G14 - Manan Patel(manandp2) and Mihir Shah(mihirs2)

Design

Our RainStorm System is split into 2 parts: a leader and workers. The leader handles the routing, task scheduling, and hosts the global resource manager. The workers directly manage each task they have and forward the results to each task's next stage.

The leader starts with the CLI for a RainStorm application, kill_task, or list_tasks. When receiving a RainStorm application, the input will be parsed into a struct. The leader then completes `<Nstages> x <Ntasks_per_stage>` iterations, distributing tasks in a round robin fashion. The round robin references a stored pointer to distribute future tasks evenly. After the initial task distribution is completed, the leader creates and caches RPC client connections with each user. Afterwards, the leader starts to asynchronously listen for lines from the final stage. When receiving tuples, it will output the lines to both a local output file for tailing and the HyDFS Destination File. There is also an asynchronous reader and sender that reads lines from the source file and sends them to their respective tasks' workers based on hashing. The leader also listens for failures from the workers and respins up the task on the next worker based on the round robin pointer from before. Once the leader knows that all stages have completed their execution and it has received everything, it will end the application and wait for a new one.

The global resource manager on the leader listens for rate updates from tasks, and auto-scales accordingly. It receives rate updates every second and logs them. If auto scaling is enabled and a task's rate goes outside a low or high watermark, RM will remove or add a task in that stage, respectively. The new task is once again added using the incrementing round robin pointer to ensure even distribution.

The worker starts and introduces itself to the leader, and then waits for an application to start. Once an application starts, the worker initializes data that is global for an application and begins to listen for new tasks. Each task is started as a new process, and its stdin and stdout are connected to the worker through pipes. A channel is used to collect all of the outputs and is consumed from to send tuples to the next stage. To send tuples, a cached tcp connection is used, and acks are sent back on the same connection. An ack is only sent after writing to the task pipe. To prevent duplication, an in memory map is used alongside the HyDFS log files. The map is used because going through the entire log file for deduplication would be very slow. When the leader tells the worker a given stage is finished, it closes the pipes, and once the last output of a task is read, it reports to the leader that it is completed. Then when all the tasks for a stage are done, the leader can tell the workers to close the pipes. When recovering from failure, the worker looks at the log file and re adds all tuples that were received but not processed.

Past MP Use

We used MP3 as the underlying distributed file system to store task logs for failure recovery. MP2 was used within MP3 as a failure detector if a node went down.

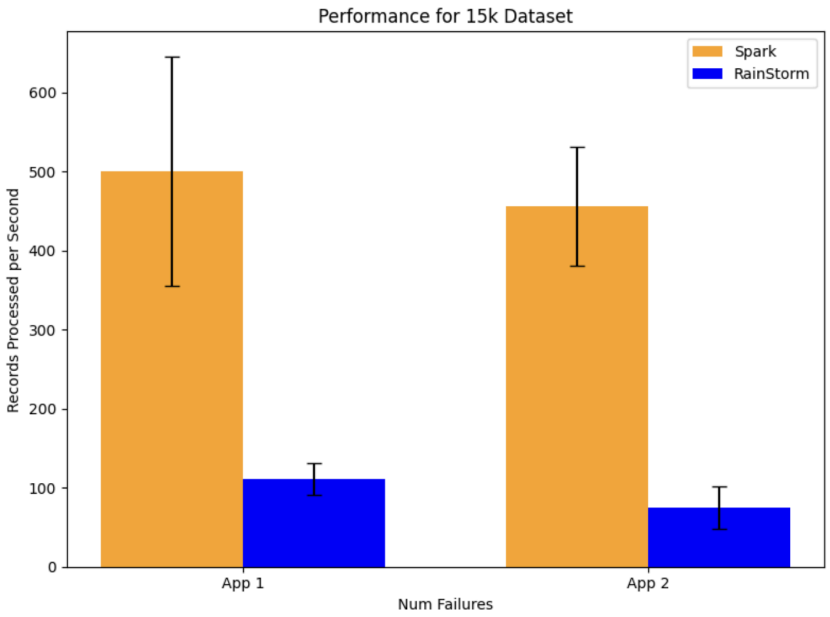
LLM Use

G14 - Manan Patel(manandp2) and Mihir Shah(mihirs2)

We used LLMs mainly to help us debug. Whenever we implemented new functionality in our program, we would test it. Since it wouldn't work the first time, we would start to debug using print statements. If we got stuck and could not pinpoint the cause of the issue, we would ask an LLM to see if it could tell us where to look. LLMs were also used to generate code based on what we had if there was a bug. Chats are stored in the LLM directory.

