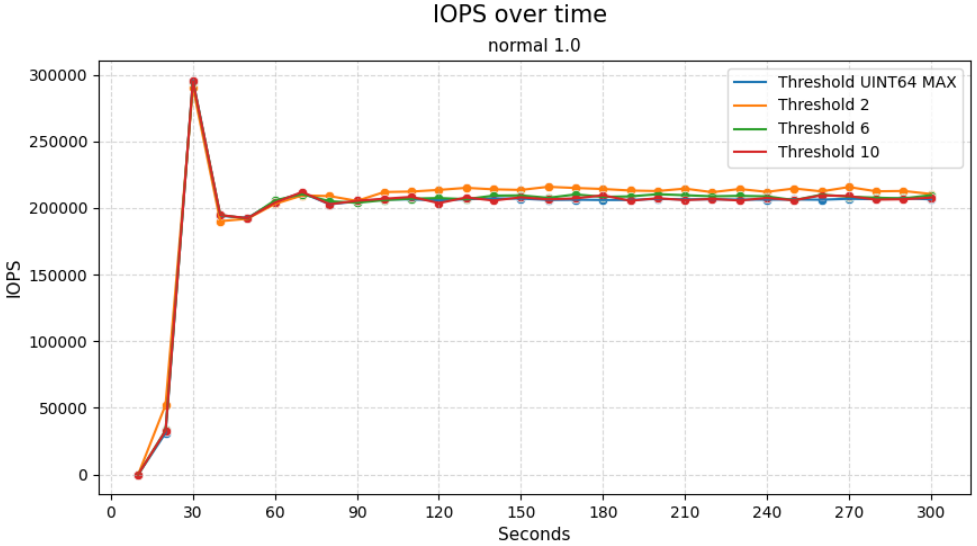
Normal 1.0

```
fio --name test --directory=/mnt/nvme0n1/ --size=576M --direct=1 --bs=4k --numjobs=4
    --time_based --runtime=300 --rw=randrw --random_distribution=normal:1.0
```



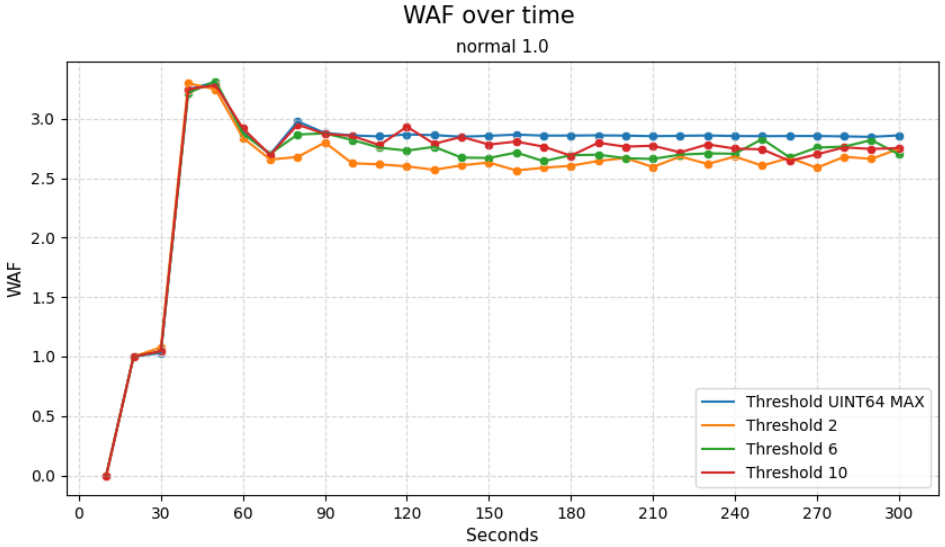
Average IOPS (higher is better)

Threshold UINT64 MAX
195758.83333

Threshold 2
201124.36667

Threshold 6
197164.43333

Threshold 10
196187.70000



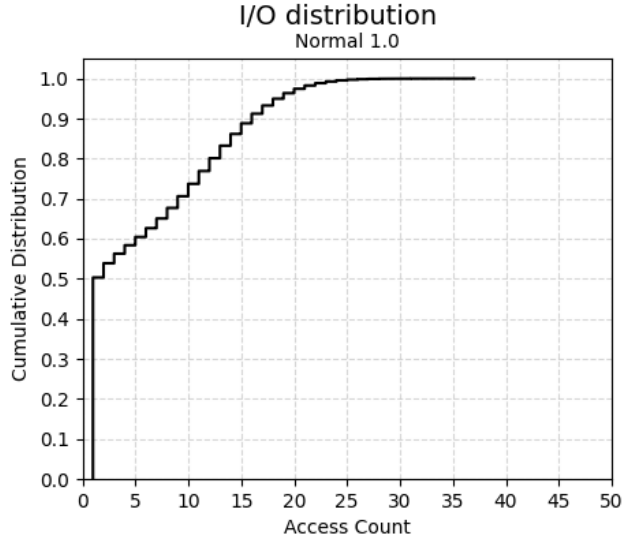
Average WAF (lower is better)

Threshold UINT64 MAX
2.66846

Threshold 2
2.49603

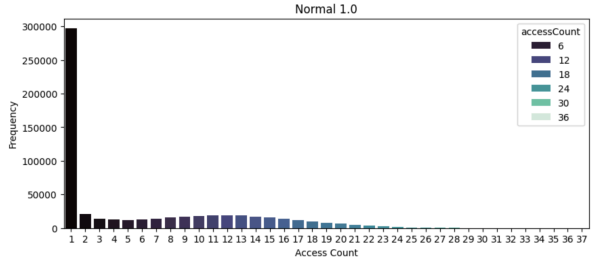
Threshold 6
2.56964

Threshold 10
2.60756



Normal 1.0 is a distribution
with a standard deviation of 1.0.

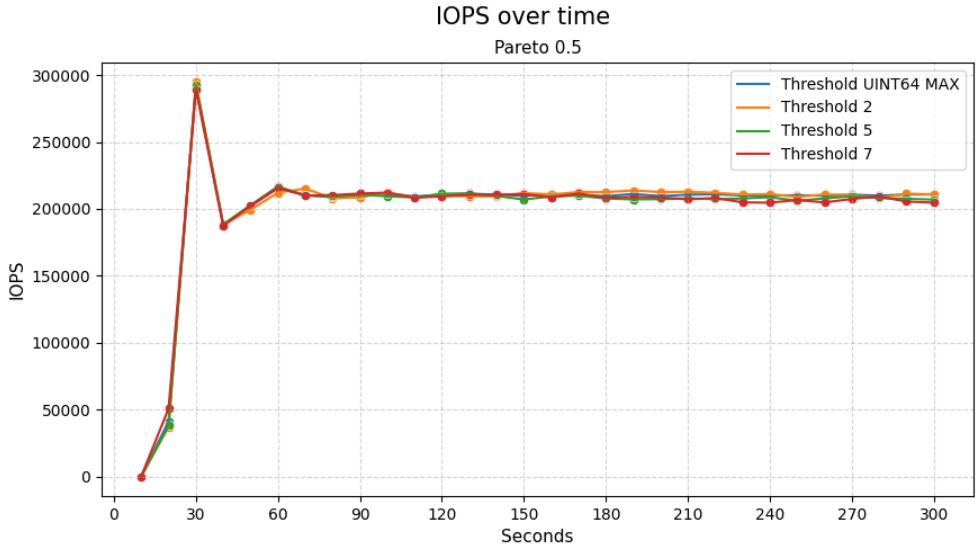
It has the characteristic that
many values are located near the average.



In the actual hot/cold separation,
It has been confirmed that
it has the best performance at threshold 2.
This point is when the cumulative distribution is around 0.5.

Pareto 0.5

fio --name test --directory=/mnt/nvme0n1/ --size=576M --direct=1 --bs=4k --numjobs=4 --time_based --runtime=300 --rw=randrw --random_distribution=pareto:0.5



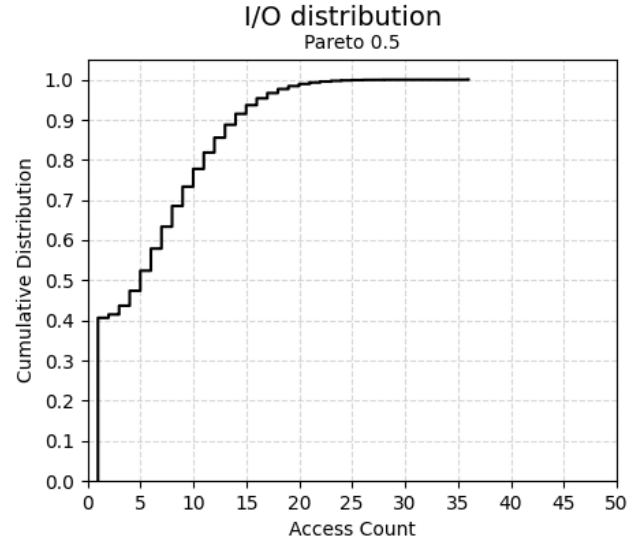
Average IOPS (higher is better)

Threshold UINT64 MAX
199586.90000

Threshold 2
199822.53333

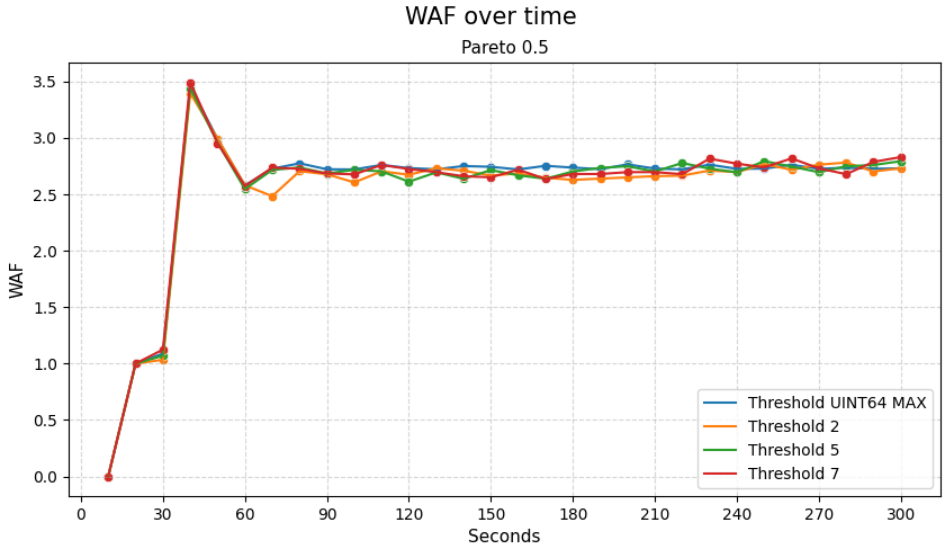
Threshold 5
198183.90000

Threshold 7
198309.83333



Pareto distribution can be understood as the **80/20 rule.**

Here, the α is 0.5. **As α gets closer to 0, some values become more important.**



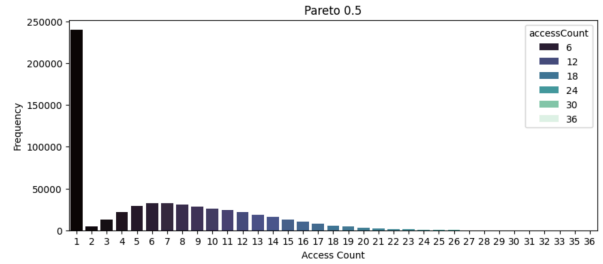
Average WAF (lower is better)

Threshold UINT64 MAX
2.56088

Threshold 2
2.51417

Threshold 5
2.53955

Threshold 7
2.54793

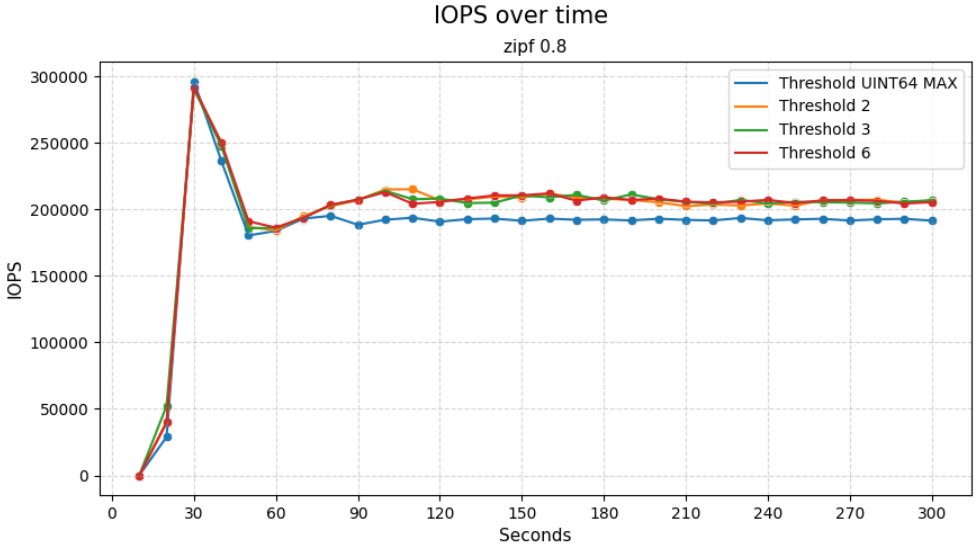


In the actual hot/cold separation,
It shows a better performance than other values at threshold 2.

