Policy Exploration for JITDs (Java)

By

Team Datum

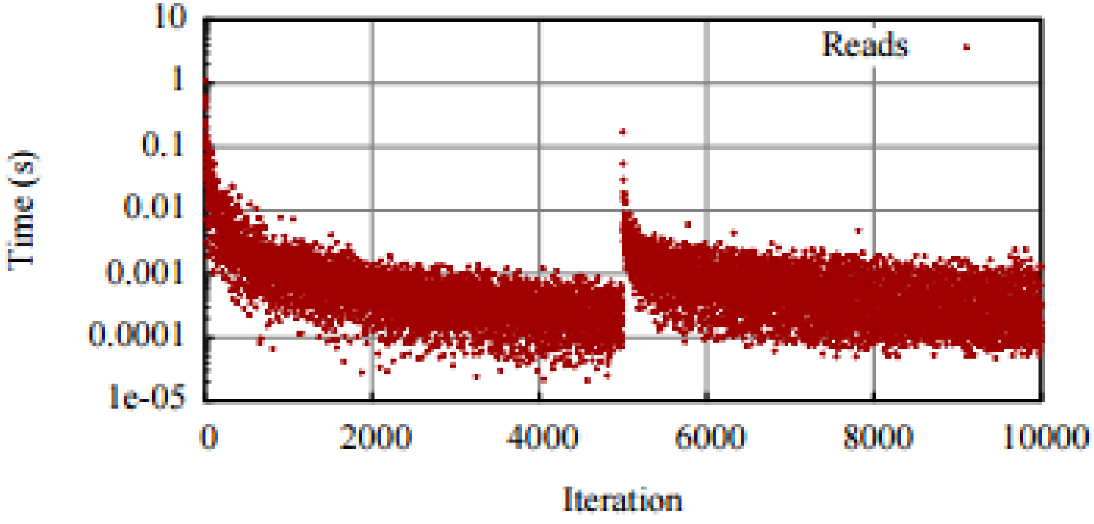
PAST : Studying Existing Implementation

- Understanding different types of Cogs: ConcatCog, ArrayCog, BTreeCog, SortedArrayCog
- Current implementation modes: Database Cracking, Adaptive Merge
- Hybrid modes : Swap, Transition

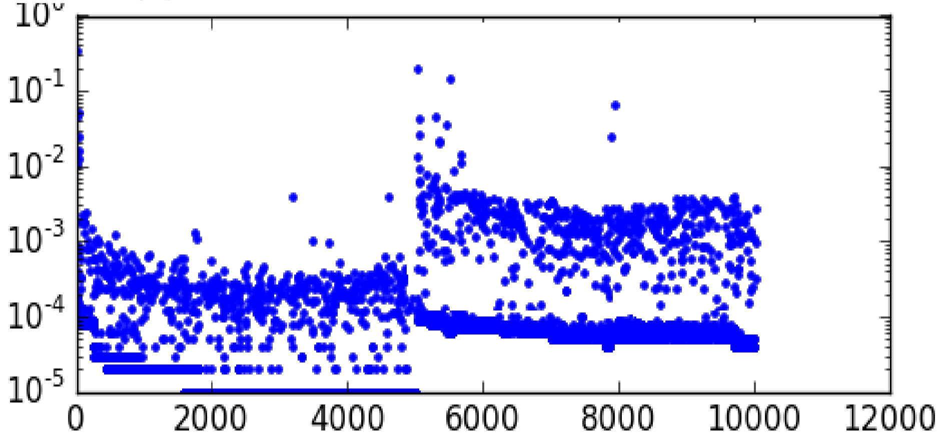
PAST: Replication

- Replicated the results from the paper by running the current implementation on many data sets of uniform distribution.
- Written Python scripts to generate graphs from the generated output file.
- Achieved near to accurate results as compared to the graphs presented in the paper for the modes:
  - Cracking
  - Adaptive Merge
  - Swap
  - Transition

PAST: Graphs – 1 (Cracking)



