




# 연구논문/작품 중간보고서

2020 학년도 제 2학기

제목	FTL의 Visualization을 위한 Web application 개발	○ 작품 ( ) 논문 ( ) ※ 해당란 체크
평가등급	지도교수 수정보완 사항	팀원 명단
A	<p>○ 입력으로 사용할 .txt 파일의 포맷 및 이를 쉽게 작성할 수 있는 방법에 대해 고민이 필요합니다.</p> <p>○ 시뮬레이션하는 SSD의 용량 및 구성을 조절할 수 있는 옵션이 있으면 좋겠습니다.</p>	<p>심정섭 </p> <p>(학번: 2014312897)</p>

2020 년 9 월 21 일

지도교수 : 서 의 성 서명 

## ■ 요약

제가 현재 졸업 작품으로 진행하고 있는 프로젝트는 'FTL Visualization'입니다. FTL의 동작 과정을 직접 애니메이션 형식으로 보여주는 프로젝트이고, Web application 형태로 제작하고 있습니다. 해당 프로젝트를 진행하는 이유는 FTL의 동작 과정을 직접 눈으로 보여줌으로써 처음 배울 때의 어려움을 덜어주기 위함입니다. 또한 애니메이션의 속도 조절과 동작 별로 끊어서 보기 등 부수적인 기능들도 제공하여 더욱 이해하기 쉽게 설계하였습니다. 개발 언어로는 HTML, CSS, Javascript를 사용하고 있습니다. Page based FTL과 Block based FTL, DFTL 등 다양한 FTL을 포함하여 개발하고 있습니다.

## ■ 서론

우선 작품의 제안배경 및 필요성에 대해 말씀드리겠습니다. 제가 진행하려는 과제는 교육 목적을 위한 FTL의 visualization tool의 제작입니다. tool의 구현을 위해서 Web을 이용할 계획입니다. 각 FTL 별로 해당 구조를 웹 화면에 띄운 뒤, 사용자가 직접 명령을 내려 그 진행 과정을 애니메이션을 통해 순차적으로 보여줄 계획입니다.

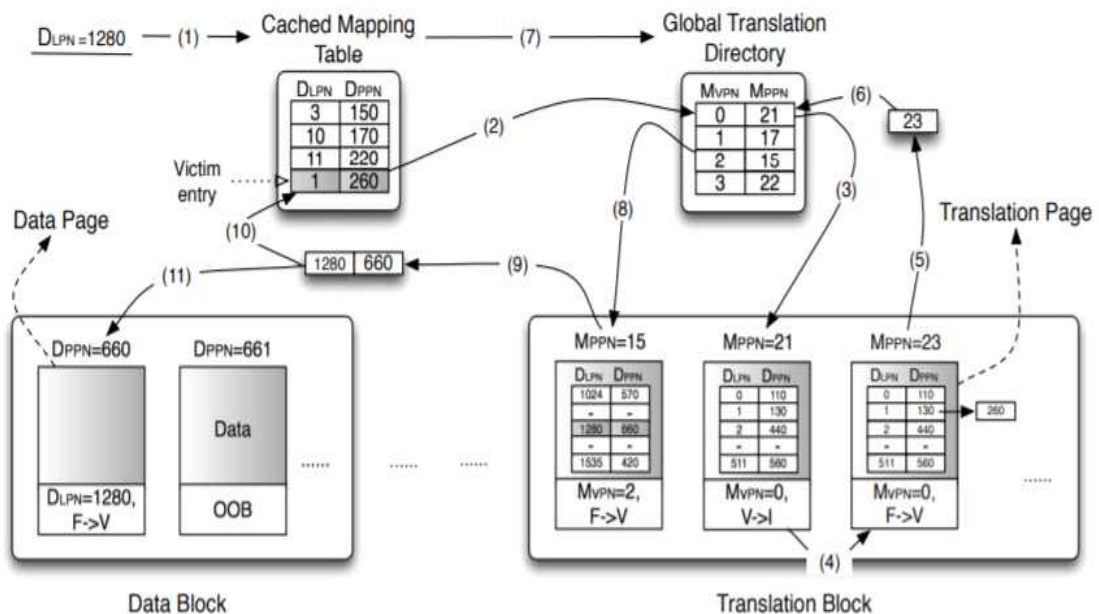


그림 1 DFTL Address Translation Process

위 그림은 DFTL 논문에서 LPN 1280에 대한 요청이 들어왔을 때, 해당 요청의 처리 과정에 대해 설명하는 그림입니다. 실제 지난 학기 임베디드 시스템 설계에 대한 수업을 들을 때에도 해당 그림을 기반으로한 PPT를 통해 DFTL의 동작에 대해 공부하였습니다. 위 그림에는 DFTL의 동작이 상세하게 그려져 있지만, 해당 개념에 대해 처음 배우는 학생에게는 이해하는데 많은 시간과 노력이 요구됩니다. 또한 위 그림을 통하여 DFTL 동작 시 발생할 수 있는 다양한 케이스에 대한 의문점을 해소하기는 쉽지 않습니다. 만일 해당 동작의 진행 과정을 직접 눈으로 보고, 시뮬레이션 해볼 수 있는 tool이 있다면 동작에 대한 이해가 훨씬 쉬어질 것입니다. 그리고 해당 tool을 이용해서 수업의 효율도 올릴 수 있다고 생각합니다.

