ECE419 Milestone 4: Distributed Systems - MapReduce

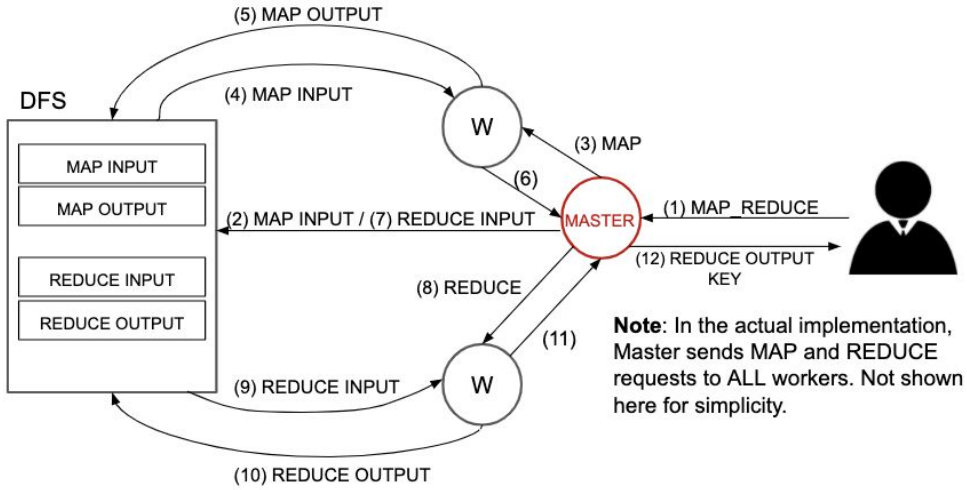
Jing Yi Li, 1002346730 | Patricia Marukot, 1002157499 | Seongju Yun, 1002275290

Demo Link:

https://drive.google.com/file/d/10ruquI0yGmPFupw5g6rVXVfo0Uy4vrHg/view?usp=sharing

Brief Description of Feature: MapReduce

For the creative feature, we implemented the MapReduce functionality as specified in the *MapReduce: Simplified Data Processing on Large Clusters* research paper. We implemented 1) The generic MapReduce workflow which can execute any user-defined MapReduce functions, and 2) Two common MapReduce functions -- Word Frequency (as outlined in the lecture slides), and Word Sorting.



Technical Details

Below outlines the design choices made to implement the steps described in the *MapReduce: Simplified Data Processing on Large Clusters* paper.

Resources

Master and workers

- When the client issues the **MapReduce** request, the server receiving the request is automatically selected as the master while the remaining participating servers (status = START, STOP) are the workers.

Map/Reduce Output storage:

- After each mapper and reducer completes its respective task, they store the results back to the DFS with a unique ID and notify the master. This way, data does not need to be passed around from master to worker and vice versa, any worker can execute a **MAP** or **REDUCE** request on *any* piece of data as long as they are given the ID to be processed.

New Request Types

Flow	Message Type	Description
CLIENT → MASTER	MAP_REDUCE	Sent to a server to trigger MapReduce on a specific KV pair. The receiving server = Master
MASTER → WORKER	MAP	Sent to workers indicating that they must execute a MAP function on a specified chunk
MASTER → WORKER	REDUCE	Sent to workers indicating that they must execute a REDUCE function on a specified chunk

Workflow

1. Client issues a MAP_REDUCE request for a KV pair to a participating server.
2. Master takes care of splitting up data and sending it to the worker nodes to be processed.
    a. To maximize utilization, the MAP tasks are split into M chunks, where M = # of active servers, and each server is responsible for a single chunk
    b. **Note:** In order for multiple reducers to be used, the number of unique keys outputted from the MAP phase must be greater than 1024 - this is a rare case for the 2 MapReduce functions that we implemented. This is due to the fact that the network overhead of sending multiple REDUCE requests introduces large latencies for only a small set of data.

