Report 2

Introduction

This semester I have worked mostly on measuring the performance of two page replacement algorithms which I have developed last semester. I have started implementing test programs to show one algorithm performs better than the other. In this process, I have fixed bugs in LRU-K implementation and had better understanding about page faults and the code flow. Later I have also spent few cycles on measuring performance by running real time server workloads

LRU-K Implementation

Most of the work for LRU-K implementation was done in last semester. This semester I have fixed few bugs(calling LRU-K related functions in right place) and got a better picture in understanding about page caches and the kernel code flow. Last semester I didn't verify whether LRU-K work as intended (i.e getting page heat value as expected). Initially I have faced issues in calculating page heat for a page when it is read from page cache. Later I understood, when a page is read from page cache, the OS doesn't know about it and no function will get triggered. Then I figured out a way to identify if page is read from page cache by reading same page from different process. When second process reads that page for first time, the page will be in page cache but not in that corresponding process page table, it will incur minor page faults and the page fault handling function in OS will get invoked. After this understanding, I wrote a simple test program to confirm LRU-K works correctly by checking the page heat value.

Synthetic Workloads

Test Program 1 (LRUK > LRU)

I have to come up with a program to show LRU-K performs better than LRU. I chose page faults as a metric to measure performance between two algorithms. Initially I'm not analytically able to reason out the number of page faults and I didn't get expected number of page faults. After several attempts, I figured out it is because of the readahead feature. Once, I disabled the readahead I got expected number of page faults. Then I have focussed on scenarios in which LRU-K beats LRU comprehensively. After several attempts, I'm able to find a scenario in which LRU-K performs better than LRU.

Setup

2GB RAM

3GB File created using dd

LRU-K threshold is 10 seconds

Three programs have to be compiled

1. cc freq.c -o freq

2. cc freq2.c -o freq2
3. cc temp.c -o temp

Run sh testscript.sh under advproject/test/lruk directory as 'sh testscript.sh <filename>'

Scenario

1. First read 512 MB array (from 1 to 1.5 GB) (./freq)

2. sleep for 10 seconds

3. Then read 512 MB array again (./freq2)

4. sleep 10 seconds

5. Read 750 MB array for first time(from 0 to 750M).

6. Again read 750M array for three times without any delay. In case of LRUK page heat value will be updated and it will be moved to active list and in case of LRU, reference bit will be set and it will be moved to active list.

7.Read 512M array(from 1 to 1.5 GB) again. In case of LRU, reference bit is set and it will be moved to active list. In case of LRU-K page heat value will be updated but the page heat value will be less than threshold value because the difference between current reference time and last reference will be atleast 60 seconds. LRU-K threshold value is 10.

At this stage,

In LRU both 750M and 512M array will be in active list and pages of 512M array will be in head of list since it is recently referred

In LRU-K 750M will be in active list but 512M will be in inactive list.

8. Now read first 2GB array for two times(./temp). Since the RAM won't have space for entire 2GB some pages from page cache will be removed to accommodate for new pages. In case of LRU, the pages of 750M array will be in the first place to be removed compared to pages of 512M array. In case of LRUK it is opposite, pages of 512M will be in first place to be removed since it is in inactive list.

9. Now read 1GB array again. At this step, LRU technique should throw large number of faults compared to LRUK.

I have run this scenario for around 30 times and the results are consistent on every trial.

The overall number of faults produced by LRU is ~912866 and by LRUK is ~771965. We could see the difference is around 150K page faults. LRUK on average 6 seconds faster than LRU. LRU-K shows around **9%** improvement over LRU