다음으로 작품의 목표에 대해 말씀드리겠습니다. 우선 해당 프로젝트의 가장 큰 목표는 FTL을 Visualization 함으로써 FTL에 대해 처음 배우는 학생들의 이해를 돕는 것입니다. 또한 FTL에 관한 수업 중에 사용함으로써 설명을 용이하게 하기 위한 목적도 있습니다. 해당 목표를 이루기 위해서 다음의 작은 목표들을 세우고 그것들을 달성하기 위해 노력하였습니다.

첫 번째로 동작 원리에 초점을 맞춰 핵심이 되는 몇몇 요소만을 남겨두고 나머지 부수적인 요소들은 최대한 없애려고 노력하였습니다. 예를 들어 DFTL에서 하나의 CMT slot에 올라가는 LPN의 개수를 1개로 설정함으로써 불필요한 연산들을 제거하였습니다. 하나의 slot에 여러 개의 LPN이 존재하는 경우, 해당 slot을 MAP BLOCK으로 내릴 때 spare에 저장되는 값을 정해줘야 하는 추가적인 고려사항이 생기게 됩니다. 보통은 해당 slot의 첫 번째 LPN을 spare 값으로 저장하지만 구현하는 사람마다 충분히 차이가 생길 수 있는 부분입니다. slot에 올라와 있는 LPN의 개수를 1개로 설정하면, 이와 같은 혼란을 미연에 방지할 수 있습니다. 또한 GTD에서 LPN의 MAP PAGE를 찾을 때에도 해당 LPN 값을 인덱스로 하여 MAP PAGE를 찾을 수 있으므로 이 역시 더욱 직관적인 이해가 가능합니다. 이 외에도 DATA BLOCK과 MAP BLOCK을 구분하여 인터페이스를 구성하는 등 FTL을 단순하게 모델링하기 위한 노력들을 하였습니다.

또한 직관적인 인터페이스를 구현하기 위해서 노력하였습니다. 복잡한 방식의 인터페이스는 사용자가 해당 Tool을 이용하기 위해 많은 부분을 신경쓰게 만듭니다. 이러한 방식은 온전히 FTL의 동작 원리에만 집중하는데 방해가 된다고 생각하여 최대한 간단한 방식으로 인터페이스를 구성하려고 노력하였습니다. 페이지의 상단에 버튼들을 모아놓음으로써 기능들을 한눈에 보기 쉽게 하였고, 버튼들의 이름 또한 직관적으로 알 수 있도록 신경을 썼습니

다. 동작을 명령했을 때의 애니메이션 역시 하이라이트 기능을 활용하여 직관적으로 진행되도록 노력하였습니다. 또한 각 애니메이션의 구분동작 마다 동작에 관련된 간단한 설명을 통해 해당 동작에 대한 이해를 도울 예정입니다.

