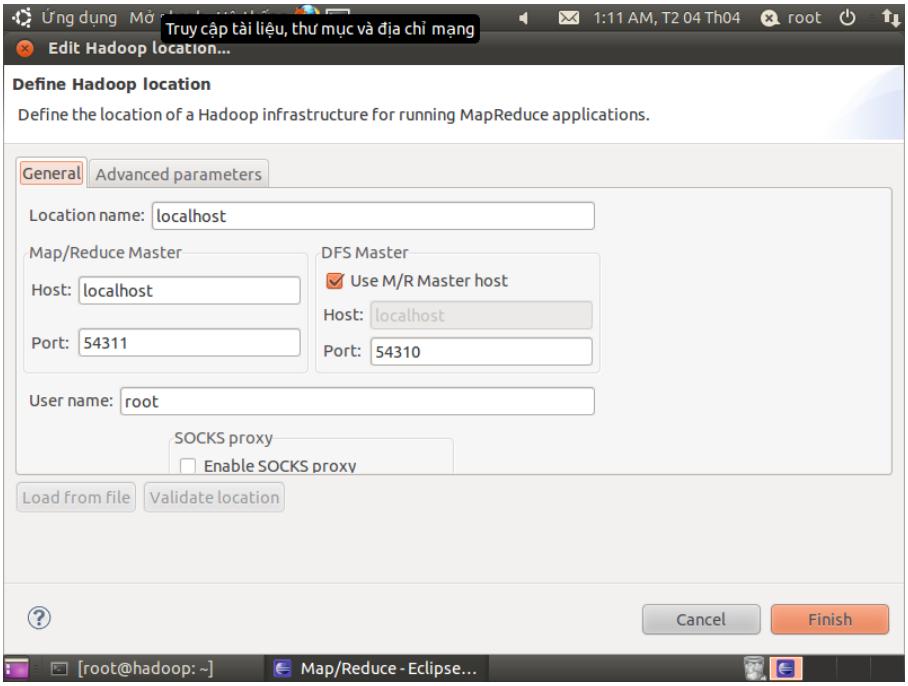
Khởi động Eclipse, bạn bấm vào nút trong vòng đỏ:



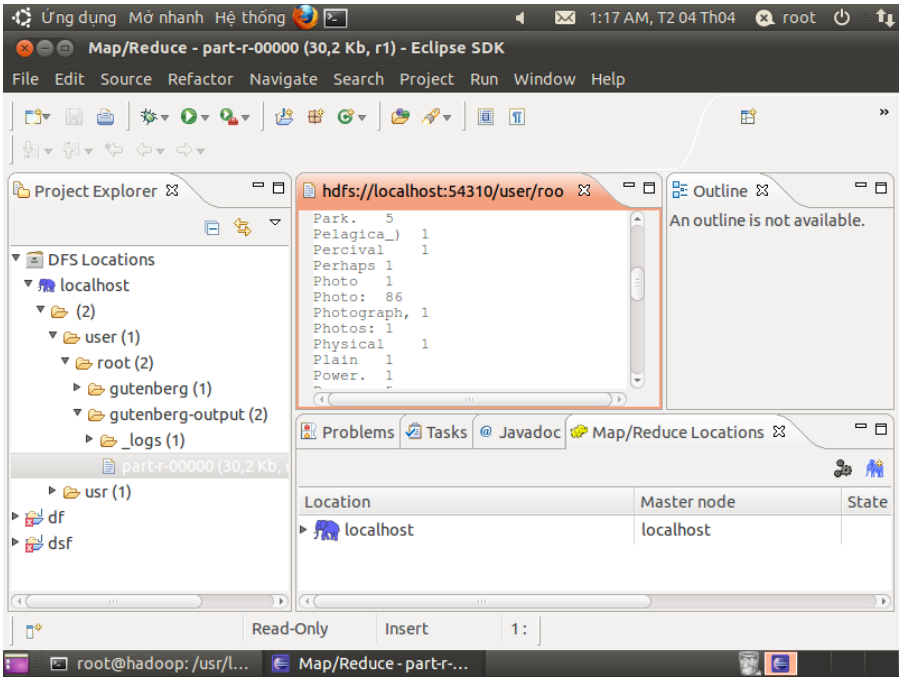
Sau đó chọn Other… => MapRecude => OK



Kích chuột phải vào phần trống của Location trong TAB Map/Recude Locations, chọn New Hadoop location… Và điền các tham số như hình dưới:



Khởi động Hadoop cluster như trên, và kiểm tra DFS như hình dưới đây



Phần III. Thành phần của Hadoop

1. Một số thuật ngữ.

- **MapReduce job** là một đơn vị của công việc mà khách hàng (client) muốn được thực hiện: nó bao gồm dữ liệu đầu vào, chương trình MapReduce, và thông tin cấu hình. Hadoop chạy các công việc (job) này bằng cách chia nó thành các nhiệm vụ (task), trong đó có hai kiểu chính là : các nhiệm vụ map (map task) và các nhiệm vụ reduce (reduce task)
- Có hai loại node điều kiển quá trình thực hiện công việc (job): một **jobtracker** và một số **tasktracker**. Jobtracker kết hợp tất cả các công việc trên hệ thống bằng cách lập lịch công việc chạy trên các tasktracker. Tasktracker chạy các nhiệm vụ (task) và gửi báo cáo thực hiện cho jobtracker, cái lưu giữ các bản nghi về quá trình xử lý tổng thể cho mỗi công việc (job)
- Hadoop chia đầu vào cho mỗi công việc MapReduce vào các mảnh (piece) có kích thước cố định gọi là các input split hoặc là các split. Hadoop tạo ra một task map cho mỗi split, cái chạy mỗi nhiệm vụ map do người sử dụng định nghĩa cho mỗi bản ghi (record) trong split.
- Có rất nhiều các split , điều này có nghĩa là thời gian xử lý mỗi split nhỏ hơn so với thời gian xử lý toàn bộ đầu vào. Vì vậy, nếu chúng ta xử lý các split một cách song song, thì quá trình xử lý sẽ tốt hơn cân bằng tải, nếu các split nhỏ, khi đó một chiếc máy tính nhanh có thể xử lý tương đương nhiều split trong quá trình thực hiện công việc hơn là một máy tính chậm. Ngay cả khi các máy tính giống hệt nhau, việc xử lý không thành công hay các công việc khác đang chạy đồng thời làm cho cần bằng tải như mong muốn, và chất lượng của cân bằng tải tăng như là chia các splits thành các hạt mịn hơn
- Mặt khác, nếu chia tách quá nhỏ, sau đó chi phí cho việc quản lý các split và của tạo ra các map task bắt đầu chiếm rất nhiều tổng thời gian của quá trình xử lý công việc. Đối với hầu hết công việc, kích thước split tốt nhất thường là kích thước của một block của HDFS, mặc định là 64MB, mặc dù nó có thể thay đổi được cho mỗi cluster (cho tất cả các file mới được tạo ra) hoặc định rõ khi mỗi file được tạo ra.
- Hadoop làm tốt nhất các công việc của nó chạy các map task trên một node khi mà dữ liệu đầu vào của nó cư trú ngay trong HDFS. Nó được gọi là tối ưu hóa dữ liệu địa phương. Bây giờ chúng ta sẽ làm rõ tại sao kích thước split tối ưu lại bằng kích thước của block: nó là kích thước lớn nhất của một đầu vào mà có thể được đảm bảo để được lưu trên một node đơn. Nếu split được chia thành 2 block, nó sẽ không chắc là bất cứ node HDFS nào lưu trữ cả hai block, vì vaayjmootj số split phải được chuyển trên mạng đến node chạy map tast, như vậy rõ ràng là sẽ ít hiệu quả hơn việc chạy toàn bộ map task sử dụng dữ liệu cục bộ.
- Các map task ghi đầu ra của chúng trên đĩa cụ bộ, không phải là vào HDFS. Tại sao lại như vậy? Đầu ra của map là đầu ra trung gian, nó được xử lý bởi reduce task để tạo ra

