Breaking Java Badly - Programmatic Destructive Testing of Java Virtual Machines (Part Three)

By
Mark Raley

In parts one and two, we presented results from over two thousand separate runs generated from two producer-consumer tests (Test One - Linked List and Test Two - Array Dequeue) for two popular OpenJDK 1.8 memory collectors (Parallel Old and G1). These tests are further organized into batteries that share the same MCL setting. In this part, we drill down into the MCL 36M test battery which contains the longest-running execution times seen.

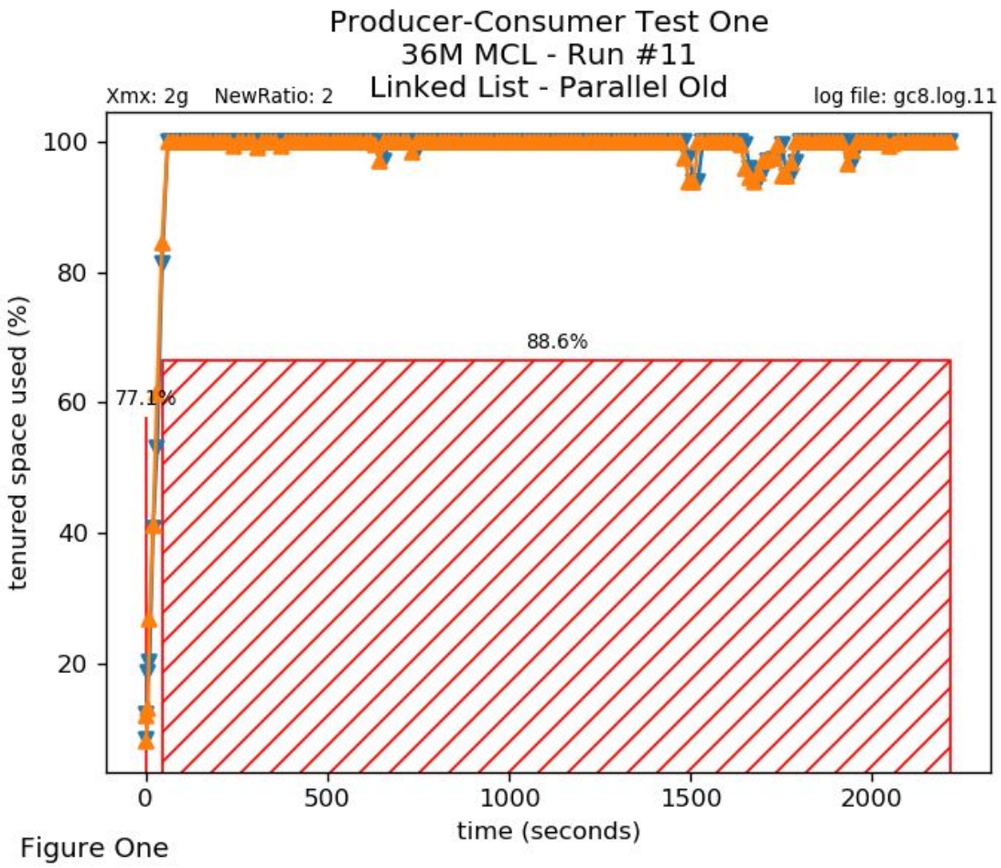

Figure One

Figure one is a scatter plot of all of the full garbage collection (FGC) entries from the saved JVM log file for run #11. The x-axis is time in seconds relative to test (and JVM) start and the y-axis is the percentage of tenured space (generation) in use relative to the maximum allowed tenured space that the JVM can ever use, as governed by the Xmx and NewRatio settings. The blue line is this percentage at the start of the FGC, and the gold line is this percentage after the FGC is complete.

In this example, the tenured space fills rapidly and remains so. There are many FGCs here in a test that takes over 2000 seconds to complete where usually about 250 seconds suffices. This plot also has an important annotation - the red hatched area denotes a region in time where the JVM memory collector is running 88.6% of the time and is the proximate cause of the delayed test completion. These FGCs may be said to be frequent and temporally adjacent. In the test setup used here (see part one for details) the JVM by default uses around six of the available eight physical cores to perform memory collection activities. In the nominal case, two cores are busy (one producer thread and one consumer thread).

Let’s contrast the abnormal run (figure one) with a healthy run (figure two).