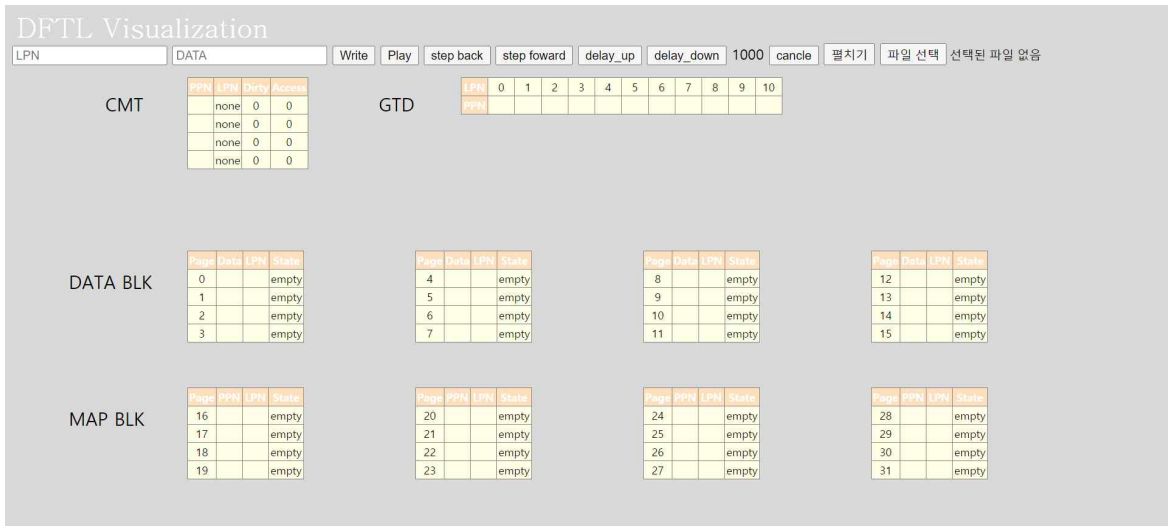


그림 2 DFTL Visualization

마지막으로 프로젝트의 전체적인 Overview에 대해서 설명드리겠습니다. 그림 2는 제가 만든 FTL Visualization 페이지 중 DFTL에 관한 페이지입니다. 상단 가장 왼쪽의 버튼은 Write 버튼으로, 입력 창에 lpn 과 data에 대한 값을 입력한 뒤 버튼을 누르게 되면 해당 입력 값들로 Write Operation이 실행되게 하는 버튼입니다. 그리고 그 오른쪽에는 Play 버튼이 있습니다. 해당 버튼은 애니메이션 진행 중에는 'Pause' 버튼으로 바뀌게 되고 애니메이션을 일시 중지하는 기능을 가집니다. 일시정지가 된 뒤에는 다시 'Play' 버튼으로 바뀌어 해당 버튼을 누르면 다시 애니메이션을 재개하게 됩니다. 그 오른쪽으로는 'step back' 버튼과 'step forward' 버튼이 위치하고 있습니다. 해당 버튼들의 기능은 애니메이션을 동작 별로 끊어서 볼 수 있게 해주는 것입니다. 해당 버튼은 'pause' 상태에서만 사용할 수 있도록 제한을 걸어 놓을 계획입니다. 그리고 그 옆에는 애니메이션의 속도를 조절할 수 있는 'delay\_up', 'delay\_down' 버튼이 있습니다. 해당 버튼들을 누르게 되면 옆에 띄워져 있는 숫자가 변하게 되는데, 숫자가 클수록 애니메이션의 딜레이가 커진다는 것을 의미합니다. 즉, 'delay\_up' 버튼을 눌러 숫자를 크게 만들수록 애니메이션의 속도가 느려지게 되며, 'delay\_down' 버튼을 누를수록 애니메이션의 속도가 빨라지게 됩니다. 'candle' 버튼은 직전에 실행했던 operation을 취소하는 기능을 가지고 있습니다. '펼치기' 버튼을 누르게 되면 다음과 같이 FTL의 목록들이 나타나게 됩니다.

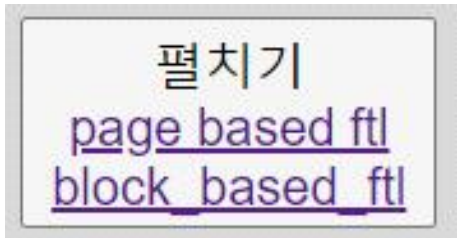


그림 3 펼치기 버튼

현재는 page based FTL과 block based FTL을 구현한 상태이기 때문에 위 목록들이 나오게 됩니다. 목록에 있는 FTL을 누르게 되면 해당 FTL의 Visualization 페이지로 이동하게 됩니다. 마지막으로 '파일 선택' 버튼은 파일로부터 미리 저장되어 있는 Operation들을 불러와서 하나 씩 실행할 수 있게 하는 기능입니다.

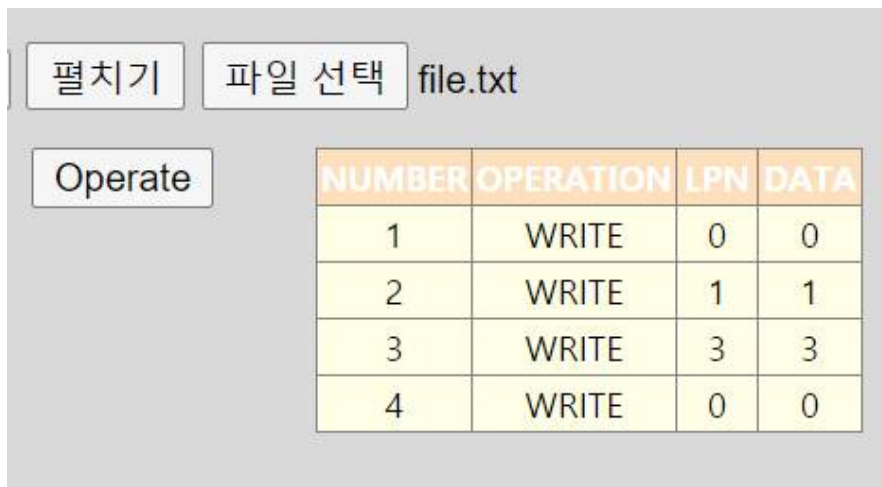


그림 4 파일 선택 버튼 기능

위와 같이 파일에 저장되어 있는 Operation들이 표에 불러와 지게 되며, 'Operate' 버튼을 통해 저장된 명령들을 하나씩 실행할 수 있습니다.

## ■ 관련연구

## Data Structure Visualizations

About  
Algorithms  
F.A.Q  
Known Bugs /  
Feature  
Requests  
Java Version  
Flash Version  
Create Your Own  
/  
Source Code  
Contact

David Galles  
Computer Science  
University of San  
Francisco

Currently, we have visualizations for the following data structures and algorithms:

- Basics
  - Stack: Array Implementation
  - Stack: Linked List Implementation
  - Queues: Array Implementation
  - Queues: Linked List Implementation
  - Lists: Array Implementation (available in java version)
  - Lists: Linked List Implementation (available in java version)
- Recursion
  - Factorial
  - Reversing a String
  - N-Queens Problem
- Indexing
  - Binary and Linear Search (of sorted list)
  - Binary Search Trees
  - AVL Trees (Balanced binary search trees)
  - Red-Black Trees
  - Splay Trees
  - Open Hash Tables (Closed Addressing)
  - Closed Hash Tables (Open Addressing)
  - Closed Hash Tables, using buckets
  - Trie (Prefix Tree, 26-ary Tree)
  - Radix Tree (Compact Trie)
  - Ternary Search Tree (Trie with BST of children)
  - B Trees
  - B+ Trees
- Sorting
  - Comparison Sorting
    - Bubble Sort
    - Selection Sort
    - Insertion Sort
    - Shell Sort
    - Merge Sort
    - Quick Sort
  - Bucket Sort
  - Counting Sort
  - Radix Sort
  - Heap Sort

