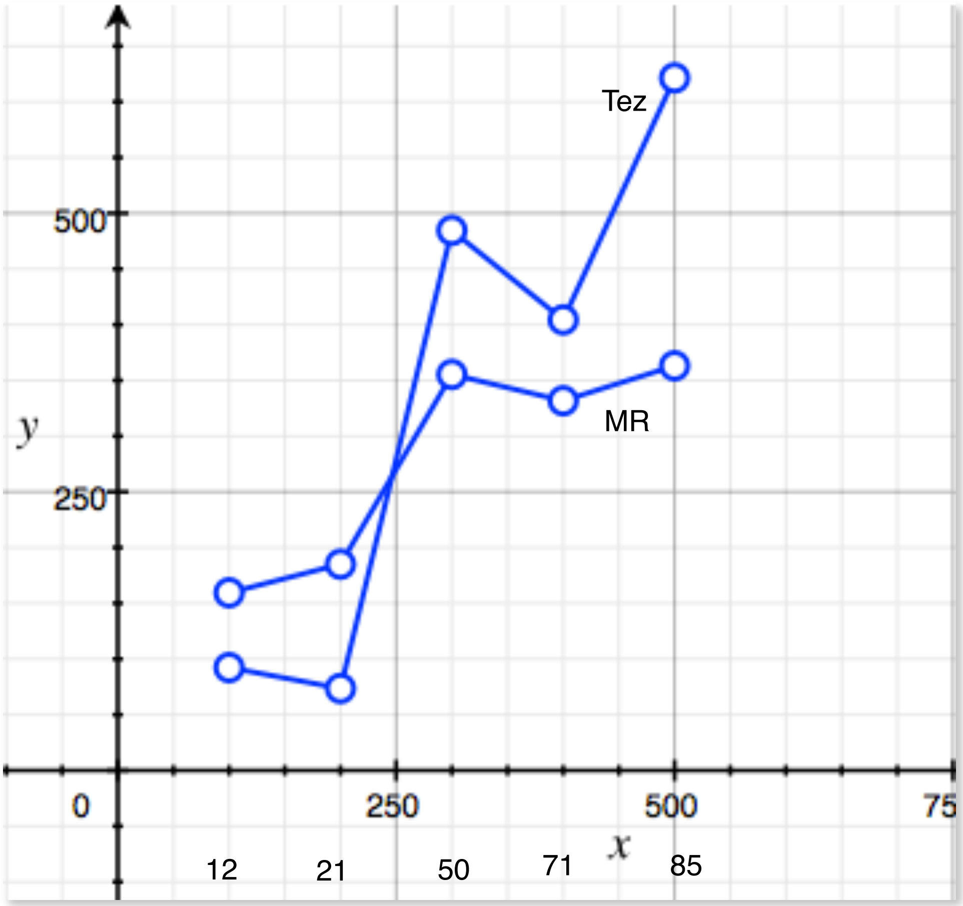
1a. Query completion time



y: query completion time for MR and Tez
x: query number

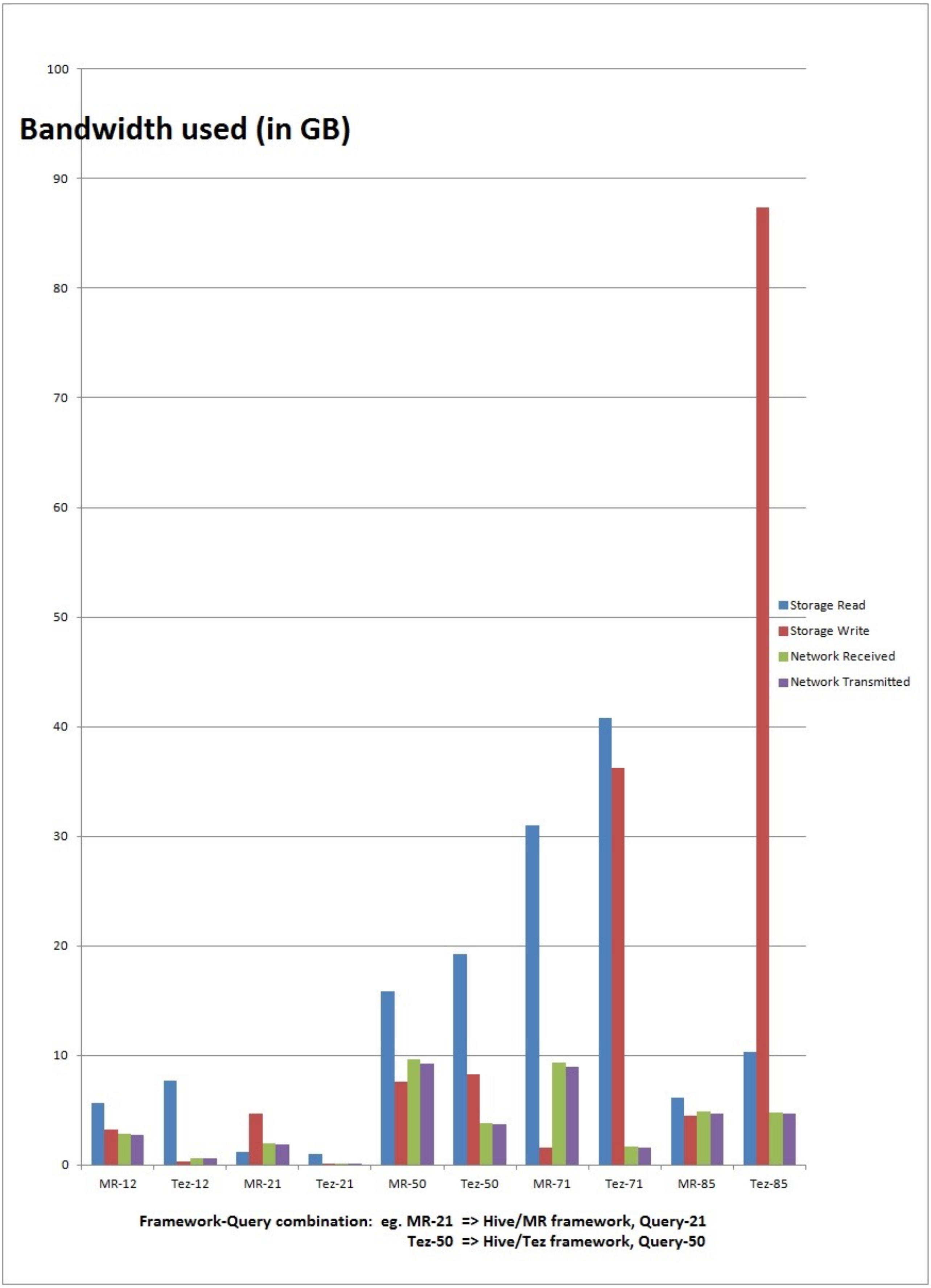
Time taken per Query(in sec)

	12	21	50	71	85
MR	159.414	185.054	355.315	331.728	363.102
Tez	92.256	73.577	484.605	404.347	621.444

Observation :
Hive/Tez is not always better than Hive/MR as shown in the above spreadsheet and visualised in the plot above.

