Java Path Finder

LINGI2251
Software Quality Assurance

By

William Glazer-Cavanagh
Vincent Tessier
Natahan Rullier

Université Catholique Louvain

7 May 2020

1 Concurrent Test Cases

The parameters which affect the number of concurrent tests cases are as follow:

1. `FILECOUNT` This parameter adds an additionnal random choice of the nuber of files per iteration.
2. `RANDOM_OPERATORS` This parameter is a boolean which specifies if the operators are random or fixed.
3. `ITERATIONS` This parameter fixes the number of random choices of files and operators since each iteration must choose a file and operator.
4. `THREAD` This parameter fixes the number of random choices of files and operators since each Thread must run iterations to choose a file and operator.

The analytical formula for the entire program would be:

$$(6^{\texttt{RANDOM\_OPERATORS}} \cdot \texttt{FILECOUNT})^{\texttt{ITERATIONS}\cdot\texttt{THREADS}} \tag{1}$$

This is because there are two random choices: the choice of a file and the choice of an operator. A fixed operator will still affect the output if there are more than one file. To combine treads with varying parameters, we can multiply the formula by each number of concurrent test cases.

For example a test with 1 iterations of 2 files with a thread having random operators and the second thread having fixed operators would give $(6^1 \cdot 2)^{1\cdot 1} \cdot (6^0 \cdot 2)^{1\cdot 1} = 12^1 \cdot 2^1 = 24$ concurrent test cases.

2 First JPF Run

```
====================================================== results
error #1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty "java.lang.AssertionError  at DaisyUserThread.run(D..."

====================================================== statistics
elapsed time:       00:00:54
states:             new=134152,visited=128860,backtracked=260811,end=2
search:             maxDepth=2740,constraints=0
choice generators:  thread=134151 (signal=95,lock=16112,sharedRef=83121,threadApi=1992,reschedule=32831), data=0
heap:               new=43454,released=59000,maxLive=1008,gcCycles=233361
instructions:       20748838
max memory:         412MB
loaded code:        classes=90,methods=1941

====================================================== search finished: 07/05/20 4:00 PM
```

Figure 1: Results of first JFP run with errors

```
====================================================== results
error #1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty "java.lang.AssertionError  at DaisyUserThread.run(D..."

====================================================== statistics
elapsed time:       00:00:02
states:             new=3120,visited=2735,backtracked=5755,end=0
search:             maxDepth=100,constraints=138
choice generators:  thread=3042 (signal=88,lock=528,sharedRef=1952,threadApi=3,reschedule=471), data=0
heap:               new=27272,released=1835,maxLive=1007,gcCycles=5003
instructions:       770064
max memory:         212MB
loaded code:        classes=90,methods=1941
```

Figure 2: Results of JFP when limiting depth to 100

```
====================================================== results
error #1: gov.nasa.jpf.vm.NoUncaughtExceptionsProperty "java.lang.AssertionError  at DaisyUserThread.run(D..."

====================================================== statistics
elapsed time:       00:00:01
states:             new=1758,visited=1454,backtracked=3134,end=0
search:             maxDepth=78,constraints=132
choice generators:  thread=1682 (signal=42,lock=325,sharedRef=1077,threadApi=3,reschedule=235), data=0
heap:               new=14740,released=921,maxLive=1006,gcCycles=2740
instructions:       386218
max memory:         150MB
loaded code:        classes=90,methods=1941
```

Figure 3: Results of JFP with shortest depth (78) which still gives an error

We see that by omitting to restrain the depth, the program takes up to 54s and 135 000 states to detect an error. However by limiting the depth to around 100, we limit the time to 2s and 1000 states. This is a significant time, computation and memory improvement.

Upon inspection of the program log, we see that the error happens when the file that we use is deleted and after we attempt to read it. This is obviously an erroneous execution and should throw an error.

There are many solutions to this problem, however, we wanted to keep the behaviour where an operation to a deleted file would throw an error as we belive it is the proper way for this program to behave. Instead of changing how the program operates, we changed how our tests would operate. Thus we made it so the random operation selector would never select a deletion. This is to avoid the case where a program would delete a file and in a subsequent iteration attempt to use this same file.

This single fix solved all errors reported by JPF. Thus we did not identify any superfluous assertions in the program for Q5 since no failed assertion was reported.