Due to the characteristics of the pareto 0.5 distribution, the range of performance changes is small.

Zipf 0.8

fio --name test --directory=/mnt/nvme0n1/ --size=576M --direct=1 --bs=4k --numjobs=4
--time_based --runtime=300 --rw=randrw --random_distribution=zipf:0.8



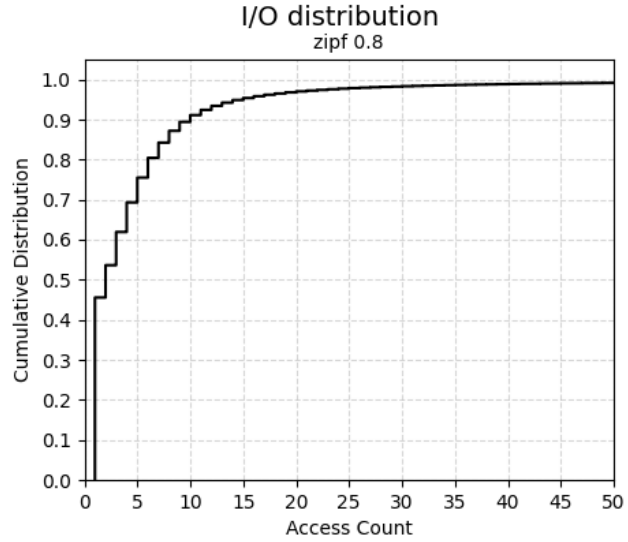
Average IOPS (higher is better)

Threshold UINT64 MAX
184808.03333

Threshold 2
197047.20000

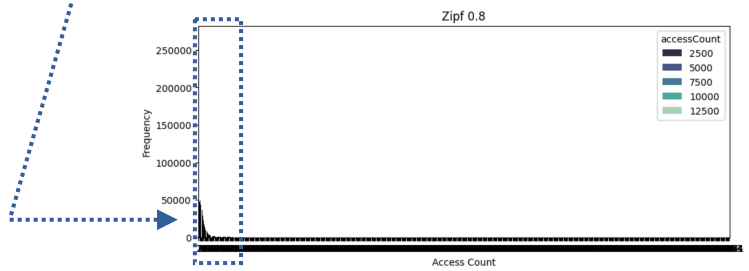
Threshold 3
197311.00000

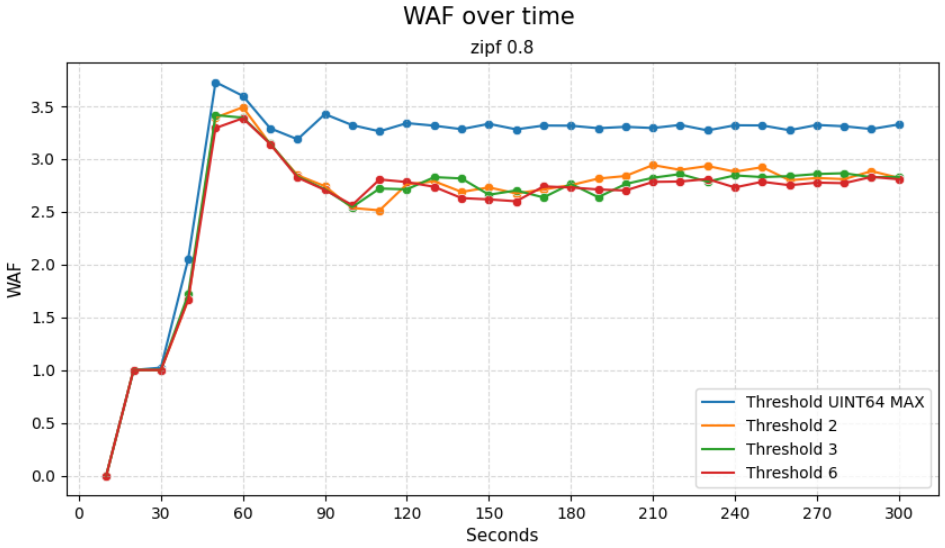
Threshold 6
197327.60000



The zipf distribution can be seen as a distribution
where some values are the majority of the total.

Here, the slope is 0.8.
As the slope increases,
it takes up more and more of the total.





Average WAF (lower is better)

Threshold UINT64 MAX
3.02412

Threshold 2
2.59349

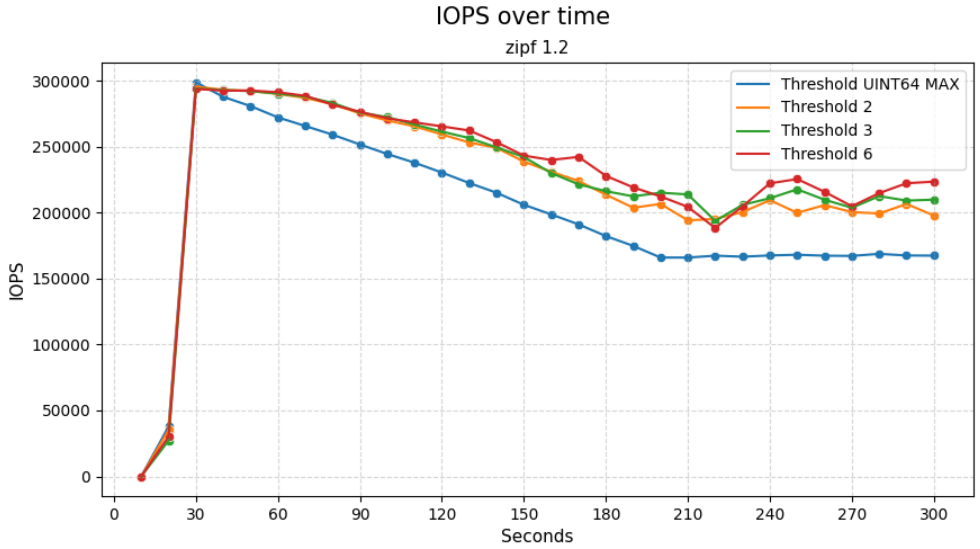
Threshold 3
2.57891

Threshold 6
2.54921

In the actual hot/cold separation,
it shows the highest performance at threshold 6.
Also, the characteristics of zipf 0.8 are revealed
by the frequency of some values.

Zipf 1.2

fio --name test --directory=/mnt/nvme0n1/ --size=576M --direct=1 --bs=4k --numjobs=4 --time_based --runtime=300 --rw=randrw --random_distribution=zipf:1.2



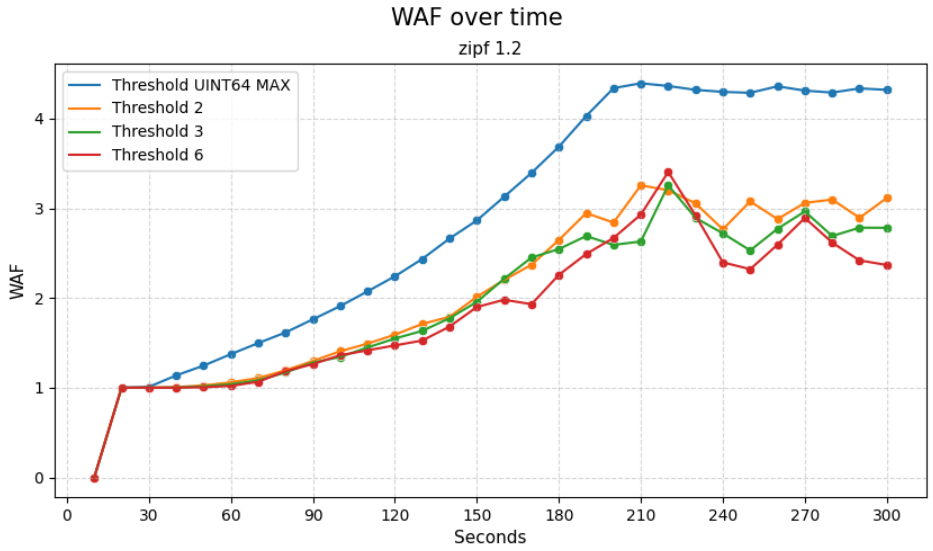
Average IOPS (higher is better)

Threshold UINT64 MAX
196544.70000

Threshold 2
222464.16667

Threshold 3
225817.86667

Threshold 6
229311.50000



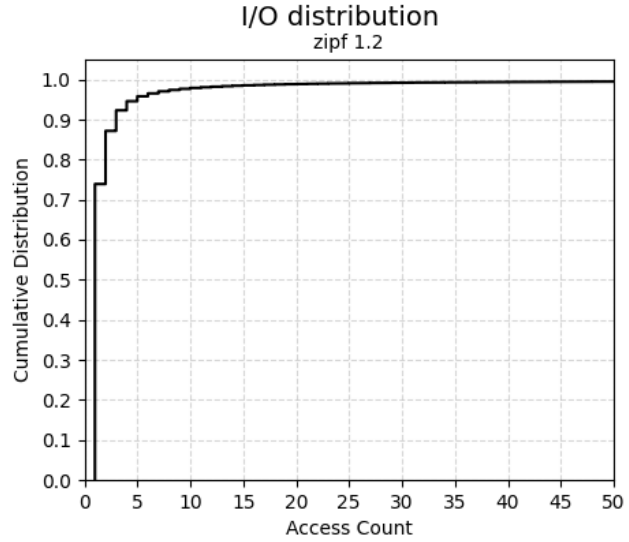
Average WAF (lower is better)

Threshold UINT64 MAX
2.89010

Threshold 2
2.07028

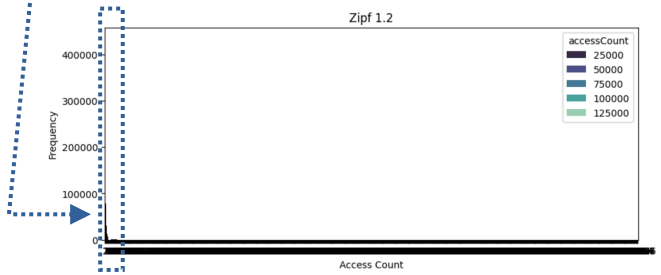
Threshold 3
1.96043

Threshold 6
1.86982



The zipf distribution can be seen as a distribution **where some values are the majority of the total.**

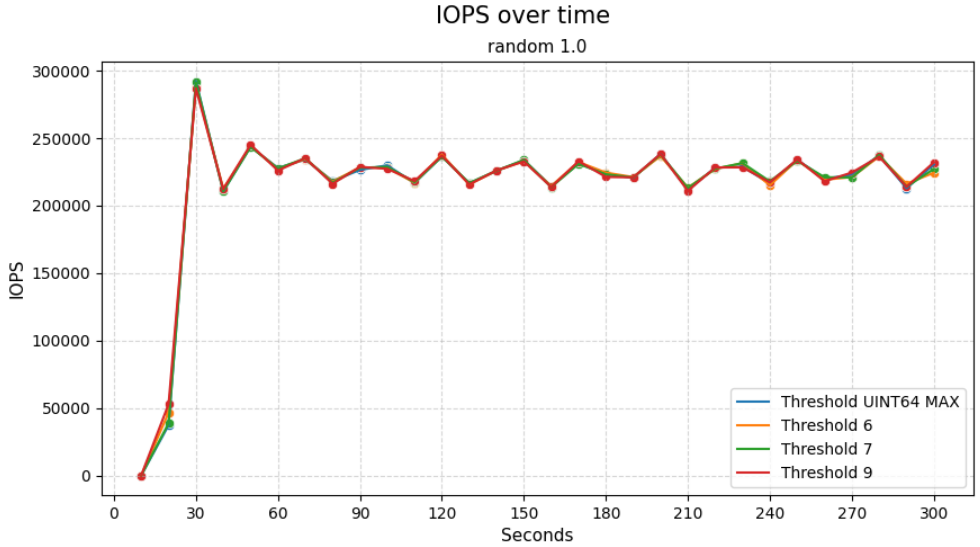
Here, the slope is 1.2.
The slope is steeper than the 0.8 confirmed earlier.



