FileSystem Space Quotas for Apache HBase

Author: Josh Elser <elserj@apache.org>

Introduction

In a multi-tenant HBase installation, ensuring that each tenant can use no more than their allotted portion of the system is key in meeting performance guarantees for each tenant. Even one misbehaving user, malicious or ignorant, can have a massive effect on all other tenants, decreasing HBase stability, increasing client latencies, and potentially losing company revenue.

Presently, HBase provides two (2) types of quotas for administrators to enable to help mitigate the aforementioned class of problems: request quotas and number-of-tables quotas. The former allows administrators to configure the maximum number and/or size of read and/or write RPC requests sent to a RegionServer. This RPC quota support allows the quotas to be configured (in order of decreasing priority) on namespaces, tables, or users. The latter quota allows administrators to limit the number of tables and regions which may be created in a namespace.

Configurable call queues for RPCs is also technically a form of implicit quota management. This functionality allows the administrator to configure the RPC Server inside of RegionServers to direct certain types of RPCs to certain call queues and, ultimately, handler threads. This ensures that a surge of a certain type of RPC (that is still within quota) does not saturate all available processing resources of that RegionServer. Write heavy requests could prevent low-latency reads from completing as fast as expected.

This document proposes an additional quota type with the goal of additional cluster utilization control for administrators: a FileSystem space quota. This quota aims to regulate the usage of filesystem space used by a namespace or table. Because nearly all operations in HBase rely on a non-full FileSystem (even reads to a degree), one tenant writing a large amount of data into HBase can deny use of the service to all other tenants.

Goals

In general, the goals are to track how HBase namespaces and tables use filesystem space and impose limitations on that use via centralized policies which define the usage limit and the action to take when the usage limit is exceeded. The general problem itself is very difficult; however, by relaxing some constraints, we can achieve an improvement over what currently exists without sacrificing performance.

Table Size

For a production system, the overwhelming majority of filesystem space used is from HFiles. By tracking the size of each Region, we can combine the sizes across all RegionServers to compute table sizes. Table sizes can be easily aggregated into namespace sizes. Each RegionServer should collect the size of each Region it hosts periodically and report the size to the Master to compute the table/namespace view of space usage on the filesystem.

Transient space usage by flushes and compactions, files in the archive directory, and the size of data in-memory is left as follow-on work. For a first implementation, only HFiles which reside in the data directory would be considered for inclusion in the size of the table for simplicity.

Namespace Size

The size of a namespace is the sum of the size of all tables contained within that namespace. It can be computed implicitly, working backwards from the size of a region, knowing which table a region belongs to, and knowing the namespace that table belongs to.

User size

One divergence from RPC quotas is that the notion of a size quota per user is difficult to calculate. Because users are only granted permissions on tables (and are not explicitly "owners" of tables), defining a user quota would require a mapping of "quota owner" to table/namespace. By providing the ability to define quotas on tables and namespaces, administrators have the control necessary to "implement" user quotas by physically grouping a user's tables together in one namespace or by setting table quotas on each table that a user interacts with.

Snapshot size

Tracking the size of Snapshots is very difficult as the size of a Snapshot might change based on the source table compacting. Quota-ownership of the Snapshot is also difficult to reason about for a number of reasons. The naive solution to tracking snapshot size is to use the physical location of the referent HFile, but this can lead to unexpected quota usage scenarios and does not effectively work when files are in the archive directory. Effectively tracking Snapshot size and building user-based quotas is left as follow-on work because this work would likely deserve its own design document. HDFS-3370 which talks about hard-links in HDFS brings up many similar design challenges. Most likely, such a design document should inspect other filesystem implementations to glean inspiration.

Quota Configuration

Quotas will be defined as size in bytes for some namespace or table. This is a logical extension to the existing quota API for configuring RPC quotas with the Java API and HBase shell API. Only an HBase superuser should be capable of setting filesystem quotas as normal users could circumvent the controls put in place to limit their disk usage.

HBase Shell:

```
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => DELETES_WITH_COMPACTIONS,
NAMESPACE => ‘search_org’, LIMIT => ‘10TB’
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => COMPACTIONS_ONLY, TABLE =>
‘inverted_index’, LIMIT => ‘500GB’
```

Java API:

```
QuotaSettingsFactory.limitNamespaceSize(“search_org”,
      SizeViolationPolicy.DELETES_ONLY, 1024 * 1024 * 1024 * 1024 * 10);
QuotaSettingsFactory.limitTableSize(“inverted_index”,
      SizeViolationPolicy.COMPACTIONS_ONLY, 1024 * 1024 * 1024 * 500);
```

Quota Calculation

In a distributed system such as HBase, it is important that we perform computation of disk utilization in a parallelized manner to avoid lagging quota enforcement.

