Assignment 1 : CS 838 : Big Data Systems – Fall 2016

Adithya Bhat (bhat5@wisc.edu)
Aparna Subramanian (asubramania7@wisc.edu)

Question 1

a. Query Completion Time

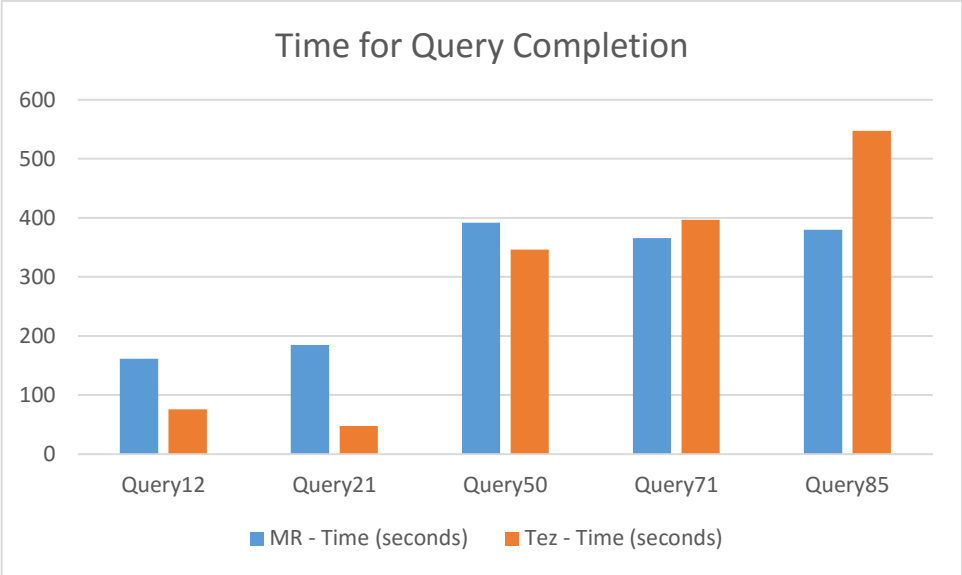


Fig 1.

Short answers –

- Hive / Tez is often, but not always better than Hive / MR.
- Difference is not always constant.
- Reason for varying difference – Tez is not simply a faster version of MR. The Hive query being performed has different representations in the MR model, and the DAG model, as can be seen from the Figures in Section 1.d. Hence, the performance difference will be on a per-query basis.
- More details, discussion at end of **section 1**, since I believe those metrics are correlated with performance.
- Query 71 and 85, Tez does worse than MR. More Reasoning at end of section 1, after DAGs, statistics, and utilization data.

b) Network / Storage Utilization

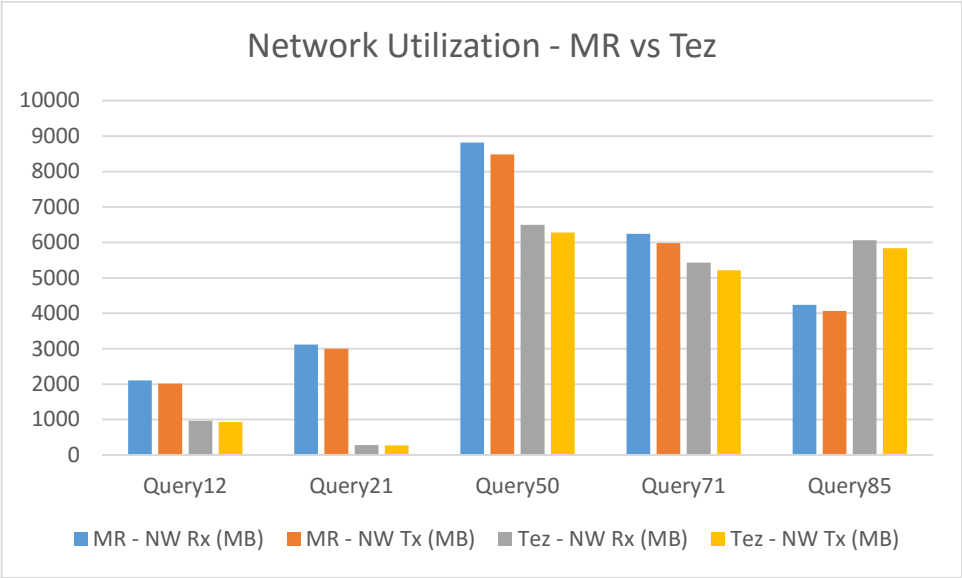


Fig 2.

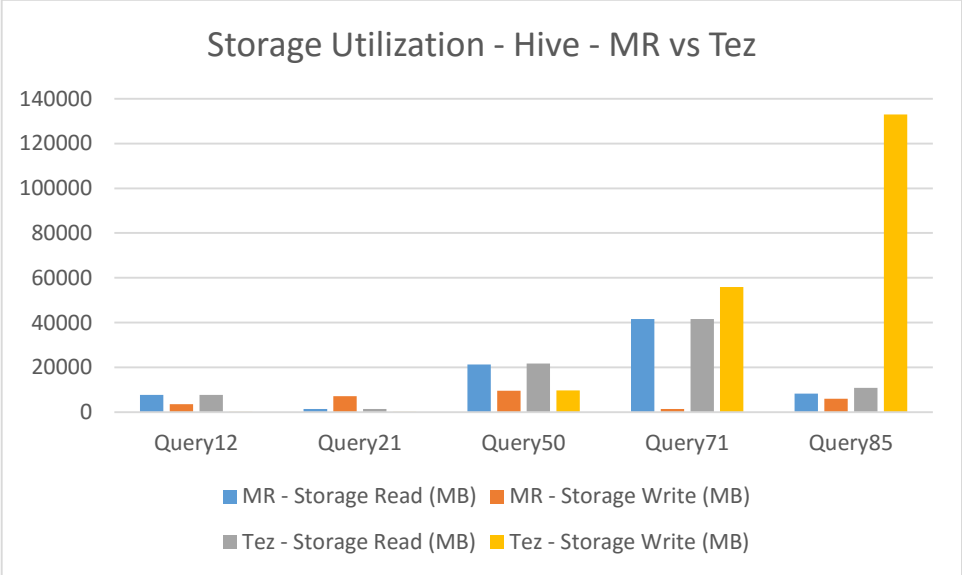


Fig 3.

Overall Pattern – Local storage utilization is much higher than Network utilization. An obvious factor is the Data – local scheduling by MR and Tez.

Another possible factor is the DAG structure, which isn't limited to the M-R model, which could reduce the number of reduce tasks, and hence shuffle / copy is reduced.

c) Task Statistics

Hive / MR	Reducers	Mappers	Ratio	Total Num Tasks
Query 12	3	39	0.08	42
Query 21	2	20	0.10	22
Query 50	88	99	0.89	187
Query 71	2	168	0.01	170
Query 85	40	52	0.77	92

Table 1

Hive / Tez	Aggregators	Readers	Ratio	Total Num Tasks
Query 12	53	2	26.50	55
Query 21	28	3	9.33	31
Query 50	5	92	0.05	97
Query 71	139	5	27.80	144
Query 85	61	25	2.44	86

Table 2

Some trends, anomalies are discussed below the respective graphs. The others are discussed in the section at the end of 1.