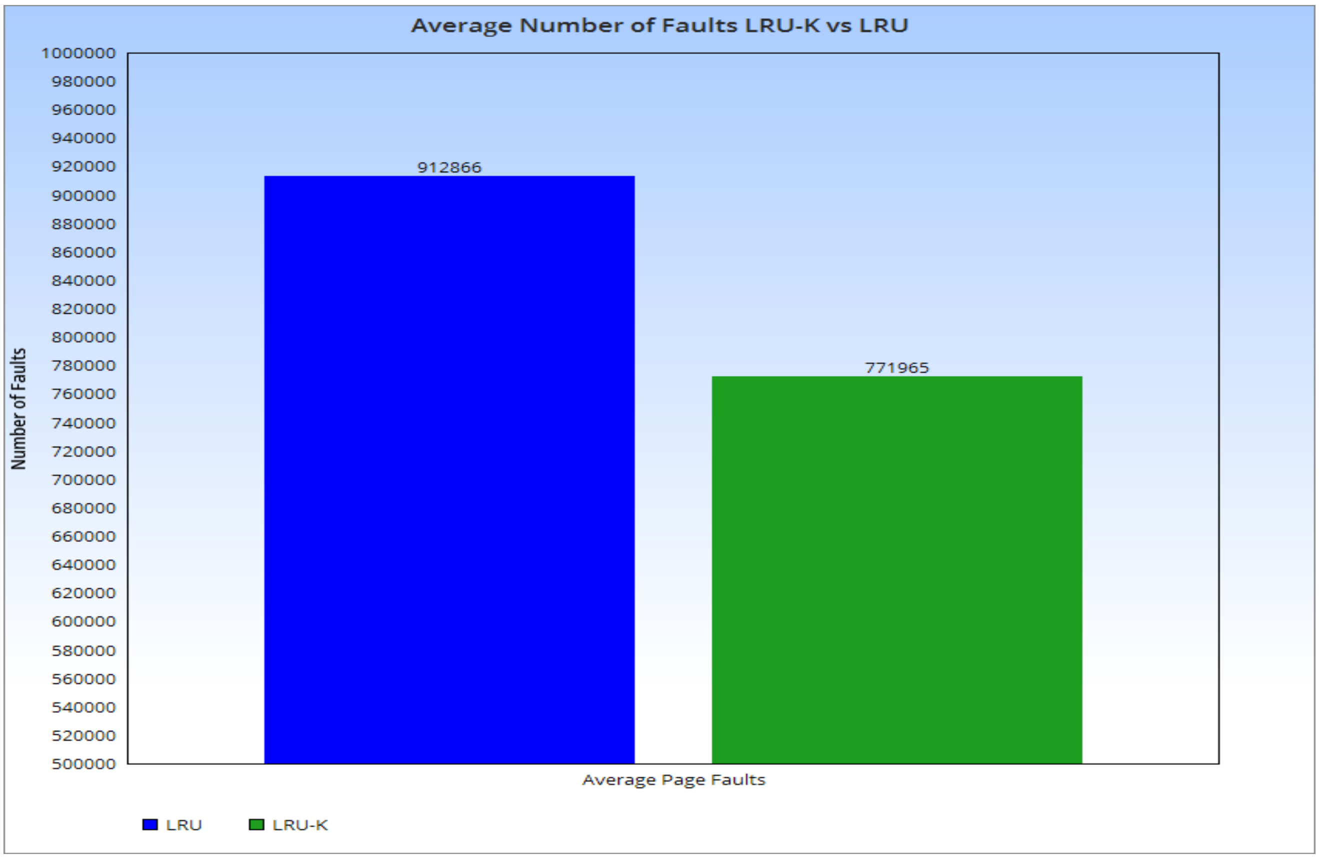
Results

The time taken by both LRU and LRU-K in executing the same program in shown below. We could notice that LRU-K is the clear winner in this case.