In the actual hot/cold separation,
it shows the best performance at threshold 6.
IOPS and WAF have changed significantly.
Also, the characteristics of zipf 1.2 are revealed by the frequency of some values.

Random 1.0

fio --name test --directory=/mnt/nvme0n1/ --size=576M --direct=1 --bs=4k --numjobs=4 --time_based --runtime=300 --rw=randrw --random_distribution=random:1.0



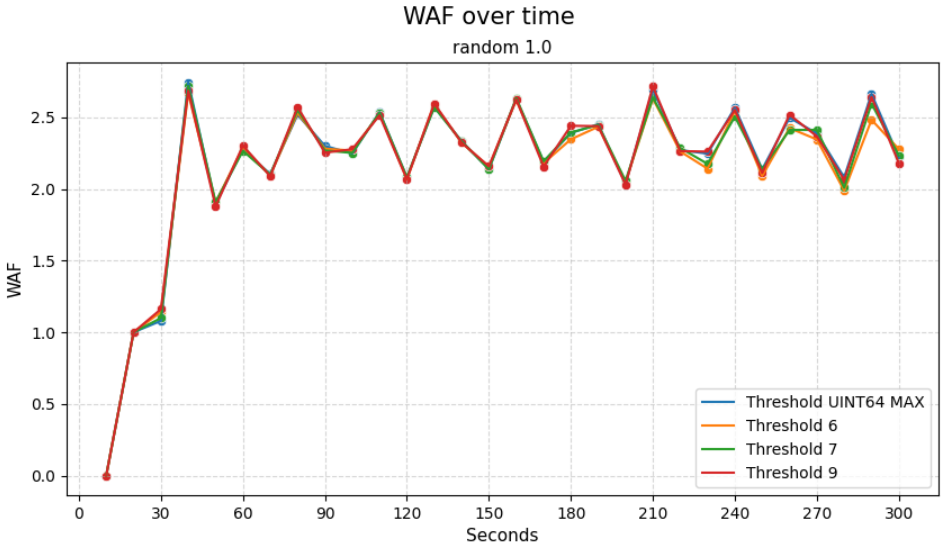
Average IOPS (higher is better)

Threshold UINT64 MAX
213997.30000

Threshold 6
214139.10000

Threshold 7
214044.83333

Threshold 9
214392.86667



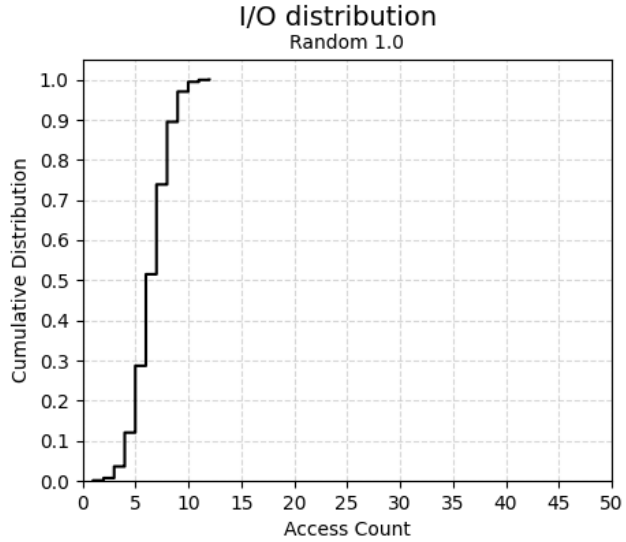
Average WAF (lower is better)

Threshold UINT64 MAX
2.17674

Threshold 6
2.15575

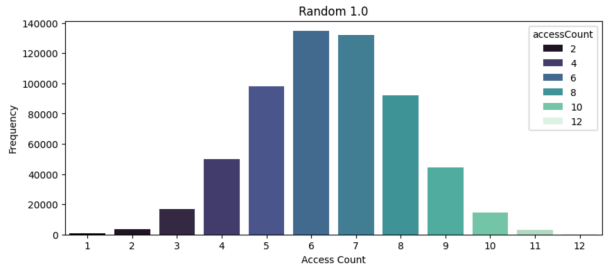
Threshold 7
2.16521

Threshold 9
2.17502



In fio, random 1.0 is uniform distribution.

It means that
the probability of occurrence of all values is the same.

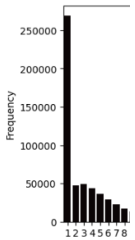


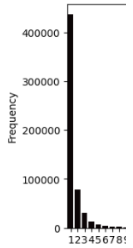
Hot/cold is a feature of workload.
Therefore, **there is no change in performance within the error range** when hot/cold is not separated, or when 6, 7, 9 is set as a threshold.

Additional analysis

About distribution and LPN

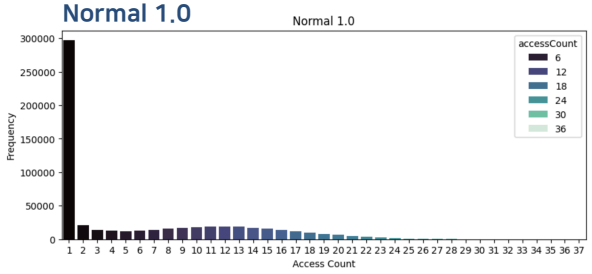
In the LPN trend graph,
it is confirmed that
**only some LPNs are accessed,
not all LPNs.**

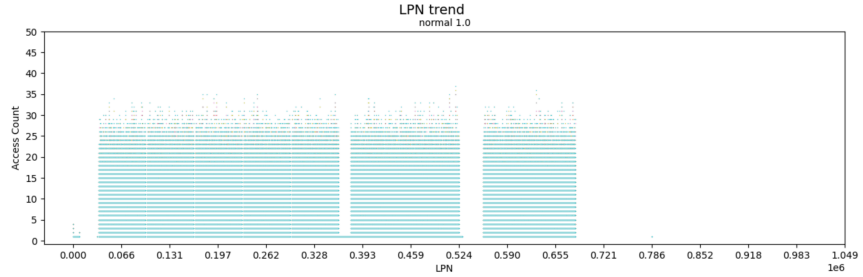




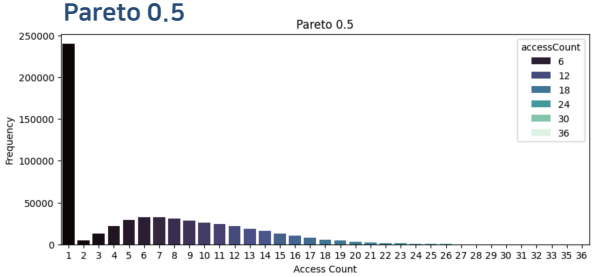
**The frequency change of some values
due to the change of zipf slope
has been confirmed.**

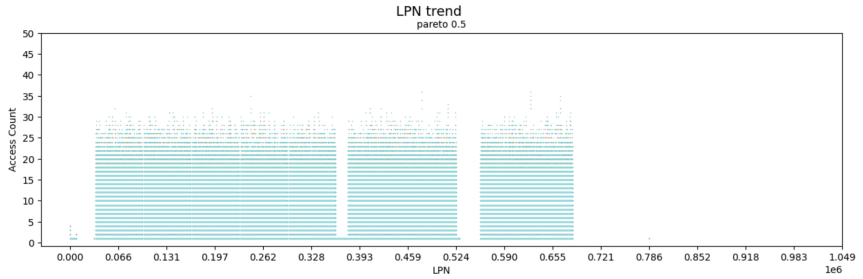
Normal 1.0





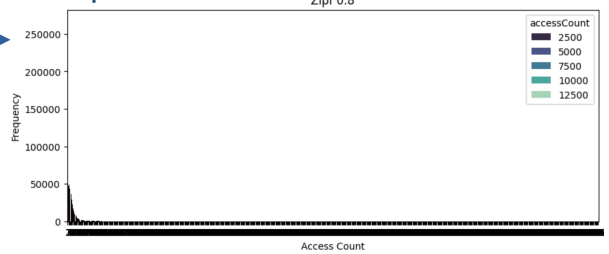
Pareto 0.5

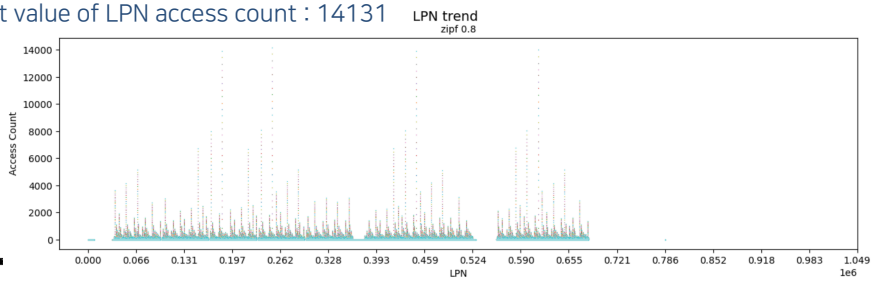




Zipf 0.8

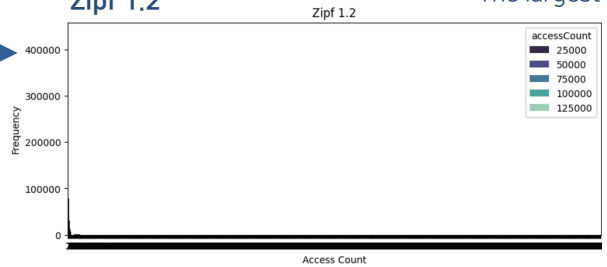
The largest value of LPN access count : 14131

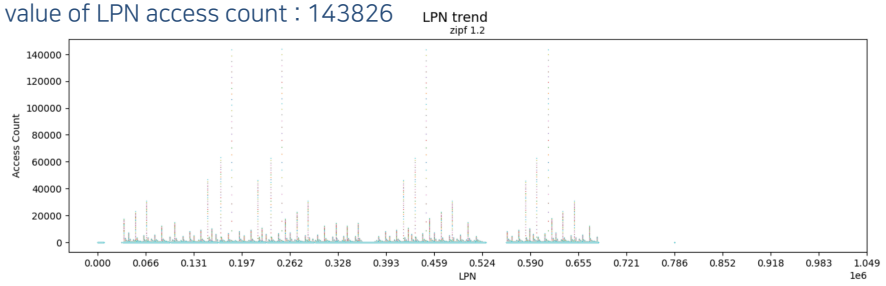




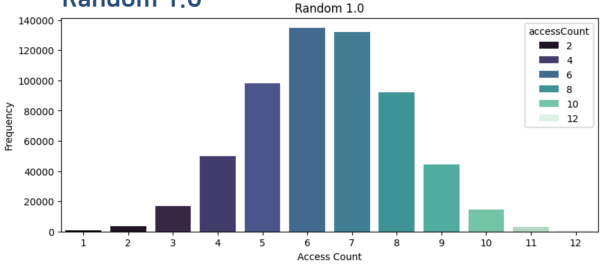
Zipf 1.2

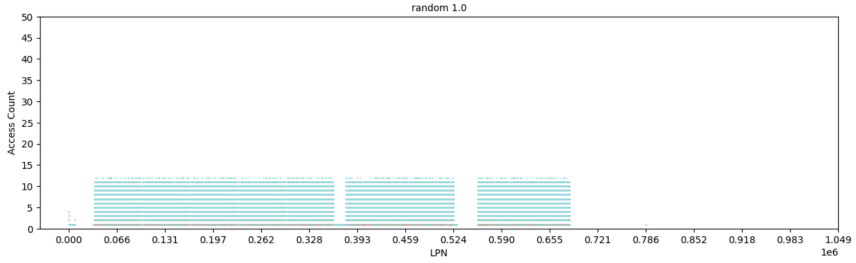
The largest value of LPN access count : 143826





Random 1.0





Implementation

Hot/cold data detection & separation mechanism, IOPS, WAF

ftl.h

```
197  struct ssd {
198      char *ssdname;
199      struct ssdparams sp;
200      struct ssd_channel *ch;
201      struct ppa *maptbl; /* page level mapping table */
202      uint64_t *rmap;     /* reverse mapptbl, assume it's stored in OOB */
203
204      /* -----  HOT, COLD ----- */
205      struct write_pointer hotWp;
206      struct write_pointer coldWp;
207      /* ---------------------- */
208
209      struct line_mgmt lm;
210
211      /* lockless ring for communication with NVMe IO thread */
212      struct rte_ring **to_ftl;
213      struct rte_ring **to_poller;
214      bool *dataplane_started_ptr;
215      QemuThread ftl_thread;
216  };
```

Declaration of 2 write pointers for hot/cold separation

Only two files ftl.h and ftl.c have been modified to add hot/cold detection & separation mechanism, IOPS, and WAF statistics modules.

ftl.c

```
1   #include "ftl.h"
2
3   /* ----- HOT, COLD ----- */
4   #define THRES UINT64_MAX
5   /* ---------------------- */
6
7   /* ----- IOPS ----- */
8   static unsigned int inputOp = 0;
9   static unsigned int outputOp = 0;
10  static FILE *IOPSData;
11  /* ---------------- */
12
13  /* ----- THROUGHPUT ----- */
14  static unsigned int totalRead = 0;
15  static unsigned int totalWrite = 0;
16  static FILE *throughputData;
17  /* ----------------------- */
18
19  /* ----- GC ----- */
20  static unsigned int reclaimedBlocks = 0;
21  static FILE *GCData;
22  /* -------------- */
23
24  /* ----- WAF ----- */
25  static unsigned int hostWrite = 0;
26  static unsigned int GCWrite = 0;
27  static double WAF = 0;
28  static FILE *WAFData;
29  /* -------------- */
30
31  /* ----- ACCESS COUNT ----- */
32  static uint64_t accessCount[1048576] = {0};
33  static char temp[] = "accessCountData";
34  static char name[64];
35  static FILE *accessCountData;
36  /* ------------------------ */
37
38  //#define FEMU_DEBUG_FTL
```

constant for hot/cold separation

THRES value can be changed to another value for hot/cold separation.

Definition of file structure to store data (IOPS)

Definition of file structure to store data (WAF)

```
# Configurable SSD Controller
secsz=512 # sector size in byt
secs_per_pg=8 # number of sect
pgs_per_blk=256 # number of pa
blks_per_pl=256 # number of bl
pls_per_lun=1 # keep it at one
luns_per_ch=8 # number of chip
nchs=2 # number of channels
ssd_size=3072 # in megabytes,
```

4GB (3GB, 25% OP)

256 * 256 * 8 * 2

Temporary name declaration considering data preprocessing

Definition of file structure to store data (Hot/cold data detection)

ssd_init_write_pointer

```
151
152 /* ----- HOT, COLD ----- */
153 static void ssd_init_write_pointer(struct ssd *ssd, char status)
154 {
155     /* ----- HOT, COLD ----- */
156     struct write_pointer *wpp;
157     if (status == 'h') { wpp = &ssd->hotWp; }
158     else { wpp = &ssd->coldWp; }
159     /* --------------------- */
160
161     struct line_mgmt *lm = &ssd->lm;
162     struct line *curline = NULL;
163
164     curline = QTAILQ_FIRST(&lm->free_line_list);
165     QTAILQ_REMOVE(&lm->free_line_list, curline, entry);
166     lm->free_line_cnt--;
167
168     /* wpp->curline is always our next-to-write super-block */
169     wpp->curline = curline;
170     wpp->ch = 0;
171     wpp->lun = 0;
172     wpp->pg = 0;
173
174     /* ----- HOT, COLD ----- */
175     wpp->blk = wpp->curline->id;
176     /* --------------------- */
177
178     wpp->pl = 0;
179 }
180 /* --------------------- */
181
```

Added parameters to distinguish whether it is hot or cold.

Set write pointer according to status

The code before modification was wpp->blk = 0;

Since 2 write pointers have been declared, it has been changed as follows.

ssd_advance_write_pointer

```
202
203 /* ----- HOT, COLD ----- */
204 static void ssd_advance_write_pointer(struct ssd *ssd, char status)
205 {
206     struct ssdparams *spp = &ssd->sp;
207
208     /* ----- HOT, COLD ----- */
209     struct write_pointer *wpp;
210     if (status == 'h') { wpp = &ssd->hotWp; }
211     else { wpp = &ssd->coldWp; }
212     /* --------------------- */
213
214     struct line_mgmt *lm = &ssd->lm;
215
216     check_addr(wpp->ch, spp->nchs);
217     wpp->ch++;
218     if (wpp->ch == spp->nchs) {
219         wpp->ch = 0;
220         check_addr(wpp->lun, spp->luns_per_ch);
221         wpp->lun++;
222         /* in this case, we should go to next lun */
223         if (wpp->lun == spp->luns_per_ch) {
224             wpp->lun = 0;
225             /* go to next page in the block */
226             check_addr(wpp->pg, spp->pgs_per_blk);
227             wpp->pg++;
228             if (wpp->pg == spp->pgs_per_blk) {
229                 wpp->pg = 0;
230                 /* move current line to {victim,full} line list */
231                 if (wpp->curline->vpc == spp->pgs_per_line) {
232                     /* all pgs are still valid, move to full line list */
233                     ftl_assert(wpp->curline->ipc == 0);
234                     QTAILQ_INSERT_TAIL(&lm->full_line_list, wpp->curline, entry);
235                     lm->full_line_cnt++;
236                 } else {
237                     ftl_assert(wpp->curline->vpc >= 0 && wpp->curline->vpc < spp->pgs_per_line);
238                     /* there must be some invalid pages in this line */
239                     ftl_assert(wpp->curline->ipc > 0);
240                     pqueue_insert(lm->victim_line_pq, wpp->curline);
241                     lm->victim_line_cnt++;
242                 }
243                 /* current line is used up, pick another empty line */
244                 check_addr(wpp->blk, spp->blks_per_pl);
245                 wpp->curline = NULL;
246                 wpp->curline = get_next_free_line(ssd);
247                 if (!wpp->curline) {
248                     /* TODO */
249                     abort();
250                 }
251                 wpp->blk = wpp->curline->id;
252                 check_addr(wpp->blk, spp->blks_per_pl);
253                 /* make sure we are starting from page 0 in the super block */
254                 ftl_assert(wpp->pg == 0);
255                 ftl_assert(wpp->lun == 0);
256                 ftl_assert(wpp->ch == 0);
257                 /* TODO: assume # of pl_per_lun is 1, fix later */
258                 ftl_assert(wpp->pl == 0);
259             }
260         }
261     }
262 }
263 /* --------------------- */
```

The same modification as ssd_init_write_pointer.

get_new_page

```
265 /* ----- HOT, COLD ----- */
266 static struct ppa get_new_page(struct ssd *ssd, char status)
267 {
268     /* ----- HOT, COLD ----- */
269     struct write_pointer *wpp;
270     if (status == 'h') { wpp = &ssd->hotWp; }
271     else { wpp = &ssd->coldWp; }
272     /* ---------------------- */ Set write pointer according to status
273
274     struct ppa ppa;
275     ppa.ppa = 0;
276     ppa.g.ch = wpp->ch;
277     ppa.g.lun = wpp->lun;
278     ppa.g.pg = wpp->pg;
279     ppa.g.blk = wpp->blk;
280     ppa.g.pl = wpp->pl;
281     ftl_assert(ppa.g.pl == 0);
282
283     return ppa;
284 }
285 /* ---------------------- */
```

The same modification as ssd_init_write_pointer.

ssd_init

```
424 void ssd_init(FemuCtrl *n)
425 {
426     struct ssd *ssd = n->ssd;
427     struct ssdparams *spp = &ssd->sp;
428
429     ftl_assert(ssd);
430
431     ssd_init_params(spp, n);
432
433     /* initialize ssd internal layout architecture */
434     ssd->ch = g_malloc0(sizeof(struct ssd_channel) * spp->nchs);
435     for (int i = 0; i < spp->nchs; i++) {
436         ssd_init_ch(&ssd->ch[i], spp);
437     }
438
439     /* initialize maptbl */
440     ssd_init_maptbl(ssd);
441
442     /* initialize rmap */
443     ssd_init_rmap(ssd);
444
445     /* initialize all the lines */
446     ssd_init_lines(ssd);
447
448     /* ----- HOT, COLD ----- */
449     /* initialize write pointer, this is how we allocate new pages for writes */
450     ssd_init_write_pointer(ssd, 'h');
451     ssd_init_write_pointer(ssd, 'c'); 2 write pointer initial settings
452     /* ---------------------- */
453
454     qemu_thread_create(&ssd->ftl_thread, "FEMU-FTL-Thread", ftl_thread, n,
455                        QEMU_THREAD_JOINABLE);
456 }
```

2 write pointers are initialized in the ssd_init function.

gc_write_page

```
698  /* move valid page data (already in DRAM) from victim line to a new page */
699  static uint64_t gc_write_page(struct ssd *ssd, struct ppa *old_ppa)
700  {
701      struct ppa new_ppa;
702      struct nand_lun *new_lun;
703      uint64_t lpn = get_rmap_ent(ssd, old_ppa);
704
705      ftl_assert(valid_lpn(ssd, lpn));
706
707      /* ----- HOT, COLD ----- */
708      if (accessCount[lpn] > THRES) { new_ppa = get_new_page(ssd, 'h'); }
709      else { new_ppa = get_new_page(ssd, 'c'); }
710      /* ---------------------- */
711
712      /* update maptbl */
713      set_maptbl_ent(ssd, lpn, &new_ppa);
714      /* update rmap */
715      set_rmap_ent(ssd, lpn, &new_ppa);
716
717      mark_page_valid(ssd, &new_ppa);
718
719      /* ----- HOT, COLD ----- */
720      /* need to advance the write pointer here */
721      if (accessCount[lpn] > THRES) { ssd_advance_write_pointer(ssd, 'h'); }
722      else { ssd_advance_write_pointer(ssd, 'c'); }
723      /* ---------------------- */
724
725      if (ssd->sp.enable_gc_delay) {
726          struct nand_cmd gcw;
727          gcw.type = GC_IO;
728          gcw.cmd = NAND_WRITE;
729          gcw.stime = 0;
730          ssd_advance_status(ssd, &new_ppa, &gcw);
731      }
732
733      /* advance per-ch gc_endtime as well */
734  #if 0
735      new_ch = get_ch(ssd, &new_ppa);
736      new_ch->gc_endtime = new_ch->next_ch_avail_time;
737  #endif
738
739      new_lun = get_lun(ssd, &new_ppa);
740      new_lun->gc_endtime = new_lun->next_lun_avail_time;
741
742      return 0;
743  }
```

Even in the process of moving valid pages, hot and cold separation works.

clean_one_block

```
767  /* here ppa identifies the block we want to clean */
768  static void clean_one_block(struct ssd *ssd, struct ppa *ppa)
769  {
770      struct ssdparams *spp = &ssd->sp;
771      struct nand_page *pg_iter = NULL;
772      int cnt = 0;
773
774      for (int pg = 0; pg < spp->pgs_per_blk; pg++) {
775          ppa->g.pg = pg;
776          pg_iter = get_pg(ssd, ppa);
777          /* there shouldn't be any free page in victim blocks */
778          ftl_assert(pg_iter->status != PG_FREE);
779          if (pg_iter->status == PG_VALID) {
780              gc_read_page(ssd, ppa);
781              /* delay the maptbl update until "write" happens */
782              gc_write_page(ssd, ppa);
783              cnt++;
784
785              /* ----- WAF ----- */
786              GCWrite += 1;
787              /* ---------------- */
788
789          }
790      }
791
792      ftl_assert(get_blk(ssd, ppa)->vpc == cnt);
793  }
```

For WAF calculation, calculate the number of pages written while executing GC.

ssd_write

```
892
893  static uint64_t ssd_write(struct ssd *ssd, NvmeRequest *req)
894  {
895      uint64_t lba = req->slba;
896      struct ssdparams *spp = &ssd->sp;
897      int len = req->nlb;
898      uint64_t start_lpn = lba / spp->secs_per_pg;
899      uint64_t end_lpn = (lba + len - 1) / spp->secs_per_pg;
900      struct ppa ppa;
901      uint64_t lpn;
902      uint64_t curlat = 0, maxlat = 0;
903      int r;
904
905      if (end_lpn >= spp->tt_pgs) {
906          ftl_err("start_lpn=%"PRIu64",tt_pgs=%d\n", start_lpn, ssd->sp.tt_pgs);
907      }
908
909      while (should_gc_high(ssd)) {
910          /* perform GC here until !should_gc(ssd) */
911          r = do_gc(ssd, true);
912          if (r == -1)
913              break;
914      }
915
916      for (lpn = start_lpn; lpn <= end_lpn; lpn++) {
917          ppa = get_maptbl_ent(ssd, lpn);
918          if (mapped_ppa(&ppa)) {
919              /* update old page information first */
920              mark_page_invalid(ssd, &ppa);
921              set_rmap_ent(ssd, INVALID_LPN, &ppa);
922          }
923
924          /* ----- HOT, COLD ----- */
925          /* new write */
926          if (accessCount[lpn] > THRES) { ppa = get_new_page(ssd, 'h'); }
927          else { ppa = get_new_page(ssd, 'c'); }
928          /* ---------------------- */
929
930          /* update maptbl */
931          set_maptbl_ent(ssd, lpn, &ppa);
932          /* update rmap */
933          set_rmap_ent(ssd, lpn, &ppa);
934
935          mark_page_valid(ssd, &ppa);
936
937          /* ----- HOT, COLD ----- */
938          /* need to advance the write pointer here */
939          if (accessCount[lpn] > THRES) { ssd_advance_write_pointer(ssd, 'h'); }
940          else { ssd_advance_write_pointer(ssd, 'c'); }
941          /* ---------------------- */
942
943          struct nand_cmd swr;
944          swr.type = USER_IO;
945          swr.cmd = NAND_WRITE;
946          swr.stime = req->stime;
947          /* get latency statistics */
948          curlat = ssd_advance_status(ssd, &ppa, &swr);
949          maxlat = (curlat > maxlat) ? curlat : maxlat;
950
951          /* ----- THROUGHPUT ----- */
952          totalWrite += spp->secs_per_pg * 512;
953          /* ----------------------- */
954
955          /* ----- WAF ----- */
956          hostWrite += 1;
957          /* ---------------- */
958
959          /* ----- ACCESS COUNT ----- */
960          accessCount[lpn] += 1;
961          /* ------------------------ */
962      }
963
```

```
924          /* ----- HOT, COLD ----- */
925          /* new write */
926          if (accessCount[lpn] > THRES) { ppa = get_new_page(ssd, 'h'); }
927          else { ppa = get_new_page(ssd, 'c'); }
928          /* ---------------------- */
929
930          /* update maptbl */
931          set_maptbl_ent(ssd, lpn, &ppa);
932          /* update rmap */
933          set_rmap_ent(ssd, lpn, &ppa);
934
935          mark_page_valid(ssd, &ppa);
936
937          /* ----- HOT, COLD ----- */
938          /* need to advance the write pointer here */
939          if (accessCount[lpn] > THRES) { ssd_advance_write_pointer(ssd, 'h'); }
940          else { ssd_advance_write_pointer(ssd, 'c'); }
941          /* ---------------------- */
```

When getting a new page, when updating write pointer,
it is divided according to hot/cold status.

```
950
951          /* ----- THROUGHPUT ----- */
952          totalWrite += spp->secs_per_pg * 512;
953          /* ----------------------- */
954
955          /* ----- WAF ----- */
956          hostWrite += 1;
957          /* ---------------- */
958
959          /* ----- ACCESS COUNT ----- */
960          accessCount[lpn] += 1;
961          /* ------------------------ */
962
```

For WAF calculation,
Calculate the number of pages written
by host request.

Save the number of LPN accesses.

*ftl_thread

```
968  static void *ftl_thread(void *arg)
969  {
970
971      /* ----- IOPS, THROUGHPUT ----- */
972      struct timespec startTime;
973      clock_gettime(CLOCK_MONOTONIC, &startTime);
974      /* ---------------------------- */
975
976      /* ----- IOPS ----- */
977      IOPSData = fopen("IOPSData.csv", "w");
978      fprintf(IOPSData, "time,read,write,iops\n");
979      /* ----------------- */
980
981      /* ----- THROUGHPUT ----- */
982      throughputData = fopen("throughputData.csv", "w");
983      fprintf(throughputData, "time,readThroughput,writeThroughput,throughput\n");
984      /* ---------------------- */
985
986      /* ----- GC, WAF ----- */
987      struct timespec startTimeGCWAF;
988      clock_gettime(CLOCK_MONOTONIC, &startTimeGCWAF);
989      /* ------------------- */
990
991      /* ----- GC ----- */
992      GCData = fopen("GCData.csv", "w");
993      fprintf(GCData, "time,reclaimedBlocks\n");
994      /* -------------- */
995
996      /* ----- WAF ----- */
997      WAFData = fopen("WAFData.csv", "w");
998      fprintf(WAFData, "time,WAF\n");
999      /* --------------- */
1000
1001     FemuCtrl *n = (FemuCtrl *)arg;
1002     struct ssd *ssd = n->ssd;
1003     NvmeRequest *req = NULL;
1004     uint64_t lat = 0;
1005     int rc;
1006     int i;
1007
1008     while (!*(ssd->dataplane_started_ptr)) {
1009         usleep(100000);
1010     }
1011
```

In Project 1, IOPS was measured every second, but in Project 2, it was measured every 10 seconds.

Columns setting(IOPS)

In Project 2, IOPS uses this start time.

Set the start time for recording in the csv file (IOPS, WAF)

Columns setting(WAF)

```
1012     /* FIXME: not safe, to handle ->to_ftl and ->to_poller gracefully */
1013     ssd->to_ftl = n->to_ftl;
1014     ssd->to_poller = n->to_poller;
1015
1016     while (1) {
1017         for (i = 1; i <= n->nr_pollers; i++) {
1018             if (!ssd->to_ftl[i] || !femu_ring_count(ssd->to_ftl[i]))
1019                 continue;
1020
1021             rc = femu_ring_dequeue(ssd->to_ftl[i], (void *)&req, 1);
1022             if (rc != 1) {
1023                 printf("FEMU: FTL to_ftl dequeue failed\n");
1024             }
1025
1026             ftl_assert(req);
1027             switch (req->cmd.opcode) {
1028             case NVME_CMD_WRITE:
1029                 lat = ssd_write(ssd, req);
1030
1031                 /* ----- IOPS ----- */
1032                 outputOp++;
1033                 /* ----------------- */
1034
1035                 break;
1036             case NVME_CMD_READ:
1037                 lat = ssd_read(ssd, req);
1038
1039                 /* ----- IOPS ----- */
1040                 inputOp++;
1041                 /* ----------------- */
1042
1043                 break;
1044             case NVME_CMD_DSM:
1045                 lat = 0;
1046                 break;
1047             default:
1048                 //ftl_err("FTL received unkown request type, ERROR\n");
1049                 ;
1050             }
```

When it's ssd write command, outputOp increases

When it's ssd read command, inputOp increases

The statistics module is managed in *ftl_thread.

*ftl_thread

```
1051
1052            req->reqlat = lat;
1053            req->expire_time += lat;
1054
1055            rc = femu_ring_enqueue(ssd->to_poller[i], (void *)&req, 1);
1056            if (rc != 1) {
1057                ftl_err("FTL to_poller enqueue failed\n");
1058            }
1059
1060            /* ----- THROUGHPUT ----- */
1061            struct timespec currentTime;
1062            clock_gettime(CLOCK_MONOTONIC, &currentTime);
1063            char *timeStamp = ctime(&(time_t){time(NULL)});
1064            /* ---------------------------- */
1065
1066            /* ----- IOPS, GC, WAF ----- */
1067            struct timespec currentTimeGCWAF;
1068            clock_gettime(CLOCK_MONOTONIC, &currentTimeGCWAF);
1069            char *timeStampGCWAF = ctime(&(time_t){time(NULL)});
1070            /* ------------------- */
1071
1072            if (currentTime.tv_sec - startTime.tv_sec >= 1) {
1073
1074                /* ----- THROUGHPUT ----- */
1075                double readThroughput = totalRead / (1024.0 * 1024.0);
1076                double writeThroughput = totalWrite / (1024.0 * 1024.0);
1077                fprintf(
1078                    throughputData, "%.15s,%.5f,%.5f,%.5f\n", timeStamp + 4,
1079                    readThroughput, writeThroughput, readThroughput + writeThroughput
1080                );
1081                fflush(throughputData);
1082                totalRead = 0;
1083                totalWrite = 0;
1084                /* ---------------------- */
1085
1086                startTime = currentTime;
1087            }
```

timeStamp plays a role in printing time on a file or file name.

```
1088
1089            if (currentTimeGCWAF.tv_sec - startTimeGCWAF.tv_sec >= 10) {
1090
1091                /* ----- IOPS ----- */
1092                fprintf(
1093                    IOPSData, "%.15s,%u,%u,%u\n", timeStampGCWAF + 4,
1094                    inputOp, outputOp, inputOp + outputOp
1095                );
1096                fflush(IOPSData);
1097                inputOp = 0;
1098                outputOp = 0;
1099                /* ---------------- */
1100
1101                /* ----- GC ----- */
1102                fprintf(GCData, "%.15s,%u\n", timeStampGCWAF + 4, reclaimedBlocks);
1103                fflush(GCData);
1104                reclaimedBlocks = 0;
1105                /* -------------- */
1106
1107                /* ----- WAF ----- */
1108                WAF = (hostWrite > 0) ? (double)(hostWrite + GCWrite) / hostWrite : 0;
1109                fprintf(WAFData, "%.15s,%.5f\n", timeStampGCWAF + 4, WAF);
1110                fflush(WAFData);
1111                hostWrite = 0;
1112                GCWrite = 0;
1113                /* --------------- */
1114
1115                /* ----- ACCESS COUNT ----- */
1116                snprintf(name, sizeof(name), "%s_%.8s.csv", temp, timeStampGCWAF + 11);
1117                accessCountData = fopen(name, "w");
1118                fprintf(accessCountData, "lpn,accessCount\n");
1119                for (unsigned int init = 0; init < 1048576; init++) {
1120                    fprintf(accessCountData, "%u" ",%" PRIi64 "\n",init,accessCount[init]);
1121                }
1122                fflush(accessCountData);
1123                fclose(accessCountData);
1124                /* ------------------------ */
1125
1126                startTimeGCWAF = currentTimeGCWAF;
1127            }
1128
1129
1130            /* clean one line if needed (in the background) */
1131            if (should_gc(ssd)) {
1132                do_gc(ssd, false);
1133            }
1134        }
1135    }
1136
1137    /* ----- IOPS, THROUGHPUT, GC, WAF ----- */
1138    fclose(IOPSData);
1139    fclose(throughputData);
1140    fclose(GCData);
1141    fclose(WAFData);
1142    /* ------------------------------------ */
1143
1144    return NULL;
1145 }
1146
```

IOPS, WAF, hot/cold detection mechanism files are recorded every 10 seconds.

IOPS calculation

WAF calculation

Record the time in the file name

Record the access count for the entire LPN

Time update

The entire LPN is recorded, so the file size is large.

Therefore, it is set differently from the recording method of other indicators.

Visualization and data preprocessing

Get statistical results and draw graphs

I'll show you normal 1.0 as an example.
Other workloads go through a similar process.
(Other workloads are also attached to the last slide.)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing modules necessary for visualization

I/O Distribution

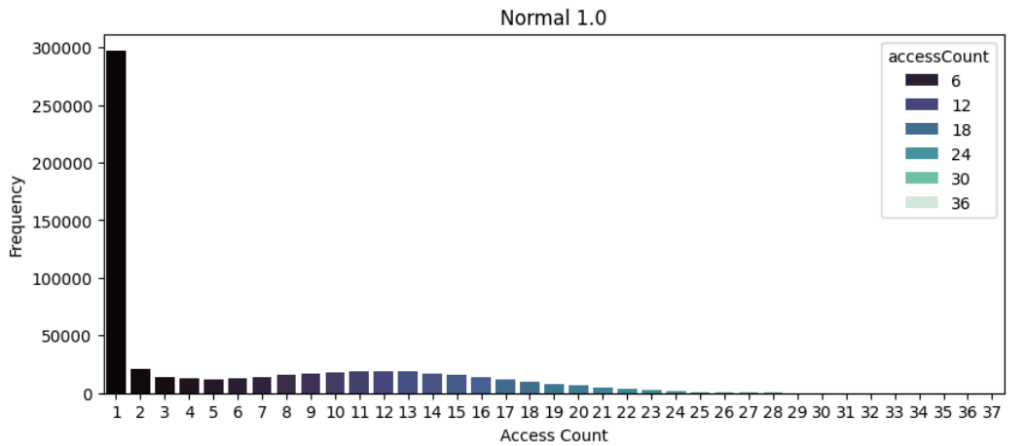
In [2]:
```
beforeBench = pd.read_csv('./max/BB.csv')
afterBench = pd.read_csv('./max/AB.csv')

preBench = pd.merge(afterBench, beforeBench, on='lpn', suffixes=('After', 'Before'))
preBench['accessCount'] = preBench['accessCountAfter'] - preBench['accessCountBefore']
preBench = preBench.loc[preBench['accessCount'] > 0]
preBench = preBench.drop(columns=['accessCountAfter', 'accessCountBefore'])
preBench.to_csv('./max/AC.csv', index=False)

bench = pd.read_csv('./max/AC.csv')
```

For precise measurement,
Remove the value while being mounted.

Extract only values that exceed 0

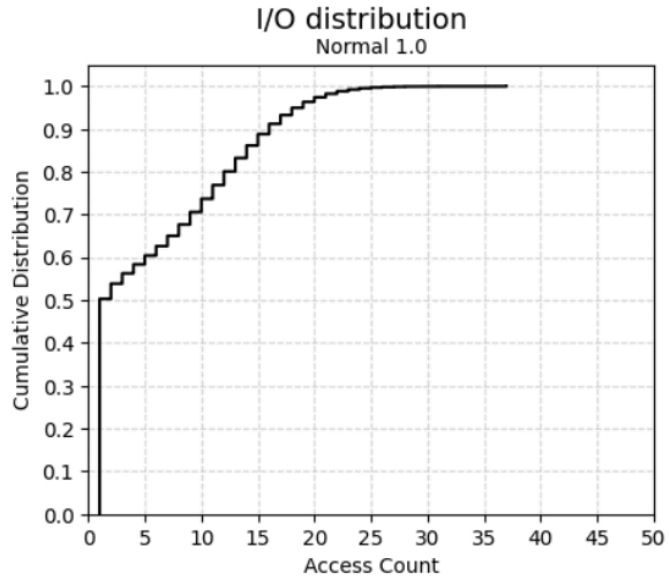
In [3]:
```
plt.figure(figsize=(10, 4))

plt.title('Normal 1.0')
plt.xlabel('Access Count')
plt.ylabel('Frequency')

sns.countplot(x='accessCount', data=bench, hue='accessCount', palette='mako')
plt.show()
```

Draw countplot



In [4]:
```
plt.figure(figsize=(5, 4))

plt.suptitle('I/O distribution', fontsize=14)
plt.title('Normal 1.0', fontsize=10)
plt.xlabel('Access Count')
plt.ylabel('Cumulative Distribution')

plt.xlim(min(bench['accessCount']), 50)
plt.ylim(0.0, 1.05)
plt.xticks(range(min(bench['accessCount'] - 1), 55, 5))
plt.yticks(np.arange(0, 1.1, 0.1))
plt.grid(True, linestyle='--', alpha=0.5)

sns.ecdfplot(bench['accessCount'], color ='black')
plt.savefig("AC.png")

plt.show()
```

Draw CDF graph



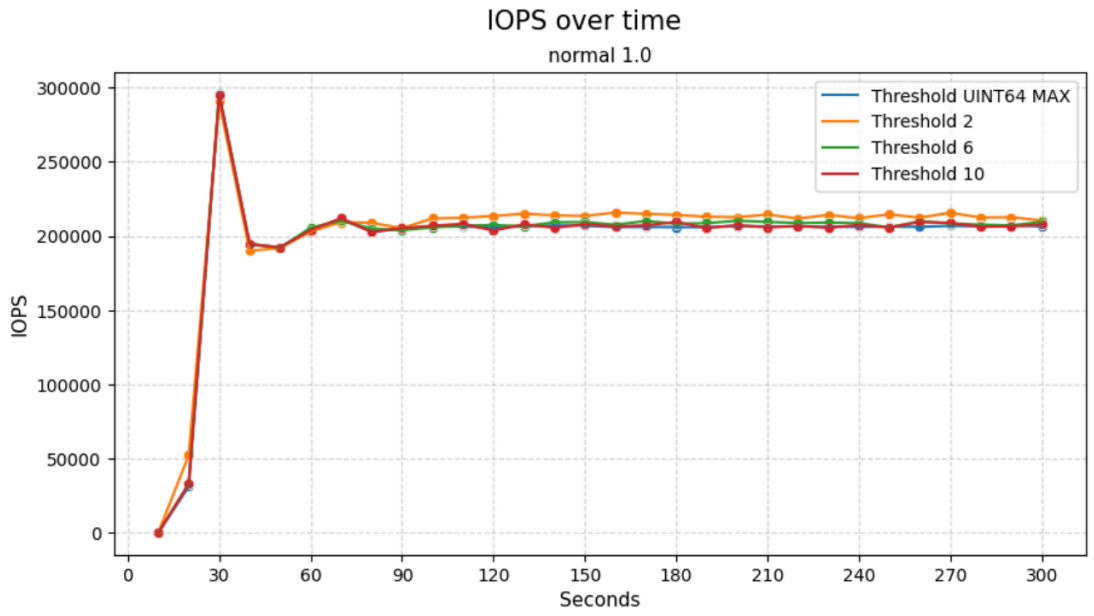
IOPS

In [5]:
```
IOPSMAX = pd.read_csv('./max/IOPS.csv')
IOPSMAX['time'] = range(1, len(IOPSMAX) + 1, 1)
IOPSMAX['time'] = 10 * IOPSMAX['time']
```
Set the time interval to 10 seconds

```
IOPS2 = pd.read_csv('./2/IOPS.csv')
IOPS2['time'] = range(1, len(IOPS2) + 1, 1)
IOPS2['time'] = 10 * IOPS2['time']

IOPS6 = pd.read_csv('./6/IOPS.csv')
IOPS6['time'] = range(1, len(IOPS6) + 1, 1)
IOPS6['time'] = 10 * IOPS6['time']

IOPS10 = pd.read_csv('./10/IOPS.csv')
IOPS10['time'] = range(1, len(IOPS10) + 1, 1)
IOPS10['time'] = 10 * IOPS10['time']

plt.figure(figsize=(10, 5))
plt.suptitle('IOPS over time', fontsize=15)
plt.title('normal 1.0', fontsize=11)
plt.xlabel('Seconds', fontsize=11)
plt.ylabel('IOPS', fontsize=11)

plt.xticks(np.arange(0, 350, 30))
plt.grid(True, linestyle='--', alpha=0.5)

sns.lineplot(x='time', y='iops', data=IOPSMAX, label='Threshold UINT64 MAX')
sns.scatterplot(x="time", y="iops", data=IOPSMAX)
```
Dot every 10 seconds

```
sns.lineplot(x='time', y='iops', data=IOPS2, label='Threshold 2')
sns.scatterplot(x="time", y="iops", data=IOPS2)

sns.lineplot(x='time', y='iops', data=IOPS6, label='Threshold 6')
sns.scatterplot(x="time", y="iops", data=IOPS6)

sns.lineplot(x='time', y='iops', data=IOPS10, label='Threshold 10')
sns.scatterplot(x="time", y="iops", data=IOPS10)

plt.savefig("IOPS.png")
plt.legend()
plt.show()
```
Find IOPS for each workload

```
print(f"Threshold UINT64 MAX : {IOPSMAX['iops'].mean():.5f}")
print(f"Threshold 2 : {IOPS2['iops'].mean():.5f}")
print(f"Threshold 6 : {IOPS6['iops'].mean():.5f}")
print(f"Threshold 10 : {IOPS10['iops'].mean():.5f}")
```




```
Threshold UINT64 MAX : 195758.83333
Threshold 2 : 201124.36667
Threshold 6 : 197164.43333
Threshold 10 : 196187.70000
```

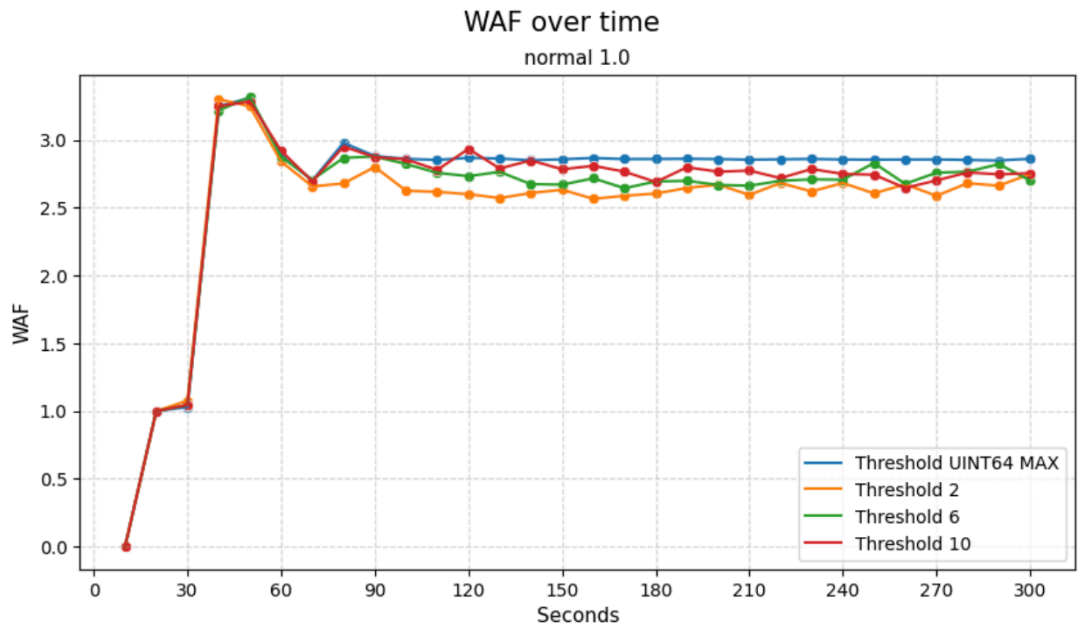
Draw a graph with the data after the data preprocessing is completed.

