Assignment - 1
CS 744 - Spark
Course: Big Data Systems

Group 10: Team Members

Divy Patel (dspatel6@wisc.edu)
Smit Shah (spshah25@wisc.edu)
Sujay Chandra Shekara Sharma (schandrashe5@wisc.edu)
Venkata Abhijeeth Balabhadruni (balabhadruni@wisc.edu)

Contributions

Divy:

1. Setting up CloudLab and experiments
2. Pagerank core logic and implementation

Smit:

1. Optimisation for code flow path with less storage
2. Debugging Spark flow in workers

Sujay:

1. Spark skeleton code for small dataset
2. Experimenting with various combinations of iterations and persistence

Abhijeeth:

1. Execution path for Spark flow on a big dataset
2. Experimenting with various combinations of the partitions, and iterations

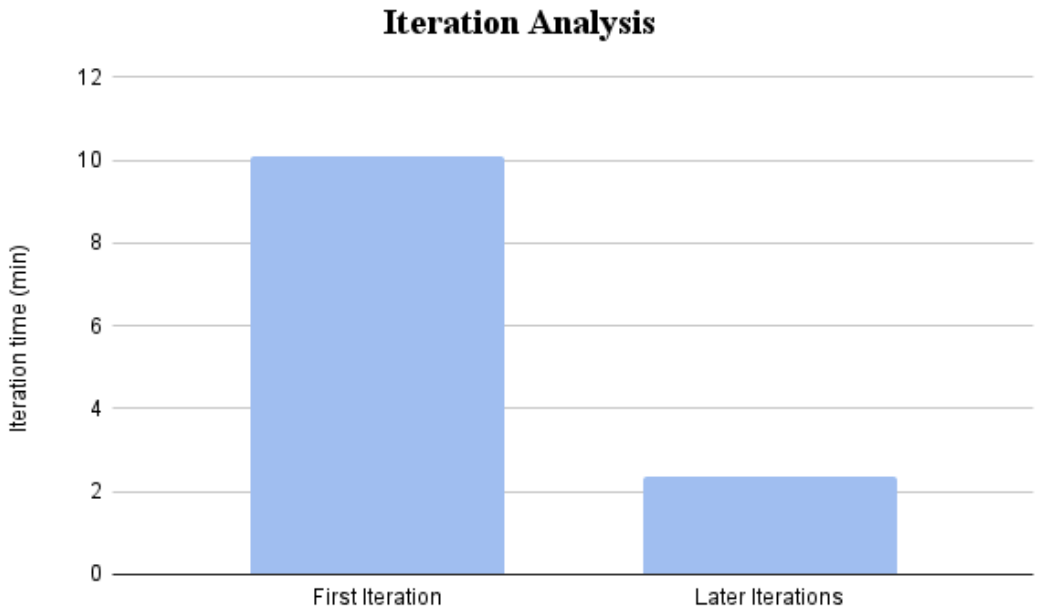
Common work done by all the members:

1. Brainstorming on the problem
2. Setting up Hadoop, Spark, and parallel-ssh on VMs

Experiment Results, Observations, and Reasoning

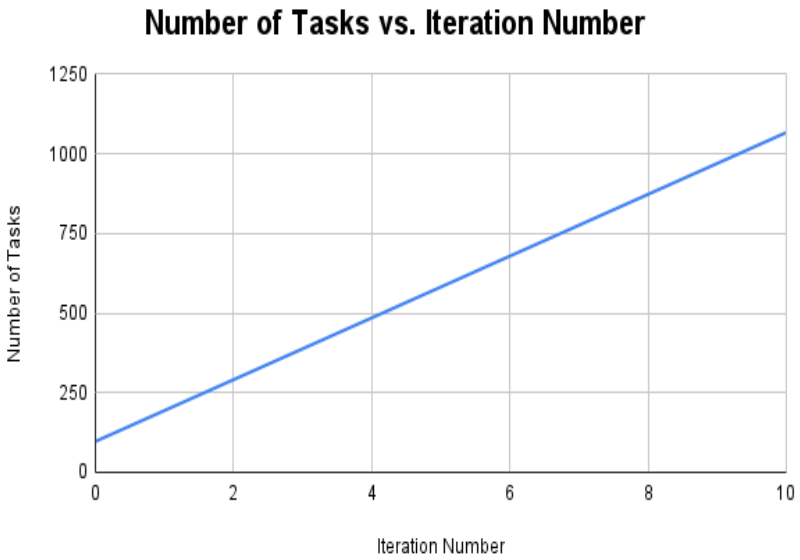
Task 1: Normal Execution

Iteration Number	Iteration Time (min)
First Iteration	10.1
Later Iterations	2.35



Total Shuffle Read: 115 GB
Total Shuffle Write: 82.6 GB
Total Duration: 33 min
Total Number of Tasks: 13774

