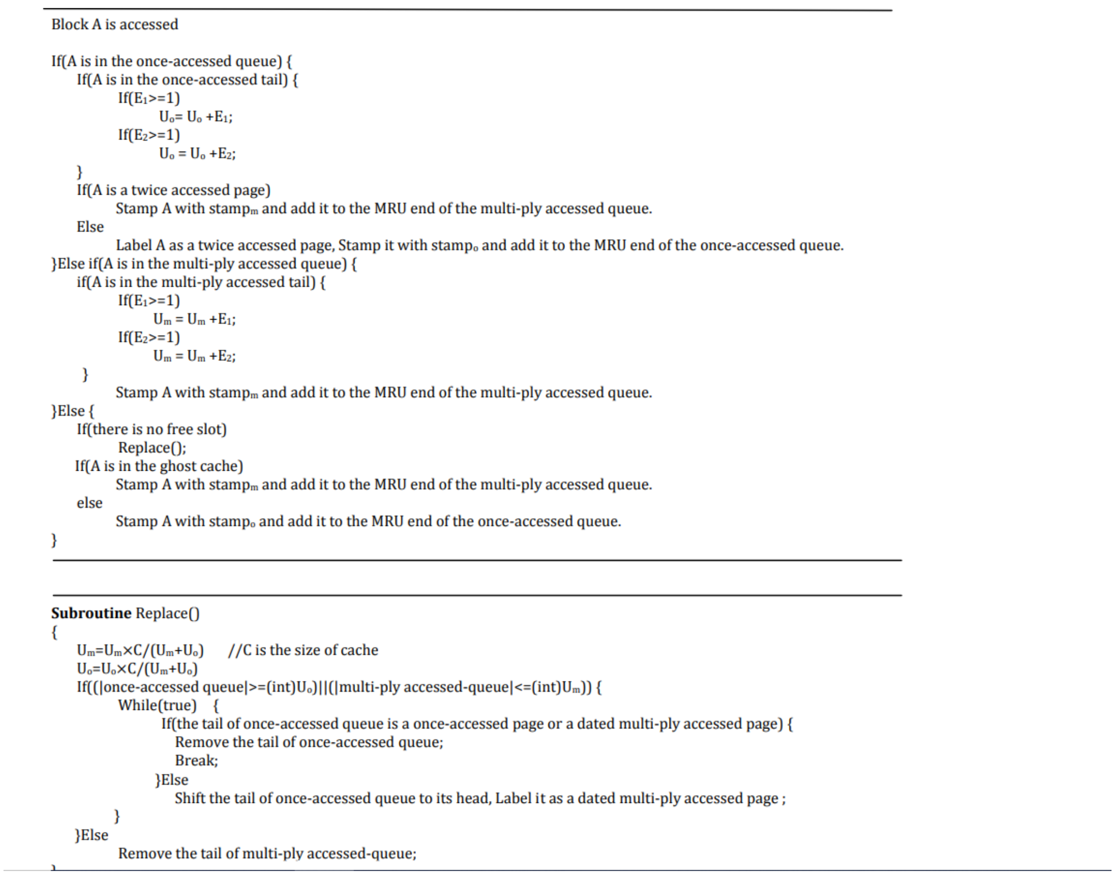
Mobile & Pervasive Computing – Project Report

George Charmanis 2443
George Koffas 2389

Brief Description of the paper: An improvement of the ARC cache replacement policy, a low overhead adaptive policy, called SSARC.

SSARC pseudocode



```
Block A is accessed

If(A is in the once-accessed queue) {
    If(A is in the once-accessed tail) {
        If($E_1$>=1)
            $U_o$= $U_o$ +$E_1$;
        If($E_2$>=1)
            $U_o$ = $U_o$ +$E_2$;
    }
    If(A is a twice accessed page)
        Stamp A with $stamp_m$ and add it to the MRU end of the multi-ply accessed queue.
    Else
        Label A as a twice accessed page, Stamp it with $stamp_o$ and add it to the MRU end of the once-accessed queue.
}Else if(A is in the multi-ply accessed queue) {
    if(A is in the multi-ply accessed tail) {
        If($E_1$>=1)
            $U_m$ = $U_m$ +$E_1$;
        If($E_2$>=1)
            $U_m$ = $U_m$ +$E_2$;
    }
        Stamp A with $stamp_m$ and add it to the MRU end of the multi-ply accessed queue.
}Else {
    If(there is no free slot)
        Replace();
    If(A is in the ghost cache)
        Stamp A with $stamp_m$ and add it to the MRU end of the multi-ply accessed queue.
    else
        Stamp A with $stamp_o$ and add it to the MRU end of the once-accessed queue.
}
```

```
Subroutine Replace()
{
    $U_m$=$U_m$×C/($U_m$+$U_o$)     //C is the size of cache
    $U_o$=$U_o$×C/($U_m$+$U_o$)
    If((|once-accessed queue|>=(int)$U_o$)||(|multi-ply accessed-queue|<=(int)$U_m$)) {
        While(true)    {
            If(the tail of once-accessed queue is a once-accessed page or a dated multi-ply accessed page) {
                Remove the tail of once-accessed queue;
                Break;
            }Else
                Shift the tail of once-accessed queue to its head, Label it as a dated multi-ply accessed page ;
        }
    }Else
        Remove the tail of multi-ply accessed-queue;
}
```

