CS 744 - Hadoop and Spark Integration

Winfred Li, Noah Cohen Kalafut, Hao-Yu Shih

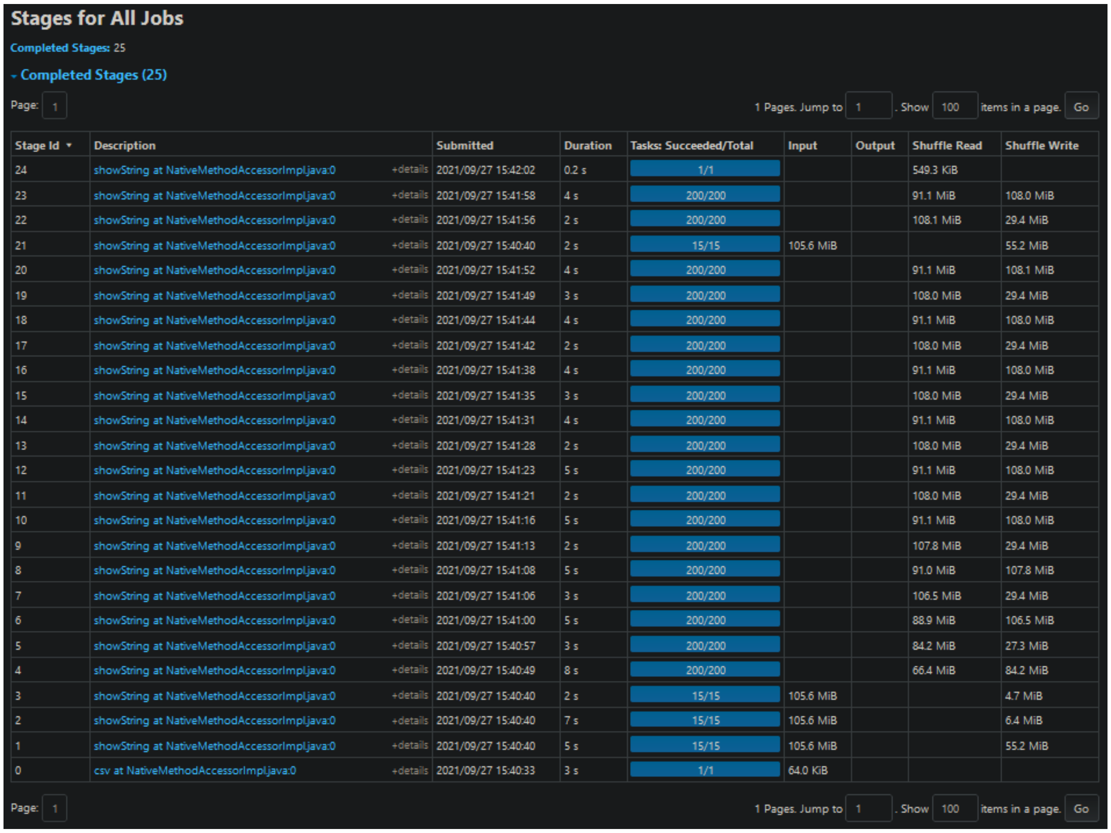
Writing the Algorithm

Writing the Page Rank algorithm for use in Spark can be split into a few parts. First, the data needs to be formatted and the initial ranks need to be assigned. We can do this by using two `DataFrame` instances, *entries* and *ranks*. *entries* can contain three columns: One for the ID of the contributing node, one for the receiving node, and one containing the pre-computed number of neighbors for the contributing node. *ranks* can store the ID and rank of each node. We can then join the two and create a new `DataFrame` *contributions* that first calculates the contribution from each node before calculating the aggregate contributions to each node. Using the expression

$$.15 + .85 * sum(contributions)$$

we can calculate the next iteration of *ranks*.

Using only default partitioning with three executors, the task takes an average of ~83.4 seconds (81.9, 86.6, 81.6) to perform 10 iterations on *web-BerkStan.txt*. An example of the stages completed can be seen below. This version of the algorithm involves a lot of small shuffle reads and writes. This appears to have a negative effect on performance. This becomes clearer in the following section.