(a) Primitive Policy: Cracking

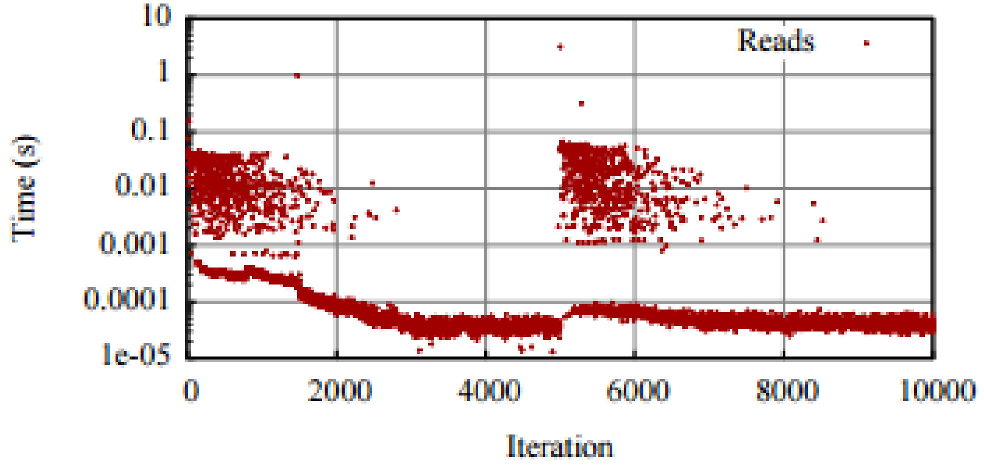


System Specifications:

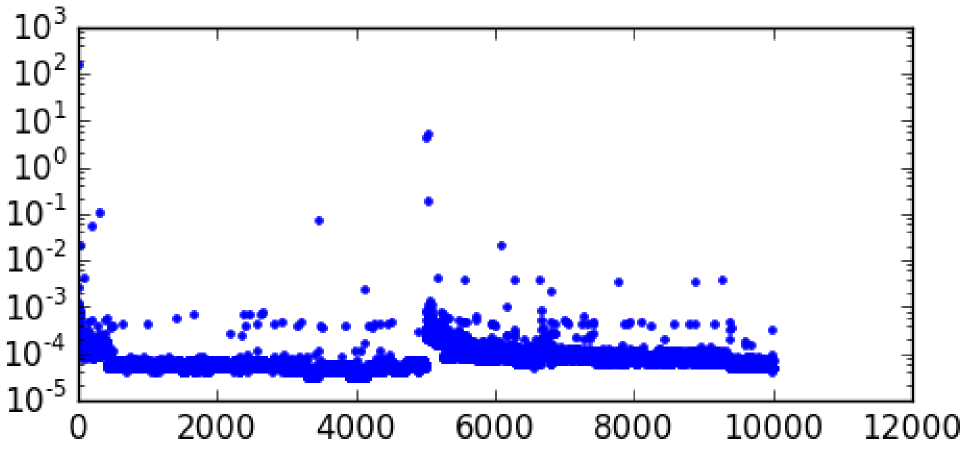
Dual core 2.40 GHz Intel i5
8 GB of RAM
Ubuntu 14.10 and JDK 1.7
JVM heap size was set to 5 GB
100MB of stack space.

Tested with :
mode cracker
init 100000000
seqread 5000
write 10000000
seqread 5000

PAST: Graphs – 2 (Adaptive Merge)



