CS 433 (2) HW5 Report

Boris Nikulin

2019-05-11

Contents

1 Data Analysis

1.1 Data Import

First we load the simulation data, filter out processes that never finished, and give the data a once over.

However, using `readr::read_tsv` to load the data take's minutes because the data per run is around 1 GB. After all the runs of the simulation, the final dataset is 4.2 GB. As a side note, the gzip comprised size of the whole data set is 444 MB. For this reason, we will primarily use `data.table` instead of `readr` and `dplyr`. `data.table::fread` reads the whole dataset in 10 s to 20 s.

```
library(data.table)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':
##
##    between, first, last

## The following objects are masked from 'package:stats':
##
##    filter, lag

## The following objects are masked from 'package:base':
##
```

```
##    intersect, setdiff, setequal, union
```

```
library(magrittr)

data <- fread('paging_sim.tsv')

data %<>% .[, time := time / 1E6]
```

```
glimpse(data)
```

```
## Observations: 119,364,413
## Variables: 7
## $ time                  <dbl> 0.0e+00, 1.0e-06, 1.6e-05, 1.8...
## $ num_pages             <int> 524288, 524288, 524288, 524288...
## $ page_size_bytes       <int> 256, 256, 256, 256, 256, 256, ...
## $ memory_size_bytes     <int> 65536, 65536, 65536, 65536, 65...
## $ replacement_algorithm <chr> "fifo", "fifo", "fifo", "fifo"...
## $ event                 <chr> "reference", "fault", "referen...
## $ page                  <int> 397863, 397863, 397863, 281415...
```

The times are in seconds.