Stages for All Jobs

Completed Stages: 25

Completed Stages (25)

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
24	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:42:02	0.2 s	1/1			549.3 KiB	
23	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:58	4 s	200/200			91.1 MiB	108.0 MiB
22	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:56	2 s	200/200			108.1 MiB	29.4 MiB
21	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:40:40	2 s	15/15	105.6 MiB			55.2 MiB
20	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:52	4 s	200/200			91.1 MiB	108.1 MiB
19	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:49	3 s	200/200			108.0 MiB	29.4 MiB
18	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:44	4 s	200/200			91.1 MiB	108.0 MiB
17	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:42	2 s	200/200			108.0 MiB	29.4 MiB
16	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:38	4 s	200/200			91.1 MiB	108.0 MiB
15	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:35	3 s	200/200			108.0 MiB	29.4 MiB
14	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:31	4 s	200/200			91.1 MiB	108.0 MiB
13	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:28	2 s	200/200			108.0 MiB	29.4 MiB
12	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:23	5 s	200/200			91.1 MiB	108.0 MiB
11	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:21	2 s	200/200			108.0 MiB	29.4 MiB
10	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:16	5 s	200/200			91.1 MiB	108.0 MiB
9	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:13	2 s	200/200			107.8 MiB	29.4 MiB
8	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:08	5 s	200/200			91.0 MiB	107.8 MiB
7	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:06	3 s	200/200			106.5 MiB	29.4 MiB
6	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:41:00	5 s	200/200			88.9 MiB	106.5 MiB
5	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:40:57	3 s	200/200			84.2 MiB	27.3 MiB
4	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:40:49	8 s	200/200			66.4 MiB	84.2 MiB
3	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:40:40	2 s	15/15	105.6 MiB			4.7 MiB
2	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:40:40	7 s	15/15	105.6 MiB			6.4 MiB
1	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:40:40	5 s	15/15	105.6 MiB			55.2 MiB
0	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 15:40:33	3 s	1/1	64.0 KiB			

Parallelism and Partitioning

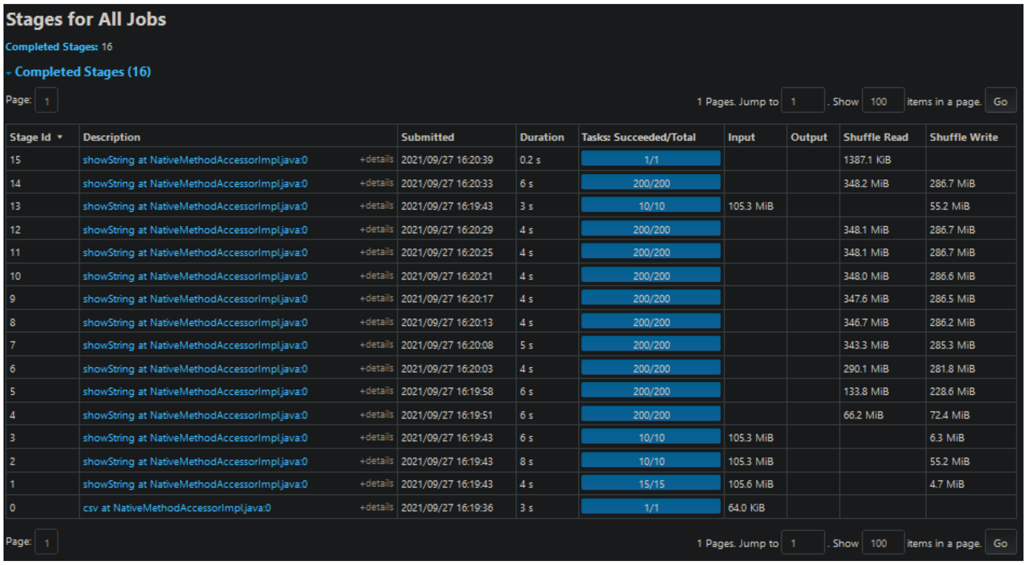
Improper partitioning can have a significant effect on performance. This is generally true, but is especially the case with the Page Rank algorithm. Many reductions need to take place each iteration, and reduction operations frequently involve accessing multiple nodes/parts at the same time. Therefore, the fewer reduction operations that are necessary, the quicker the algorithm will be. Additionally, we want to keep parts of the data that are used frequently together on the same system.

