Final Project Proposal - 532

Anthony Zalev, Matt Zambetti, Varsha Jawahar, Xinzhi Liang

April 14, 2023

1 Overview

For the final project, we wish to expand the data processing tasks in Homework 3 and explore ways to improve the performance. Specifically, we wish to compare the performance of using PySpark RDD, Pyspark DF, native Python, and native SQL on the word counting task we implemented in homework 3 and explore the reasoning behind the performance difference. We will compare how these each of these tools run on a single machine vs in a distributed system using Unity clusters.

The goal of this project is to better understand how each of the four configurations approach data processing, and what those differences result in - run-time, where we may see errors, etc.

2 Questions and Clarifications

Q) How many nodes do we need?
A) At a minimum any number strictly greater than 1. Realistically we would want more so we overcome any overhead involved. Our findings should be significant with less than 10 nodes. We would like to achieve this with the help of the UMass Unity Cluster.
Q) How are we going to chunk the dataset?
A) The dataset is a text file. A coordinator will chunk the texts into blocks of a predetermined size (divided equally among the word-counting workers). Blocks will be sent to worker methods/threads/machines before getting a result returned, which the coordinator will be responsible for aggregating.

3 Dataset

We will be performing the word counting task on The Complete Work of William Shakespeare provided by Project Gutenberg. The data will first be pre-processed to remove all punctuation and lowercase all letters in the document (for better readable results). Below we provided a portion of the post-processed document.

```
the sonnets
by william shakespeare
```

```
                    1
from fairest creatures we desire increase
that thereby beautys rose might never die
but as the riper should by time decease
his tender heir might bear his memory
but thou contracted to thine own bright eyes
feedst thy lights flame with selfsubstantial fuel
making a famine where abundance lies
thy self thy foe to thy sweet self too cruel
thou that art now the worlds fresh ornament
and only herald to the gaudy spring
within thine own bud buriest thy content
and tender churl makst waste in niggarding
  pity the world or else this glutton be
  to eat the worlds due by the grave and thee
```

4 Methods

As the input data is a document, each tool must first read the document and then starts the word-counting process. A time stamp will be collected for the reading time of the document using each tool, this is to make sure the reading time does not drastically affect the total run-time of each method.

For a single machine, the implementation would be standard (with respect to each tool). We would implement and run the code on a single machine while keeping track of the total run time.

For a distributed system, we plan on using RPCs or sockets to connect multiple nodes. A coordinator node and a worker node. Workers would each have their own thread pool similar to part 2. The coordinator would give large file chunks divided equally to word readers who would give blocks to word-count nodes. The number of chunks would depend on the number of Unity clusters we are able to use and will be included in our report.

5 Milestones

5.1 Implementations

The single machine and distributed system implementation will be completed for each configuration (PySpark RDD, PySpark DF, native Python, and native SQL) to calculate the total word count of a document.
Below is a pseudocode example of the implementation, with *word_count* be some function that returns the count of words of a given string/block of data.

$D \leftarrow read(\text{"}data.txt\text{"})$
$agg \leftarrow 0$
for $block \in D$ **do**

$agg \leftarrow agg + word\_count(block)$
end for
return agg

5.2 Data collecting

1. Collect run-time data for each implementation.
2. Collect other metrics of interest (ex: CPU usage).

5.3 Report

1. Produce run-time comparison plots/tables to support our findings.
2. Research reasons behind the differences in configurations and include them in the final report (will be cited)

6 Final Goals

As previously mentioned in the Overview, our main goal is to explore data processing in various domains. Below we have listed our Final Project goals along with the timeline which should encompass everything from the implementation of our scripts, to how we will compare the configurations and summarize our findings.

1. 04/18 Have sets of scripts that run a single node. The sets are being done in a PySpark implementation, in the various forms of SQL, and native Python.
2. 04/25 Request and set up Unity Clusters
3. 05/01 Implement distributed versions of the Pyspark, SQL, and native Python scripts on the Unity cluster.
4. 05/10 Create graphs and charts that compare the run-time and other metrics of interest (ex: CPU usage) to accurately compare which implementation processes the data most efficiently.
5. 05/14 Generate the final report as described below.

Our Final report will include the following sections:

1. A simple overview of our goal, and some hypotheses (e.g. native SQL would be faster than PySpark DF but slower than PySpark RDD because ...)
2. Scripts that were used (the major scripts that performed the outlined tasks)
3. Diagrams displaying data collected and some discussions about each
   (a) run-time comparison across all four tools

(b) run-time comparison of each tool on distributed machines vs on single node

(c) other metrics of interest compared across appropriate tools on a single machine (e.g. CPU usage)

(d) an overview of the data chunking process for the distributed system

4. Final conclusion including discussion on our hypotheses and any further knowledge from research.