그림 5 Data Structure Visualizations 메인 화면

위 그림은 저의 FTL Visualzation 프로젝트의 모티브가 된 웹 어플리케이션입니다 [1]. 해당 사이트에서는 기본적인 Array와 Linked list부터 Recursion, Indexing, 그리고 Sorting 알고리즘들까지 다양한 자료 구조의 Visualization을 제공합니다. 데이터베이스 수업을 들을 당시 B+ Tree 자료구조에 대해 배웠었는데, 코너 케이스들에서의 동작이 상당히 까다로웠습니다. 그때 교수님께서 학생들의 이해를 돕기 위해 위 사이트를 소개해주셨고 직접 알고리즘을 시뮬레이션 해보며 이해에 많은 도움을 받을 수 있었습니다. 그래서 추후에 임베디드 시스템 설계 수업에서 FTL에 대해 배울 때에도 이러한 사이트가 있었다면 큰 도움을 받았을 것 같다는 생각을 하게 되어 해당 프로젝트를 졸업 작품 주제로 선정하게 되었습니다.

해당 프로젝트를 진행하기로 결정한 뒤, 각 FTL의 알고리즘들을 리마인드 하는 작업이 필요했습니다. 이 과정에서 임베디드 시스템 설계 수업을 들었던 신동균 교수님의 수업 자료가 올라와 있는 사이트를 참조하였습니다 [2]. 가장 먼저 기본적인 Page based FTL과 Block based FTL을 구현하기 위해 관련 자료들을 참조했습니다. Page based FTL은 PMT(Page Mapping Table)라는 테이블을 통해 각 LPN에 관한 정보를 관리합니다. PMT에서 하나의 slot에서는 하나의 LPN에 대한 PPN 정보가 담기게 되며, 그 정보를 토대로 해당 LPN의 DATA가 적혀 있는 BLOCK을 찾아가는 방식으로 알고리즘이 구성됩니다. LPN의 순서에 상관없이 첫 번째 BLOCK부터 마지막 BLOCK 까지 순

차적으로 정보들을 저장하는 방식이며, FREE BLOCK이 1개가 남게 되었을 때 Garbage Collection을 진행합니다. Garbage Collection은 Greedy 방식과 Cost-Benefit 방식 등 상황에 따라 여러 알고리즘이 존재하며, Victim Block으로 선정된 Block의 Valid Page들을 Free BLOCK으로 옮기는 작업을 수행하게 됩니다. 해당 알고리즘은 PMT를 통해서 빠르게 LPN의 DATA가 담긴 PAGE를 찾을 수 있다는 장점이 있지만, LPN의 정보를 하나하나 관리하다 보니 PMT의 용량이 커지게 됩니다. 이는 PMT를 올려놓는 RAM의 크기가 커야한다는 것을 의미하고, 이는 SSD가 높은 가격으로 책정되어 경쟁성을 잃어버리는 문제를 낳게 됩니다. 또한 Mobile 기기처럼 이동성이 중요하고 RAM의 크기가 한정된 기기에서는 사용할 수 없다는 단점도 있습니다.

이를 보완하기 위하여 등장한 알고리즘이 Block Based FTL입니다. 해당 FTL은 LPN들을 BLOCK 단위로 관리함으로써 RAM에 올려지는 테이블의 크기를 줄이는 방식을 사용합니다. BLOCK Based FTL에서는 PMT 대신 BMT(Block Mapping Table)라고 불리는 테이블이 존재하는데, 해당 테이블을 통해 찾고자 하는 LPN의 DATA가 저장되어 있는 BLOCK의 정보를 알 수 있습니다. 이때 LPN의 정보들은 BLOCK 단위로 묶여서 저장되게 됩니다. 예를 들어서 한 BLOCK의 페이지의 개수가 4개라고 하면, 모든 LPN들은 4개 단위로 묶여서 같은 BLOCK에 저장되게 됩니다. 따라서 LPN 0번부터 LPN 3번이 같은 BLOCK에 저장되게 되고, LPN 4번부터 LPN 7번이 같은 BLOCK에 저장되는 방식으로 작동하게 됩니다. 이를 통해 BMT에서는 LPN의 개수 만큼의 slot을 만들지 않아도 되고,  $(\text{LPN의 개수} / \text{BLOCK 당 페이지의 개수})$  만큼의 slot만 만들면 되므로 그 크기가 현저하게 줄어들게 됩니다. 데이터의 저장은 LPN의 값을 offset으로 사용하여 저장되게 됩니다. 예를 들어 4개의 페이지가 있는 BLOCK에서는 LPN 0번이 첫 번째 페이지에 저장되게 되고, LPN 1번이 두 번째 페이지, LPN 2번이 세 번째 페이지에 저장되는 방식입니다. 그런데 만일 LPN 2번의 정보를 저장한 상태에서 다시 LPN 0번의 정보를 저장하려고 하면, NAND Flash의 특성상 순차적인 Write 동작 밖에 할 수 없기 때문에 정보를 저장할 수 없는 문제가 발생합니다. BLOCK Based FTL에서는 이 상황을 Garbage Collection을 통해 새로운 Free BLOCK에 데이터를 적어주는 방식으로 해결합니다. 기존의 LPN 0번의 정보가 담긴 페이지를 invalid 처리한 후, 새로운 BLOCK에 갱신되는 LPN 0번의 데이터를 적어주고 나머지 Valid 페이지들을 옮겨주게 되면 다른 정보의 변경 없이 LPN 0번의 정보를 갱신할 수 있게 됩니다. 페이지의 이동이 끝난 뒤에는 원래의 BLOCK을 EMPTY 상태로 만들어 다시 FREE BLOCK으로 사용할 수 있게끔 합니다. 이