To achieve these goals, we can partition the dataset based on the receiving node during each iteration. This will move all values we need to aggregate into similar places. Implementing this provides a performance benefit of ~28 seconds, and lowers the number of required stages from 25 to 15. Due to the smart pre-allocation, fewer shuffles need to be performed, thereby saving time.

We can additionally implement finer partitioning for our *ranks* `DataFrame`. This does not seem to help and produces no noticeable difference in shuffle size in any stage, implying that this is either default behavior or equivalent. Perhaps this will work better in tandem with our previous optimization?

Implementing both of these changes together provides an average performance benefit of ~28.4 seconds (~33.6%) (55.5, 54.8, 55.9) over automatic partitioning. The latter change does not appear to have a significant effect on the time gain. Perhaps Spark's auto-partitioning works well with the latter optimization.

This benefit can be examined in the Spark UI below of a sample run that took 55.9 seconds. When compared to our original solution above, the difference is apparent. Fewer shuffles with larger read/writes occur, thus saving time while running the algorithm despite each individual shuffle taking longer. From this, it seems that the overhead and coordination involved in shuffling scales relatively slowly when compared to the size of the shuffle.



Stages for All Jobs

Completed Stages: 16

- Completed Stages (16)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

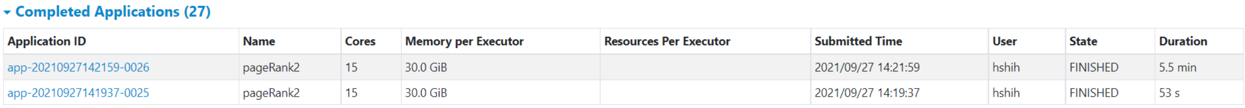
Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
15	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:39	0.2 s	1/1			1387.1 KiB	
14	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:33	6 s	200/200			348.2 MiB	286.7 MiB
13	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:19:43	3 s	10/10	105.3 MiB			55.2 MiB
12	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:29	4 s	200/200			348.1 MiB	286.7 MiB
11	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:25	4 s	200/200			348.1 MiB	286.7 MiB
10	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:21	4 s	200/200			348.0 MiB	286.6 MiB
9	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:17	4 s	200/200			347.6 MiB	286.5 MiB
8	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:13	4 s	200/200			346.7 MiB	286.2 MiB
7	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:08	5 s	200/200			343.3 MiB	285.3 MiB
6	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:20:03	4 s	200/200			290.1 MiB	281.8 MiB
5	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:19:58	6 s	200/200			133.8 MiB	228.6 MiB
4	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:19:51	6 s	200/200			66.2 MiB	72.4 MiB
3	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:19:43	6 s	10/10	105.3 MiB			6.3 MiB
2	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:19:43	8 s	10/10	105.3 MiB			55.2 MiB
1	showString at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:19:43	4 s	15/15	105.6 MiB			4.7 MiB
0	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 16:19:36	3 s	1/1	64.0 KiB			

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Appropriate Persistence of Dataset

In the start of the code, we persist the table, which consists of ID and the number of ID's neighbors. We found out that the time to finish all the tasks has dramatically declined from 6~7 minutes to less than 1 minute.