1.2 Data Analysis

```
run_group <- c(
    'num_pages',
    'page_size_bytes',
    'memory_size_bytes',
    'replacement_algorithm'
)

(data_time <- data[, .(time_max = max(time)), mget(run_group)])
```

```
##     num_pages page_size_bytes memory_size_bytes
##  1:    524288             256             65536
##  2:    524288             256             65536
##  3:    524288             256             65536
##  4:    524288             256             65536
##  5:    524288             256          16777216
##  6:    524288             256          16777216
##  7:    524288             256          16777216
##  8:    524288             256          16777216
##  9:     16384            8192             65536
## 10:     16384            8192             65536
## 11:     16384            8192             65536
## 12:     16384            8192             65536
## 13:     16384            8192          16777216
```

```
## 14:     16384            8192          16777216
## 15:     16384            8192          16777216
## 16:     16384            8192          16777216
##     replacement_algorithm  time_max
##  1:                  fifo 14.284713
##  2:           rng:mt19937  6.528269
##  3:       rng:minstd_rand  6.686142
##  4:                   lru 16.293392
##  5:                  fifo  7.209118
##  6:           rng:mt19937  6.364927
##  7:       rng:minstd_rand  6.473702
##  8:                   lru  9.296487
##  9:                  fifo 14.972721
## 10:           rng:mt19937  5.647755
## 11:       rng:minstd_rand  5.795382
## 12:                   lru 16.359806
## 13:                  fifo  6.653178
## 14:           rng:mt19937  5.742128
## 15:       rng:minstd_rand  5.787111
## 16:                   lru  8.094985
```

```
(data_event_count <- data[, .N, mget(append(run_group, 'event'))])
```

```
##     num_pages page_size_bytes memory_size_bytes
##  1:    524288             256             65536
##  2:    524288             256             65536
##  3:    524288             256             65536
##  4:    524288             256             65536
##  5:    524288             256             65536
##  6:    524288             256             65536
##  7:    524288             256             65536
##  8:    524288             256             65536
##  9:    524288             256             65536
## 10:    524288             256             65536
## 11:    524288             256             65536
## 12:    524288             256             65536
## 13:    524288             256             65536
## 14:    524288             256             65536
## 15:    524288             256             65536
## 16:    524288             256             65536
## 17:    524288             256          16777216
## 18:    524288             256          16777216
## 19:    524288             256          16777216
## 20:    524288             256          16777216
## 21:    524288             256          16777216
## 22:    524288             256          16777216
```

```
## 23:    524288             256          16777216
## 24:    524288             256          16777216
## 25:    524288             256          16777216
## 26:    524288             256          16777216
## 27:    524288             256          16777216
## 28:    524288             256          16777216
## 29:    524288             256          16777216
## 30:    524288             256          16777216
## 31:    524288             256          16777216
## 32:    524288             256          16777216
## 33:     16384            8192             65536
## 34:     16384            8192             65536
## 35:     16384            8192             65536
## 36:     16384            8192             65536
## 37:     16384            8192             65536
## 38:     16384            8192             65536
## 39:     16384            8192             65536
## 40:     16384            8192             65536
## 41:     16384            8192             65536
## 42:     16384            8192             65536
## 43:     16384            8192             65536
## 44:     16384            8192             65536
## 45:     16384            8192             65536
## 46:     16384            8192             65536
## 47:     16384            8192             65536
## 48:     16384            8192             65536
## 49:     16384            8192          16777216
## 50:     16384            8192          16777216
## 51:     16384            8192          16777216
## 52:     16384            8192          16777216
## 53:     16384            8192          16777216
## 54:     16384            8192          16777216
## 55:     16384            8192          16777216
## 56:     16384            8192          16777216
## 57:     16384            8192          16777216
## 58:     16384            8192          16777216
## 59:     16384            8192          16777216
## 60:     16384            8192          16777216
## 61:     16384            8192          16777216
## 62:     16384            8192          16777216
## 63:     16384            8192          16777216
## 64:     16384            8192          16777216
##     num_pages page_size_bytes memory_size_bytes
##     replacement_algorithm     event       N
##  1:                  fifo reference 5000000
```

```
##  2:                fifo     fault 2778912
##  3:                fifo   replace 2778656
##  4:                fifo     flush 2483210
##  5:          rng:mt19937 reference 5000000
##  6:          rng:mt19937     fault  336407
##  7:          rng:mt19937   replace  336151
##  8:          rng:mt19937     flush   61818
##  9:      rng:minstd_rand reference 5000000
## 10:      rng:minstd_rand     fault  336087
## 11:      rng:minstd_rand   replace  335831
## 12:      rng:minstd_rand     flush   61562
## 13:                 lru reference 5000000
## 14:                 lru     fault 2632039
## 15:                 lru   replace 2631783
## 16:                 lru     flush 2336294
## 17:                fifo reference 5000000
## 18:                fifo     fault  514905
## 19:                fifo   replace  449369
## 20:                fifo     flush  300719
## 21:          rng:mt19937 reference 5000000
## 22:          rng:mt19937     fault  332699
## 23:          rng:mt19937   replace  267163
## 24:          rng:mt19937     flush   58969
## 25:      rng:minstd_rand reference 5000000
## 26:      rng:minstd_rand     fault  332344
## 27:      rng:minstd_rand   replace  266808
## 28:      rng:minstd_rand     flush   58657
## 29:                 lru reference 5000000
## 30:                 lru     fault  482163
## 31:                 lru   replace  416627
## 32:                 lru     flush  266250
## 33:                fifo reference 5000000
## 34:                fifo     fault 2965735
## 35:                fifo   replace 2965727
## 36:                fifo     flush 2943957
## 37:          rng:mt19937 reference 5000000
## 38:          rng:mt19937     fault   31530
## 39:          rng:mt19937   replace   31522
## 40:          rng:mt19937     flush   16567
## 41:      rng:minstd_rand reference 5000000
## 42:      rng:minstd_rand     fault   31487
## 43:      rng:minstd_rand   replace   31479
## 44:      rng:minstd_rand     flush   16468
## 45:                 lru reference 5000000
## 46:                 lru     fault 2663297
```

```
## 47:                   lru   replace 2663289
## 48:                   lru     flush 2641522
## 49:                  fifo reference 5000000
## 50:                  fifo     fault  249038
## 51:                  fifo   replace  246990
## 52:                  fifo     flush  233736
## 53:           rng:mt19937 reference 5000000
## 54:           rng:mt19937     fault   31497
## 55:           rng:mt19937   replace   29449
## 56:           rng:mt19937     flush   16505
## 57:       rng:minstd_rand reference 5000000
## 58:       rng:minstd_rand     fault   31249
## 59:       rng:minstd_rand   replace   29201
## 60:       rng:minstd_rand     flush   16314
## 61:                   lru reference 5000000
## 62:                   lru     fault  213375
## 63:                   lru   replace  211327
## 64:                   lru     flush  197729
##     replacement_algorithm     event       N
```