Measurments

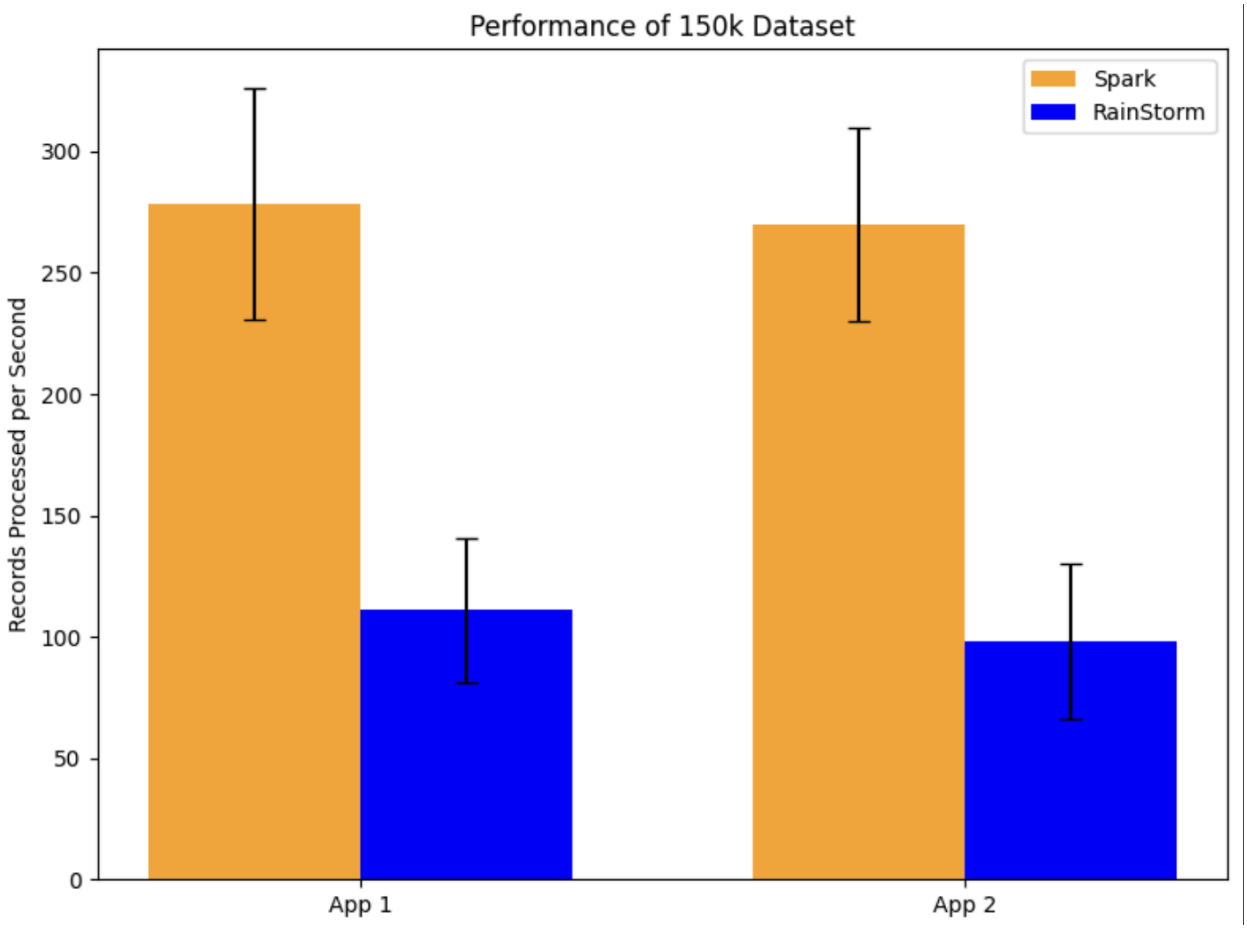
Dataset 1:
https://gis-cityofchampaign.opendata.arcgis.com/datasets/f5aaae43ed6642cc944117f4f4221adb_37/explore?location=40.112368%2C-88.279450%2C12.70&showTable=sdtrue



Spark processed faster, which means the bandwidth is higher since it is moving more tuples at once. However, the standard deviation of RainStorm was lower. This is surprising because we would have expected that Spark would have a much tighter standard deviation.

G14 - Manan Patel(manandp2) and Mihir Shah(mihirs2)

Dataset 2: https://www.kaggle.com/datasets/edenbd/150k-lyrics-labeled-with-spotify-valence



The results are similar for the bigger dataset as well. It is expected that the standard deviations and tuple rate are similar across both applications.