렇게 함으로써 Block Based FTL에서는 기존의 Page Based FTL의 단점이던 PMT의 용량을 줄이고, 한번의 BMT 테이블 참조만으로도 LPN의 데이터에 접근할 수 있게 됩니다. 하지만 LPN들을 BLOCK 단위로 묶어서 관리하기 때문에 잦은 GC 연산으로 인한 response time의 증가가 발생할 수 있습니다. 따라서 해당 FTL은 RAM의 용량이 극도로 제한적인 환경에서 적합하게 사용될 수 있는 알고리즘입니다.

다음으로 등장한 것은 Page Based FTL과 Block Based FTL의 특성을 합친 Hybrid Mapping 방식의 FTL 알고리즘입니다. 제가 진행하는 프로젝트에서는 그 중에서도 BAST(or Log Block Scheme)을 구현할 계획입니다. Hybrid Mapping 방식은 small write의 handling이 용이한 Page Mapping 방식의 장점과 Translation Table의 Overhead가 적은 Block Mapping 방식의 장점을 합친 방식의 FTL입니다. BAST는 hybrid mapping의 초기 버전으로, Block mapping을 관리하는 Data Block들과 Page Mapping을 관리하는 Log Block들로 구성됩니다. 하나의 Log Block은 하나의 Data Block들과 1 to 1으로 Mapping 되며, Data Block에서 업데이트 된 정보들을 관리합니다. 이렇게 함으로써 기존에 Block Based Mapping에서 LPN의 업데이트가 일어날 때마다 GC 동작을 실행해야 했던 Overhead를 줄일 수 있게 됩니다. 그리고 Log Block의 개수를 제한적으로 운용함으로써 기존의 Page Based Mapping에서 Translation Table에 많은 용량이 필요했던 부분을 어느 정도 완화할 수 있게 됩니다. 해당 FTL의 GC 방식은 Switch Merge, Partial Merge, Full Merge 이렇게 총 3가지 방식이 존재하게 됩니다. Switch Merge에서는 Log block을 그대로 Data Block으로 바꾸는 동작을 실행합니다. Switch Merge는 Mapping되는 Data Block의 모든 페이지들이 invalid 혹은 empty 상태이고, Log block에 LPN들이 순차적으로 저장되어 있을 때만 사용 가능합니다. 다음으로 Partial Merge 동작은 Data Block의 Valid 페이지를 Log Block으로 이동시킨 뒤, Log Block을 Data Block으로 대체하는 동작입니다. Partial Merge는 Switch Merge와 마찬가지로 Log Block의 LPN들이 순차적으로 저장되어 있고, Data Block의 Valid page가 존재하고 있을 때 사용합니다. 마지막으로 Full Merge는 Log Block에 LPN들이 순서가 뒤섞여 저장되어 있을 때 사용하게 되는데, 아예 새로운 Free Block을 하나 선정하여 해당 Data block과 Log Block에 있는 Valid page 들을 순서에 맞게 옮겨준 뒤 두 Block들을 모두 Free Block으로 바꾸게 됩니다. Full Merge는 Switch Merge와 Partial Merge가 모두 불가능할 때 사용하게 되며, 가장 큰 Overhead를 갖게 됩니다.



마지막으로 수업때 가장 높은 비중으로 배웠던 DFTL에 대해서도 관련 연구를 조사하였습니다. DFTL은 Deman-based FTL의 약자로 2009년 ASPLOS 학회지를 통해 처음 소개되었습니다 [3]. DFTL은 수업 당시 교수님께서 아직도 현업에서 사용 중이라고 말씀하실 정도로 중요도가 높은 알고리즘이었습니다. DFTL은 CMT와 GTD 라는 Translation Table을 사용하고, Block들을 Data Block과 Map Block 2 그룹으로 나누어 사용합니다. CMT는 최근에 참조 되었던 LPN들을 Caching 해놓은 Table입니다. RAM의 크기가 제한된 상태에서 Flash를 거치지 않고 빠르게 데이터를 참조하기 위해 사용됩니다. CMT의 크기는 매우 작기 때문에 대다수의 LPN들은 CMT에 Caching 되어 있지 않습니다. 다른 LPN들을 참조하기 위해서는 GTD를 통해 해당 LPN의 정보가 저장되어 있는 Map Block의 페이지를 찾아야 합니다. 해당 Map Block 페이지에는 LPN에 대한 실제 PPN 정보가 저장되어 있습니다. 그 PPN은 Data Block에 존재하게 되고, 해당 PPN을 참조함으로써 실제 LPN의 DATA를 얻을 수 있습니다. DFTL은 Computer Science에서 주요한 아이디어 중 하나인 Temporal Locality와 Spatial Locality를 활용하는 알고리즘입니다. 따라서 최근에 참조된 LPN과 연속된 LPN들의 데이터들을 함께 CMT에 Caching 해 놓고, 다음 번에 다시 참조되었을 때 빠르게 참조할 수 있게끔 합니다. CMT에 Caching 되어 있지 않은 LPN에 대해서는 Data를 얻기 위해 2번의 Flash Memory 참조를 거쳐야 하는 Overhead가 있지만 실제 환경에서는 한번 참조된 LPN과 그 근처의 LPN들을 반복하여 참조하는 경우가 대다수이기 때문에 실제 제품에서 활용되고 있다고 알고 있습니다.