đầu ra cuối cùng , và một khi công việc được hoàn thành đầu ra của map có thể được bỏ đi. Vì vậy việc lưu trữ nó trong HDFS, với các nhân bản, là không cần thiết. Nếu các node chạy maptask bị lỗi trước khi đầu ra map đã được sử dụng bởi một reduce task, khi đó Hadoop sẽ tự động chạy lại map task trên một node khác để tạo ra một đầu ra map.

- Khi "chạy Hadoop" có nghĩa là chạy một tập các trình nền - daemon, hoặc các chương trình thường trú, trên các máy chủ khác nhau trên mạng của bạn. Những trình nền có vai trò cụ thể, một số chỉ tồn tại trên một máy chủ, một số có thể tồn tại trên nhiều máy chủ. Các daemon bao gồm:

- NameNode
- DataNode
- SecondaryNameNode
- JobTracker
- TaskTracker

2. Các trình nền của Hadoop

2.1. NameNode

Là một trình nền quan trọng nhất của Hadoop - các NameNode. Hadoop sử dụng một kiến trúc master/slave cho cả lưu trữ phân tán và xử lý phân tán. Hệ thống lưu trữ phân tán được gọi là Hadoop File System hay HDFS. NameNode là master của HDFS để chỉ đạo các trình nền DataNode slave để thực hiện các nhiệm vụ I/O mức thấp. NadeNode là nhân viên kế toán của HDFS; nó theo dõi cách các tập tin của bạn được phân kia thành các block, những node nào lưu các khối đó, và "kiểm tra sức khỏe" tổng thể của hệ thống tệp phân tán.

Chức năng của NameNode là nhớ (memory) và I/O chuyên sâu. Như vậy, máy chủ lư trữ NameNode thường không lưu trữ bất cứ dữ liệu người dùng hoặc thực hiện bất cứ một tính toán nào cho một ứng dụng MapReduce để giảm khối lượng công việc trên máy. Điều này có nghĩa là máy chủ NameNode không gấp đôi (double) như là DataNode hay một TaskTracker.

Có điều đáng tiếc là có một khía cạnh tiêu cực đến tầm quan trọng của NameNode nó có một điểm của thất bại của một cụm Hadoop của bạn. Đối với bất cứ một trình nền khác, nếu các nút máy của chúng bị hỏng vì lý do phần mềm hay phần cứng, các Hadoop cluster có thể tiếp tục hoạt động thông suốt hoặc bạn có thể khởi động nó một cách nhanh chóng. Nhưng không thể áp dụng cho các NameNode.

2.2. DataNode

Mỗi máy slave trong cluster của bạn sẽ lưu trữ (host) một trình nền DataNode để thực hiện các công việc nào đó của hệ thống file phân tán - đọc và ghi các khối HDFS

tới các file thực tế trên hệ thống file cục bộ (local filesytem). Khi bạn muốn đọc hay ghi một file HDFS, file đó được chia nhỏ thành các khối và NameNode sẽ nói cho các client của bạn nơi các mỗi khối trình nền DataNode sẽ nằm trong đó. Client của bạn liên lạc trực tiếp với các trình nền DataNode để xử lý các file cục bộ tương ứng với các block. Hơn nữa, một DataNode có thể giao tiếp với các DataNode khác để nhân bản các khối dữ liệu của nó để dự phòng.

Hình 2.1 minh họa vai trò của NameNode và DataNode. Trong các số liệu này chỉ ra 2 file dữ liệu, một cái ở /user/chuck/data1 và một cái khác ở /user/james/data2. File Data1 chiếm 3 khối, mà được biểu diễn là 1 2 3. Và file Data2 gồm các khối 4 và 5. Nội dung của các file được phân tán trong các DataNode. Trong minh họa này, mỗi block có 3 nhân bản. Cho ví dụ, lock 1 (sử dụng ở data1) là được nhân bản hơn 3 lần trên hầu hết các DataNodes. Điều này đảm bảo rằng nếu có một DataNode gặp tai nạn hoặc không thể truy cập qua mạng được, bạn vẫn có thể đọc được các tệp tin.

Các DataNode thường xuyên báo cáo với các NameNode. Sa khi khởi tạo, mỗi DataNode thông báo với NameNode của các khối mà nó hiện đang lưu trữ. Sau khi Mapping hoàn thành, các DataNode tiếp tục thăm dò ý kiến NameNode để cung cấp thông tin về thay đổi cục bộ cũng như nhận được hướng dẫn để tạo, di chuyển hoặc xóa các blocks từ đĩa địa phương (local).

2.3. Secondary NameNode

Các Secondary NameNode (SNN) là một trình nền hỗ trợ giám sát trạng thái của các cụm HDFS. Giống như NameNode, mỗi cụm có một SNN, và nó thường trú trên một máy của mình. Không có các trình nền DataNode hay TaskTracker chạy trên cùng một server. SNN khác với NameNode trong quá trình xử lý của nó không nhận hoặc ghi lại bất cứ thay đổi thời gian thực tới HDFS. Thay vào đó, nó giao tiếp với các NameNode bằng cách chụp những bức ảnh của siêu dữ liệu HDFS (HDFS metadata) tại nhưng khoảng xác định bởi cấu hình của các cluster.

Như đã đề cập trước đó, NameNode là một điểm truy cập duy nhất của lỗi (failure) cho một cụm Hadoop, và các bức ảnh chụp SNN giúp giảm thiểu thời gian ngừng (downtime) và mất dữ liệu. Tuy nhiên, một NameNode không đòi hỏi sự can thiệp của con người để cấu hình lại các cluster sẻ dụng SSN như là NameNode chính.

2.4. JobTracker

Trình nền JobTracker là một liên lạc giữa ứng dụng của bạn à Hadoop. Một khi bạn gửi mã nguồn của bạn tới các cụm (cluster), JobTracker sẽ quyết định kế hoạch thực hiện bằng cách xác định những tập tin nào sẽ xử lý, các nút được giao các nhiệm vụ khác nhau, và theo dõi tất cả các nhiệm vụ khi dúng đang chạy. Nếu một nhiệm vụ (task) thất bại (fail), JobTracker sẽ tự động chạy lại nhiệm vụ đó, có thể trên một node khác, cho đến một giới hạn nào đó được định sẵn của việc thử lại này.

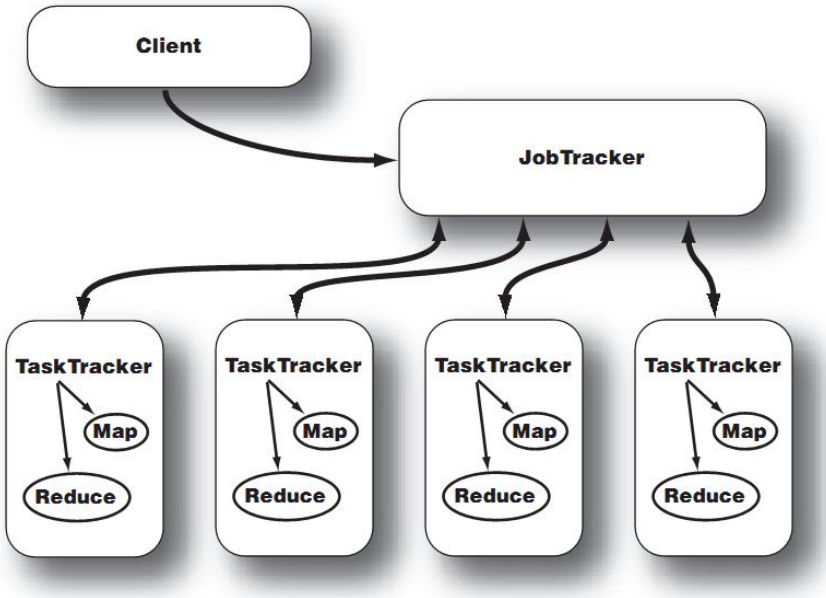
Chỉ có một JobTracker trên một cụm Hadoop. Nó thường chạy trên một máy chủ như là một nút master của cluster.

2.5. TaskTracker

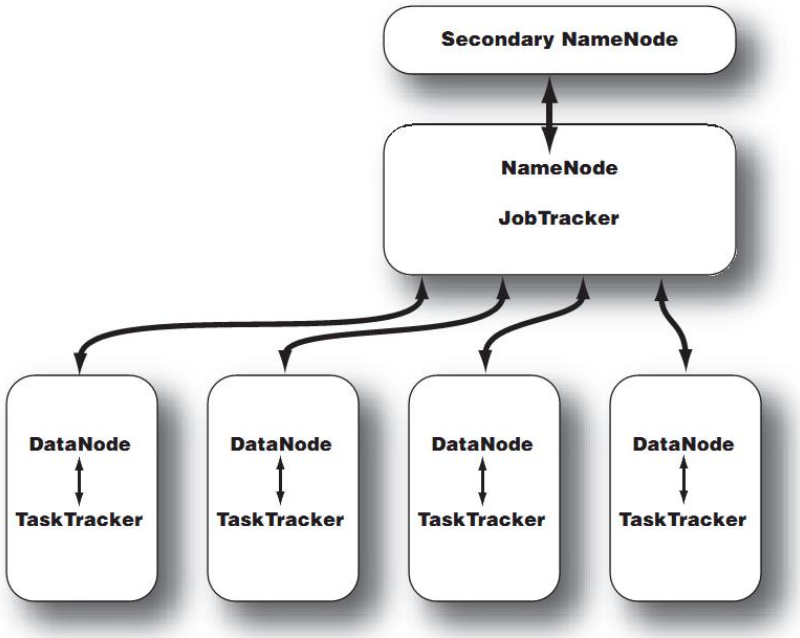
Như với các trình nền lưu trữ, các trình nền tính toán cũng phải tuân theo kiến trúc master/slave: JobTracker là giám sát tổng việc thực hiện chung của một công việc MapRecude và các taskTracker quản lý việc thực hiện các nhiệm vụ riêng trên mỗi node slave. Hình 2.2 minh họa tương tác này.

Mỗi TaskTracker chịu trách nhiệm thực hiện các task riêng mà các JobTracker giao cho. Mặc dù có một TaskTracker duy nhất cho một node slave, mỗi TaskTracker có thể sinh ra nhiều JVM để xử lý các nhiệm vụ Map hoặc Reduce song song.

Một trong những trách nhiệm của các TaskTracker là liên tục liên lạc với JobTracker. NeeusJobTracker không nhận được nhịp đập từ mootjTaskTracker trong vòng một lượng thời gian đã quy định, nó sẽ cho rằng TaskTracker đã bị treo (cashed) và sẽ gửi lại nhiệm vụ tương ứng cho các nút khác trong cluster.



Hình 2.2 Tương tác giữa JobTracker và TaskTracker. Sau khi client gọi JobTracker bắt đầu công việc xử lý dữ liệu, các phân vùng JobTracker làm việc và giao các nhiệm vụ Map và Recude khác nhau cho mỗi TaskTracker trong cluster.



Hình 2.3 Cấu trúc liên kết của một nhóm Hadoop điển hình. Đó là một kiến trúc master/slave trong đó NameNode và JobTracker là Master và DataNode & TaskTracker là slave.

Cấu trúc liên kết này có một node Master là trình nền NameNode và JobTracker và một node đơn với SNN trong trường hợp node Master bị lỗi. Đối với các cụm nhỏ, thì SNN có thể thường chú trong một node slave. Mặt khác, đối với các cụm lớn, phân tách NameNode và JobTracker thành hai máy riêng. Các máy slave, mỗi máy chỉ lưu trữ một DataNode và Tasktracker, để chạy các nhiệm vụ trên cùng một node nơi lưu dữ liệu của chúng.

Chúng tôi sẽ thiết lập một cluster Hadoop đầy đủ với mẫu như trên bằng cách đầu tiên thiết lập các nút Master và kiểm soát kênh giữa các node. Nếu một cluster Hadoop của bạn đã có sẵn, bạn có thể nhay qua phần cài đặt kênh Secure Shell (SSH) giữa các node. Bạn cũng có một vài lựa chọn để chạy Hadoop là sử dụng trên Một máy đơn, hoặc chế độ giả phân tán. Chúng sẽ hữu dụng để phát triển. Cấu hình Haddop để chạy trong hai node hoặc các cluster chuẩn (chế độ phân tán đầy đủ) được đề cập trong chương 2.3

Phần IV. Lập trình MapReduce cơ bản

1. Tổng quan một chương trình MapReduce

Như chúng ta đã biết, một chương trình MapReuduce xử lý dữ liệu bằng cách tao thác với các cặp (key/value) theo công thức chung:

map: (K1,V1) ⟶ list(K2,V2)
reduce: (K2,list(V2)) ⟶ list(K3,V3)

Trong phần này chúng ta học chi tiết hơn về từng giai đoạn trong chương trình MapReduce điển hình. Hình 3.1 biểu diễn biểu đồ cao cấp của toàn bộ quá trình, và chúng tôi tiếp tục mỏ xẻ từng phần:

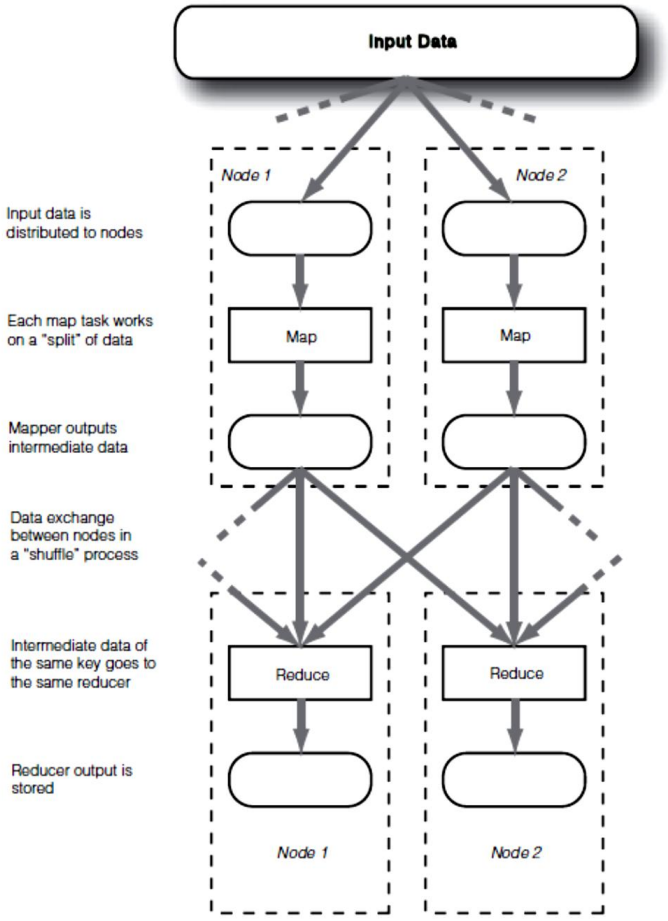


Figure 3.1 The general MapReduce data flow. Note that after distributing input data to different nodes, the only time nodes communicate with each other is at the "shuffle" step. This restriction on communication greatly helps scalability.

2. Các loại dữ liệu mà Hadoop hỗ trợ

MapReduce framework có một các định nghĩa cặp khóa key/value tuần tự để có thể di chuyển chúng qua mạng, và chỉ các lớp hỗ trợ kiểu tuần tự có chúng năm giống như key và value trong framework.

Cụ thể hơn, các lớp mà implement giao diện Writable có thể làm value, và các lớp mà implement giao diện WritableComparable<T> có thể làm cả key và value. Lưu ý rằng giao diện WritableComparable<T> là một sự kết hợp của Writeable và giao diện java.lang.Comparable<T>. Chúng ta cần yêu cầu so sánh các khóa bởi vì chúng sẽ được sắp xếp ở giai đoạn reduce, trong khi giá trị thì đơn giản được cho qua.

Hadoop đi kèm một số lớp được định nghĩa trước mà implement WritableComparable, bao gồm các lớp bọ cho tát cả các loại dữ liệu cơ bản như trong bảng 3.1 sau:

Table 3.1 List of frequently used types for the key/value pairs. These classes all implement the `WritableComparable` interface.

Class	Description
`BooleanWritable`	Wrapper for a standard Boolean variable
`ByteWritable`	Wrapper for a single byte
`DoubleWritable`	Wrapper for a Double
`FloatWritable`	Wrapper for a Float
`IntWritable`	Wrapper for a Integer
`LongWritable`	Wrapper for a Long
`Text`	Wrapper to store text using the UTF8 format
`NullWritable`	Placeholder when the key or value is not needed

Bạn cũng có thể tùy chỉnh một kiểu dữ liệu bằng cách implement Writable (hay WritableComparable<T>). Như ví dụ 3.2 sau, lớp biểu diễn các cạnh trong mạng, như đường bay giữa hai thành phố:

```
import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

import org.apache.hadoop.io.WritableComparable;

public class Edge implements WritableComparable<Edge>{

	private String departureNode; //Node khoi hanh
	private String arrivalNode;	//Node den

	public String getDepartureNode(){
		return departureNode;
	}

	@Override
	public void readFields(DataInput in) throws IOException {
		// TODO Auto-generated method stub
		departureNode = in.readUTF();
		arrivalNode = in.readUTF();
	}

	@Override
	public void write(DataOutput out) throws IOException {
		// TODO Auto-generated method stub
		out.writeUTF(departureNode);
		out.writeUTF(arrivalNode);
	}

	@Override
	public int compareTo(Edge o) {
		// TODO Auto-generated method stub
		return (departureNode.compareTo(o.departureNode) != 0)?
				departureNode.compareTo(departureNode):
				arrivalNode.compareTo(o.arrivalNode);
	}
}
```

Lớp **Edge** thực hiện hai phương thức **readFields()** và **write()** của giao diện Writeable. Chúng làm việc với lớp Java **DataInput** và **DataOutput** để tuần tự nội dung của các lớp. Thự hiện phương pháp **compareTo()** cho interface **Comparable**. Nó trả lại giá trị -1, 0, +1.

Với kiểu dữ liệu được định nghĩa tại giao diện, chúng ta có thể tiến hành giai đoạn đầu tiên của xử lý luồng dữ liệu như trong hình 3.1: mapper.

2.1. Mapper

Để phục làm một **Mapper**, một lớp implements từ interface **Mapper** và kế thừa từ lớp **MapReduceBase**. Lớp **MapReduceBase**, đóng vai trò là lớp cơ sở cho cả **mapper** và **reducer**. Nó

bao gồm hai phương thức hoạt động hiệu quả như là hàm khởi tạo và hàm hủy của lớp:

- `void configure(JobConf job)` – trong hàm nay, bạn có thể trích xuất các thông số cài đặt hoặc bằng các file XML cấu hình hoặc trong các lớp chính của ứng dụng của bạn. Gọi cái hàm này trước khi xử lý dữ liệu.
- `void close()` – Như hành động cuối trước khi chấm dứt nhiệm vụ map, hàm này nên được gọi bất cứ khi nào kết thúc – kết nối cơ sở dữ liệu, các file đang mở.

Giao diện Mapper chịu trách nhiệm cho bước xử lý dữ liệu. Nó sử dụng Java Generics của mẫu `Mapper<K1,V1,K2,V2>` chỗ mà các lớp key và các lớp value mà implements từ interface `WriteableComparable` và `Writable`. Phương pháp duy nhất của nó để xử lý các cặp (key/value) như sau:

```
void map(K1 key, V1 value, OutputCollector<K2,V2> output,
            Reporter reporter
      ) throws IOException
```

Phương thức này tạo ra một danh sách (có thể rỗng) các cặp (K2, V2) từ một cặp đầu vào (K1, V1). `OuputCollector` nhận kết quả từ đầu ra của quá trình mapping, và `Reporter` cung cấp các tùy chọn để ghi lại thông tin thêm về mapper như tiến triển công việc.

Hadoop cung cấu một vài cài đặt Mapper hữu dụng. Bạn có thể thấy một vài cái như trong bản 3.2 sau:

Bảng 3.2. Một vài lớp thực hiện Mapper được định nghĩa trước bởi Hadoop

- `IdentityMapper<K,V>` : với cài đặt Mapper <K, V, K, V> và ánh xạ đầu vào trực tiếp vào đầu ra
- `InverseMapper<K,V>` : với cài đặt Mapper<K, V, V, K> và đảo ngược cặp (K/V)
- `RegexMapper<K>` : với cài đặ Mapper<K, Text, Text, LongWritable> và sinh ra cặp (match, 1) cho mỗi ánh xạ (match) biểu thức thường xuyên.
- `TokenCountMapper<K>` : với cài đặt Mapper<K, Text, Text, LongWritable> sinh ra một cặp (token, 1) khi một giá trị đầu vào là tokenized.

2.2. Reducer

Với bất cứ cài đặt `Mapper`, một `reducer` đầu tiên phải mở rộng từ lớp MapReduce base để cho phép cấu hình và dọn dẹp. Ngoài ra, nó cũng phải implement giao diện `Reducer` chỉ có một phương thức duy nhất sau:

```
void reduce(K2 key, Iterator<V2> values,
      OutputCollector<K3,V3> output, Reporter reporter
) throws IOException
```

Khi nhận được các task từ đầu ra của các Mapper khác nhau, nó sắp xếp các dữ liệu đến theo các khóa của các cặp (key/value) và nhóm lại các giá trị cùng khóa. Hàm `reduce()` được gọi sau đó, nó sinh ra một danh sách (có thể rỗng) các cặp (K3, V3) bằng cách lặp lại trên các giá trị

được liên kết với khóa đã cho. `OutputCollector` nhận từ đầu ra của quá trình reduce và ghi nó ra đầu ra file. `Reporter` cung cấp tùy chọn ghi lại thông tin thêm về reducer như là một tiến triển công việc.

Bảng 3.3 liệt kê một vài reducer cơ bản được triển khai cung cấp bởi Hadoop

- `IdentityReducer<K, V>` : với cài đặt Reducer <K, V, K, V> và ánh xạ đầu vào trực tiếp vào đầu ra
- `LongSumReducer<K>` : với cài đạt Reducer <K, LongWritable, K, LongWritable> và quyết định thổng hợp tất cả các giá trị tương tứng với các key đã cho

Có một bước quan trọng giữa 2 bước map và reduce: chỉ đạo kết quả của các Mapper tới các Reducer. Đây là trách nhiệm của partitioner (phân vùng).

2.3. Partitioner – chuyển hướng đầu ra từ Mapper

Với nhiều reducer, chúng ta cần một vài cách để xác định một trong những cặp (key/value) là đầu ra của một mapper được gửi đi. Hành vi mặc định là băm key để xác định reducer. Hadoop thực thi kiến lược này bằng cách sử dụng lớp `HashPartitioner`. Thỉnh thoáng lớp này sẽ làm việc hiệu quả. Trở lại ví dụ `Edge` như trong phần 3.2.1.

Giả sử bạn sử dụng lớp Edge để phân tích dữ liệu thông tin chuyến bay để xác định số lượng hành khác khởi hành từ mỗi sân bay. Ví dụ như dữ liệu sau:

```
(San Francisco, Los Angeles) Chuck Lam
(San Francisco, Dallas) James Warren
...
```

Nếu bạn sử dụng `HashPartitioner`, hai dòng sẽ được gửi tới 2 reducer khác nhau. Số các điểm khởi hành sẽ được xử lý 2 lần và cả hai lần đều sai.

Làm thế nào chúng ta có thể tùy chỉnh Partitioner cho ứng dụng của bạn? Trong tình hình này, chúng ta muốn tất cả các đường bay với một điểm khởi hành sẽ được gửi tới cùng một reducer. Điều này được dễ làm bằng cách băm `departureNode` của `Edge`:

```
public class EdgePartitioner implements Partitioner<Edge, Writable>{

      @Override
      public int getPartition(Edge key, Writable value, int
      numPartitions){
      return key.getDepartureNode().hashCode() % numPartitions;
      }

      @Override
      public void configure(JobConf conf) { }
}
```

Phần V. Sơ lược về các thuật toán tin sinh

5.1. Thuật toán Blast

Ý tưởng của BLAST dựa trên cơ sở xác suất rằng những chuỗi bắt cặp trình tự (alignment) thường sở hữu nhiều đoạn chuỗi con có tính tương tự cao. Những chuỗi con này được mở rộng để tăng tính tương tự trong quá trình tìm kiếm.

Thuật toán của BLAST có 2 phần, một phần tìm kiếm và một phần đánh giá thống kê dựa trên kết quả tìm được.

Thuật toán tìm kiếm của BLAST bao gồm 3 bước sau:

Bước 1: BLAST tìm kiếm các chuỗi con ngắn với chiều dài cố định W có tính tương tự cao (không cho phép khoảng trống gaps) giữa chuỗi truy vấn và các chuỗi trong cơ sở dữ liệu.

Những chuỗi con với chiều dài W được BLAST gọi là một từ (word). Giá trị W tham khảo cho Protein là 3 và DNA là 11.

Những chuỗi con này được đánh giá cho điểm dựa trên ma trận thay thế (Substitutionsmatrix) BLOSUM hoặc PAM, những chuỗi con nào có số điểm lớn hơn một giá trị ngưỡng T (threshold value) thì được gọi là tìm thấy và được BLAST gọi là Hits.

Ví dụ, khi cho sẵn các chuỗi AGTTAH và ACFTAQ và một từ có chiều dài W = 3, BLAST sẽ xác định chuỗi con TAH và TAQ với số điểm theo ma trận PAM là 3 + 2 + 3 = 8 và gọi chúng là một Hit.

Bước 2: BLAST tiếp tục tìm kiếp những cặp Hits tiếp theo dựa trên cơ sở những Hit đã tìm được trong bước 1. Những cặp Hits này được BLAST giới hạn bởi một giá trị cho trước d, gọi là khoảng cách giữa những Hits. Những cặp Hits có khoảng cách lớn hơn d sẽ bị BLAST bỏ qua.

Giá trị d phụ thuộc vào độ dài W ở bước 1, ví dụ nếu W = 2 thì giá trị d đề nghị là d=16.

Bước 3: Cuối cùng BLAST mở rộng những cặp Hits đã tìm được theo cả hai chiều và đồng thời đánh số điểm. Quá trình mở rộng kết thúc khi điểm của các cặp Hits không thể mở rộng thêm nữa.

Một điểm chú ý ở đây là phiên bản gốc của BLAST không cho phép chỗ trống (gap) trong quá trình mở rộng, nhưng ở phiên bản mới hơn đã cho phép chỗ trống.

Những cặp Hits sau khi mở rộng có điểm số cao hơn một giá trị ngưỡng S (threshold value) thì được BLAST gọi là "cặp điểm số cao" (high scoring pair) HSP.

Ví dụ, với chuỗi AGTTAHTQ và ACFTAQAC với Hit TAH và TAQ sẽ được mở rộng như sau:

```
AGTTAHTQ

xxx|||x

ACFTAQAC
```

Những cặp HSP đã tìm được được BLAST sắp xếp theo giá trị đánh giá giảm dần, đưa ra màn hình, và thực hiện phần đánh giá thống kê trên những cặp HSP này.

Trong phần đánh giá thống kê, BLAST dựa trên cơ sở đánh giá của một cặp HSP để tính ra một giá trị gọi là ''Bit-Score'', giá trị này không phụ thuộc vào ma trận thay thế và được sử dụng để đánh giá chất lượng của các bắt cặp. Giá trị càng cao chứng tỏ khả năng tương tựu của các bắt cặp càng cao. Ngoài ra BLAST tính toán một giá trị trông đợi E-Score (Expect-Score) phụ thuộc vào Bit-Score. Giá trị E-Score này thể hiện xác suất ngẫu nhiên của các bắt cặp, giá trị càng thấp càng chứng tỏ những bắt cặp này được phát sinh theo quy luật tự nhiên, ít phụ thuộc vào tính ngẫu nhiên.