RegionServers

RegionServers should compute the utilization for the Regions which they are hosting and report it to the Master. This information can be computed at a configurable period, potentially leveraging the existing `RegionServerReport` mechanism to the Master. There is acknowledgement that this approach is “loose”, in that the system may not react quickly enough to prevent disk usage from exceeding the configured limit. This is desirable as it avoids impacting write performance of the system and the understanding that the system will enact the quota violation policy “soon”. Action on quota violation is primarily driven by the frequency in which these reports are sent to the Master.

Master

The Master is responsible for aggregating the disk utilization reports by Region and determining which quotas are within boundaries and which are in violation. A chore would be invoked at some period relative to the frequency at which RegionServers are reporting disk utilization. It is acceptable that the view of disk usage is delayed as RegionServers may be delayed in computing or reporting this information. We can rely on the fact that one Region should only be hosted on zero or one RegionServers. In the case that a Region is not assigned anywhere, the last received report can continue to be used. The Master must take care in handling missing reports so that removing a quota prematurely is avoided. When a quota violation is noticed, it is then responsible for ensuring that all Regions encompassed by that quota are informed that the Region should have the violation policy enforced.

Since the Master is relying on information to flow in from RegionServers, the Master also needs to handle the situation where insufficient disk usage reports are received, delaying a determination on quota acceptance/violation for a table/namespace. The Master should have some configurable threshold on reports received for a table before making a determination (e.g. reports for 90% of regions must be received before the Master will alter the quota violation state).

Quota Enforcement

When the Master observes that a violation has occurred, ZooKeeper can be leveraged to implement a low-latency (order seconds) notification to RegionServers which would match the transient-data-only use of ZooKeeper that HBase follows. More advanced notification schemes could be implemented with the goal of reducing the time between quota violation and quota enforcement or guaranteeing RegionServer acknowledgement of quota violations as follow-on work. The Procedure v2 "notification bus" feature is a logical advancement. The upper bound on noticing and distributing quota violation should be as follows: `zookeeper.session.timeout + (RegionServer disk utilization report period) + (Master quota computation period)`

When a quota limit is exceeded, the Master would instruct RegionServers to enable an enforcement policy for the namespace/table which violated the quota.

Quota Violation Policies

A quota violation policy is the action the system takes when the disk usage quota (applied to a table or namespace) is violated. We have a number of violations policies which can be used based on the requirements of the system. A quota on some table/namespace has exactly one violation policy; however, violation policy implementations are often similar to one another, a composition of some less-strict policy. Proposed policies from most strict to least strict are:

Disable table

This is the "brute-force" policy, disabling the table(s) which have violated the policy. This removes near all chance that there is any impact of table(s) over quota. For most users, this is likely not a good choice as reads should still be served. One hypothetical situation is when there are multiple active clusters hosting the same data and, because of quota violation, we know that one copy of the data does not have all of the data it should have. Thus, we disable the table until it can be verified to be consistent by the administrator.

Reject all writes/bulk-imports and compactions

This policy rejects all writes and bulk imports to the Region which the quota applies. Compactions for this Region are also disabled to prevent the system from using more (temporary) space during the compaction. The only resolution in this case is administrator intervention to increase the quota policy being violated.

Reject all writes/bulk-imports

Same as above except that compactions are still allowed. This allows users to set/alter a TTL on the table(s) and then perform a compaction to reduce the total space used. This expectation is that this violation policy comes with the understanding that space will slightly rise before it is ultimately reduced.

Allow deletes but reject writes/bulk-imports

Same as above except that submitting delete mutations is still allowed. This allows users to run processes to delete data in the system. Like the previous policy, the expectation is that this violation policy comes with the understanding that space will rise before the deletions are propagated to disk and a compaction actually removes data previously stored on disk. TTL configuration and compactions can also be used to remove data.

When a quota violation policy is enabled, the table owner should not be allowed to remove this policy. The expectation is that the Master will automatically remove the policy (however, the HBase superuser should still have permission). Automatic removal of the quota violation policy after the violation is resolved can be accomplished via the same mechanisms that it was originally enforced; however, the system should not immediately disable the violation policy when the violation is resolved. This would lead to situations where a table spends an excessive amount of time transitioning between policy enforcement and no policy enforcement. A configurable threshold (e.g. usage is <95% of the limit) would prevent the scenario where a table constantly flips between acceptance and violation.