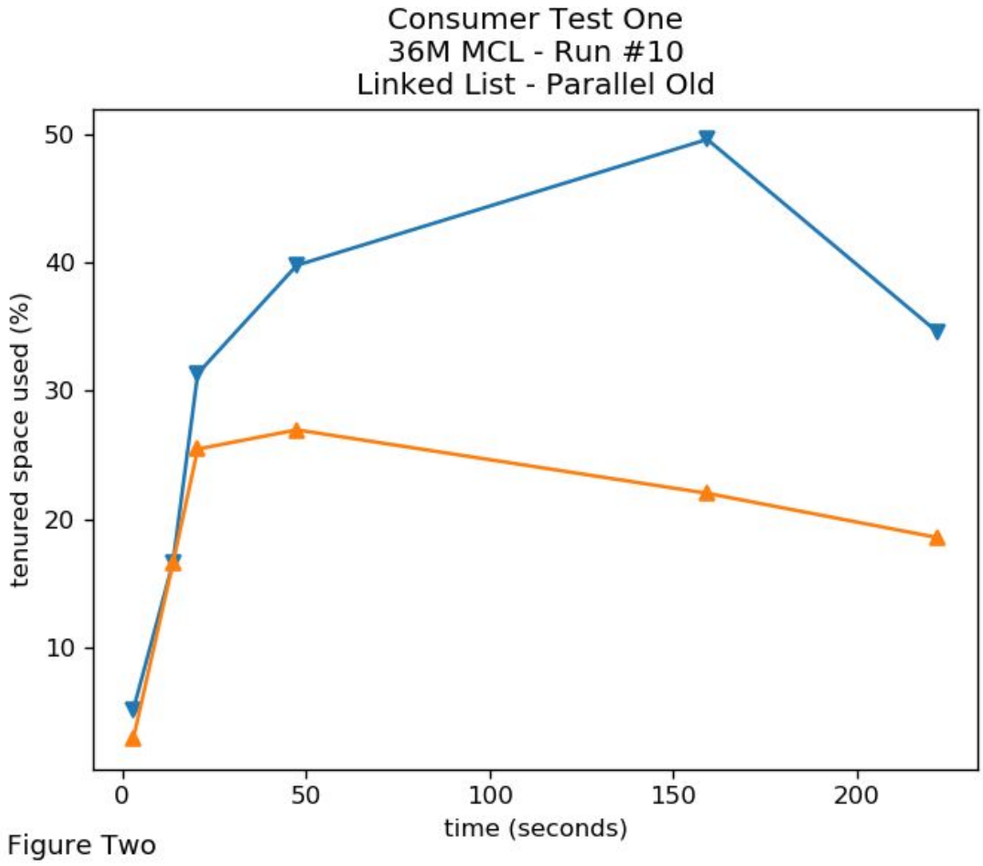

Figure Two

In figure two we can count the FGCs visually and see that the tenured space is never more than about 50% in use with a run time of about 250 seconds. This is a superior run. While in general the producer and consumer threads are well balanced by the JVM, sometimes the producer gets to run more often for a variety of reasons including underlying conditions deeper in the stack upon which the JVM depends - including thread scheduling and prioritization.

