HBase Time-based Data Tiering using Persistent BucketCache (Preview)

Date published: 2024-10-01
Date modified: 2024-10-09

Legal Notice

This document has been released as part of a technical preview for features described herein. Technical preview components are provided as a convenience to our customers for their evaluation and trial usage. These components are provided ‘as is’ without warranty or support. Further, Cloudera assumes no liability for the usage of technical preview components, which should be used by customers at their own risk.

Contents

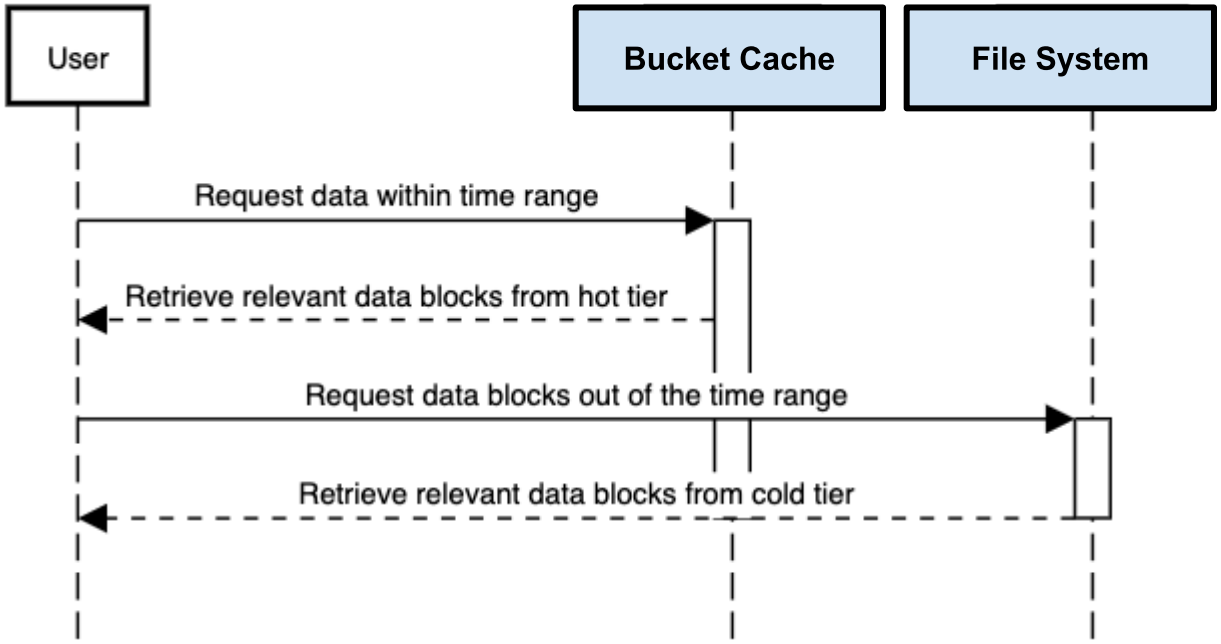
HBase Time-based Data Tiering using Persistent BucketCache

You can configure time-based priority caching for HBase, define a specific time range for data to remain in the cache, and choose to evict older data. Based on the configuration, more recent data is given a higher priority.

Introduction

Using this functionality, you can optimize storage efficiency and access performance by segregating data based on its recency. By keeping recent data in the cache and evicting older data, the system aims to provide more flexible control over the cache allocation and eviction logic through configuration, allowing to define time priorities for cached data. The need for a more extensive cache allocation mechanism becomes even more critical on HBase deployments where cache access reflects on significant performance gains, such as when using cloud storage as the underlying file system.

The following figure depicts how data is fetched between the bucket cache (hot tier or SSD storage) and file system (cold tier or cloud storage).



Components

The following are the key components involved in the time-based data tiering functionality.

- Hot tier (SSD storage): This tier consists of high-speed local storage, typically SSD storage, where recent data is cached for fast access. This is based on the BucketCache implementation in HBase.
- Cold tier (Cloud storage): This tier comprises cost-effective cloud storage for storing older data.
- Compaction: Organizes data into files based on the provided time range and compaction strategy. Date-tiered compaction is a date-aware store file compaction strategy that categorizes data into files based on time, making it a prerequisite for time-based data tiering.
- Eviction policy: Determines which data blocks should be evicted from the cache based on the provided time range.
- Prefetch logic: Controls the prefetching of data files, ensuring that only relevant files within the user-specified time range are processed.

Workflow

The following workflow shows how data is processed with the time-based data tiering functionality.