Impact of Quota Violation Policy

While the quota violation policy defines the action the system performs when active, there are a number of systems which are affected.

“Live” Write Access

As one would expect, every violation policy outlined would disable the ability to write new data into the system. This means that any Mutation implementation other than Deletes would be rejected by the system. Depending on the violation policy, Deletes still may be accepted.

“Bulk” Write Access

Bulk loading HFiles can be an extremely effective way to increase the overall throughput of ingest into HBase. Quota management is very relevant because large HFiles have the potential to quickly violate a quota. Clients group HFiles by Region boundaries and send the file for each column family to the RegionServer presently hosting that Region. The RegionServer will ultimately inspect each file, ensuring that it should be loaded into this Region, and then, sequentially, load each file into the correct column family. As a part of the precondition-check of the file's boundaries before loading it, the quota state should be inspected to determine if loading the next file will violate the quota. If the RegionServer determines that it will violate the quota, it should not load the file and inform the client that the file was not loaded because it would violate the quota.

Read Access

In most cases, the ability to read the data stored in HBase may be affected. By introducing quotas, a goal is to ensure that HDFS remains healthy and brings(?) a higher level of availability to HBase users. By guaranteeing that there is always free space in HDFS, we can assume a higher level of health of the physical machines and the DataNodes. This leaves the HDFS reserved space percentage as a fail-safe mechanism.

Metrics and Insight

Quotas should ideally be listed on the HBase Master UI. The list of defined quotas should be present as well as those quotas whose violation policy is being enforced. The list of tables/namespaces with enforced violation policies should also be presented via the JMX metrics exposed by the Master.

Overlapping Quota Policies

With the ability to define a quota policy on namespaces and tables, we have to define how the policies are applied. A table quota should take precedence over a namespace quota.

For example, consider a collection of tables in a namespace: `n1.t1`, `n1.t2`, and `n1.t3`. The namespace quota is 100GB. As long as the sum of the usage of each table is less than 100GB, all tables can accept new writes.

Table/Namespace	**Quota**	**Utilization**
`n1`	100GB	80GB
`n1.t1`	10GB	5GB
`n1.t2`	*unset*	50GB
`n1.t3`	*unset*	25GB

In the above scenario, writes to all tables are allowed, because `n1.t1` is less than 10GB in size and the sum of all tables is less than 100GB.

Table/Namespace	**Quota**	**Utilization**
`n1`	100GB	60GB

`n1.t1`	10GB	15GB
`n1.t2`	*unset*	30GB
`n1.t3`	*unset*	15GB

In this case, writes to `n1.t1` are denied because the table quota is violated, but writes to `n1.t2` and `n1.t3` are still allowed because they are within the namespace quota. The violation policy for the table quota on `n1.t1` would be enacted.

Table/Namespace	**Quota**	**Utilization**
`n1`	100GB	108GB
`n1.t1`	10GB	8GB
`n1.t2`	*unset*	50GB
`n1.t3`	*unset*	50GB

In this case, writes to all tables would be disallowed because the sum of the utilization by all tables exceeds the namespace quota limits. The namespace quota violation policy would be applied to all tables in the namespace.

Table/Namespace	**Quota**	**Utilization**
`n1`	100GB	115GB
`n1.t1`	10GB	15GB
`n1.t2`	*unset*	50GB
`n1.t3`	*unset*	50GB

In this case, `n1.t1` violates the quotas set on both the namespace and the table level. The table quota violation policy would be the policy to become active in this case. For `n1.t2` and `n1.t3`, the namespace quota violation policy would take effect.

On HDFS-provided Quotas

To play devil's advocate, quota support could be implemented/leveraged in Apache Hadoop HDFS instead of the implementation described here. Ideally, this would be beneficial as it would reduce the amount of code in HBase and be "easily" handled via general HDFS unavailability. However, there are a number of drawbacks to this approach.

