Report 2

Tamilmani Manoharan(110385545)

Advance Project

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

I have to come up with two programs to show LRU-K performs better than LRU and LRU better than LRU-K. 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. I have disabled readahead using posix_fadvise from userspace.

Test Program 1 (LRU-K > LRU)

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. This can be configured in grub file.

3GB File created using dd (`dd if=/dev/urandom of=3g.txt bs=1048576 count=3072`)

LRU-K threshold is set to 10 seconds

Three programs have to be compiled under advproject/test/lruk directory

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>'. Number of faults incurred will be printed in console and time taken is redirected to two logs. results-lru.txt is for LRU and results-lruk.txt for LRU-K.

The test scenario is explained in detail in advproject/test/lruk/scenario.txt.

Results

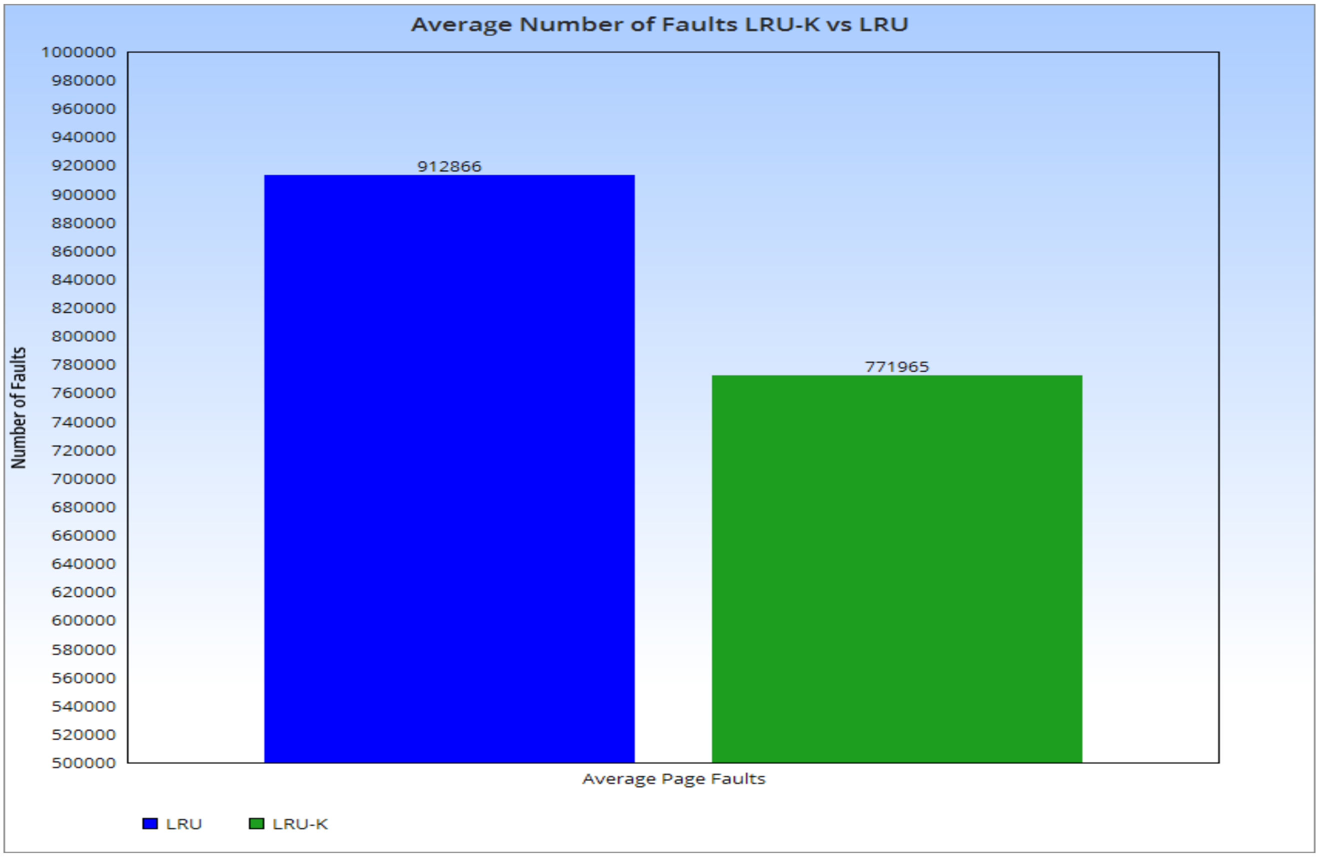
The time taken by both LRU and LRU-K in executing the same program is shown below. We could notice that LRU-K is the clear winner in this case. The results are computed for around 30 trials