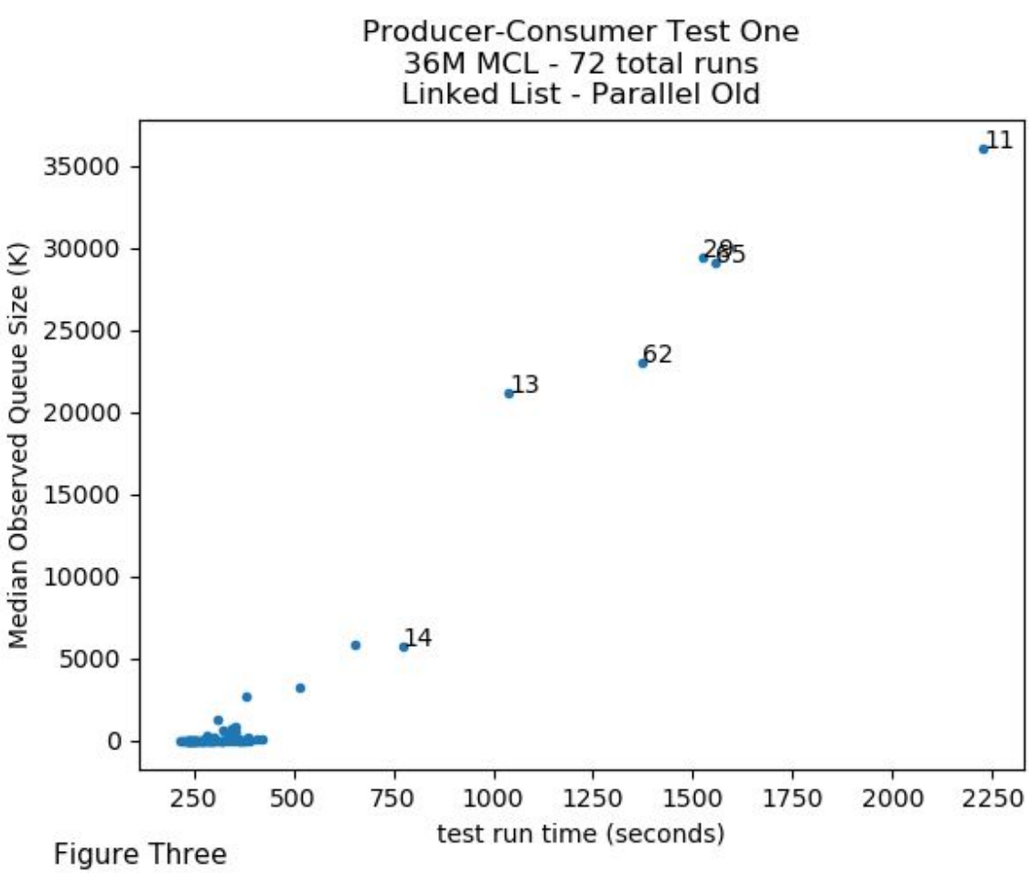

Figure Three

Figure three gives us the bigger picture of the complete 36M test battery. It plots the median queue depth of each test run against their execution time, 72 data points in all. During these tests, the consumer emits a time-stamp as well as the current queue depth every 32 million (32M) queue elements processed. The y-axis in figure three is the median value of these results on a per-test run basis. Outliers in the dataset have their run numbers displayed.

By viewing the data in this way we can see a clear correlation between observed queue depth with high execution times, adding further confirmation that excess capacity is the proximate cause of the observed performance and instability issues under test. The rightmost seven points specifically match the seven outliers seen in part one figure one column 36M, and all show long FGC walls.

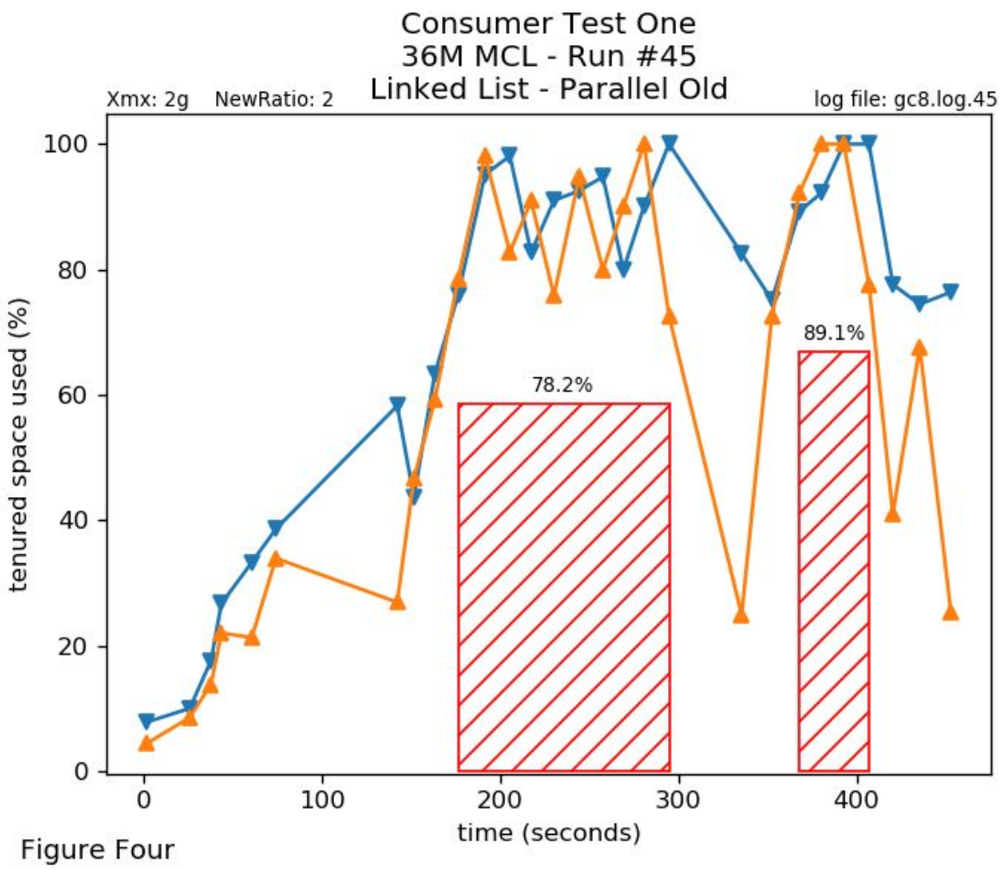

Figure Four

Figure four is an example of how even shorter runs of around 500 seconds (non-outliers) can show detectable and measurable signs of a heap under pressure.

In the next article (part four) on GitHub, we'll make available the python code that is used to generate these plots as well as a working definition for temporal adjacency.