5.2. Thuật toán Landau-Vishkin

5.2.1. Một số khái niệm

Cho các chuỗi $T = t_1 \dots t_n$ và $P = p_1 \dots p_m$ với $|T| = n$ và $|P| = m$ $(m \leq n)$ trên một bảng chữ cái $\sum$, chúng ta có một vài định nghĩa sau:

- ε là một chuỗi rỗng
- P là chuỗi con (substring) của T khi $m \leq n$ và $p_1 \dots p_m = t_i \dots t_{i+m-1}$ với $i \geq 1$ và $i + m - 1 \leq n$. Nếu $m < n$ ta nói rằng P là chuỗi con hoàn toàn của T
- P là tiền tố của T nếu $m \leq n$ và $p_i = t_i$ với $1 \leq i \leq m$ nếu $m < n$ thì ta nói rằng P là tiền tố hoàn toàn của T
- P là hậu tốt của T nếu $p_1 \dots p_m = t_i \dots t_{i+m-1}$ với $i + m + 1 = n$. Nếu $i > 1$ thì chúng ta nói rằng P là hậu tố hoàn toàn của T. Chúng ta cũng nói rằng $T_i = t_i \dots t_n$ khi $i \geq 1$ là hậu tố thứ i của T (đó là hậu tố của T bắt đầu từ vị trí i)
- Tiền tố chung dài nhất - $longest\ common\ prefix$ (LCP) của T và P là chuỗi lớn nhất $L = l_1 \dots l_k$ mà $0 \leq k \leq m$ và $l_1 \dots l_k = p_1 \dots p_k = t_1 \dots t_k$. Nếu $k = 0$ thì $L = \varepsilon$. Chú ý là $LCP_{P,T}(i,j)$ biểu diễn LCP của $P_i$và T_j. Nếu P và T là rõ ràng trong ngữ cảnh thì chúng ta viết đơn giản là $LCP(i,j)$.
- Phần mở rộng chung dài nhất – $longest\ common\ extension$ (LCE) của T và P tại vị trí (i,j) là độ dài của LCP của P_i và T_j. Nếu P và T là rõ ràng trong ngữ cảnh, chúng ta có thể viết đơn giản là $LCE(i,j)$.

Trong phần này chúng ta gọi chuỗi T là $text$ và chuỗi chuỗi P là $pattern$.

5.2.2. Khớp xâu xấp xỉ (*Approximate String Matching*)

Định nghĩa 1: *Edit distance* (Khoảng cách sửa đổi)

Khoảng cách sửa đổi giữa hai xâu $P = p_1 \dots p_m$ và $T = t_1 \dots t_n$ là số lượng tối thiểu các hoạt động càn thiết để chuyển đổi P thành T hay T thành P, trong đó các hoạt động được định nghĩa như sau:

- *Thay thế*: Khi một kí tự p_i của P được thay thế bằng một kí tự t_j của T
- *Thêm:* Khi một kí tự p_i của P được thêm vào vị trí j của T
- *Xóa:* Khi một kĩ tự p_i được xóa khỏi P

Một chuỗi các hoạt động cần thiết để chuyển đổi P thành T được gọi là bản ghi sửa đổi (edit transcript) của P thành T.

Một xắp hàng (alignment) của P và T là một dại diện của các hoạt động áp dụng trên P và T, thường đặt một chuỗi lên trên một chuỗi khác, và làm đầy bằng các dấu gạch ngang ('-') vào vị trí trong P và T tại những chỗ mà một khoảng trống được thêm vào để mỗi kí tự hoặc khoảng trống trên một trong hai string đối diện là kí tự duy nhất hoặc khoảng trống duy nhất trên P và T.

Định nghĩa 2: *Approximate string matching with k differences* (khớp xâu xấp xỉ với k điểm khác)

Khớp xâu với k điểm khác giữa một khuôn mẫu P và văn bản T là vấn đề của việc tìm kiếm mỗi cặp vị trí (i, j) trong T sao cho khoảng cách sửa đổi giữa P và $t_i \dots t_j$ nhiều nhất là k.

5.2.3. Giải pháp quy hoạch động

Chúng ta có thể tìm thấy khoảng cách sửa đổi $D(j, j)$ giữa hai chuỗi $p_1 \dots p_i$ và $t_1 \dots t_j$ từ khoảng cách:

- $D(i-1, j-1)$ giữa $p_1 \dots p_{i-1}$ và $t_1 \dots t_{j-1}$
- $D(i-1, j)$ giữa $p_1 \dots p_{i-1}$ và $t_1 \dots t_j$
- $D(i, j-1)$ giữa $p_1 \dots p_i$ và $t_1 \dots t_{j-1}$

Bằng cách giải quyết quan hệ đệ quy:

$$D(i,j) = \begin{cases} i + j \\ \min[D(i-1, j-1) + d, D(i-1, j) + 1, D(i, j-1) + 1] \\ \quad với\ d = 0\ nếu\ p_i = t_j\ hoặc\ 1\ nếu\ p_i \neq t_j \\ \quad Nếu\ j = 0\ hay\ i = 0\ thì\ ngược\ lại \end{cases}$$

Mối quan hệ này có thể được tính toán bằng một ma trận quy hoạch động đơn giản $O(mn)$ sử dụng một bảng quy hoạch động $n + 1 \times m + 1$.

5.2.4. Cơ bản về thuật toán Landau-Vishkin

Landau-Vishkin trình diễn một thuật toán $O(kn)$ cho vấn để khớp xâu xấp xỉ với k điểm khác. Thuật toán này chia thành hai pha: pha tiền xử lý và pha lặp.

Trong pha tiền xử lý, các pattern và text được tiền xử lý với tính toán LCE có $O(1)$

Trong pha lặp, thuật toán lặp k lần trên mỗi đường chéo của bảng quy hoạch động và tìm ra tất cả các xắp hàng (match) của P với nhiều nhất k điểm khác.

Phần VI. Sơ lược về BlastReduce

6.1. Tóm tắt:

Thế hệ tiếp theo của máy trình tự DNA sinh ra một chuỗi dữ liệu với tốc độ chưa từng thấy, nhưng các thuật toán alignment xử lý trình tự đơn truyền thống cố gắng đấu tranh để theo kịp với chúng. BlastReduce là một thuật toán đọc mapping song song mới tối ưu cho sắp xếp dữ liệu chuỗi từ các máy tham chiếu tới bộ gen, để sử dụng trong phân tích sự đa dạng của sinh học, bao gồm khám phá SNP, kiểu gen, và cá thể gen. Nó được mô hình hóa sau khi sử dụng thuật toán liên kết chuỗi BLAST, nhưng sử dụng cài đặt hadoop của MapReduce để xử lý song song trên nhiều node tính toán. Để đánh giá hiệu quả của nó, BlastReduce đã được sử dụng để map cữ liệu chuỗi thế hệ tiếp theo với một tham chiếu tới hệ gen của vi khuẩn ở một loạt các cấu hình. Kết quả cho thấy quy mô của BlastReduce tăng tuyến tính theo số lượng các xử lý chuỗi, và với sự tăng tốc như tăng số lượng bộ vi xử lý. Trong một cấu hình kiêm tốn với 24 bộ vi xử lý, BlastReduce nhanh gấp 250 lần BLAST xử lý trên một nhân, và giảm thời gian xử lý từ vài ngày xuống còn vài phút ở cùng mức độ nhạy cảm.