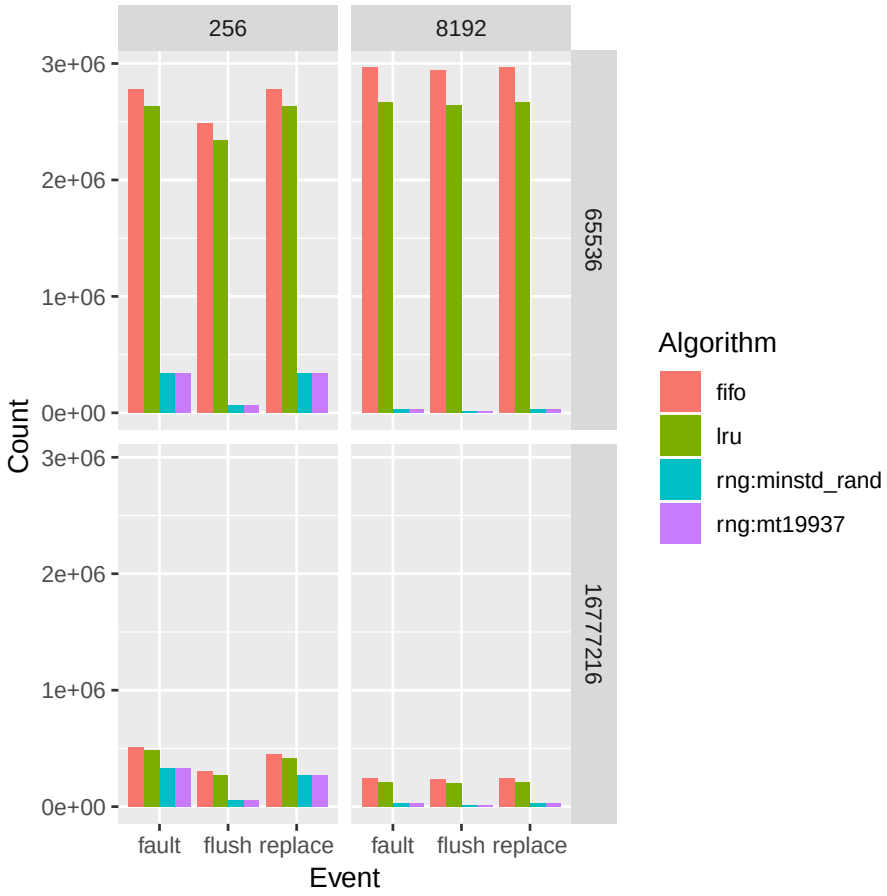
```
data_event_count %<>% .[event != 'reference']
```

Another side note, awk takes around 10 s per simulation to generate the counts while `data.table` gets all the counts at once in under 10 s.

1.3 Data Visualization

```
library(ggplot2)

ggplot(data_event_count,
       aes(event, N, fill = replacement_algorithm)) +
    geom_col(position = 'dodge') +
    facet_grid(memory_size_bytes ~ page_size_bytes) +
    labs(x = 'Event', y = 'Count', fill = 'Algorithm')
```



rng:mt19937 refers to the C++ `random` library and is a 32 bit mersenne twister default constructed. rng:minst_rand also corresponds to the `random` library and is the newer 1993 minimum standard LCG. It is also default constructed.