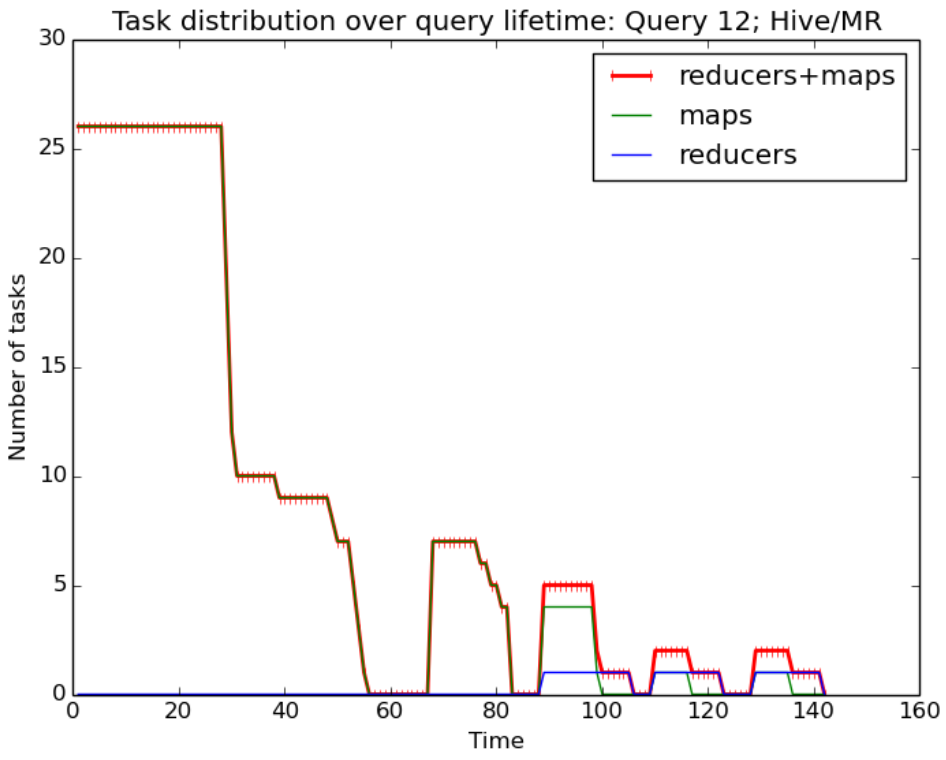


Fig 4.a

Note: the gaps between stages – **overhead of stitching** MR applications / stages together.

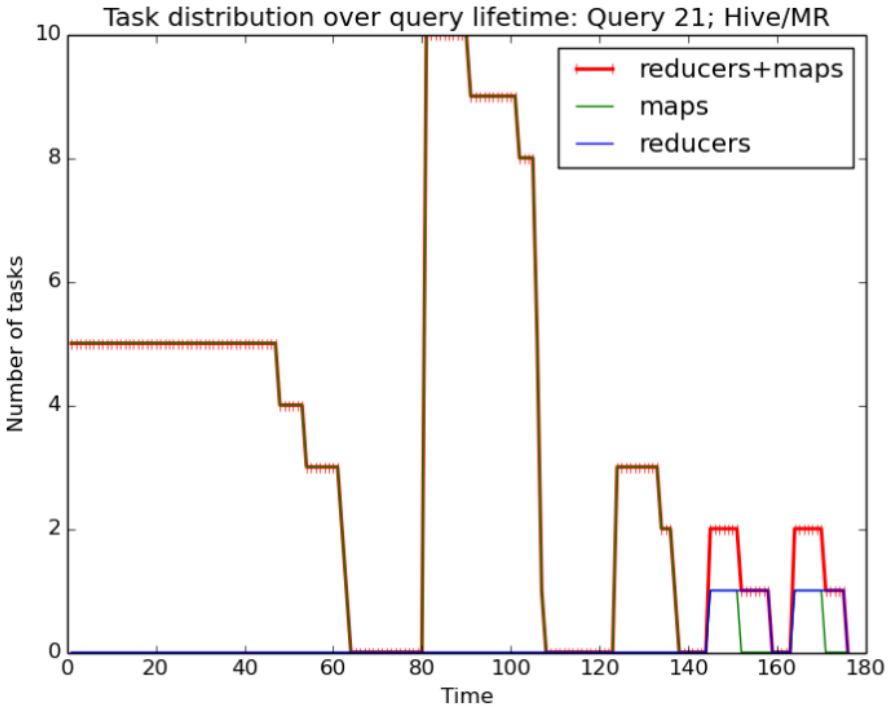


Fig 4b

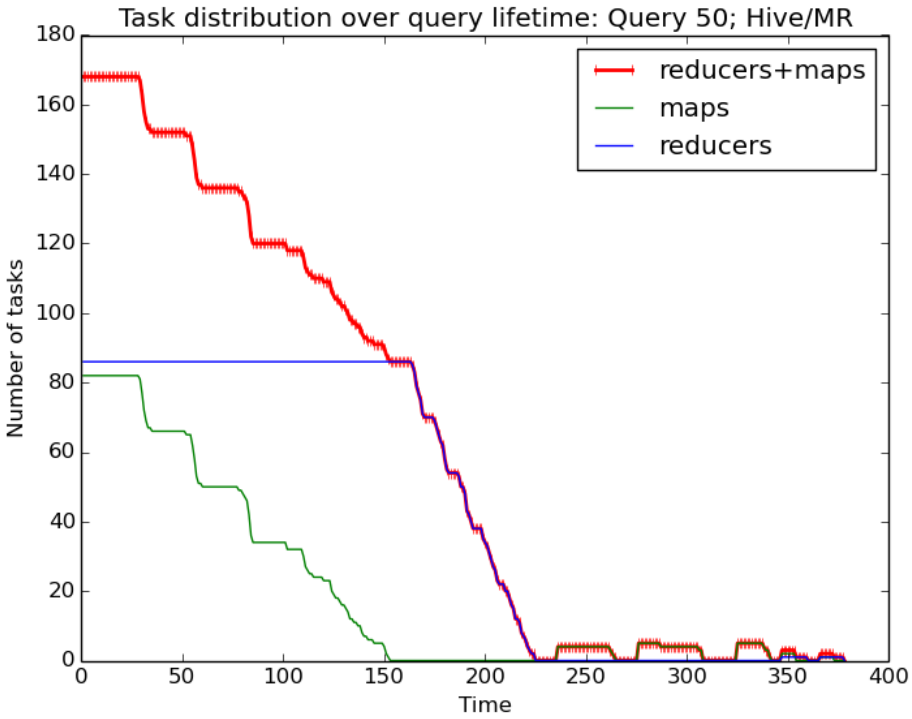


Fig 4c

Note that the step size for task completion is around 16, which is the number of vCPUs in our cluster.

Another interesting point is that Reduce tasks are **created at the same time** as the Map tasks. However, default **slowstart.completedmaps = 1**, so none will be scheduled until all maps complete.. Perhaps it would save some CPU cycles if they weren't even created until some CPU is freed up by completion of map tasks? This can even be statically found out from the config file.