3 Correct Program

The statistics for the corrected program with no depth limit are as follows:

```
====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:04:44
states:             new=639816,visited=930012,backtracked=1569828,end=3
search:             maxDepth=2693,constraints=0
choice generators:  thread=639816 (signal=5674,lock=79925,sharedRef=398904,threadApi=1901,reschedule=153412), data=0
heap:               new=1880436,released=155244,maxLive=1009,gcCycles=1387354
instructions:       47556464
max memory:         413MB
loaded code:        classes=86,methods=1900
```

Figure 4: Statistics for correct program

We measured about 640 000 states and an execution of 5 minutes with a maximum used memory of 413MB which seems to be a cap set by the computer on which we executed the programs since subsequent runs of JPF will all show a max memory of 413MB.

4 Various Conditions

4.1 Thread#2 Write

```
====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:00:02
states:             new=11199,visited=14828,backtracked=26027,end=0
search:             maxDepth=100,constraints=509
choice generators:  thread=10970 (signal=174,lock=1478,sharedRef=7165,threadApi=3,reschedule=2150), data=0
heap:               new=29055,released=2242,maxLive=1006,gcCycles=22905
instructions:       1105863
max memory:         413MB
loaded code:        classes=86,methods=1900
```

Figure 5: Statistics for thread 2 write operation

This figure will serve as a baseline to start our analysis. We can see that the number of states is of about 11 000 and since the `elapsed time` is not precise enough and the `max memory` metric is constant, we will use the number of new states for the analysis.

We will use this formula to measure the approximate number of new states that the test case should have compared to the baseline ($S_{t-1} = 11\ 000$).

$$S_t = S_{t-1} \cdot \frac{\text{New Concurrent Test Cases}}{\text{Baseline Concurrent Test Cases}} \tag{2}$$

4.2 Thread#2 Random Operation

```
====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:00:10
states:             new=51379,visited=68309,backtracked=119688,end=0
search:             maxDepth=100,constraints=2391
choice generators:  thread=50114 (signal=911,lock=7205,sharedRef=31915,threadApi=3,reschedule=10080), data=187
heap:               new=140544,released=12521,maxLive=1007,gcCycles=104760
instructions:       5901770
max memory:         414MB
loaded code:        classes=86,methods=1900
```

Figure 6: Statistics for thread 2 random operation

According to the formula (2), with an additional random operation we should see an increase of the states S_t:

$$S_t = S_{t-1} \cdot \frac{(5^1 \cdot 1)^{1 \cdot 1}}{(5^0 \cdot 1)^{1 \cdot 1}} = 11\ 000 \cdot 5 = 55\ 000$$

Since the first thread is constant we can ignore it in this ratio. Also note that since we removed the random deletion operation, the number of radom operations are of 5 instead of 6.

We can observe that the actual number being about 52 000, our actual ratio of $52\ 000 \div 11\ 000 = 4.7$ is very close to 5 and the results seem accurate.

4.3 Thread#2 Random Operation Twice

```
====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:00:10
states:             new=52621,visited=69291,backtracked=121912,end=0
search:             maxDepth=100,constraints=2629
choice generators:  thread=51190 (signal=911,lock=7395,sharedRef=32693,threadApi=3,reschedule=10188), data=207
heap:               new=140341,released=11926,maxLive=1007,gcCycles=106716
instructions:       5720586
max memory:         413MB
loaded code:        classes=86,methods=1900
```

Figure 7: Statistics for thread 2 random operation twice

According to the formula (2), with two additional random operations we should see an increase of the states S_t:

$$S_t = S_{t-1} \cdot \frac{(5^1 \cdot 1)^{2 \cdot 1}}{(5^0 \cdot 1)^{1 \cdot 1}} = 11\ 000 \cdot 275 = \ 000$$

Here is where the prediction seem off. The additional random iterations should blow up way more the number of test cases. The predicted 25 time ratio

is orders of magnitude away from the actual $52\ 000 \div 11\ 000 = 4.7$. Even more interesting is that the number of states do not change much from the 4.2 test case and look very similar.