Our approach: We implemented the SSARC algorithm using the Python programming language. We used the SimPy simulation library to create a virtual broadcast link of pages. Each page is a class Page object with an ID and a time stamp of broadcast. Since technical implementation stuff is not specified, like the queues or the ghost cache size, we had to make some assumptions.

Assumptions:

- The once-accessed and the multi-ply accessed queue size is CACHE SIZE (environment variable).

- The ghost cache size is CACHE SIZE / 4. We preferred to have a small ghost cache.

All the sizes in the paper are in KB but in our implementation, we use Bytes.

According to the size table of the paper, the cache size cannot be less than 64 KB, 64B in our case. This is a limitation that also applies to our implementation.

Code Structure: Our code consists of 4 files:

- config.py
- cl_serv_components.py
- BroadcastLink.py
- main.py

config.py: All the environment variables, prototypes of functions and initializations. Also, this file contains the functions used to calculate the values needed in SSARC (emergencies, utilities).

cl_serv_components.py: All the logic and implementation of client and server. Basically, the implementation of SSARC algorithm.

BroadcastLink.py: Some helpful methods that can be used when implementing a broadcast link.

main.py: The main file to run our program.

Implementation Highlights:

There are some vague points in the paper that need to be cleared out:

- The requested pages that are not in any of the queues are stored in the ghost cache. If there is no free slot in the ghost cache, the req. pages are stored in the once-accessed queue.
- If a requested page is already in the ghost cache, it is removed and stored in the multi-ply queue.

Mobile & Pervasive Computing – Project Report

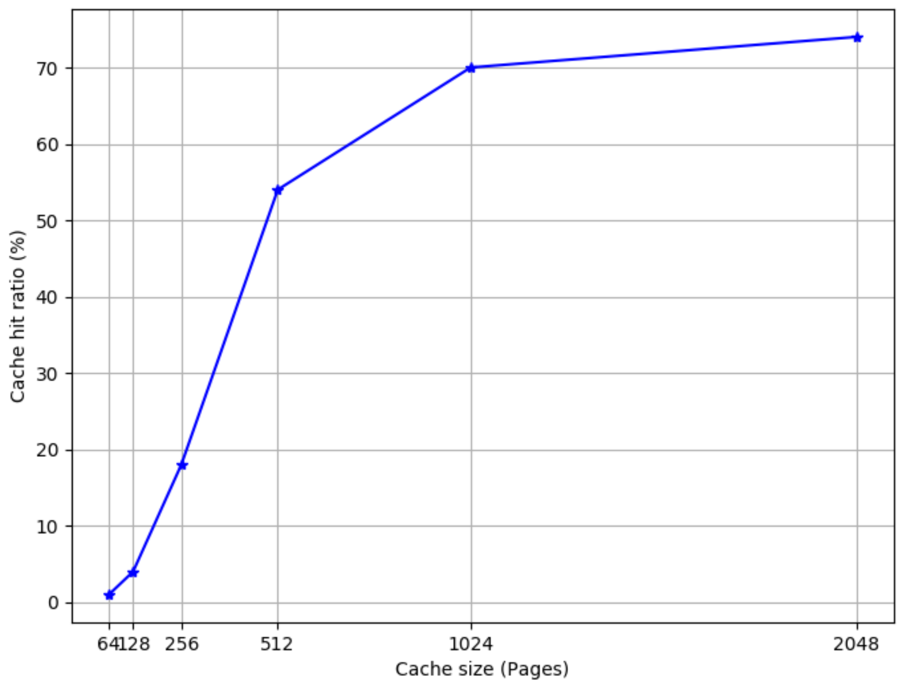
George Charmanis 2443
George Koffas 2389

Experimental Details:

The items/pages that we broadcasted are random, with an equal possibility of being broadcasted. As a result, we cannot expect a tremendous performance since this is not an accurate depiction of a real scenario. The number of pages is CACHE SIZE * 2 + random_number (environment variable). The random number is somewhere between (1,10).

Performance Measurements:

To measure the performance of the algorithm, we calculated the cache hit ratio while changing both the number of the broadcasted pages and the cache size.


Cache hit ratio (%)
Cache size (Pages)