Reason:
Above result seems to be unexpected as Tez is supposed to work faster on SQL Queries. In some cases Tez works faster only after tuning certain parameters (which we will see in further questions) whereas MR works good even if certain parameters are not tuned, which could be the possible reason for the results shown above. We are using default parameter values in above case.

1b. Amount of network/storage read/write bandwidth used during the query lifetime



Observations and reason for Anamoly

Network read and write is less in Tez compared to MR because unlike MR, Tez does not write to HDFS for every reducer.
Tez performs local reads, so it does not access HDFS every time.
In Query 71 however, Tez could not perform better than MR in this regard and the reason could be complicated SQL Query so that Tez performed some trade-off on time vs read/write.

Except for query 12 and 21, both storage read/write is higher with Tez.
So the general trend is that Network read and write is less in Tez but storage read/write is higher with Tez.

1c. Task Statistics

Query 12

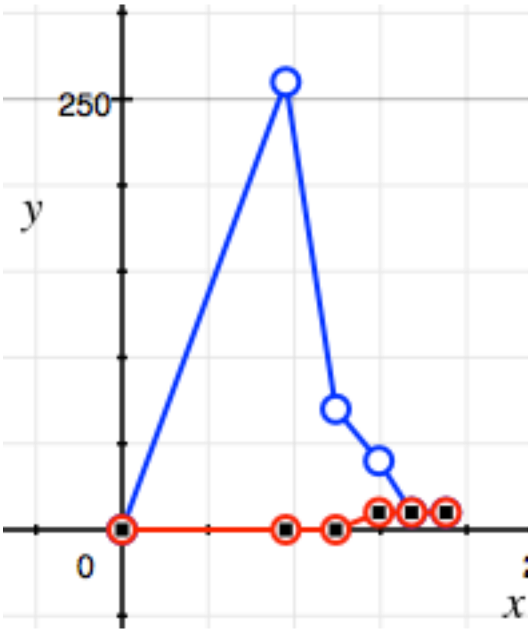
Hive/MR total number of tasks; ratio of tasks(aggregator/reader) : 42 ; 3/39
Hive/Tez total number of tasks; ratio of tasks : 55 ; 3/52

MR

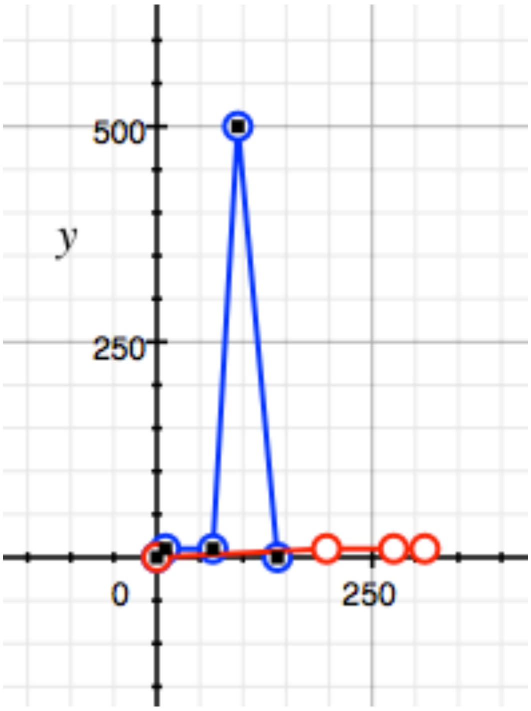
Time from start (scaled, not actual)	95	124	149	168	188	Total
Readers	26	7	4	1	1	39
Aggregators	0	0	1	1	1	3

Tez

Time from start (scaled, not actual)	10	65	94	197	275	311	Total
Readers	1	1	50	0	0	0	52
Aggregators	0	0	0	1	1	1	3



MR



Tez

Read Aggregators

Query 21

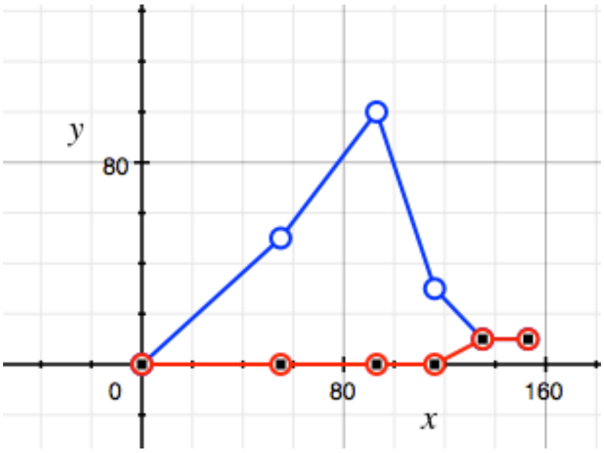
Hive/MR total number of tasks; ratio of tasks : 22 : 2/20
Hive/Tez total number of tasks; ratio of tasks : 30: 27/3

MR-1

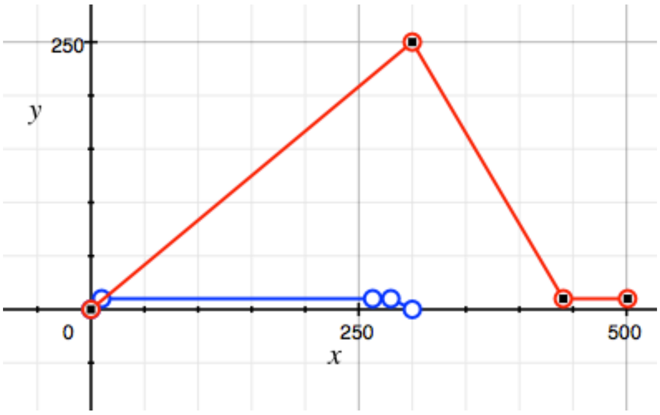
Time from start (sec)	55	93	116	135	153	Total
Readers	5	10	3	1	1	20
Aggregators	0	0	0	1	1	2

Tez-1

Time from start (sec)	10	263	280	300	441	501	Total
Readers	1	1	1	0	0	0	3
Aggregators	0	0	0	25	1	1	27