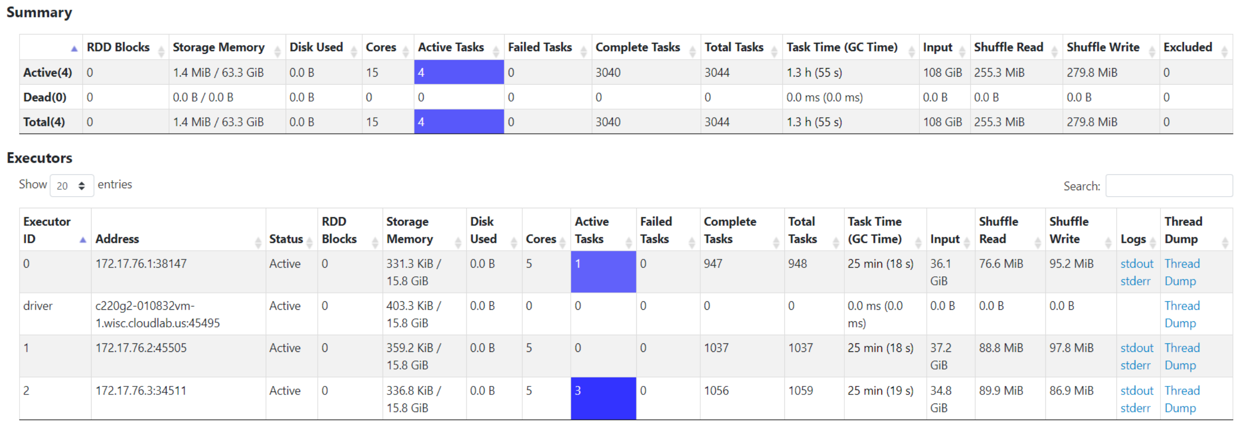
Runtime



▾ Completed Applications (27)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20210927142159-0026	pageRank2	15	30.0 GiB		2021/09/27 14:21:59	hshih	FINISHED	5.5 min
app-20210927141937-0025	pageRank2	15	30.0 GiB		2021/09/27 14:19:37	hshih	FINISHED	53 s

Non Persistence



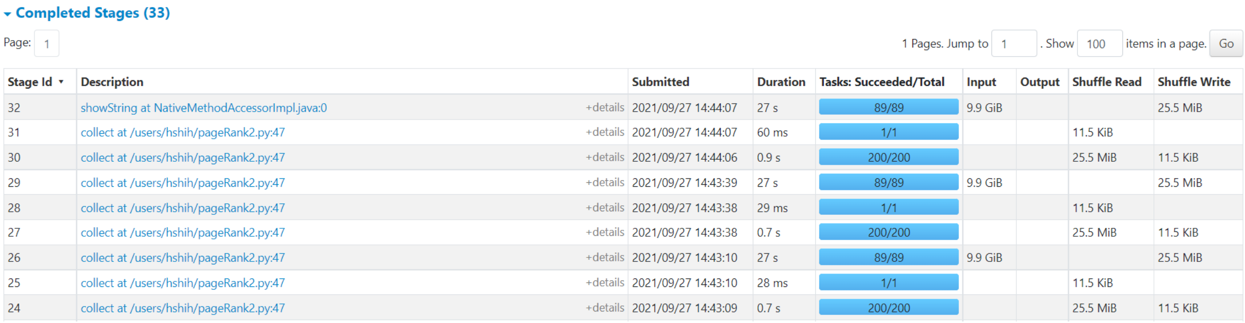
Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(4)	0	1.4 MiB / 63.3 GiB	0.0 B	15	4	0	3040	3044	1.3 h (55 s)	108 GiB	255.3 MiB	279.8 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(4)	0	1.4 MiB / 63.3 GiB	0.0 B	15	4	0	3040	3044	1.3 h (55 s)	108 GiB	255.3 MiB	279.8 MiB	0

Executors

Show 20 entries Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
0	172.17.76.1:38147	Active	0	331.3 KiB / 15.8 GiB	0.0 B	5	1	0	947	948	25 min (18 s)	36.1 GiB	76.6 MiB	95.2 MiB	stdout stderr	Thread Dump
driver	c220g2-010832vm-1.wisc.cloudlab.us:45495	Active	0	403.3 KiB / 15.8 GiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	172.17.76.2:45505	Active	0	359.2 KiB / 15.8 GiB	0.0 B	5	0	0	1037	1037	25 min (18 s)	37.2 GiB	88.8 MiB	97.8 MiB	stdout stderr	Thread Dump
2	172.17.76.3:34511	Active	0	336.8 KiB / 15.8 GiB	0.0 B	5	3	0	1056	1059	25 min (19 s)	34.8 GiB	89.9 MiB	86.9 MiB	stdout stderr	Thread Dump

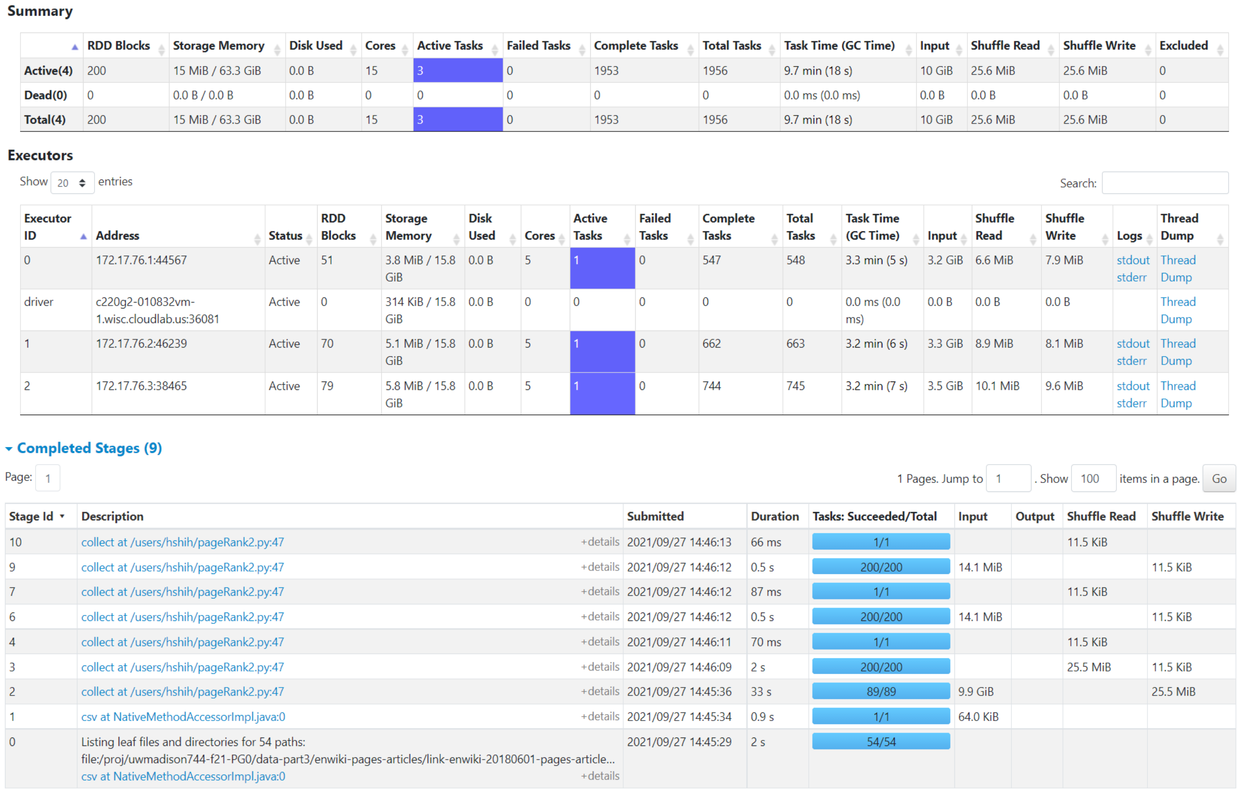


▾ Completed Stages (33)

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
32	showString at NativeMethodAccessorImpl.java:0	+details	2021/09/27 14:44:07	27 s	89/89	9.9 GiB			25.5 MiB
31	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:44:07	60 ms	1/1			11.5 KiB	
30	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:44:06	0.9 s	200/200			25.5 MiB	11.5 KiB
29	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:43:39	27 s	89/89	9.9 GiB			25.5 MiB
28	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:43:38	29 ms	1/1			11.5 KiB	
27	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:43:38	0.7 s	200/200			25.5 MiB	11.5 KiB
26	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:43:10	27 s	89/89	9.9 GiB			25.5 MiB
25	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:43:10	28 ms	1/1			11.5 KiB	
24	collect at /users/hshih/pageRank2.py:47	+details	2021/09/27 14:43:09	0.7 s	200/200			25.5 MiB	11.5 KiB

Persistence



Summary

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Excluded
Active(4)	200	15 MiB / 63.3 GiB	0.0 B	15	3	0	1953	1956	9.7 min (18 s)	10 GiB	25.6 MiB	25.6 MiB	0
Dead(0)	0	0.0 B / 0.0 B	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B	0
Total(4)	200	15 MiB / 63.3 GiB	0.0 B	15	3	0	1953	1956	9.7 min (18 s)	10 GiB	25.6 MiB	25.6 MiB	0

Executors

Show 20 entries

Search:

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write	Logs	Thread Dump
0	172.17.76.1:44567	Active	51	3.8 MiB / 15.8 GiB	0.0 B	5	1	0	547	548	3.3 min (5 s)	3.2 GiB	6.6 MiB	7.9 MiB	stdout stderr	Thread Dump
driver	c220g2-010832vm-1.wisc.cloudlab.us:36081	Active	0	314 KiB / 15.8 GiB	0.0 B	0	0	0	0	0	0.0 ms (0.0 ms)	0.0 B	0.0 B	0.0 B		Thread Dump
1	172.17.76.2:46239	Active	70	5.1 MiB / 15.8 GiB	0.0 B	5	1	0	662	663	3.2 min (6 s)	3.3 GiB	8.9 MiB	8.1 MiB	stdout stderr	Thread Dump
2	172.17.76.3:38465	Active	79	5.8 MiB / 15.8 GiB	0.0 B	5	1	0	744	745	3.2 min (7 s)	3.5 GiB	10.1 MiB	9.6 MiB	stdout stderr	Thread Dump

Completed Stages (9)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

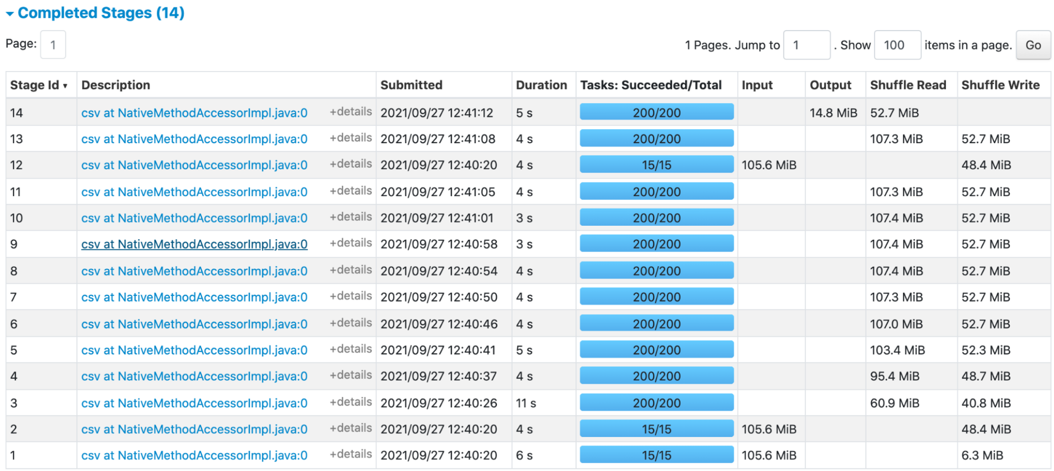
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
10	collect at /users/hshih/pageRank2.py:47 +details	2021/09/27 14:46:13	66 ms	1/1			11.5 KiB	
9	collect at /users/hshih/pageRank2.py:47 +details	2021/09/27 14:46:12	0.5 s	200/200	14.1 MiB			11.5 KiB
7	collect at /users/hshih/pageRank2.py:47 +details	2021/09/27 14:46:12	87 ms	1/1			11.5 KiB	
6	collect at /users/hshih/pageRank2.py:47 +details	2021/09/27 14:46:12	0.5 s	200/200	14.1 MiB			11.5 KiB
4	collect at /users/hshih/pageRank2.py:47 +details	2021/09/27 14:46:11	70 ms	1/1			11.5 KiB	
3	collect at /users/hshih/pageRank2.py:47 +details	2021/09/27 14:46:09	2 s	200/200			25.5 MiB	11.5 KiB
2	collect at /users/hshih/pageRank2.py:47 +details	2021/09/27 14:45:36	33 s	89/89	9.9 GiB			25.5 MiB
1	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 14:45:34	0.9 s	1/1	64.0 KiB			
0	Listing leaf files and directories for 54 paths: file:/proj/uwmadison744-f21-PG0/data-part3/enwiki-pages-articles/link-enwiki-20180601-pages-article... csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 14:45:29	2 s	54/54				

We can see the application will run some stages recursively. One of the stages that need to cost 30 to 40 seconds is the main factor that affects the runtime most. With persistence, the application only needs to run it one time, but it will run it recursively without persistence. This is due to the fact that the table we persist will reuse again and again in the loop. If we don't persist the data, spark will load that data in each iteration.