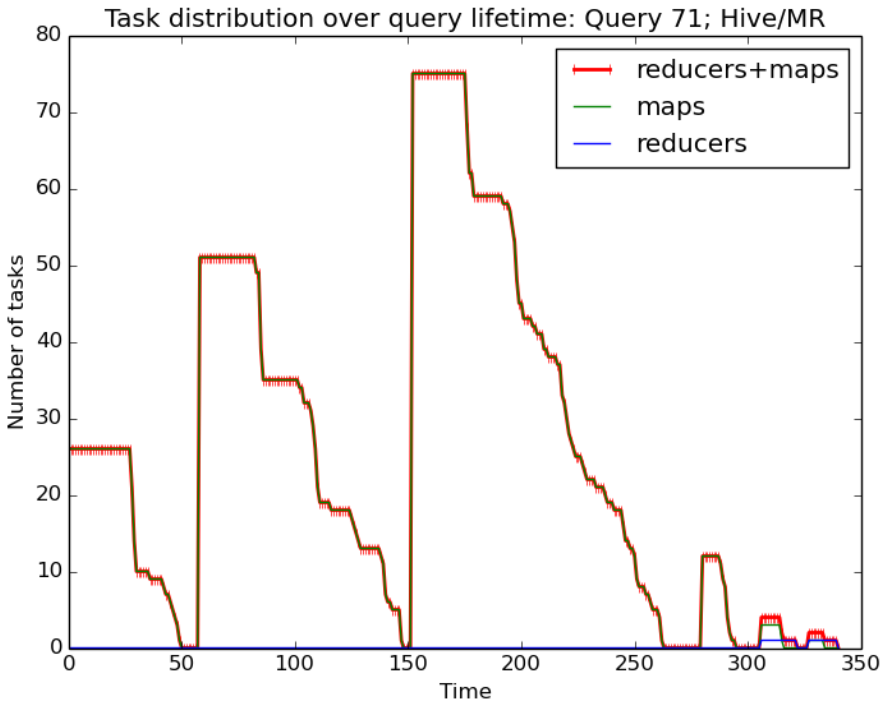


Fig 4d

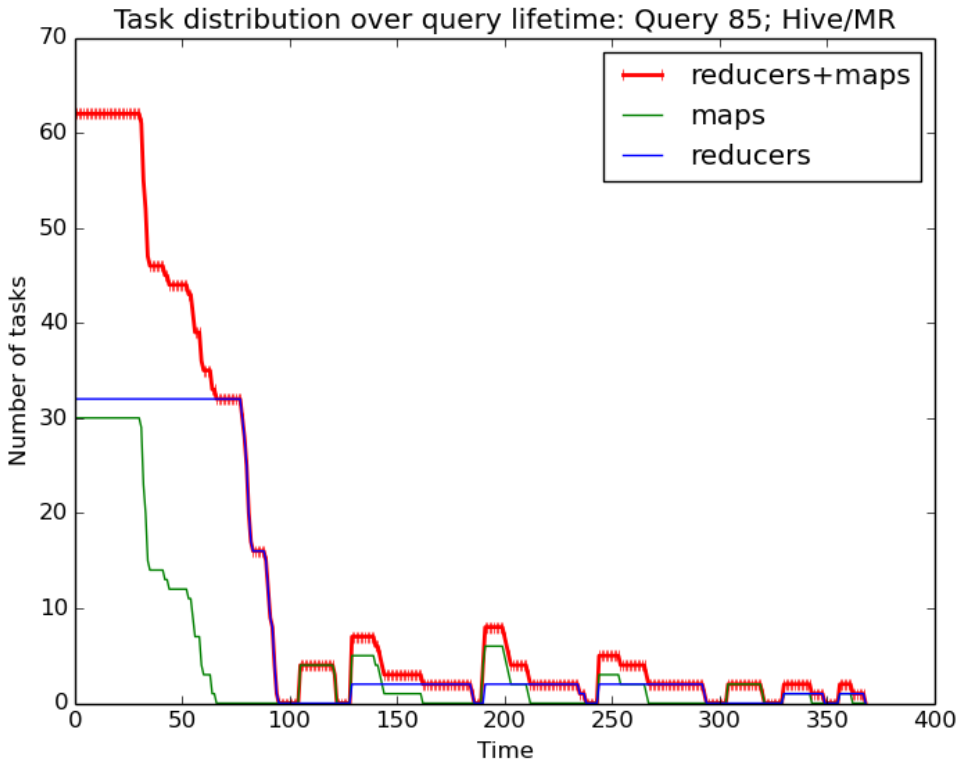


Fig 4e

Again note, intial drop = 16 = vCPU count. Also note that the Reduce tasks complete almost immediately after the map tasks are done, for the first 16. Perhaps such a high number of tasks aren't needed, reducing cross machine data shuffle.

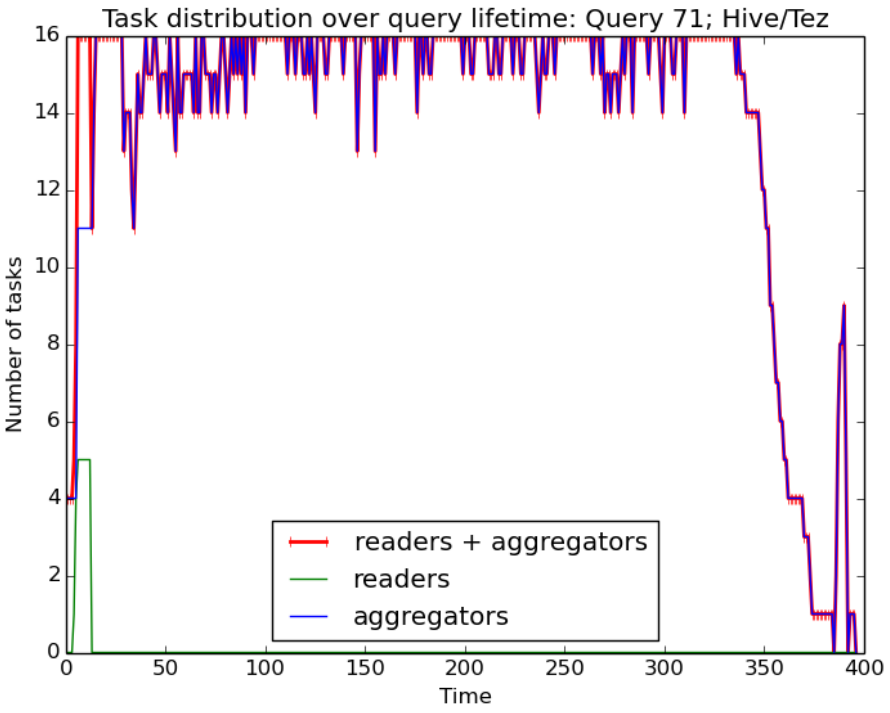

Task distribution over query lifetime: Query 71; Hive/Tez
Number of tasks
Time
readers + aggregators
readers
aggregators
0
2
4
6
8
10
12
14
16
0
50
100
150
200
250
300
350
400