MR



Tez

Read

Aggregators

Query 50

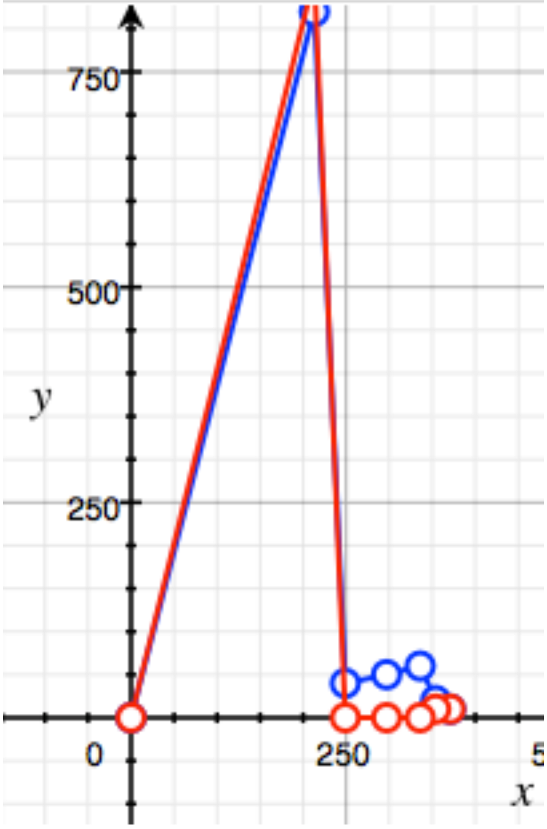
MR-2

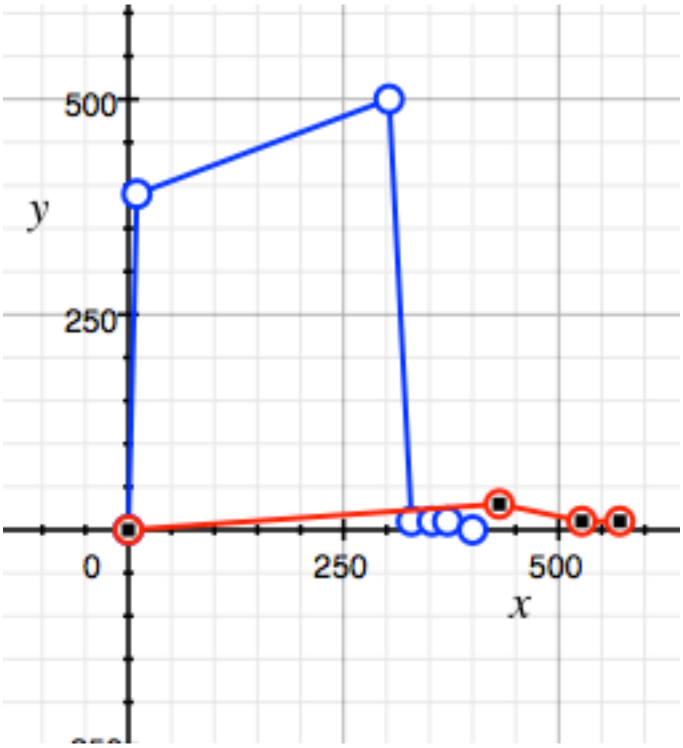
Time from start (sec)	213	249	297	336	354	371	Total
Readers	82	4	5	6	2	1	100
Aggregators	86	0	0	0	1	1	88

Tez-2

Time from start (sec)	10	303	329	353	371	431	527	571
Readers	39	50	1	1	1	0	0	0
Aggregators	0	0	0	0	0	3	1	1

Hive/MR total number of tasks; ratio of tasks : 188 : 88/100
Hive/Tez total number of tasks; ratio of tasks : 97: 5/92





Query 71

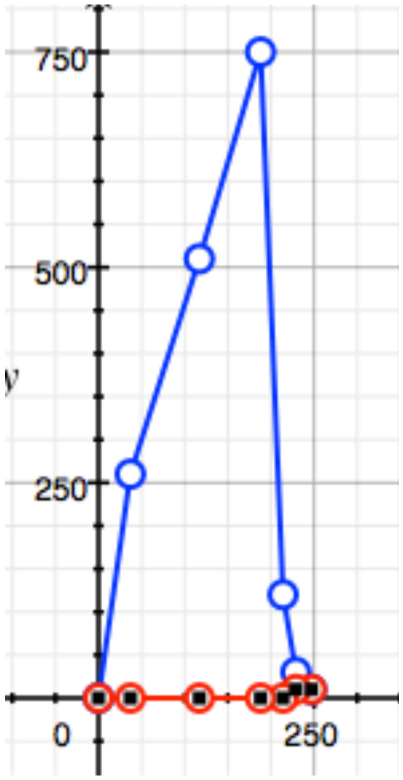
Hive/MR total number of tasks; ratio of tasks : 170: 2/168
Hive/Tez total number of tasks; ratio of tasks : 152: 147/152

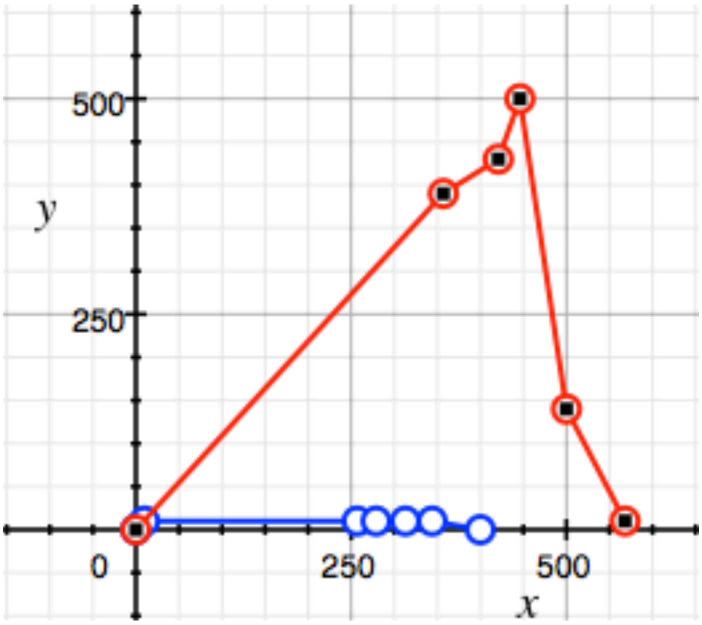
MR-3

Time from start (sec)	37	117	188	214	229	247	Total
Input	26	51	75	12	3	1	168
Aggregators	0	0	0	0	1	1	2