6.2. Read Mapping

Sau khi lập trình tự AND mới được tạo ra để đọc thường được xắp hàng hoặc ánh xạ với chuỗi bộ gen tham khảo để tìm các vùng mà diễn ra việc đọc từng khoảng một. Một thuật toán ánh xạ đọc (read mapping) báo cáo tất cả các sắp hàng (alignment) mà có điểm trong ngưỡng điểm, thường thể hiện như số lượng tối đa có thể chấp nhận được của sự khác nhau giữa read và bộ gen tham chiếu (nói chung hầu hết là vào khoảng 1%-10% của chiều dài read). Các thuật toán liên kết có thể cho phép chỉ các mismatche là khác nhau, vấn đề k-mismatch, hoặc nó cũng có thể xem xét sắp hàng có dấu cách (gapped alignment) trong trường hợp thêm hoặc xóa kí tự, vấn đền k-khác nhau). Thuật toán xắp hàng chuỗi Smith-Waterman cổ điểntính toán các xắp hàng có dấu cách sử dụng quy hoạch động. Nó xem xét tất cả các xắp hàng có thể của một cặp các chuỗi với thời gian tỉ lệ thuận với độc dài của chúng. Một biến thể của thuật toán Smith-Waterman, được gọi là xắp hàng dải, bản chất cũng sử dụng quy hoạch động nhưng hạn chế việt tìm kiếm các xắp hàng với một số lượng nhỏ sự khác biệt. Với một cặp đơn các chuỗi tính toán một xắp hàng Smith-Waterman thì thường là một hoạt động nhanh, nhưng sẽ nên tính toán không khả thi khi só lượng các chuỗi tăng.

Thay vào đó, các nhà nghiên cứu sử dụng kĩ thuật hạt giống và mở rộng để đẩy nhanh tốc độ tìm kiếm rất giống với xắp hàng. Cái quan trọng là sự quan sát mà sự sắp hàng rất gióng nhau phải chắc chắn có ý nghĩa sắp xếp. Bằng cách sử dụng nguyên lý lồng chim bồ câu, với 20bp đọc align với một cái khác, bạn phải có ít nhất một xắp hàng 10bp trong một xắp hàng nào đó. Nói chung, một aligment có độ dài đấy đủ là m bp đọc với e mismatch phải chứa ít nhất một xắp hàng m/(e+1) bp. Một số thuật toán xắp chuỗi tuần tự, bao gồm cả thuật toán phổ biến là công cụ BLAST và MUMmer sử dụng kĩ thuật này để xắp hàng nhanh. Trong giai đoạn hạt giống, các công cụ này tìm kiếm các chuỗi con mà giống nhau giữa hai chuỗi. Ví dụ, BLAST xây dụng

một bảng băm có đọ dài cố định gối các chuỗi con được gọi là k-mers của chuỗi tham khảo để tìm hạt giống, và MUMmer xây dụng cây hậu tố của chuỗi tham khảo để tìm biến chiều dài lớn nhất của hạt giống. Sau đó trong pha mở rộng các công cụ tính toán giá chính xác trong dài xắp hàng Smith-Waterman giới hạn với chuỗi con tương đối ngắn gần các hạt giống được chia sẻ. Kĩ thuật này có thể giảm đáng kể thời gian cần thiết để xắp hàng các chỗi tại một mức nhạy cảm. Dù vậy, sự nhạy cả tăng bằng nhiều các khác nhau, chiều dài hạt giống giảm, hay số các lượng các hạt giống match ngẫu nhiên sẽ làm tăng tổng thời gian tính toán.

Thuật toán xắp hàng k-difference Landau-Vishkin là một thuậ toán quy hoạch động thay thế để xác định nếu hai xắp hàng hai chuỗi với hầu hết có k-difference. Không giống như thuật toán quy hoạch động Smith-Waterman, mà xây dựng tất cả các xắp hàng có thể, thuật toán Landau-Vishkin xây dựng chỉ các sắp hàng giống nhau tới mà có số lượng các điểm khác là cố định bằng cách tính toán có bao nhiêu kí tự trong sking có thể được xắp hàng với i=0 tới k điểm khác nhau. Số lượng các kí tự mà được sắp hàng sử dụng I điểm khác được tính toán từ kết quả của (i-1) bằng cách tính toán chính xác phần mở rộng có thể sau mở đầu một mismatch, một điểm thêm vào hoặc xóa từ cuối của xắp hàng i-1. Thuật toán kết thúc khi i=k+1, cho thấy không tồm tại xắp hàng k-difference cho các trình tự, hoặc kết thúc của chuỗi đã đạt được. Thuật toán nay rất nhan hơn so với thuậ toán Smitch-WaterMan đầy đủ với số lượng k nhỏ, bởi vì chỉ một số lượng nhỏ các xắp hàng tiềm năng.

6.3. Thuật toán BlastReduce

BlastReduce là một thuật toán đọc ánh xạ song song (parallel read mapping algorithm) viết bằng Java với Hadoop. Nó được mô hình trên thuật toán BLAST, và được tối ưu cho ánh xạ các đoạn read nhỏ từ các máy chuỗi thế hệ tiếp theo tới bộ gen tham khảo. Giống như BLAST, nó là thuật toán hạt giống và mở rộng, sử dụng các từ có độ dài cố định làm hạt giống. Nhưng không giống với BLAST, BlastReduce sử dụng thuật toán Landau-Vishkin để mở rộng các hạt giống một cách nhanh chóng để tìm các xắp hàng với hầu hết k-difference.

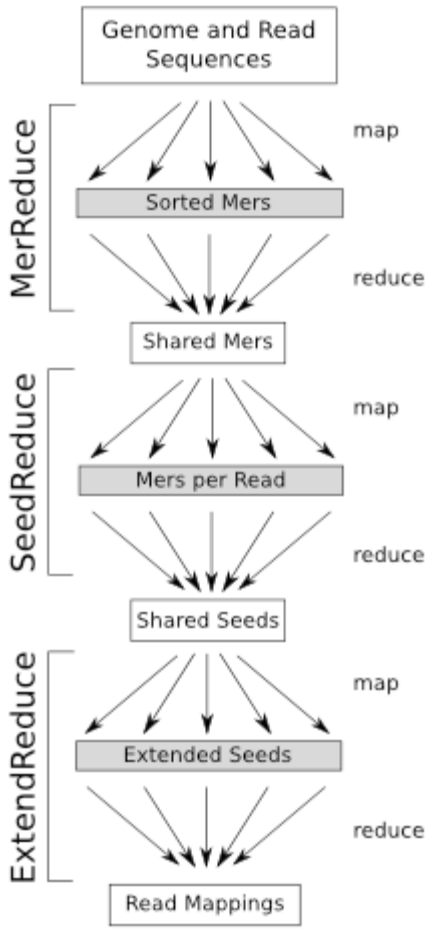
Cái thuật toán mở rộng này thích hợp hơn cho các đoạn read ngắn với số lượng nhỏ các khác biệt (thường k=1 hoặc k=2 cho 25-50bp read). Kích thước hạt giống (s) là tự động được tính toán dựa trên độ dài của read và số lượng tối đa các khác biệt (k) do người sử dụng đưa ra.

