Dataset Properties

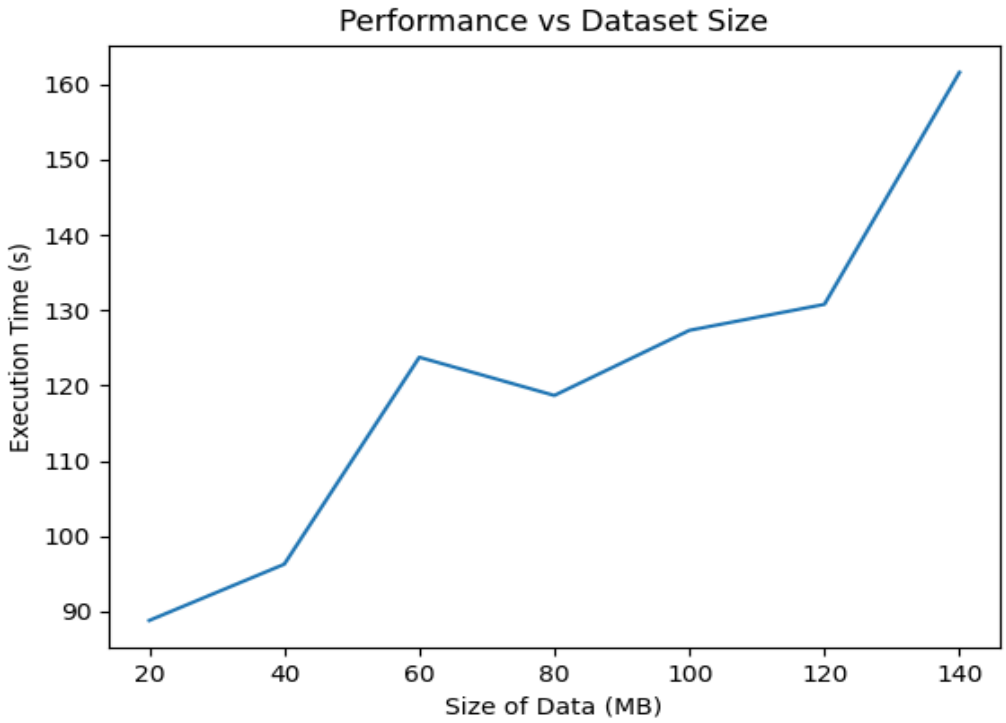
Our dataset was initially 3GB, a json file consisting of data regarding scientific research papers uploaded to arXiv.org. The title, authors, paper submitter, paper abstract, and the date the paper was updated were some of the fields included. Initially, we attempted to load all 3GB of data onto the Pi cluster, but ran into storage problems. To fix this, we made sure that our code correctly compiled on all 3GB of data on my laptop, before filtering the json file to consist only of the title column, which reduced the file size to 200MB.

Problem Implemented

The problem we implemented was calculating the term frequency-inverse document frequency (TF-IDF) of every word across all titles in our dataset. A low TF-IDF score for a word indicates a frequent usage across documents, whereas a high score indicates infrequent usages. To start, we planned to calculate the TF-IDF on the entry abstracts, where a high score for a word would suggest that it is frequent, thus important, within the document, but rare across all documents. Essentially, finding high scoring words would potentially reveal unique research topics outlined within paper abstracts. However, the abstracts were too long to be scalable to a large dataset, as a vocabulary needs to be created across all documents (this caused out of memory errors). Instead, we calculated the TF-IDF across titles, so the vocabulary is significantly smaller and more manageable. This is still informative as to what unique concepts might be–in titles, every word in the title typically appears only once, so the TF-IDF is simplified to the IDF. Again, a low score indicates frequent usages, which may signal frequently covered research topics/ strategies, and a high score could indicate unique or novel concepts.

Results

We chose to evaluate execution time as the performance measure against the size of the data used. The following graph shows our results using 3 Pis, 4 cores per Pi.



The above graph shows what can be expected when processing more data–the time increases for larger files. It is interesting that the line is not linear, with a spike at 60MB. The line may become more steady with more repetitions of the same parameters (averaging the execution time for each file size).