Performance with Failure

We examine the effect of worker failure on cluster performance on the Berkeley dataset. Without failures, the task takes approximately 60 seconds to complete. When a worker is manually killed after approximately 25% of the job, the completion time increases to 74 seconds, and when a worker is killed after 75% of the job, the completion time is 78 seconds. We can examine the stage logs to get a better sense of this behavior.

No Failures

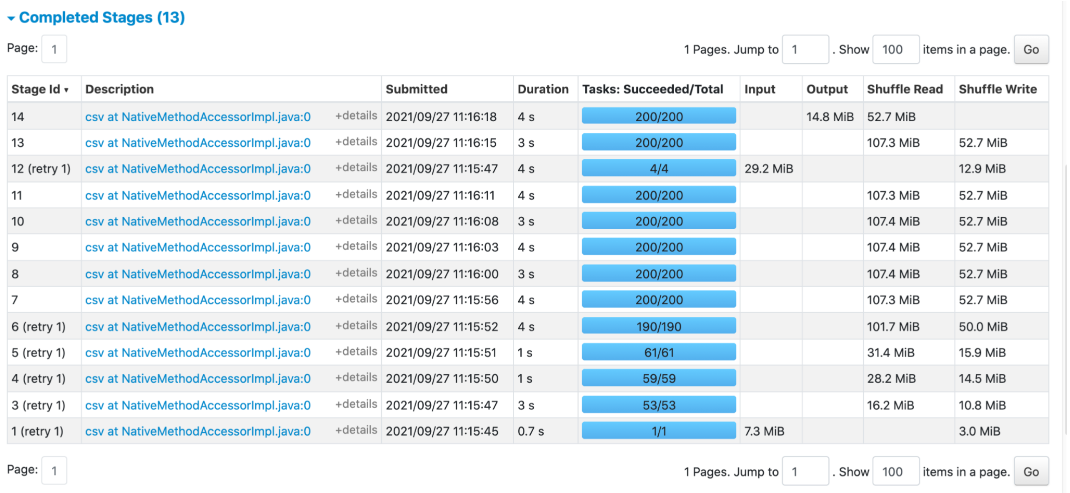


▾ Completed Stages (14)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:41:12	5 s	200/200		14.8 MiB	52.7 MiB	
13	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:41:08	4 s	200/200			107.3 MiB	52.7 MiB
12	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:20	4 s	15/15	105.6 MiB			48.4 MiB
11	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:41:05	4 s	200/200			107.3 MiB	52.7 MiB
10	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:41:01	3 s	200/200			107.4 MiB	52.7 MiB
9	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:58	3 s	200/200			107.4 MiB	52.7 MiB
8	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:54	4 s	200/200			107.4 MiB	52.7 MiB
7	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:50	4 s	200/200			107.3 MiB	52.7 MiB
6	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:46	4 s	200/200			107.0 MiB	52.7 MiB
5	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:41	5 s	200/200			103.4 MiB	52.3 MiB
4	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:37	4 s	200/200			95.4 MiB	48.7 MiB
3	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:26	11 s	200/200			60.9 MiB	40.8 MiB
2	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:20	4 s	15/15	105.6 MiB			48.4 MiB
1	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 12:40:20	6 s	15/15	105.6 MiB			6.3 MiB