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

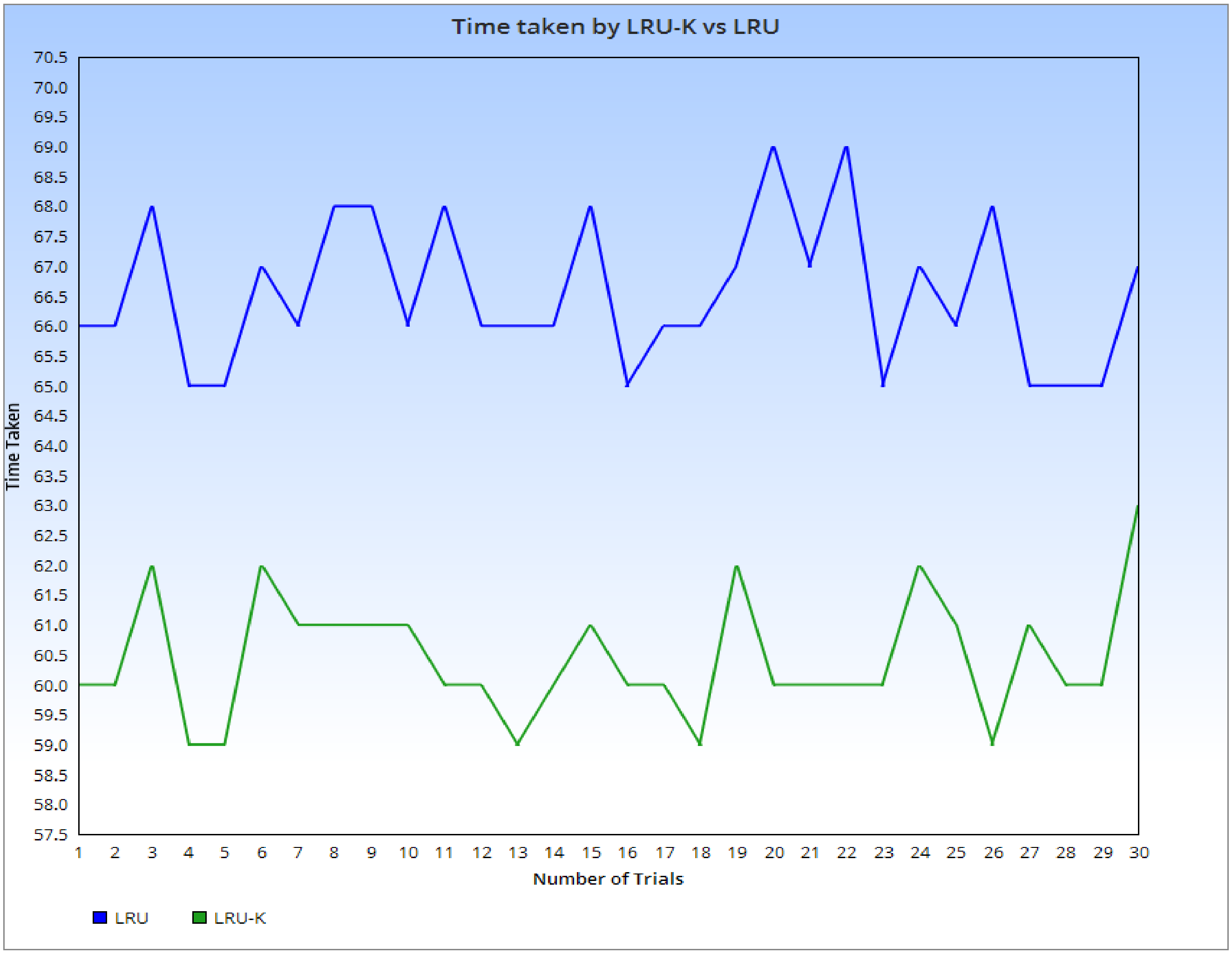
Average Time Taken by LRU and LRU-K (in Seconds)

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>LRU-K)

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 LRU-K.

