Wiki Operacional PSafe

HBase Good Practices

Ability to Scan

A wise selection of row key can be used to co-locate related records in the same region. This is very beneficial in range scans since HBase will have to scan only a limited number of regions to obtain the results. On the other hand, if the row key is chosen poorly, range scans may need to scan multiple region servers for the data and subsequently filter the unnecessary records, thereby increasing the I/O requirements for the same request.

Also, keep in mind that HBase scan rates are about 8 times slower than HDFS scan rates. Therefore, reducing I/O requirements has a significant performance advantage, even more so compared to data stored in HDFS.

Attributes - noatime and nodiratime

If you are mounting disks purely for Hadoop and you use ext3 or ext4, or the XFS file system, we recommend that you mount the disks with the noatime and nodiratime attributes.

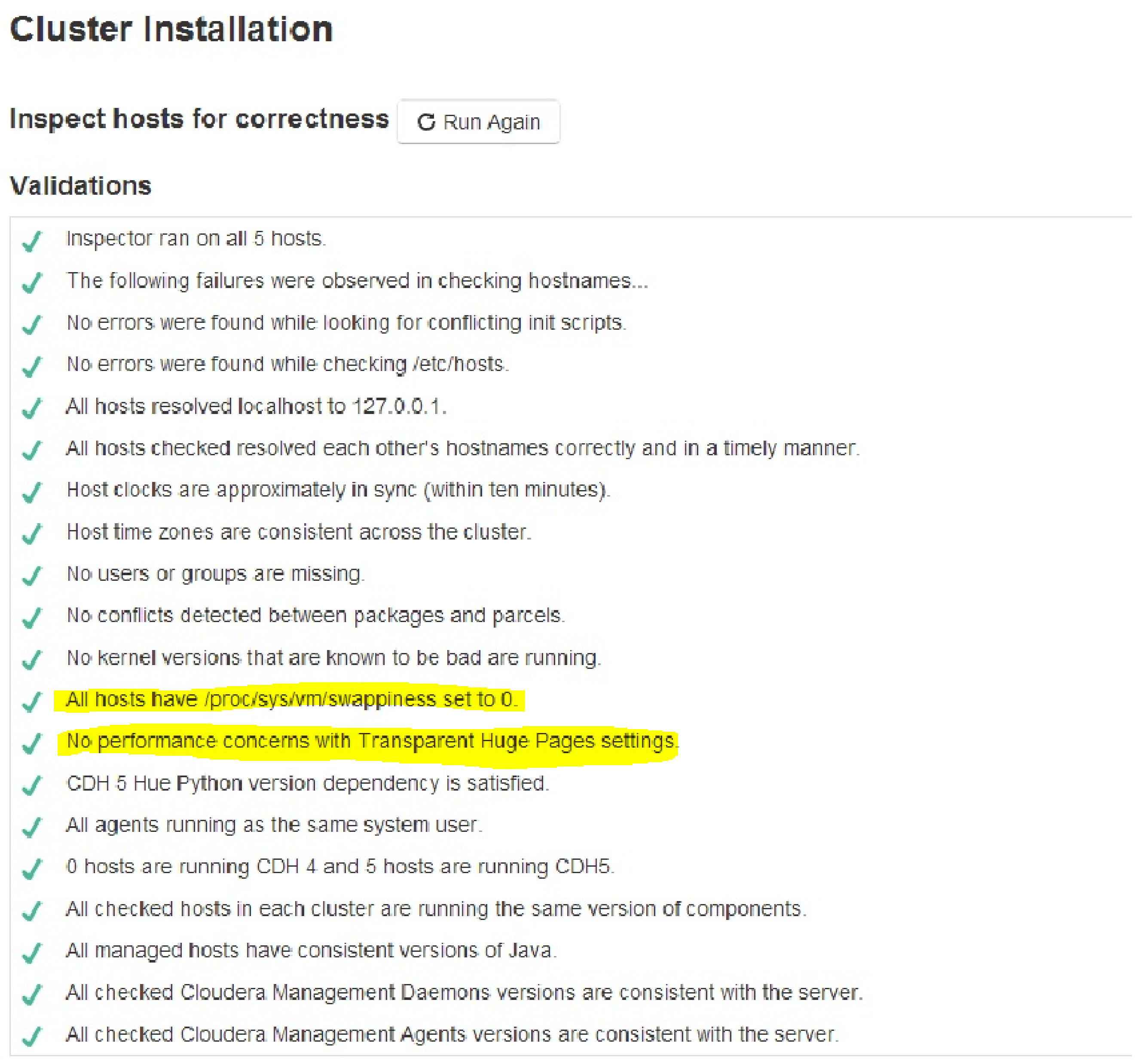
Block Cache

The block cache is a least recently used (LRU) cache that caches data blocks in memory. The idea behind the caching is that recently fetched records have a high likelihood of being requested again in the near future. However, the size of block cache is limited so it's important to use it wisely.

By default, HBase reads records in chunks of 64 KB from the disk. Each of these chunks is called an HBase block. When an HBase block is read from the disk, it will be put into the block cache. However, this insertion into the block cache can be bypassed if desired.