이렇게 Page Based FTL, Block Based FTL, Hybrid Mapping FTL, 그리고 DFTL 까지 총 4개의 FTL에 대한 관련 연구를 조사하였고, 이를 토대로 FTL Visualization 프로젝트를 진행하고 있습니다.

## ■ 제안 작품 소개

FTL Visualization 프로젝트는 HTML, CSS, Javascript를 사용해서 구현하였습니다. 각 FTL 알고리즘의 구현은 실제 연산이 진행되는 객체와 애니메이션의 정보를 저장하는 객체를 따로 두어 step back/foward와 cancel 등 부수적인 기능들의 구현을 용이하게 하였습니다. Page Based FTL을 예시로 들어 FTL Visualization의 전반적인 구현 세부사항에 대해 설명 드리겠습니다. Page Based FTL에서는 stat\_save\_arr라는 이름의 Array를 선언하여 FTL의 모든 명령들을 관리하게 됩니다. Write 또는 Read 하나의 명령에 관한 정보들이 stat\_save\_arr의 하나의 인덱스에 담기게 됩니다. stat\_save\_arr 배열 안의 각

인덱스는 하나의 Object를 할당 받아 정보들을 관리하게 됩니다. 각 Object 안에 저장되는 정보로는 PMT의 PPN에 관한 정보들, 그리고 각 페이지별 저장된 LPN과 데이터, state에 관한 정보들이 저장됩니다. 또한 GC 연산을 위해서 항상 하나의 Free Block을 남겨두어야 하기 때문에 Free Block으로 선정된 블록의 인덱스 정보를 가지고 있습니다. 마지막으로 현재 write 해야 할 페이지의 인덱스에 관한 정보도 가지고 있습니다. stat\_save\_arr에는 FTL의 현재 상태에 대한 정보들과 별개로 각 연산을 수행한 뒤, 해당 연산에 매칭되는 애니메이션의 정보를 따로 저장하게 됩니다. 애니메이션의 정보는 우선 motion이라는 변수에 해당 애니메이션의 이름을 저장하고, 각 motion 별로 필요한 정보들을 따로 저장해 두게 됩니다. 하나의 Operation이 끝나고 난 뒤에는 animation\_func 함수가 불리면서 미리 저장해 두었던 애니메이션 정보들을 순차적으로 하나씩 실행시키게 됩니다. 다음으로 각 버튼들의 기능 구현에 관해 설명 드리겠습니다.

먼저 Write 기능의 구현에 대해 설명하겠습니다. 입력 창에 LPN과 DATA 값을 입력하고 Write 버튼을 누르게 되면 해당 LPN과 DATA를 가지고 ftl\_write 함수가 실행되게 됩니다. ftl\_write 함수에서는 우선 stat\_save\_arr의 인덱스를 하나 증가시킨 뒤에 해당 인덱스에 직전의 인덱스의 정보들을 복사하는 연산을 수행합니다. 그러고나서는 GC 조건을 체크한 뒤 조건을 만족하게 되면 garbage\_collection 함수를 호출하게 됩니다. 해당 함수에서는 우선 각 블록 별로 invalid page의 개수를 센 뒤, 가장 많은 invalid page를 가진 블록을 victim으로 선정합니다. 그리고 해당 블록을 순회하면서 각 페이지를 empty 상태로 만든 뒤, 'make empty' motion을 저장하게 됩니다. 해당 motion에는 victim page의 번호가 추가적인 정보로 함께 저장되게 됩니다. 만일 해당 페이지가 valid page일 경우에는 Free 블록으로 페이지를 옮겨주어야 합니다. 페이지를 옮긴 뒤에는 'copy page'라는 이름으로 motion을 저장하고 원래 페이지의 정보들과 옮겨질 페이지의 넘버를 같이 저장합니다. 그런 뒤에 PMT에서 옮겨진 페이지에 닮긴 LPN에 대한 정보를 바꿔줍니다. 마찬가지로 해당 연산을 마친 뒤에는 'change pmt' motion을 애니메이션으로 저장합니다. garbage\_collection 함수가 끝난 뒤에 PMT를 확인하여 새로운 데이터를 쓰고자 하는 LPN에 매핑되어 있는 PPN이 있는지 확인합니다. 만일 존재한다면 해당 PPN의 상태를 invalid 상태로 바꿔줍니다. 그런 뒤에는 'make invalid' motion을 애니메이션으로 저장합니다. 그리고 나서는 PMT에서 LPN의 매핑된 PPN의 값을 새로운 페이지의 넘버로 바꾼 뒤 새로운 페이지에 DATA를 적고, LPN 값을 저장하고, 상태를 valid로 바꿔줍니다. 마찬가지로