Fig 5a

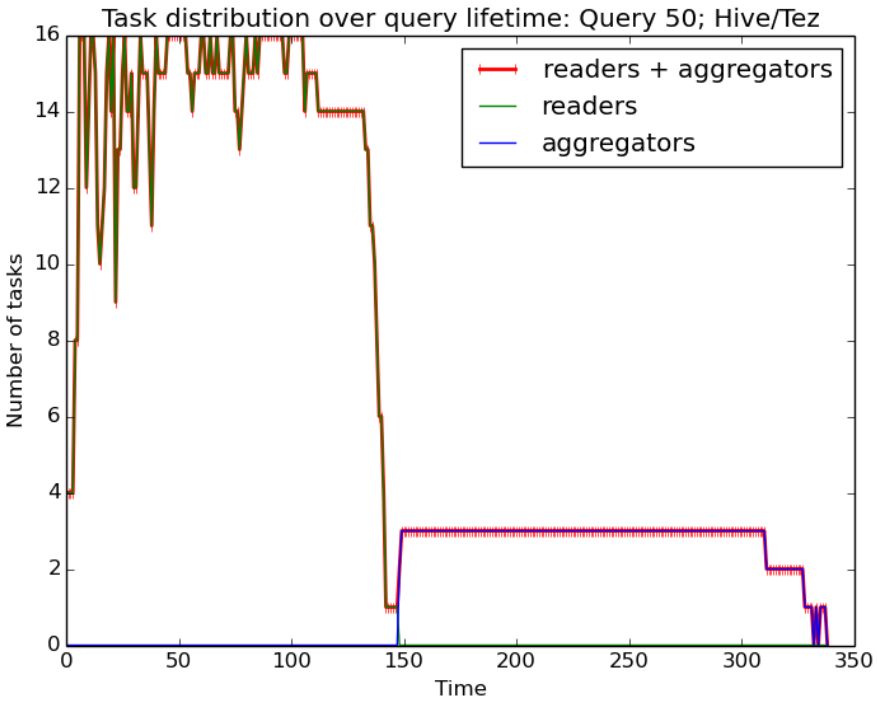

Task distribution over query lifetime: Query 50; Hive/Tez
Number of tasks
Time
readers + aggregators
readers
aggregators
0
2
4
6
8
10
12
14
16
0
50
100
150
200
250
300
350

Fig 5b

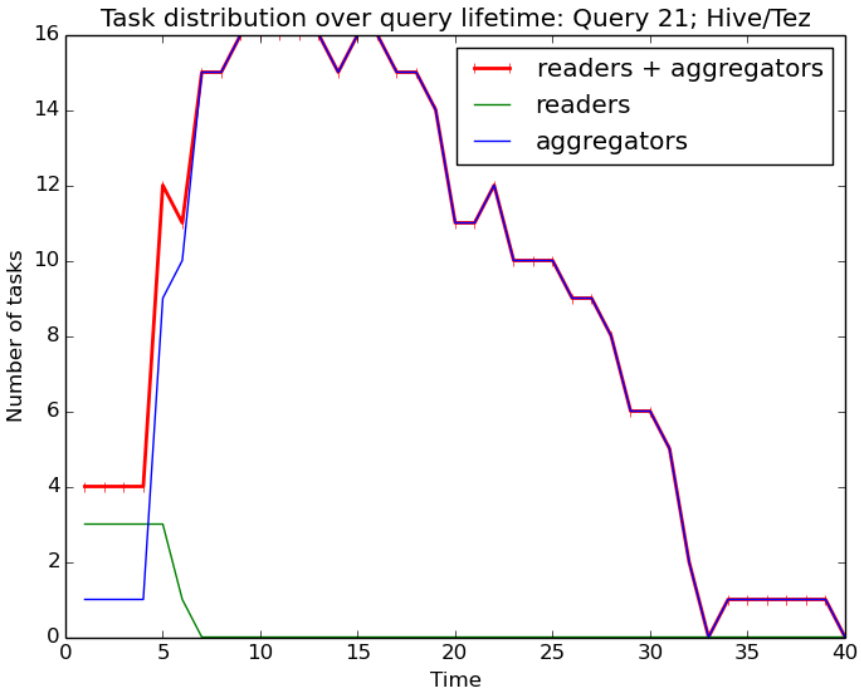

Task distribution over query lifetime: Query 21; Hive/Tez
readers + aggregators
readers
aggregators
Number of tasks
Time
0
2
4
6
8
10
12
14
16
0
5
10
15
20
25
30
35
40

Fig 5c


Task distribution over query lifetime: Query 12; Hive/Tez
readers + aggregators
readers
aggregators
Number of tasks
Time
0
2
4
6
8
10
12
14
16
0
10
20
30
40
50
60
70