25%



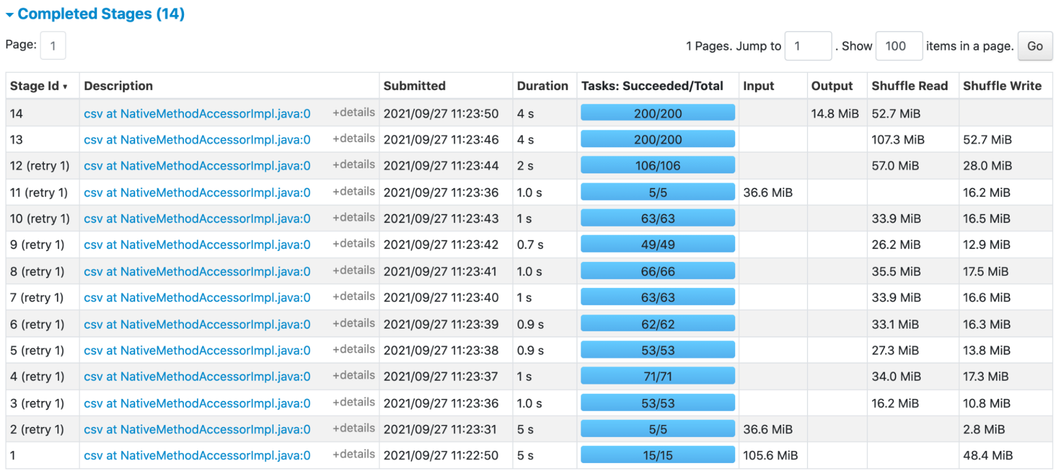
▾ Completed Stages (13)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:16:18	4 s	200/200		14.8 MiB	52.7 MiB	
13	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:16:15	3 s	200/200			107.3 MiB	52.7 MiB
12 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:15:47	4 s	4/4	29.2 MiB			12.9 MiB
11	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:16:11	4 s	200/200			107.3 MiB	52.7 MiB
10	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:16:08	3 s	200/200			107.4 MiB	52.7 MiB
9	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:16:03	4 s	200/200			107.4 MiB	52.7 MiB
8	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:16:00	3 s	200/200			107.4 MiB	52.7 MiB
7	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:15:56	4 s	200/200			107.3 MiB	52.7 MiB
6 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:15:52	4 s	190/190			101.7 MiB	50.0 MiB
5 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:15:51	1 s	61/61			31.4 MiB	15.9 MiB
4 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:15:50	1 s	59/59			28.2 MiB	14.5 MiB
3 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:15:47	3 s	53/53			16.2 MiB	10.8 MiB
1 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:15:45	0.7 s	1/1	7.3 MiB			3.0 MiB

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

75%



▾ Completed Stages (14)

Page: 1 | 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
14	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:50	4 s	200/200		14.8 MiB	52.7 MiB	
13	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:46	4 s	200/200			107.3 MiB	52.7 MiB
12 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:44	2 s	106/106			57.0 MiB	28.0 MiB
11 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:36	1.0 s	5/5	36.6 MiB			16.2 MiB
10 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:43	1 s	63/63			33.9 MiB	16.5 MiB
9 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:42	0.7 s	49/49			26.2 MiB	12.9 MiB
8 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:41	1.0 s	66/66			35.5 MiB	17.5 MiB
7 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:40	1 s	63/63			33.9 MiB	16.6 MiB
6 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:39	0.9 s	62/62			33.1 MiB	16.3 MiB
5 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:38	0.9 s	53/53			27.3 MiB	13.8 MiB
4 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:37	1 s	71/71			34.0 MiB	17.3 MiB
3 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:36	1.0 s	53/53			16.2 MiB	10.8 MiB
2 (retry 1)	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:23:31	5 s	5/5	36.6 MiB			2.8 MiB
1	csv at NativeMethodAccessorImpl.java:0 +details	2021/09/27 11:22:50	5 s	15/15	105.6 MiB			48.4 MiB

We can see that as the failure occurs later in the job lifetime, more stages must be retried, which makes sense due to the wide dependency nature of the task. This is also why we observe slightly longer completion times for the 75% failure case. Another point to note is that the repeated stages do not take as long as they did on the first attempt. This is also reflected in the read and write statistics where the amount of data being read and written is substantially less for the repeated tasks. This is due to the fact that only the computations involving the killed worker must be repeated. Therefore, while worker failure can certainly have a detrimental effect on performance, it seems that increasing the number of workers could help alleviate this problem since less recomputation would need to be done in the event of a failure.

Contributions

Winfred set up the CloudLab server and mounted the disks. Noah set up Slack, GitHub, and documents. Each group member then individually worked through adding keys, installing Hadoop and Spark, and implementing both algorithms.

Once everything was implemented, we decided to use Hao-Yu's code as a base for part 3 due to its optimization and Noah's for part 2. From there, each group member modified the code/tested it in different scenarios and wrote about the performance for each task. Noah wrote on writing the algorithm and parallelism, Hao-Yu wrote on persistence, and Winfred wrote on failure.