(b) Primitive Policy: Adaptive Merge



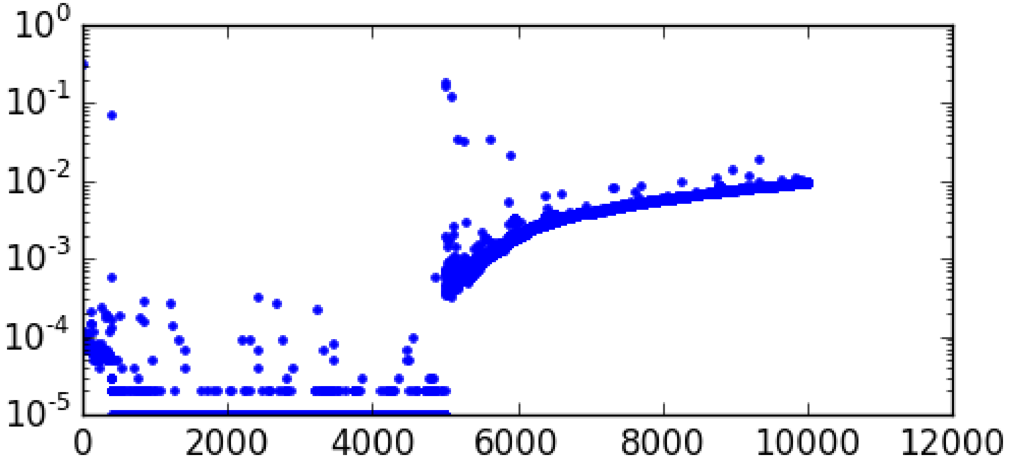
System Specifications:

Dual core 2.40 GHz Intel i5
8 GB of RAM
Ubuntu 14.10 and JDK 1.7
JVM heap size was set to 5 GB
100MB of stack space.

Tested with :
mode simplemerge
init 10000000
seqread 5000
write 10000000
seqread 5000

PAST: Challenges Faced

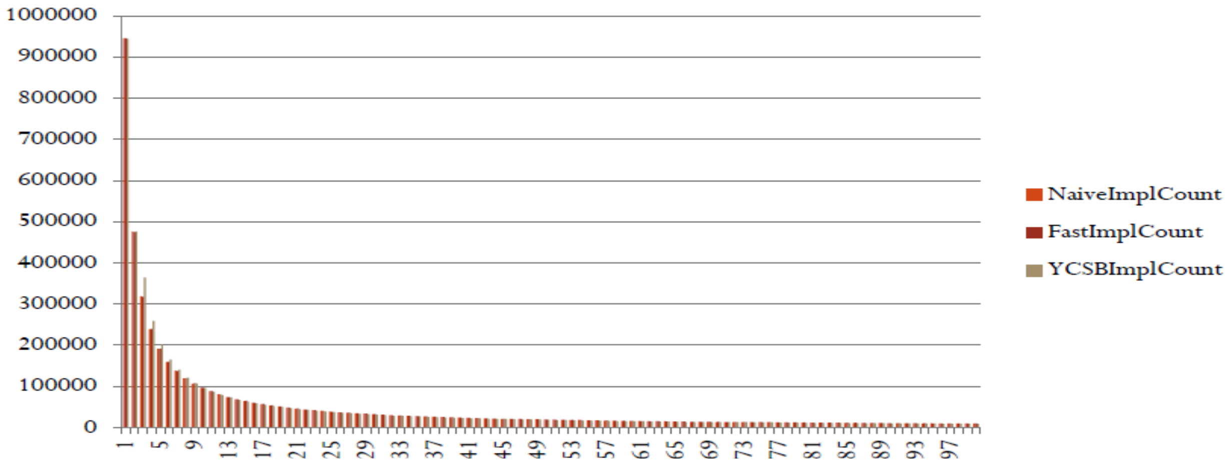
- Performance breakdown due to Garbage Collection
- Performance issue with Push Down Adaptive Merge raising the runtimes to exponentially increasing curve.
- Tuned the java garbage collector parameters to allocate high heap sizes and avoid GC invocation at runtime. Obtained similar performance.



PAST: YCSB’s Zipfian Workload

- Operated the current implementation on other workloads: Zipfian

- Challenges Faced: Irregular Cracking behavior on Zipfian workload with / without splaying.