Testing

A cluster of 15 KVServers running on separate UG machines was used to perform all tests. The cluster size was deliberately reduced for some tests to analyze the variation in performance. To minimize load on the cluster, ECSClient heartbeats (for failure detection - Milestone 3) were disabled; as well, the ECSClient and the Client are run in separate UG machines for the same reason.

Functional Testing

Functional and performance tests were completed simultaneously. Testing was trivial: for word frequency, count how many times each word appears, for sorting, manually sort the words and verify the results.

Performance Testing

Performance testing was completed for the following variables:

1. Input document size (i.e. # of words in the document)
2. Number of participating nodes

Tests were completed for both the word frequency and sorting MR functions. Each performance test was completed 3 times and the times were averaged for more accurate results.

UG machines:

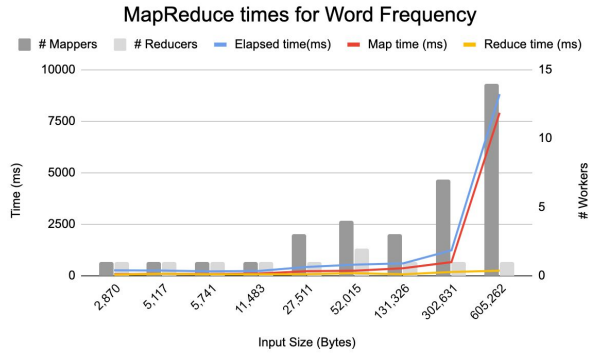
Note: Client & ECS running on ug204. Port 50009 was used for all servers

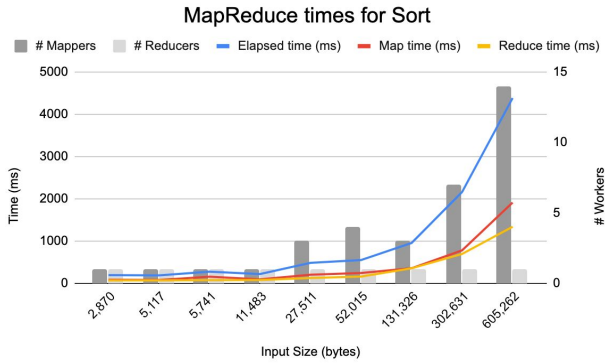
UG machine #	Server name	eno1 IPv4	UG machine #	Server name	eno1 IPv4
205	server1	128.100.13.205	213	server9	128.100.13.213
206	server2	128.100.13.206	214	server10	128.100.13.214
207	server3	128.100.13.207	215	server11	128.100.13.215
208	server4	128.100.13.208	216	server12	128.100.13.216
209	server5	128.100.13.209	217	server13	128.100.13.217
210	server6	128.100.13.210	218	server14	128.100.13.218
211	server7	128.100.13.211	219	server15	128.100.13.219
212	server8	128.100.13.212			

Performance Test I: *Input Document Size*

Test Description:

- Varied the size of the input document to be passed into MapReduce
- *Control variable:* Number of nodes = 8 (up to 15 nodes)





Measurements for WORD_FREQ:

Input Size (bytes)	Doc size (words)	Elapsed time(ms)	Map time (ms)	Reduce time (ms)	M	R
2,870	512	269.33	77	59.33	1	1
5,117	800	259	100	102.67	1	1
5,741	1024	222.66	90.67	79	1	1
11,483	2048	239.33	112.333	75	1	1
27,511	5000	432.66	224	84	3	1
52,015	8000	544	254.33	130.66	4	2
131,326	24000	608.66	370.33	74	3	1
302,631	9000	1241	668.33	192	7	1
605,262	18000	8814.33	7908	248.67	14	1

Measurements for SORT:

Input Size (bytes)	Doc size	Elapsed time (ms)	Map time (ms)	Reduce time (ms)	M	R
2,870	512	196	88	71.67	1	1
5,117	800	191.67	81	71.667	1	1
5,741	1024	279.33	160.667	78.33	1	1
11,483	2048	222.67	97.67	82	1	1
27,511	5000	487	207.33	128	3	1
52,015	8000	552.67	246.33	162	4	1
131,326	24000	957	358.67	358.33	3	1
302,631	9000	2161.33	781.67	699	7	1
605,262	18000	4382	1911.67	1340.67	14	1