WAF

In [6]:
```
WAFMAX = pd.read_csv('./max/WAF.csv')
WAFMAX['time'] = range(1, len(WAFMAX) + 1, 1)
WAFMAX['time'] = 10 * WAFMAX['time']  Set the time interval to 10 seconds

WAF2 = pd.read_csv('./2/WAF.csv')
WAF2['time'] = range(1, len(WAF2) + 1, 1)
WAF2['time'] = 10 * WAF2['time']

WAF6 = pd.read_csv('./6/WAF.csv')
WAF6['time'] = range(1, len(WAF6) + 1, 1)
WAF6['time'] = 10 * WAF6['time']

WAF10 = pd.read_csv('./10/WAF.csv')
WAF10['time'] = range(1, len(WAF10) + 1, 1)
WAF10['time'] = 10 * WAF10['time']

plt.figure(figsize=(10, 5))
plt.suptitle('WAF over time', fontsize=15)
plt.title('normal 1.0', fontsize=11)
plt.xlabel('Seconds', fontsize=11)
plt.ylabel('WAF', fontsize=11)

plt.xticks(np.arange(0, 350, 30))
plt.grid(True, linestyle='--', alpha=0.5)

sns.lineplot(x='time', y='WAF', data=WAFMAX, label='Threshold UINT64 MAX')
sns.scatterplot(x="time", y="WAF", data=WAFMAX)  Dot every 10 seconds

sns.lineplot(x='time', y='WAF', data=WAF2, label='Threshold 2')
sns.scatterplot(x="time", y="WAF", data=WAF2)

sns.lineplot(x='time', y='WAF', data=WAF6, label='Threshold 6')
sns.scatterplot(x="time", y="WAF", data=WAF6)

sns.lineplot(x='time', y='WAF', data=WAF10, label='Threshold 10')
sns.scatterplot(x="time", y="WAF", data=WAF10)

plt.savefig("WAF.png")
plt.legend()
plt.show()
```

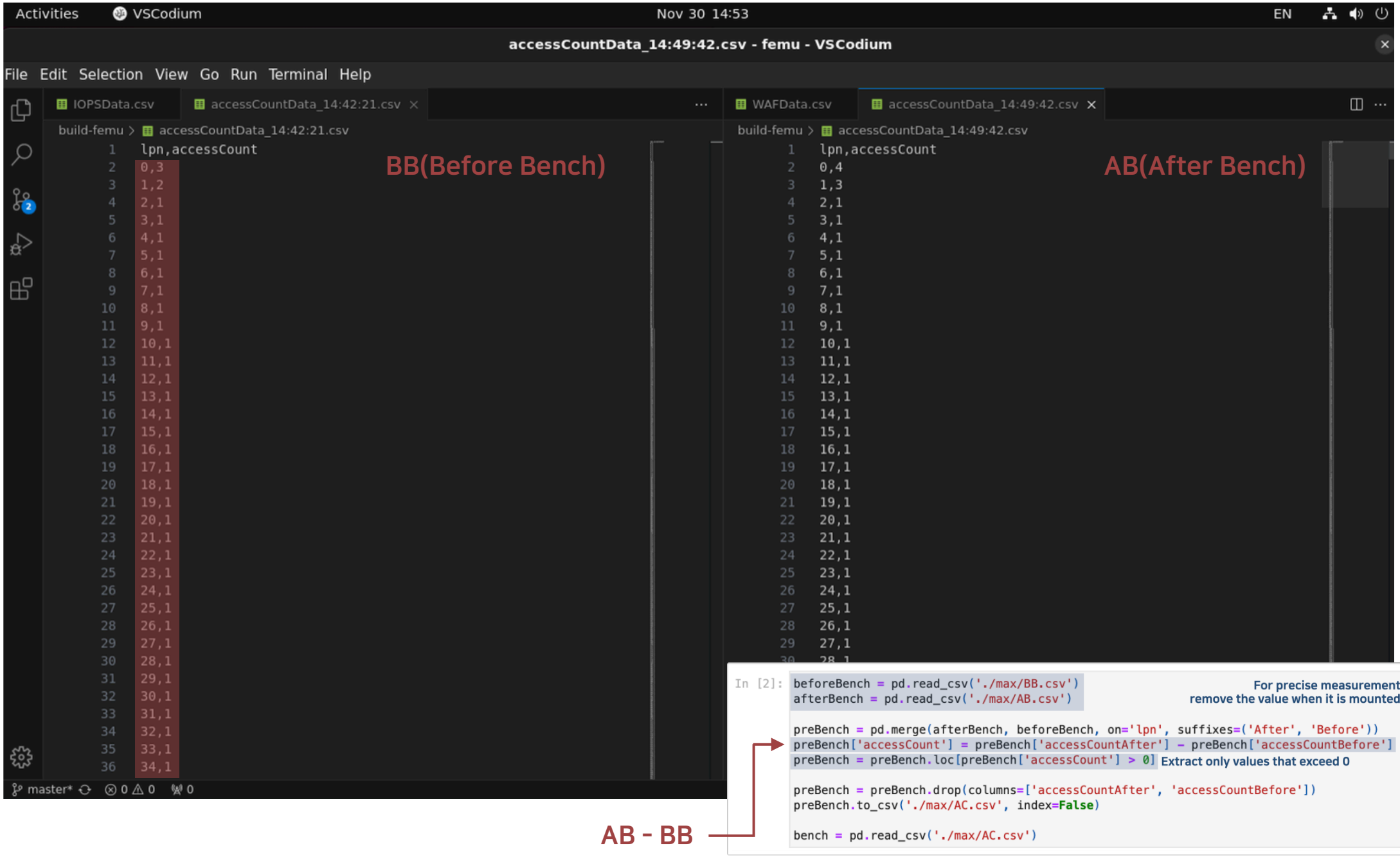
Find WAF for each workload

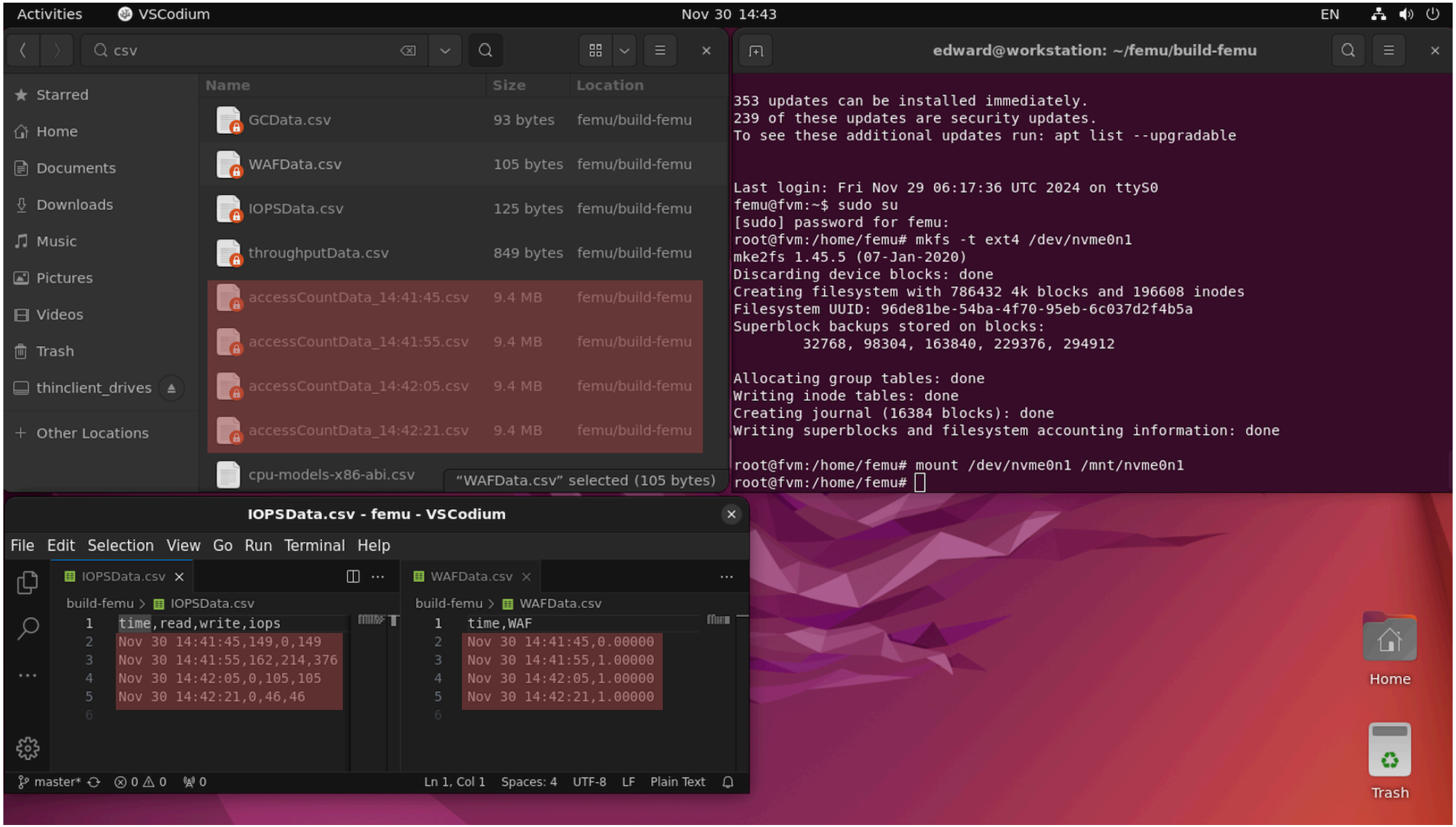
```
print(f"Threshold UINT64 MAX : {WAFMAX['WAF'].mean():.5f}")
print(f"Threshold 2 : {WAF2['WAF'].mean():.5f}")
print(f"Threshold 6 : {WAF6['WAF'].mean():.5f}")
print(f"Threshold 10 : {WAF10['WAF'].mean():.5f}")
```




```
Threshold UINT64 MAX : 2.66846
Threshold 2 : 2.49603
Threshold 6 : 2.56964
Threshold 10 : 2.60756
```

Draw a graph in a way similar to IOPS over time.


Activities
VSCodium
Nov 30 14:53
EN
accessCountData_14:49:42.csv - femu - VSCodium
File Edit Selection View Go Run Terminal Help
IOPSData.csv
accessCountData_14:42:21.csv
build-femu > accessCountData_14:42:21.csv
lpn,accessCount
0,3
1,2
2,1
3,1
4,1
5,1
6,1
7,1
8,1
9,1
10,1
11,1
12,1
13,1
14,1
15,1
16,1
17,1
18,1
19,1
20,1
21,1
22,1
23,1
24,1
25,1
26,1
27,1
28,1
29,1
30,1
31,1
32,1
33,1
34,1
BB(Before Bench)
WAFData.csv
accessCountData_14:49:42.csv
build-femu > accessCountData_14:49:42.csv
lpn,accessCount
0,4
1,3
2,1
3,1
4,1
5,1
6,1
7,1
8,1
9,1
10,1
11,1
12,1
13,1
14,1
15,1
16,1
17,1
18,1
19,1
20,1
21,1
22,1
23,1
24,1
25,1
26,1
27,1
AB(After Bench)
master*
In [2]:
beforeBench = pd.read_csv('./max/BB.csv')
afterBench = pd.read_csv('./max/AB.csv')
For precise measurement, remove the value when it is mounted.
preBench = pd.merge(afterBench, beforeBench, on='lpn', suffixes=('After', 'Before'))
preBench['accessCount'] = preBench['accessCountAfter'] - preBench['accessCountBefore']
preBench = preBench.loc[preBench['accessCount'] > 0]
Extract only values that exceed 0
preBench = preBench.drop(columns=['accessCountAfter', 'accessCountBefore'])
preBench.to_csv('./max/AC.csv', index=False)
bench = pd.read_csv('./max/AC.csv')
AB - BB



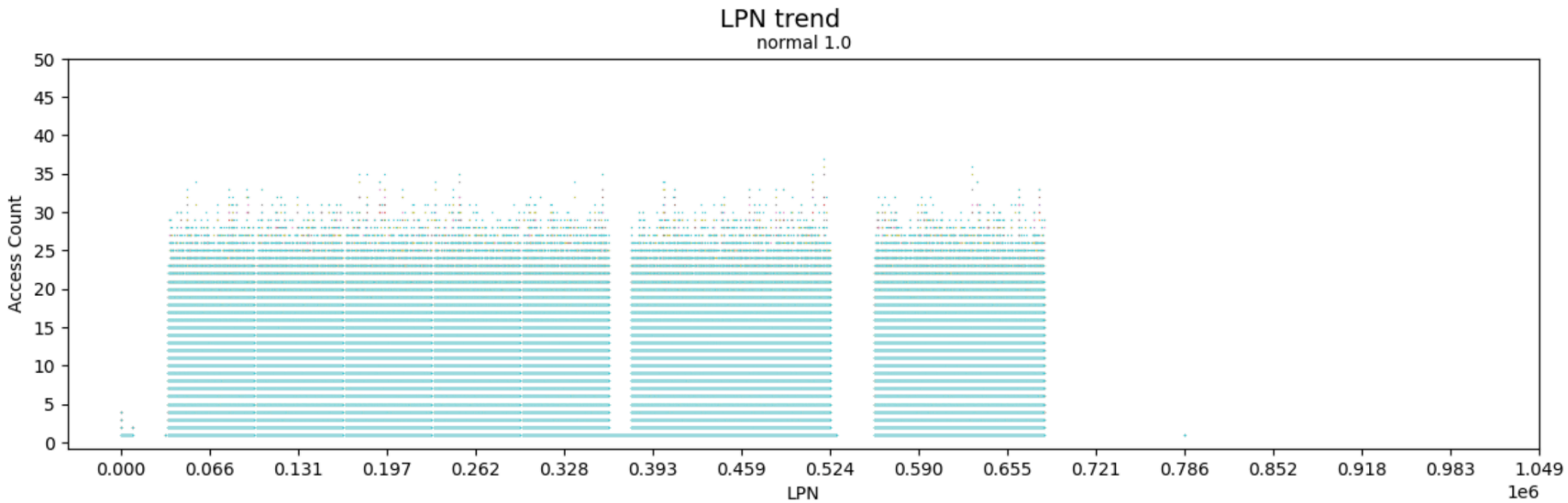
Since the statistics generated in the mount process are not related to the workload,
These are ignored or erased and not reflected.

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

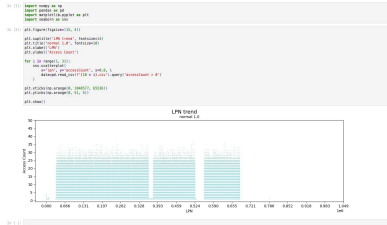
[2]:
```
plt.figure(figsize=(15, 4))

plt.suptitle('LPN trend', fontsize=14)
plt.title('normal 1.0', fontsize=10)
plt.xlabel('LPN')
plt.ylabel('Access Count')

for i in range(1, 31):
    sns.scatterplot(
        x='lpn', y='accessCount', s=0.8, \
        data=pd.read_csv(f'{10 * i}.csv').query('accessCount > 0')
    )

plt.xticks(np.arange(0, 1048577, 65536))
plt.yticks(np.arange(0, 51, 5))

plt.show()
```

Record the access count of LPN recorded every 10 seconds

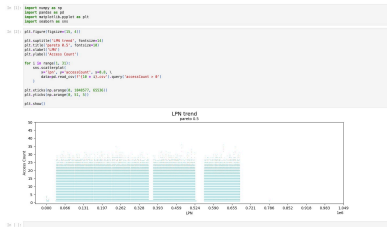
Other workloads also wrote code in a similar way.
(Codes for other workloads are also attached next to it.)



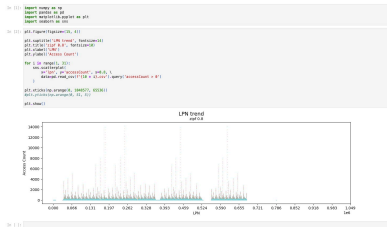
Normal 1.0



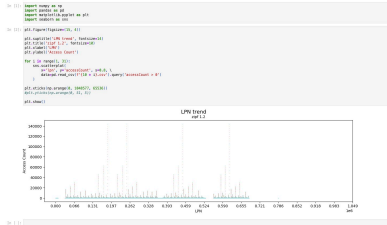
Pareto 0.5



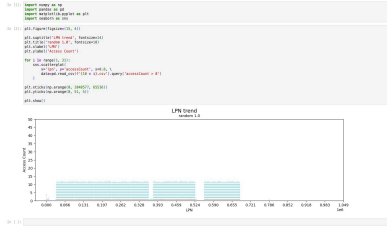
Zipf 0.8



Zipf 1.2

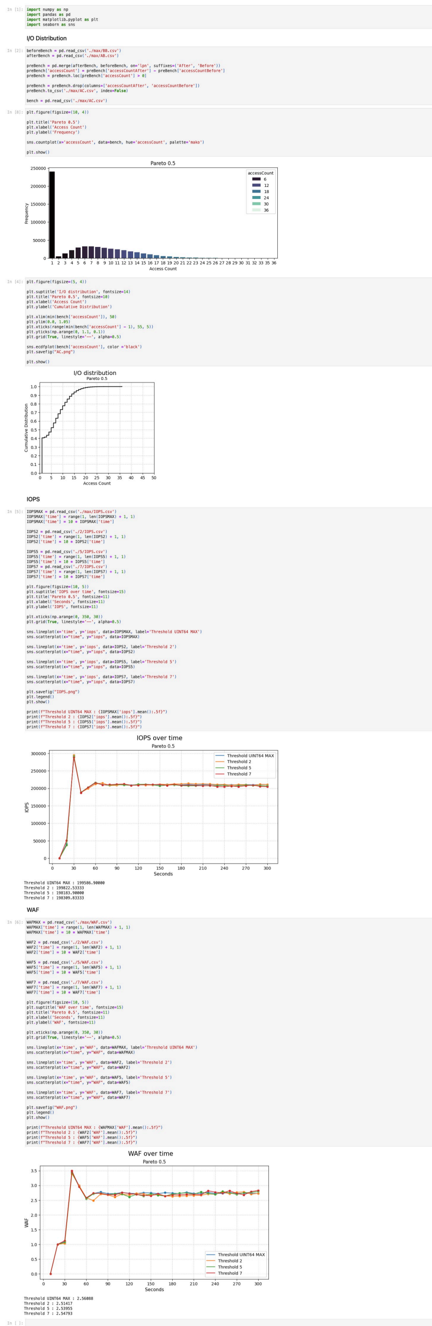


Random 1.0

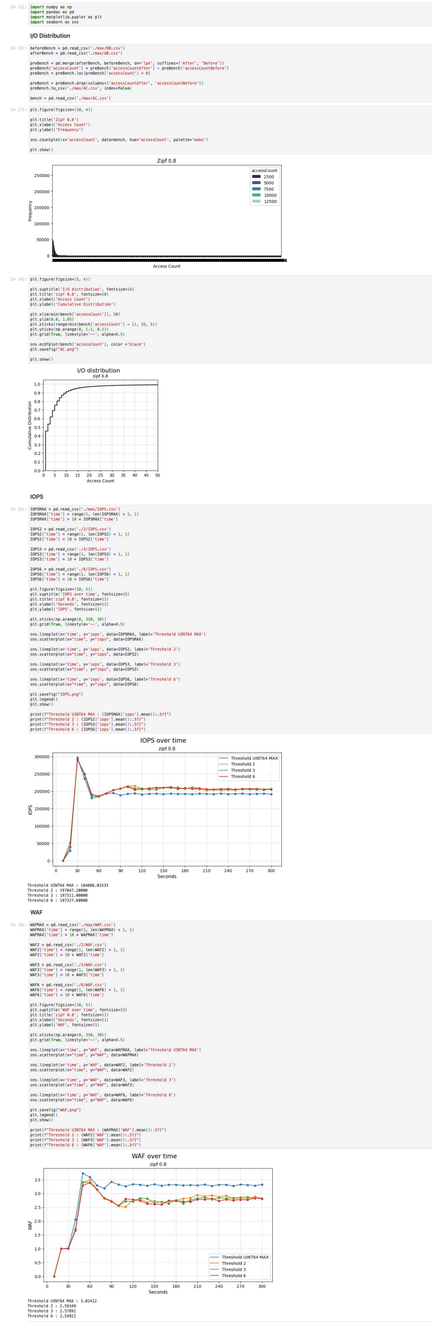


Normal 1.0

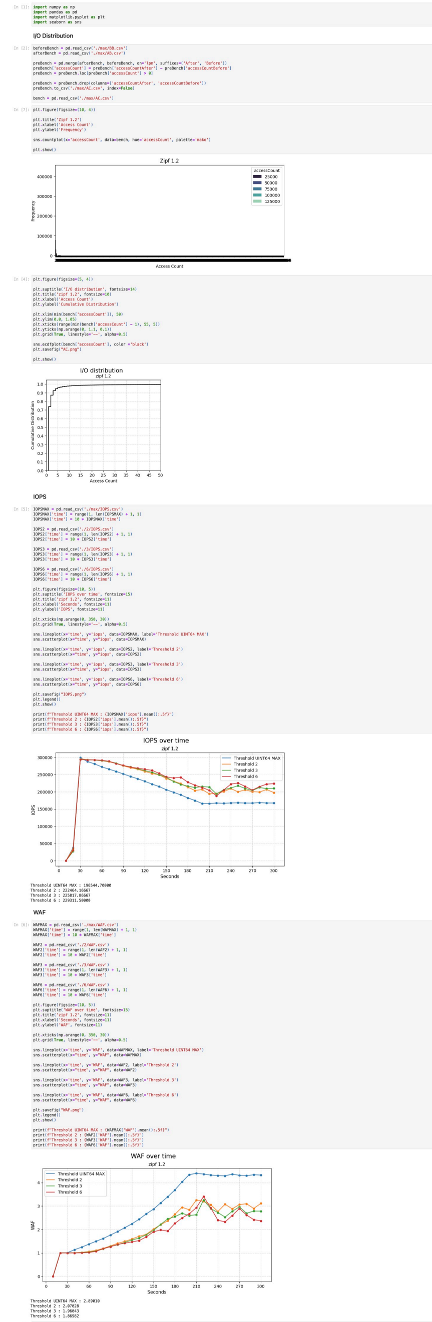
Pareto 0.5



Zipf 0.8



Zipf 1.2



Random 1.0