Salient features for a BigData Benchmark

(Dhruba Borthakur, WBDB-2012)

Recently, I was asked to write up about my vision of a BigData Benchmark. That begs the question: What is BigData? Does it refer to a dataset that is large in size, and if so, what is large? Does it refer to the type of data that such a data store contains? Shall we refer to BigData only if it does not conform to a relational schema? Here are some of my random thoughts.

Software industry professionals have started to use the term BigData to refer to data sets that are typically many magnitudes larger than traditional databases. The largest Oracle database or the largest NetApp filer could be many hundred terabytes at most, but BigData refers to storage sets that can scale to many hundred petabytes. Thus, the first and foremost chracteristics of a BigData store is that a single instance of it can be many petabytes in size. These data stores can have a multitude of interfaces, starting from traditional SQL-like queries to customized key-value access methods. Some of them are batch systems while others are interactive systems. Again, some of them are organized for full-scan-index-free access while others have fine-grain indexes and low latency access. How can we design a benchmark(s) for such a wide variety of data stores? Most benchmarks focus on latency and throughput of queries, and rightly so. However, in my opinion, the key to designing a BigData benchmark lies in understanding the deeper commonalities of these systems. A BigData benchmark should measure latencies and throughput, but with a great deal of variations in the workload, skews in the data set and in the presence of faults. I list below some of the common characteristics that distinguish BigData installations from other data storage systems.

Elasticity of resources

A primary feature of a BigData System is that it should be elastic in nature. One should be able to add software and hardware resources when needed. Most BigData installations do not want to pre-provision for all the data that they might collect in the future, and the trick to be cost-efficient is to be able to add resources to a production store without incurring downtime. A BigData system typically has the ability to decommission parts of the hardware and software without off-lining the service, so that obselete or defective hardware can be replaced dynamically. In my mind, this is one of the most important features of a BigData system, thus a benchmark should be able to measure this feature. The benchmark should be such that we can add and remove resources to the system when the benchmark is concurrently executing.

Fault Tolerance

The Elasticity feature described above indirectly implies that the system has to be fault-tolerant. If a workload is running on your system and some parts of the system fails, the other parts of the system should configure themselves to share the work of the failed parts. This means that the service does not fail even in the face of some component failures. The benchmark should measure this aspect of BigData systems. One simple option could be that the benchmark itself introduces component failures as part of its execution.

Skew in the data set

Many big data systems take in un-curated data. That means there are always data points that are extreme outliers and introduces hotspots in the system. The workload on a BigData system is not uniform; some small parts of it is are major hotspots and incur tremendously higher load than the rest of the system. Our benchmarks should be designed to operate on datasets that have large skew and introduce workload hotspots.

There are a few previous attempts to define a unified benchmark for BigData. Dewitt and Stonebraker touched upon a few areas in their SIGMOD paper. They describe experiments that use a grep task, a join task and a simple sql aggregation query. But none of those experiments are done in the presence of system faults, neither do they add or remove hardware when the experiment is in progress. Similarly, the YCSB benchmark proposed by Cooper and Ramakrishnan suffers from the same deficiency.

How would I run the experiments proposed by Dewitt and Stonebraker? Here are some of my early thoughts:

- Focus on a 100 node experiment only. This is the setting that is appropriate for BigData systems.
- Increase the number of URLs such that the data set is at least a few hundred terabytes.
- Make the benchmark run for at least an hour or so. The workload should be a set of multiple queries. Pace the workload so that the there is constant fluctuations in the number of inflight queries.
- Introduce skew in the data set. The URL data should be such that maybe 0.1% of those URLs occur 1000 times more frequently that other URLs.

- Introduce system faults by killing one of the 100 nodes once every minute, keep it shutdown for a minute, then bring it back online and then continue with process with the remainder of the nodes till the entire benchmark is done.

My hope is that there is somebody out there who can repeat the experiments with the modified settings listed above and present their findings. This research would greatly benefit the BigData community of users and developers!