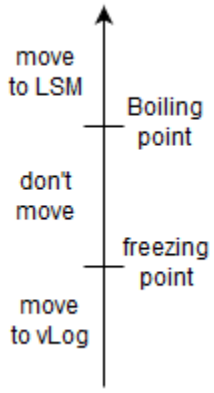
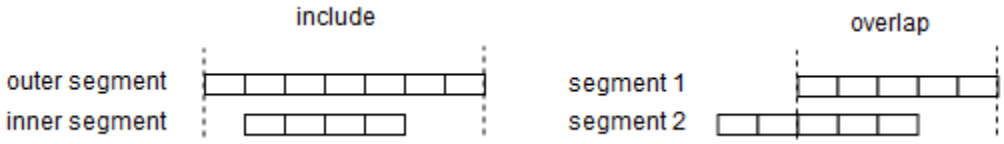
Design of Manager and Segment Table

I have implemented Segment Table in *badger**stat* and Manager in *badger*. The *segment.go* and *segment_test.go* are the segment table's implementation and test codes. The *manager.go* and *manager_test.go* are the manager's implementation and test codes. To run the test codes, type *go test* in the command line under each directory. Segment Table and Manager are designed to be thread safe.

To decide whether to move the data between LSM tree and vLog, I use two threshold values: freezing point and boiling point. If the segment heat is above boiling point, its data will be passed to the move process and will be moved from vLog to LSM. If the segment heat is below freezing point, its data will be passed to the move process and will be moved from LSM to vLog. The use of both boiling point and freezing point instead of one threshold value is to avoid thrashing: if a segment's heat is change frequently between one interval, we shouldn't move it.



The segment table uses coldest first replacement principle. During the store of segment, it keeps the coldest segment and if there's no free space, it will replace it. Notice that segment table doesn't replace any segment above boiling point. This is to keep the consistency of segment table: hot segment is still in LSM, and we will lose track of it if this segment entry is replaced.



For the coalesce of the segments, I use two methods: Inclusive Coalesce and Overlapping Coalesce. One segment includes another if the outer segment starts earlier and ends later than inner segment. Two segments overlap if they have common sub-segment. The coalesce is done by combining the segments and adding together the heats.

- Inclusive Coalesce: Two segments in the Segment Table will coalesce if one segment includes the other and the outer segment is above boiling point. The requirement that outer segment is above boiling point is necessary. Consider the following range queries: [1,2], [1,2], [1,10], [1,2], [1,2], [1,2] … [1,2] should not be coalesced into [1,10] because [1,10] is just an accidental range query and data within [2,10] are brought into LSM tree wrongly.
- Overlapping Coalesce: Two segments in the Segment Table will coalesce if two segments overlap and both are above boiling point. The requirement that both segments are above boiling point is necessary for the similar reason above. However, the Overlapping Coalesce has its problems. For example, it may combine two segments a bit above the boiling point and well above the boiling point together, so that the former one won't be easily cooled down.

Maybe we could add a similarity feature to decide whether to coalesce. For now, it is hard to judge because data may have a bias distribution. Consider [1,8] and [1,9]. These two segments are similar in number, but huge amounts of data may be stored within [8,9], causing the coalesce problematic.

Segment Table also has a cooling system. The cooling is done by multiplying heats by a float between 0 and 1. The multiplication will deduct more for large heat, making data frequently range queried in the past cool down more easily.

Manager is embedded in the DataBase and builds the connections among Transaction, Move Process and Stat (Segment Table for now). It will do the following works:

1. When user uses a transaction to load data, Manager will decide where to store value. For value size larger than 62KB, Manager will store it in vLog directly. Otherwise, it will ask Stat where to store the data based on whether the heat is above boiling point. Manager will always keep a cache about the start and end of the segment asked before. Therefore, for a sequential load, Manager could give an immediate answer without consulting Stat.
2. When user uses an iterator to do range query, Manager will add segment and its heat to the Stat through Manager.
3. When user uses a transaction to do random search, it will ask Manager to cool the segments. Manager won't do the cooling immediately. It keeps a count of how many times it has been asked to do a cooling and will cool the segments only after a threshold.
4. When a segment is above boiling point but still in the vLog, it will send a request to Manager to move the value to LSM tree. Manager will then send the request to the Move Process in the background.
5. When a segment is below freezing point but still in the LSM tree, it will send a request to Manager to move the value to vLog. Manager will then send the request to the Move Process in the background.
6. When the Move Process has done a move, it will then the work to Manager. After Manager has received the work, it will update the stats in the Segment Table.

For now, there is a new API to set the data through manager:

```
err := txn.HybridSet([]byte("key"), []byte("value"))
if err != nil {
  return err
}
```

This API should replace the traditional txn.Set in the future.

One problem of current design is that there are too many things to configure. We need to configure the length of Segment Table, boiling point, freezing point, size of cache in Manager, threshold to trigger a cooling, multiplicator in cooling…