Fig 5d

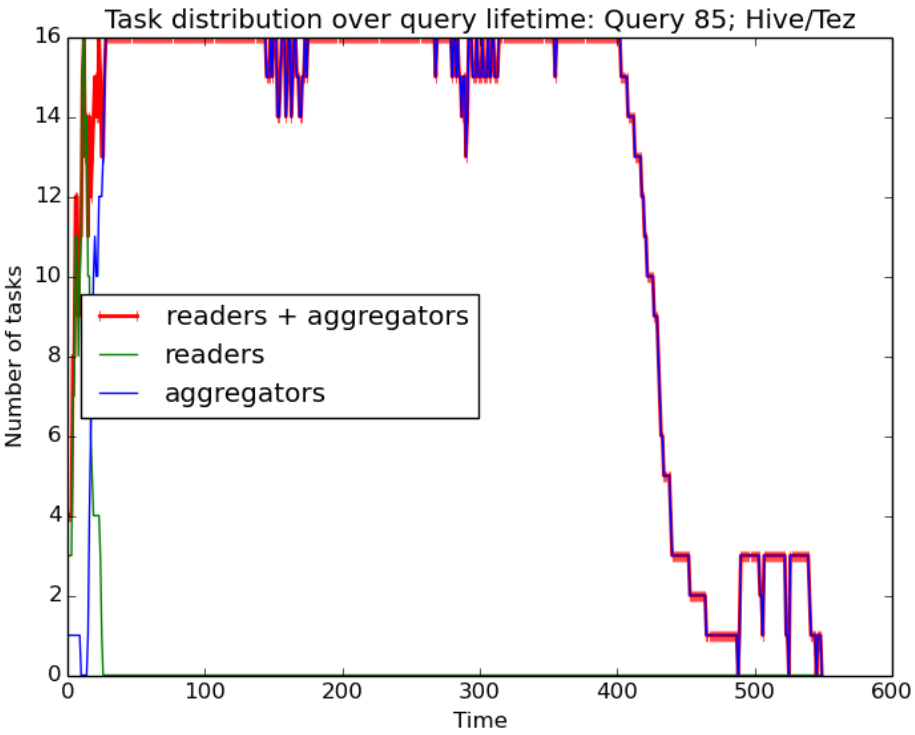


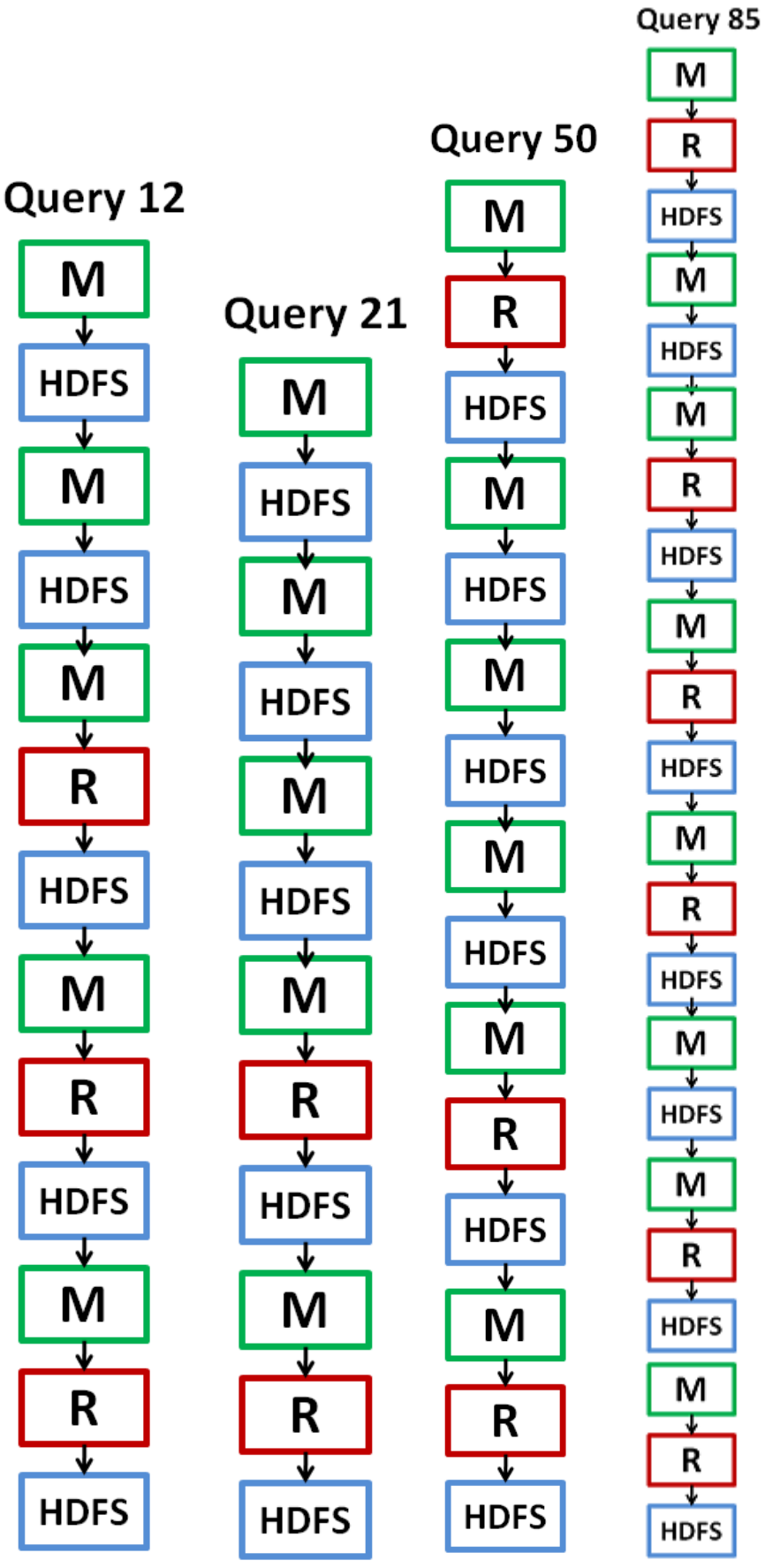
Fig 5e

Notes: In all of the graphs, the maximum number of tasks is 16, which is the number of vCPUs. This seems to imply that **Tez works more closely with YARN**, perhaps more dynamically / interactively, in order to perform better.

More discussion at the end of section 1.

d) DAGs

MapReduce


Query 12
M
HDFS
M
HDFS
M
R
HDFS
M
R
HDFS
M
R
HDFS
Query 21
M
HDFS
M
HDFS
M
HDFS
M
R
HDFS
M
R
HDFS
Query 50
M
R
HDFS
M
HDFS
M
HDFS
M
HDFS
M
R
HDFS
M
R
HDFS
Query 85
M
R
HDFS
M
HDFS
M
R
HDFS
M
R
HDFS
M
R
HDFS
M
R
HDFS
M
R
HDFS

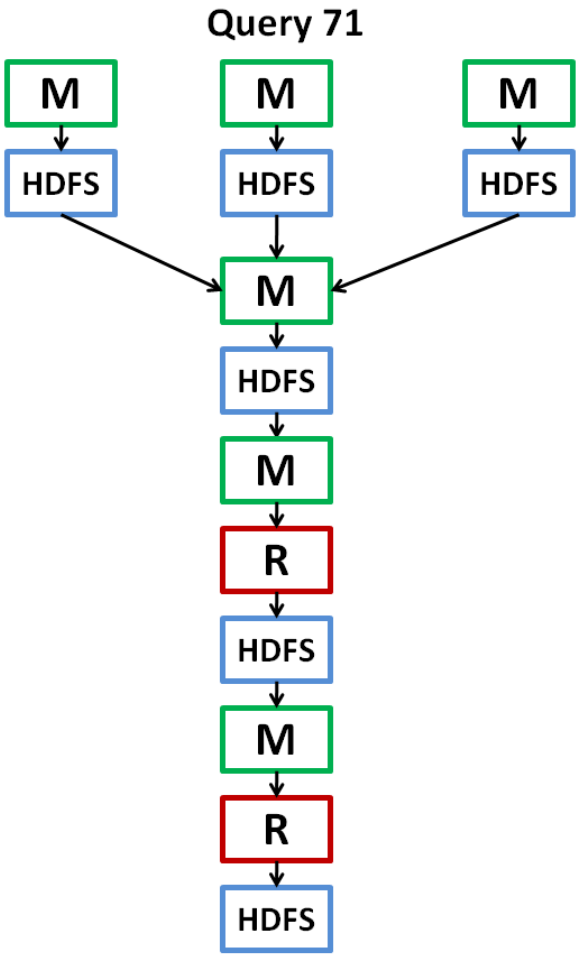


Fig 6. MR Sequences

For 71, stage 2 does a regular read from HDFS.

