CS 425 Distributed Systems: RainStorm Stream-Processing Framework

Group 71: Jiayuan Hong (jh79), Shiyang Zhao (sz89)

Design

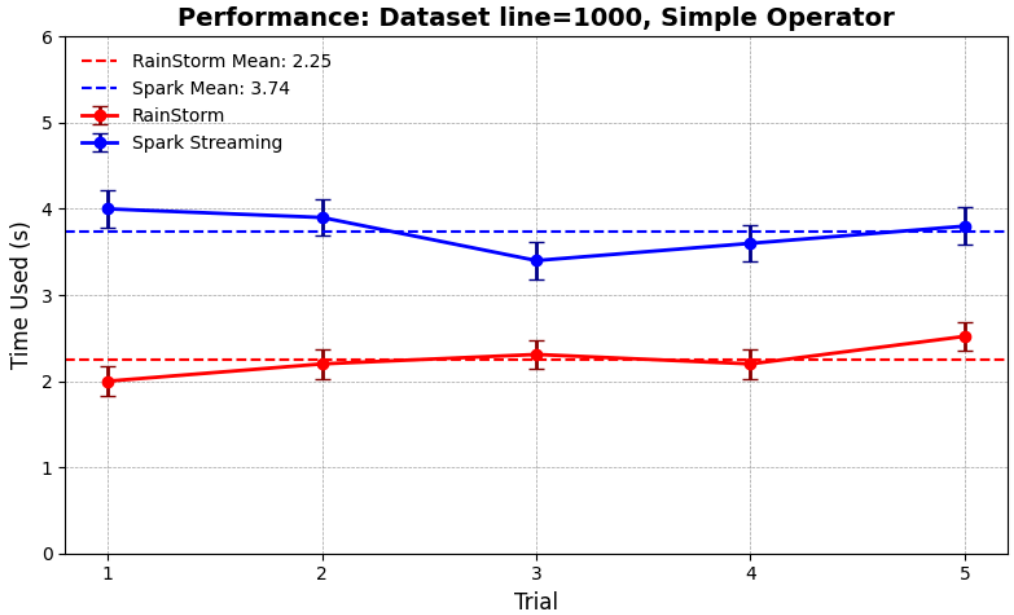
We implemented a distributed stream-processing framework called Rainstorm for real-time data processing. The RainStorm consists of three parts. Leader-Worker Model, a fault-tolerant leader coordinates task scheduling, failure handling, and exactly-once delivery semantics. Workers execute user-defined operators, process streams, and manage logs for fault recovery. Here are the core features:

- Stream Processing: Real-time transformations without barriers between stages.
- Operators:
  - Transform: Applies a user-defined function.
  - FilteredTransform: Filters and transforms records.
  - AggregateByKey: Maintains a running state for aggregation.
- Fault Tolerance:
  - Persistent logging in HyDFS ensures duplicate detection.
  - Exactly-once semantics achieved through unique IDs and ack.
- Performance Optimizations:
  - Batch processing for logs and HyDFS writes.
  - Hash partitioning for load balancing.

Performance

We selected 2 dataset, one contains 1000 lines and the other contains 3000 lines. We tested simple (app-1) and complex (app-2) on Spark and RainStorm and plot 4 figures.