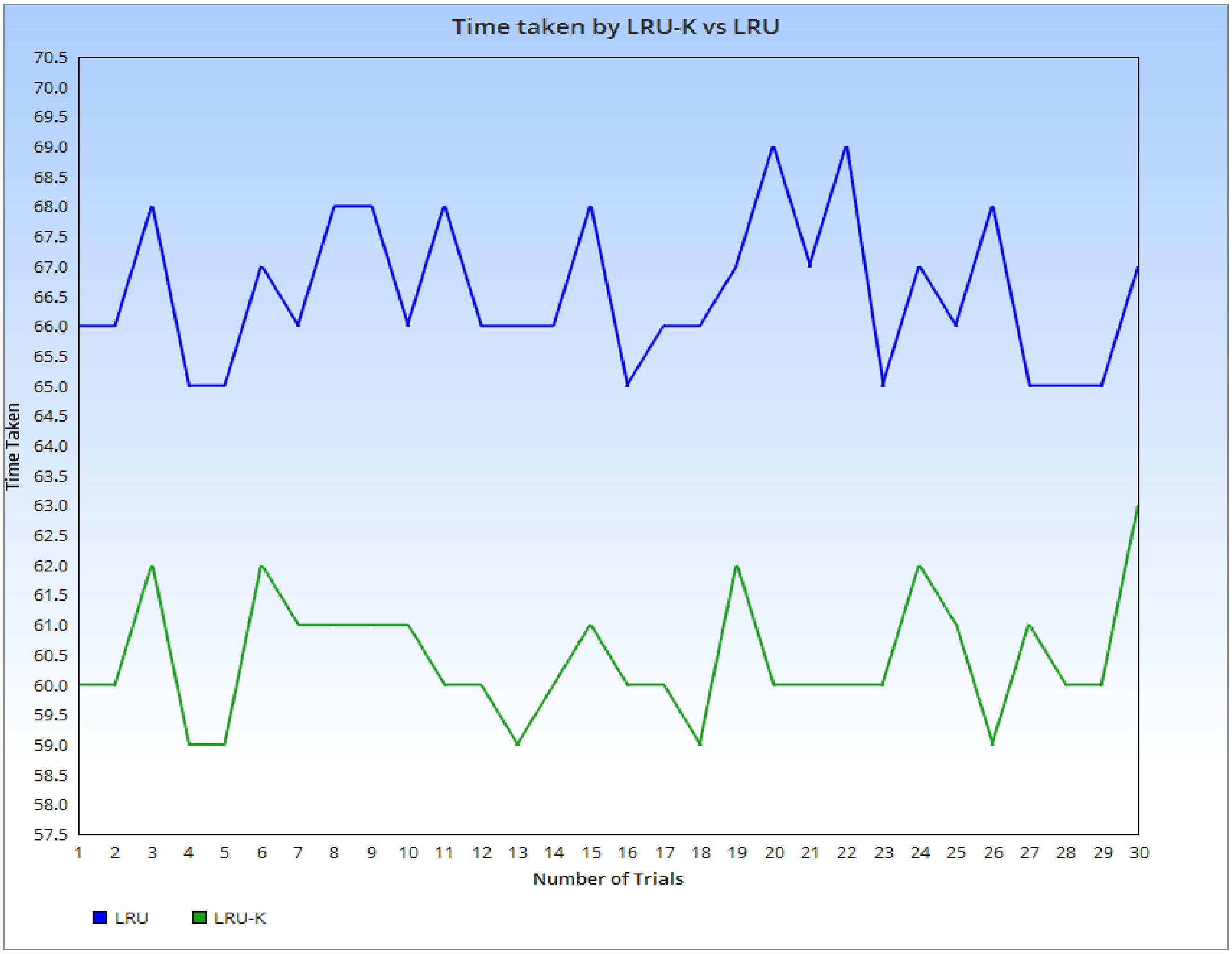
Average Time Taken by LRU and LRU-K (in Seconds)

LRU	LRU-K
66.5	60.4

The average number of faults produced by LRU and LRU-K is indicated by chart below



The time taken by LRU and LRU-K for each trial



Test Program 2 (LRU>LRUK)

I have focused on implementing a test program to show LRU performs better than LRU-K. After several attempts, I'm able to find one such scenario where LRU performs better than LRUK.

Setup

2GB RAM

3GB File created using dd

LRU-K threshold is 10 seconds

Three programs have to be compiled

1. cc freq.c -o freq
2. cc freq2.c -o freq2
3. cc temp.c -o temp

Run sh testscript.sh under advproject/test/lru directory as 'sh testscript.sh <filename>'

Scenario

1. Read 1GB array for first time (from 0 to 1G)

2. Sleep for 15 seconds

3. Read the same 1GB array again

4. Sleep for 15 seconds

5. Read 512MB array for first time (from 1G to 1.5G)

6. Read 512MB array again for four times without a delay. In case of both LRU and LRU-K, 512MB pages will be in active list and the reference bit is set.

7. Read 1GB array again(from 0 to 1G). In case of LRU, reference bit will be set and it is moved to active list. In case of LRU-K, since these array pages are accessed after LRUK threshold seconds, it won't be moved to active list and it remains in inactive list

8. Read 2GB array (from 0 to 2G) two times. In case of LRU, 512 MB pages will be replaced first to accommodate for new pages and in case of LRU-K, 1GB pages will be replaced first. So In case of LRU-K more number of 1GB pages will be replaced