Setup

2GB RAM. This can be configured in grub file.

3GB File created using dd (`dd if=/dev/urandom of=3g.txt bs=1048576 count=3072`)

LRU-K threshold is set to 10 seconds

Three programs have to be compiled under advproject/test/lru directory

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>'. Number of faults incurred will be printed in console and time taken is redirected to two logs. results-lru.txt is for LRU and results-lruk.txt for LRU-K.

The test scenario is explained in detail in advproject/test/lru/scenario.txt.

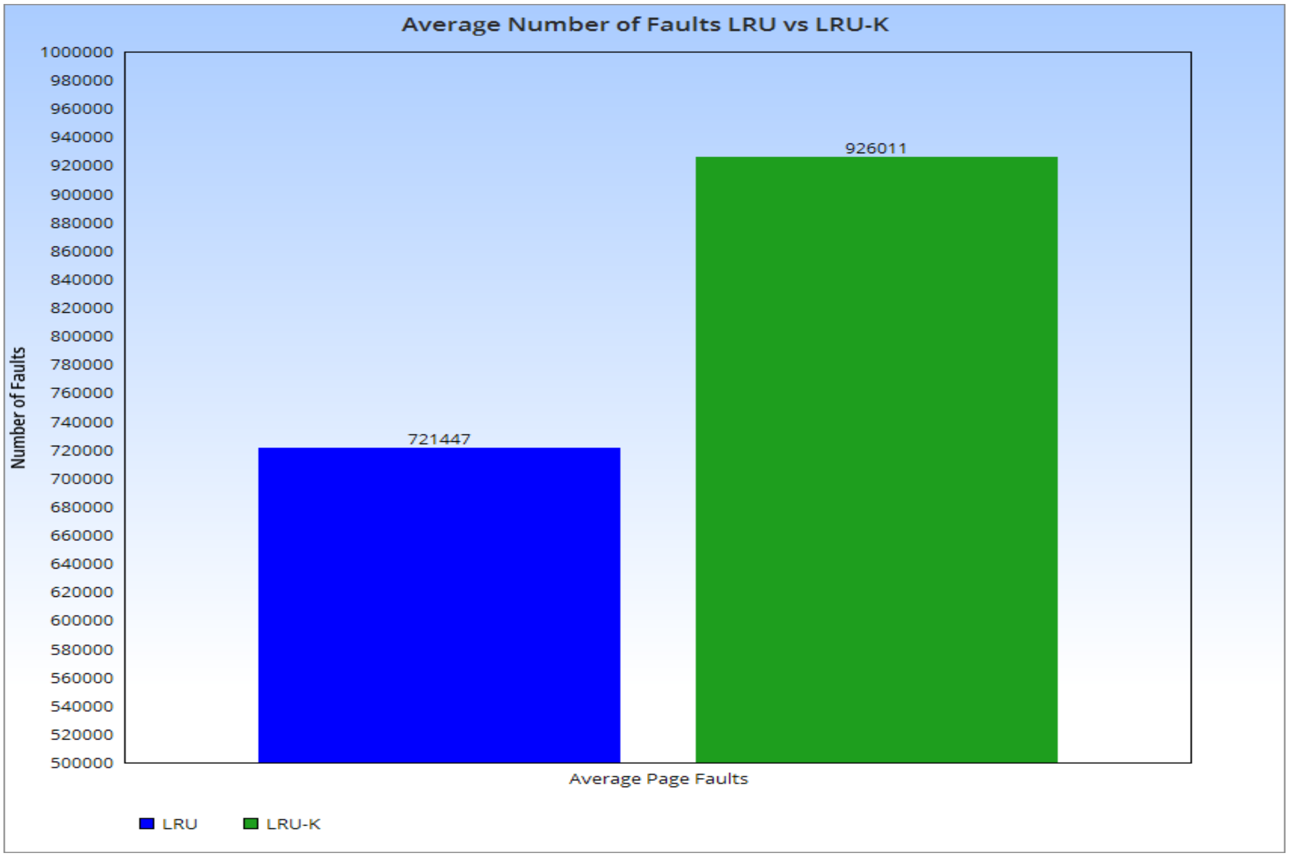
Results

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. The results are computed for around 30 trials.