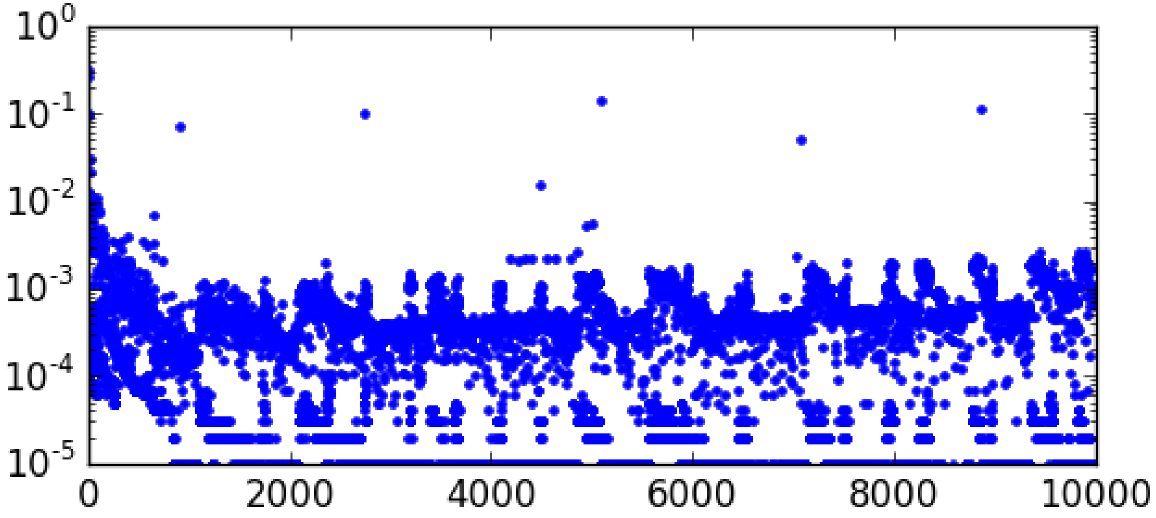


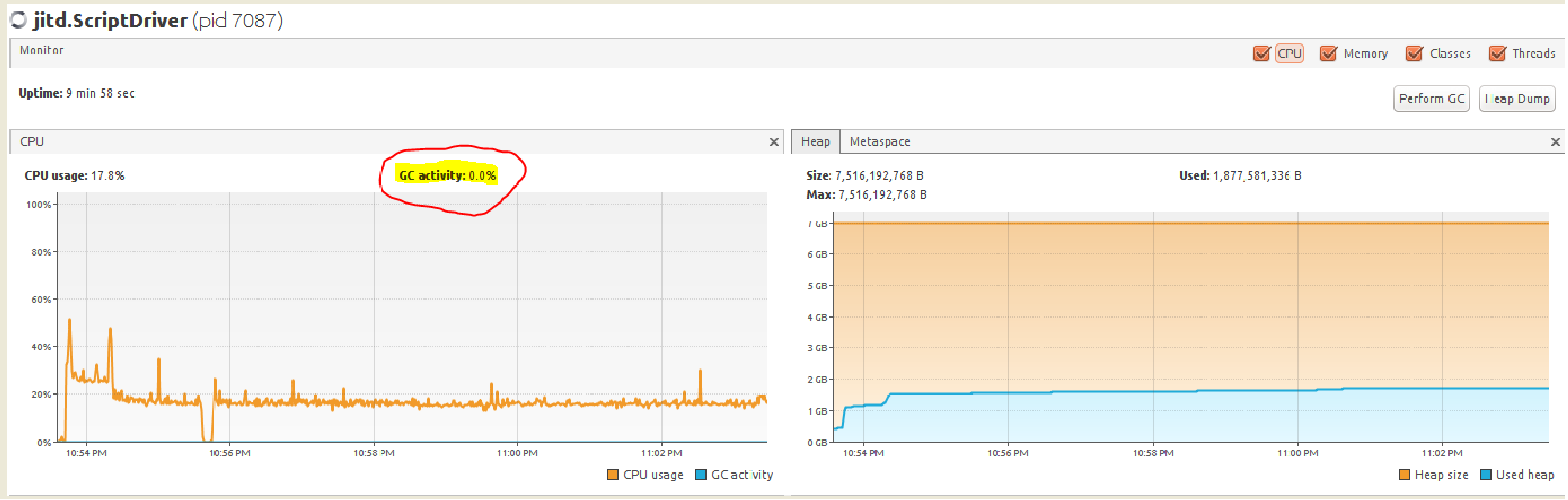
Figure *: Cracking operation with Zipfian Distribution,*
Tested as:
KeyRange1000000
Load 1000000
Reads 1000

CURRENT : Profiling

- Studying the low level implementation along with Profiling to check the space and time complexities at various blocks of code.