9. Now read 1GB array of pages again. As expected, LRU-K throws larger number of faults compared to LRU.

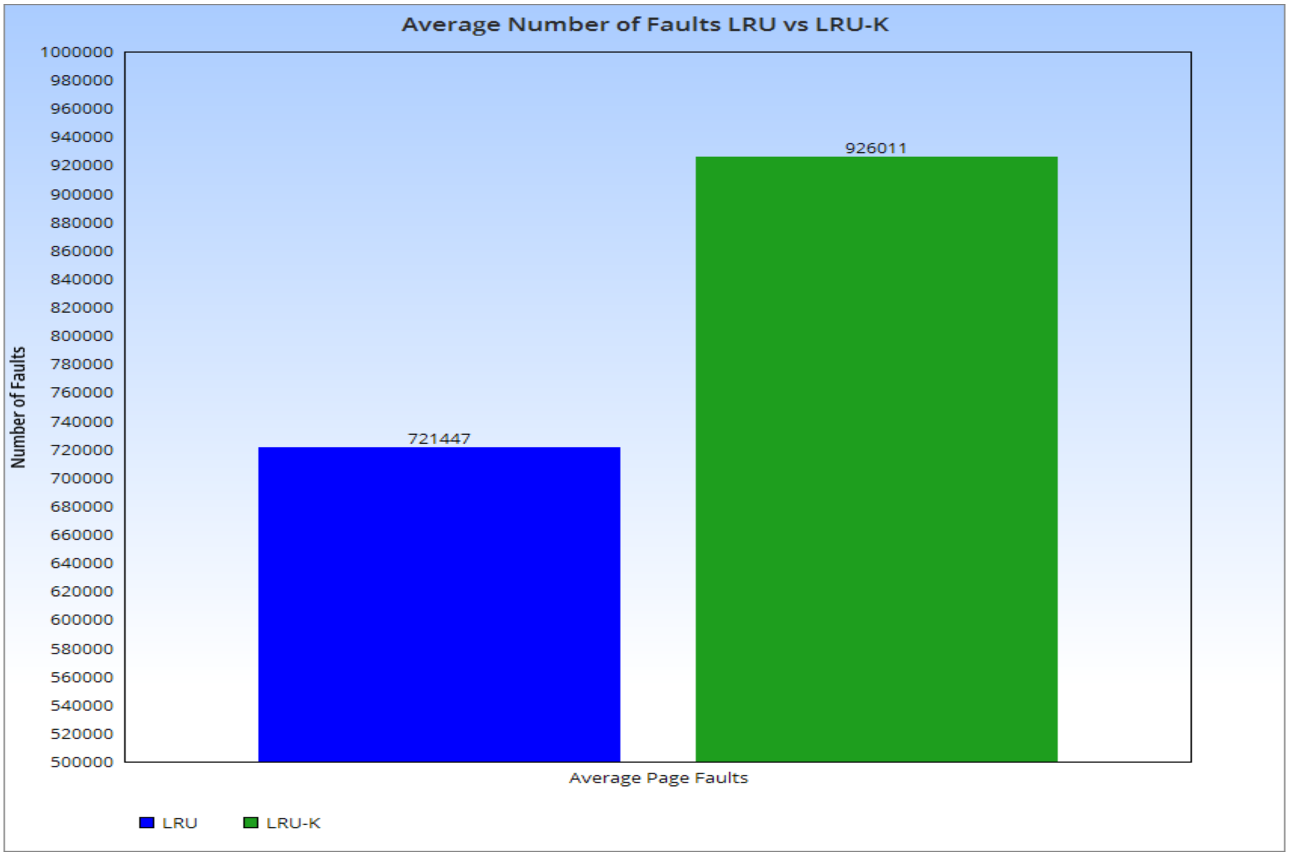
The overall number of faults produced by LRU is ~721447 and by LRUK is ~926011. We could see the difference is around 200K page faults. LRU on average 9 seconds faster than LRU-K. LRU shows around **11%** improvement over LRU-K