Iteration Number	Number of Tasks
0	97
1	194
2	291
3	388
4	485
5	582
6	679
7	776
8	873
9	970
10	1067



Observations

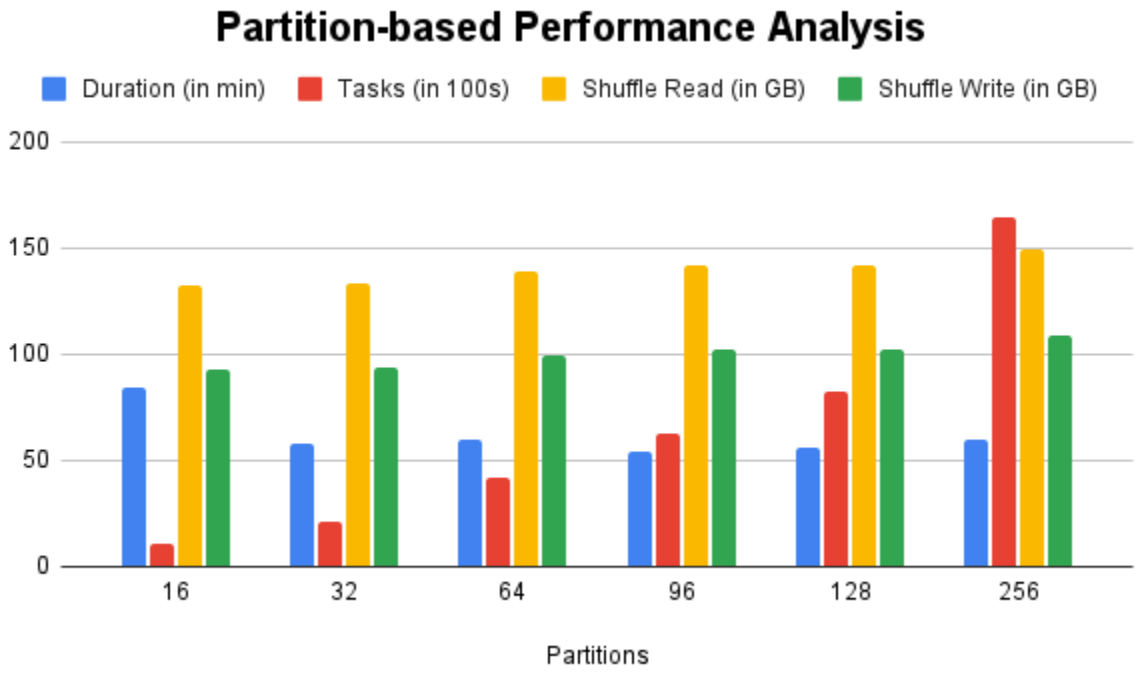
- The first iteration takes a significantly long time when compared to subsequent iterations.
- Also, we observed that the number of tasks is increasing linearly with iterations.

Reasons

- The first iteration takes longer than the rest because the data is loaded into Spark as an RDD from HDFS. Subsequent iterations just utilize the RDDs.
- Based on debugging the number of partitions after each transformation, we found that after the join operation in each iteration, the number of partitions increased.

Task 2: Impact of Number of Partitions on Execution

Partitions	Duration (in min)	Tasks (in 100s)	Shuffle Read (in GB)	Shuffle Write (in GB)
16	84	11.21	132.7	92.7
32	58	21.45	133.7	93.7
64	60	41.93	139	99.1
96	54	62.41	141.6	101.8
128	56	82.89	142.2	102.3
256	60	164.81	149.2	109.1



Observations

- As part of this task, we tried to examine the impact of partitioning the RDDs across the executors on the execution of Spark jobs.