- Used the Profilers :
    - ➢Visual VM
    - ➢JProfiler

CURRENT : Profiling



jitd.ScriptDriver (pid 7087)

Uptime: 9 min 58 sec

CPU usage: 17.8%

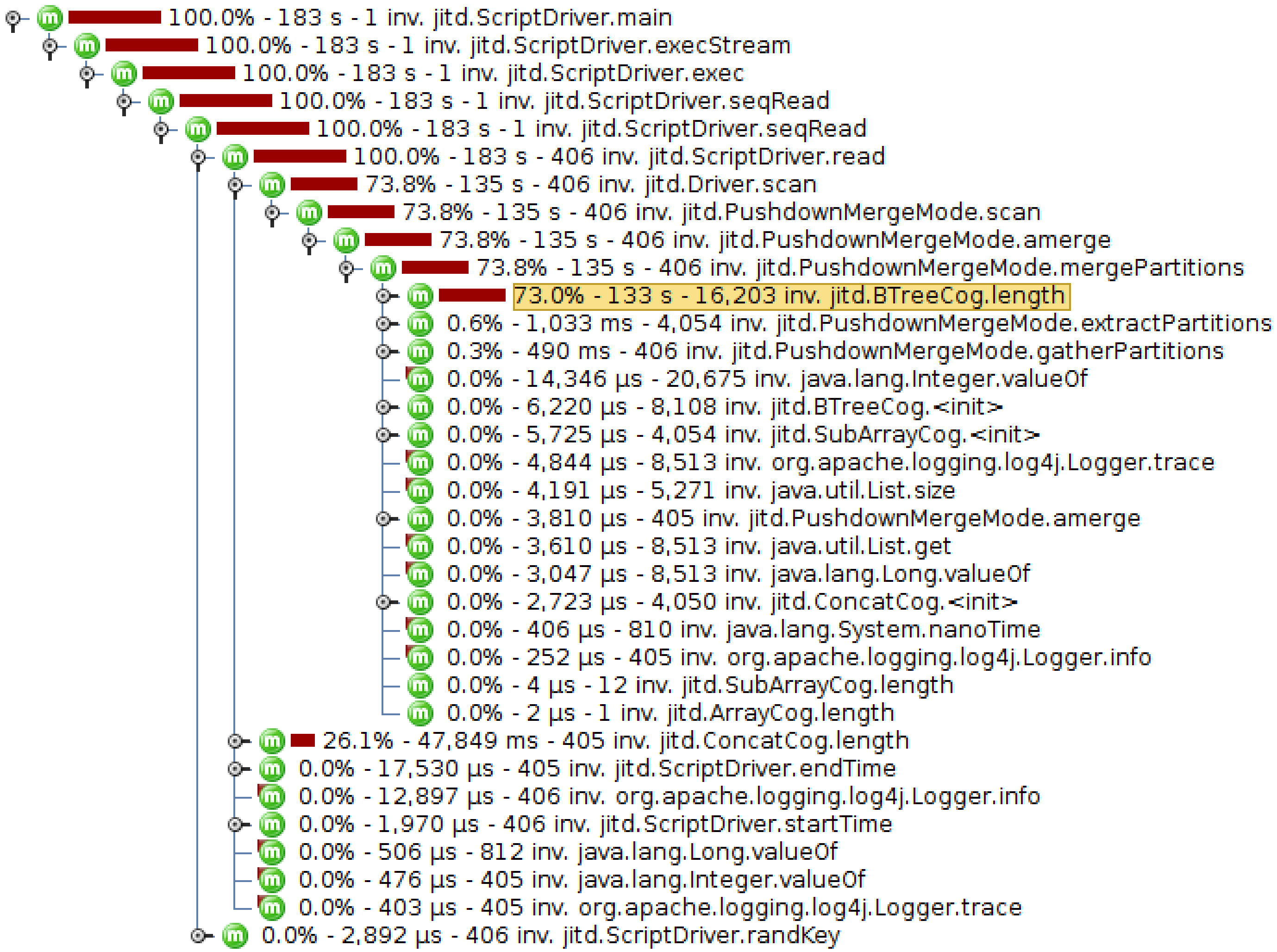
GC activity: 0.0%

Size: 7,516,192,768 B

Max: 7,516,192,768 B

Used: 1,877,581,336 B

CURRENT : Profiling

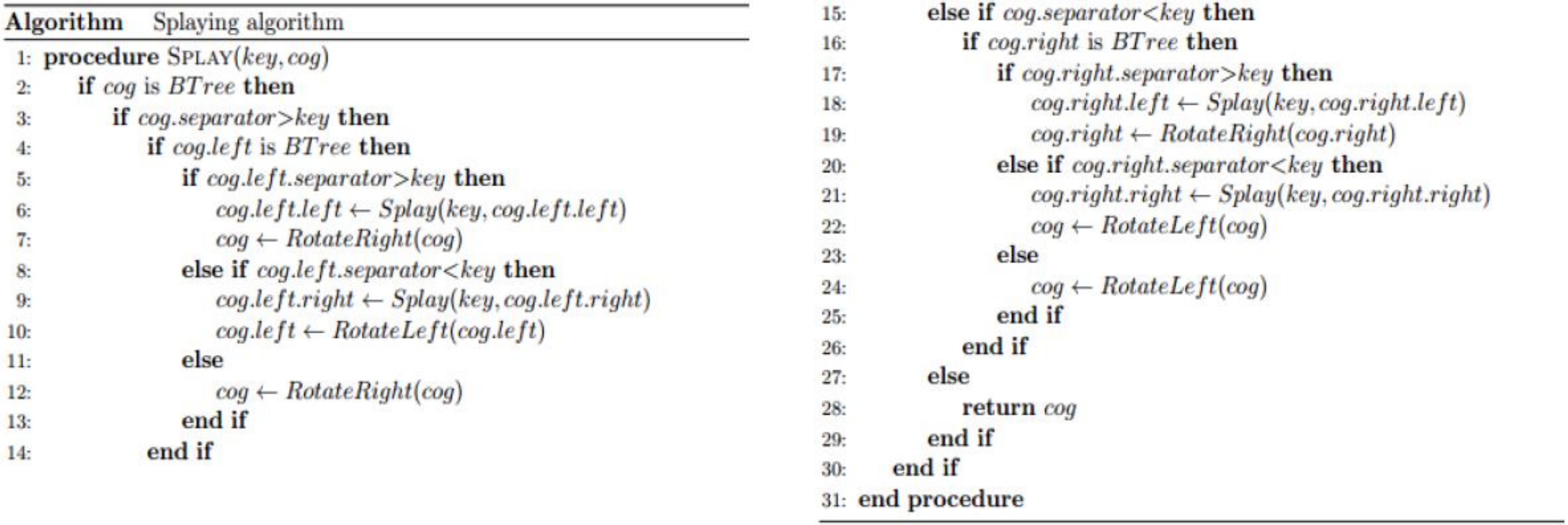


```
100.0% - 183 s - 1 inv. jitd.ScriptDriver.main
  100.0% - 183 s - 1 inv. jitd.ScriptDriver.execStream
    100.0% - 183 s - 1 inv. jitd.ScriptDriver.exec
      100.0% - 183 s - 1 inv. jitd.ScriptDriver.seqRead
        100.0% - 183 s - 1 inv. jitd.ScriptDriver.seqRead
          100.0% - 183 s - 406 inv. jitd.ScriptDriver.read
            73.8% - 135 s - 406 inv. jitd.Driver.scan
              73.8% - 135 s - 406 inv. jitd.PushdownMergeMode.scan
                73.8% - 135 s - 406 inv. jitd.PushdownMergeMode.amerge
                  73.8% - 135 s - 406 inv. jitd.PushdownMergeMode.mergePartitions
                    73.0% - 133 s - 16,203 inv. jitd.BTreeCog.length
                    0.6% - 1,033 ms - 4,054 inv. jitd.PushdownMergeMode.extractPartitions
                    0.3% - 490 ms - 406 inv. jitd.PushdownMergeMode.gatherPartitions
                    0.0% - 14,346 µs - 20,675 inv. java.lang.Integer.valueOf
                    0.0% - 6,220 µs - 8,108 inv. jitd.BTreeCog.<init>