Conclusions:

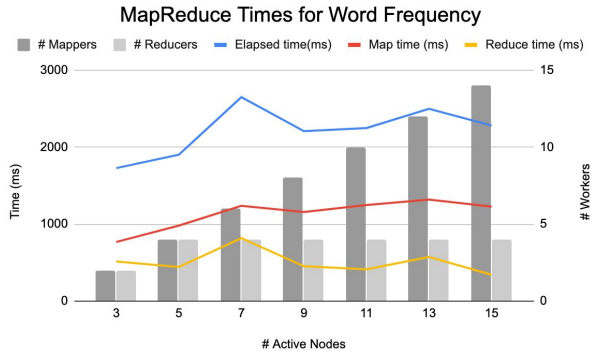
Expected outcomes:

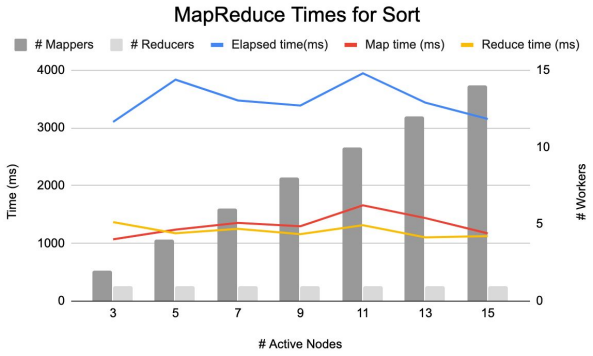
- ✓ Execution times should increase as the input size increases due to processing times at the mapper nodes
- ✓ The number of mappers used should increase as the input size increases

Note: Due to our implementation, the number of reducers does not vary greatly. In order for multiple reducers to be used, the number of unique keys outputted from the MAP phase must be greater than 1024 - this is a rare case for the 2 MapReduce functions that we implemented.

Performance Test II: *Number of participating nodes*
Test Description:

- Varied the number of active nodes participating in MR
- *Control variable*: Input document size = 600KB





Measurements for WORD_FREQ:

# Active Nodes	Elapsed time(ms)	Map time (ms)	Reduce time (ms)	M	R
3	1727.667	768	515	2	2
5	1901.667	980.667	443	4	4
7	2649	1238	817.333	6	4
9	2206.667	1157.667	452.667	8	4
11	2248.333	1247.667	413.333	10	4
13	2498	1319	574.333	12	4
15	2280.667	1228.333	342	14	4

Measurements for SORT:

# Active Nodes	Elapsed time(ms)	Map time (ms)	Reduce time (ms)	M	R
3	3103	1069.333	1367.667	2	1
5	3834.333	1239	1175.333	4	1
7	3474.667	1354	1250.667	6	1
9	3387.667	1297.333	1159	8	1
11	3945.667	1658.333	1314.333	10	1
13	3436.667	1438.667	1103.333	12	1
15	3154.667	1173.333	1125.667	14	1

Expected outcomes:

- ✖ Execution times should decrease as you increase the number of active nodes
- ✓ The # of mappers should increase as the number of active nodes increases

Conclusions:
The measurements show that performance essentially plateaued from 3 allocated nodes; despite increases in compute resources, execution time stayed relatively the same. We believe the reason for this anomaly is due to the input size used to conduct our tests being too small: in order for MapReduce to be effective, the amount of data processed per map/reduce task needs to far surpass the overhead spent on network communications, replication, and non-data-centric processing. The input size will likely have to be in the order of gigabytes (a similar load to Google's average workload). Unfortunately the UG disk quota limit disallowed us to verify this hypothesis; however, given a large enough input, and various optimizations in network communications and storage, we believe that the execution time for MapReduce tasks should decrease proportionally to the increase in number of nodes allocated.