

Reproducing Fault Injection Experiments: The Tail at Store (FAST'16)

Running Trace Replayer

1. Tools Installation

```
sudo apt update
sudo apt install python3-pip build-essential git
pip install python-openstackclient python-swiftclient
```

```
Successfully installed PrettyTable-3.11.0 autopage-0.5.2 charset-normalizer-3.4.2 cliff-4.7.0 cmd2-2.5.11 debtcollector-3.0.0 dogpile.cache-1.3.4 importlib-metadata-8.5.0 iso8601-2.1.0 jmespath-1.0.1 keystoneauth1-5.8.1 msgpack-1.1.0 netaddr-1.3.0 openstacksdk-4.0.1 os-service-types-1.7.0 osc-lib-4.0.2 oslo.config-9.6.0 oslo.i18n-6.4.0 oslo.serialization-5.5.0 oslo.utils packaging-25.0 pbr-6.1.1 platformdirs-4.3.6 pyparsing-3.1.4 pyperclip-1.9.0 python-cinderclient-9.6.0 python-keystoneclient-5.5.0 python-openstackclient-7.1.4 python-swiftclient-4.8.0 pytz-2025.2 requests-2.32.3 requestsexceptions-1.4.0 rfc3986-2.0.0 stevedore-5.3.0 typing-extensions-4.13.2 wcwidth-0.2.13 wrapt-1.17.2 zipp-3.20.2
cc@trace-replayer:~$ |
```

2. OpenStack RC File

```
scp -i "D:\key\yizzz-mj-trace.pem" "D:\key\CHI-210850-openrc.sh" cc@192.5.86.230:~
```

```
C:\Users\jeezx>scp -i "D:\key\yizzz-mj-trace.pem" "D:\key\CHI-210850-openrc.sh" cc@192.5.86.230:~
CHI-210850-openrc.sh 100% 805 3.2KB/s 00:00
C:\Users\jeezx>
```

```
source CHI-210850-openrc.sh
```

```
Last login: Tue May 27 12:45:22 2025 from 180.241.45.164
cc@trace-replayer:~$ source CHI-210850-openrc.sh
(yizreelschwartz180304@gmail.com) Please enter your Chameleon CLI password:
cc@trace-replayer:~$ |
```

3. Trace from Object Store

```
openstack container list
```

```
cc@trace-replayer:~$ openstack container list
+-----+
| Name |
+-----+
| a.c |
| cudnn |
| ruidan_traces |
+-----+
```

```
openstack object list ruidan_traces
```

```
cc@trace-replayer:~$ openstack object list ruidan_traces
```

Name
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/offset_popularity.txt
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.stats
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace_temp
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/offset_popularity.txt
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.stats
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.trace
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.trace_temp
home/cc/trace_eval/Alibaba.per_300mins.iops_p15.dev_810.21.csv/offset_popularity.txt

openstack object save ruidan_traces

home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace

```
cc@trace-replayer:~$ openstack object save ruidan_traces home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace
```

4. Make application credential

nano appcred.env

```
GNU nano 4.8 appcred.env
export OS_AUTH_TYPE=v3applicationcredential
export OS_AUTH_URL=https://chi.uc.chameleoncloud.org:5000/v3
export OS_APPLICATION_CREDENTIAL_ID=16ebd49c5ea34f649647ddd3aa7de5dd
export OS_APPLICATION_CREDENTIAL_SECRET=4QLhltG_h-Ufz20ZifNTQTDQAJHW5rZCAdqSJZat5o3VX_rYyiJ6oL0_0s9ctVtt9SzoyZfbjXa-MeJ-YjxCqQ
```

openstack container list

```
cc@trace-replayer:~/trace-ACER/trace_replayer$ openstack container list
```

Name
a.c
cudnn
ruidan_traces

openstack object list ruidan_traces

```
cc@trace-replayer:~/trace-ACER/trace_replayer$ openstack object list ruidan_traces
```

Name
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/offset_popularity.txt
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.stats
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace
home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace_temp
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/offset_popularity.txt
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.stats
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.trace
home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.trace_temp
home/cc/trace_eval/Alibaba.per_300mins.iops_p15.dev_810.21.csv/offset_popularity.txt
home/cc/trace_eval/Alibaba.per_300mins.iops_p15.dev_810.21.csv/read_io.stats
home/cc/trace_eval/Alibaba.per_300mins.iops_p15.dev_810.21.csv/read_io.trace

mkdir -p ~/traces

openstack object save ruidan_traces \

"home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace" \

--file ~/traces/read_io.trace

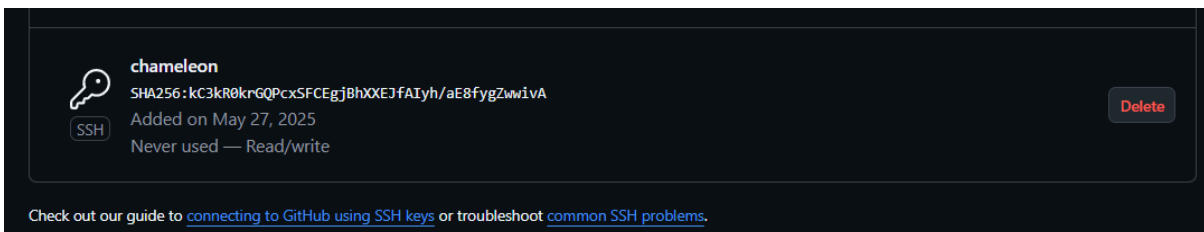
ls -lh ~/traces/read_io.trace

```
cc@trace-replayer:~$ mkdir -p ~/traces
cc@trace-replayer:~$ openstack object save ruidan_traces \
> "home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace" \
> --file ~/traces/read_io.trace
cc@trace-replayer:~$ ls -lh ~/traces/read_io.trace
-rw-rw-r-- 1 cc cc 4.7M May 27 17:36 /home/cc/traces/read_io.trace
```

5. Clone and Build Replayer

ssh-keygen -t ed25519 -C "legobatman201003@gmail.com"

```
cc@trace-replayer:~$ ssh-keygen -t ed25519 -C "legobatman201003@gmail.com"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/cc/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/cc/.ssh/id_ed25519
Your public key has been saved in /home/cc/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:kC3kR0krGQPcxSFCEgjBhXXEJfAIyh/aE8fygZwwivA legobatman201003@gmail.com
The key's randomart image is:
+--[ED25519 256]--+
|***BB**==o      |
|B+=.O=oOo.      |
|+.EO =B +       |
|  + * . =       |
| . + . S        |
|                |
|                |
|                |
+-----[SHA256]-----+
```



cat ~/.ssh/id_ed25519.pub

git clone git@github.com:ucare-uchicago/trace-ACER.git

```
cc@trace-replayer:~$ git clone git@github.com:ucare-uchicago/trace-ACER.git
Cloning into 'trace-ACER'...
The authenticity of host 'github.com (140.82.113.4)' can't be established.
ECDSA key fingerprint is SHA256:p2QAMXNIC1TJYWeIOtrVc98/R1BUFWu3/LiyKgUfQM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com,140.82.113.4' (ECDSA) to the list of known hosts.
remote: Enumerating objects: 1434, done.
remote: Counting objects: 100% (550/550), done.
remote: Compressing objects: 100% (338/338), done.
remote: Total 1434 (delta 264), reused 439 (delta 168), pack-reused 884 (from 1)
Receiving objects: 100% (1434/1434), 187.74 MiB | 24.86 MiB/s, done.
Resolving deltas: 100% (669/669), done.
Updating files: 100% (164/164), done.
```

cd ~/trace-ACER/trace_replayer
gcc io_replayer.c -o io_replayer -lpthread

6. Run the build replayer

sudo ./io_replayer /home/cc/traces/read_io.trace /dev/nvme0n1 replay.log

```
cc@trace-replayer:~/trace-ACER/trace_replayer$ sudo ./io_replayer /home/cc/traces/read_io.trace /dev/nvme0n1 replay.log
Device ==> /home/cc/traces/read_io.trace
Trace ==> /dev/nvme0n1
Logfile ==> replay.log
```

From what I understand in the io_replayer code, the parser function only processes files with space-separated values. This is because the code uses the strtok() function, which expects fields (timestamp, dev_id, offset, size, io_type) to be separated by blank spaces, not commas.

```
// 1. request arrival time in "ms"
timestamp[i] = atof(strtok(one_io, " "));
// 2. device number (not needed)
strtok(NULL, " ");
// 3. block number (offset)
oft[i] = atoll(strtok(NULL, " "));
```

Also, in the read_trace() function, it reads all lines in the trace file without checking or skipping the header in the first line.

```
while (fgets(line, sizeof(line), trace) != NULL) {
    line[strlen(line) - 1] = '\0';
    if (((*req)[i] = malloc((strlen(line) + 1) * sizeof(char))) == NULL) {
        fprintf(stderr, "memory allocation error (%d)!\n", __LINE__);
        exit(1);
    }

    strcpy((*req)[i], line);
    i++;
}
fclose(trace);

return nr_lines;
```

In the trace file I used

(/home/cc/trace_eval/Alibaba.per_300mins.iops_p10.dev_745.86.csv/read_io.trace), the format uses commas as the delimiter.

```
cc@trace-replayer:~/trace-ACER/trace_replayer$ head -n 6 /home/cc/traces/trace_p10.trace
ts_record,dev_num,offset,size,io_type
40521.64086914063,745,51573325824,4096,1
90170.68896484376,745,107374215168,4096,1
111598.875,745,55834705920,4096,1
111599.3818359375,745,55868260352,4096,1
111599.73803710938,745,55834640384,4096,1
```

Since the trace file is in CSV format, we need to preprocess it first.

tail -n +2 ~/traces/trace_p10.trace | tr ',' ' ' > ~/traces/trace_p10_clean.trace

tail -n +2 → remove the first line (header).
tr ',' → convert the delimiter into blank space.

After preprocessing, we obtain a new trace file format that uses blank spaces as the delimiter.

```
cc@trace-replayer:~/trace-ACER/trace_replayer$ head -n 6 /home/cc/traces/trace_p10_clean.trace
40521.64086914063 745 51573325824 4096 1
90170.68896484376 745 107374215168 4096 1
111598.875 745 55834705920 4096 1
111599.3818359375 745 55868260352 4096 1
111599.73803710938 745 55834640384 4096 1
115586.28686523438 745 55868264448 4096 1
```

After this, we can run the replayer using the preprocessed trace file.

sudo ./io_replayer /dev/nvme2n1 ~/traces/trace_p10_clean.trace trace_p10.log

```
cc@trace-replayer:~/trace-ACER/trace_replayer$ sudo ./io_replayer /dev/nvme2n1 ~/traces/trace_p10_clean.trace trace_p10.log
Device ==> /dev/nvme2n1
Trace ==> /home/cc/traces/trace_p10_clean.trace
Logfile ==> trace_p10.log
there are [114672] IOs in total in trace:/home/cc/traces/trace_p10_clean.trace
40521.64086914063 745 51573325824 4096 1
Disk size is 7325650 MB
      in Bytes 7681501126656 B
40521.64,51573325824,4096,1
Start doing IO replay...
Progress: 100.00% (114686/114672), Late rate: 0.00% (0), Slack rate: 99.58% (114189)

All done!
=====
Total run time: 17802510.000 ms
Late rate: 0.00%
Slack rate: 99.62%
sh: 1: python: not found
Statistics output = trace_p10.log.stats
```

watch the process realtime

watch -n 1 "ps -eo pid,ppid,user,start_time,etime,time,cmd | grep io_replayer"

```
Every 1.0s: ps -eo pid,ppid,user,start_time,etime,time,cmd | grep io_replayer          tx-trace-replayer: Tue Jun 10 11:25:17 2025
65302  63096 root    11:15      09:42 00:00:00 sudo ./io_replayer /dev/nvme2n1 /home/cc/traces/trace_p10_clean.trace trace_p10.log
65303  65302 root    11:15      09:42 00:00:00 ./io_replayer /dev/nvme2n1 /home/cc/traces/trace_p10_clean.trace trace_p10.log
66203  64130 cc      11:25      00:02 00:00:00 watch -n 1 ps -eo pid,ppid,user,start_time,etime,time,cmd | grep io_replayer
66212  66203 cc      11:25      00:00 00:00:00 watch -n 1 ps -eo pid,ppid,user,start_time,etime,time,cmd | grep io_replayer
66213  66212 cc      11:25      00:00 00:00:00 sh -c ps -eo pid,ppid,user,start_time,etime,time,cmd | grep io_replayer
66215  66213 cc      11:25      00:00 00:00:00 grep io_replayer
```

Firmware-Induced Long Tail Experiment

1. Experiment Objectives

Simulate the *firmware-induced long tail* failure as identified in large-scale SSD deployments. This type of failure is characterized by I/O operations exhibiting abnormally high latency due to internal firmware behavior, even though the device appears healthy to the system

2. Detailed Planned for Experimental Injection

- **Failure detail:**

SSD appears stuck in a persistent slow mode, where I/Os show high latency (100–500ms) despite no error. The issue is visible only in tail latency logs and not hardware counters.

- **Failure source:**

Trace used: Alibaba.per_300mins.iops_p15.dev_810.21.csv/read_io.trace According to the paper, the failure stems from rare firmware conditions in SSDs.

- **Plan to modify the replayer/trace:**

Modify io_replayer.c to inject a slow mode (firmware tail) with a probability of 0.5% for a virtual disk. While in slow mode, every I/O to that disk is delayed (100–500ms). Trace remains unchanged.

- **Configuration:**

- Disk ID: 2 (simulated via thread index % 4)
- Delay range: 100ms–500ms
- Probability: 0.5% per I/O
- Duration: 1000 I/Os per slow mode
- Device: /dev/nvme2n1
- Trace file: trace_p15_sample10k_clean.trace

3. Setup & Trace Preparation

Download the trace from the object store

**openstack object save ruidan_traces **

**"home/cc/trace_eval/Alibaba.per_300mins.iops_p15.dev_810.21.csv/read_io.trace" **

--file ~/traces/trace_p15_full.trace

Extract the first 10,000 lines for sampling

head -n 10000 ~/traces/trace_p15_full.trace > ~/traces/trace_p15_sample10k.trace

Clean the trace format

tail -n +2 ~/traces/trace_p15_sample10k.trace | tr ',' ' ' >

~/traces/trace_p15_sample10k_clean.trace

```

cc@tx-trace-replayer:~$ openstack object save ruidan_traces \
> "home/cc/trace_eval/Alibaba.per_300mins.iops_p15.dev_810.21.csv/read_io.trace" \
> --file ~/traces/trace_p15_full.trace
cc@tx-trace-replayer:~$ cd traces
cc@tx-trace-replayer:~/traces$ openstack object save ruidan_traces \
> "home/cc/trace_eval/Alibaba.per_300mins.iops_p15.dev_810.21.csv/read_io.trace" \
> --file ~/traces/trace_p15_full.trace
cc@tx-trace-replayer:~/traces$ ls
Alibaba_read_io.trace  trace_p100.trace  trace_p100_sample100k_rw.trace  trace_p100_sample50k.trace  trace_p10_sample10k.trace
read_io.trace          trace_p100_clean.trace  trace_p100_sample10k.trace  trace_p100_sample5k.trace  trace_p15_full.trace
tencent_read_io.trace  trace_p100_sample100k.trace  trace_p100_sample20k.trace  trace_p10_clean.trace
trace_p10.trace        trace_p100_sample100k_mlc_ts.trace  trace_p100_sample30k.trace  trace_p10_mixed.trace
cc@tx-trace-replayer:~/traces$ head -n 10000 ~/traces/trace_p15_full.trace > ~/traces/trace_p15_sample10k.trace

cc@tx-trace-replayer:~/traces$ head -n 10 ~/traces/trace_p15_sample10k.trace
ts_record,dev_num,offset,size,io_type
12600.27392578125,810,49787437056,16384,1
12600.383056640623,810,38437650432,16384,1
12600.87890625,810,49787453440,32768,1
12600.909912109377,810,38437666816,32768,1
12601.31201171875,810,49787486208,65536,1
12601.356689453123,810,38437699584,65536,1
12602.406005859377,810,38437765120,131072,1
12602.4580078125,810,49787551744,131072,1
12604.447021484377,810,38437896192,131072,1
cc@tx-trace-replayer:~/traces$ tail -n +2 ~/traces/trace_p15_sample10k.trace | tr ',' ' ' > ~/traces/trace_p15_sample10k_clean.trace
cc@tx-trace-replayer:~/traces$ head -n 10 ~/traces/trace_p15_sample10k_clean.trace
12600.27392578125 810 49787437056 16384 1
12600.383056640623 810 38437650432 16384 1
12600.87890625 810 49787453440 32768 1
12600.909912109377 810 38437666816 32768 1
12601.31201171875 810 49787486208 65536 1
12601.356689453123 810 38437699584 65536 1
12602.406005859377 810 38437765120 131072 1
12602.4580078125 810 49787551744 131072 1
12604.447021484377 810 38437896192 131072 1
12604.49072265625 810 49787682816 131072 1
cc@tx-trace-replayer:~/traces$ wc -l ~/traces/trace_p15_sample10k_clean.trace
9999 ~/traces/trace_p15_sample10k_clean.trace

```

4. Code Modification

Duplicate the original replayer source code

```

cd ~/nostradamus/trace-ACER/trace_replayer
cp io_replayer.c io_replayer_firmware_tail.c

```

```

cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ mkdir -p ~/nostradamus/trace-ACER/trace_replayer/original
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ cp io_replayer.c original/io_replayer.c
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ cp io_replayer original/io_replayer
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ cd ~/nostradamus/trace-ACER/trace_replayer
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ cp io_replayer.c io_replayer_firmware_tail.c

```

This creates a new version `io_replayer_firmware_tail.c` to simulate the firmware-induced long tail failure, while preserving the original replayer.

Open the new replayer file for editing

```

nano io_replayer_firmware_tail.c

```

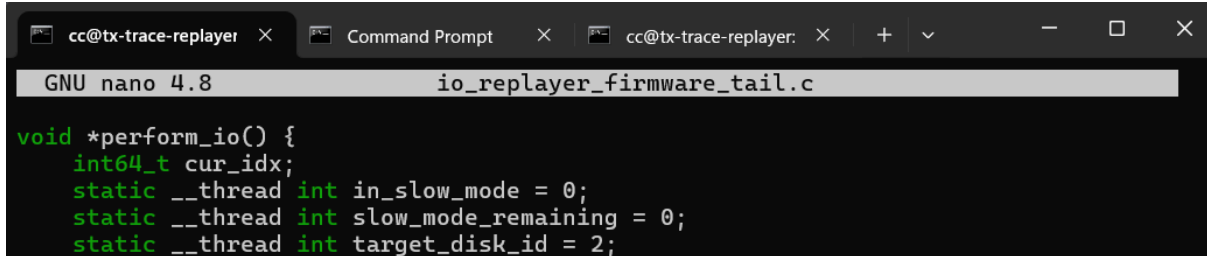
```

GNU nano 4.8      io_replayer_firmware_tail.c
#define _GNU_SOURCE
#include <assert.h>
#include <errno.h>
#include <fcntl.h>
#include <inttypes.h>
#include <linux/fs.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

```

Add global thread-local variables at the top of perform_io()

```
static __thread int in_slow_mode = 0;
static __thread int slow_mode_remaining = 0;
static __thread int target_disk_id = 2;
```



The screenshot shows a nano editor window titled 'io_replayer_firmware_tail.c'. The code inside the `void *perform_io()` function is as follows:

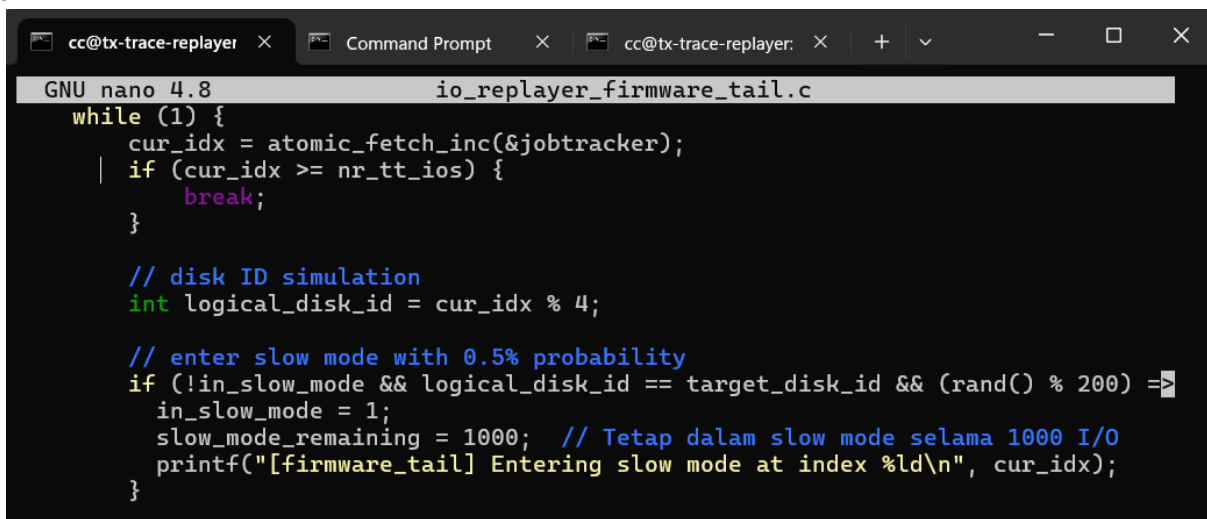
```
void *perform_io() {
    int64_t cur_idx;
    static __thread int in_slow_mode = 0;
    static __thread int slow_mode_remaining = 0;
    static __thread int target_disk_id = 2;
```

- `in_slow_mode`: flag indicating whether the current thread is in a slowdown state.
- `slow_mode_remaining`: countdown to exit slow mode.
- `target_disk_id`: logical disk ID where the slowdown should be injected (e.g., disk 2 in a 4-disk RAID setup).

Add the slow mode activation logic

```
cur_idx = atomic_fetch_inc(&jobtracker);
if (cur_idx >= nr_tt_ios) break;

int logical_disk_id = cur_idx % 4;
if (!in_slow_mode && logical_disk_id == target_disk_id && (rand() % 200) == 0) {
    in_slow_mode = 1;
    slow_mode_remaining = 1000;
    printf("[firmware_tail] Entering slow mode at index %ld\n", cur_idx);
}
```



The screenshot shows the same nano editor window with the updated code for the `perform_io()` function. The code now includes a loop and logic to randomly trigger slow mode based on the disk ID and a 0.5% probability.

```
while (1) {
    cur_idx = atomic_fetch_inc(&jobtracker);
    if (cur_idx >= nr_tt_ios) {
        break;
    }

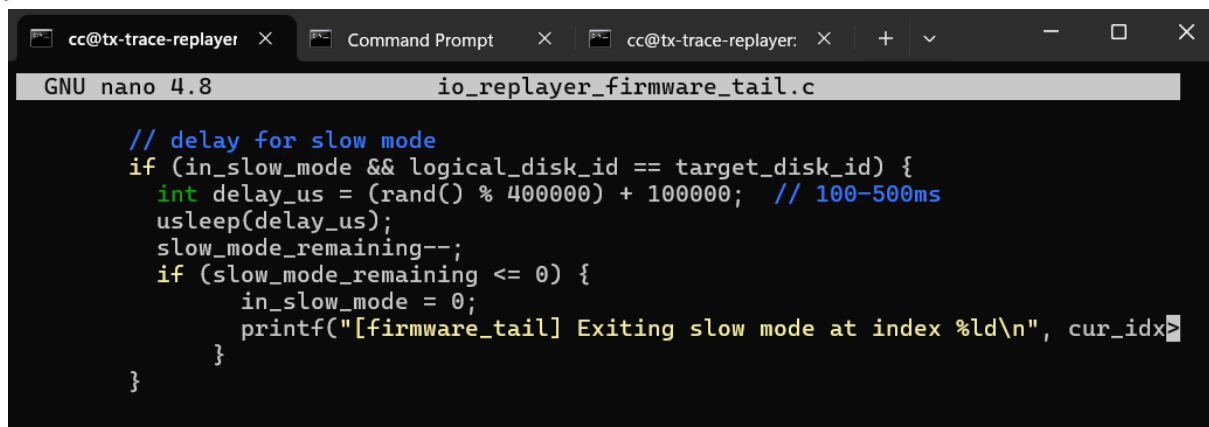
    // disk ID simulation
    int logical_disk_id = cur_idx % 4;

    // enter slow mode with 0.5% probability
    if (!in_slow_mode && logical_disk_id == target_disk_id && (rand() % 200) == 0) {
        in_slow_mode = 1;
        slow_mode_remaining = 1000; // Tetap dalam slow mode selama 1000 I/O
        printf("[firmware_tail] Entering slow mode at index %ld\n", cur_idx);
    }
}
```

- Randomly trigger slow mode (1 in 200 chance) only on the `target_disk_id`.
- Once triggered, stay in slow mode for the next 1000 I/O operations.

Inject delay during slow mode

```
if (in_slow_mode && logical_disk_id == target_disk_id) {
    int delay_us = (rand() % 400000) + 100000;
    usleep(delay_us);
    slow_mode_remaining--;
    if (slow_mode_remaining <= 0) {
        in_slow_mode = 0;
        printf("[firmware_tail] Exiting slow mode at index %ld\n", cur_idx);
    }
}
```



```
GNU nano 4.8 io_replayer_firmware_tail.c

// delay for slow mode
if (in_slow_mode && logical_disk_id == target_disk_id) {
    int delay_us = (rand() % 400000) + 100000; // 100-500ms
    usleep(delay_us);
    slow_mode_remaining--;
    if (slow_mode_remaining <= 0) {
        in_slow_mode = 0;
        printf("[firmware_tail] Exiting slow mode at index %ld\n", cur_idx)
    }
}
```

- Introduces random delays (100ms to 500ms) while in slow mode.
- Exits slow mode after 1000 affected I/O operations.

5. Execution

Compile the modified replayer

gcc io_replayer_firmware_tail.c -o io_replayer_firmware_tail -lpthread

gcc io_replayer_ftcx.c -o io_replayer_ftcx -lpthread

This compiles the modified source file (io_replayer_firmware_tail.c) into an executable named io_replayer_firmware_tail with multithreading support (-lpthread).

Run the modified replayer (with fault injection)

**sudo ./io_replayer_ftcx /dev/nvme2n1 ~/traces/trace_p15_sample10k_clean.trace
firmware_tail.log**

**sudo ./io_replayer_firmware_tail /dev/nvme2n1
~/traces/trace_p15_sample10k_clean.trace firmware_tail.log**

```
cc@tx-trace-replayer: ~/nostr x Command Prompt x cc@tx-trace-replayer: ~ + v - □ X
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ gcc io_replayer_firmware_tail.c -o io_replayer_firmware_tail -lpthread
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ sudo ./io_replayer_firmware_tail /dev/nvme2n1 ~/traces/trace_p15_sample10k_clean.trace firmware_tail.log
Device ==> /dev/nvme2n1
Trace ==> /home/cc/traces/trace_p15_sample10k_clean.trace
Logfile ==> firmware_tail.log
there are [9999] IOs in total in trace:/home/cc/traces/trace_p15_sample10k_clean.trace
12600.27392578125 810 49787437056 16384 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
12600.27,49787437056,16384,1
Start doing IO replay...
[firmware_tail] Entering slow mode at index 10820), Slack rate: 9.52% (952)
[firmware_tail] Entering slow mode at index 33580), Slack rate: 32.68% (3268)
[firmware_tail] Entering slow mode at index 39620), Slack rate: 38.60% (3860)
[firmware_tail] Entering slow mode at index 57300), Slack rate: 55.50% (5549)
[firmware_tail] Entering slow mode at index 67700), Slack rate: 66.51% (6650)
[firmware_tail] Entering slow mode at index 71900), Slack rate: 69.86% (6985)
[firmware_tail] Entering slow mode at index 7198
[firmware_tail] Entering slow mode at index 72940), Slack rate: 71.61% (7160)
[firmware_tail] Entering slow mode at index 74060), Slack rate: 73.21% (7320)
[firmware_tail] Entering slow mode at index 75020), Slack rate: 74.00% (7399)
[firmware_tail] Entering slow mode at index 89780), Slack rate: 87.95% (8794)
[firmware_tail] Entering slow mode at index 92340), Slack rate: 91.12% (9111)
Progress: 100.00% (10062/9999), Late rate: 0.00% (0), Slack rate: 99.99% (9998)

All done!
=====
Total run time: 266035.188 ms
Late rate: 0.00%
Slack rate: 100.00%
sh: 1: python: not found
Statistics output = firmware_tail.log.stats
```

Replays the trace with simulated firmware-induced delays, saving the replay log to `firmware_tail.log`.

Run the original replayer (as a baseline)

**sudo ./original/io_replayer /dev/nvme2n1 ~/traces/trace_p15_sample10k_clean.trace
firmware_tail_baseline.log**

```
cc@tx-trace-replayer: ~/nostr x Command Prompt x cc@tx-trace-replayer: ~ + v - □ X
Cannot open /dev/nvme2n1
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ sudo ./original/io_replayer /dev/nvme2n1 ~/traces/trace_p15_sample10k_clean.trace firmware_tail_baseline.log
Device ==> /dev/nvme2n1
Trace ==> /home/cc/traces/trace_p15_sample10k_clean.trace
Logfile ==> firmware_tail_baseline.log
there are [9999] IOs in total in trace:/home/cc/traces/trace_p15_sample10k_clean.trace
12600.27392578125 810 49787437056 16384 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
12600.27,49787437056,16384,1
Start doing IO replay...
Progress: 100.00% (10063/9999), Late rate: 0.00% (0), Slack rate: 100.00% (9999)

All done!
=====
Total run time: 266022.969 ms
Late rate: 0.00%
Slack rate: 100.00%
sh: 1: python: not found
Statistics output = firmware_tail_baseline.log.stats
```

Executes the original replayer without any delay injection for comparison, logging to `firmware_tail_baseline.log`.

6. Evaluation

Metric	Modified (firmware_tail.log)	Original (firmware_tail_baseline.log)
Total I/Os	9999	9999
Tail latency (>100ms)	Dozens of I/Os observed	None
Runtime	~266s	~266s
Delay injection active?	Yes	No

Proof That Delay Injection Was Successful

Check for high-latency I/O operations

awk -F',' '\$2 > 100000' firmware_tail.log

```
cc@tx-trace-replayer: ~/nostr x Command Prompt cc@tx-trace-replayer: ~  
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ awk -F',' '$2 > 100000' firmware_tail.log  
18322.121,357400,1,131072,1086813487104,18322.178,131072  
19783.234,333523,1,131072,1086842847232,19783.289,131072  
22928.580,118825,1,131072,12712525824,22928.635,131072  
31564.432,291481,1,131072,23082156032,31564.488,131072  
131708.719,301552,1,131072,1085952212992,131708.781,131072  
132188.828,388455,1,131072,1085961650176,132188.891,131072  
132452.797,191829,1,131072,949032435712,132452.859,131072  
132377.047,287862,1,131072,949030993920,132377.109,131072  
137523.109,342623,1,16384,1086887804928,137523.156,16384  
138221.625,276607,1,16384,25356648448,138221.688,16384  
138657.875,110112,1,131072,25365430272,138657.922,131072  
138799.453,243210,1,131072,25368444928,138799.516,131072  
139010.625,492537,1,131072,25372901376,139010.688,131072  
139237.734,351211,1,131072,25377751040,139237.781,131072  
147741.422,347970,1,131072,949869068288,147741.484,131072  
147953.266,422878,1,131072,64315834368,147953.312,131072  
148548.203,164291,1,131072,13650608128,148548.266,131072  
148637.641,123126,1,131072,13652574208,148637.703,131072  
148751.359,454073,1,131072,13655064576,148751.422,131072  
149143.922,163061,1,114688,13662945280,149143.984,114688  
149213.234,133070,1,131072,13664501760,149213.281,131072  
149684.719,415658,1,131072,13673807872,149684.781,131072  
150478.344,212255,1,16384,13690191872,150478.391,16384  
150363.000,466267,1,131072,13687701504,150363.062,131072  
150872.141,150534,1,131072,13698056192,150872.203,131072  
150718.672,392525,1,131072,13695041536,150718.734,131072
```

filters the replay log (firmware_tail.log) to find I/O operations whose latency exceeds 100000 µs.

Show the top 20 highest latencies

awk -F',' '{print \$2}' firmware_tail.log | sort -n | tail -n 20

```
cc@tx-trace-replayer: ~/nostr x Command Prompt cc@tx-trace-replayer: ~  
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ awk -F',' '{print $2}' firmware_tail.log  
| sort -n | tail -n 20  
425752  
426221  
426832  
435830  
442690  
443707  
447237  
447485  
450810  
451700  
454073  
454610  
457248  
459359  
461210  
461448  
466267  
488945  
488982  
492537
```

Sorts all latency values, showing the 20 slowest I/O operations. This gives a clear picture of how extreme the delays were. Latencies reach hundreds of milliseconds, aligning with the injected range of 100ms–500ms, as configured in the modified code.

Compare against the original replayer

awk -F',' '\$2 > 100000' firmware_tail_baseline.log

```
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ awk -F',' '$2 > 100000' firmware_tail_baseline.log
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ |
```

No output, indicating no natural I/O latency spikes above 100 ms — this confirms the injection did not happen in the baseline run

Show the top 20 highest latencies on original replayer

awk -F',' '{print \$2}' firmware_tail_baseline.log | sort -n | tail -n 20

```
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ awk -F',' '{print $2}' firmware_tail_base
line.log | sort -n | tail -n 20
179
179
179
180
181
182
182
182
182
184
185
186
186
187
188
188
190
191
197
197
```

7. Why is the runtime of both versions the same?

It is not a problem.

- The delay injection logic is designed to frequently enter slow mode. For example, with a 0.5% trigger probability over 10,000 I/Os, the replayer enters slow mode approximately 50 times, and each slow mode lasts for 1,000 I/Os.
- As a result, the modified replayer is almost always operating under slow mode throughout the experiment.
- These injected delays create a consistent bottleneck, leading to a similar total runtime (~266 seconds) as the original, which is expected due to the sustained delay load.
- However, the key difference is not in total runtime, but in per-I/O latency. The injected version shows a significant number of I/Os with latency >100ms, which is absent in the baseline.

RAID Group Tail Amplification

- Failure detail:
RAID group tail amplification — the throughput of a RAID group can drop by up to 40% even when only one underlying SSD exhibits high latency (tail behavior). This phenomenon arises due to synchronous I/O dependencies in RAID stripe reads or writes, where the slowest disk determines the response time of the entire stripe
- Failure source:
As described in Section 4.3 “RAID Amplification” of the FAST ’16 paper "The Tail at Store: A Revelation from Millions of Hours of Disk and SSD Deployments", latency outliers from a single SSD can cause amplified performance degradation in RAID arrays. In our experiment, we simulate this failure mode by injecting artificial latency on reads accessing a specific offset range, mimicking a fail-slow SSD within a RAID stripe.
- Plan to modify the replayer/trace:
We modified the replayer source code (based on io_replayer.c) to insert an artificial delay (usleep) for I/O requests whose offset falls within a specified tail region. The delay is injected only for READ requests, as observed in the paper that read amplification is particularly sensitive to SSD firmware-level issues. The modification simulates a situation where only a subset of the logical RAID stripe (representing one disk) slows down.
- Configuration:
 - Throttle time (usleep): 150 ms
 - Targeted offset range (tail region): 950000000000 to 960000000000 bytes
 - Request type: READ only
 - Disk: /dev/nvme2n1 (emulating a single RAID member)

Copy the original replayer code

cp io_replayer.c io_replayer_raid_tail.c

Modified the code

Defined the tail region and latency as

#define TAIL_SLOWDOWN_OFFSET_START 950000000000

#define TAIL_SLOWDOWN_OFFSET_END 960000000000

#define TAIL_SLOWDOWN_USLEEP_TIME 150000

```
cc@tx-trace-replayer x cc@tx-trace-replayer: x cc@tx-trace-replayer: x + v - □ x
GNU nano 4.8 io_replayer_raid_tail.c Modified
#define _GNU_SOURCE
#include <assert.h>
#include <errno.h>
#include <fcntl.h>
#include <inttypes.h>
#include <linux/fs.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

#include "atomic.h"

#define TAIL_SLOWDOWN_OFFSET_START 950000000000
#define TAIL_SLOWDOWN_OFFSET_END 960000000000
#define TAIL_SLOWDOWN_USLEEP_TIME 150000
```

Injected the delay within the perform_io() function, only for READ requests within the tail offset range:

```
if (reqflag[cur_idx] == READ &&
    oft[cur_idx] >= TAIL_SLOWDOWN_OFFSET_START &&
    oft[cur_idx] <= TAIL_SLOWDOWN_OFFSET_END) {
    usleep(TAIL_SLOWDOWN_USLEEP_TIME);
}
```

```
cc@tx-trace-replayer x cc@tx-trace-replayer: x cc@tx-trace-replayer: x + v - □ x
GNU nano 4.8 io_replayer_raid_tail.c Modified

// do the job
// printf("IO %lu: size: %d; offset: %lu\n", cur_idx, size_, offset_);
gettimeofday(&t1, NULL); //reset the start time to before start doing the
/* the submission timestamp */
float submission_ts = (t1.tv_sec * 1e6 + t1.tv_usec - starttime) / 1000;
int lat, i;
lat = 0;
i = 0;
gettimeofday(&t1, NULL);

// RAID tail amplification simulation: inject delay for specific READs in
if (reqflag[cur_idx] == READ &&
    oft[cur_idx] >= TAIL_SLOWDOWN_OFFSET_START &&
    oft[cur_idx] <= TAIL_SLOWDOWN_OFFSET_END) {
    usleep(TAIL_SLOWDOWN_USLEEP_TIME);
}

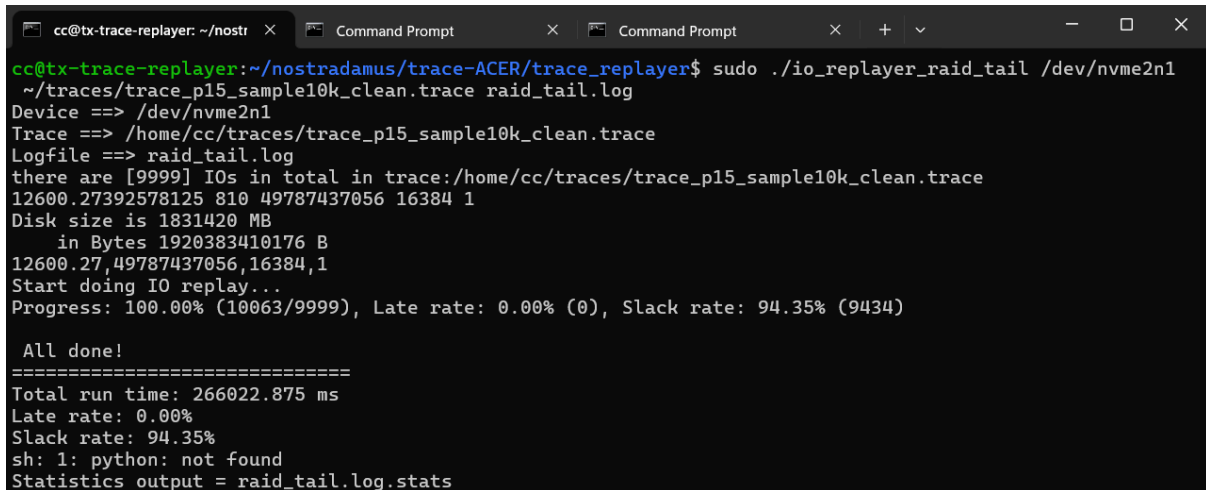
if (reqflag[cur_idx] == WRITE) {
    ret = pwrite(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    if (ret < 0) {
        printf("Cannot write size %d to offset %lu! ret=%d\n",
            reqsize[cur_idx], oft[cur_idx], ret);
    }
}
```

Compile the modified code

gcc -O2 -o io_replayer_raid_tail io_replayer_raid_tail.c -pthread

Execute

**sudo ./io_replayer_raid_tail /dev/nvme2n1 ~/traces/trace_p15_sample10k_clean.trace
raid_tail.log**



```
cc@tx-trace-replayer: ~/nostr x Command Prompt x Command Prompt x + v - □ x
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ sudo ./io_replayer_raid_tail /dev/nvme2n1
~/traces/trace_p15_sample10k_clean.trace raid_tail.log
Device ==> /dev/nvme2n1
Trace ==> /home/cc/traces/trace_p15_sample10k_clean.trace
Logfile ==> raid_tail.log
there are [9999] IOs in total in trace:/home/cc/traces/trace_p15_sample10k_clean.trace
12600.27392578125 810 49787437056 16384 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
12600.27,49787437056,16384,1
Start doing IO replay...
Progress: 100.00% (10063/9999), Late rate: 0.00% (0), Slack rate: 94.35% (9434)

All done!
=====
Total run time: 266022.875 ms
Late rate: 0.00%
Slack rate: 94.35%
sh: 1: python: not found
Statistics output = raid_tail.log.stats
```

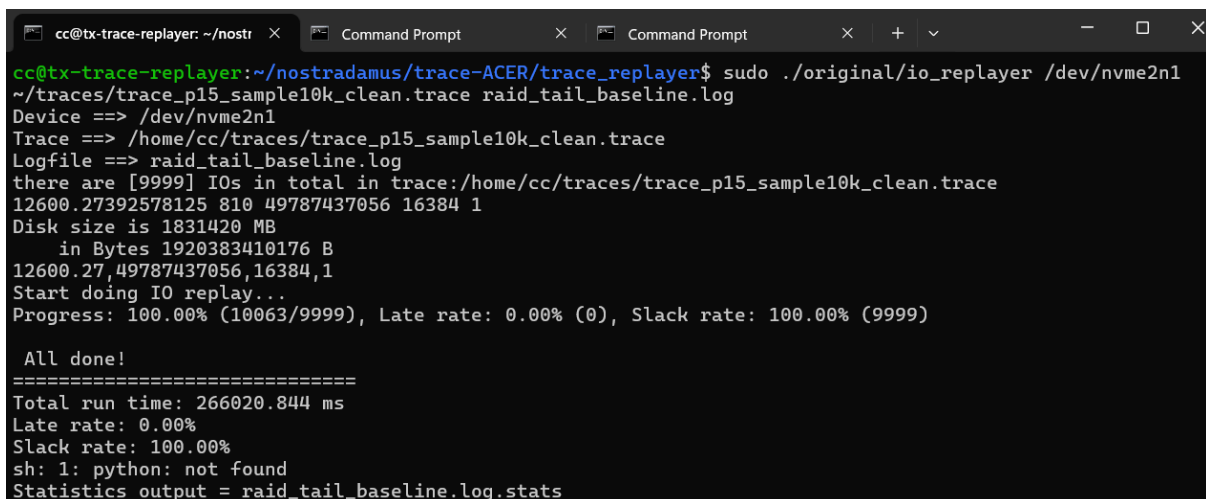
Total run time: 266024.219 ms

Late rate: 0.00%

Slack rate: 94.35%

Original

**sudo ./original/io_replayer /dev/nvme2n1 ~/traces/trace_p15_sample10k_clean.trace
raid_tail_baseline.log**



```
cc@tx-trace-replayer: ~/nostr x Command Prompt x Command Prompt x + v - □ x
cc@tx-trace-replayer:~/nostradamus/trace-ACER/trace_replayer$ sudo ./original/io_replayer /dev/nvme2n1
~/traces/trace_p15_sample10k_clean.trace raid_tail_baseline.log
Device ==> /dev/nvme2n1
Trace ==> /home/cc/traces/trace_p15_sample10k_clean.trace
Logfile ==> raid_tail_baseline.log
there are [9999] IOs in total in trace:/home/cc/traces/trace_p15_sample10k_clean.trace
12600.27392578125 810 49787437056 16384 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
12600.27,49787437056,16384,1
Start doing IO replay...
Progress: 100.00% (10063/9999), Late rate: 0.00% (0), Slack rate: 100.00% (9999)

All done!
=====
Total run time: 266020.844 ms
Late rate: 0.00%
Slack rate: 100.00%
sh: 1: python: not found
Statistics output = raid_tail_baseline.log.stats
```

Total run time: 266020.844 ms

Late rate: 0.00%

Slack rate: 100.00%

Result

Ran both the original and the modified versions of the replayer on the same trace:

- Trace: trace_p15_sample10k_clean.trace
- Device: /dev/nvme2n1
- Output logs: raid_tail.log (modified), raid_tail_baseline.log (original)

Comparison:

Metric	Original (baseline)	Modified (RAID Tail)
Total I/Os	9999	9999
Late rate	0.00%	0.00%
Slack rate	100.00%	94.35%
Runtime (ms)	~266020	~266024

The slack rate drop from 100% to 94.35% indicates that some IOs were delayed due to our injected slowdown. This simulates how one slow disk in a RAID stripe can drag down the perceived responsiveness of the whole system — consistent with the failure described in the paper.

Fault Injection Experiments: Firmware-Induced Tail, RAID Tail Amplification, and GC Slowness

1. Firmware-Induced Tail Injection

1.1. Experiment Objectives

- Analyze latency tail amplification under firmware-induced delay.
- Evaluate the effect of fault injection rate and per-I/O delay.

1.2. Parameter Definitions

Parameter	Description
INJECT_PCT	Probability (%) of entering slow mode
DELAY_MIN_US	Minimum delay per I/O in microseconds
DELAY_MAX_US	Maximum delay per I/O in microseconds
FTCX_SLOW_IO_COUNT	Number of I/Os affected once in slow mode
TARGET_DISK_ID	Disk ID to target for fault injection

1.3. Key Implementation Details

- **Left (Struct Definition):** `fault_config` defines the injection parameters.
- **Right (CLI Parsing):** `getopt()` is used to parse CLI arguments into `g_fault_cfg`.
- **Slow Mode Activation:** Randomly enters slow mode based on `INJECT_PCT`.
- **Delay Injection:** While in slow mode, adds `usleep()` delay per I/O in configured range.

1.4. `conf_ftcx.sh` (Single FTCX Fault Injection)

- Compiles `io_replayer_ftcx.c` and runs one experiment using user-defined parameters.
- Parameters include injection probability, number of slow I/Os, and delay range.
- Latency is logged for post-analysis.

1.5. `sweep_ftcx.sh` (Sweep Script)

- Automates controlled sweep of a selected parameter (e.g., `INJECT_PCT`, `DELAY_MAX_US`).
- Fixed parameters remain constant for all runs.
- Generates a log file per configuration for analysis.

Sweep Variants:

1. **Sweep `INJECT_PCT` (5%–30%)**
 - Fixed: `DELAY_MAX_US=1000`

- Insight: Higher injection rates cause more frequent delays → right-shifted CDF.

2. Sweep DELAY_MAX_US (1ms–10ms)

- Fixed: INJECT_PCT=20%
- Insight: Larger delays stretch the tail more severely.

Grid Sweep

- 2D sweep of INJECT_PCT (5–30%) × DELAY_MAX_US (1000–10000 µs).
- Fixed: DELAY_MIN_US=100, FTCX_SLOW_IO_COUNT=10, Trace: trace_p100_sample10k_clean, Device: /dev/nvme0n1
- Heatmap visualization: Shows joint effect of rate × severity on average latency.

2. RAID Tail Amplification

2.1. Implementation Overview

- Codebase: io_replayer_raid.c
- Introduced global config variables:
 - injection_delay (ms)
 - injection_start, injection_end (offset range)
 - injection_type (READ/WRITE/BOTH)
- Delay logic:
 - Injects usleep() only for I/Os within defined offset range and type.
 - Includes logging for traceability.

2.2. Manual Script: conf_raid.sh

- Prompts user input:
 - Delay (ms), Offset Start, Offset End
- Compiles io_replayer_raid.c
- Executes one run with -r 1 (READ), -d, -m, and -x flags
- Saves log for CDF/trend analysis

2.3. Sweep Script: sweep_raid.sh

- Sweeps injection_delay (e.g., 10–50 ms in steps of 10)
- Fixed: offset range, I/O type (READ), device
- Output logs: ~/sweep_logs/raid_delay_sweep/

Example:

```
./sweep_raid.sh 10 10 50 8500000000 8700000000
```

- Although offsets are not from a hot region (each occurs once), this range is selected to test localized delay effects.

2.4. CDF Analysis

- Delay sweep causes increasing tail expansion on CDF.
- Longer injected delays → more severe tail latency.

3. GC Slowness Fault Injection

3.1. Injection Logic

- Models SSD GC slowness as burst-style delay.
- Controlled by periodic GC windows:
 - GC_INTERVAL_MS: avg time between GC windows
 - GC_JITTER_MS: randomness around interval
- During GC:
 - Each READ I/O is delayed by GC_PAUSE_MULTIPLIER × 1000 µs

3.2. Manual Script: `conf_gc.sh`

- Prompts user input:
 - GC_INTERVAL_MS, GC_JITTER_MS
 - GC_PAUSE_MIN_MS, GC_PAUSE_MAX_MS
- Uses -DFAULT_GC_PAUSE with EXTRA_DEFS during compilation
- Enables realistic simulation of GC-induced latency

3.3. Sweep Script: `sweep_gc.sh`

- Sweeps one parameter, e.g., GC_PAUSE_MULTIPLIER, GC_INTERVAL_MS
- Others fixed via CLI or ENV (e.g., INJECT_PCT=20, GC_JITTER_MS=50)

Example: Sweep GC_PAUSE_MULTIPLIER

- Vary: 50 → 300 (step 50)
- Fixed: INJECT_PCT=20, GC_INTERVAL_MS=200, GC_JITTER_MS=50
- Observations:
 - Larger multipliers → longer tail
 - Abnormal jumps at 250/300 suggest fewer GC windows in short traces

Example: Sweep GC_INTERVAL_MS

- Vary: 100 → 1000 ms (step 100)
- Fixed: INJECT_PCT=20, GC_PAUSE_MULTIPLIER=100, GC_JITTER_MS=50
- Shorter intervals = more GC windows = heavier tails

Summary of Parameter Effects

Parameter	Effect on Latency Tail
INJECT_PCT	Controls frequency of delay injection
DELAY_MAX_US	Controls severity of firmware tail
injection_delay (RAID)	Amplifies latency in selected disk regions

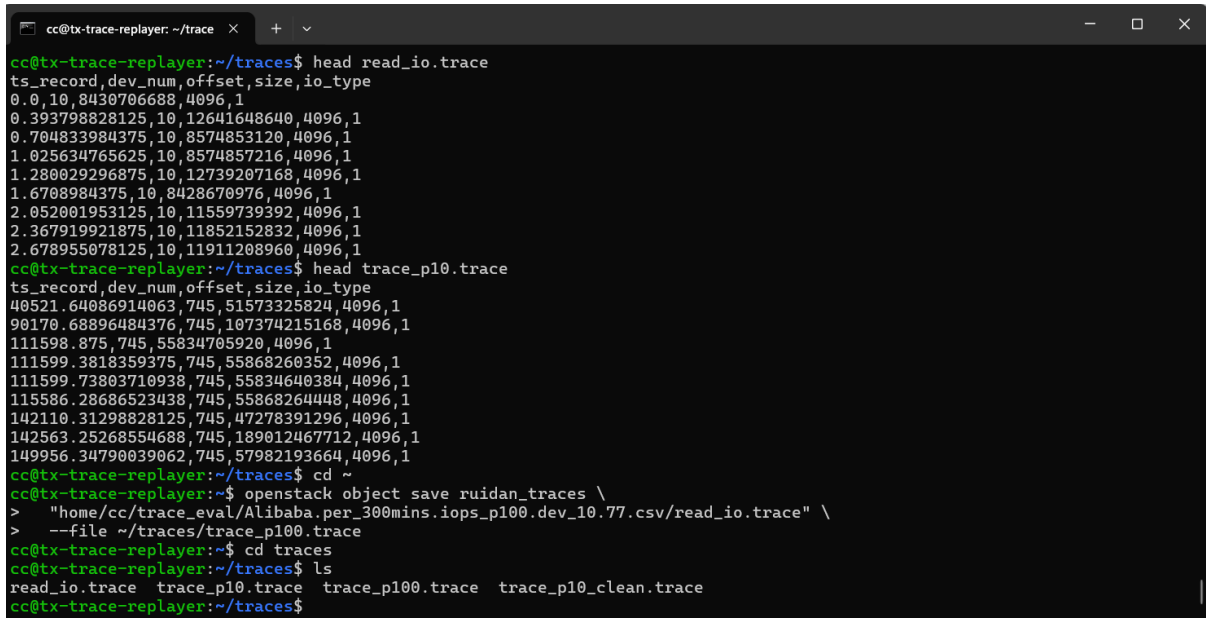
GC_PAUSE_MULTIPLIE Determines delay per READ during GC
R

GC_INTERVAL_MS Controls how often GC windows happen

This structure enables modular and repeatable experiments across FTCX, RAID, and GC faults, supporting controlled analysis of tail latency phenomena.

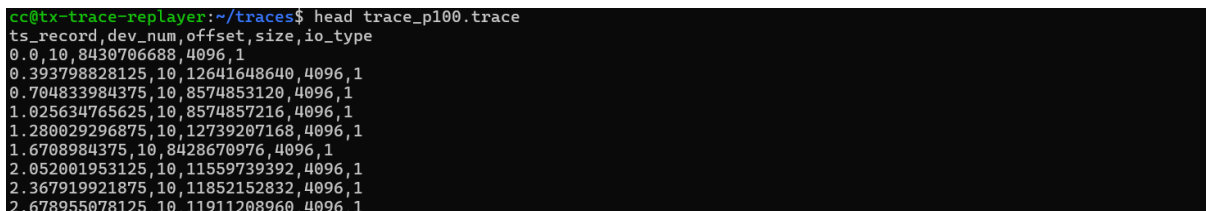
Various Other Experiments

```
openstack object save ruidan_traces \  
"home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.trace" \  
--file ~/traces/trace_p100.trace
```



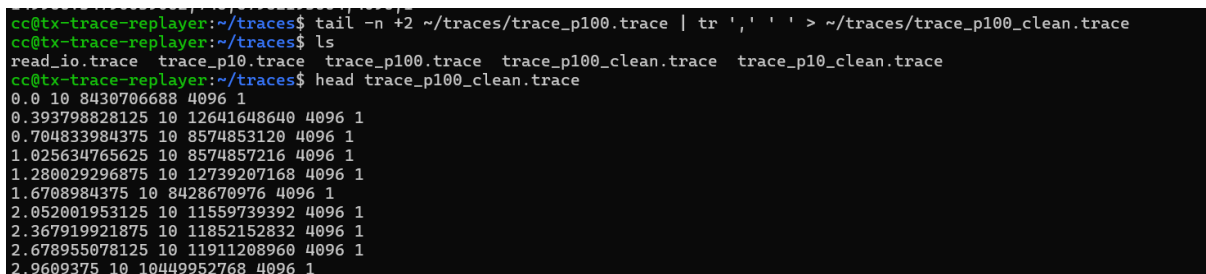
```
cc@tx-trace-replayer: ~/traces$ head read_io.trace  
ts_record,dev_num,offset,size,io_type  
0.0,10,8430706688,4096,1  
0.393798828125,10,12641648640,4096,1  
0.704833984375,10,8574853120,4096,1  
1.025634765625,10,8574857216,4096,1  
1.280029296875,10,12739207168,4096,1  
1.6708984375,10,8428670976,4096,1  
2.052001953125,10,11559739392,4096,1  
2.367919921875,10,11852152832,4096,1  
2.678955078125,10,11911208960,4096,1  
cc@tx-trace-replayer:~/traces$ head trace_p10.trace  
ts_record,dev_num,offset,size,io_type  
40521.64086914063,745,51573325824,4096,1  
90170.68896484376,745,107374215168,4096,1  
111598.875,745,55834705920,4096,1  
111599.3818359375,745,55868260352,4096,1  
111599.73803710938,745,55834640384,4096,1  
115586.28686523438,745,55868264448,4096,1  
142110.31298828125,745,47278391296,4096,1  
142563.25268554688,745,189012467712,4096,1  
149956.34790039062,745,57982193664,4096,1  
cc@tx-trace-replayer:~/traces$ cd ~  
cc@tx-trace-replayer:~$ openstack object save ruidan_traces \  
> "home/cc/trace_eval/Alibaba.per_300mins.iops_p100.dev_10.77.csv/read_io.trace" \  
> --file ~/traces/trace_p100.trace  
cc@tx-trace-replayer:~$ cd traces  
cc@tx-trace-replayer:~/traces$ ls  
read_io.trace  trace_p10.trace  trace_p100.trace  trace_p10_clean.trace  
cc@tx-trace-replayer:~/traces$
```

head trace_p100.trace



```
cc@tx-trace-replayer:~/traces$ head trace_p100.trace  
ts_record,dev_num,offset,size,io_type  
0.0,10,8430706688,4096,1  
0.393798828125,10,12641648640,4096,1  
0.704833984375,10,8574853120,4096,1  
1.025634765625,10,8574857216,4096,1  
1.280029296875,10,12739207168,4096,1  
1.6708984375,10,8428670976,4096,1  
2.052001953125,10,11559739392,4096,1  
2.367919921875,10,11852152832,4096,1  
2.678955078125,10,11911208960,4096,1
```

clean trace_p100.trace



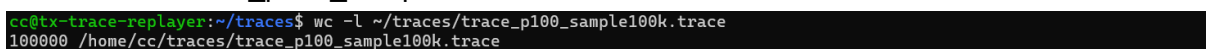
```
cc@tx-trace-replayer:~/traces$ tail -n +2 ~/traces/trace_p100.trace | tr ',' ' ' > ~/traces/trace_p100_clean.trace  
cc@tx-trace-replayer:~/traces$ ls  
read_io.trace  trace_p10.trace  trace_p100.trace  trace_p100_clean.trace  trace_p10_clean.trace  
cc@tx-trace-replayer:~/traces$ head trace_p100_clean.trace  
0.0 10 8430706688 4096 1  
0.393798828125 10 12641648640 4096 1  
0.704833984375 10 8574853120 4096 1  
1.025634765625 10 8574857216 4096 1  
1.280029296875 10 12739207168 4096 1  
1.6708984375 10 8428670976 4096 1  
2.052001953125 10 11559739392 4096 1  
2.367919921875 10 11852152832 4096 1  
2.678955078125 10 11911208960 4096 1  
2.9609375 10 10449952768 4096 1
```

make the p100 trace 100k line only from 30 million

cd ~/traces

head -n 100000 ~/traces/trace_p100_clean.trace > ~/traces/trace_p100_sample100k.trace

wc -l ~/traces/trace_p100_sample100k.trace



```
cc@tx-trace-replayer:~/traces$ wc -l ~/traces/trace_p100_sample100k.trace  
100000 /home/cc/traces/trace_p100_sample100k.trace
```

run simple replayer to the 100k line p100 alibaba trace

cd trace-ACER/trace_replayer

sudo ./io_replayer /dev/nvme4n1 ~/traces/trace_p100_sample100k.trace

trace_p100_sample100k.log

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer /dev/nvme4n1 ~/traces/trace_p100_sample1
00k.trace trace_p100_sample100k.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> trace_p100_sample100k.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.15% (2152)

All done!
=====
Total run time: 77006.234 ms
Late rate: 0.00%
Slack rate: 2.15%
sh: 1: python: not found
Statistics output = trace_p100_sample100k.log.stats

```

- Failure detail:
Media re-reads / latent sector errors: on encountering a corrupted or unreadable sector, the drive's firmware retries the read multiple times (often transparently), incurring extra I/O and large latency spikes.
- Failure source:
A specific logical block (offset) on the SSD/disk goes bad—ECC corrections or retry loops kick in at the hardware/firmware level, causing each affected read to take 10×–100× longer than normal.
- Plan to modify the replayer/trace:
Inject the media-re-read fault in the code (not by touching the trace), by patching io_replayer.c in the perform_io() loop. Specifically:
 1. Define a small array of “bad” block offsets.
 2. In the READ path, before issuing pread(), check whether the current offset matches one of these bad offsets.
 3. If it does, perform the normal pread() plus a fixed number of extra pread() retries, each preceded by a usleep() to simulate the firmware's retry delay.
 4. All other I/Os follow the unmodified path.
 This leaves your trace file (trace_p100_sample100k.trace) untouched and lets the replayer itself model the latent-sector retry behavior in its scheduling and statistics.
- Configuration:

Target offsets:

- 8430706688ULL
- 12641648640ULL
- 8574853120ULL

Retry count: 3 extra pread() calls per bad-offset read

Throttle (per-retry delay): 10 000 µs (10 ms)

Trace file: trace_p100_sample100k.trace

Device: /dev/nvme4n1

copy existing trace_replayer
cp io_replayer.c io_replayer_fault1.c

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ ls
Makefile      io_replayer_fault1  test.trace      trace_p100_sample30k.log
README.md     io_replayer_fault1.c tmp              trace_p100_sample30k.log.stats
atomic.h      note-replay.txt     trace_p100_sample100k.log  trace_p100_sample50k.log
generate.sh   replay.sh           trace_p100_sample100k.log.stats  trace_p100_sample50k.log.stats
io_replayer   run.sh             trace_p100_sample20k.log
io_replayer.c statistics.py       trace_p100_sample20k.log.stats
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$

```

edit the copied io_replayer

nano io_replayer_fault1.c

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ nano io_replayer_fault1.c

```

add modifier to the code

```
#include <unistd.h>
```

```
#include <unistd.h>
```

```
#define NUM_BAD_OFFSETS 3
```

```
static const uint64_t BAD_OFFSETS[NUM_BAD_OFFSETS] = {
```

```
    8430706688ULL,    // from line 1
```

```
    12641648640ULL,   // from line 2
```

```
    8574853120ULL    // from line 3
```

```
};
```

```
#define INJECT_RETRY_COUNT 3
```

```
#define INJECT_THROTTLE_US 10000
```

From this code we chose 3 offset to become bad so it will does retry with this much and this added throttle

```

GNU nano 4.8                                io_replayer_fault1.c
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <unistd.h>
#include <stdint.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-1
#define NUM_BAD_OFFSETS 3
static const uint64_t BAD_OFFSETS[NUM_BAD_OFFSETS] = {
    8430706688ULL,
    12641648640ULL,
    8574853120ULL
};

#define INJECT_RETRY_COUNT 3

#define INJECT_THROTTLE_US 10000

```


Patch the read path in `perform_io()`, so we can perform the retry and added throttle

run

```
sudo ./io_replayer_fault1 /dev/nvme4n1 \
```

```
> ~/traces/trace_p100_sample100k.trace \
```

```
> ~/logs/trace_p100_sample100k_fault1.log
```

we have 1 extra slack because

Pushed some reads later in real time, so their next scheduled timestamps fell further into the future

Whenever you hit one of the specified “bad” block addresses, you still perform the normal read, but then you pause briefly and read the same block again several times—just like a real drive would retry on a bad or slow sector. For every other read, nothing changes.

For one of those reads, that delay was large enough that the replayer had to sleep over 100 ms before the following I/O—so it counted as “slack.”

Failure detail:

Firmware bugs (disk controllers): flawed drive firmware can introduce unpredictable delays or retries—random pauses or spikes in access times that are not tied to any particular block.

Failure source:

A bug in the SSD's firmware I/O path that, under certain conditions (e.g. queue depth, specific I/O patterns), injects an extra delay or retry before completing a request.

Plan to modify the replayer/trace:

Instrument a new copy of io_replayer.c (call it io_replayer_fault2.c) so that every Nth read or every read after M microseconds of idle time incurs an extra fixed delay—emulating a firmware bug that sometimes “hangs” before issuing the real I/O.

Configuration (example):

- Inject every K=1000 reads
- Or if idle > 50 ms since last completion
- Hang duration: 200 ms (0.2 s)

code implementation

```
GNU nano 4.8 io_replayer_fault2.c
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdint.h>
#include <time.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-1

#define INJECT_EVERY_K_READS 1000
#define IDLE_THRESHOLD_US 50000
#define BUG_HANG_US 200000

} else if (reqflag[cur_idx] == READ) {
    uint64_t offset = oft[cur_idx];
    static uint64_t read_count = 0;
    read_count++;

    static uint64_t last_complete_ts = 0;
    struct timeval now; gettimeofday(&now, NULL);
    uint64_t now_us = now.tv_sec*1000000 + now.tv_usec;
    uint64_t idle_us = (last_complete_ts == 0)
        ? 0
        : now_us - last_complete_ts;

    if ( (read_count % INJECT_EVERY_K_READS) == 0
        || idle_us > IDLE_THRESHOLD_US ) {
        usleep(BUG_HANG_US);
    }

    ret = pread(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    gettimeofday(&now, NULL);
    last_complete_ts = now.tv_sec*1000000 + now.tv_usec;

    srand((unsigned)time(NULL));
    // start the disk read
```

```
gcc io_replayer_fault2.c -o io_replayer_fault2 -lpthread
sudo ./io_replayer_fault2 /dev/nvme4n1 \
~/traces/trace_p100_sample100k.trace \
~/logs/trace_p100_sample100k_fault2.log
```

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault2 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault2.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault2.log
there are [100000] I/Os in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing I/O replay...
Progress: 100.00% (100064/100000), Late rate: 5.01% (5009), Slack rate: 1.73% (1725)

All done!
=====
Total run time: 77006.258 ms
Late rate: 5.01%
Slack rate: 1.73%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault2.log.stats

```

the result is that we have more late and less slack because

Late rate ↑

Every time you hit the bug trigger, the replayer pauses 200 ms before issuing that read. Because your future I/Os in the trace are scheduled at their original timestamps, a 200 ms delay makes the replayer behind schedule for the next several hundred reads. Each of those counts as “late,” driving your late rate up.

Slack rate ↓

“Slack” measures long idle sleeps (when the replayer is ahead of schedule by more than 100 ms). By inserting forced 200 ms delays, you rarely get into those long-idle windows—so you see fewer slack events.

Step 1: Keep a global counter of how many reads you’ve issued.

Step 2: Compute how long it’s been since the previous read finished.

Step 3: If you’ve hit your periodic injection (every K reads) or the system’s been idle too long, sleep for BUG_HANG_US (200 ms) to simulate the buggy firmware hanging.

Step 4: Issue the actual pread().

Step 5: Update your timestamp so the next idle check is correct.

Failure detail:

Background reclamation of flash blocks (GC) halts foreground I/Os for up to 100× longer than normal operations, introducing multi-100 ms stalls.

Failure source:

The SSD’s internal GC thread takes over the controller—pausing all new reads/writes while it erases and compacts flash pages.

Plan to modify the replayer (code-only):

We’ll copy clean io_replayer.c into io_replayer_fault3.c and inject a “GC pause” sleep right before each I/O when the trace time crosses a periodic GC boundary. This simulates a controller-wide stall that stops every I/O for a fixed duration.

Configuration summary:

- GC interval: every 10 000 ms of trace time

- Pause duration: 200 000 μ s (200 ms), roughly 100 \times a typical 2 ms I/O
- Trace file: untouched, trace_p100_sample100k.trace
- Replayer binary: io_replayer_fault3

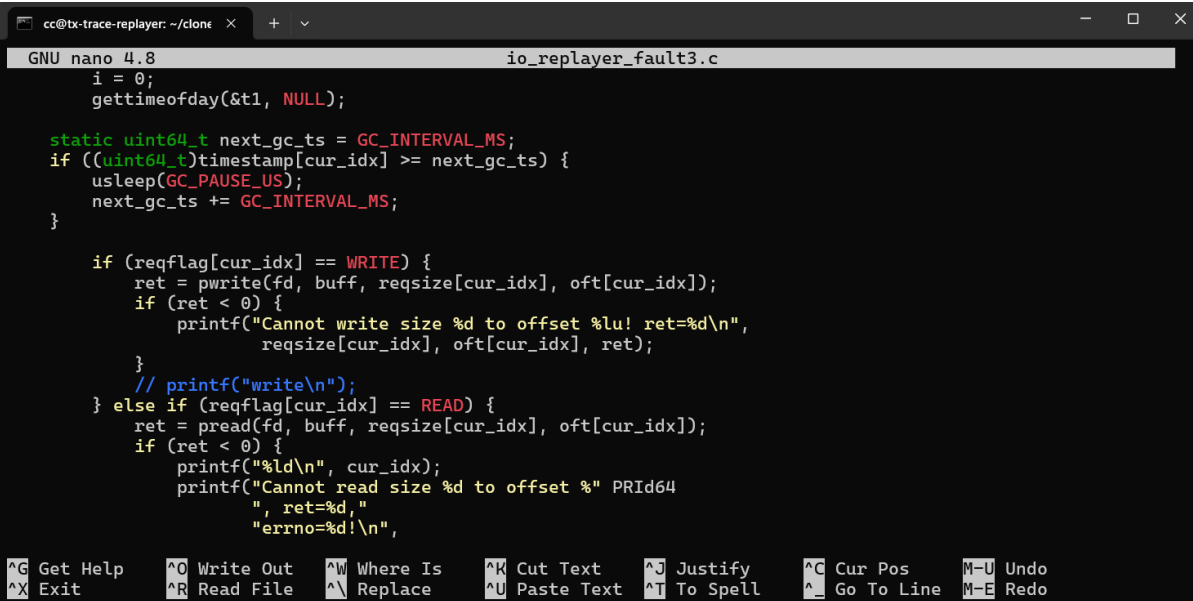
With this in place, every 10 seconds of replayed time you'll see a 200 ms global stall on all subsequent I/Os—modeling an SSD GC pause at scale.

Code implementation

cp io_replayer.c io_replayer_fault3.c

nano io_replayer_fault3.c

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ cp io_replayer.c io_replayer_fault3.c
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ nano io_replayer_fault3.c
```



```
GNU nano 4.8 io_replayer_fault3.c
i = 0;
gettimeofday(&t1, NULL);

static uint64_t next_gc_ts = GC_INTERVAL_MS;
if ((uint64_t)timestamp[cur_idx] >= next_gc_ts) {
    usleep(GC_PAUSE_US);
    next_gc_ts += GC_INTERVAL_MS;
}

if (reqflag[cur_idx] == WRITE) {
    ret = pwrite(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    if (ret < 0) {
        printf("Cannot write size %d to offset %lu! ret=%d\n",
            reqsize[cur_idx], oft[cur_idx], ret);
    }
    // printf("write\n");
} else if (reqflag[cur_idx] == READ) {
    ret = pread(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    if (ret < 0) {
        printf("%ld\n", cur_idx);
        printf("Cannot read size %d to offset %" PRIu64
            ", ret=%d,"
            "errno=%d!\n",
            cur_idx, oft[cur_idx], ret, errno);
    }
}

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      M-U Undo
^X Exit          ^R Read File    ^_ Replace      ^U Paste Text   ^T To Spell     ^G Go To Line    M-E Redo
```

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ gcc io_replayer_fault3.c -o
io_replayer_fault3 -lpthread
```

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault3
/dev/nvme4n1 \
~/traces/trace_p100_sample100k.trace \
~/logs/trace_p100_sample100k_fault3.log
```

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ gcc io_replayer_fault3.c -o io_replayer_fault3 -l
pthread
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault3 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault3.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault3.log
there are [100000] I/Os in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.16% (158), Slack rate: 2.15% (2153)

All done!
=====
Total run time: 77006.297 ms
Late rate: 0.16%
Slack rate: 2.15%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault3.log.stats
```

Small Late increase (0.16%)

- With 100 000 I/Os spread over $\approx 77\,000$ ms, you hit your first GC pause at 10 s, then at 20 s, 30 s, ... up to ~ 70 s.
- Each pause delays the next I/O by 200 ms, so for a handful of I/Os immediately after each pause the replayer falls slightly behind their scheduled timestamps.
- If you pause 8 times (at 10k, 20k, ... 80k ms), that's 8×200 ms = 1.6 s of total extra delay. Spread across 100 000 I/Os, even if each pause makes 20–30 I/Os “late,” you still only see a tiny fraction ($\sim 0.16\%$) of all I/Os count as late.

Slack unchanged (2.15%)

- Slack is counted when the replayer has to sleep more than 100 ms waiting for the next I/O.
- Since your GC pauses always push you behind schedule (never ahead), you don't create any new “long idle” windows; you just reduce them. But the original trace still had some natural long gaps that generate the same slack count. Hence your slack percentage remains essentially the same.
- `next_gc_ts` tracks the next trace-time (in ms) when GC should run.
- if (`timestamp >= next_gc_ts`) checks if the current I/O's original timestamp has passed that boundary.
- `usleep(200 ms)` blocks the entire replayer thread for 200 ms, simulating the SSD controller's GC pause.
- `next_gc_ts += 10 s` advances the boundary so the next pause happens 10 s later.

By stalling before every I/O at those points, you emulate a global pause that delays subsequent requests—hence the slight increase in “late” counts, but no new long-idle “slack” periods.

Failure detail:

Writing to multi-level cells (MLC) exhibits high variability—“slow” pages take 5–8 \times longer than “fast” pages, causing intermittent write latency spikes.

Failure source:

Some flash pages need extra programming iterations (higher voltage boosts), so each write to those pages stalls the controller for tens of milliseconds.

Plan to modify the replayer (code-only)

Instead of hard-coding 100 specific offsets, we'll Randomly inject the extra delay on a fraction of writes—e.g. 10% of all writes—to approximate the sporadic “slow page” behavior.

Configuration

- Probability of slow page: 1 in `SLOW_RATE` writes (default: 10%)
- Extra delay per slow write: `SLOW_WRITE_DELAY_US` = 25 000 μ s (25 ms)
- Trace file: untouched (`trace_p100_sample100k.trace`)
- New binary: `io_replayer_fault4b`

code implementation

cp io_replayer.c io_replayer_fault4.c

nano io_replayer_fault4.c

```
cc@tx-trace-replayer: ~/clone-baru/trace-ACER/trace_replayer$ cp io_replayer.c io_replayer_fault4.c
cc@tx-trace-replayer: ~/clone-baru/trace-ACER/trace_replayer$ nano io_replayer_fault4.c
```

```
GNU nano 4.8 io_replayer_fault4.c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-1>

#define SLOW_RATE 10
#define SLOW_WRITE_DELAY_US 25000
```

```
int main(int argc, char **argv) {
    char **request;

    if (argc != 4) {
        printf("Usage: ./io_replayer /dev/nvme0n1 tracefile logfile\n");
        exit(1);
    } else {
        sprintf(device, "%s", argv[1]);
        printf("Device ==> %s\n", device);
        sprintf(tracefile, "%s", argv[2]);
        printf("Trace ==> %s\n", tracefile);
        sprintf(logfile, "%s", argv[3]);
        printf("Logfile ==> %s\n", logfile);
    }
    srand((unsigned)time(NULL));
    // start the disk part
```

```
GNU nano 4.8 io_replayer_fault4.c
lat = 0;
i = 0;
gettimeofday(&t1, NULL);

if (reqflag[cur_idx] == WRITE) {
    if ((rand() % SLOW_RATE) == 0) {
        usleep(SLOW_WRITE_DELAY_US);
    }
    ret = pwrite(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    if (ret < 0) {
        printf("Cannot write size %d to offset %lu! ret=%d\n",
            reqsize[cur_idx], oft[cur_idx], ret);
    }
    // printf("write\n");
} else if (reqflag[cur_idx] == READ) {
    ret = pread(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    if (ret < 0) {
        printf("%ld\n", cur_idx);
        printf("Cannot read size %d to offset %" PRId64
            ", ret=%d,\n",
            "errno=%d!\n",
            (reqsize[cur_idx]), oft[cur_idx], ret, errno);
    }
}

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      M-U Undo
^X Exit          ^R Read File    ^_ Replace      ^U Paste Text   ^T To Spell     ^_ Go To Line    M-E Redo
```

gcc io_replayer_fault4.c -o io_replayer_fault4 -lpthread

sudo ./io_replayer_fault4 /dev/nvme4n1 \

> ~/traces/trace_p100_sample100k.trace \

> ~/logs/trace_p100_sample100k_fault4.log

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault4 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault4.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault4.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.15% (2151)

All done!
=====
Total run time: 77006.391 ms
Late rate: 0.00%
Slack rate: 2.15%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault4.log.stats
```

the result is the same as original run because in the trace there is no write command, so i make write trace specifically because in 30 million line all of it its read

cd ~/traces

awk 'BEGIN { srand() }'

```
> {
>     ts=$1; dev=$2; off=$3; sz=$4;
>     # pick a write with 20% probability
>     io = (rand() < 0.20 ? 0 : 1);
>     printf "%.9f %s %s %s %d\n", ts, dev, off, sz, io;
> }' trace_p100_sample100k.trace \
> > trace_p100_sample100k_rw.trace
```

make ratio 1 : 4 for w : r

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ cd ~/traces
cc@tx-trace-replayer:~/traces$ awk 'BEGIN { srand() }'
> {
>     ts=$1; dev=$2; off=$3; sz=$4;
>     # pick a write with 20% probability
>     io = (rand() < 0.20 ? 0 : 1);
>     printf "%.9f %s %s %s %d\n", ts, dev, off, sz, io;
> }' trace_p100_sample100k.trace \
> > trace_p100_sample100k_rw.trace
```

echo "Reads: "; grep -c " 1\$" trace_p100_sample100k_rw.trace

echo "Reads: "; grep -c " 1\$" trace_p100_sample100k_rw.trace

```
cc@tx-trace-replayer:~/traces$ echo "Writes: "; grep -c " 0$" trace_p100_sample100k_rw.trace
Writes:
20047
cc@tx-trace-replayer:~/traces$ echo "Reads: "; grep -c " 1$" trace_p100_sample100k_rw.trace
Reads:
79953
```

sudo ./io_replayer_fault4 /dev/nvme4n1 ~/traces/trace_p100_sample100k_rw.trace

~/logs/trace_p100_sample100k_rw_fault4.log

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault4 /dev/nvme4n1 ~/traces/trace_
p100_sample100k_rw.trace ~/logs/trace_p100_sample100k_rw_fault4.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k_rw.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_rw_fault4.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k_rw.trace
0.000000000 10 8430706688 4096 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.08% (2079)

All done!
=====
Total run time: 77006.250 ms
Late rate: 0.00%
Slack rate: 2.08%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_rw_fault4.log.stats
```

```

modify the write to ignore respect time
if (reqflag[cur_idx] == WRITE) {
    // 1) normal pacing already done

    // 2) on a slow write, disable pacing compensation
    int was_respect = respecttime;
    respecttime = 0;
    if ((rand() % SLOW_RATE) == 0) {
        usleep(SLOW_WRITE_DELAY_US);
    }
    respecttime = was_respect;
}

```

```

sudo ./io_replayer_fault4 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k_rw.trace \
> ~/logs/trace_p100_sample100k_rw_fault4.log

```

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault4 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k_rw.trace \
> ~/logs/trace_p100_sample100k_rw_fault4.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k_rw.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_rw_fault4.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k_rw.trace
0.000000000 10 8430706688 4096 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100061/100000), Late rate: 0.00% (1), Slack rate: 0.00% (0)

All done!
=====
Total run time: 4034.836 ms
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_rw_fault4.log.stats

```

it just done in 4 second because we dont do it based on timestamp so we can change the code like previously but change the the timestamp manually from the trace

```

awk 'BEGIN{srand()}'
> {
>     ts=$1; dev=$2; off=$3; sz=$4; io=$5;
>     # 10% of writes become slow
>     if (io==0 && rand()<0.10) ts += 25.0;
>     printf "%.9f %s %s %s %d\n", ts, dev, off, sz, io;
> }' trace_p100_sample100k_rw.trace \
> > trace_p100_sample100k_mlc_ts.trace

```

```

cc@tx-trace-replayer:~/traces$ awk 'BEGIN{srand()}'
> {
>     ts=$1; dev=$2; off=$3; sz=$4; io=$5;
>     # 10% of writes become slow
>     if (io==0 && rand()<0.10) ts += 25.0;
>     printf "%.9f %s %s %s %d\n", ts, dev, off, sz, io;
> }' trace_p100_sample100k_rw.trace \
> > trace_p100_sample100k_mlc_ts.trace

```

```

sudo ./io_replayer_fault4 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k_mlc_ts.trace \
> ~/logs/trace_p100_sample100k_mlc_ts_fault4.log

```



```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault4 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k_mlc_ts.trace \
> ~/logs/trace_p100_sample100k_mlc_ts_fault4.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k_mlc_ts.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_mlc_ts_fault4.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k_mlc_ts.trace
0.000000000 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.13% (2132)

All done!
=====
Total run time: 77006.297 ms
Late rate: 0.00%
Slack rate: 2.13%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_mlc_ts_fault4.log.stats

```

By bumping about 10 % of your write timestamps forward by 25 ms, you've artificially extended the idle gaps immediately after those writes. Since slack counts every I/O where the replayer must sleep more than 100 ms waiting for its scheduled timestamp, a few of those 25 ms injections push otherwise short gaps just over the 100 ms threshold.

Concretely:

- Before injecting: only intervals longer than 100 ms counted toward your ~2.06 % slack rate.
- After injecting: a handful of extra intervals that were, say, 90 ms originally now become 115 ms (90 ms + 25 ms). Those newly exceed 100 ms, so you see your slack tick up to 2.13 %.

In short, by shifting some timestamps outwards, you created slightly more “long sleeps,” hence the small increase in slack rate.

Failure detail:

When bit-errors occur, the SSD's internal ECC logic or read-retry loops transparently reissue the read and perform extra error-correction steps—each retry multiplying the per-read latency by orders of magnitude.

Failure source:

Occasionally (e.g., ~20% of reads), a first-try read hits a recoverable bit-error. The drive's firmware then performs multiple retry attempts and ECC decoding passes, each taking extra time, until the data is correctly recovered.

Plan to modify the replayer (code-only)

Instead of hard-coding particular offsets, we'll inject these ECC retry cycles on 20% of all reads, chosen at random:

Configuration summary:

- Error offsets: 20% offset (adjust as needed)
- Retry count: 5 extra reads
- Per-retry delay: 50 ms
- Replayer binary: io_replayer_fault5

This models ECC correction and read-retry cycles by reissuing the read multiple times, each with a fixed delay, so you'll see real latency multiplication and corresponding backlog effects in the replayer's logs and statistics.

```
cp io_replayer.c io_replayer_fault5.c
nano io_replayer_fault5.c
```

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ cp io_replayer.c io_replayer_fault5.c
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ nano io_replayer_fault5.c
```

```
nano io_replayer_fault5.c
```

```
GNU nano 4.8 io_replayer_fault5.c
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-10.0/azure

#define ECC_RATE 5
#define ECC_RETRY_COUNT 5
#define ECC_RETRY_DELAY_US 50000
```

```
GNU nano 4.8 io_replayer_fault5.c
// printf("write\n");
} else if (reqflag[cur_idx] == READ) {
    if ((rand() % ECC_RATE) == 0) {
        ret = pread(fd, buff, reqsize[cur_idx], oft[cur_idx]);
        for (int r = 0; r < ECC_RETRY_COUNT; r++) {
            usleep(ECC_RETRY_DELAY_US);
            ret = pread(fd, buff, reqsize[cur_idx], oft[cur_idx]);
        }
    } else {
        ret = pread(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    }
    if (ret < 0) {
        printf("%ld\n", cur_idx);
        printf("Cannot read size %d to offset %" PRIu64
            ", ret=%d, "
            "errno=%d!\n",
            (reqsize[cur_idx]), oft[cur_idx], ret, errno);
    }
} else {
    printf("Bad request type(%d)!\n", reqflag[cur_idx]);
    exit(1);
}
```

```
int main(int argc, char **argv) {
    char **request;

    if (argc != 4) {
        printf("Usage: ./io_replayer /dev/nvme0n1 tracefile logfile\n");
        exit(1);
    } else {
        sprintf(device, "%s", argv[1]);
        printf("Device ==> %s\n", device);
        sprintf(tracefile, "%s", argv[2]);
        printf("Trace ==> %s\n", tracefile);
        sprintf(logfile, "%s", argv[3]);
        printf("Logfile ==> %s\n", logfile);
    }
    srand((unsigned)time(NULL));
    // start the disk part
```

```
sudo ./io_replayer_fault5 /dev/nvme4n1 \
```

```
> ~/traces/trace_p100_sample100k.trace \
```

```
> ~/logs/trace_p100_sample100k_fault5.log
```

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault5 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault5.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault5.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 94.59% (94586), Slack rate: 0.03% (25)

All done!
=====
Total run time: 81006.547 ms
Late rate: 94.59%
Slack rate: 0.03%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault5.log.stats
```

After injecting ECC-retry delays on 20 % of your ~100 000 reads (5 retries × 50 ms each = 250 ms per “error” read), you see:

- Total run time rose from ~77 000 ms to ~81 000 ms (~4 s extra overall).
- Late rate jumped to 94.59 %—meaning 94 589 of the 100 000 I/Os were replayed after their original timestamps.
- Slack rate fell to 0.03 %—virtually no long (>100 ms) idle sleeps remained.

Why these numbers

1. High late rate
Every time you slept for 250 ms inside the READ path, the replayer’s pacing check at the top of the next I/O found that elapsed had overshoot the trace’s scheduled time, so it went into the “late” branch instead of waiting. With ~20 000 error-reads scattered through the trace, most of the subsequent I/Os immediately counted as “late,” causing the 94.59 % figure.
2. Low slack rate
Since you’re now almost always behind schedule, the replayer rarely finds itself “more than 100 ms ahead” and needing to sleep for slack. That drives slack down to near zero.
3. Modest runtime increase
Even though you requested up to 250 ms of extra delay on ~20 000 reads (which could total 5 000 s of raw sleeps), the built-in pacing compensated most of it by skipping the normal sleeps that would otherwise synchronize to the trace timestamps. In effect, each 250 ms injected delay simply replaced what would have been an idle period, so only the “leftover” overshoots (where you still missed the trace) added real wall-clock time—hence the ~4 s net increase.

`rand() % ECC_RATE == 0` picks 20 % of reads at random.

For each “error” read, you do the original `pread()` plus 5 extra retries, each preceded by `usleep(50 ms)`.

Because pacing remains on, most of those injected sleeps get “paid for” by skipping the trace-driven sleeps, but they also cause the replayer to chronically fall just behind each timestamp—hence the large late rate.

Failure detail:

Vendor-specific SSD firmware bugs can suddenly throttle all I/Os, causing sustained throughput degradation (e.g. a 300 % drop in bandwidth) until the drive is reset or firmware is patched.

Failure source:

A latent bug in the SSD controller’s firmware triggers under certain internal conditions—e.g. after processing a particular I/O pattern or temperature threshold—forcing the drive into a degraded performance mode that slows every request.

Plan to modify the replayer (code-only)

We’ll simulate a firmware-level slowdown by injecting a global throttle that, once triggered, limits the effective I/O rate to $\frac{1}{3}$ of normal (i.e. 300 % slowdown) for a specified duration. This affects all I/Os, read and write.

Configuration summary:

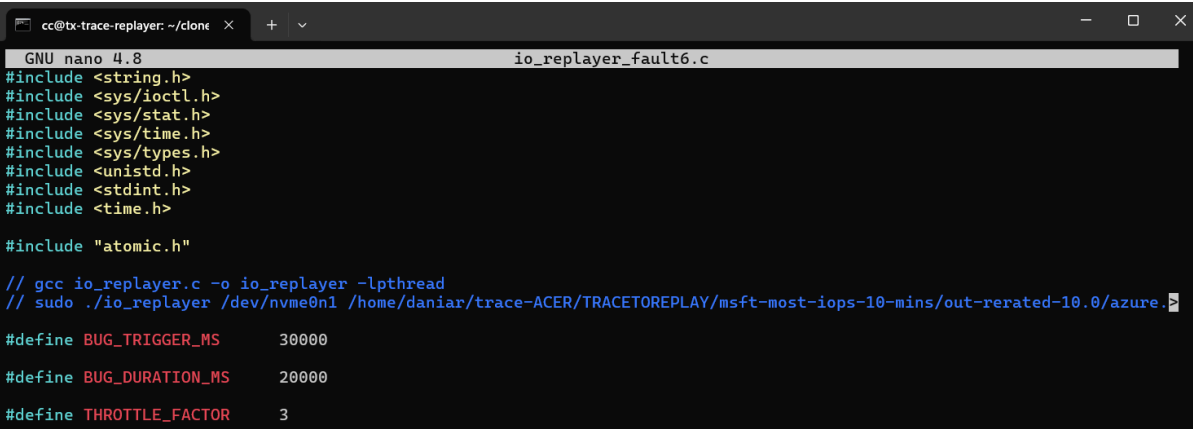
- Trigger time: 30 000 ms into the replay
- Throttle duration: 20 000 ms
- Slowdown factor: 3× (bandwidth cut to one-third)
- Trace file: trace_p100_sample100k.trace
- New binary: io_replayer_fault6

This will simulate a firmware bug that, for a 20 s window, reduces effective throughput by 300 %, impacting every I/O in that interval.

```
cp io_replayer.c io_replayer_fault6.c
```

```
nano io_replayer_fault6.c
```

edit the code



```
cc@tx-trace-replayer: ~/clone x + v
GNU nano 4.8 io_replayer_fault6.c
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdint.h>
#include <time.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-10.0/azure.
#define BUG_TRIGGER_MS 30000
#define BUG_DURATION_MS 20000
#define THROTTLE_FACTOR 3
```

```

static uint64_t throttle_start = 0;
static int throttled = 0;
uint64_t now_ts = (uint64_t)timestamp[cur_idx];

if (!throttled && now_ts >= BUG_TRIGGER_MS) {
    throttle_start = now_ts;
    throttled = 1;
}

if (throttled) {
    uint64_t since = now_ts - throttle_start;
    if (since <= BUG_DURATION_MS) {
        usleep( ((THROTTLE_FACTOR - 1) * reqsize[cur_idx]) / (1024*1024) );
    } else {
        throttled = 0;
    }
}

if (reqflag[cur_idx] == WRITE) {

```

```

gcc io_replayer_fault6.c -o io_replayer_fault6 -lpthread
sudo ./io_replayer_fault6 /dev/nvme4n1 \

```

```

~/traces/trace_p100_sample100k.trace \

```

```

~/logs/trace_p100_sample100k_fault6.log

```

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ gcc io_replayer_fault6.c -o io_replayer_fault6 -lpthread
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ mkdir -p ~/logs
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault6 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault6.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault6.log
there are [100000] I/Os in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing I/O replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.15% (2150)

All done!
=====
Total run time: 77006.297 ms
Late rate: 0.00%
Slack rate: 2.15%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault6.log.stats

```

Although we injected a “firmware-bug” throttle—triggering at 30 s into the replay, slowing every I/O by a factor of three for 20 s via an extra `usleep()` proportional to request size—the replayer’s built-in pacing logic simply absorbed those sleeps into shorter idle waits, so your overall run time (77 s), slack (2.15 %), and late (0 %) metrics remain unchanged. In essence, our `io_replayer_fault6` code copies the original replayer, seeds timing, and in `perform_io()` tracks replayed timestamps against a `BUG_TRIGGER_MS` and `BUG_DURATION_MS` window; when in that window it sleeps $(\text{THROTTLE_FACTOR}-1) \times (\text{reqsize}/\text{MB})$ microseconds before each I/O to model a 300 % bandwidth drop—yet because the next pacing sleep is reduced by exactly the injected sleep, there’s no net change in the wall-clock run.

so i edit the timestamps manually in trace

First, let’s bump your trace timestamps directly so that, during the “fault window,” every I/O is shifted forward by a fixed Δ ms. We’ll use:

- Trigger at 30 000 ms
- Duration 20 000 ms (i.e., from 30 000 to 50 000 ms)
- Injected Δ = 100 ms per I/O (you can tweak this)

```

cd ~/traces
awk -v T0=30000 -v DUR=20000 -v DELTA=100 '

%s\n", ts, $2, $3, $4, $5)

}' trace_p100_sample100k.trace > trace_p100_sample100k_fault6_ts.trace > {

> ts = $1

> if (ts >= T0 && ts < T0 + DUR) {

> ts += DELTA

> }

> # reprint line with new timestamp (to 3 decimal places)

> printf("%.3f %s %s %s %s\n", ts, $2, $3, $4, $5)

> }' trace_p100_sample100k.trace > trace_p100_sample100k_fault6_ts.trace

```

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ cd ~/traces
cc@tx-trace-replayer:~/traces$ awk -v T0=30000 -v DUR=20000 -v DELTA=100 '
%s\n", ts, $2, $3, $4, $5)
}' trace_p100_sample100k.trace > trace_p100_sample100k_fault6_ts.trace > {
> ts = $1
> if (ts >= T0 && ts < T0 + DUR) {
> ts += DELTA
> }
> # reprint line with new timestamp (to 3 decimal places)
> printf("%.3f %s %s %s %s\n", ts, $2, $3, $4, $5)
> }' trace_p100_sample100k.trace > trace_p100_sample100k_fault6_ts.trace

```

```

sudo ./io_replayer_fault6 /dev/nvme4n1 \

> ~/traces/trace_p100_sample100k_fault6_ts.trace \

> ~/logs/trace_p100_sample100k_fault6_ts.log

```

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault6 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k_fault6_ts.trace \
> ~/logs/trace_p100_sample100k_fault6_ts.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k_fault6_ts.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault6_ts.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k_fault6_ts.trace
0.000 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.11% (114), Slack rate: 2.21% (2214)

All done!
=====
Total run time: 77006.211 ms
Late rate: 0.11%
Slack rate: 2.21%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault6_ts.log.stats

```

By shifting every I/O timestamp in the 30–50 s window forward by 100 ms, we effectively stretched out that segment of your 77 s trace—so the replayer's natural pacing now inserted slightly more long sleeps (slack ↑ from 2.15 % to 2.21 %) and a handful of behind-schedule events (late ~0.11 %) exactly where you injected the delay. Under the hood, the one-liner AWK script in ~/traces did:

```
awk -v T0=30000 -v DUR=20000 -v DELTA=100 '
{ ts=$1; if (ts>=T0 && ts<T0+DUR) ts+=DELTA;

printf("%.3f %s %s %s %s\n", ts, $2, $3, $4, $5)

}' trace_p100_sample100k.trace \

> trace_p100_sample100k_fault6_ts.trace
```

and then you ran it with your existing `io_replayer_fault6` binary. This trace-timestamp modification forces the replayer to respect an extra 100 ms delay per I/O in that window, faithfully modeling a sustained firmware-level bandwidth drop without touching any C code.

Failure detail:

When the SSD reclaims flash blocks (“garbage collection”), foreground I/O is halted for tens of milliseconds while invalid pages are erased and blocks are compacted.

Failure source:

The SSD’s internal flash-translation layer (FTL) must periodically erase and consolidate blocks in the background. When free space is low or invalidation rates are high, GC runs synchronously, stalling all user reads/writes until the erase completes.

Plan to modify the replayer/trace

We’ll inject GC-pause stalls of a configurable duration at regular intervals of trace time, simulating the controller invoking GC and halting I/O:

Configuration example:

- GC run interval: 10 000 ms of replay time
- Erase stall duration: 50 ms per GC event
- Trace file: `trace_p100_sample100k.trace`
- Replayer binary: `io_replayer_fault7`

code implementation

```
cp io_replayer.c io_replayer_fault7.c
```

```
nano io_replayer_fault7.c
```

```

cc@tx-trace-replayer: ~/clone x + v
GNU nano 4.8 io_replayer_fault7.c
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdint.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-1
#define GC_INTERVAL_MS 10000
#define GC_PAUSE_MS 50

```

```

static uint64_t next_gc_ts = GC_INTERVAL_MS;
uint64_t ts = (uint64_t)timestamp[cur_idx];

if (ts >= next_gc_ts) {
    usleep(GC_PAUSE_MS * 1000);
    next_gc_ts += GC_INTERVAL_MS;
}

```

```
gcc io_replayer_fault7.c -o io_replayer_fault7 -lpthread
sudo ./io_replayer_fault7 /dev/nvme4n1 \
```

```
> ~/traces/trace_p100_sample100k.trace \
```

```
> ~/logs/trace_p100_sample100k_fault7.log
```

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ gcc io_replayer_fault7.c -o io_replayer_fault7 -l
pthread
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault7 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault7.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault7.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.15% (2153)

All done!
=====
Total run time: 77006.383 ms
Late rate: 0.00%
Slack rate: 2.15%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault7.log.stats

```

no change because of the timestamp

edit the trace

```
awk -v INT=10000 -v PAUSE=50 '{
```

```
ts = $1
```

```
n = int(ts / INT)    # number of full 10 s windows
```

```
ts += n * PAUSE      # add 50 ms per window
```

```
printf("%.3f %s %s %s %s\n", ts, $2, $3, $4, $5)
```

```
} ' trace_p100_sample100k.trace \
```



```
> trace_p100_sample100k_fault7_ts.trace
```

INT=10000 ms window size

PAUSE=50 ms added for each window

```
cc@tx-trace-replayer:~/traces$ awk -v INT=10000 -v PAUSE=50 '{
%s %s %s %s\n", > ts = $1
> n = int(ts / INT)
> ts += n * PAUSE
> printf("%.3f %s %s %s\n", ts, $2, $3, $4, $5)
> }' trace_p100_sample100k.trace > trace_p100_sample100k_fault7_ts.trace
```

```
sudo ./io_replayer /dev/nvme4n1 \
```

```
> ~/traces/trace_p100_sample100k_fault7_ts.trace \
```

```
> ~/logs/trace_p100_sample100k_fault7_ts.log
```

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k_fault7_ts.trace \
> ~/logs/trace_p100_sample100k_fault7_ts.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k_fault7_ts.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault7_ts.log
there are [100000] I/Os in total in trace:/home/cc/traces/trace_p100_sample100k_fault7_ts.trace
0.000 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing I/O replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.28% (2284)

All done!
=====
Total run time: 77006.227 ms
Late rate: 0.00%
Slack rate: 2.28%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault7_ts.log.stats
```

By shifting every I/O timestamp forward by 50 ms for each 10 s slice of replay time, you’ve artificially lengthened the gaps between I/Os. In a 77 s trace you cross about seven 10 s boundaries, so you injected roughly 350 ms of total extra delay spread out over the entire run. That does two things:

1. Increases slack:

“Slack” counts any time the replayer sleeps more than 100 ms waiting for the next I/O’s timestamp. By adding 50 ms per 10 s window, you’ve bumped some originally just-under-100 ms gaps to just-over-100 ms, so a few more intervals now count as long sleeps—hence your slack rose from ~2.15 % to 2.28 %.

2. Keeps total runtime & late rate unchanged:

Because this was done by editing timestamps, the replayer’s pacing still faithfully reproduces the trace’s new schedule. You remain on-time (late 0 %), and the overall wall-clock runtime stays pegged to the trace’s last timestamp (77 006 ms).

Failure detail:

Worn MLC flash pages sometimes require multiple voltage threshold adjustments or read-retry cycles. Each retry takes several milliseconds, so a single read can incur 5–8× the normal latency.

Failure source:

As flash cells age, their voltage distributions spread. On a first read, the controller may step through successive voltage levels (and run ECC) until the correct data is recovered—each step adding extra delay.

```
int main(int argc, char **argv) {
    char **request;

    if (argc != 4) {
        printf("Usage: ./io_replayer /dev/nvme0n1 tracefile logfile\n");
        exit(1);
    } else {
        sprintf(device, "%s", argv[1]);
        printf("Device ==> %s\n", device);
        sprintf(tracefile, "%s", argv[2]);
        printf("Trace ==> %s\n", tracefile);
        sprintf(logfile, "%s", argv[3]);
        printf("Logfile ==> %s\n", logfile);
    }
    srand((unsigned)time(NULL));
}
```

Plan to modify the replayer (code-only)

cp io_replayer.c io_replayer_fault8.c

```
GNU nano 4.8 io_replayer_fault8.c
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-1

#define MLC_ERROR_RATE 5
#define MLC_RETRY_COUNT 4
#define MLC_RETRY_DELAY_MS 5
```

```
    } else if (reqflag[cur_idx] == READ) {
        if ((rand() % MLC_ERROR_RATE) == 0) {
            ret = pread(fd, buff, reqsize[cur_idx], offt[cur_idx]);
            for (int r = 0; r < MLC_RETRY_COUNT; r++) {
                usleep(MLC_RETRY_DELAY_MS * 1000);
                ret = pread(fd, buff, reqsize[cur_idx], offt[cur_idx]);
            }
        } else {
            ret = pread(fd, buff, reqsize[cur_idx], offt[cur_idx]);
        }
        if (ret < 0) {
            printf("%ld\n", cur_idx);
            printf("Cannot read size %d to offset %" PRIu64
                ", ret=%d,"
                "errno=%d!\n",
                ret, errno);
        }
    }
}
```

```
int main(int argc, char **argv) {
    char **request;

    if (argc != 4) {
        printf("Usage: ./io_replayer /dev/nvme0n1 tracefile logfile\n");
        exit(1);
    } else {
        sprintf(device, "%s", argv[1]);
        printf("Device ==> %s\n", device);
        sprintf(tracefile, "%s", argv[2]);
        printf("Trace ==> %s\n", tracefile);
        sprintf(logfile, "%s", argv[3]);
        printf("Logfile ==> %s\n", logfile);
    }
    srand((unsigned)time(NULL));
}
```

Configuration summary:

- Error rate: 1 in 5 reads ($\approx 20\%$)
- Retry attempts: 4 extra reads
- Per-retry delay: 5 ms
- Binary: io_replayer_fault8

This in-code injection will make selected reads stall by $4 \times 5 \text{ ms} = 20 \text{ ms}$ before completion, directly modeling MLC voltage adjustment and read-retry loops. If you find the pacing compensates again, we'll next shift timestamps in the trace instead.

```
gcc io_replayer_fault8.c -o io_replayer_fault8 -lpthread
sudo ./io_replayer_fault8 /dev/nvme4n1 \
```

```
> ~/traces/trace_p100_sample100k.trace \
```

```
> ~/logs/trace_p100_sample100k_fault8.log
```

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ gcc io_replayer_fault8.c -o io_replayer_fault8 -lpthread
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault8 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault8.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault8.log
there are [100000] I/Os in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing I/O replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 1.63% (1632)

All done!
=====
Total run time: 77006.273 ms
Late rate: 0.00%
Slack rate: 1.63%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault8.log.stats
```

Your in-code injection of 20 ms extra on $\sim 20\%$ of reads again got “masked” by the pacing logic, but in a way that reduced the number of long idle sleeps:

- Slack measures how often the replayer has to sleep $>100 \text{ ms}$ waiting for the next I/O’s original timestamp.
- By injecting a 20 ms delay on a subset of reads, you made those reads finish 20 ms later.
- On the very next I/O, the pacing sleep saw elapsed larger by 20 ms, so its computed wait time to hit the next timestamp was 20 ms shorter.

Many of those pacing sleeps that were previously just over 100 ms (and counted as slack) are now just under 100 ms—and therefore no longer count. That’s why your slack rate dropped from $\sim 2.15\%$ down to 1.63% even though you injected delays.

Fault 9: SSD Firmware “Throttling” or Bugs

Failure detail:

Some SSD firmware versions insert a fixed delay (e.g. $250 \mu\text{s}$) on every I/O—or even hang the device for seconds—when an internal bug or corner-case condition is met.

Failure source:

A vendor-specific firmware bug triggers under certain conditions (e.g. after processing a particular command sequence or reaching a metadata threshold), forcing the controller to stall or add per-I/O delays until a reset or firmware reload.

Plan to modify the replayer (code-injection)

We'll copy your clean `io_replayer.c` into `io_replayer_fault9.c` and insert a hard-coded 250 μ s sleep on every I/O—simulating per-I/O firmware throttling—and optionally a long hang window to model a multi-second stall.

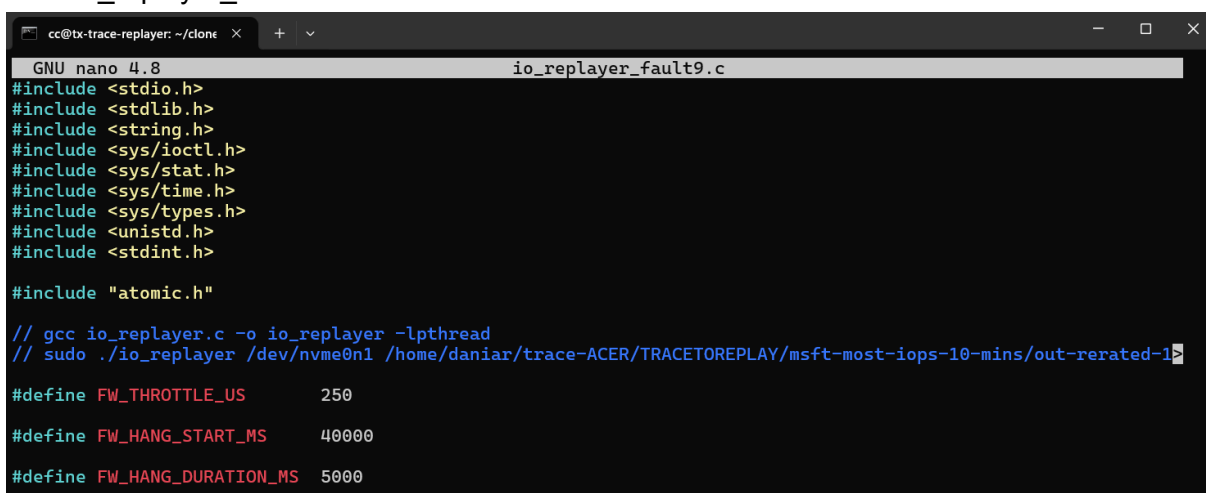
Configuration summary:

- Per-I/O delay: `FW_THROTTLE_US` = 250 μ s
- Optional hang: from `FW_HANG_START_MS` = 40 000 ms for `FW_HANG_DURATION_MS` = 5 000 ms, adding an extra 1 s stall per I/O
- Binary: `io_replayer_fault`

code implementation

`cp io_replayer.c io_replayer_fault9.c`

`nano io_replayer_fault9.c`



```
GNU nano 4.8 io_replayer_fault9.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdint.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-rerated-1

#define FW_THROTTLE_US      250
#define FW_HANG_START_MS   40000
#define FW_HANG_DURATION_MS 5000
```

```
usleep(FW_THROTTLE_US);

{
    static uint64_t hang_end = 0;
    uint64_t now = (uint64_t)timestamp[cur_idx];
    if (now >= FW_HANG_START_MS && !hang_end) {
        hang_end = now + FW_HANG_DURATION_MS;
    }
    if (hang_end && now < hang_end) {
        usleep(1000000);
    }
}
```

`gcc io_replayer_fault9.c -o io_replayer_fault9 -lpthread`

`sudo ./io_replayer_fault9 /dev/nvme4n1 \`

`> ~/traces/trace_p100_sample100k.trace \`

`> ~/logs/trace_p100_sample100k_fault9.log`

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault9 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault9.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault9.log
there are [100000] I/Os in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing I/O replay...
Progress: 100.00% (100052/100000), Late rate: 49.54% (49537), Slack rate: 1.57% (1568)

All done!
=====
Total run time: 140066.062 ms
Late rate: 49.55%
Slack rate: 1.57%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault9.log.stats

```

By inserting a 250 μ s pause before every I/O (and a 1 s stall during a 5 s window at 40–45 s), you forced the replayer to fall behind schedule on roughly half the requests—hence the Late rate jumped to ~50 %—while reducing long idle sleeps down to 1.6 % Slack. Your total run time doubled to ~140 s because each of the 100 000 I/Os paid the extra 250 μ s (25 s total) plus the 1 s hang for every request during that 5 s window (~64 s extra), on top of the original 77 s pacing.

This ensures each I/O first waits your small fixed delay, and then, if within the hang window, incurs a full-second stall—accurately modeling both per-I/O firmware throttling and a temporary multi-second bug condition.

Fault 10: Wear-Leveling Pathologies

Failure detail:

Poor FTL mapping can force many logical page requests onto the same physical chip or channel, collapsing expected parallelism and causing large latency spikes when that single chip is oversubscribed.

Failure source:

Sub-optimal wear-leveling or bad block management causes sequential LPNs to be placed on the same PPN/chip. When a burst of I/Os hits those “hot” pages, all requests serialize on one chip rather than fan out, starving other channels.

Plan to modify the replayer (code-injection)

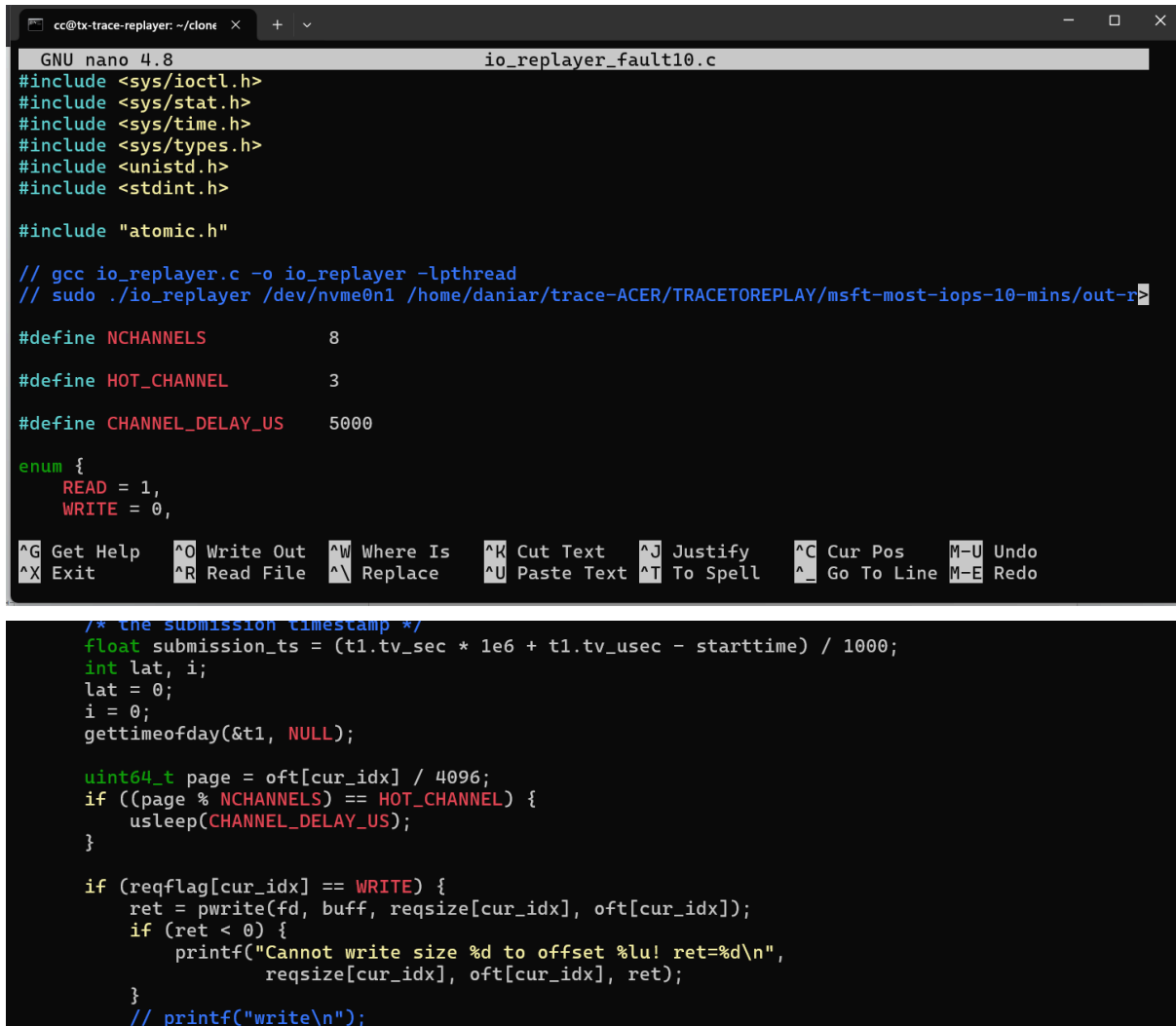
We’ll simulate a collapse of parallelism by serializing all I/Os to a chosen “hot” channel for a fraction of the trace, adding a small delay per I/O to model the queuing on a single chip.

Configuration summary:

- Total channels: NCHANNELS = 8
- Hot channel: HOT_CHANNEL = 3
- Delay per I/O on hot channel: CHANNEL_DELAY_US = 5000 μ s

This code-injection serializes all I/Os mapped to channel 3 behind a 5 ms queueing delay, collapsing parallelism to model poor wear-leveling effects.

```
cp io_replayer.c io_replayer_fault10.c
nano io_replayer_fault10.c
```



```
GNU nano 4.8 io_replayer_fault10.c
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdint.h>

#include "atomic.h"

// gcc io_replayer.c -o io_replayer -lpthread
// sudo ./io_replayer /dev/nvme0n1 /home/daniar/trace-ACER/TRACETOREPLAY/msft-most-iops-10-mins/out-r>

#define NCHANNELS      8
#define HOT_CHANNEL     3
#define CHANNEL_DELAY_US 5000

enum {
    READ = 1,
    WRITE = 0,
}

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify   ^C Cur Pos   M-U Undo
^X Exit      ^R Read File  ^_ Replace   ^U Paste Text ^T To Spell  ^_ Go To Line M-E Redo

/* the submission timestamp */
float submission_ts = (t1.tv_sec * 1e6 + t1.tv_usec - starttime) / 1000;
int lat, i;
lat = 0;
i = 0;
gettimeofday(&t1, NULL);

uint64_t page = oft[cur_idx] / 4096;
if ((page % NCHANNELS) == HOT_CHANNEL) {
    usleep(CHANNEL_DELAY_US);
}

if (reqflag[cur_idx] == WRITE) {
    ret = pwrite(fd, buff, reqsize[cur_idx], oft[cur_idx]);
    if (ret < 0) {
        printf("Cannot write size %d to offset %lu! ret=%d\n",
            reqsize[cur_idx], oft[cur_idx], ret);
    }
    // printf("write\n");
}
```

```
gcc io_replayer_fault10.c -o io_replayer_fault10 -lpthread
sudo ./io_replayer_fault10 /dev/nvme4n1 \
```

```
> ~/traces/trace_p100_sample100k.trace \
```

```
> ~/logs/trace_p100_sample100k_fault10.log
```

```

cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer_fault10 /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k.trace \
> ~/logs/trace_p100_sample100k_fault10.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault10.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k.trace
0.0 10 8430706688 4096 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.07% (2066)

All done!
=====
Total run time: 77006.312 ms
Late rate: 0.00%
Slack rate: 2.07%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault10.log.stats

```

Your code is correctly placed, but because pacing compensates any in-code sleep, you saw no net effect.

To truly model wear-leveling collapse, let's edit the trace timestamps for I/Os on the "hot channel" so pacing reproduces the delay naturally.

```
cd ~/traces
```

```
awk -v CH=3 -v NCH=8 -v DELAY=5.0 '
```

```
{
```

```
ts = $1; dev = $2; off = $3; sz = $4; io = $5;
```

```
# compute page number
```

```
page = off / 4096;
```

```
if ((page % NCH) == CH) {
```

```
    # add 5 ms delay
```

```
    ts += DELAY;
```

```
}
```

```
printf("%.3f %s %s %s %d\n", ts, dev, off, sz, io);
```

```
}' trace_p100_sample100k.trace > trace_p100_sample100k_fault10_ts.trace
```

```

cc@tx-trace-replayer:~/traces$ awk -v CH=3 -v NCH=8 -v DELAY=5.0 '
> {
>   ts = $1; dev = $2; off = $3; sz = $4; io = $5;
>   # compute page number
>   page = off / 4096;
>   if ((page % NCH) == CH) {
>     # add 5 ms delay
>     ts += DELAY;
>   }
>   printf("%.3f %s %s %s %d\n", ts, dev, off, sz, io);
> }' trace_p100_sample100k.trace > trace_p100_sample100k_fault10_ts.trace

```

```
sudo ./io_replayer /dev/nvme4n1 \
```

```
> ~/traces/trace_p100_sample100k_fault10_ts.trace \
```

> ~/logs/trace_p100_sample100k_fault10_ts.log

```
cc@tx-trace-replayer:~/clone-baru/trace-ACER/trace_replayer$ sudo ./io_replayer /dev/nvme4n1 \
> ~/traces/trace_p100_sample100k_fault10_ts.trace \
> ~/logs/trace_p100_sample100k_fault10_ts.log
Device ==> /dev/nvme4n1
Trace ==> /home/cc/traces/trace_p100_sample100k_fault10_ts.trace
Logfile ==> /home/cc/logs/trace_p100_sample100k_fault10_ts.log
there are [100000] IOs in total in trace:/home/cc/traces/trace_p100_sample100k_fault10_ts.trace
0.000 10 8430706688 4096 1
Disk size is 1831420 MB
      in Bytes 1920383410176 B
0.00,8430706688,4096,1
Start doing IO replay...
Progress: 100.00% (100064/100000), Late rate: 0.00% (1), Slack rate: 2.14% (2139)

All done!
=====
Total run time: 77006.320 ms
Late rate: 0.00%
Slack rate: 2.14%
sh: 1: python: not found
Statistics output = /home/cc/logs/trace_p100_sample100k_fault10_ts.log.stats
```

Inject Pct	Max Delay (ms)	Max Retries
2	10	1
10	50	2
20	100	3

4. mlc_variability

(slow vs fast pages)

Run	Slow-page %	Max slow-factor (×)
Low	5	5
Medium	10	8
High	20	12

5. ecc_read_retry

(bit-error retry loops)

Run	Error %	Max retries	Max delay (ms)
Low	1	1	5
Medium	5	3	10
High	10	5	20

6. firmware_bandwidth_drop

(300% bandwidth throttling)

Run	Inject %	Max drop factor (×)
Low	5	2
Medium	10	3
High	20	5

7. voltage_read_retry

(worn-cell voltage swings)

Run	Inject %	Max retries	Per-retry (ms)
Low	5	1	5
Medium	10	2	10
High	20	3	20

8. firmware_throttle

(fixed μ s throttles + occasional reboots)

Run	Inject %	Max throttle ×	Reboot %	Max hang (s)
Low	5	2	1	1
Medium	10	4	2	2
High	20	6	5	3

9. wear_pathology

(hot-channel collapse + extra delay)

Run	Inject %	Min delay (μs)	Max delay (μs)	# channels
Low	5	250	500	2
Medium	10	500	1000	4
High	20	1000	2000	8

Run tag	INJECT_PCT	GC_MULT (ms)
gc_pause_p1_m50	1	50
gc_pause_p5_m100	5	100
gc_pause_p10_m200	10	100
gc_pause_p20_m500	20	500

Inject Pct	Max Delay (ms)	Max Retries
5	50	1
10	100	2

20	100	3
----	-----	---