- Data ingestion:
    - Data is automatically flushed from the memstore to the cache based on a predefined configuration (cacheOnWrite). Hence the recent data always stays in the cache.
- Compaction:
    - When the date-tiered compaction is enabled, the recent data within the specified time range is compacted into smaller files, while older data is grouped into larger files. Additionally, if cacheCompactedDataOnWrite is enabled, the data is sent for caching, and a time-based data tiering is performed again to cache only "hot" data while declining to cache the "cold" data.
- Eviction:
    - The eviction policy determines which data blocks in the cache do not belong to the specified time range.
    - When the cache reaches its capacity and evictions must be triggered, the blocks containing data outside the user-provided time range are prioritized for eviction but not necessarily evicted every time. If only a small portion of the data within the specified time range is sufficient to make space for the incoming blocks, the eviction process stops, and the remaining data outside the time range stays in the cache.
- Prefetching:
    - When accessing data within the specified time range, the prefetch logic ensures that only relevant files are fetched from the cloud storage for caching.
    - Files outside the specified time range are not prefetched, reducing unnecessary data transfer and cache usage.

Configuration details

Know the configuration parameters to enable the time-based data tiering functionality and set the data age priorities.

1. Log in to the HBase shell.
2. Use the `ALTER` command to modify the following configuration parameters at the table or column family level.

Configuration parameter	Description	Supported values
hbase.hstore.datatiering.type	Set this parameter to enable the time-based data tiering at the table or column family	TIME_RANGE Default value: NONE

Configuration parameter	Description	Supported values
	levels.	
hbase.hstore.datatiering.hot.age.millis	Specifies the time duration for which the data is considered hot and must have the highest priority to be kept in the cache. Set this parameter at the table level to implement the time-based data-tiering for all the table's column families, or at the column-family level to set the functionality for the column family.	Default value: 7 Days or 604800000 milliseconds
hbase.hstore.engine.class	Set this parameter to enable the date-tiered compaction at the table or column family level.	org.apache.hadoop.hbase.regionserver.DateTieredStoreEngine

Enabling the time-based data tiering

Know how to enable the time-based data tiering in HBase.

About this task

To use the data tiering functionality at the table or column-family level, you must set the global data tiering configuration, **hbase.regionserver.datatiering.enable** to true. This configuration activates the data tiering across your HBase instances.

This task demonstrates how to enable the global data tiering across multiple HBase instances.

Note:
This functionality is only beneficial when the write access pattern primarily consists of sequential writes based on the data arrival time to the backend, and the read access pattern mainly involves accessing the recent data.

Prerequisites

- Ensure that the bucket cache is enabled for the cluster. This is mandatory for the data tiering to function. For more information, see HBase persistent BucketCache.
- Before enabling the time-based data tiering at the table or column-family level, you must enable the data-tiered compaction using the **hbase.hstore.engine.class** configuration. For more information on Apache data-tiered compaction, see Date Tiered Compaction.

Steps

1. Log in to the Cloudera Manager as an Administrator.
2. Select the **HBase** service.
3. Go to **Configuration > Advanced > HBase Service Advanced Configuration Snippet (Safety Valve) for hbase-site.xml**.
4. Set the value of **hbase.regionserver.datatiering.enable** as `true` to enable the global data tiering.
5. Click **Save Changes**.
6. Restart the HBase service.

Post-requisites

- After enabling the global data tiering configuration, you can start using the data tiering feature at the table or column-family level. This feature is effective only when used with the date-tiered compaction strategy. Cloudera recommends enabling the date-tiered compaction with the additional configurations to optimize it for the tables or column families and data-tiering.
  Refer to the following example.

```
$ hbase > alter '<table_name>', CONFIGURATION =>
{'hbase.hstore.datatiering.type' => 'TIME_RANGE',
'hbase.hstore.datatiering.hot.age.millis' => '<hot_age>'}
```

- Run the following command in the HBase shell to check whether the data tiering functionality is enabled.

```
$ hbase shell
$ hbase > describe <table_name>
```

Expected output:

```
hbase:009:0> describe 'usertable'
Table usertable is ENABLED
usertable, {TABLE_ATTRIBUTES => {METADATA => {
  'hbase.hstore.datatiering.hot.age.millis' => '3600000',
  'hbase.hstore.datatiering.type' => 'TIME_RANGE',
  'hbase.hstore.engine.class' =>
'org.apache.hadoop.hbase.regionserver.DateTieredStoreEngine'
}}}
COLUMN FAMILIES DESCRIPTION
{NAME => 'cf', BLOOMFILTER => 'ROW', IN_MEMORY => 'false', VERSIONS
=> '1', KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING =>
'NONE', COMPRESSION => 'NONE', TTL => 'FOREVER', MIN_VERSIONS =>
'0', BLOCKCACHE => 'true', BLOCKSIZE => '1024', REPLICATION_SCOPE =>
'0'}

1 row(s)
Quota is disabled
Took 0.0392 seconds
```