Challenges Faced

Storage

On multiple occasions, we ran out of disk space on the pis from installing essential libraries, to copying the data file, to running spark submissions. This challenge was detrimental to our progress, and in order to overcome it, we had to take intensive measures to allow for more storage space. On all of the Pis, we removed all old and unused MPI directories and files. We also removed large applications such as Chromium and Firefox browsers on worker nodes. Quite late in the project we were able to acquire 4 flash drives that could be added to each Pi to allow for maximal data storage. With that, we could secure-copy all the files and libraries we needed onto the flash drives, which eliminated storage errors. That being said, there were more permissions issues that came about as the drives were located in a root-level path.

Wifi

In order to transfer the script and dataset onto the Pis, we planned to ssh into the Pis and secure-copy the files. However, ssh-ing from a remote device was blocked by Stetson's firewall. We needed a static ip address on the Stetson guest wifi in order to remotely connect to

the master Pi. Once we had this, we were able to ssh in to the master using this ip address instead.

Master-Worker Interactions

A critical part of the process of using a cluster computer is getting the nodes to communicate. After a significant amount of time spent installing Spark, setting up SSH keys, and tweaking .conf and .env files, we were able to get the start-master.sh and start-workers.sh to run and the master was able to host the Spark UI over the web. However, the workers did not show up in the UI, and it was determined that the master could not see the worker nodes. A simple solution was found after much investigation: change the /etc file to not have the localhost IP address (127.0.0.1) and then add all of the Pi's addresses. Once this solution was implemented, the workers appeared on the master Spark UI as "Alive" with their statistics listed on the side.

Submitting Spark Job

Submitting Spark jobs perhaps was one of the most prominent issues we ran into. From running out of data storage, to permissions denied, refused connections and disconnections, the submitting of jobs proved to be a battle. We formulated the following command to submit an application successfully:

/opt/spark/bin ./spark-submit --master spark://172.18.2.243:7077 –conf spark.dynamicAllocation.enabled=false –num-executors 3 –executor-cores 4 /media/raspberries/EXTERNDV/Pi_Files/idf.py

This command, in combination with the python application code, required tuning over multiple days to finally get to submit and successfully complete a job using the cluster computer.

Time Restraints

Time was an outstanding issue in this project. It took intensive effort to get great results with such a large amount of data. Over 24 hours were invested in the project, and progress was slow. Despite the high stress scenario that was created, the project was a strong learning experience in coding, using linux command line and operating system, and using Spark.

Conclusion

Our exploration of performance evaluation, particularly focusing on execution time as the primary measure, sheds light on the intricacies and challenges of processing large datasets on a distributed computing environment. Leveraging the power of Raspberry Pi clusters, we embarked on a journey to analyze term frequency-inverse document frequency (TF-IDF) across scientific research paper titles, aiming to unveil unique research topics and patterns.

Despite encountering formidable obstacles, including storage limitations, network constraints, and intricate master-worker interactions, our perseverance and strategic problem-solving allowed us to navigate through these challenges. By meticulously optimizing

storage utilization, overcoming network barriers, and fine-tuning Spark job submissions, we gradually transformed setbacks into stepping stones toward success.

Our results, depicted graphically, demonstrate the impact of data size on execution time, providing valuable insights into the scalability and efficiency of our approach. Through rigorous analysis, we unearthed the intricate relationship between dataset properties and computational performance, illuminating avenues for further optimization and refinement.

As we reflect on the challenges faced and lessons learned, it becomes evident that this endeavor served not only as a technical exploration but also as a profound learning experience. The amalgamation of coding proficiency, Linux command-line expertise, and Spark utilization culminated in a holistic understanding of distributed computing paradigms.

In essence, while the journey was fraught with obstacles and time constraints, the culmination of our efforts underscores the resilience and adaptability essential in tackling complex computational endeavors. As we look toward the horizon, armed with newfound knowledge and insights, we stand poised to embark on future endeavors with confidence and conviction.