Tez

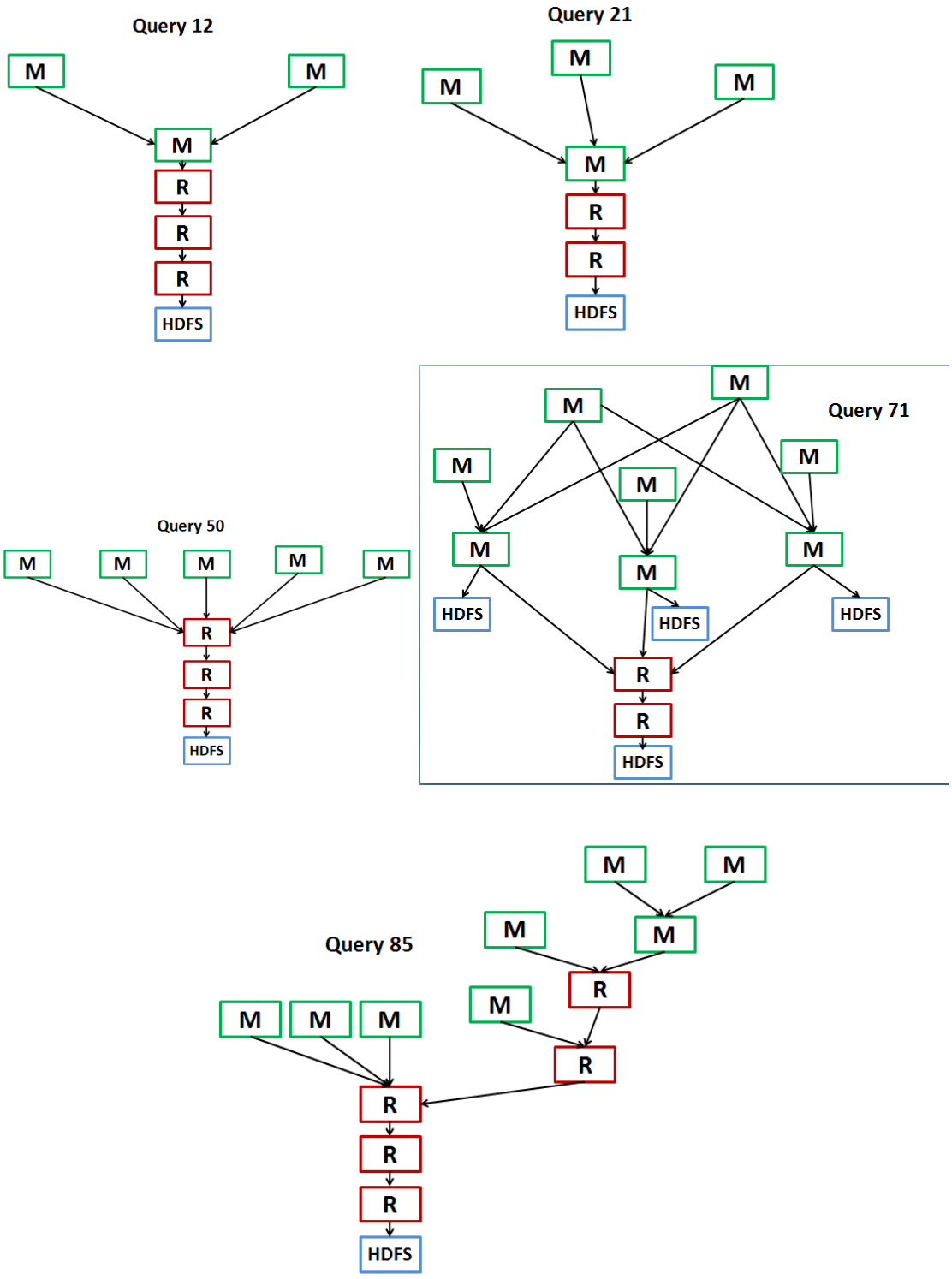


Fig 7. Tez DAGs

1 a,b,c,d. Discussion:

- Single application, so the overhead of creating, submitting, and scheduling multiple MR applications is avoided in Tez.
  E.g. The Task distribution over query lifetime of Hive/MR - 12, 71, 85 very clearly show the stages, with sizeable periods of inactivity in between.
- Query operations do not have to be artificially converted to the Map Reduce programming model, the DAG is a more general model.
  - One point to note is that HDFS (persistence + reliability) is NOT enforced between stages.
- Another effect of the DAG vs MR model is that lesser reduce tasks can happen, which can mean less data shuffling.
- It appears that Tez works with / utilizes YARN better than MR does.
  E.g. Number of tasks in Tez never exceeds the number of vCPUs (16).
- Container Reuse is enabled by default in Tez.
- Query 85, 71 – anomaly due to the structure of the tasks, and amount of data written being much higher. (Tez performs slightly worse for 71, significantly worse for 85, relative to MR)
  - Aggregators / Data flow is much more in the DAG of these queries, as can be seen in Fig 5d, 5e.
    - Query 85 reads, writes 2GB more over the network
    - Query 71 writes 55 GB more to disk, Query 85 120 GB more.
  - Query 71 involves UNION and JOIN operations, while the others do not.
  - Query 85 has a lot of disjunction of conjunctions and vice versa (ORs of ANDs)
  - The other queries have a single point where one aggregator gets data from multiple readers, but these queries have multiple layers (a hierarchy) of aggregations.
  - Potentially interesting point :
    - For stage 2 onwards of MR/85, the number of tasks is **lesser** than the number of available cores.
    - Tez/85 on the other hand, utilizes all the 16 vCPUs for as long as possible, until the end.
    - However, it is probably the case that the CPU is not the bottleneck here, and by trying to maximize CPU utilization, we increase contention on the other resources.
  - Perhaps Tez tries to keep shared objects in memory, which does more harm than good? If said object is large, and hence paged out. No solid data on this, just a thought.

2. Parameter Tuning

The parameter values set can be seen in the log files in HDFS for verification.

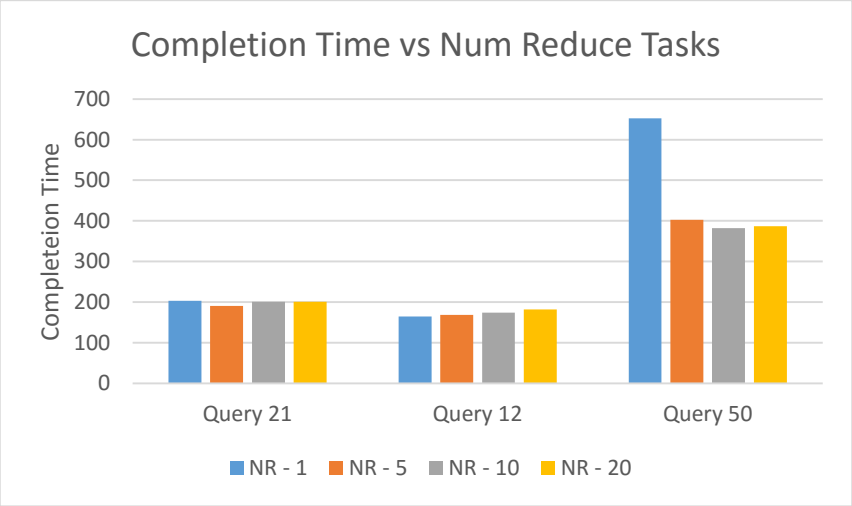
- MR - conf.xml, or in the tez-history file.

Abbreviations :

- NR – Number of Reducers
- PC – Parallel Copies
- CM – Completed Maps

a) MapReduce

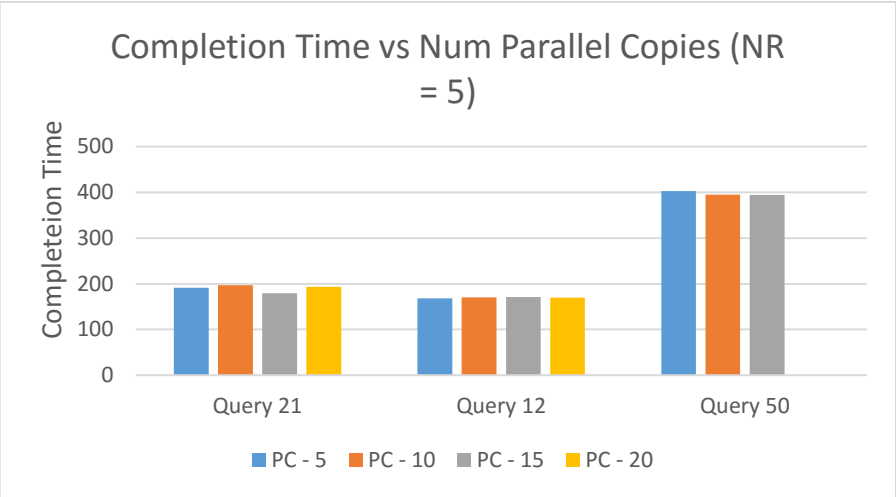
 # mapred.reduce.tasks = (1 | 5 | 10 | 20)