Tez-3

Time from start (sec)	10	257	279	313	344	357	421	446	500	568	Total
Input	1	1	1	1	1	0	0	0	0	0	5
Aggregators	0	0	0	0	0	39	43	50	14	1	147





Query 85

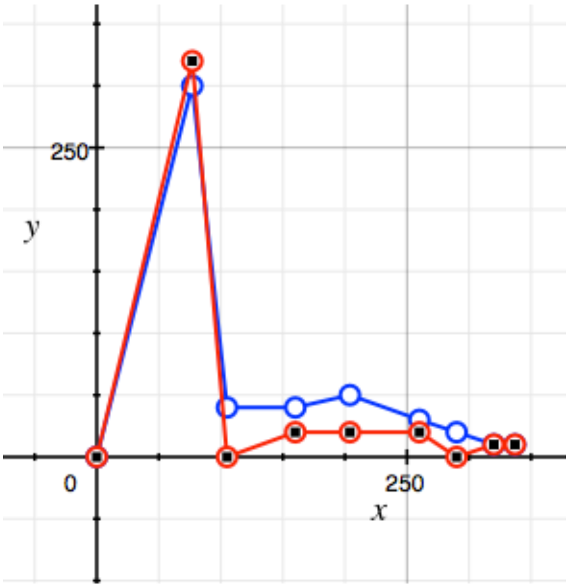
Hive/MR total number of tasks; ratio of tasks : 90 40/50
Hive/Tez total number of tasks; ratio of tasks : 83: 61/22

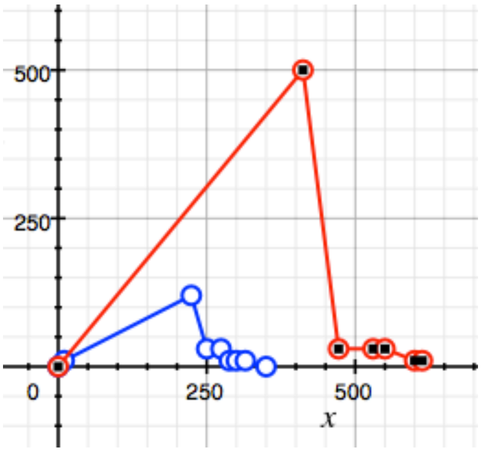
MR-4

Time from start (sec)	77	105	160	204	260	290	320	337	Total
Input	30	4	4	5	3	2	1	1	50
Aggregators	32	0	2	2	2	0	1	1	40

Tez-4

Time from start (sec)	10	224	250	274	288	300	315	412	472	530	550	600	613
Input	1	12	3	3	1	1	1	0	0	0	0	0	0
Aggregators	0	0	0	0	0	0	0	50	3	3	3	1	1





MR

Tez

Read

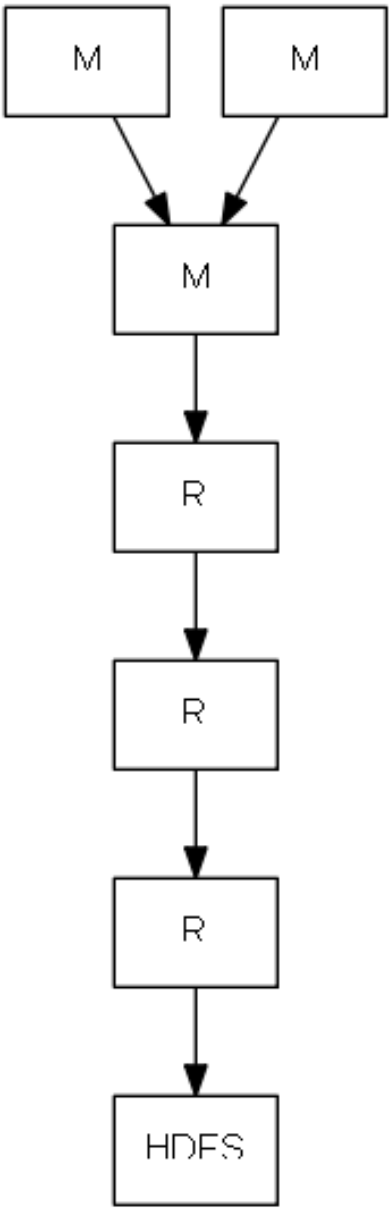
Aggregators

Is there any correlation between these metrics and performance? Why/why not?

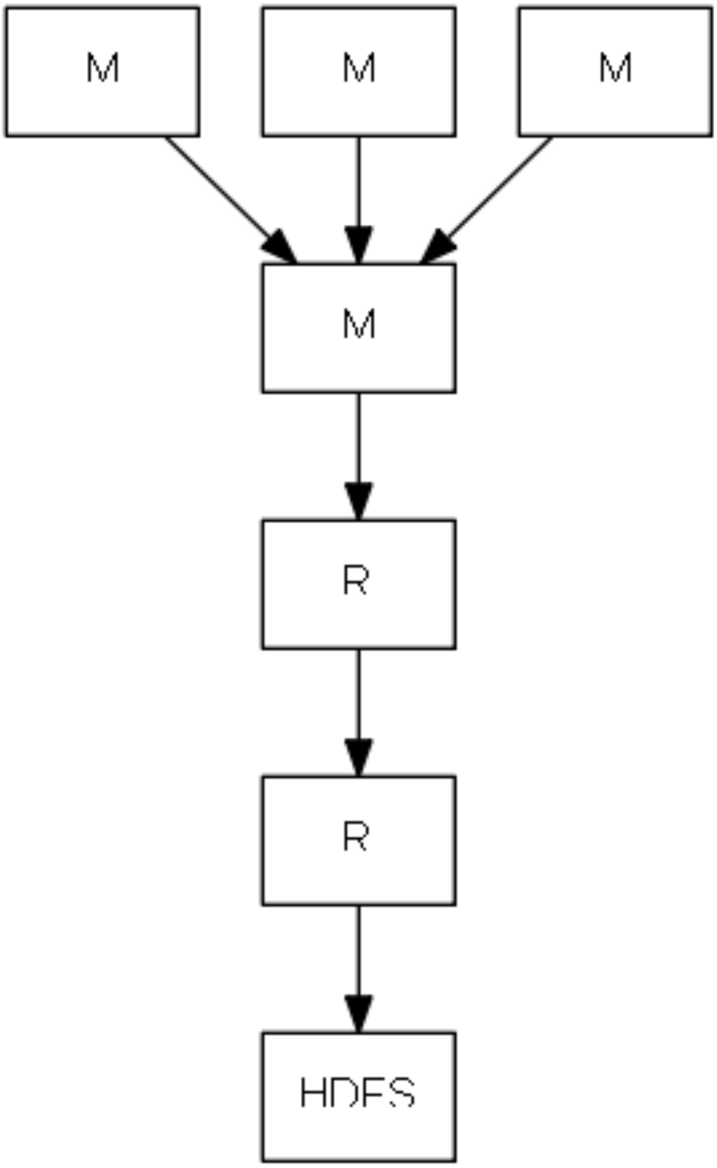
One correlation that we are able to observe from parts 1a and 1c is that larger number of tasks is leading to better time performance. In query 12 and 21, we can see Tez is issuing more tasks than MR, leading to better performance of Tez. In query 50,71 and 85, MR is issuing more tasks than Tez, leading to better performance of MR. This might not hold true in all cases but it seems the case here.

