Daniel Kulenkamp
CSE330 Project 3 Final
April 13, 2018
Professor Syrotiuk

Project 3 Final Submission - Design and Analysis

For the final submission, the Adaptive Replacement Cache (ARC) page replacement algorithm was implemented. Below is a description of the implementation as well as a table including the hit ratios for several cache sizes on the several provided trace files, for both ARC and LRU.

The included Makefile generates an executable named arc.

Usage: ./arc cache_size file_name

Similar to how the LRU algorithm was implemented, this implementation used doubly linked lists (the standard c++ library list) and hash maps (standard c++ library as well). Four separate lists were used with corresponding maps, to maintain the two "lists" in the algorithm. It seemed easier to manage to split the L_1 and L_2 into two separate lists and move elements between them. The algorithm was implemented closely to the pseudocode presented in the paper.

There weren't any major issues during implementation besides typical bugs from mistyping.

Again, as with the LRU implementation, this implementation is fast. This is most likely due to the constant time map lookups, insertion times, and transport times. It ran in tens of seconds on a MacBook Pro and in under a minute on general.asu.edu.

Below is a table of hit ratios for various trace files, for both the LRU implementation and the ARC implementation. Values were rounded to the nearest hundreth.

	OLTP.lis		P3.lis		P6.lis	
c	LRU	ARC	LRU	ARC	LRU	ARC
1024	33.21%	39.26%	1.05%	1.13%	0.71%	0.84%
2048	42.78%	46.39%	1.15%	1.54%	0.86%	1.52%
4096	51.24%	53.09%	1.32%	2.33%	1.09%	3.17%

Table 1: Hit ratios for ARC and LRU for OLTP.lis, P3.lis, P6.lis for cache sizes $c = 1024, 2048, 4096$