                    0.0% - 5,725 µs - 4,054 inv. jitd.SubArrayCog.<init>
                    0.0% - 4,844 µs - 8,513 inv. org.apache.logging.log4j.Logger.trace
                    0.0% - 4,191 µs - 5,271 inv. java.util.List.size
                    0.0% - 3,810 µs - 405 inv. jitd.PushdownMergeMode.amerge
                    0.0% - 3,610 µs - 8,513 inv. java.util.List.get
                    0.0% - 3,047 µs - 8,513 inv. java.lang.Long.valueOf
                    0.0% - 2,723 µs - 4,050 inv. jitd.ConcatCog.<init>
                    0.0% - 406 µs - 810 inv. java.lang.System.nanoTime
                    0.0% - 252 µs - 405 inv. org.apache.logging.log4j.Logger.info
                    0.0% - 4 µs - 12 inv. jitd.SubArrayCog.length
                    0.0% - 2 µs - 1 inv. jitd.ArrayCog.length
            26.1% - 47,849 ms - 405 inv. jitd.ConcatCog.length
            0.0% - 17,530 µs - 405 inv. jitd.ScriptDriver.endTime
            0.0% - 12,897 µs - 406 inv. org.apache.logging.log4j.Logger.info
            0.0% - 1,970 µs - 406 inv. jitd.ScriptDriver.startTime
            0.0% - 506 µs - 812 inv. java.lang.Long.valueOf
            0.0% - 476 µs - 405 inv. java.lang.Integer.valueOf
            0.0% - 403 µs - 405 inv. org.apache.logging.log4j.Logger.trace
          0.0% - 2,892 µs - 406 inv. jitd.ScriptDriver.randKey
```