Đầu vào cho ứng dụng là một file đa fasta chứa một hoặc nhiều chuỗi tham khảo. Các file này đầu tiên được chuyển đổi thanh SequenceFile nén của Hadoop thích hợp để xử lý với Hadoop.SequenceFile mà không hỗ trợ các chuỗi có trình tự lớn hơn 65.565 ký tự để các chuỗi dài phân tách thành các khối. Các chuỗi được lưu trữ ở dạng cặp key-value trong SequenceFile như `(id, SeqInfo)` trong đó SeqInfor là một bộ (`sequence, start-offset, tag`), trong đó start_offset là độ lệch (`offset`) của khối chứa trong chuỗi đầy đủ. Những cái khối này xếp xếp chồng lên nhau với `s-1` bp để tất cả các hạt giống được biểu diễn chỉ một lần và các chuỗi tham khảo sẽ được báo rằng `tag=1`. Sau khi chuyển đổi, SequenceFile được copy vào HDFS để thuật toán read mapping có thể được thực thi.

Thuật toán read mapping yêu cầu 3 vòng MapReduce, và như mô tả dưới hình 1. Hai vòng đầu tiên là MerReduce và SeedReduce, tìm tất cả các match lớn nhất mà có độ dài ít nhất là s, và vòng cuối cùng là ExtendReduce, mở rộng các hạt giống với thuật toán Landau-Vishkin vào tất cả các xắp cặp với hầu hết k-difference.

6.3.1. MerReduce: tính các Mer giống nhau

Vòng MapReduce tìm các mer có độ dài s mà giống nhau giữa chuỗi read và chuỗi tham khảo. màm map xử lý tất cả các khối một cách độc lập, và các mer chỉ có trong read hoặc chỉ trong chuỗi tham khảo sẽ tự động được loại bỏ. ExtenReduce cần các chuỗi flanking – xếp chính xác các hạt giống cho xắp hàng, nhưng HDFS thì không hiệu quả cho truy cập ngẫu nhiên. Do đó, Flanking chuỗi (lên tới `<độ dài của read> - s + k` bp) bao gôm các mer của chuỗi read và chuỗi tham khảo vì vậy chúng sẽ có sẵn khi cần thiết.



Hình 1. Tổng quan về thuật toán BlastReduce sử dụng 3 vòng MapReduce. Các file tạm được sử dụng một cách nội bộ bởi MapReduce là Shared.

Map: Đối với mỗi mer trong chuỗi đầu vào, đầu ra của hàm map là (`mer, Merpos`), trong đó `MerPos` là cặp (`id, position, tag, left_flank, right_flank`). Nếu chuỗi là read (`tag =0`) vì thể tạo ra các bản ghi MerPost cho các chuỗi bổ xung đảo ngược. Màm map tạo ra tất cả là `s(M+N)` mer, trong đó M là tổng độ dài của chuỗi read, và `N` là tổng độ dài của chuỗi tham khảo. Sau khi tất cả các hàm map hoàn thành, Madoop sẽ sắp xếp nội bộ các cặp key-value, nhóm chúng tất cả các cặp mà có cùng mer vào một danh sách duy nhất các bản ghi `Merpos`.

Reduce: hàm reduce tạo ra các thông tin vị trí về các mer mà giống nhau ít nhất giữa một chuỗi tham khảo và một chuỗi đọc. Nó đòi hỏi phải có hai đường tuyền thông giữa một danh sách các bản ghi Merpos với mỗi mer. Nó đầu tiên sẽ quét danh sách để tìm bản ghi Merpos từ chuỗi tham khảo. Sau đó nó quét danh sách lần thứ hai và kết quả đầu ra là một cặp key-value `(read_id, ShareMer)` cho mối mer nào đó xuất hiện trong read và chuỗi tham khảo. Một sharedMer là một cụm bao gồm `(read_position, ref_id, ref_position, read_left_flank, read_right_flank, ref_left_flank,`

ref_right_flank).

6.3.2. SeedReduce: kết hợp các Mer nhất quán

Vòng MapReduce giảm thiểu số lượng các hạt giống bằng cách sát nhập các mer giống nhau vào trong một hạt giống lớn. Hai mer giống nhau sẽ kết hợp nếu chúng lệch 1bp trong chuỗi read và chuỗi reference. Hai mer phù hợp có thể được trộn lại một cách an toàn khi chúng ánh xạ thới các xắp đoạn giống nhau.

Map: Hàm map tạo ra các cặp giống (read_id, SharedMer) giống với đầu vào. Sau khi hàm Map kết thúc, tất cả các bản ghi SharedMer từ chuỗi đọc đã cho sẽ được nhóm nội bộ với nhau trong pha Reduce

Reduce: với mối danh sách SharedMer đầu tiên được sắp xếp bằng cách đọc vị trí, và các mer phù hợp được đặt (collasces) vào các hạt giống. Hạt giống cuối cùng chính xác được nối tất cả lại thành một chuỗi lớn nhấ có độ dài tối thiểu là s bp. Đầu ra là các cặp (read_id, ShareSeed) trong đó SharedSeed là một cặp bao gồm (read_position, seed_length, ref_id, target_position, read_left_flank, read_right_flank, ref_left_flank, ref_right_flank).

6.3.3. ExtendReduce: mở rộng các hạt giống

Cái vòng MapReduce mở rộng các hạt giống xắp hàng (alignment) vào trong một xắp hàng không chính xác sử dụng thuật toán k-dfference Landau-Vishkin.

Map: Đối với mỗi SharedSeed, đoạn mã cố gắng mở rộng các hạt giống giống nhau và nối các xắp hàng với nhiều nhất k-difference. Nếu như liên kết tồn tại, đầu ra là cặp (read_id, AlignmentInfo), trong đó AlignmentInfo là một cặp (ref_id, ref_align_start, ref_align_end, num_differences). Sau khi tất cả các hàm map hoàn thành, Hadoop nhóm tất cả các AlignmentInfor mà giống read cho hàm reduce.

Reduce: Các hàm reduce lọc các xắp hàng trùng lặp, vì chúng có thể chứa nhiều hạt giống trong cùng một xắp hàng (alignment). Với mỗi read, đầu tiên sắp xếp các bản nghi Alignment bằng trường ref_align_start, và sau đó đầu ra duy nhất là cặp (read_id, AlignmentInfo) mà khác nhau trường ref_align_start.

Đầu ra của ExtenReduce là một file chứa tất cả các xắp hàng mà mỗi read với k-difference. File này được copy vào trong HDFS thành một hệ thống file định kì, hoặc tập tin HDFS có thể được xử lý với các công cụ báo cáo đi kèm.

Tài liệu tham khảo

[1]. Michael C. Schatz - *BlastReduce: High Performance Short Read Mapping with MapReduce*

[2]. Martin Tompa - *Biological Sequence Analysis*

[3]. Stephen F. Altschul', Warren Gish', Webb Miller Eugene W. Myers3 and David J. Lipmanl - *Basic Local Alignment Search Tool*

[4]. eTutorials.org - *Basic local alignment search tool (blast)*

[5]. Rodrigo C´esar de Castro Miranda1, Mauricio Ayala-Rinc´on1, and Leon Solon1 - *Modifications of the Landau-Vishkin Algorithm Computing Longest Common Extensions via Suffix Arrays and Efficient RMQ computations*

[6]. Ricardo Baeza-Yates and Gaston H. Gonnet - *A New Approach to String searching*