Benchmarking HBase in Distributed Mode

Ashmit Chhabra **Nishant Saurav**

1.)Objective :

HBase is modelled after Google BigTable and is part of the world's most popular big data processing platform, Apache Hadoop. In this era of Big Data revolution, Datastores and its capabilities has taken a forefront. Will HBase be a dominant role in the competitive and fast-growing NoSQL database market? We wanted to get some insight on this issue by testing scalability of HBase for interactive social media actions which involves random reads and writes operations.
Main objective of the our project is to check impact of vertical scaling and horizontal scaling on HBase.

2.) Summary and Findings

For assignment we had benchmarked HBase on stand-alone installation which uses local file system. For project we have done single node benchmarking (pseudo-distributed mode) using a better system and optimized code. We noted considerable improvement in HBase performance after vertically scaling up even though in pseudo- distributed mode there is an extra overhead for I/O operation because HDFS sits over local file system providing extra layer of communication. The results can be found in section 3.
To check HBase performance in distributed mode. We created clusters on Google computer engine . We tested performance on two node cluster and four node cluster with varying user counts (10k , 30k and 50k) with client code saved in a separate instance. Details of these experiments can be found in section 3.

We observed that on changing from 2 node cluster to 4 node cluster there is not much change in throughput of list friend , view profile and update actions. One reason that can be attributed to this behaviour is the way social media interacts. Some keys can be very popular this might lead to one key getting lots of hits which can become bottleneck in performance. HBase is advertised to work well for random reads and writes but in case of popular keys (in BG this is decided by zipfian distribution) reads and writes will not be random any more. One solution that we can think of to this problem is using memcache with gumball technique to prevent race conditions.

Another observation that we made was that for list friend action on moving to 4 node cluster from 2 node cluster, there is a very slight increase in throughput for user count 10K and 30K but these is significant decrease in throughput for 50K user. One reason that can be attributed to is that there is no pre-splitting leading to bad load distribution and every read is going to a single node.

.

3.) Description

Single Node (Pseduo – Distributed mode)

- 8GB RAM, i-5 processor, 256 ssd.

Load(in secs)	Thread1	Thread 10	Thread 100
No Image	180	90	90
Image Size 12KB	180	90	90
Image Size 512KB	390	300	320

View Profile Throughput(actions/sec)	Thread1	Thread 10	Thread 100
No Image	4685.02	13240.92	12740.83
Image Size 12KB	3409.31	9226.02	8900.77
Image Size 512KB	517.85	831.64	633.60

Observation for single node. Increase in thread count reduces the load time but there is not much change on going from 10 thread to 100 thread, this might be because of I/O overhead of garbage collector and thrashing. View Profile action throughput increases with increase in thread but image size has negative impact of throughput.

Results of Distrubuted Node:

Used Google Compute engine instances. All instances were same with configuration as

- 3.8 GB memory, n1 Standard-1 from GCE
- 20 GB hardisk space

Experiment 1 : Load time , varying User Count and Thread Count (result in sec)

	Thread =1			Thread =10			Thread =100		
	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K
No Image	488.56	1438.08	2369.50	210.88	569.126	931.83	153.35	377.667	575.42
Image Size 12KB	504.24	1512.45	2611.84	213.15	685.576	1049.03	156.62	478.105	715.806

Table 1 : Showing results of load on 2 Node Cluster

	Thread =1			Thread =10			Thread =100		
	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K
No Image	681.76	1900	3232.9	214.9	601.4	1039.9	125.9	333.4	469.6
Image Size 12KB	677.89	2051.3	3467	226.56	760.6	1162.59	129.6	425.25	617.4

Table 2: Showing results of load on 4 Node Cluster

Observation from loading phase: Comparing table 1 and table 2 , we can see that increasing the thread count decreased the load time. Comparing a 2 Node cluster performance with a 4 node cluster, we can see that there is not much difference in load time except when thread count = 100.

Experiment 2 : View Profile (Max Execution time = 180 sec, throughput in action/sec)

	Thread =1			Thread =10			Thread =100		
	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K
No Image	1450.96	1516.31	1454.83	5299.43	5067.51	4870.65	6041.24	5376.73	5507.67
Image Size 12KB	1109.49	1095.44	1126.20	2987.04	3331.54	3306.97	2899.76	3428.19	3818.13

Table 3: Showing results of View Profile on 2-Node cluster varying user count , thread count and Image size.

	Thread =1			Thread =10			Thread =100		
	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K	User =10K	User =30K	User =50K
No Image	1677.59	1598.99	1537.21	5798.59	5542.79	4843.96	7533.03	7394.79	6656.26
Image Size 12KB	1208.62	1172.14	1248.89	3633.34	3293.29	3533.59	3818.13	3525.80	4011.95

Table 4: Showing results of View Profile on 2-Node cluster varying user count , thread count and Image size.

Observation from View Profile benchmarking: Increasing the number of user count doesn't change the throughput much. Throughput is almost same as we scale from 2 node-cluster to 4 node cluster with maximum increase observed when thread count is 100.