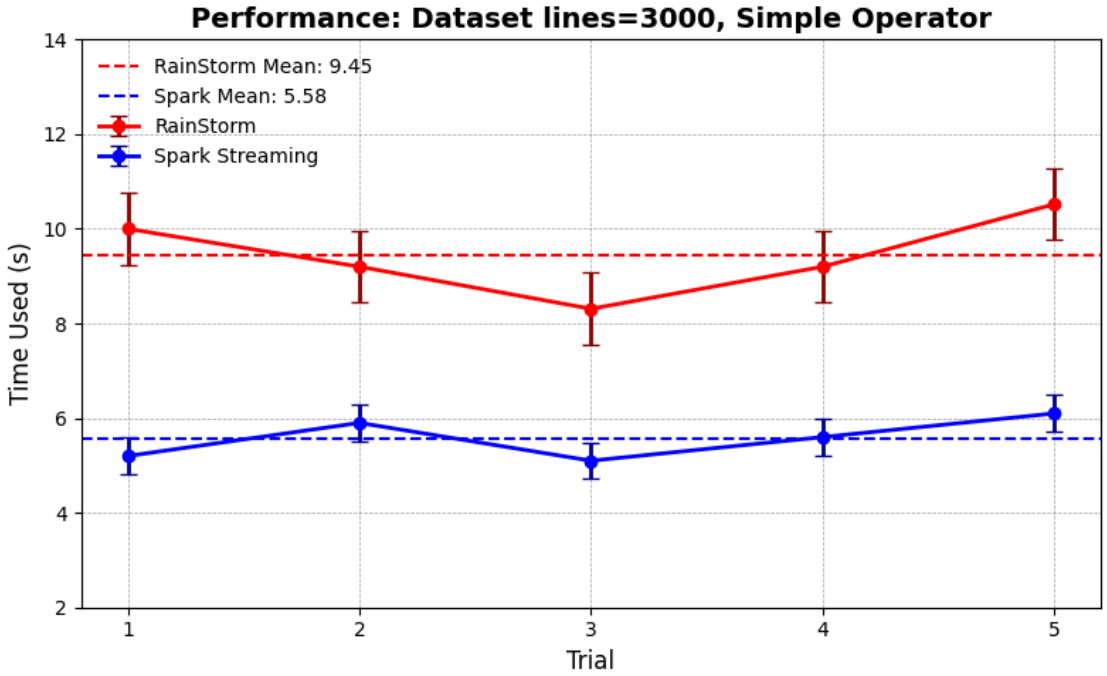
1. Spark vs RainStorm with 1000 lines, using a simple operator:



We can see that when the dataset is relatively small, our framework exceeds the performance of Spark. We can see that our framework is 1.6x faster than the Spark.

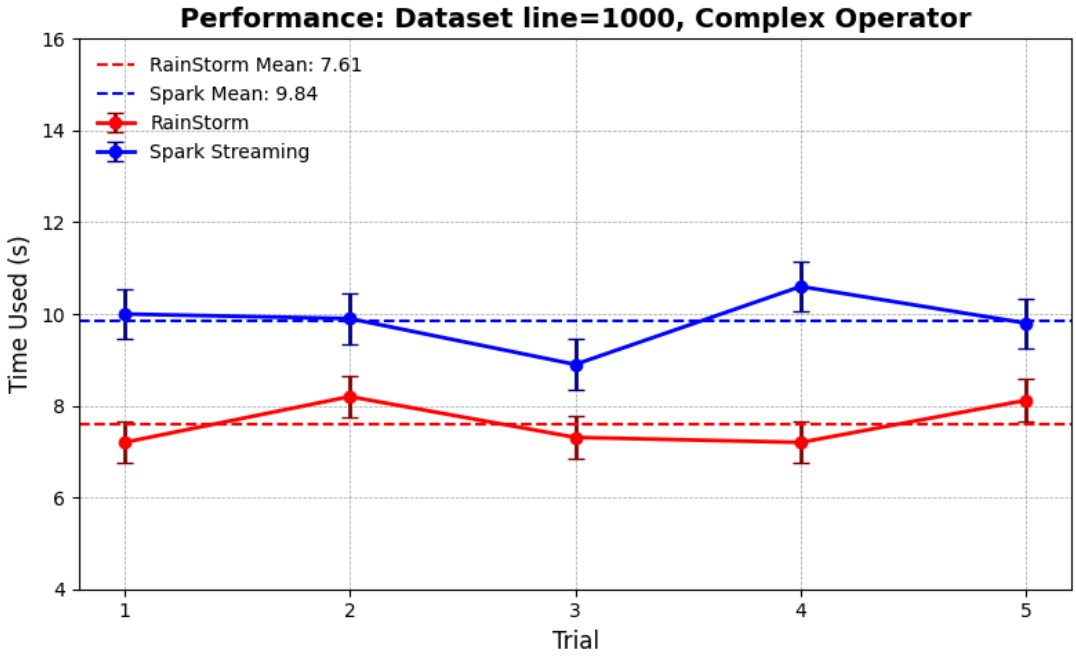
This is because our framework is very lightweight and optimized for handling smaller datasets, allowing it to execute operations efficiently without the overhead that larger frameworks like Spark introduce. In contrast, Spark is designed to handle large-scale distributed data processing. While this makes it powerful for processing massive datasets, the associated overhead can result in slower performance for smaller datasets.