각각의 연산들은 'change pmt', 'write data', 'write spare', 'make valid'라는 이름의 motion으로 저장되게 됩니다. 여기까지 Write 명령의 모든 연산을 마친 뒤에는 animation\_func 함수를 불러서 미리 저장해 놓았던 motion 들을 순차적으로 실행하게 됩니다. Javascript의 setInterval 라이브러리를 이용하여 animation\_func 함수가 매 delay 시간 마다 불리도록 설정해 놓습니다. animation\_func 함수에서는 우선 이번에 실행 되어야 할 motion의 이름을 확인합니다. motion의 이름이 'make invalid'인 경우에는 우선 invalid 되어야 할 페이지를 'focus'라는 class에 추가합니다. 해당 클래스에 추가된 객체는 색이 변하면서 하이라이트 되게 됩니다. 그리고 나서는 setTimeout 라이브러리를 이용하여 일정 시간 뒤에 'focus' class를 제거하면서 동시에 해당 페이지의 state를 'invalid'로 바꾸게 됩니다. 'make empty' motion의 경우는 'make invalid'와 마찬가지로 해당 페이지를 'focus' class에 추가한 뒤, setTimeout을 이용해 class를 제거하면서 해당 페이지의 state를 'empty'로 바꿔 줍니다. 다른 motion들 또한 마찬가지로 'focus' class를 이용하여 하이라이트 한 뒤, 일정 시간 후에 객체의 정보를 바꾸는 방법으로 애니메이션을 진행시키게 됩니다.

다음으로는 'play' 버튼의 기능 구현에 대해 설명 드리겠습니다. 해당 버튼이 눌렸을 때 먼저 현재 애니메이션이 진행 중인지 아니면 멈춰 있는 상태인지 판단하여야 합니다. 만일 애니메이션이 진행 중인 상황이면 버튼의 이름을 'pause'로 바꾸고 clearInterval 라이브러리를 이용하여 animation\_func 함수에 걸어 놓은 setInterval을 해제시킵니다. 그렇게 되면 더 이상 애니메이션이 진행되지 않으면서 멈춘 상황이 됩니다. 그 상황에서 다시 버튼을 누르게 되면 버튼 이름을 다시 'play'로 바꾸고 setInterval 라이브러리를 이용해서 다시 animation\_func이 매 delay 마다 불리도록 해줍니다.

다음은 step back / step forward 버튼에 대해 설명하겠습니다. 우선 step back 버튼이 눌리게 되면 step\_back\_func 함수가 호출되게 됩니다. 해당 함수에서는 현재의 stat\_save\_arr 인덱스에 저장되어 있는 PMT 객체와 Block 객체들을 모두 remove 시켜 줍니다. 그리고 나서는 직전의 인덱스에 저장되어 있던 PMT 객체와 Block 객체들을 붙여 넣기 하게 됩니다. 그렇게 되면 화면에 표시되는 PMT와 Block들은 한 단계 전 motion의 상태로 돌아가게 됩니다. step forward 버튼은 step back 버튼과 반대로 motion을 한 단계 진행시키게 됩니다. step\_forward\_func 에서는 간단하게 animation\_func을 한번 호출하는 방식으로 motion을 한 단계만 진행시키게 됩니다.

delay\_up / delay\_down 버튼은 단순히 미리 선언해 놓았던 delay 변수의 값

을 100 만큼 증가시키고 감소시키는 연산을 수행합니다. 해당 delay 변수는 animation\_func 함수의 호출 간격을 결정하는 변수로서 사용됩니다.

cancel 버튼은 현재 상태에서 PMT 객체와 Block 객체들을 모두 remove 시킨 뒤, LPN 인덱스를 처음 Block이 진행되기 전으로 바꾸고 다시 PMT 객체와 Block 객체들을 붙여 넣습니다. 그렇게 하면 현재 진행 중인 명령의 motion 들이 모두 취소되고 명령이 들어오기 직전의 상태로 돌아가게 됩니다.

마지막으로 '파일 선택' 버튼의 기능의 구현에 관해 설명 드리겠습니다. 해당 버튼은 type 속성을 'file'로 설정하고 accept 속성을 'text/plain'으로 설정하여 클릭시 내 컴퓨터에 저장되어 있는 파일을 선택하는 화면이 나오고, 파일 선택 후에 확인 버튼을 누르면 openFile 함수가 불리도록 설정되어 있습니다. openFile 함수에서는 FileReader() 객체를 통해 해당 파일의 내용들을 읽어 오게 됩니다. 읽어온 내용들은 split 메소드를 사용하여 스페이스 바와 엔터 기준으로 모두 잘라줍니다. 그런 뒤에는 나뉘어진 단어들을 하나씩 읽으면서 read 연산인지 write 연산인지 확인하는 과정을 거칩니다. 확인한 뒤에는 Operation 객체에 해당 명령들을 하나씩 담아 주게 됩니다. 추후에 'Operate' 버튼이 눌리게 되면 해당 연산에 따라 ftl\_write 혹은 ftl\_read 함수를 호출하는 방식으로 구현하였습니다.

