Apache HBASE Do's and Don'ts

Summary

- HBase is good, but not an RDBMS or HDFS replacement
- Good configuration means good operation
- Regular Monitoring is required

When to use HBase

The most important consideration when looking at HBase is that, while it is a great solution to many problems, it is not a silver bullet. HBase is not optimized for classic transactional applications or even relational analytics. It is also not a complete substitute for HDFS when doing large batch MapReduce.

With that caveat out the way – why should you use HBase? If your application has a variable schema where each row is slightly different, then you should look at HBase. As an example, doing a modeling exercise using a standard relational schema; When you can't add columns fast enough and most of them are NULL in each row, you should consider HBase. If you find that your data is stored in collections, for example some meta data, message data or binary data that is all keyed on the same value, then you should consider HBase. If you need key based access to data when storing or retrieving, then you should consider HBase.

Supporting Services

Once you're convinced that HBase is a good fit for your application, here are some tips you need to consider when deploying it. There are a few supporting services that are important and one that's required. If you haven't looked at ZooKeeper before, now is the time. HBase uses ZooKeeper for various distributed coordination services such as master election. As HBase develops and grows it continues to rely on ZooKeeper for additional functionality, making it a key part of the system. In addition, you should have proper network services in place such as NTP and DNS. HBase depends on all nodes in the cluster having closely synchronized clocks and referring to each other consistently. Using NTP and DNS ensures that you won't run into odd behaviors when one node A thinks that the time is tomorrow and node B thinks it's yesterday. You'll also prevent situations where the master node tells node C to serve a region but node C doesn't know its own name and doesn't answer. Using NTP and DNS will save a lot of headaches as you get started.

Most important consideration when selecting HBase is to make sure you have a use case that fits. The most important thing to do when using HBase is to monitor the system. Monitoring

is key to successful HBase operations. As is the case with many distributed systems, HBase is susceptible to cascading failures. If one node starts swapping it can lose contact with the master, causing another sever to pick up the load and becoming over burdened. That second server will fail and the failure will cascade. You need to monitor the memory, the CPU, the I/O and the network latency and bandwidth on each of your HBase nodes to make sure they are operating within healthy parameters. Monitoring is the most important practice to operating a healthy HBase cluster.

HBase DON'Ts

There are also a few use patterns to avoid. For example, don't expect to use HBase as a wholesale replacement for every one of your relational databases. HBase is great at many things but it doesn't replace relational databases. For a start, it doesn't talk SQL, have an optimizer, support cross record transactions or joins. If you don't use any of these in your database application then HBase could very well be the perfect fit.

Be careful when running mixed workloads on an HBase cluster. When you have SLAs on HBase access independent of any MapReduce jobs (for example, a transformation in Pig and serving data from HBase) run them on separate clusters. HBase is CPU and Memory intensive with sporadic large sequential I/O access while MapReduce jobs are primarily I/O bound with fixed memory and sporadic CPU. Combined these can lead to unpredictable latencies for HBase and CPU contention between the two. A shared cluster also requires fewer task slots per node to accommodate for HBase CPU requirements (generally half the slots on each node that you would allocate without HBase). Also keep an eye on memory swap. If HBase starts to swap there is a good chance it will miss a heartbeat and get dropped from the cluster. On a busy cluster this may overload another region, causing it to swap and a cascade of failures.