2. Spark vs RainStorm with 3000 lines, using simple operator:



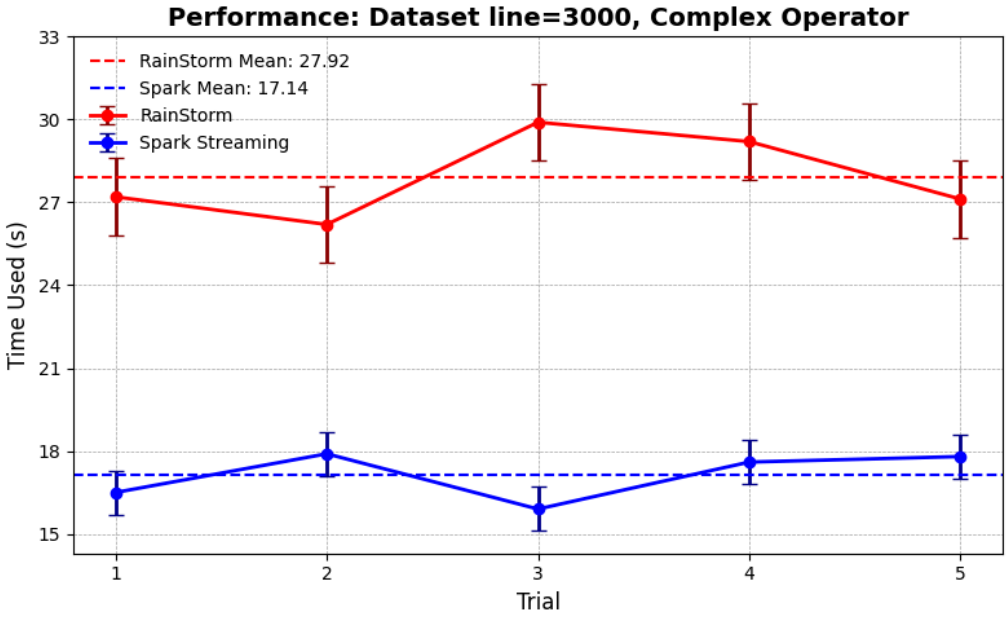
When the dataset size increases significantly, Spark outperforms RainStorm, we can see that Spark is 1.7x faster than the RainStorm. This shows that Spark is fully optimized for handling large-scale distributed data processing. Its sophisticated scheduling algorithms, efficient data partitioning, and robust fault-tolerance mechanisms allow it to effectively manage the increased workload, maintaining scalability and consistent performance even as the dataset grows.
In contrast, RainStorm, while highly efficient for smaller datasets, begins to experience performance degradation with larger datasets. This is due to the overhead introduced by its fault-tolerance mechanisms, such as exactly-once delivery and state recovery, which become more resource-intensive as the dataset size increases. Additionally, RainStorm's lightweight design, which gives it an edge with smaller datasets, limits its ability to compete with Spark's distributed processing capabilities at scale.

3. Spark vs RainStorm with 1000 lines, using complex operator:



When using complex operators, we see that the time increases around 2.5-3x for both Spark and RainStorm compared to simple operators. This increase is because of additional computational workload and data shuffling required for executing multiple stages (Transform → FilteredTransform → AggregateByKey).

Spark vs RainStorm with 3000 lines, using complex operator



For datasets with 3000 lines and a complex operator, Spark performance exceeds RainStorm more because of its fully optimized design for large-scale data processing.