HOMEWORK – 3

Data Intensive Computing

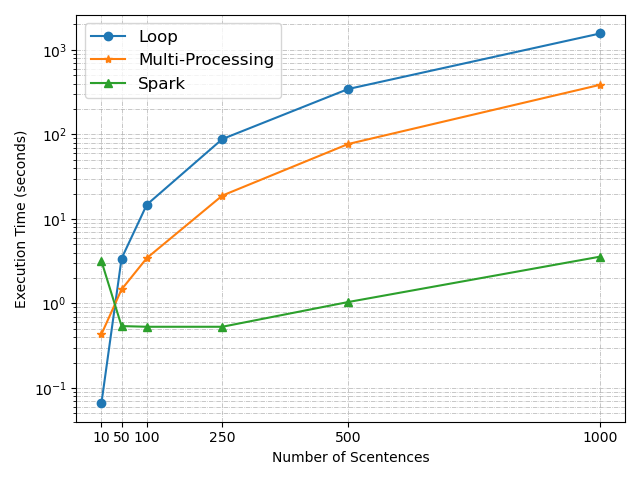
Nikhil Saji

50592329

# Task 1.1

Time taken for each set of sentences

* 10 Sentences:
  + Spark – 3.18 seconds
  + Multi-process – 0.432 seconds
  + For-loop – 0.066 seconds
* 50 Sentences:
  + Spark – 0.54 seconds
  + Multi-process – 1.466 seconds
  + For-loop – 3.402 seconds
* 100 Sentences:
  + Spark – 0.53 seconds
  + Multi-process – 3.438 seconds
  + For-loop – 14.735 seconds
* 250 Sentences:
  + Spark – 0.53 seconds
  + Multi-process – 18.900 seconds
  + For-loop – 88.130 seconds
* 500 Sentences:
  + Spark – 1.04 seconds
  + Multi-process – 77.052 seconds
  + For-loop – 344.748 seconds
* 1000 Sentences:
  + Spark – 3.58 seconds
  + Multi-process – 386.267 seconds
  + For-loop – 1564.122 seconds



Inferences –

* For loop performs well when number of sentences are less and it execution time worsen exponentially when the number of sentences are high.
* Multi-process performs well compared to a single loop as it can parallelize tasks effectively. But the execution time increases when the sentences count is high,
* Spark performs best among all as it can finish the task in just matter of seconds which shows it is highly optimized for handling distributed computations, maintaining consistent performance even with larger dataset.

# Task 1.2

Time taken for n\_inputs

* 1000 Inputs:
  + Spark – 0.49 seconds
  + Non-Spark – 0.18 seconds
* 5000 Inputs:
  + Spark – 0.35 seconds
  + Non-Spark – 0.96 seconds
* 10000 Inputs:
  + Spark – 0.31 seconds
  + Non-Spark – 2.8 seconds
* 50000 Inputs:
  + Spark – 0.38 seconds
  + Non-Spark – 350.29 seconds
* 100000 Inputs:
  + Spark – 0.4 seconds
  + Non-Spark – 847.21 seconds

A graph with a line and a point

Description automatically generated

Inferences –

* For non-spark version the execution time increases dramatically as number of sentences increases.
* There is a huge jump in execution time between 10000 and 50000 sentences.
* This steep rise shows a significant scalability issue with this method for very large datasets hence it is vital we use Spark.
* As in for Spark, execution time remains rather constant regardless of number of sentences.
* This shows spark’s distributed architecture efficiently handles scaling and large-scale data processing of tasks can be done in an efficient manner.

## Task 1.3

Time taken for n birds:

* 200 Birds –
  + Spark - 0.0048 seconds
  + Non-Spark - 0.0037 seconds
* 1000 birds –
  + Spark - 0.0037 seconds
  + Non-Spark - 0.0288 seconds
* 5000 birds –
  + Spark - 0.0106 seconds
  + Non-Spark - 0.6647 seconds
* 10000 birds –
  + Spark - 0.0217 seconds
  + Non-Spark - 3.2473 seconds

A graph with blue and orange lines

Description automatically generated

Inferences –

* Both Spark and non-spark implementation performs well with a smaller number of birds but spark implementation takes a bit more time to execute.
* When the count of birds increases significantly, the execution time for non-spark increases steeply showing poor scalability whereas spark can complete the process efficiently and faster
* It demonstrates the effectiveness of distributed computing for handling larger datasets.
* Hence non-spark implementation is good for a smaller dataset but for a bigger dataset, spark would be ideal.