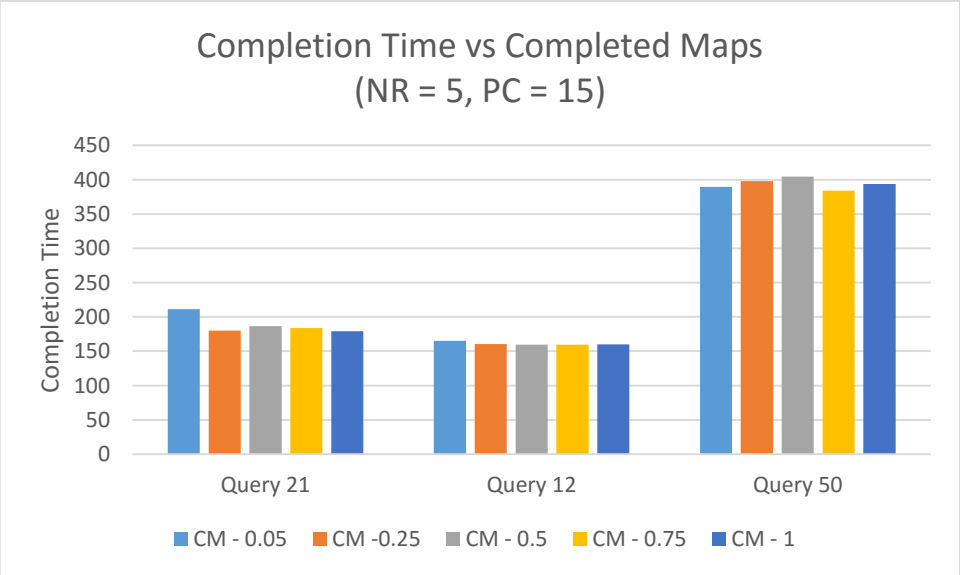
- Sets the default number of reduce **tasks** per **job**.
- Increasing this can improve performance if there is **insufficient parallelism** as the number of tasks that have been configured is lesser than the available slots.
- However, if the number of reduce jobs outnumbers the number of available slots, performance will not see significant gains.

Performance and parameter selection – please see section after all 3 params are selected.

 # mapreduce.reduce.shuffle.parallelcopies = (5 | 10 | 15 | 20)
Sets the default number of parallel transfers run by **reduce** during the copy(**shuffle**) phase.



 # mapreduce.job.reduce.slowstart.completedmaps = (0.05 | 0.25 | 0.50 | 0.75 | 1)



- MR - Fraction of Map tasks to complete before scheduling a Reduce task.
- If the Reduce task is not commutative, associative, reduce will have to wait for all the map tasks to complete, even if it is scheduled.

Discussion

		Time to complete				Network Data Read				Network Data Sent		
PC - 5	Paramete	**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50
CM - 1	NR - 1	202.964	164.526	652.449		**2245**	2884	8037		**2159**	2767	7781
	NR - 5	190.984	168.392	402.34		3178	2163	9674		3067	2089	9354
	NR - 10	**200.802**	174.084	382.27		**2408**	2388	8177		**2320**	2297	7928
	NR - 20	**200.86**	181.763	387.113		2031	2191	8977		1958	2108	8676
NR - 5	Paramete	**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50
	PC - 5	190.984	168.392	402.34		3178	2163	9674		3067	2089	9354
	PC - 10	196.805	170.293	395.182		2099	2026	9229		2017	1945	8946
	PC - 15	179.348	170.965	393.918		3382	2077	6748		3256	1998	6541
	PC - 20	193.375	169.729	-		3169	2395	-		3061	2301	-
NR - 5	Paramete	**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50
PC - 15	CM - 0.05	211.563	164.984	389.456		2517	2245	8795		2426	2155	8464
	CM -0.25	180.213	160.561	397.843		2198	2515	7239		2113	2411	6966
	CM - 0.5	186.674	159.539	404.678		3312	2435	8389		3191	2333	8109
	CM - 0.75	183.961	159.784	383.806		2586	2378	9489		2488	2284	9191
	CM - 1	179.348	159.91	393.918		3382	2077	6748		3256	1998	6541

- For Query 21, NR = 5 performed best. However, I do not believe this to be a very statistically significant claim, as the variation between the various values is tiny, and could be less than the variation between runs.
- This value is not the best value for other runs. More-over, network utilization is much higher for NR=5.
- That said, from the end-user perspective, we believe that run-time is the best indicator. Even though more data is read/written to the network, the network bandwidth is high enough for it to not result in worse performance. We use this logic for all our parameter selection.
    - We are also ignoring cumulative CPU time, which might matter.
- In most cases, storage usage is not significantly different, hence not adding those columns.
- Ie, we assume the user cares more about time than money.
- Highly recommend reading the above table, data speaks significantly.
- NR = 5
    - Might not be statistically significant
    - On a side by side comparison of NR = 1 and NR = 5,
        - Cumulative CPU time is worse.
        - The last two stages perform more work, as the 2nd last stage has 5 reducers instead of one, last stage has 4 mappers instead of one. (Interesting point! – one per machine)
- PC = 15
    - This definitely seems statistically significant.
    - Compared with logs for PC = 5 and PC = 20
    - Cumulative time is also less here
    - This also corresponds closely to the number of vCPUs, though this might not be causation.
    - Simply put, parallel copying seems to speed things up.
    - 15 might just be a right fit for this cluster size, given the 16 vCPUs, etc.

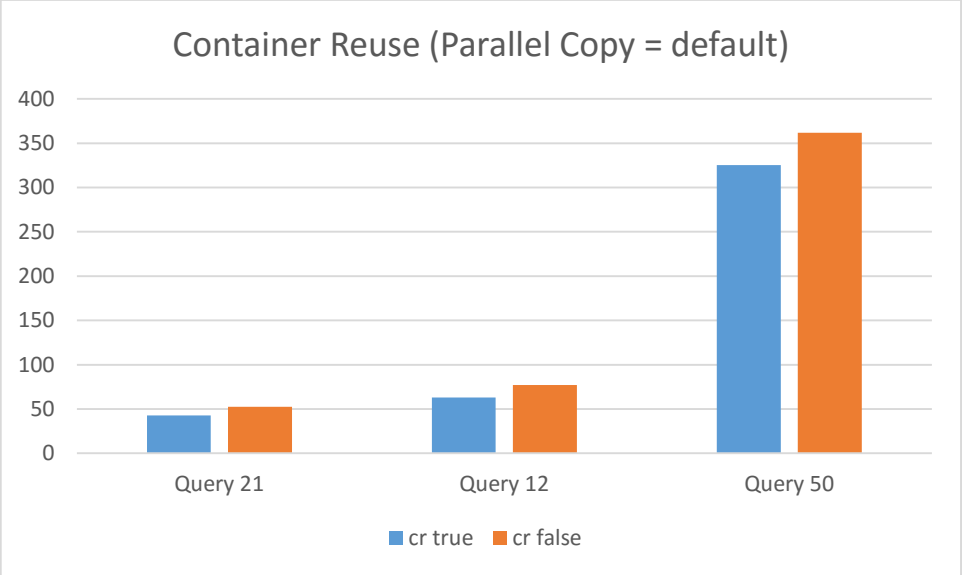
- CM = 1 (Completed Maps)
    - Only the last two stages have both Maps and Reduces
        - We have artificially set the number of Reducers in the 4th stage to 5.
    - Query 21 has both an **order by,** and a **limit 100**
    - Thus, at the last, it needs all the data to be computed, and ordering needs to be done, no matter what intermediate sequence of stages takes place.
    - Hence, starting eagerly and then processing additional data might actually hurt the reduce process start as it starts it's merge step without some of the data if cm != 1.

b) Tez

Abbreviations :