1d. DAGs

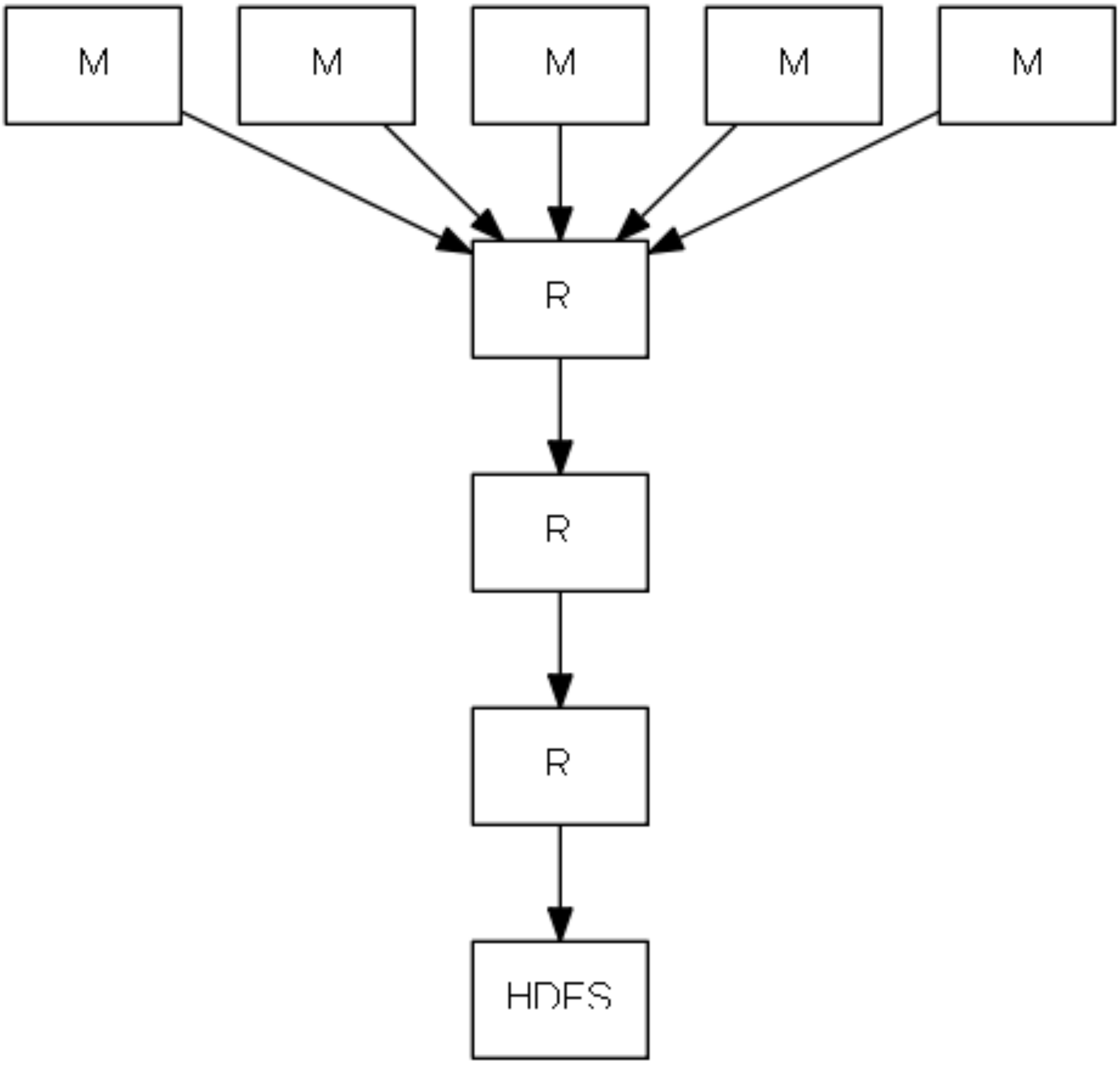
Tez



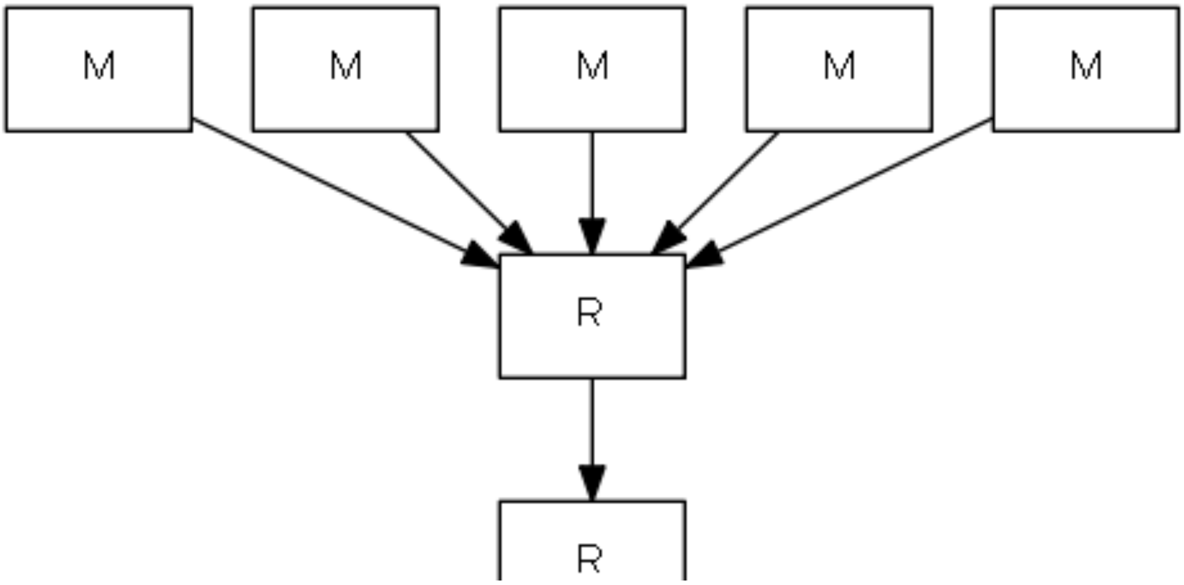
Hive/Tez query 12

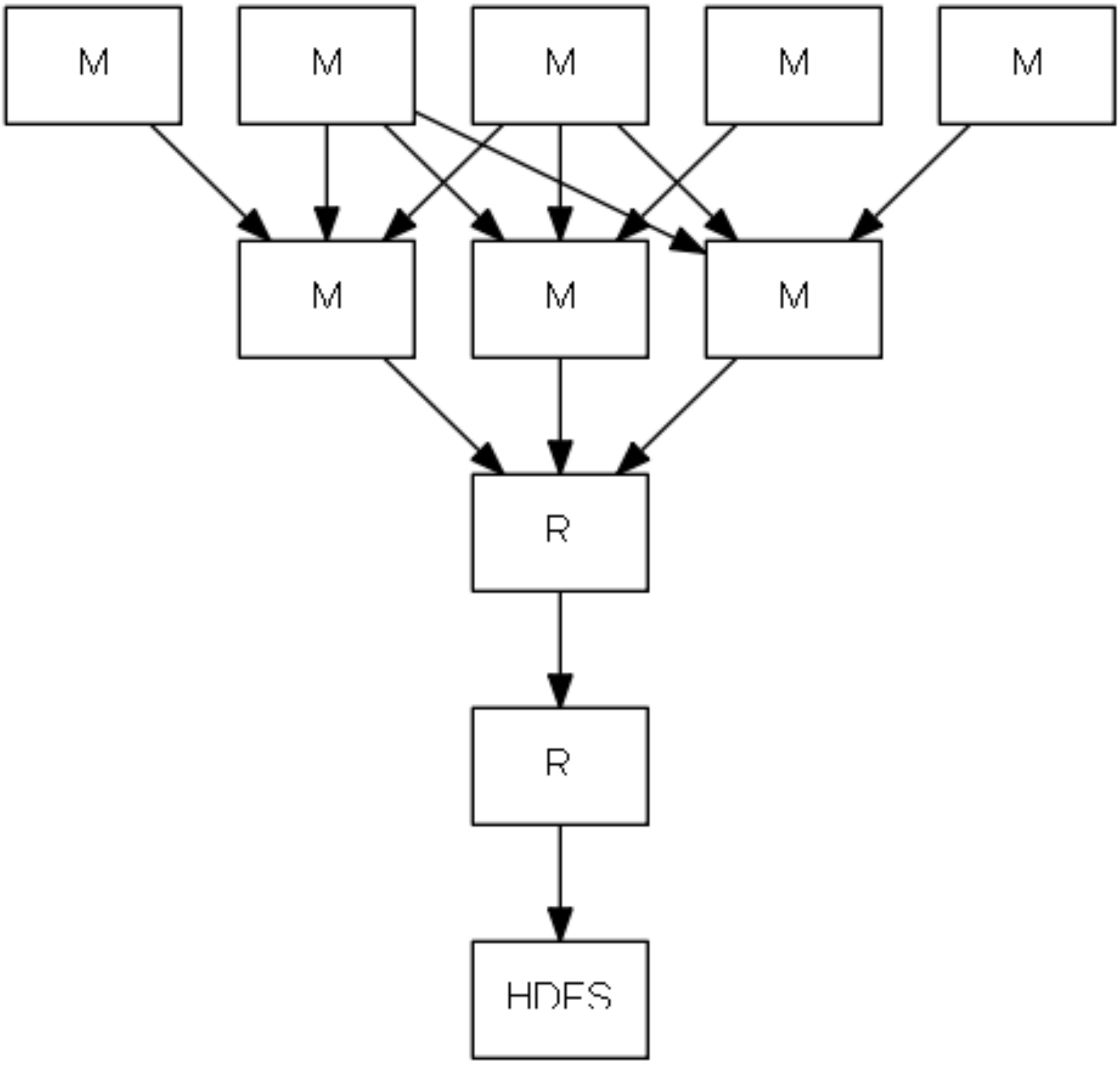


Hive/Tez query 21

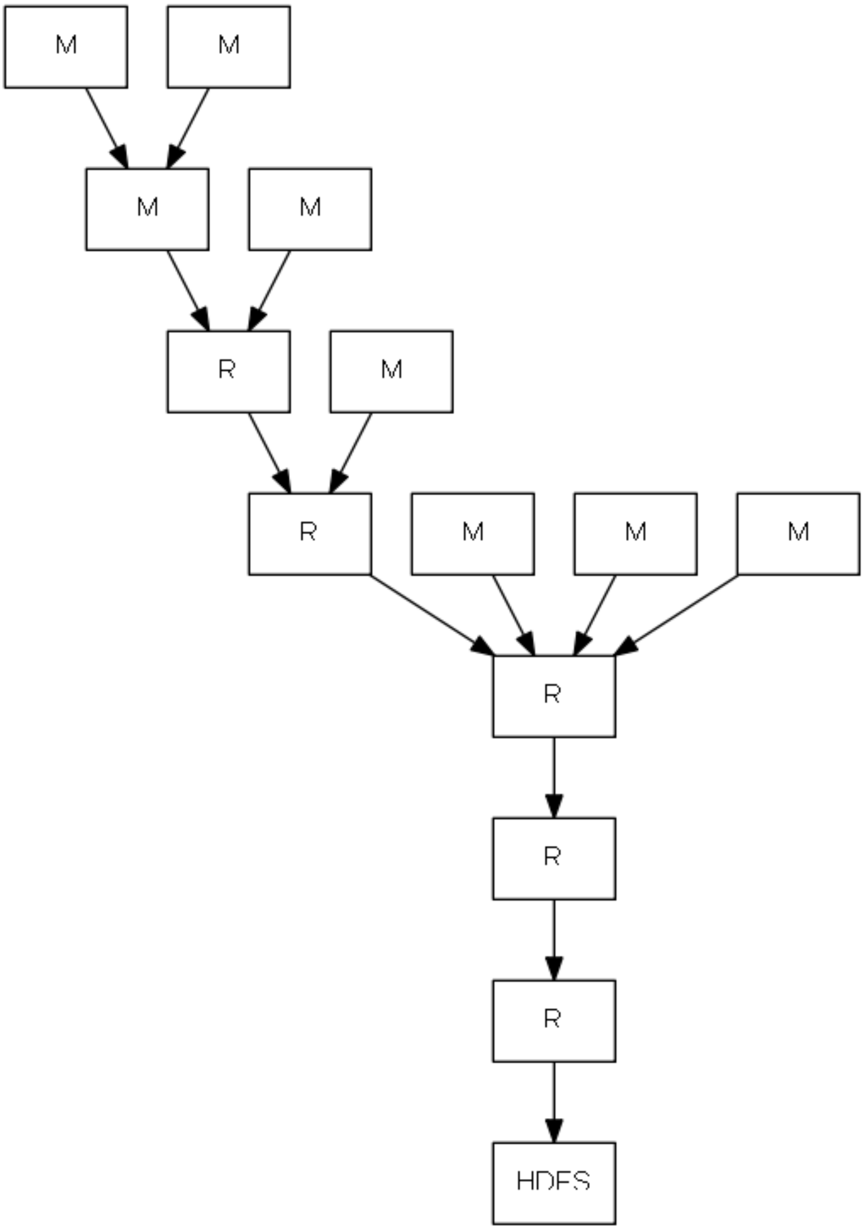


