# Space and Time Tradeoffs

I was recently asked to create and compare two solutions to a telephone call analysis problem. As often happens between engineers, a preference develops for a particular solution. We often select the “fastest” solution, based on “Big Oh” analysis. However, as discussed in a previous article on the vagaries of such an analysis, we should consider the actual performance in the expected area of application (please see <https://www.linkedin.com/pulse/lies-damn-algorithm-analysis-donald-trummell/>.) However, algorithms often trade off space performance. These tradeoffs need to be considered as well. Let’s explore one interesting example of space-time tradeoff.

## Interviewer’s Problem

We have a log file of telephone calls. The log has a sequence of unordered call times, start and end, recorded to the nearest minute. The sequence initially spans a day of calls. We need to find the maximum number of operators required to answer those calls during the peak portion of a day. Assuming one operator per active call, this number corresponds to the maximum number of simultaneously active calls.

## Solution Exercise

We explored two algorithms: “Binning” and “Event monitoring”. Binning uses the time interval of interest, a day in this case, to create counters (bins) that span the interval to the desired granularity; a minute for our problem. Binning records the count of active calls during each minute of the day in the 1,440 bins. The maximum call count of this bin collection is the desired maximum active call value.

Event monitoring tracks the active calls for the entire data set and records the maximum call count during processing. Even monitoring assumes a new call starts an “event” and records all the in-progress calls. Any call that ended prior to the start time of the new call is evicted from the active call list, and the new call is added to the list. Tracking the maximum size of the adjusted active call list will yield the maximum number of calls during the day.

Naive Event monitoring has severe performance issues when using an array list to record active calls because the entire list has to be examined with each call, and the list entries must be reshuffled after each removal. This performance problem is addressed by using a Java priority queue. *Note that event monitoring requires ordered input (ascending)!*

You may view examples of all three coding approaches in the DemoDev repository (see references below):

* Binning: ***SolutionBins.java***
* Event Monitoring: ***SolutionEvents.java***
* Queue-based Event Monitoring: ***SolutionEventsQueue.java***

For the initial problem conditions, all three approaches are nominally of time complexity ***O(n),*** where n is the number of calls in the log. After examining the Binning approach, which has a dependency on a small, fixed number of bins, the interviewer altered the problem. He proposed a much longer time interval, significantly exceeding a day, and therefore greatly increased the number of bins. He wanted an algorithm not hampered by the time interval of interest. This prompted examining the Event Monitoring approach.

## Testing Data

Analytical Performance analysis is a very useful tool, and a great exercise to understand the algorithm, but should be backed by actual performance evaluation in the context of expected production usage.

Unfortunately, testing of JVM based applications often run into complexities of garbage collection, threading, and operating system load variation. This means that the predicted and actual performance results may vary for different inputs of the same scale. For example, running performance tests in the Eclipse IDE will be influenced by the memory consumed by Eclipse, as well as background threads performing Eclipse actions.

The first performance challenge was to generate representative data for testing (see ***CallGenerator.java*** in DemoDev.) The generated telephone calls where between one and twenty-five minutes in duration, spread over many days, with each day having three peak call periods. This mock test data was created to study the actual behavior of the code under expected production conditions. A sample of call data count distribution for a single day is:

Dfsdsfsdf

Sdfsdfsdf

Sdfsdfsdf

Sdfsdfsdf

This mock telephone call distribution was replicated over the required multi-day test intervals by advancing the start-end times as needed.

## Testing Conditions

Timed runs were recorded for 1, 10, 20, 40, 50, 60, 70, 80, 90, 100, 110, and 120 days with a call volume of 250,000 calls a day. Both Binning and Event Queue were timed in random order. Timing was run outside the Eclipse IDE as suggested above to minimize uncontrolled variation in results.

## Testing Results Summary

Binning and Event Monitoring had similar performance characteristics up to 100 days. The Binning algorithm outperformed Event Monitoring over that interval of interest, and was often faster than Event Monitoring even when the input data were sorted. Beyond 100 days, the Event Monitoring was faster.

Binning is a popular technique that can return much more information than just the maximum number of calls. A study of this nature would try an answer the question: “what is the probability of a missed call given N operators”, and binning would return that information. Also, rarely can log data be relied on for ordering. Usually a sort is required, so the binning approach is less complicated.