1. Use of Hadoop’s HDFS would be required for space quotas. HBase has been shown to run on other distributed FileSystems and relying on HDFS’s implementation would leave other FileSystems to implement their own solution to the problem. With the approach in this document, we can solve the problem once for all FileSystems.
2. HDFS’s current quota support is tied to specific directories. Due to HBase’s current layout of data spread across multiple directories (notably, the archive directory), this quota support would not be sufficient for our use-cases.
3. For write-ahead-logs, which are across all tables/namespaces, there is no means to appropriately limit disk usage at the granularity outlined in this document. While a different write-ahead-log filesystem layout is possible, this would be a very impactful change that would (likely) be difficult to implement correctly (resulting in data-loss).

Example Use

This outlines a general approach to how this feature would be used in a real environment. Consider a business organization which is comprised of three groups: Production Website, Beta-testing Website, and “R&D”.

The Production Website group manages the customer-facing website which is created using data from HBase. The beta-testing website which is a staging area for changes ultimately heading towards the production website. This group vets these potential changes to the website before they are rolled out to all users. The R&D group is the team that experiments with new features and changes. Some of their features will make it to beta test, and maybe, eventually, the production website. Because the company is small, they presently have one shared HDFS and HBase cluster which all groups use. Customers using the website drive revenue for the company, so the customers should never see errors (as it would result in loss of revenue, present and future). As such, the HBase administrator chooses to enable filesystem space quotas, in addition to RPC quotas, for each group.

Because each organization is ultimately building on top of the same collection of HBase tables, each organization has their own HBase namespace.

```
hbase> create_namespace ‘prod_website’
hbase> create_namespace ‘beta_website’
hbase> create_namespace ‘research’
```

The HBase cluster has 10TB of replicated space. The production website group gets the majority of space, the beta website group less, and the research group the least.

```
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => DELETES_WITH_COMPACTIONS,
NAMESPACE => ‘prod_website’, LIMIT => ‘7500GB’
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => COMPACTIONS_ONLY, NAMESPACE
=> ‘beta_website’, LIMIT => ‘1TB’
```

```
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => REJECT_ALL, NAMESPACE =>
'research', LIMIT => '500GB'
```

Reserving 5% of space as “reserved space”, the production website group gets roughly 80% of the space. When they violate their quota, they have the most freedom to recover (the ability to perform compactions or issue deletes to free up space). The beta website is granted roughly 10% of cluster space so that they can do testing over an adequate data set. However, to prevent too much data from being loaded, when they exceed their quota, they may only use TTL and compactions to remove data. Finally, the research group gets roughly 5% of cluster space to run their experimentations. If they exceed their quota, both compactions and deletes are prevented to ensure that these experiments do not interfere with the beta testing or production website.

Consider a problem where, in an attempt to load more data for scale testing, the beta website group miscalculates the raw data they need to process and load, causing them exceed their quota. As they have multiple teams concurrently loading/testing the system, they decide they want to delete the other ongoing test data so that they can complete the scale testing that they already started. So, they must ask the administrator to give them the ability to delete just part of the data.

```
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => DELETES_WITH_COMPACTIONS,
NAMESPACE => 'beta_website', LIMIT => '1TB'
```

After the administrator makes this change, the beta website group can delete the other data sets in the namespace so that they can finish their scale testing experiment. After they have deleted the data sets, the administrator can change the policy back.

```
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => COMPACTIONS_ONLY, NAMESPACE
=> 'beta_website', LIMIT => '1TB'
```

Consider another scenario where the R&D team wants to run a larger test than they have the ability to run before sending a feature to the beta website group. They talk to the administrator who agrees to temporarily change the space quotas from 80%/10%/5% to 75%/5%/15% due to sufficient free space on the cluster but only on the off-peak hours (Friday night through early Monday morning):

```
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => DELETES_WITH_COMPACTIONS,
NAMESPACE => 'prod_website', LIMIT => '7125GB'
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => COMPACTIONS_ONLY, NAMESPACE
=> 'beta_website', LIMIT => '500GB'
hbase> set_quota TYPE => SIZE, VIOLATION_POLICY => REJECT_ALL, NAMESPACE =>
'research', LIMIT => '1500GB'
```

The administrator makes the quota limit changes for space and creates a task that will run early Monday before website traffic resumes to replace the previous rules. The administrator knows that even if the R&D group does use the full extra TB of space that he granted them temporarily, the cluster itself would have enough free space. If, for some reason, the R&D group did not clean up their data before the quota limits were reset, the R&D group will have to contact the administrator to set forth a plan to remove the extra data they had loaded which they cannot clean up on their own (as it may impact beta/production).

Closing

Thanks to Devaraj Das, Enis Söztutar, and Clay Baenziger for their suggestions, feedback and guidance.