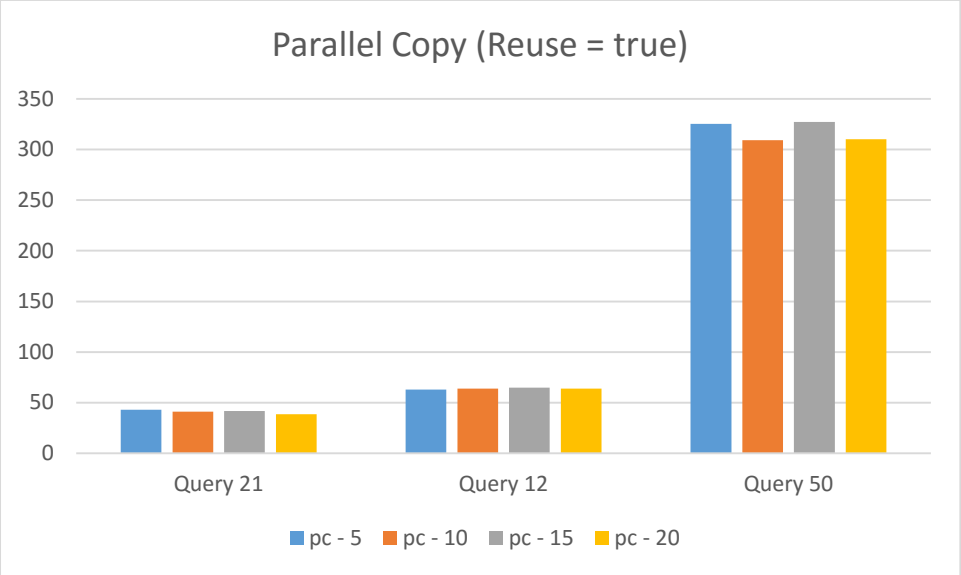
- CR – Container Reuse
- PC – Parallel Copies

 # tez.am.container.reuse.enabled=(true|false)



- Fairly straightforward, expected result that container reuse improves the performance.
- The overheads of allocating and deallocating containers is avoided.
- Side benefits – sessions, shared objects, etc.

 # tez.runtime.shuffle.parallel.copies=(5|10|15|20)



c) Use the best values from a. and b. for queries 12 and 50.

Tez

	Time				N/w Read MB				N/w Write MB		
Param	Query 21	Query 12	Query 50		Query 21	Query 12	Query 50		Query 21	Query 12	Query 50
cr true	42.867	62.989	325.233		129	990	5439		125	952	5280
cr false	52.633	76.953	361.775		129	1220	5901		124	1174	5717

	Time				N/w Read MB				N/w Write MB		
Param	Query 21	Query 12	Query 50		Query 21	Query 12	Query 50		Query 21	Query 12	Query 50
pc - 5	42.867	62.989	325.233		129	990	5439		125	952	5280
pc - 10	41.149	63.792	309.383		241	676	5478		233	651	5302
pc - 15	41.666	64.858	327.151		362	1162	5705		350	1118	5556
pc - 20	38.645	63.98	310.068		54	864	5707		53	832	5524

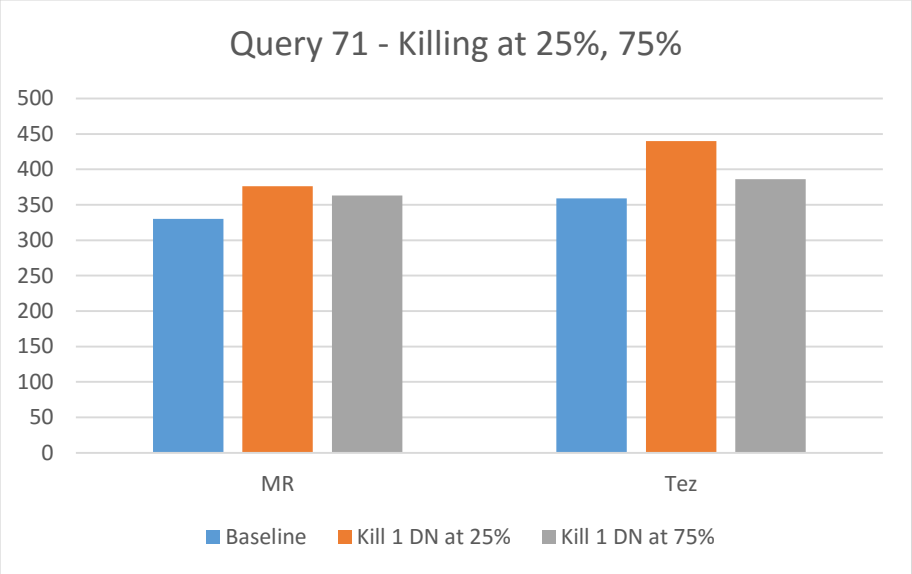
- Container reuse seems to be a universally good idea.
- Parallel Copy values seem statistically significant, but the right value varies by query.

MapReduce

- Query 12 appears to be a sufficiently small, well optimized MR application with a small number of stages, and hence, no parameter significantly changes its performance.
    - All deltas are are tiny changes.
- Query 50's first stage has a large MR application, hence NR seems to affect it more strongly.
    - Other parameter values do not show significant variation (one or two %)

		Time to complete				Network Data Read				Network Data Sent		
PC - 5	Paramete	**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50
CM - 1	NR - 1	202.964	164.526	652.449		**2245**	2884	8037		**2159**	2767	7781
	NR - 5	190.984	168.392	402.34		3178	2163	9674		3067	2089	9354
	NR - 10	**200.802**	174.084	382.27		**2408**	2388	8177		**2320**	2297	7928
	NR - 20	**200.86**	181.763	387.113		2031	2191	8977		1958	2108	8676
NR - 5	Paramete	**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50
	PC - 5	190.984	168.392	402.34		3178	2163	9674		3067	2089	9354
	PC - 10	196.805	170.293	395.182		2099	2026	9229		2017	1945	8946
	PC - 15	179.348	170.965	393.918		3382	2077	6748		3256	1998	6541
	PC - 20	193.375	169.729	-		3169	2395	-		3061	2301	-
NR - 5	Paramete	**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50		**Query 21**	Query 12	Query 50
PC - 15	CM - 0.05	211.563	164.984	389.456		2517	2245	8795		2426	2155	8464
	CM -0.25	180.213	160.561	397.843		2198	2515	7239		2113	2411	6966
	CM - 0.5	186.674	159.539	404.678		3312	2435	8389		3191	2333	8109
	CM - 0.75	183.961	159.784	383.806		2586	2378	9489		2488	2284	9191
	CM - 1	179.348	159.91	393.918		3382	2077	6748		3256	1998	6541

Question 3.



- Since we set the enable speculative execution, multiple instances of some map/reduce tasks may be executed in parallel.
- Hence the delay / effect of losing one datanode would probably be higher with speculative execution turned off.
- Killed only DataNode, not the Tez or the MR execution workers.
    - Only storage is affected, not compute.
- Query 71 – MR just performs map tasks for almost 80% of its lifetime, so only reading is affected.
- Query 71 – Tez - aggregators are running for most of the lifetime, and the storage write in MR is around 25x more than Tez, so I guess the DAG model differs significantly. Write operations are affected too, instead of only read operations.

- Also, Tez uses **less network and more storage than MR** for Query 71, so perhaps the data-local partitioning scheduling is done better on Tez. This would also explain why Tez sees a bigger delay when the DataNode is killed at 25%, as the Tez child worker will switch to remote I/O, or, dynamic updating happens, but has overhead. Remote write happens anyway due to the Replication, but now two remote writes have to happen instead of just one. Also note that local storage writes are humongous in Query 71, almost 25x the MR storage written.
- Reading is probably affected more, on a per MB basis, but the writes probably make the difference between MR and Tez larger.