A bad choice in choosing the row key can lead to sub-optimal population of the block cache. For example, if you choose your row key to be a hash of some attribute, the block cache will be populated with random records which will have a very low likelihood of resulting in a cache hit. An alternative design in such a case would be to salt the first part of the key with something meaningful that allows records fetched together to be close in row key sort order.

CDH Config Test



Cluster Installation

Inspect hosts for correctness

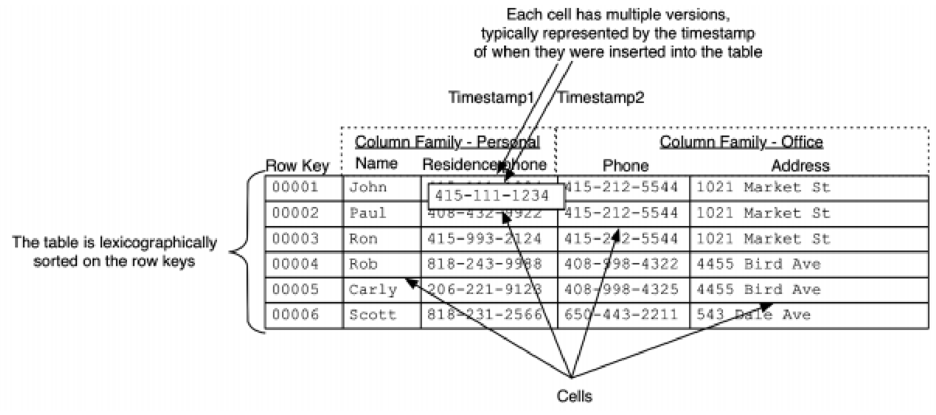
Validations

- ✓ Inspector ran on all 5 hosts.
- ✓ The following failures were observed in checking hostnames...
- ✓ No errors were found while looking for conflicting init scripts.
- ✓ No errors were found while checking /etc/hosts.
- ✓ All hosts resolved localhost to 127.0.0.1.
- ✓ All hosts checked resolved each other's hostnames correctly and in a timely manner.
- ✓ Host clocks are approximately in sync (within ten minutes).
- ✓ Host time zones are consistent across the cluster.
- ✓ No users or groups are missing.
- ✓ No conflicts detected between packages and parcels.
- ✓ No kernel versions that are known to be bad are running.
- ✓ All hosts have /proc/sys/vm/swappiness set to 0.
- ✓ No performance concerns with Transparent Huge Pages settings.
- ✓ CDH 5 Hue Python version dependency is satisfied.
- ✓ All agents running as the same system user.
- ✓ 0 hosts are running CDH 4 and 5 hosts are running CDH5.
- ✓ All checked hosts in each cluster are running the same version of components.
- ✓ All managed hosts have consistent versions of Java.
- ✓ All checked Cloudera Management Daemons versions are consistent with the server.
- ✓ All checked Cloudera Management Agents versions are consistent with the server.

Only OKs are expected.

Column Families

A table in HBase would look like Figure 1.


Each cell has multiple versions, typically represented by the timestamp of when they were inserted into the table
Timestamp1
Timestamp2
The table is lexicographically sorted on the row keys
415-111-1234
Cells

	Column Family - Personal		Column Family - Office	
Row Key	Name	Residencephone	Phone	Address
00001	John	415-111-1234	415-212-5544	1021 Market St
00002	Paul	408-432-9922	415-212-5544	1021 Market St
00003	Ron	415-993-2124	415-212-5544	1021 Market St
00004	Rob	818-243-9988	408-998-4322	4455 Bird Ave
00005	Carly	206-221-9123	408-998-4325	4455 Bird Ave
00006	Scott	818-231-2566	650-443-2211	543 Dale Ave

The Official recommendation for the number of column families per table was 3 or less.

Column families are grouped together on disk, so grouping data with similar access patterns reduces overall disk access and increases performance.

Column Qualifiers

The column family and column qualifier names are repeated for each row. Therefore, keep the names as short as possible to reduce the amount of data that HBase stores and reads.

Compaction

A few configuration recommendations include disabling auto-compaction (by default it happens every 24 hours from the time you start HBase) and schedule it to run every day at an off-peak time.

Compression

You should also configure compression (such as LZO) and explicitly put the correctly configured HBase conf directory in your CLASSPATH.

While creating tables in hbase shell, specify the per-column family compression flag:

```
create 'mytable', {NAME=>'colfam:', COMPRESSION=>'lzo'}
```

http://wiki.apache.org/hadoop/UsingLzoCompression [http://wiki.apache.org/hadoop/UsingLzoCompression]

Disks

We recommend JBOD or RAID0 for the DataNode disks, because you don't need the redundancy of RAID, as HDFS ensures its data redundancy using replication between nodes. So, there is no data loss when a single disk fails.

Distribution

Row key determines how records for a given table are scattered throughout various regions of the HBase cluster. In HBase, all the row keys are sorted and each region stores a range of these sorted row keys. Each region is pinned to a region server (i.e. a node in the cluster).

A well-known anti-pattern is to use timestamp for row keys because it would make most of the put and get requests to be focussed on a single region and hence a single region server which somewhat defeats the purpose of having a distributed system. It's best to choose row keys so the load on the cluster is fairly distributed.

Garbage Collector

As HBase runs within JVM, the JVM Garbage Collection(GC) settings are very important for HBase to run smoothly, with high performance. In addition to the general guideline for configuring HBase heap settings, it's also important to have the HBase processes output their GC logs and then tune the JVM settings based on the GC logs' output.

Key Prefix

Use a key prefix that distributes well based on your use case. If you prefix your key by timestamp or any similar value that, when sorted, is stored or queried in a batch then you will likely overload each region server in turn instead of evenly distributing the load.

Local Disk I/O

If you run MapReduce, as MapReduce stores its temporary files on TaskTracker's local file system, you might also like to set up MapReduce to spread its disk I/O.

Mixed workloads on an HBase cluster

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.

Rack Awareness

Hadoop has the concept of "Rack Awareness ". Administrators are able to define the rack of each DataNode in the cluster. Making Hadoop rack-aware is extremely important.

Region Servers

There are two basic rules of thumb that can be used to select the right number of regions for a given table. These rules demonstrate a tradeoff between performance of the put requests versus the time it takes to run a compaction on the region.

Put Performance All regions in a region server receiving put requests will have to share the region server's memstore. Therefore, the more regions that exist in a region server, the less memstore space available per region. This results in smaller flushes which in turn leads to more minor compactions.

Compaction time A larger region takes longer to compact. The empirical upper limit on the size of a region is around 20 GB.

Reserved blocks

Another method for optimization is to reduce the percentage of reserved blocks of the ext3 or ext4 filesystems. By default, some of the filesystem blocks are reserved for use by privileged processes. This is to avoid situations wherein user processes fill up the disk spaces that are required by the system daemons, in order to keep working. This is very important for the disks hosting the operating system but less useful for the disks only used by Hadoop.

Row Key Size

In general, a shorter row key is better than a longer row key due to lower storage overhead and faster performance. However, longer row keys often lead to better get/scan properties. Therefore, there is always a tradeoff involved in choosing the right row key length.

Swapping

Tune the Linux parameter to avoid swapping.

Time to Live

Limit TTL is a good practive to reduce space used for versioning.