Splaying

- Added splay operations to the BTree Cog to make the root cog nearly balanced.
- It follows the traditional approaches of Zig and Zag operations of a Splay Tree.
- Currently, splay policy has been integrated into the Cracking mode which splays the root cog with the given lower bound of the range query.

Splaying Algorithm



Algorithm Splaying algorithm

```
procedure SPLAY($key, cog$)
    if $cog$ is $BTree$ then
        if $cog.separator > key$ then
            if $cog.left$ is $BTree$ then
                if $cog.left.separator > key$ then
                    $cog.left.left \leftarrow Splay(key, cog.left.left)$
                    $cog \leftarrow RotateRight(cog)$
                else if $cog.left.separator < key$ then
                    $cog.left.right \leftarrow Splay(key, cog.left.right)$
                    $cog.left \leftarrow RotateLeft(cog.left)$
                else
                    $cog \leftarrow RotateRight(cog)$
                end if
            end if
        else if $cog.separator < key$ then
            if $cog.right$ is $BTree$ then
                if $cog.right.separator > key$ then
                    $cog.right.left \leftarrow Splay(key, cog.right.left)$
                    $cog.right \leftarrow RotateRight(cog.right)$
                else if $cog.right.separator < key$ then
                    $cog.right.right \leftarrow Splay(key, cog.right.right)$
                    $cog \leftarrow RotateLeft(cog)$
                else
                    $cog \leftarrow RotateLeft(cog)$
                end if
            end if
        else
            return $cog$
        end if
    end if
end procedure
```

Work in Progress..

- Debugging splaying along with cracking.
- Analyzing the issue for null pointers during merge phase, which happens for the initial load of 100m.
- Understanding the behavior while running the implementation over YCSB’s Zipfian workloads with and without splaying.

Further works to do..

- Debugging splaying along with adaptive merge.
- Splaying with Adaptive Merge on different read / write ratios given by YCSB (Zipfian).
- Comparing the above results with uniform workloads to determine final policies.

Thank you..!!

Questions?