Results

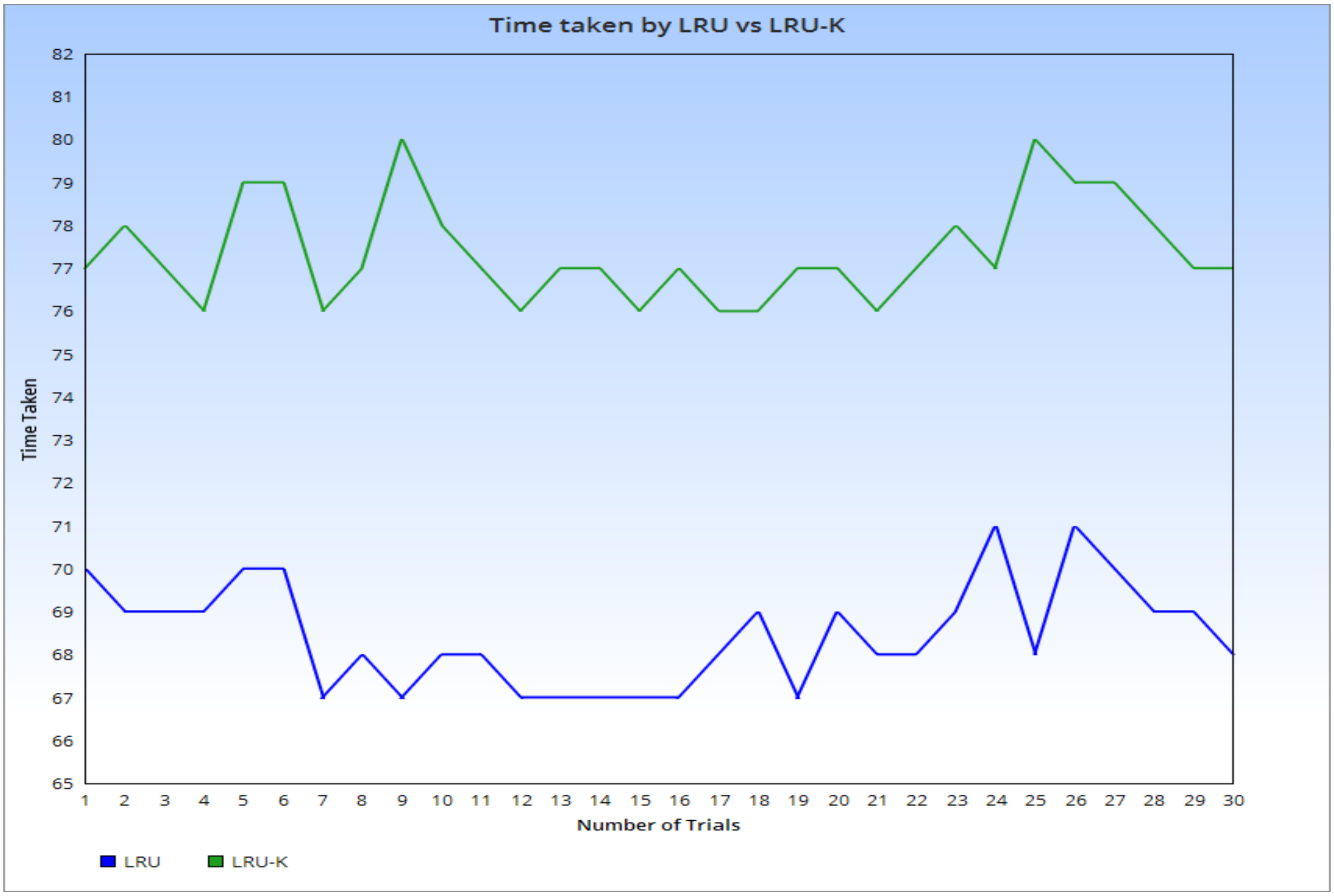
Average Time Taken by LRU and LRU-K (in Seconds)

LRU	LRU-K
68.4	77.6

The average number of page faults produced by LRU and LRU-K is shown below



The time taken by LRU and LRU-K for each trial



Real Server Workload Measurements

I have tried to measure LRU and LRU-K performance for media streaming server workloads. There is already a setup done which benchmarks few parameters like number of client connection, number of requests and replies and reply rate. Mediastream client runs in one container and server runs in another container. The clients using httpperf initiates four requests to server for various video qualities. The client first does this for 25 sessions. In this case, 100 connections were made by client successfully in both LRU and LRU-K. In case of 500 sessions, only one or two times I see client benchmark succeeded without any crash. Otherwise the benchmark getting failed due to errors and repeat that test till it finds maximum number of connections that can be made. During that process its getting crashed.

One trial in both LRU and LRU-K has succeeded and I have included readings only for those trials since all other trials are crashed

LRU-K

Threshold is set as 30 seconds

Total connections = 2000

Total errors = 0

Percentage failure = 0

Requests: 33892

Replies: 33892

Reply rate: 25.90

Reply time: .10

Net I/O: 299463.4

LRU

Total connections = 928

Total errors = 0

Percentage failure = 0

Requests: 15728

Replies: 15728

Reply rate: 11.95

Reply time: .10

Net I/O: 146746.6

We cannot arrive ar a conclusion based on one trial.