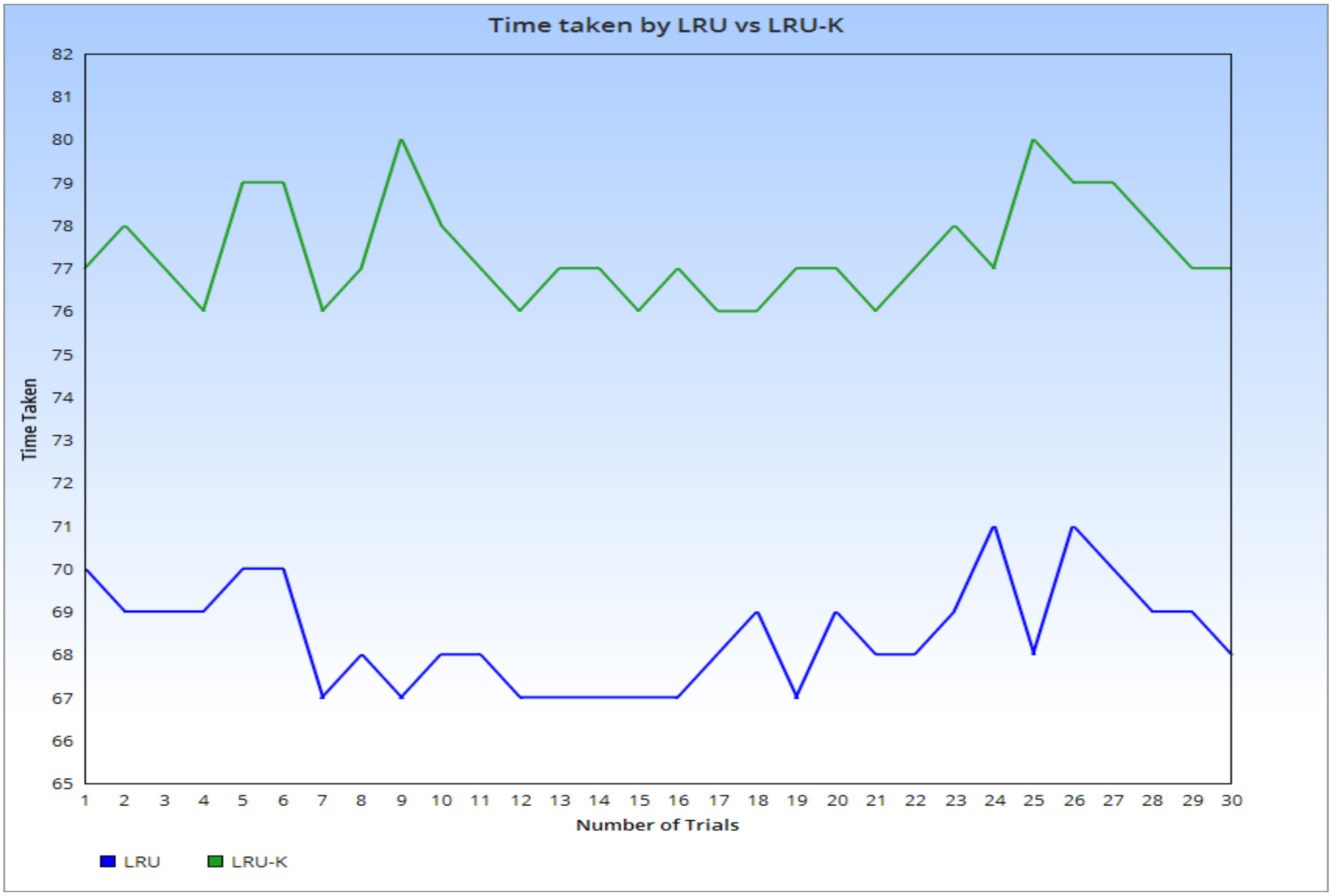
Average Time Taken by LRU and LRU-K (in Seconds)

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 at a conclusion based on one trial.