UBS Weight Volume Average Price calculator design

The design is made based on the assumptions as below,
State enum is not involved into the business logic as it is not specified in the assignment and it is not able to get it clarified by hiring manager in weekend, so it always takes the default value FIRM.
The whole calculation logic is based on that:

1. WVAP is at instrument level across all available markets
2. there are 50 available markets specified in the assignment

In terms of OO, CalculatorImpl has a priceManager. That is responsible for keeping all the recent two way prices cross all markets and they are grouped by instruments.
The current implementation takes a nested HashMap to be priceManager

instrument0————>[market0——>recentTwoWayPrice0]
|————>[market1——>recentTwoWayPrice1]
|————>[marketN——>recentTwoWayPriceN]

instrument1————>[market0——>recentTwoWayPrice0]
|————>[market1——>recentTwoWayPrice1]
|————>[marketN——>recentTwoWayPriceN]

instrument2————>[market0——>recentTwoWayPrice0]
|————>[market1——>recentTwoWayPrice1]
|————>[marketN——>recentTwoWayPriceN]

Whenever the new twoWayPrice comes in, the calculator would firstly locate at which market slot the twoWayPrice needs to be replaced.

For example recentTwoWayPriceN gets updated for marketN, so this particular slot have to be updated initially as below,
instrument0————>[market0——>recentTwoWayPrice0]
|————>[market1——>recentTwoWayPrice1]
|————>[marketN——>**recentTwoWayPriceN**]
Then the calculation logic would get triggered to figure out the latest WVAP according to WVAP formula and returns it to caller.

Further plan

- DI framework is not included in the current design due to the simplicity
  - priceManager would get updated with cache capacity and then get injected into Calculator with the help of some dependency injection framework such as Spring
- Multithread mode to achieve more effective calculation process
  The calculations cross markets should be partitioned and each thread would take each partition and passes the result back aggregator thread to get final result for WVAP.

  Option1
  Runnable + threadExecutor for light weight WVAP calculations

  Option2
  Akka -> each assignee actor takes one calculation task and pass the result back to assigner actor, then aggregate the partial results to the final result.

  Option3
  Same approach could also be implemented by Spark

  Both option2 and option3 are for heavy weight WVAP calculations with intensive IO operations