- The Spark jobs were submitted for the varying number of partitions and were measured against the completion time, the number of tasks, and shuffle read/write.
- We saw a linear increase in the number of tasks as we increased the number of partitions for the Spark job.
- When a Spark job was configured for 16 partitions, the Spark job took ~84 minutes to complete.
- From our data points, we observed an initial decrease in completion time when we increased the number of partitions from 16 to 32.
- Shuffle read/write slowly increased with the increasing number of partitions.

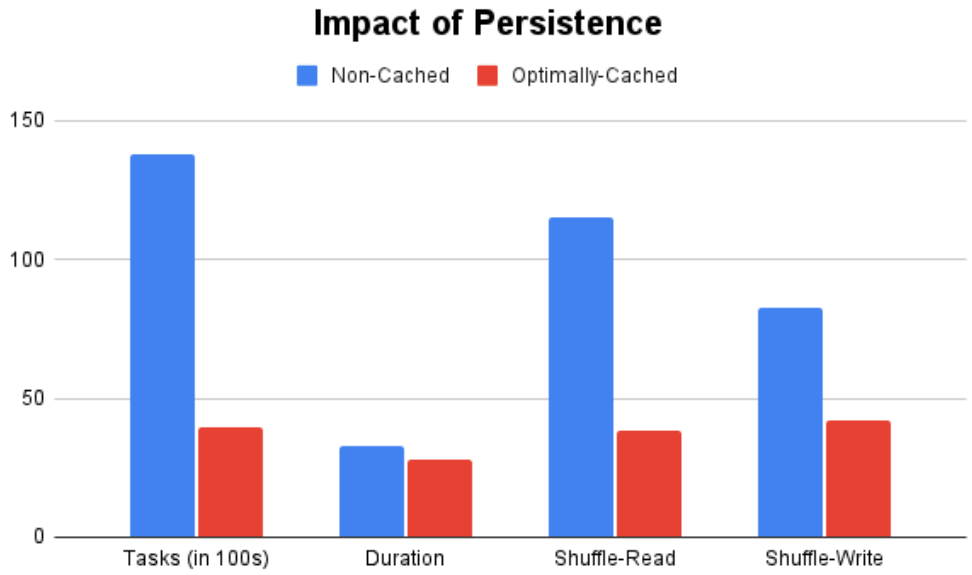
Reasons

- We saw a decrease in completion time on increasing the number of partitions from 16, as this leads to smaller individual partitions of datasets being scheduled on executors and hence, now RDDs involved with the tasks can fit in memory without having to make any disk spills.
- Also increasing the number of partitions increases the magnitude of parallelism among the executors, and increases the effective utilization of the cluster.
- We also observed that increasing the number of partitions beyond a certain point did not benefit much as it increased the amount of compute needed to synchronize tasks and hence, added overhead.
- We were able to confirm the above threshold by following calculations, by considering the size of the enwiki-articles dataset being ~4 GB and default partition size being 128 MB:

  4 GB(Dataset size) / 128 MB(Default partition size) => ~32
- Intuitively, the number of tasks increased as we increased the number of partitions due to smaller datasets being scheduled on the executors.

Task 3: Impact of Persisting on Execution

	Nothing Cached	Optimally Cached
Tasks (in 100s)	137.74	39.77
Duration	33	28
Shuffle-Read	115	38.6
Shuffle-Write	82.6	42.2



Observations

- As part of this task, we tried to examine the impact of persisting RDDs on the execution of the Spark PageRank jobs.
- When a Spark application was configured to cache only those RDDs that did not change across the execution of the iterations, we saw a significant decrease in the number of shuffle operations.
- In the case of read shuffle, we saw a ~66% decrease, and for write shuffle, a ~49% decrease.
- Subsequently, this optimal caching of such static RDDs across the iterations reduced the number of tasks being spawned by the driver by ~71%.

- Apart from that we also saw that the PageRank Spark job was completed sooner than when the job was run with nothing cached.
- In contrast, when every intermediate RDD was blindly being persisted, we saw a lot of disk spills and these delayed the completion of the Spark application by more than 1 hour.

Reasons

- We believe the reason behind the significant decrease in the number of shuffle operations and faster completions is due to avoiding recomputing an RDD during each iteration, which is not expected to be modified across the iterations.
- Spark being Cache-Aware and Partition-Aware spawns tasks on favorable workers to preserve data locality.
- Observations also infer that persisting heavily irrationally can cause the cluster to under-perform, as there will be frequent disk spills and memory swapping due to underlying page-replacement algorithms.

Task 4: Impact of Failures on Execution

Overview stages screenshot:

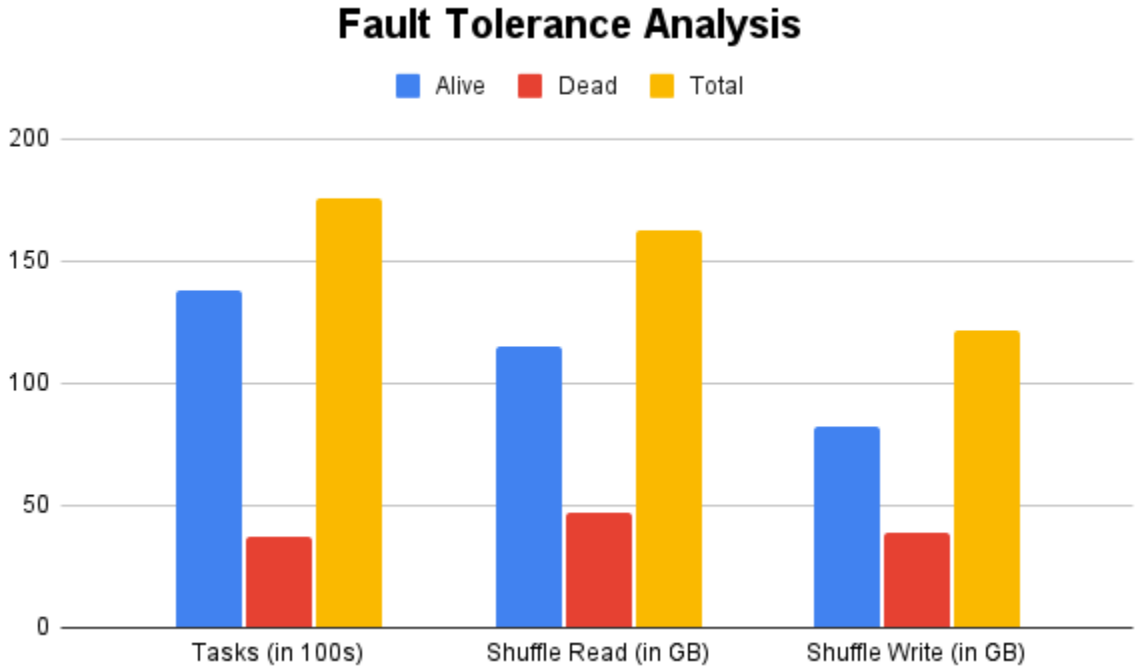
Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
21	top at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:79	+details	2024/02/07 10:48:49	50 s	1067/1067			3.7 GiB	
20	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:47:06	1.7 min	1067/1067			4.4 GiB	3.7 GiB
19	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:45:22	1.7 min	1067/1067			7.2 GiB	4.4 GiB
18	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:43:14	2.1 min	970/970			4.4 GiB	3.6 GiB
17 (retry 1)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:42:01	1.2 min	647/647			4.6 GiB	2.8 GiB
16 (retry 1)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:41:25	35 s	315/315			1600.9 MiB	1277.8 MiB
15 (retry 1)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:39:14	2.2 min	392/392			2.9 GiB	1750.6 MiB
14 (retry 1)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:38:31	42 s	345/345			1983.7 MiB	1533.5 MiB
13 (retry 1)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:37:01	1.5 min	105/105			2.1 GiB	1933.4 MiB
12 (retry 1)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:36:28	33 s	230/230			1512.4 MiB	1135.5 MiB
11 (retry 1)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:33:57	2.5 min	223/223			2.2 GiB	1508.2 MiB
10 (retry 1)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:33:16	41 s	255/255			1943.8 MiB	1393.9 MiB
9 (retry 1)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:32:14	1.0 min	79/79			2.0 GiB	1935.8 MiB
8 (retry 1)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:31:42	31 s	165/165			1519.2 MiB	1045.1 MiB
7 (retry 1)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:30:09	1.6 min	216/216			2.6 GiB	1619.9 MiB
6 (retry 2)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:29:29	40 s	164/164			1892.1 MiB	1242.9 MiB
5 (retry 2)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:28:05	1.4 min	185/185			3.0 GiB	2.0 GiB
4 (retry 2)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:27:16	49 s	129/129			2.1 GiB	1337.5 MiB
3 (retry 2)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:25:48	1.5 min	133/133			3.6 GiB	2.1 GiB
2 (retry 2)	reduceByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:75	+details	2024/02/07 10:23:07	2.7 min	87/87			2.7 GiB	1938.0 MiB
1 (retry 2)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:19:24	3.7 min	91/91	4.5 GiB		1746.4 MiB	2.9 GiB
0 (retry 2)	groupByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:61	+details	2024/02/07 10:18:19	1.1 min	45/45	4.6 GiB			1696.8 MiB