Hive/Tez query 50



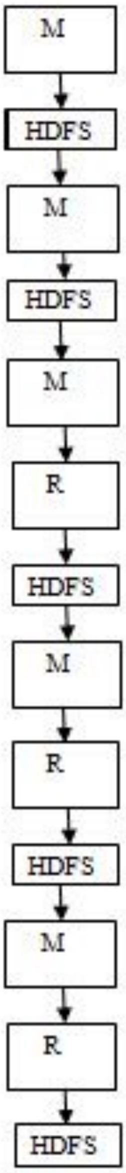


Hive/Tez query 71

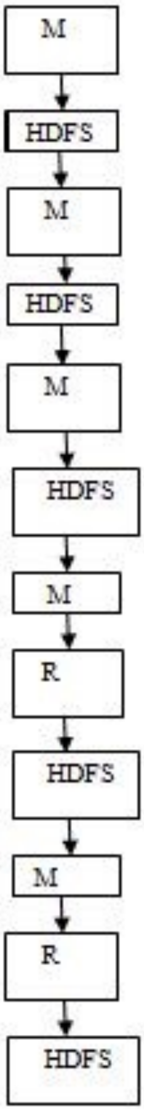


Hive/Tez query 85

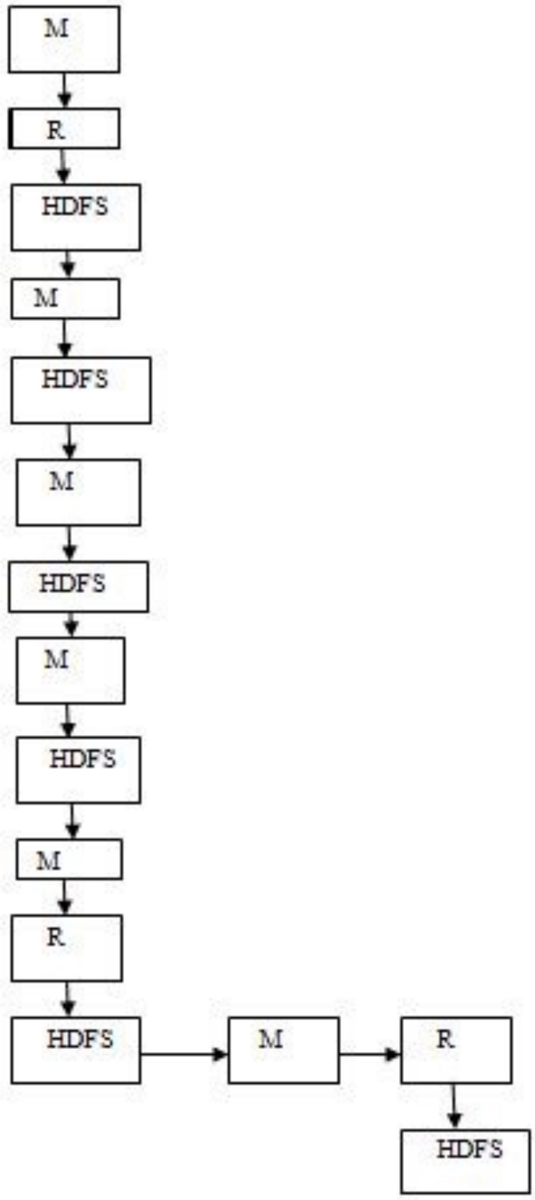
MR