4.4 2 Files

```
====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:00:08
states:             new=39373,visited=51306,backtracked=90679,end=0
search:             maxDepth=100,constraints=1912
choice generators:  thread=37803 (signal=672,lock=5553,sharedRef=24009,threadApi=3,reschedule=7566), data=701
heap:               new=114396,released=13721,maxLive=1010,gcCycles=78989
instructions:       7266330
max memory:         413MB
loaded code:        classes=86,methods=1900
```

Figure 8: Statistics for 2 files

According to the formula (2), with two additional random operations we should see an increase of the states S_t:

$$S_t = S_{t-1} \cdot \frac{(5^0 \cdot 2)^{2 \cdot 1}}{(5^0 \cdot 1)^{1 \cdot 1}} = 11\ 000 \cdot 4 = 44\ 000$$

Since both threds have two files, we predict 44 000 states which is very close to the actual 39 000 states. This prediction seems accurate

4.5 3 Files

```
====================================================== results
no errors detected

====================================================== statistics
elapsed time:       00:00:17
states:             new=83024,visited=108879,backtracked=191903,end=0
search:             maxDepth=100,constraints=4191
choice generators:  thread=80093 (signal=1485,lock=12187,sharedRef=50241,threadApi=3,reschedule=16177), data=10
27
heap:               new=263121,released=41638,maxLive=1016,gcCycles=166337
instructions:       22926717
max memory:         413MB
loaded code:        classes=86,methods=1900

====================================================== search finished: 07/05/20 1:21 PM
```

Figure 9: Statistics for 3 files

According to the formula (2), with two additional random operations we should see an increase of the states S_t:

$$S_t = S_{t-1} \cdot \frac{(5^0 \cdot 2)^{3 \cdot 1}}{(5^0 \cdot 1)^{1 \cdot 1}} = 11\ 000 \cdot 8 = 88\ 000$$

Since both threds have three files, we predict 88 000 states which is very close to the actual 80 000 states since it is in the same order of magnitude. This prediction also seems accurate.

4.6 3 Threads

```
==================================================== snapshot #1
thread java.lang.Thread:{id:0,name:main,status:WAITING,priority:5,isDaemon:false
,lockCount:0,suspendCount:0}
  waiting on: DaisyUserThread@1e9
  call stack:
        at java.lang.Thread.join(Thread.java)
        at DaisyTest.main(DaisyTest.java:57)

thread DaisyUserThread:{id:1,name:Thread-1,status:WAITING,priority:5,isDaemon:fa
lse,lockCount:1,suspendCount:0}
  waiting on: daisy.Mutex@1c0
  call stack:
        at java.lang.Object.wait(Object.java)
        at daisy.Mutex.acq(Mutex.java:30)
        at daisy.LockManager.acq(Daisy.java:67)
        at daisy.DaisyLock.lock_file(Daisy.java:178)
        at daisy.DaisyDir.openDirectory(DaisyDir.java:90)
        at daisy.DaisyDir.lookup(DaisyDir.java:251)
        at DaisyUserThread.run(DaisyUserThread.java:71)

thread DaisyUserThread:{id:3,name:Thread-3,status:WAITING,priority:5,isDaemon:fa
lse,lockCount:1,suspendCount:0}
  waiting on: daisy.Mutex@1c0
  call stack:
        at java.lang.Object.wait(Object.java)
        at daisy.Mutex.acq(Mutex.java:30)
        at daisy.LockManager.acq(Daisy.java:67)
        at daisy.DaisyLock.lock_file(Daisy.java:178)
        at daisy.DaisyDir.openDirectory(DaisyDir.java:90)
        at daisy.DaisyDir.lookup(DaisyDir.java:251)
        at DaisyUserThread.run(DaisyUserThread.java:71)
```

Figure 10: Statistics for 3 thread

For this test an error showed up which seems to indicate that there is a deadlock in the program. We see that two thread concurrently try to open the same file which apparently results in an error. Since it was mentionned on the moodle forum not to bother with it, we will not try to find the solution to this problem.

5 Conclusion

Apart from the test case 4.3, our analytic formula seems accurate. We will thus assume it is reliable unless there are many iterations for the JPF program. We also managed to fix the tests by removing the random delete operations. There are surely many ways to fix this, but we did not want to change the program behaviour. We could in the future explore other changes to the program to avoid this behaviour in the test cases.