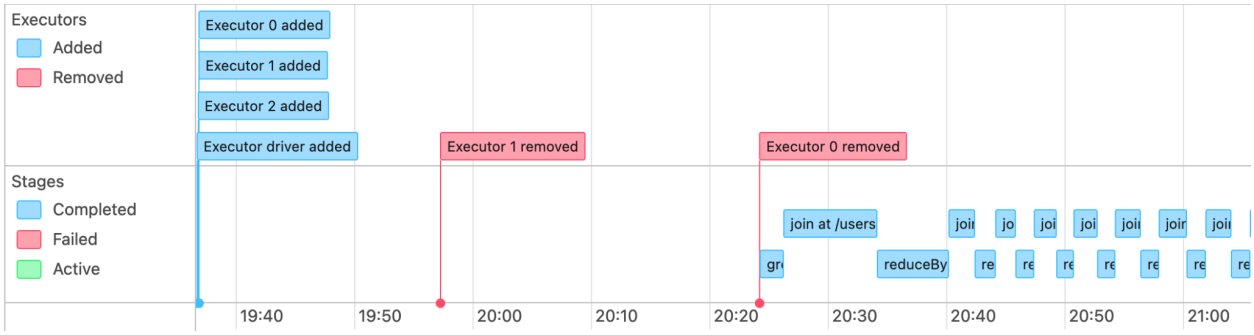
Detailed Screenshot on Failure Recovery:

1 (retry 2)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 10:19:24	3.7 min	91/91	4.5 GiB		1746.4 MiB	2.9 GiB
1 (retry 1)	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 09:59:01	2.3 min	69/69	3.3 GiB		1337.2 MiB	2.2 GiB
1	join at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:67	+details	2024/02/07 09:44:02	5.8 min	194/194	9.9 GiB		3.6 GiB	6.1 GiB
0 (retry 2)	groupByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:61	+details	2024/02/07 10:18:19	1.1 min	45/45	4.6 GiB			1696.8 MiB
0 (retry 1)	groupByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:61	+details	2024/02/07 09:58:09	52 s	31/31	3.3 GiB			1219.7 MiB
0	groupByKey at /users/Abhi01/Big-Data-Assignments/Assignment 1/pagerank.py:61	+details	2024/02/07 09:42:19	1.7 min	97/97	9.9 GiB			3.6 GiB

Duration: 70 min
Total # of Tasks: 17537
Shuffle Read: 162.2 GB
Shuffle Write 121.2 GB

	Alive	Dead	Total
Tasks (in 100s)	137.74	37.63	175.37
Shuffle Read (in GB)	115	47.2	162.2
Shuffle Write (in GB)	82.6	38.6	121.2





Observations

- On killing a worker during the 25% and 75% completion of a Spark job, we saw that executor 1 and executor 0 were removed respectively.
- The total number of tasks executed to complete the job was more than in the case of normal execution (17537 vs 13774).
- The driver rescheduled the failed tasks on the available executors at that point in time.

Reasons

- Spark enables fault-tolerance by rescheduling the failed tasks on the available executors.
- Some of the tasks had to be redone again due to the workers being terminated which is the reason for the total number of tasks being higher.
- The RDDs are being recomputed by leveraging the stored Lineage in the form of DAGs(Directed Acyclic Graphs).