Item 199
git_comments:

1. Initialize with non-empty onlineSegments

git_commits:

1. **summary:** Broker time segment pruner(#6189): (#6462)
   message: Broker time segment pruner(#6189): (#6462) - Fix bug in init(). - Add support for IN operator, e.g. select * from mytable where timeColumn in (2001-01-01, 2021-01-01) - Add tests for simple date format.

github_issues:

1. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
   label: code-design
2. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
3. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
4. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
   label: code-design
5. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
6. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
   label: code-design
7. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
   label: documentation
8. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
9. **title:** Support Timestamp Pruning of segments in Broker
   body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.
   label: code-design
10. **title:** Support Timestamp Pruning of segments in Broker
    body: For time based tables, segments can easily pile up. High qps would lead to a lot of unnecessary work being done server side to read and eliminate segments.

github_issues_comments:

1. **body:** For time range pruning on broker we have two solutions: 1. **Naive O(n)** Currently, for each table, brokers keep track of all its online segments. PartitionSegmentPruner maintains a mapping from segment names to partition info. Same as partition pruning, we can have a time range pruner which keeps a mapping from segment names to time range. Pruning is done by looping through online segments, getting their partition infos and filtering against query time range, this will cost O(n)(n is # of online segments). **Pros**: easy to implement **Cons**: long time to compute the segment pruning for tables with large number of segments 2. **O(m*logn)** using some specialized data structure Instead of

keeping the time range to segment mapping, we can have some specialized ordered data structure (e.g. interval search tree: a augmented balanced BST) optimized for searching intercepted ranges against query range. Pruning is done by searching all intercepted ranges from the interval search tree, and checking if they are online. This will cost O(m*logn) (m is # of qualified segments). There’s two implementations: **Implementation 1**: a concurrent augmented bst **Implementation 2**: a read only augmented bst, a new tree will be built and atomicly swapped with the old one when there’s an external view change or a segment metadata updates **Pros**: reduced time to prune for large number of segments **Cons**: * More data (1.5 - 2 * naive solution) stored on brokers because of additional pointers and auxiliary info for tree node * For Implementation 1, hard to code (a research topic), and reading performance will be harmed due to locking * For Implementation 2, when there’s an external view change or segment metadata updates, the whole tree needs to be gced. Due to performance requirements and since the pruner is read heavy (thousands qps in production, and external view change happens few times a day), the solution 2 implementation 2 is considered a better approach.
   label: code-design
2. The particular cases where this is necessary is when we have a large amount of data necessitating a large number of segments. In this particular situation, we want to make sure this scales. The naive solution does not scale particularly well on time when you get to millions of segments. The specialized data structure scales well on time, but is potentially a memory hog. The brokers like to keep _all_ of the routing tables in memory. That's going to get hungry. Can we quantify this? If there's 20 million segments across 10 tables, is it a gig of memory? 100 gigs? On the difficulty to implement front, why not just start with a [Java Sorted Set](https://docs.oracle.com/javase/8/docs/api/java/util/SortedSet.html) on start timestamp? Then filter on end timestamp on those results? Or use a pre canned library built for this specific task.
3. In fact, why not create a generic SortedSet based pruner that can be used for any numeric or sortable column?Then, this can be used for other columns such as sequence numbers. Which would be another very valuable use case for us
4. **body:** @noahprince22 If we keep the start timestamp only, we cannot effectively prune segments because we don't know the upper bound. Keeping start time may help for your use case but it's not a generic solution. (i.e. time filter on queries can be made with no end timestamp) Simple math: Let's assume that we roughly store 100bytes for each segment (we need to store segment name, start & end timestamps, and some other info). ``` 100 bytes /segment * 20 million segments = ~2GB ``` It indeed requires GBs of memory; however, having 20millions of segments for Pinot cluster is a bit extreme use cases. If you set your segment size to be a reasonable size (200-300MB per segment), you won't have 20million segments. (200MB * 20 million segments = 4PB). To support this many segments, we probably need to read the metadata from disk instead of keeping everything in memory. Also, we probably will need to do a lot of other optimizations on brokers to support the cluster at this scale. IMO, we can first start with what @jtao15 suggested and see how they perform on your use case. How do you think?
   label: code-design
5. Copied and pasted the discussion from slack channel: [slack discussions] (https://docs.google.com/document/d/1qbiwzTVNeO_syBrvDL-Y-s_xep8nUtJAi1S9KJFfq4A/edit?usp=sharing)
6. **body:** I would suggest going with the easier approach (approach 1) and establish some baseline first. I would doubt that we can run into performance issue for the naive implementation as the partition based one works pretty good so far. The naive one should also work better if we have multiple pruners chained together. We can start the advanced one if we really observe performance issue from the naive one.
   label: code-design
7. **body:** @jtao15 Can you please confirm if this feature is done and tested? Also, is there any documentation for how to enable it?
   label: documentation
8. @Jackie-Jiang Yes, the feature is done and tested. It can be enabled by specifying time pruner in routing config: ``` "routing": { "segmentPrunerTypes": ["Time"] } ``` The supported query operators are `RANGE`, `=`, `<`, `<=`, `>`, `>=`, `IN`, `AND`,`OR`. E.g. `Select * FROM table WHERE timeColumn BETWEEN 20 AND 30 OR timeColumn >= 40`. Time pruner will filter out segments which don't have records in [20, 30] or [40, *).

github_pulls:

github_pulls_comments:

github_pulls_reviews:

jira_issues:

jira_issues_comments: