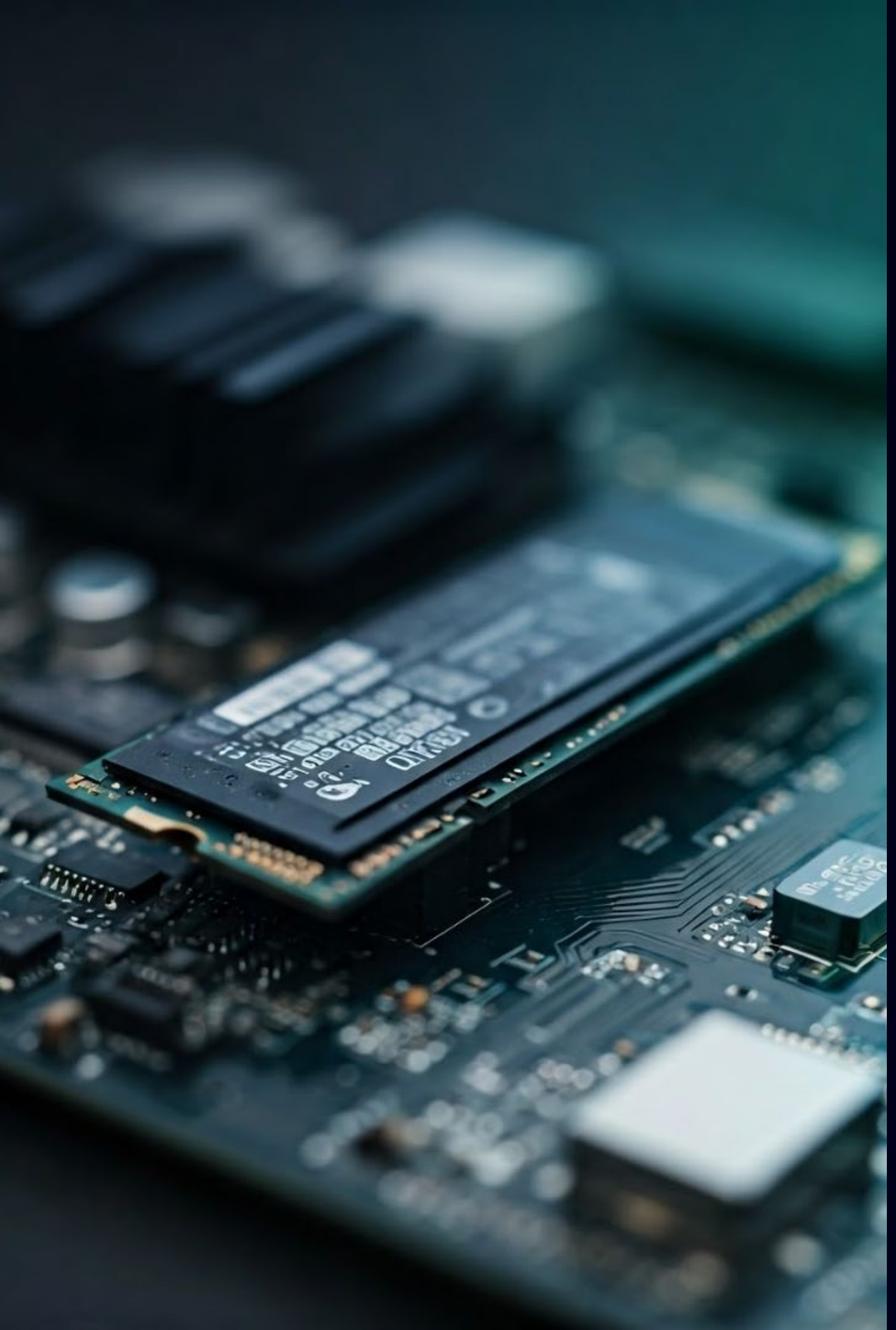
Improving GC Efficiency through Hot/Cold Separation

CSE4009 System Programming

성주원 | 컴퓨터학부 3학년 | 한양대(ERICA)



Implementing Hot/Cold Separation In FEMU

Explanation of Source code

I will explain how I modified the source code to implement the h ot and cold separation

Tests configuration

Before showing the analysis of tests, I will show which kinds of tests were performed

Analysis of Each tests

We will investigate IOPS, WAFS, and CDF graph of each tests after.

Implementation of Hot/Cold Separation

1

Initialize Hot/Cold Pointers

We will allocate hot and cold pointers separately in ssd_init_write_pointer function

2

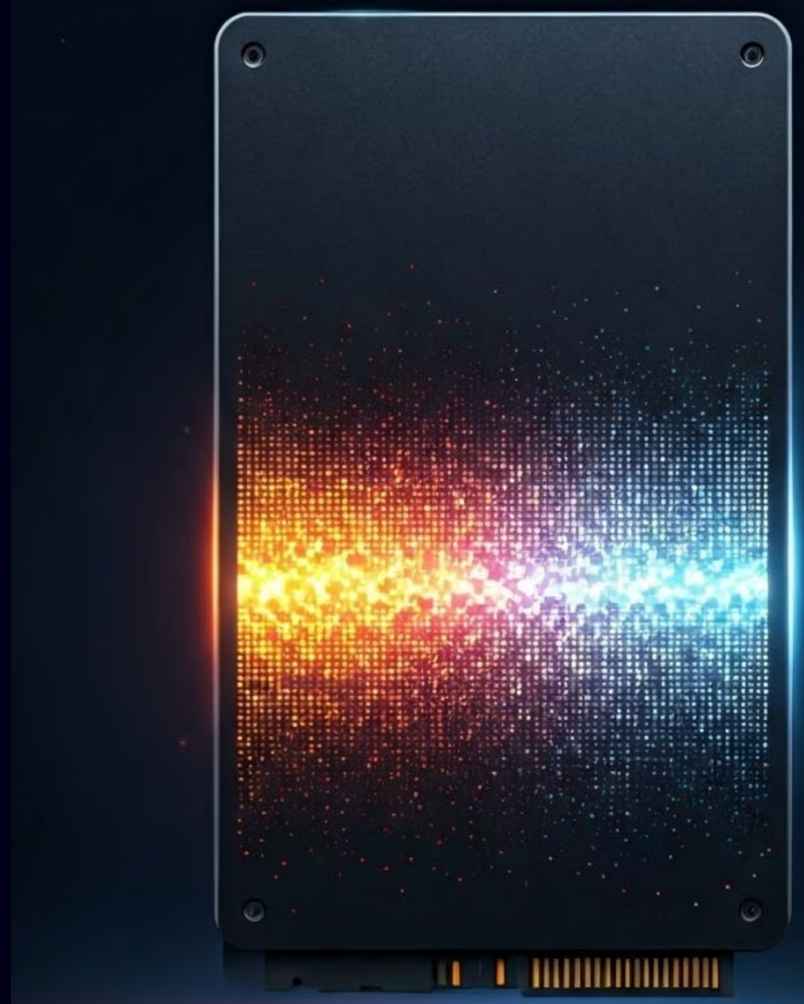
Record LPN and get new page

When the write I/O comes, the ssd_write function triggers and we will record LPN access counts and get new page for pointer depending on the number of LPN access counts.

3

Advance the write pointer

After we record our writes to pointer, we need to advance our write pointers for our future write operation.



Initialize Hot/Cold Pointers

Part of ssd_init function

```
/* initialize write pointer, this is how we allocate new pages for writes */
ssd->lba_write_counts=g_malloc0(sizeof(uint64_t) * spp->tt_pgs);
ssd_init_write_pointer(ssd);
```

```
static void ssd_init_write_pointer(struct ssd *ssd)
{
    /*
    struct write_pointer *wpp = &ssd->wp;
    */
    struct line_mgmt *lm = &ssd->lm;

    struct line *hotline = NULL;
    struct line *coldline = NULL;

    //allocate hot line
    hotline = QTAILQ_FIRST(&lm->free_line_list);
    QTAILQ_REMOVE(&lm->free_line_list, hotline, entry);
    lm->free_line_cnt--;

    //allocate cold line
    coldline = QTAILQ_FIRST(&lm->free_line_list);
    QTAILQ_REMOVE(&lm->free_line_list, coldline, entry);
    lm->free_line_cnt--;

    ssd->was_hot_data=false;
    //configure cold line
    struct write_pointer *cold_wp = &ssd->cold_wp;
    cold_wp->curline = coldline;
    cold_wp->ch = 0;
    cold_wp->lun = 0;
    cold_wp->pg = 0;
    cold_wp->blk = coldline->id;
    cold_wp->pl = 0;

    //configure hot line
    struct write_pointer *hot_wp = &ssd->hot_wp;
    hot_wp->curline = hotline;
    hot_wp->ch = 0;
    hot_wp->lun = 0;
    hot_wp->pg = 0;
    hot_wp->blk = hotline->id;
    hot_wp->pl = 0;
}
```

1

Initialize write pointer by calling ssd_init_write_pointer function

When mounting ssd, the ssd_init function is called, and to initializ e the write pointer, the function calls ssd_init_write_pointer

We will allocate the array into the size of # of pages.

2

Allocate the free line to write pointer

We’ve created the hot and cold line pointer. So after, we need to allocate the free line to use them.

Since the first operations will all use cold data, we will set was_hot_data flag into false. (Will be discussed later)

3

Configure write pointers

We need to update the write pointers. Since it is initialization part, the channel, lun, page, and plane should be 0. But the current line it points should be allocated free line and therefore, the block id s hould also be the id of allocated free line.

Record LPN and get new page

Part of ssd_write function

```
    //increase the write counts of lpn
    ssd->lba_write_counts[lpn]++;

    /* new write(giving our current ssd and lpn) */
    ppa = get_new_page(ssd,lpn);
```

```
static struct ppa get_new_page(struct ssd *ssd, int lpn)
{
    //if the access count is larger than the threshold, allocate hot line
    if (ssd->lba_write_counts[lpn]>=hot_cold_threshold){
        struct write_pointer *hot_wpp = &ssd->hot_wp;
        struct ppa ppa;
        ppa.ppa = 0;
        ppa.g.ch = hot_wpp->ch;
        ppa.g.lun = hot_wpp->lun;
        ppa.g.pg = hot_wpp->pg;
        ppa.g.blk = hot_wpp->blk;
        ppa.g.pl = hot_wpp->pl;
        ftl_assert(ppa.g.pl == 0);
        ssd->was_hot_data=true;
        return ppa;
    }
    //Else, we will allocate the cold line
    else{
        struct write_pointer *cold_wpp = &ssd->cold_wp;
        struct ppa ppa;
        ppa.ppa = 0;
        ppa.g.ch = cold_wpp->ch;
        ppa.g.lun = cold_wpp->lun;
        ppa.g.pg = cold_wpp->pg;
        ppa.g.blk = cold_wpp->blk;
        ppa.g.pl = cold_wpp->pl;
        ftl_assert(ppa.g.pl == 0);
        ssd->was_hot_data=false;
        return ppa;
    }

}
```

1

Increase the access counts of LPN and call the get_new_page function

We need to increase the access count of LPN we access to decide whether we need to allocate the LPN to cold or hot line. After, we will call get_new_page function to get hot or cold line.

2

Return write pointer based on the LPN access counts

get_new_page function will receive the ssd and LPN to return page for the LPN. We will check the number of writes of LPN by us ing lba_write_counts array we’ve made and updated. If the number of write counts is larger, we will return hot pointer. If not, we will r eturn cold pointer. Then, we update the was_hot_data flag for ssd_advance_write_pointer function

Advance the write pointer

Part of ssd_write function

```
        /* need to advance the write pointer here */
        ssd_advance_write_pointer(ssd);
```

```
static void ssd_advance_write_pointer(struct ssd *ssd)
{
    struct ssdparams *spp = &ssd->sp;
    struct write_pointer *wpp;
    struct line_mgmt *lm = &ssd->lm;

    //Change target write pointer depending on what was used
    if(ssd->was_hot_data==true){
        wpp=&ssd->hot_wp;
    }
    else{
        wpp=&ssd->cold_wp;
    }

    check_addr(wpp->ch, spp->nchs);
    wpp->ch++;
    if (wpp->ch == spp->nchs)
    {
        wpp->ch = 0;
        check_addr(wpp->lun, spp->luns_per_ch);
        wpp->lun++;
        /* in this case, we should go to next lun */
        if (wpp->lun == spp->luns_per_ch)
        {
            wpp->lun = 0;
            /* go to next page in the block */
            check_addr(wpp->pg, spp->pgs_per_blk);
            wpp->pg++;
```

1

Call the ssd_advance_write_pointer for future write

After we update the mapping table, we need to advance our write pointer for next write operation. In order to do so, we call ssd_advance_write_pointer function

2

Advance the hot or cold write pointer

Depending on the was_hot_data flag, we will advance our pointer. The rest of the function remains the same.

Analyzing Test Workloads

Normal distribution 1.0 Workload

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

Zipfian distribution 0.8

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

Uniform distribution

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

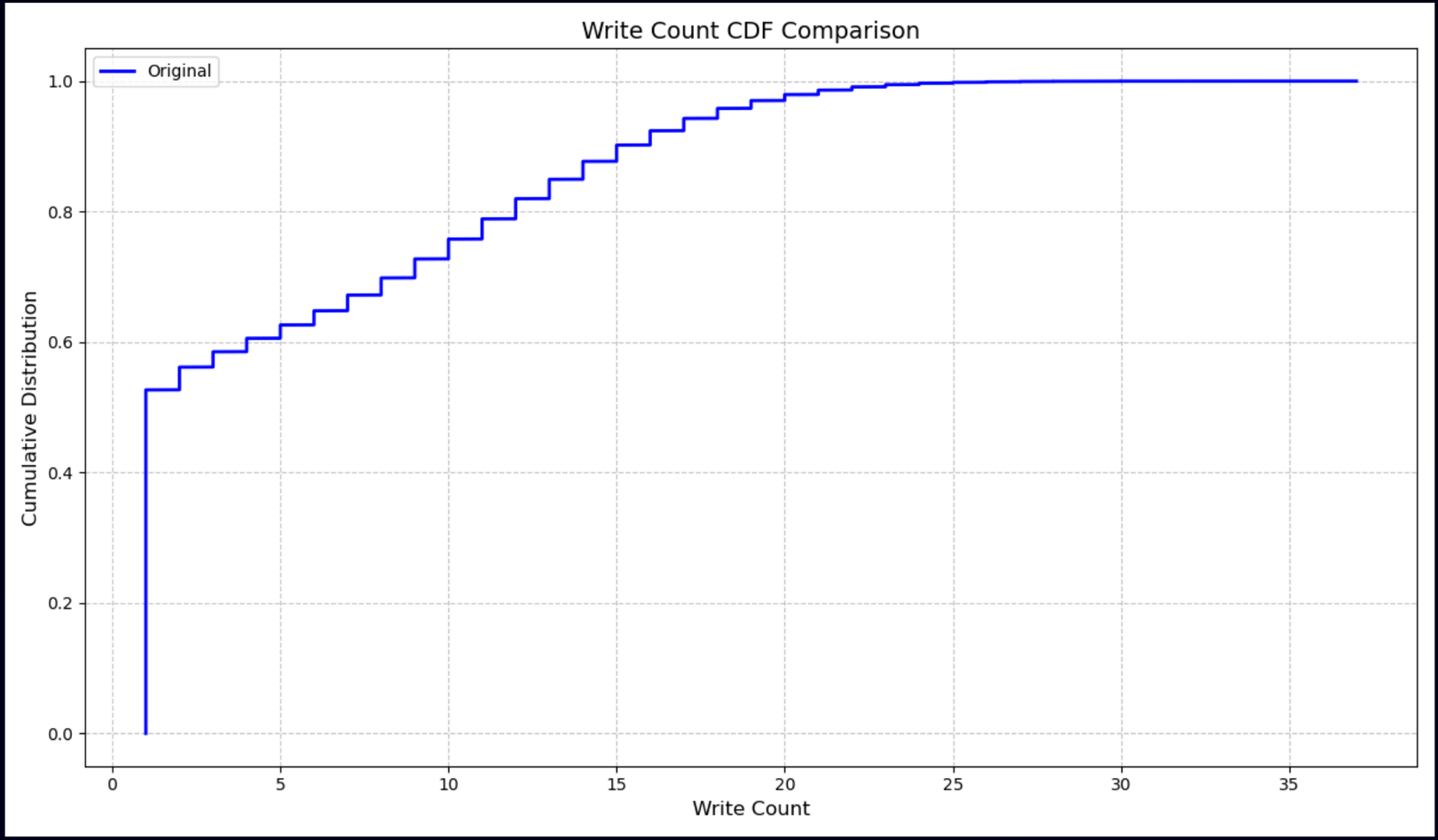
Pareto distribution 0.5 Workload

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

Zipfian distribution 1.2

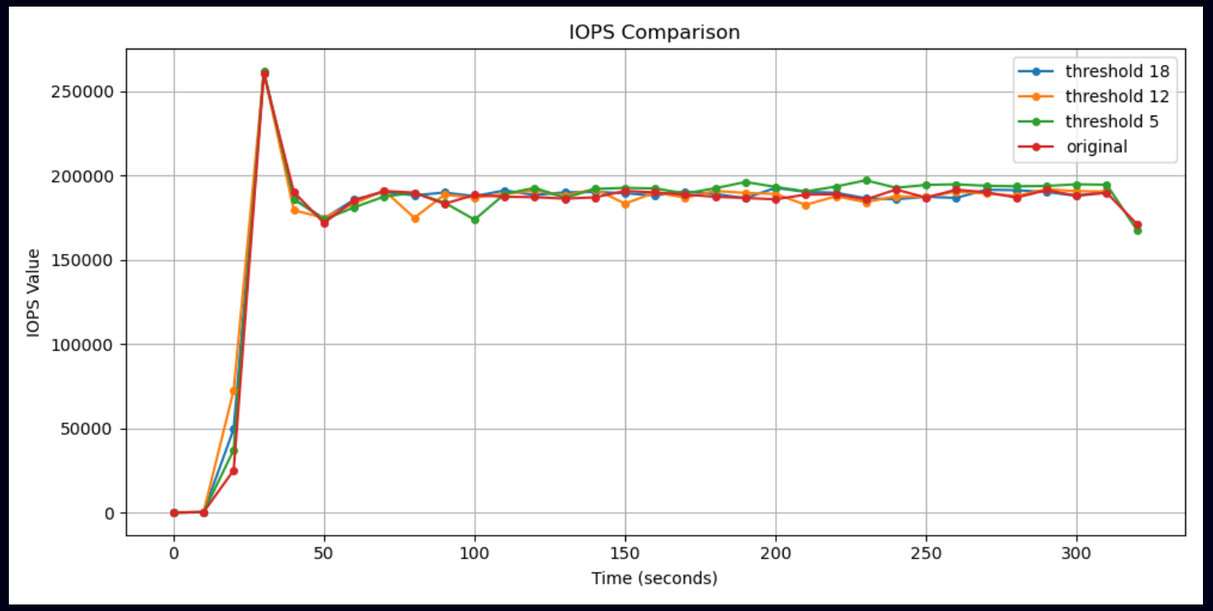
```
fio
--directory=/mnt/nvme0n1
--name=fio_test
--size=576m
--direct=1
--bs=4k
--numjobs=4
--time_based --runtime=300
--rw=randrw
--random_distribution=zipf:1.2
```

Normal Distribution 1.0 (Cumulative Distribution)

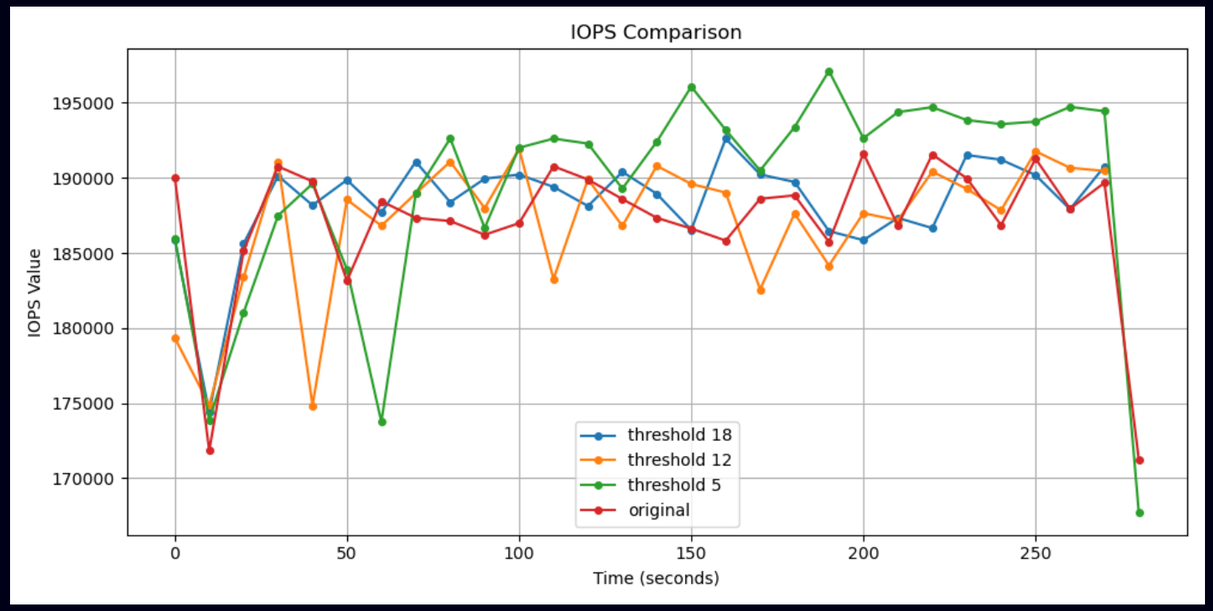


Since normal distribution 1.0 follows the normal distribution shape, the distribution will be concentrated on certain point. Howeve r, the gradient of CDF graph is even across the graph. Which means the mean of the distribution might be more than 25, so we we ren’t able to check the dramatically high gradient in the graph.

Normal Distribution 1.0 (IOPS)



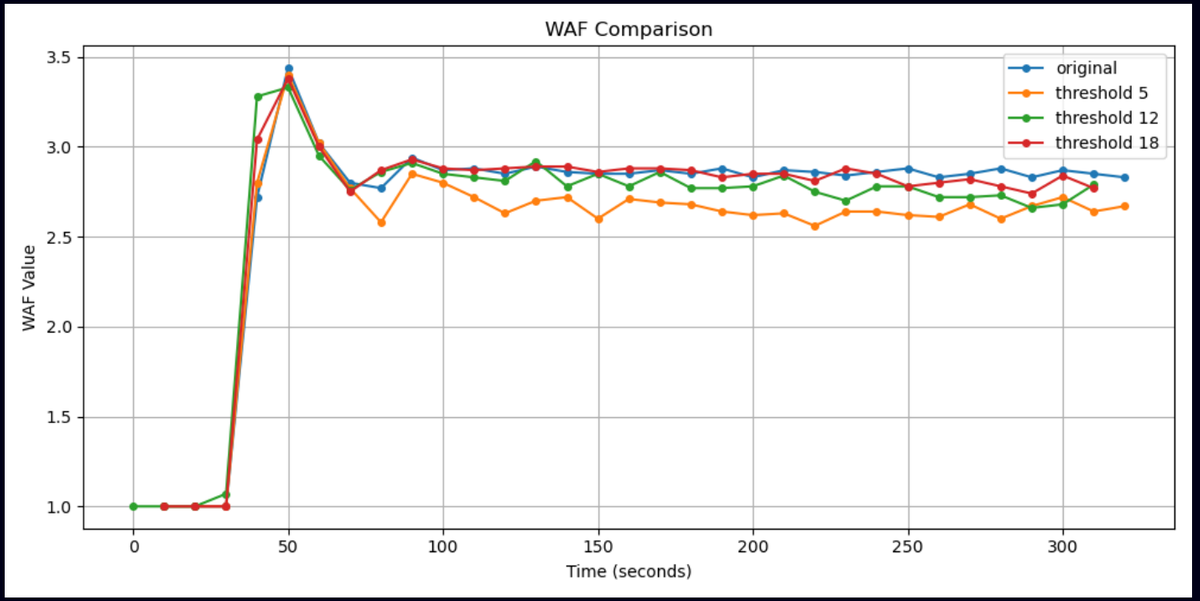
Original

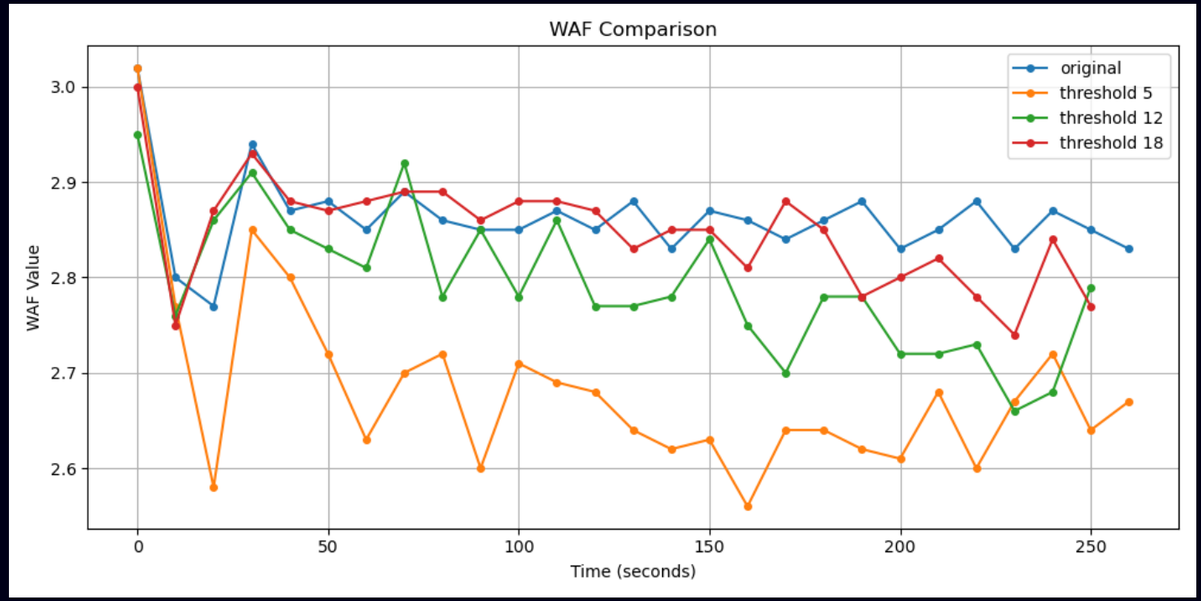


Emphasized

When we see original graph, it might be hard to see the difference because we weren’t able to check the mean of the write count distribution. But when we remove initial values and emphasize the graph, we can see threshold 5 has best IOPS values. From this , we can guess at the WAF graph, the threshold 5 will have lowest value. But the other threshold doesn’t show big difference from the original values.

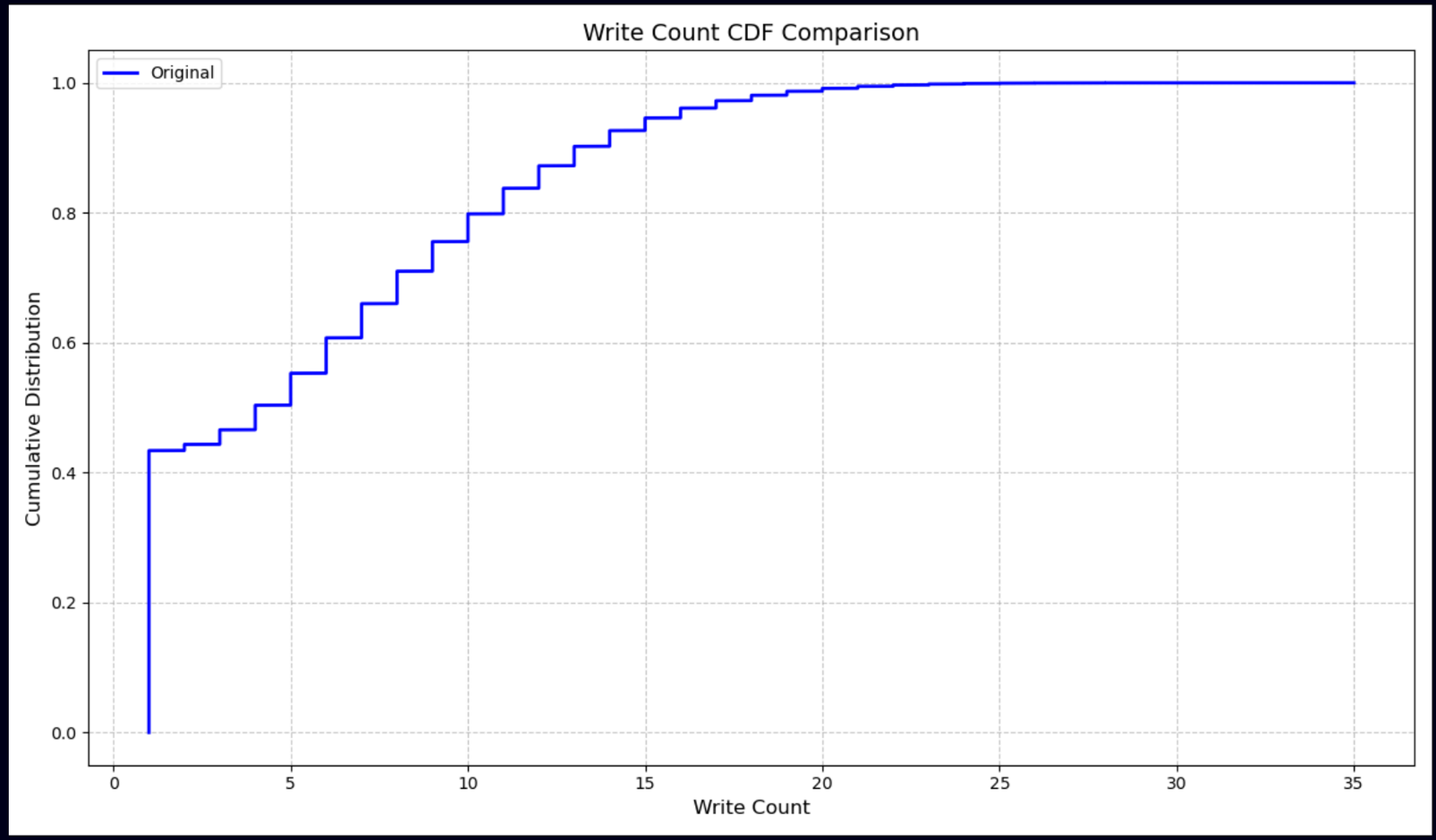
Normal Distribution 1.0 (WAF)





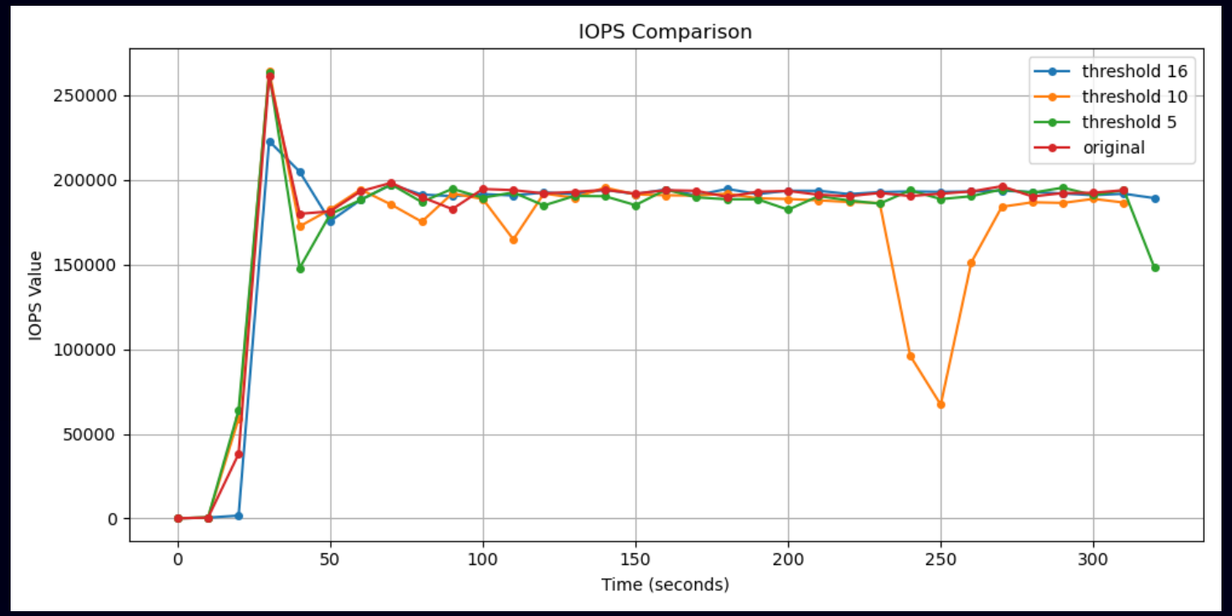
From the previous IOPS graph, we've guessed that the threshold 5 will show lowest WAF values because it showed best perform ance among 4 environment. As we can see from the graph, it showed significantly low values compare to other values when we e mphasized the graph. Before, the other threshold values didn't show dramatically better IOPS values compared to original one. B ut still we can see GC efficiency has increased from the WAF graph.

Pareto Distribution 0.5 (Cumulative Distribution)

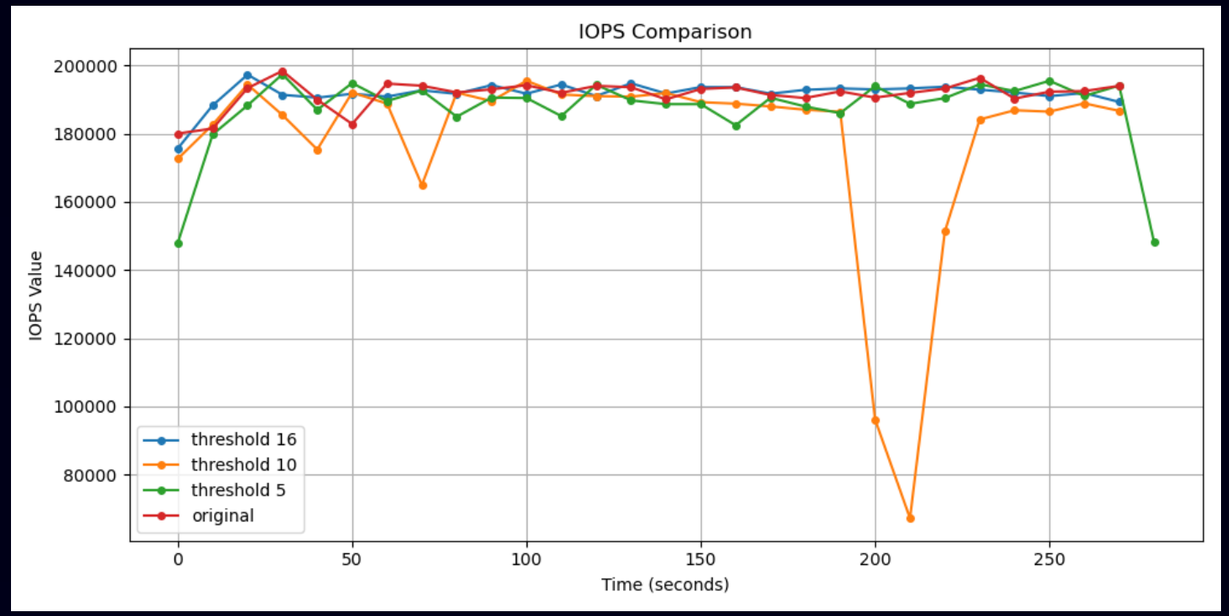


The Pareto distribution means the data will be accessed radically. What it means is that cold data will be accessed very few time s and on the other hand, the hot data will be accessed much more frequently. But because its alpha values is 0.5, it wouldn’t be v ery radical. But still, we can guess that the CDF graph will show greater gradient compared to the normal distribution 1.0 CDF gra phs. As we can see, the gradient is greater.

Pareto Distribution 0.5 (IOPS)



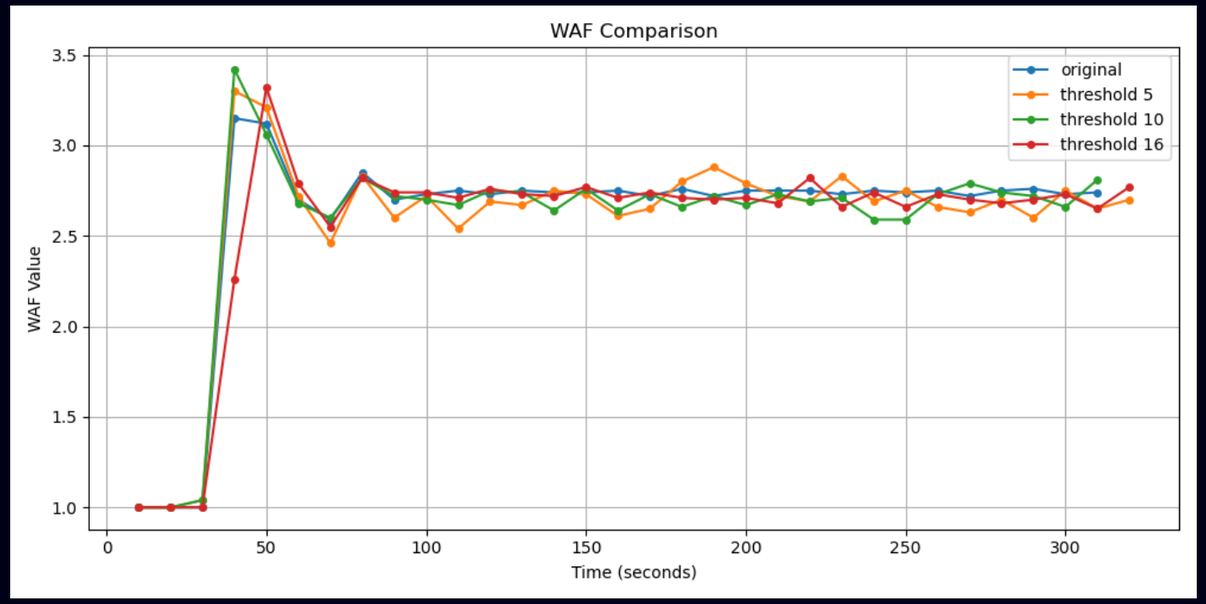
Original



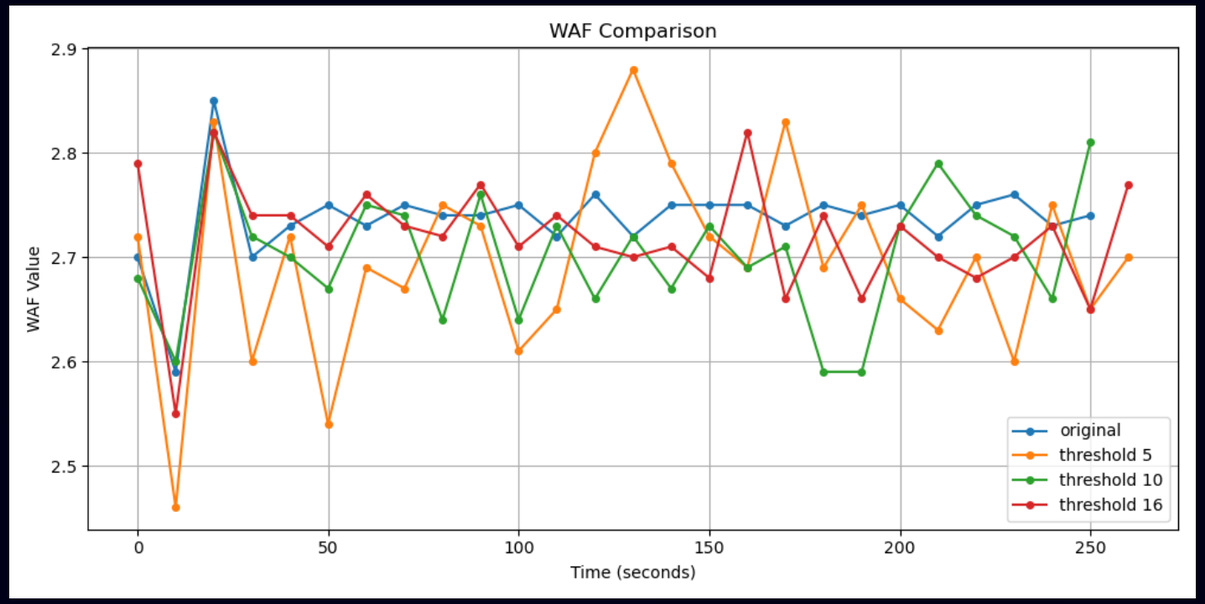
Emphasized

As we can see from the graph, it doesn’t show great difference between our hot/cold separation and original value. This might be because since we’ve tested in pareto distribution 0.5, as I’ve said before, it wasn’t very radically accessed. So Hot/Cold separatio n doesn’t show great impact on IOPS graph.

Pareto Distribution 0.5 (WAF)



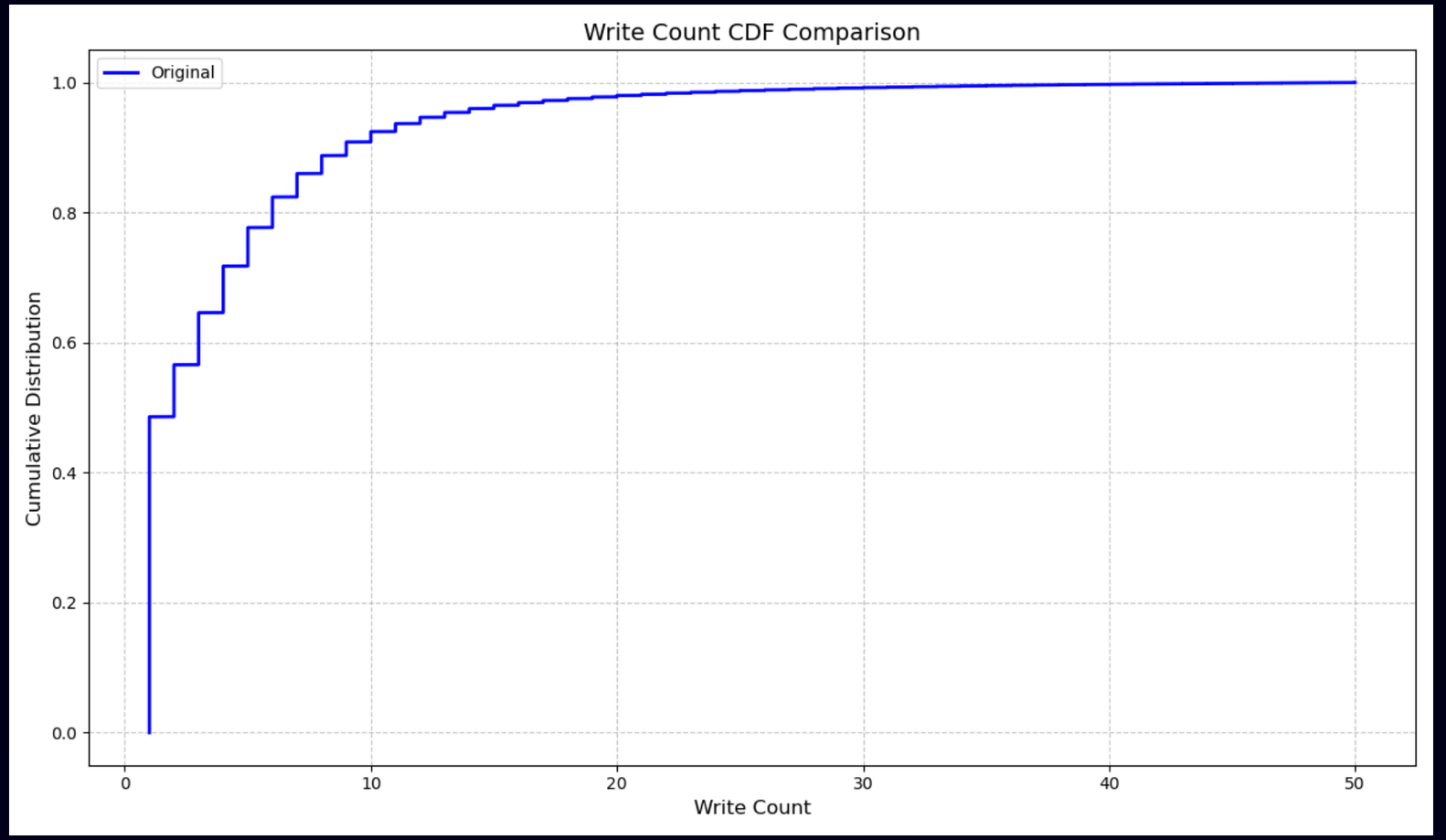
Original



Emphasized

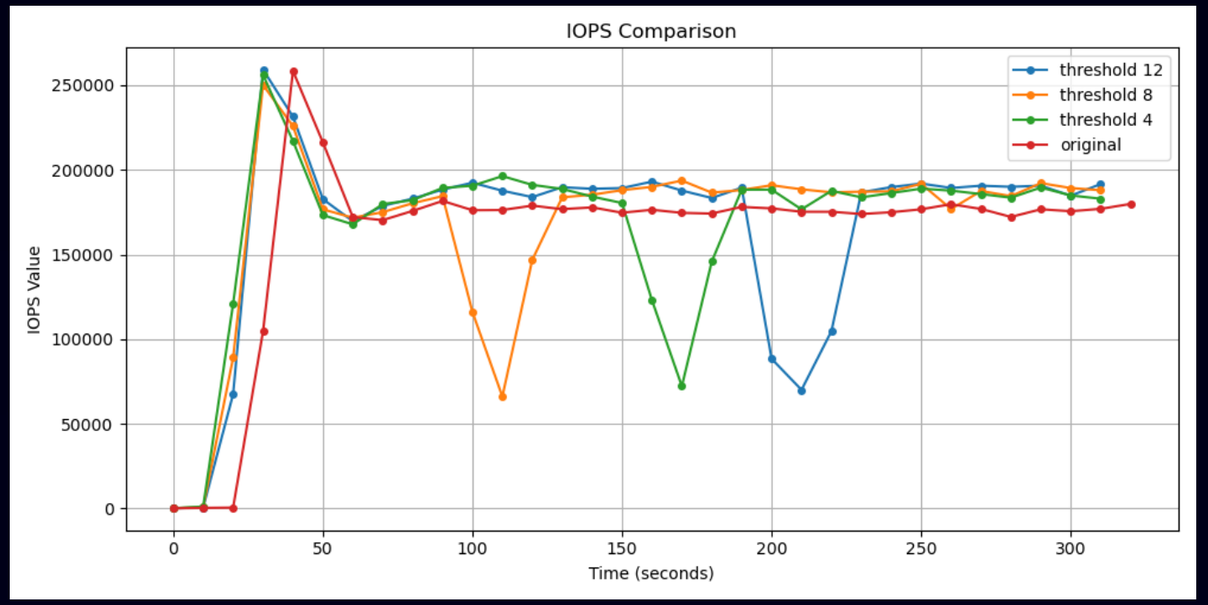
Since IOPS graph didn’t show great difference, WAF graph doesn’t show great difference either from the original. However, if we emphasize the graph, in an average, it showed small improvement. For example, threshold 10 has lower WAF than original in an a verage. Which means it showed small GC efficiency improvement throughout hot/cold separation.

Zipfian Distribution 0.8 (Cumulative Distribution)

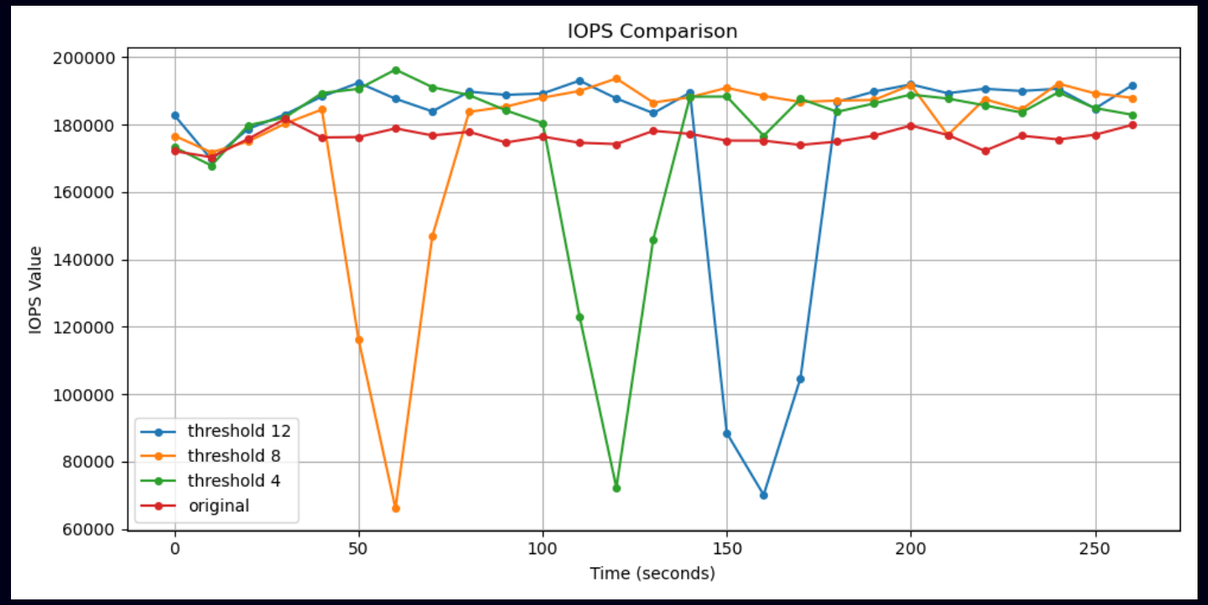


As you can see from the graph, it differs from normal distribution and pareto distribution. Compared to the Zipfian distribution 1.2(which will be showed in the future slide), it grows more gently because the number of Zipfian 0.8 LPN with lower write count has more than Zipfian 1.2 due to alpha value.

Zipfian Distribution 0.8 (IOPS)



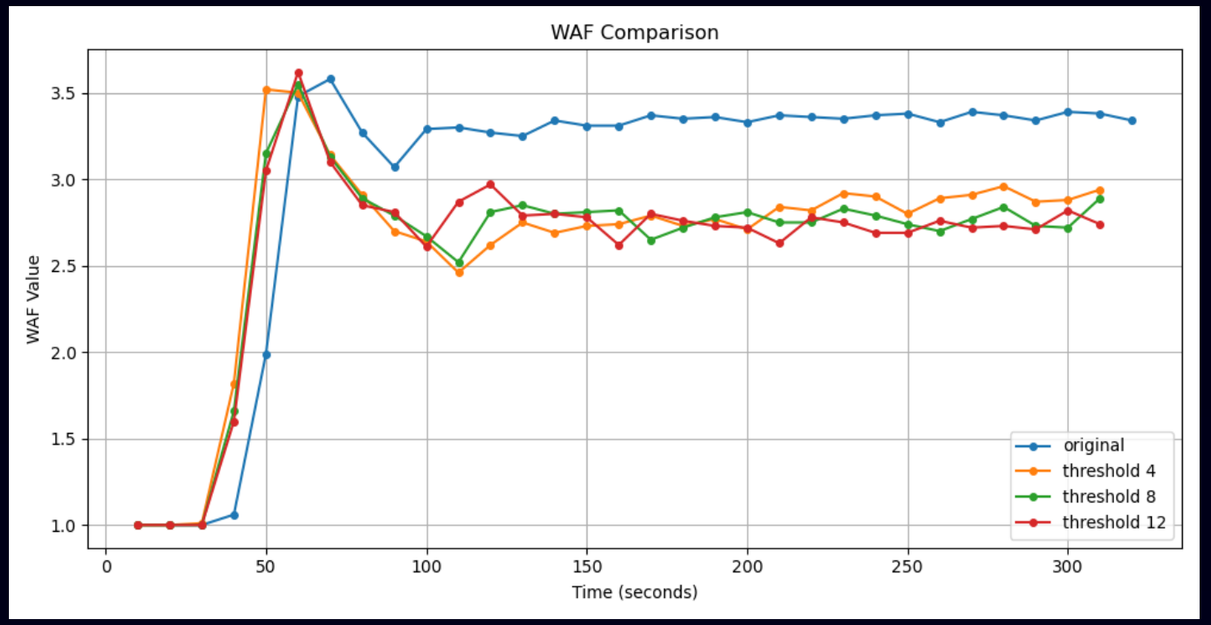
Original



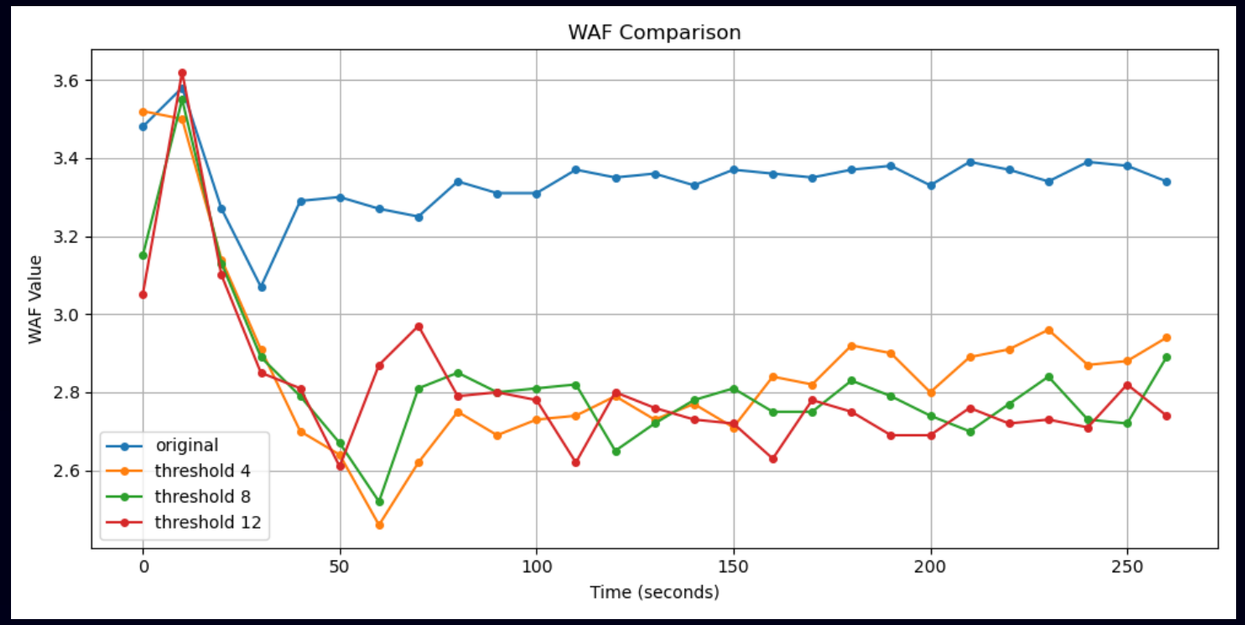
Emphasized

When we see the graph with initial values exclude, we can see that the hot/cold separation showed more efficiency compared to t he original(no hot/cold separation).There are some outlier showing when we apply hot/cold separation. My guess is when LPN ch anges from cold to hot, I think outlier values is showing due to change of line. Before looking into the WAF graph, we can expect WAF with hot/cold separation will show lower value compared to the original.

Zipfian Distribution 0.8 (WAF)



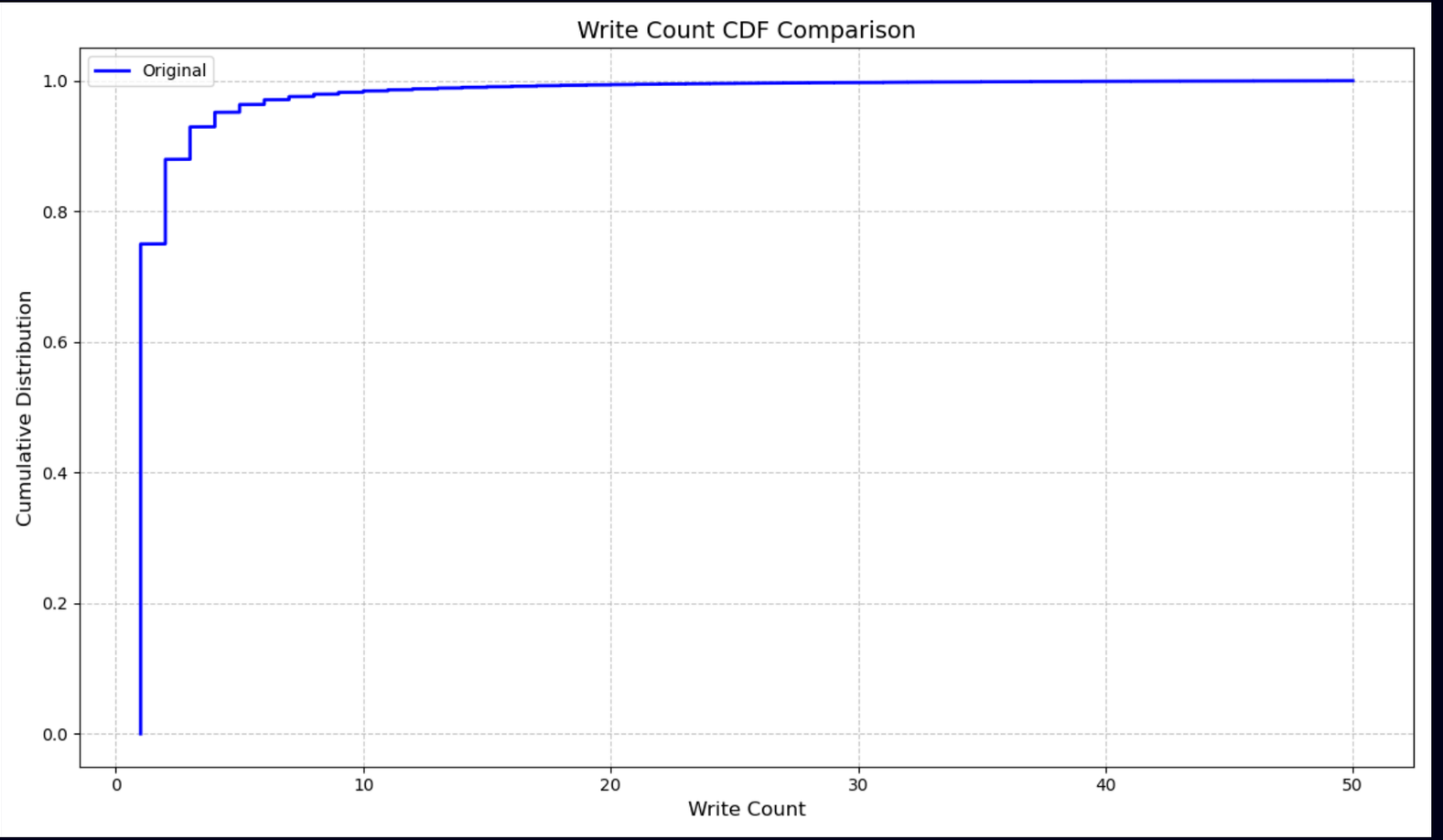
Original



Emphasized

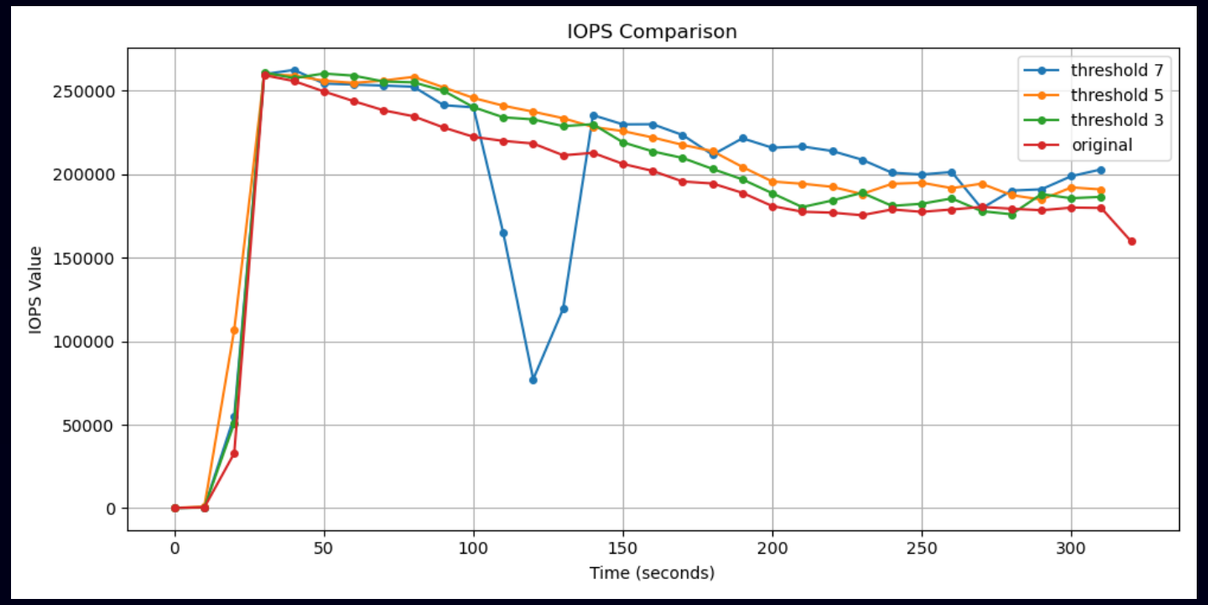
As we've guessed from the previous slide, the hot/cold separation WAF shows significantly lower value than original value. Which means due to the hot/cold separation, the GC efficiency has increased. At th e initial part, the threshold 4 showed lowest WAF but as write operation increases, the threshold 8 and 10 showed lower WAF compared to threshold 4

Zipfian Distribution 1.2 (Cumulative Distribution)

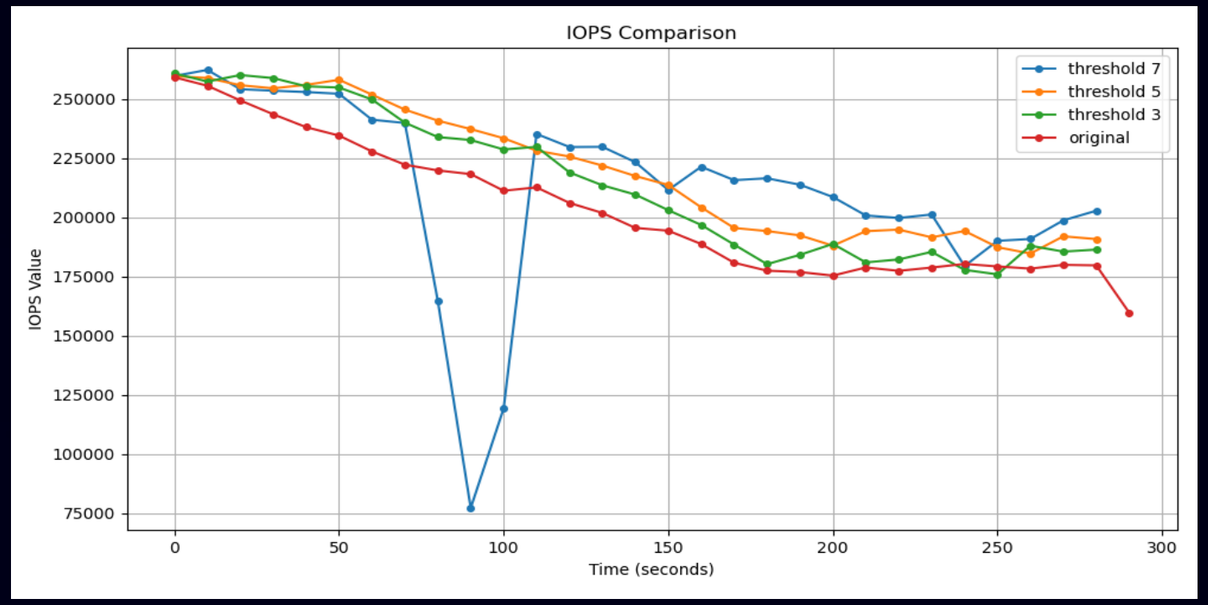


Compared to Zipfian distribution 0.8, we can see the graph radically increased because Zipfian 1.2 has LPN that has high write co unts. Because alpha values is higher, the number of write counts that are low are much more than Zipfian 0.8. So as the graph pr oves, the Zipfian 1.2 shows relatively higher value at the initial part compared to Zipfian 0.8.

Zipfian Distribution 1.2 (IOPS)



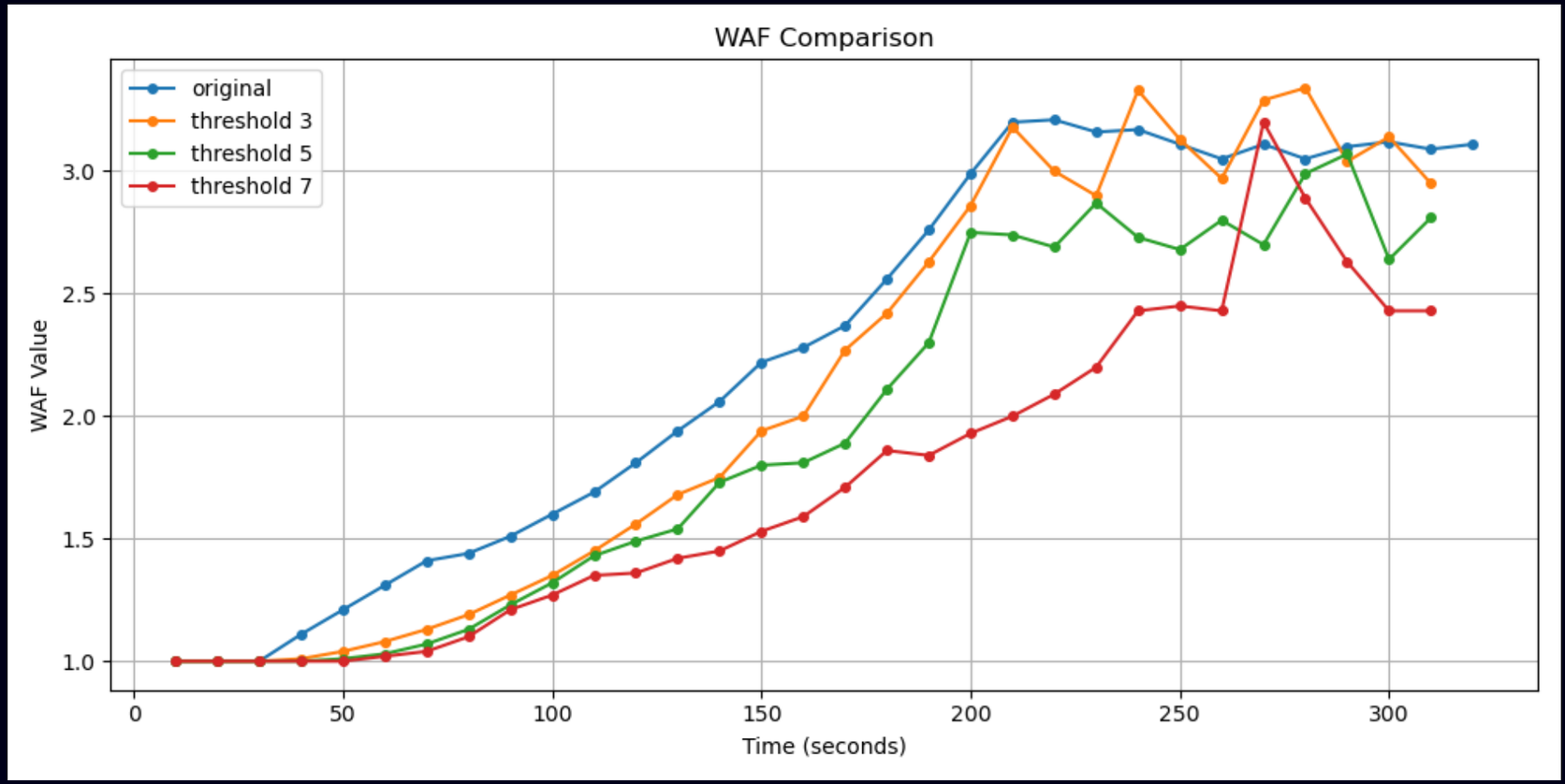
Original



Emphasized

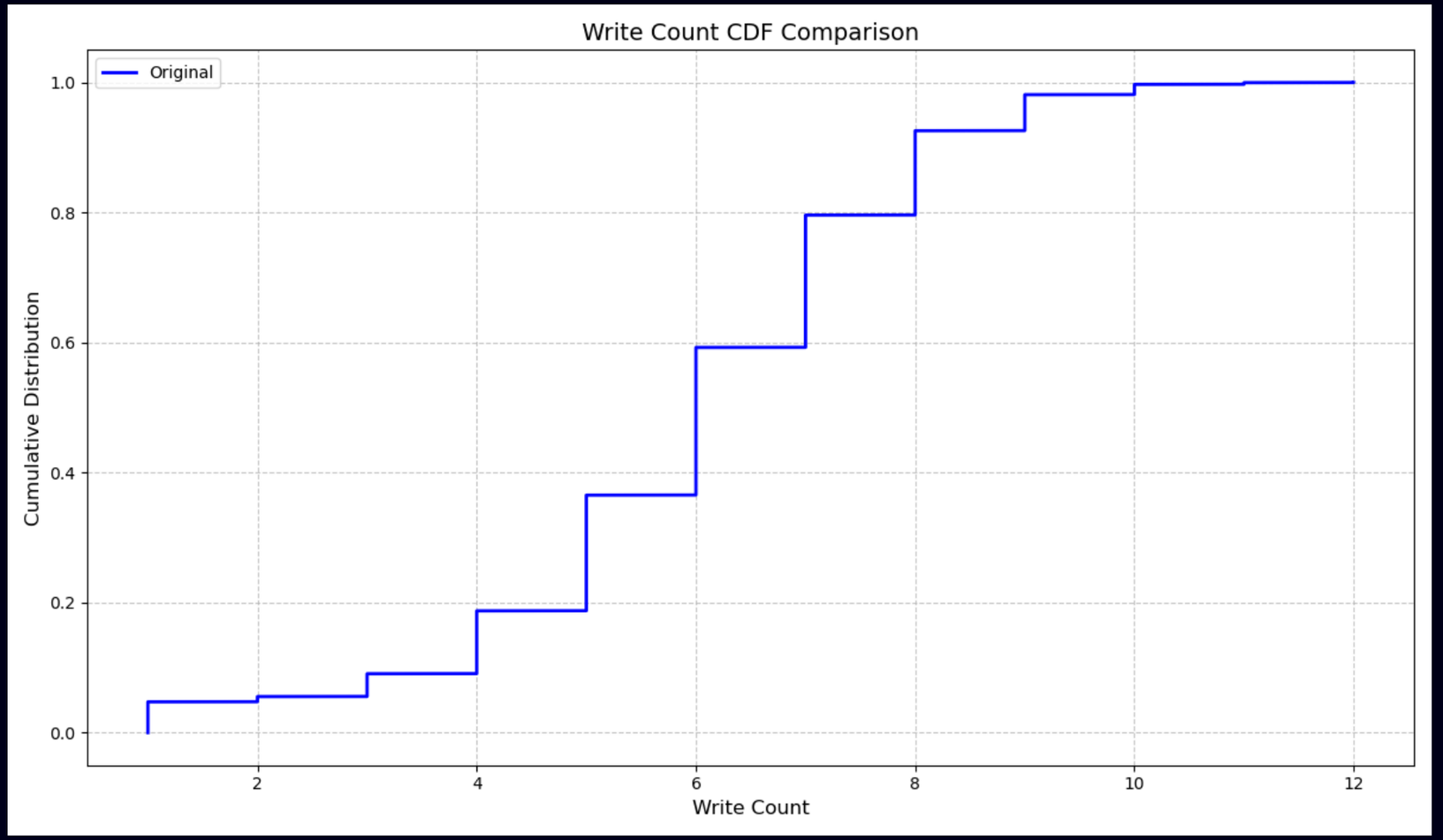
From the graph, we can see that the hot/cold separation showed increased IOPS than the original IOPS. There is outlier value at threshold 7 but except for the outlier value, all the hot/cold separation shows b etter performance than original. Because Zipfian 1.2 is more skewed than 0.8, the scale of increased I OPS performance is lower than 0.8

Zipfian Distribution 1.2 (WAF)



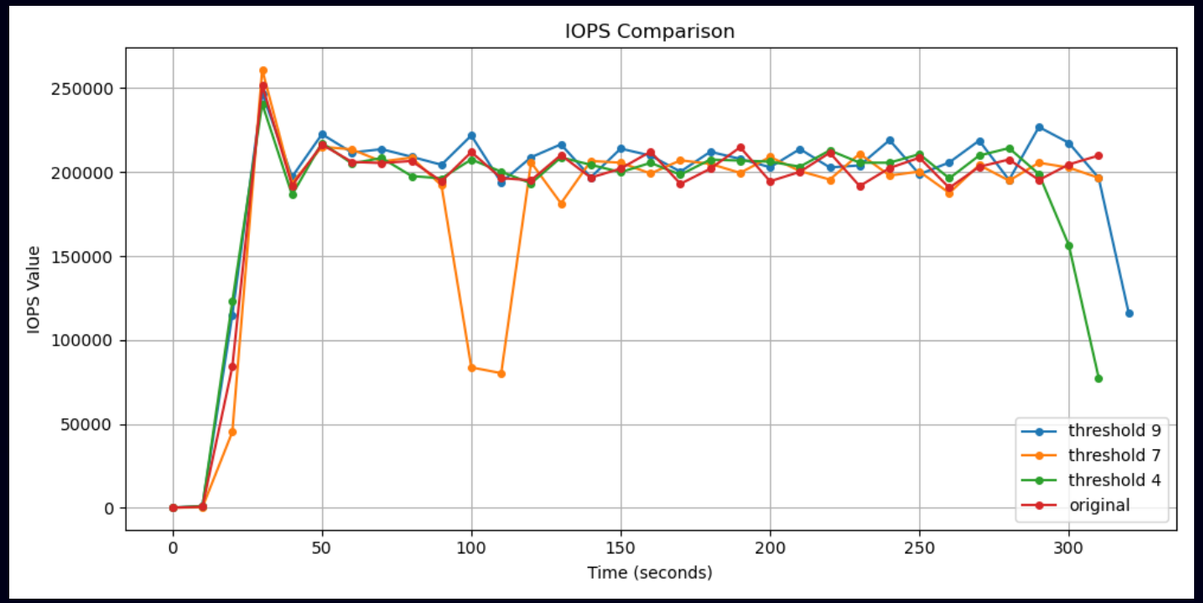
As we could've guessed from last slide, the IOPS with hot/cold separation was relatively high compare d to original, which means GC efficiency has increased, and this leads to lower WAF value. This graph proves our thought.

Random Distribution 1.0 (Cumulative Distribution)

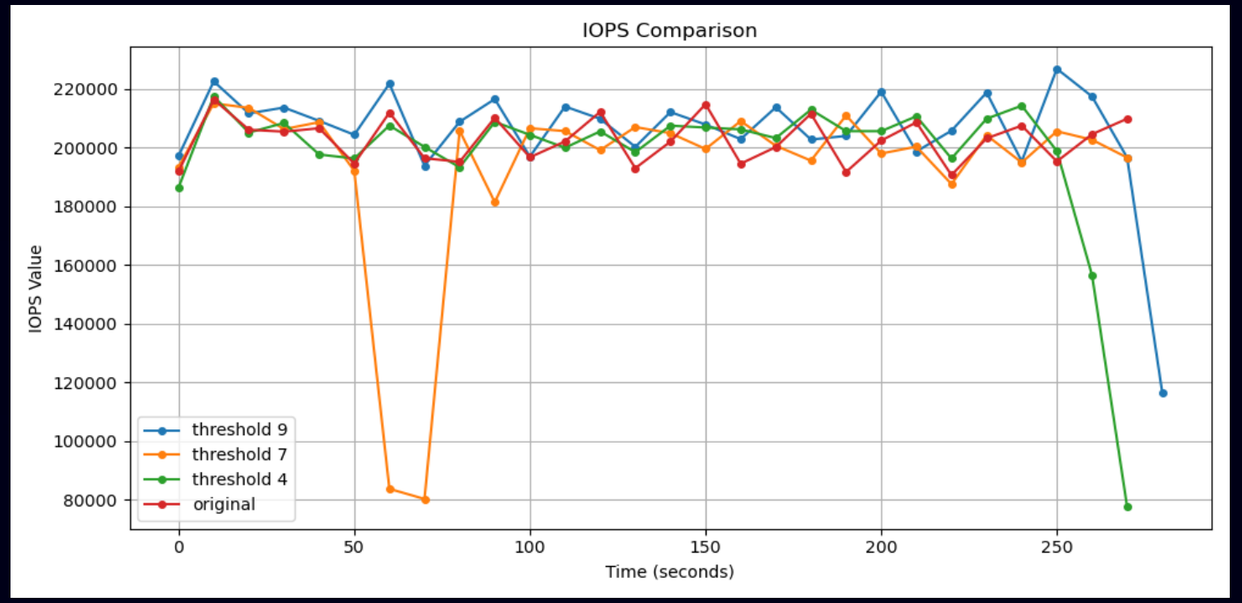


The random distribution makes all the LPN distribute write operation almost evenly. So, this leads write counts maximum lower c ompared to other test loads. As you can see from the graph, throughout write counts, it grows almost evenly(same gradient throu ghout graph), which means LPN write counts are almost even.

Random Distribution 1.0 (IOPS)



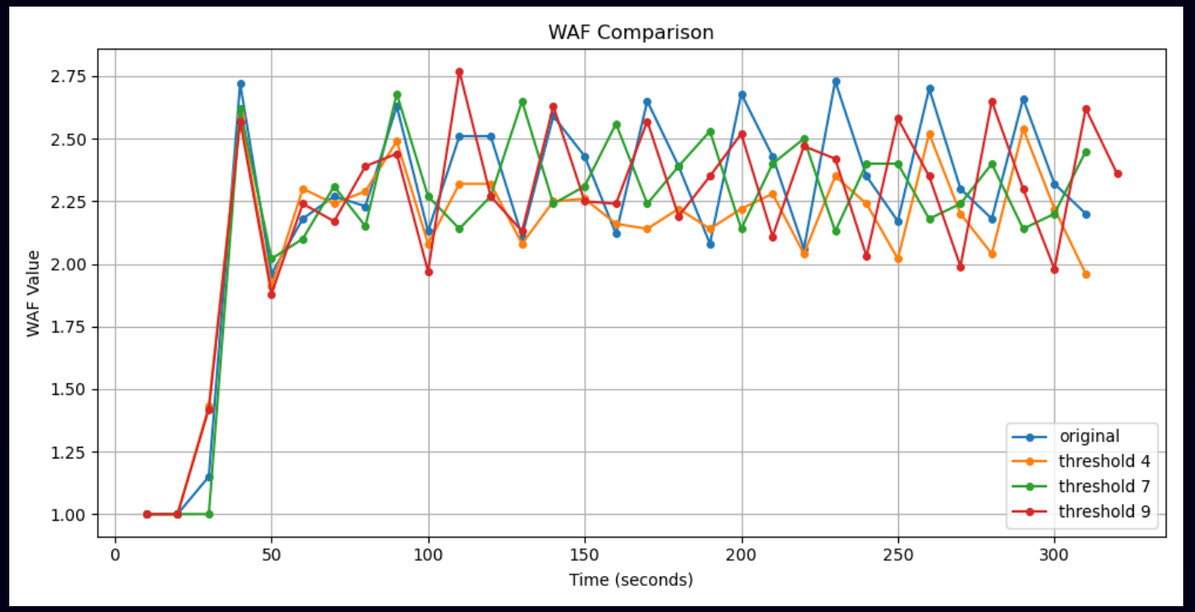
Original



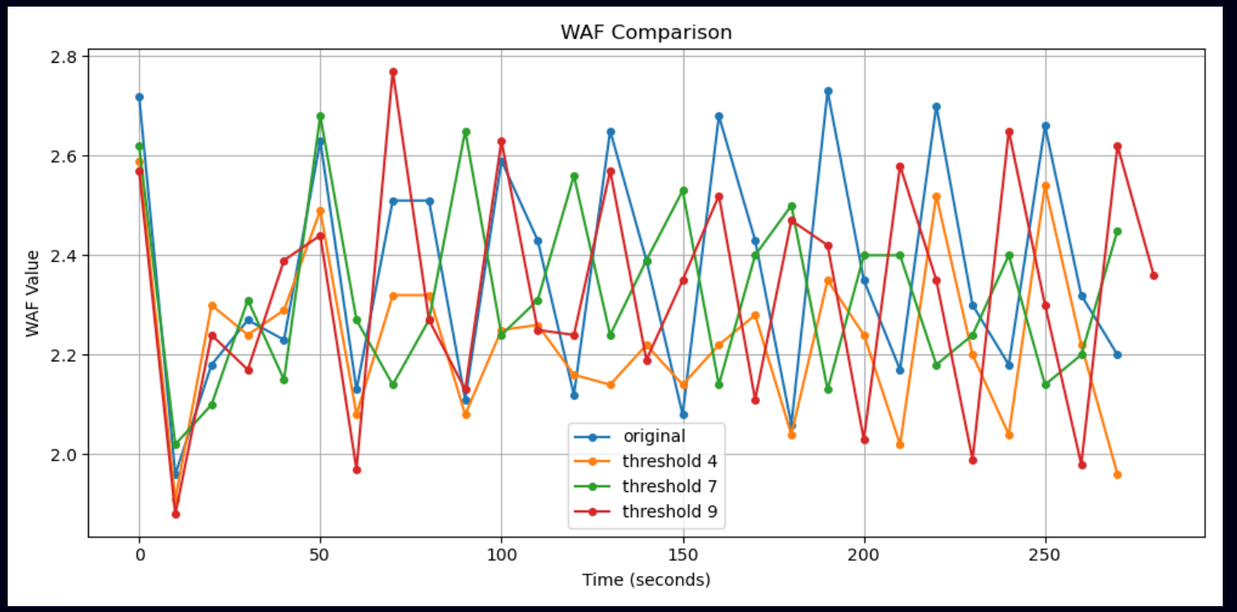
Emphasized

The random distribution distributes write operation almost evenly to all LPN. So, we can guess that the hot/cold operation won’t show good performance compared to the other test loads. As we can see fro m the IOPS graph, even if we emphasize the size of graph, whether we applied hot/cold separation or n ot, it shows similar IOPS

Random Distribution 1.0 (WAF)



Original



Emphasized

Since write operation is distributed throughout all the LPN evenly, the WAF will also won’t show better p
erformance. So, as we can see from the graph, the WAF between original and hot/cold separation does
n’t show a big difference.

Concluding with overall Analysis

Although not all the test showed better WAF or IOPS performance, some sh owed better than the original ones. By this, we can tell that hot and cold separation showed better efficiency on GC operation.

Identifying I/O Access Patterns

1

Data Access Frequency

Analyzing the frequency of data access over time helps identify frequently accessed data (hot).

2

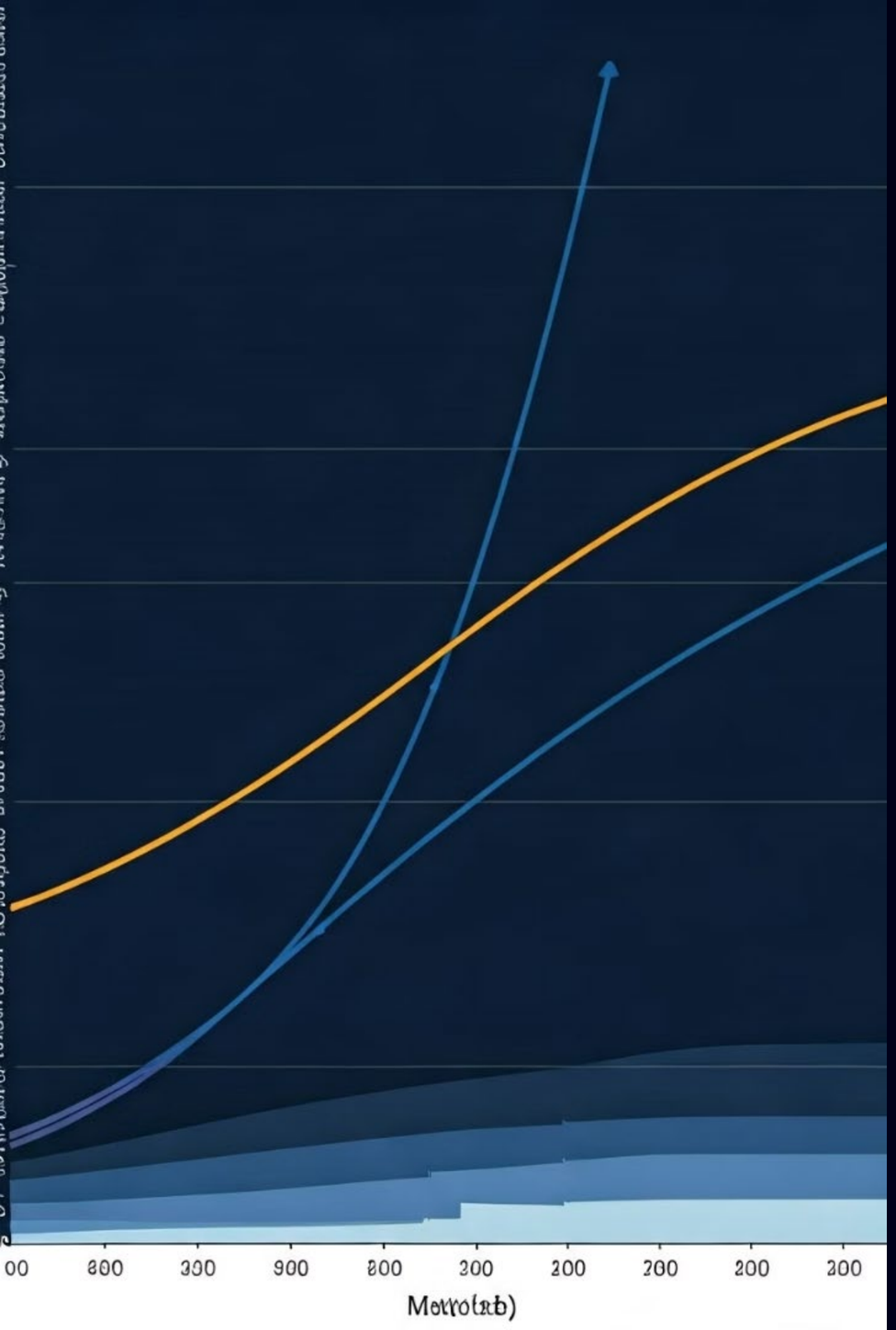
Sequential vs. Random Access

Determining the access patterns, sequential or random, provides insights into data locality.

3

Read vs. Write Operations

Analyzing the ratio of read and write operations reveals the nature of the workload and its impact on storage.



Determining the Optimal Threshold

1 Threshold Value Impact

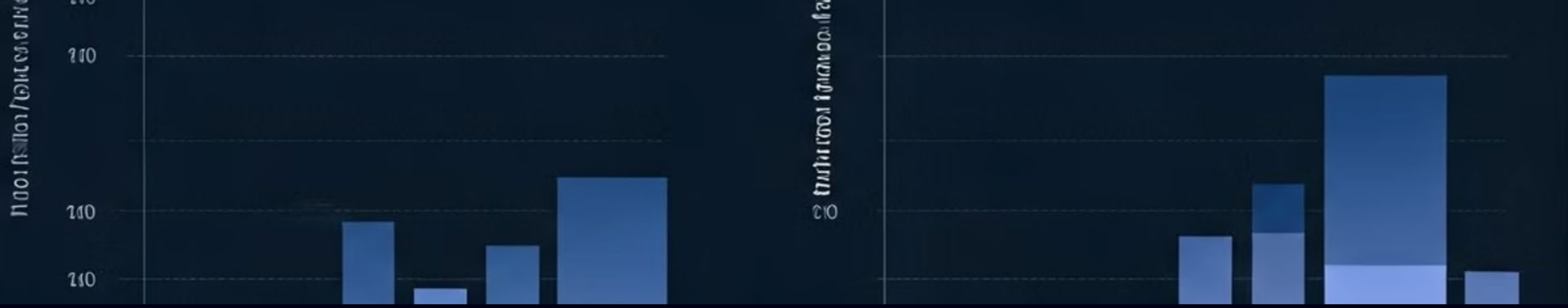
The threshold value determines the boundary between hot and cold data, affecting GC efficiency.

2 Performance Optimization

Finding the optimal threshold maximizes GC efficiency while minimizing performance overhead.

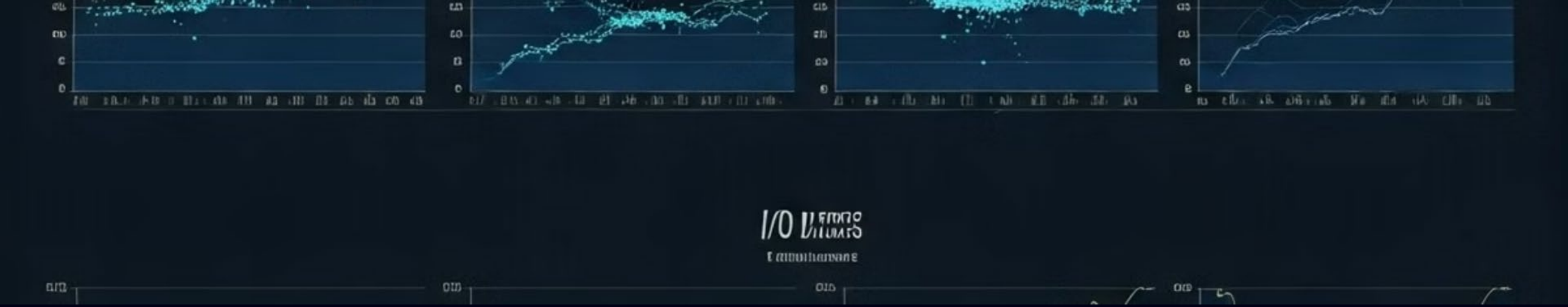
3 Workload-Specific Optimization

The optimal threshold may vary across different workloads, requiring individual analysis and fine-tuning.



Evaluating Performance Increase

Metric	Baseline (No Separation)	Optimal Threshold
IOPS (Operations Per Second)	1000	1500
Latency (Milliseconds)	10	5
Garbage Collection Time (Seconds)	30	15



Visualizing I/O Access Distribution

Data Access Frequency

Visualize the distribution of data access frequency for different workloads.

I/O Operations over Time

Illustrate the temporal patterns of read a nd write operations.

Latency Analysis

Track the changes in latency over time, i dentifying bottlenecks and performance i ssues.