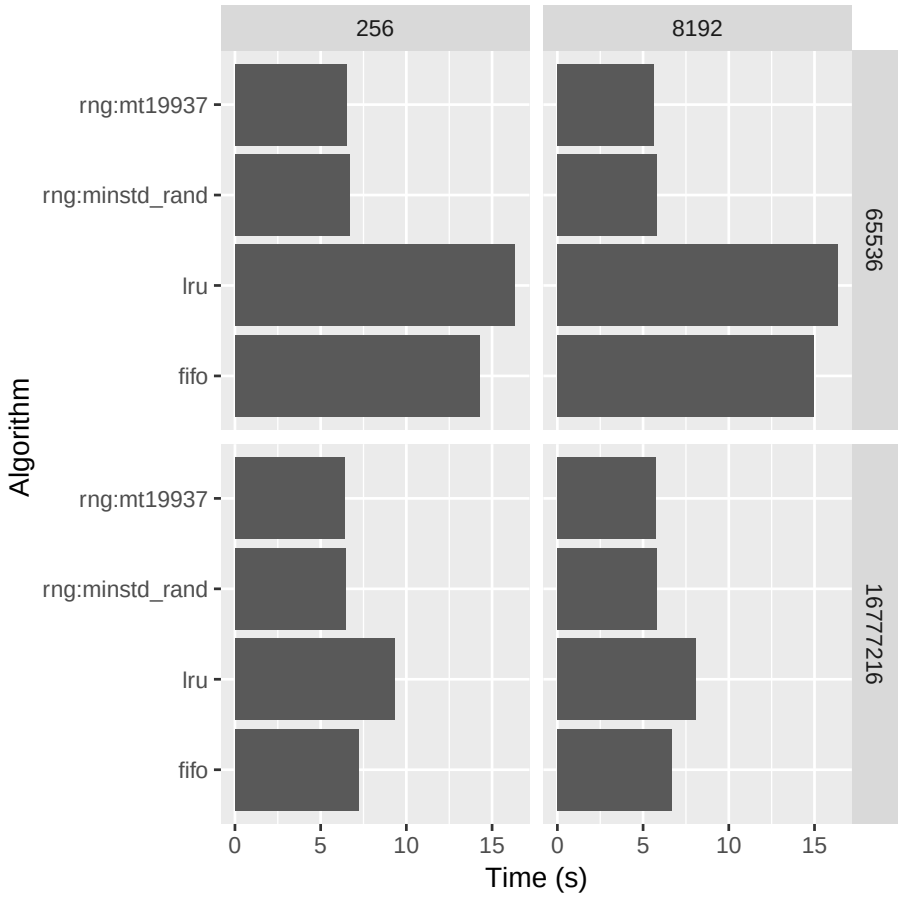
The bar graph above has some key takeaways. Page size, shown in the horizontal facets, does not reduce the event counts nearly as much as memory size, show in the vertical facets. At the end of the day, more memory is best.

Another takeaway is that when there is insufficient memory, page size has no effect or a small adverse affect on the number of events excluding the random algorithm. Page size does have an effect when memory size is not the limiting factor.

Lastly, random page replacement is unfairly good for how simple the algorithm is and is primarily affected by page size unlike the other algorithms.

```
ggplot(data_time, aes(replacement_algorithm, time_max)) +
    geom_col() +
```

```
    facet_grid(memory_size_bytes ~ page_size_bytes) +
    labs(x = 'Algorithm', y = 'Time (s)') +
    coord_flip()
```



The times for the final events, which included the time to log each event, are pretty similar.

I expected LRU to take longer as LRU is the same FIFO queue, but with an extra table for book keeping.

What surprised me was the two different random number generation algorithms taking nearly the same amount of time. I expected for the mersenne twister to take longer than the LCG due to the complexity differences between them. However, the LCG took around 0 s to 0.1 s longer in every case.