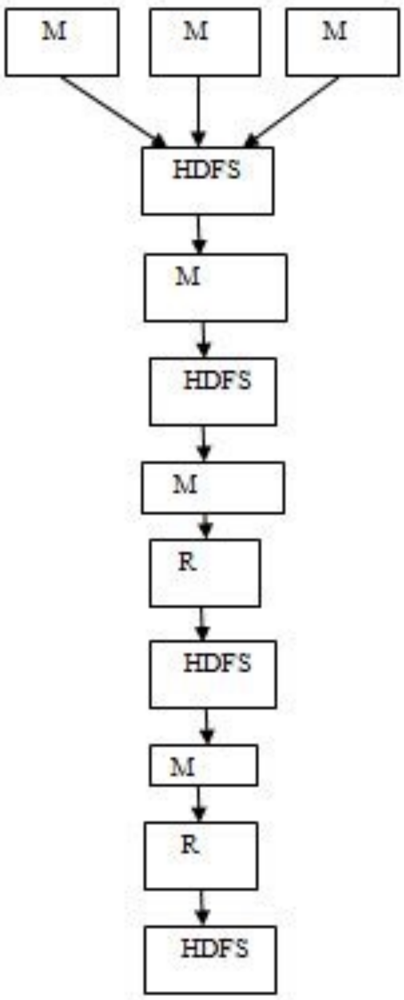
MR Query 12



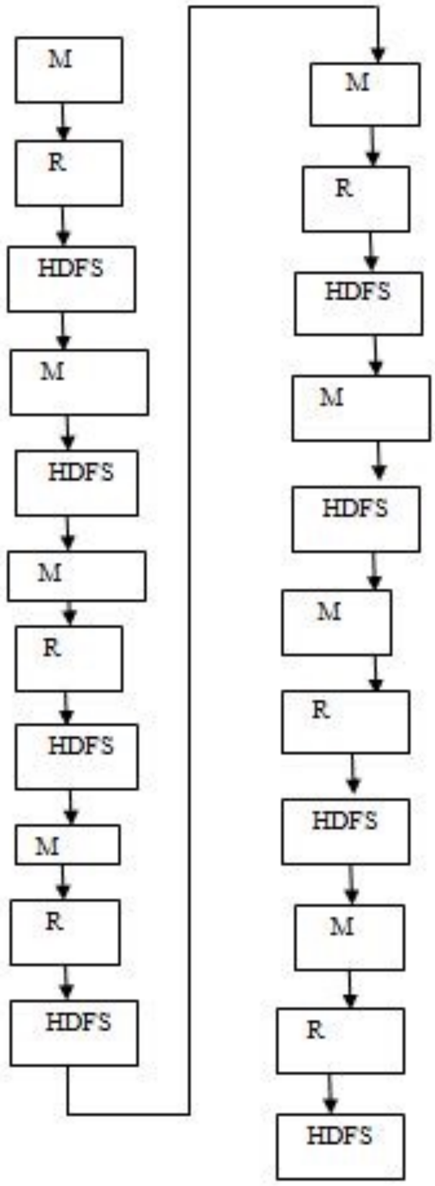
MR Query 21



MR Query 50



MR Query 71



MR Query 85

2a. Performance Variation in Hive / MR for SQL Query 21:

First , we check performance variation for
A. mapred.reduce.tasks

1	*182.053 seconds (default settings)*
5	189.459 seconds
10	186.707 seconds
20	188.937 seconds

Setting mapred.reduce.tasks =*1*(best value), we check performance variation for
B. mapreduce.reduce.shuffle.parallelcopies

10	*180.142 seconds*
15	182.468 seconds
20	183.450 seconds

Setting mapred.reduce.tasks =*1* and mapreduce.reduce.shuffle.parallelcopies = *10* (best values), we check performance variation for
C. mapreduce.job.reduce.slowstart.completedmaps

0.05	178.786 seconds
0.25	182.914 seconds
0.50	*176.927 seconds*
0.75	182.938 seconds

So by using
1. mapred.reduce.tasks =1
2. mapreduce.reduce.shuffle.parallelcopies = 10
3. mapreduce.job.reduce.slowstart.completedmaps = 0.50
We get best time as *176.927 seconds ,* an improvement over default parameter settings (*182.053 seconds)*

Explanation :

As per our graph in 1c , the number of mappers at the time when reducers were setup was 1, and optimally number of reducers should be 0.95/1.75 times the number of mappers, hence number of optimal reducers would be 1 for mappers=1,which is correct as per the experiment.

The property mapred.reduce.parallel.copies (which defaults to 5) defines how many threads are started per Reduce task to fetch Map output.
Reducers start shuffling based on a threshold of percentage of mappers that have finished.
As we can see from 1c, number of early mappers was high for this query so it is possible that number of mappers finished reading work early and so it would be optimal to start a bit higher number of threads to copy input from mappers to reducers. (but not too high as it increases NW congestion)

mapreduce.job.reduce.slowstart.completedmaps indicates when do reducers start.
Here the most efficient value turned out to be a median value of 0.5 because as we saw through previous parameter that around 10 mappers finished early so starting reducers early is a good thing in this case.
However not too early because starting reducers too early might "hog up" reduce slots while only copying data and waiting for mappers to finish. Another job that starts later that will actually use the reduce slots now can't use them.

2b. Performance Variation in Hive / Tez for SQL Query 21:

First , we check performance variation for
A. tez.am.container.reuse.enabled

false	73.577 seconds (default settings)
true	*66.853 seconds*

Setting mapred.reduce.tasks =*true* (best value), we check performance variation for
B. tez.runtime.shuffle.parallel.copies

10	*48.762 seconds*
15	55.794 seconds
20	56.587 seconds

So by using
1. tez.am.container.reuse.enabled =true
2. tez.runtime.shuffle.parallel.copies = 10
We get best time as *48.762 seconds ,* an improvement over default parameter settings (*73.577 seconds.*

Explanation:

Re-using containers has the advantage of not needing to allocate each container via the YARN ResourceManager.As happened with MR, Tez also performs better with parallel copies set to 10 as it speeds up the reading from Mappers to Reducers.

2c. Performance Variation in Hive/MR and Hive / Tez for SQL Queries 12 and 50:

Hive/MR

Query 12

By using
1. mapred.reduce.tasks =5
2. mapreduce.reduce.shuffle.parallelcopies = 20
3. mapreduce.job.reduce.slowstart.completedmaps = 0.25
We get best time as *161.219 seconds*

Explanation :
From 1c ,we can see that number of mappers at the time reducers are set up is =4, so it would be optimal to start around 4 reducers in this case. We set up mapred.reduce.tasks =5 and it performed better than default value of 1.
Going by the previous logic of a large number of early mappers, more number of parallel copies would be suited.

Query 50

By using
1. mapred.reduce.tasks =10
2. mapreduce.reduce.shuffle.parallelcopies = 15
3. mapreduce.job.reduce.slowstart.completedmaps = 1
We get best time as *326.669 seconds*
Explanation :
As we can see from 1c also, a large number of reducers are being setup in the early phase of computation itself, which conforms with the reason why mapreduce.job.reduce.slowstart.completedmaps = 1 gave optimal value.

Hive / Tez

Query 12

By using
1. tez.am.container.reuse.enabled =true
2. tez.runtime.shuffle.parallel.copies = 5
We get best time as *71.57 seconds*

Query 50

By using
1. tez.am.container.reuse.enabled =true
2. tez.runtime.shuffle.parallel.copies = 10
We get best time as *330.189 seconds*

Explanation :
Usually for Tez, setting tez.am.container.reuse.enabled =true gives optimal results while the number of parallel copies being setup may vary depending upon the number of early mappers

As we can infer, settings in 2a and 2b for Query 21 do not necessarily give the best performance for Q12 and Q50 for MR and Tez.
Reason: The SQL Queries vary a lot from each other, hence they require independent fine tuning of the various parameters.

3. Variations in query job completion time between Hive/MR and Hive/Tez on failing a DataNode at 2 instants on Query71:

Hive/MR

Normal	*476.127 seconds*
25%	*409.802 seconds*
75%	*394.097 seconds*

Hive/Tez

Normal	*381.06 seconds*
25%	*441.26 seconds*
75%	*426.03 seconds*

Reason :

As we can see that in Hive/MR , the performance is unexpectedly getting better than normal when failing a data node at 25% and 75% respectively.
The case of 25% could be when only mappers were running and the case of 75% could be when both mappers and reducers were running.
As per 1c, we can see that ,for this query 71, number of mappers was too high in MR while number of reducers were high in Tez. So one reason for this unexpected result in case of MR could be the high number of mappers running , which manage the data loss both at 25% and 75%. (Since the data node is failing when mappers are being used mostly.)

In Hive/Tez, performance is as expected since it has less number of mappers and its performance gets affected in case a data node is down.