CSCI4180 Tutorial-4

Assignment 1 Review (Part 2)

REN, Yanjing
yjren22@cse.cuhk.edu.hk

Oct. 5, 2022

Assignment 1

- Part 1 - Getting Started on Hadoop
  - Configure VMs and install Hadoop
  - Run WordCount program with provided two datasets
- Part 2, 3, 4 - Implementations
  - Word Length Count
  - *N*-gram Initial Count
  - Counting *N*-gram Initial Relative Frequencies

Part 1: Configure Hadoop

- Failed to start Hadoop / Yarn
  - Check “/etc/hosts”, “/etc/hostnames” to see whether the IPs, aliases and bindings are correct
    - Remove line with “127.0.0.1”/“127.0.1.1” for fully-distributed mode
  - Check Java installation
  - Check Hadoop configurations
    - Fully distributed mode sample configs: Tutorial 2
    - Pseudo distributed mode sample configs: Assignment-1

Part 1: Configure Hadoop

- Format NameNode
  - You only need to format NameNode **once** after Hadoop installation
    - You can start/stop Hadoop **many times**
  - If you really need to format NameNode again
    - Stop HDFS and yarn
    - On **each VM**, remove the temporary directories and logs:

```
rm -rf $HADOOP_HOME/tmp
rm -rf $HADOOP_HOME/logs
mkdir -p $HADOOP_HOME/logs
```

    - Format NameNode
    - Start HDFS and yarn

Part 2: Word Length Count

- Requirements
  - You are **not** allowed to implement in-mapper combining
    - Check the lecture notes for the definitions
  - Notes
    - A word may include characters, **numbers and punctuations**
    - “dog.”: length is 4, but not 3

Part 3: *N*-gram Initial Count

- FAQ
  - How to pass N as an argument into your program?
  - How to get N-gram Initials?
  - How to speed up your program?
    - Strongly encouraged, but not compulsory

Part 3: *N*-gram Initial Count

- How to pass N as an argument into your program?
  - Initialize Configuration in main function
  - Get N from Configuration class in Mapper
  - Do **NOT** modify Hadoop configuration files to add N
- Example

```
Configuration conf = new Configuration();
conf.set("N", args[2]); // set N
```

```
Configuration conf = context.getConfiguration();
N = Integer.parseInt(conf.get("N")); // get N of N-gram
```

Part 3: *N*-gram Initial Count

- How to get N-gram Initials?
  - Hint: breakdown the problem
  - How to get N-gram
    - Hint: Maintain a list of words
      - When the list size goes to N, you get an N-gram
      - Think about how you get the next N-gram
  - How to get initials from an N-gram
    - String. charAt(index)

Part 3: *N*-gram Initial Count

- How to speed up your program?
- Approach 1 - Pairs (Lec.4 pp. 25 - 26)
  - Each mapper emits intermediate < N-gram Initial, count >.
  - Each reducer sums up all counts associated with the same N-gram Initials.
- Pros:
  - Easy to understand and implement
- Cons:
  - Generate many key-value pairs, and even redundant key-value pairs without in-mapper combining.

Part 3: *N*-gram Initial Count

- Approach 2 - Stripes (Lec.4 pp. 27 - 31)

Example:

(a, b) → 1
(a, c) → 2
(a, d) → 5
(a, e) → 3
(a, f) → 2

a → { b: 1, c: 2, d: 5, e: 3, f: 2 }

- You can use **MapWritable** to be the value of map-outputs
  - MapWritable is to convert a Map data structure (e.g., HashMap) into a serialized format (e.g., {b:1, c:2, d:5, e:3, f:2})
    - It is one of the IO types. It is different from the Map function in MapReduce
    - MapWritable API

Part 3: *N*-gram Initial Count

- Approach 2 - Stripes (Lec.4 pp. 27 - 31)
  - Mapper
    - Stores co-occurrence information in an associative array for each particular word;
    - Emits intermediate key-value pairs with words as keys and corresponding associative arrays as values.
  - Reducer
    - Aggregates the counts in associative arrays.

$$
\begin{array}{ll}
 & a \rightarrow \{\, b\colon 1,\ \ d\colon 5,\ e\colon 3 \,\} \\
+ & a \rightarrow \{\, b\colon 1,\ c\colon 2,\ d\colon 2,\ \ f\colon 2 \,\} \\
\hline
 & a \rightarrow \{\, b\colon 2,\ c\colon 2,\ d\colon 7,\ e\colon 3,\ f\colon 2 \,\}
\end{array}
$$