Experiment 3 : List Friends Action(Join behaviour) (Max Execution time = 180 sec, throughput in action/sec)

		Thread = 1			Thread = 10			Thread = 100		
		User = 10K	User = 30K	User = 50K	User = 10K	User = 30K	User = 50K	User = 10K	User = 30K	User = 50K
	10 Friends	502.093	513.07	569.06	1219.26	1154.64	1351.01	1366.98	1019.42	1701.76
	20 Friends	400.093	387.158	450.240	829.29	753.658	962.358	919.775	842.917	1114.97
No Image	50 Friends	205.433	195.447	220.290	348.688	343.530	383.812	345.81	211.211	247.095
	10 Friends	363.174	431.666	517.195	731.18	952.388	1194.85	709.324	818.12	1424.07
	20 Friends	209.679	305.686	373.421	334.832	539.965	672.406	307.785	499.412	707.21
Image Size 12KB	50 Friends	107.452	157.39	185.664	155.805	371.19	288.067	97.359	224.69	249.117

Table 5: Showing results of List Friend Action on 2-Node cluster varying user count , thread count, friends count and Image size.

		Thread = 1			Thread = 10			Thread = 100		
		User = 10K	User = 30K	User = 50K	User = 10K	User = 30K	User = 50K	User = 10K	User = 30K	User = 50K
	10 Friends	569.06	571.24	521.562	1351.01	1404.24	1262.82	1701.76	1786.24	1598.17
	20 Friends	450.240	435.65	334.720	962.358	931.37	662.993	1114.97	1049.03	656.943
No Image	50 Friends	220.290	219.42	190.449	383.812	377.52	292.784	247.095	391.85	298.758
	10 Friends	517.195	484.62	397.712	1194.85	1206.93	812.563	1424.07	1504	908.753
	20 Friends	373.421	372.65	242.287	672.406	797.47	459.913	707.21	790.78	485.503
Image Size 12KB	50 Friends	185.664	187.41	130.952	288.067	306.92	273.084	249.117	322.36	452.046

Table 6: Showing results of List Friend Action on 4-Node cluster varying user count , thread count, friends count and Image size

Observation from List friend benchmarking: Increasing the number of user count has considerable effect on list friend action. For a particular cluster , If we keep the thread count constant , increasing user count to 30K doesn't affect the throughput much but increasing user count to 50K caused noticeable drop in throughput.

If We look at performance change in going from 2 node cluster to 4 node cluster we can observe that there is a slight increase in throughput for user count 10K and 30K but these is significant decrease in throughput for 50K user. One reason we can think of is every read request is going to a same node and since it's a join behaviour between two different tables which might be at two different regions servers , this can cause network overheard. On solution to this could be to pre-split the tables so that there is equal distribution across cluster as Hbase auto-splits a region when it reaches 10GB.

Experiment 4 : Impact Of Updates (Max Execution time = 180 sec, throughput in action/sec, staleness of read ops in %)

	Thread =1			Thread =10			Thread =100		
	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K
Throughput	875.45	866.92	951.92	2871.02	3174.39	3073.45	2960.39	4553.33	4214.27
Staleness	0	0	0	0.0416	0	0.036	0.1063	0.12	0.0601

Table 7: Showing results of Symmetric Very Low Updates Action on 2-Node cluster varying user count and thread count

	Thread =1			Thread =10			Thread =100		
	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K
Throughput	866.92	916.56	880.30	3174.39	3128.91	3101.29	4553.33	4745.41	4583.71
Staleness	0	0.561	0.604	0.0	0.0411	0.0446	0.12	0.084	0.07148

Table 8: Showing results of Symmetric Very Low Updates Action on 4-Node cluster varying user count and thread count

	Thread =1			Thread =10			Thread =100		
	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K
Throughput	846.54	980.04	908.3039	2652.98	3070.455	2912.78	2752.04	3925.48	2486.09
Staleness	0.12457	0.188	0.0847	0.183	0.279	0.1674	0.2692	0.411	0.2708

Table 9: Showing results of Symmetric Low Updates Action on 2-Node cluster varying user count and thread count

	Thread =1			Thread =10			Thread =100		
	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K
Throughput	980.04	935.99	912.808	3070.455	3003.603	2993.413	3925.48	4111.03	4090.93
Staleness	0.188	0.123	0.10559	0.279	0.185	0.1392	0.411	0.281	0.2327

Table 10: Showing results of Symmetric Low Updates Action on 4-Node cluster varying user count and thread count

	Thread =1			Thread =10			Thread =100		
	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K
Throughput	726.843	750.13	777.23	2346.85	2470.05	2576.02	2492.44	3559.50	2730.646
Staleness	0.32829	0.495	0.325	0.3941	0.575	0.407	0.5537	0.70	0.55563

Table 11: Showing results of Symmetric High Updates Action on 2-Node cluster varying user count and thread count

	Thread =1			Thread =10			Thread =100		
	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K	User=10K	User=30K	User=50K
Throughput	750.13	735.04	758.8145	2470.05	2590.61	2498.258	3559.50	3810.86	3857.4528
Staleness	0.495	0.346	0.2985	0.575	0.427	0.3674	0.7012	0.565	0.494

Table 12: Showing results of Symmetric High Low Updates Action on 4-Node cluster varying user count and thread count

Observation from Update actions benchmarking :Throughput is same when we moved from 2 node cluster to 4 node cluster except in case of thread count = 100 where throughput has changed considerably. Also as the thread count, user count, rate of updates are increasing we observe there is increase in staleness reported by BG.

4.) Further Activities

1. We can try checking performance with memcache and gumball technique.
2. Decrease the size of HFile (similar to SStables of Big table) as they might cause network congestion during random reads.
3. Check HBase Performance for batch processing tasks using making use of Map-Reduce paradigm.