Part 3: *N*-gram Initial Count

- Approach 2 - Stripes (Lec.4 pp. 27 - 31)
  - Pros:
    - Generate fewer key-value pairs
    - Make better use of combiners
  - Cons:
    - Memory size of associative arrays in mappers could be huge
- **In-mapper combining** applies to both Pairs and Stripes

Part 4: N-gram Initials Relative Frequencies

- Requirements
  - Output
    - **ONLY** output sequences with frequency ≥ θ
    - Both float and double data types are acceptable

Part 4: N-gram Initials Relative Frequencies

- FAQ
  - How to pass N and θ as arguments into your program?
  - How to get N-gram Initials relative frequencies?
  - How to speed up your program?
    - Strongly encouraged, but not compulsory

Part 4: N-gram Initials Relative Frequencies

- How to pass N and θ as arguments into your program?
  - Similar to Part 3
  - Get theta in Reducer

Part 4: N-gram Initials Relative Frequencies

- Approach 1 - Pairs (Lec.4 pp. 34-36)
  - Make sure the same word goes to the same reducer
    - For example, use **partitioner**
    - It controls which of the reduce tasks the intermediate key is sent to for reduction
    - The key is used to derive the partition, typically by a hash function
    - Refer to the API
  - Make sure (a,*) comes first, before individual counts
    - You can implement **WritableComparable** to achieve order inversion.
    - Refer to **WritableComparable**'s API

Part 4: N-gram Initials Relative Frequencies

- Approach 2 - Stripes (Lec.4 p. 33)
  - It is more straightforward than pairs approach.
  - Sum all the counts in the associative array for each word.
  - For example,
    - Add “*” into associative arrays to record the total values
    - a → {b:1, c:2, d:5, e:3, f:2, *:13}
  - Drawbacks:
    - Memory size of associative arrays in mappers could be huge

Bonus

- Bonus(5%, optional)
  - **(2%)** Configure Hadoop in ***fully distributed mode*** to run Part 4's program
  - **(3%)** The top 3 groups with the smallest running time in Part 4 will receive the bonus marks
    - **Prerequisite**: Pass Part 2, 3, 4

Bonus

- Approaches
  - Find the bottleneck via ***hprof***: Link
  - Optimize the program by modifying the algorithm or designing more efficient data structures
  - Configure some parameters of Hadoop
    - **Note:** you can only do this in your program dynamically (see **Lec. 3 p. 39**)
    - **DO NOT** modify the Hadoop configuration files

Remarks

- Dataset
  - A problem when using the Shakespeare data caused by the directory structures
    - Reason: MapReduce does **NOT** support write folders with sub-folders recursively
    - Solution
      - Move all the .txt files to the top directory and write

Remarks

- Errors in running MapReduce with Shakespeare dataset
  - Container launch failed
    - Solution: add the following configurations in mapred-site.xml

```
<property>
        <name>mapreduce.job.maps</name>
        <value>2</value>
</property>

<property>
        <name>yarn.app.mapreduce.am.containerlauncher.threadpool-initial-size</name>
        <value>5</value>
</property>
```

Remarks

- Submission
  - You must submit the following files and strictly follow the file names format
    - WordLengthCount.java
    - NgramInitialCount.java
    - NgramInitialRF.java
  - You may submit additional files if you **really need to**. But in this case, please attach your customized **Makefile**.
  - Package the above files to asgn1-*SID*.tar.gz and submit it to Blackboard system
  - We allow late submissions until the start of the demo time (which is on the next day), but there will be a penalty of **20 marks**.
  - **NO** late submissions are allowed after the demo starts, and **zero marks** will be given.

Remarks

- Demo
  - For Part 2-4, your programs will be tested on TAs' platforms.
  - Tune Hadoop configuration dynamically inside the code (see **Lec. 3 p. 39**) if necessary.
  - You are **NOT** allowed to modify any configuration files during demo, we use the default configurations to test your program (see Tutorial 2)
  - **NO** code change is allowed during the demo.
    - We only allow slight modifications of the code (e.g., changing **1 - 2** lines of code), with a penalty of **10 marks** for each change.

Remarks

- Suggestions
  - To verify your program's correctness, use data you are familiar with to generate sample output
  - Test your program with large datasets
    - Find potential problems in memory
    - Check performance
  - Do assignments as early as possible!
  - **Backup** your code/data on VM instances
  - **Don't cheat!** We will figure out the cases easily
  - Ask questions and discuss on **Piazza**

Q & A