## ■ 구현 및 결과분석

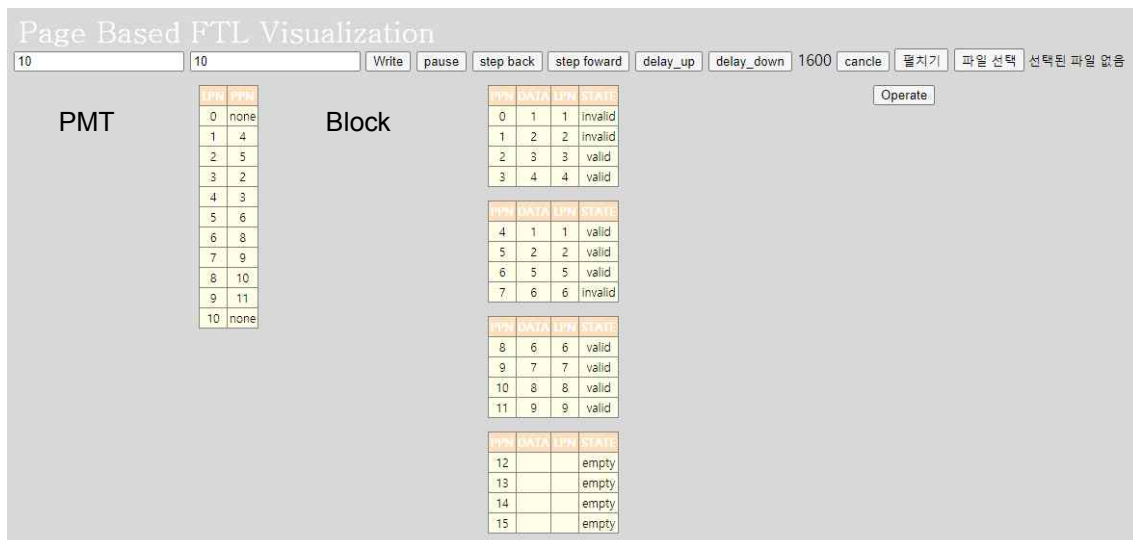


그림 6 Page Based FTL Visualization

Page Based FTL의 위 상황에서 LPN에 10, DATA에 10의 입력 값을 넣은 상태로 Write 버튼을 눌렀을 때의 결과에 대해 설명 드리겠습니다. Free 블록이 1개 밖에 남지 않은 상황이기 때문에 Write 버튼을 누를 시에

Garbage\_collection 함수가 불리게 됩니다. 첫 번째 블록의 invalid 페이지의 개수가 2개로 가장 많기 때문에 해당 블록이 Victim으로 선정되게 됩니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel 끝내기 파일 선택 선택된 파일 없음

Operate

LPN	PPN
0	none
1	4
2	5
3	2
4	3
5	6
6	8
7	9
8	10
9	11
10	none

PMT

Block	LPN	DATA	LPN	STATE
0				empty
1	2	2		invalid
2	3	3		valid
3	4	4		valid
4	1	1		valid
5	2	2		valid
6	5	5		valid
7	6	6		invalid
8	6	6		valid
9	7	7		valid
10	8	8		valid
11	9	9		valid
12				empty
13				empty
14				empty
15				empty

그림 7 page Based FTL: make empty

먼저 첫 번째 페이지는 invalid page이기 때문에 위처럼 'make empty' motion이 실행되어집니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel 끝내기 파일 선택 선택된 파일 없음

Operate

LPN	PPN
0	none
1	4
2	5
3	2
4	3
5	6
6	8
7	9
8	10
9	11
10	none

PMT

Block	LPN	DATA	LPN	STATE
1				empty
2	3	3		valid
3	4	4		valid
4	1	1		valid
5	2	2		valid
6	5	5		valid
7	6	6		invalid
8	6	6		valid
9	7	7		valid
10	8	8		valid
11	9	9		valid
12				empty
13				empty
14				empty
15				empty

그림 8 page Based FTL: make empty2

다음 페이지도 마찬가지로 invalid page 이기 때문에 'make empty' motion 만 실행되어 페이지를 비워줍니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel null기 파일 선택 선택된 파일 없음 Operate

**PMT**

LPN	PPN
0	none
1	4
2	5
3	2
4	3
5	6
6	8
7	9
8	10
9	11
10	none

**Block**

PPN	DATA	LPN	STAT
0			empty
1			empty
2			empty
3			none
4	1	1	valid
5	2	2	valid
6	5	5	valid
7	6	6	invalid
8	6	6	valid
9	7	7	valid
10	8	8	valid
11	9	9	valid
12	3	3	valid
13			empty
14			empty
15			empty

그림 9 Page Based FTL: make empty & copy page

세 번째 페이지는 valid page 였으므로 'make empty'를 통해 페이지를 비워준 뒤, 'copy page' 모션을 통해 해당 페이지를 Free 블록의 첫 번째 페이지로 이동시키게 됩니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel null기 파일 선택 선택된 파일 없음 Operate

**PMT**

LPN	PPN
0	none
1	4
2	5
3	12
4	2
5	6
6	8
7	9
8	10
9	11
10	none

**Block**

PPN	DATA	LPN	STAT
0			empty
1			empty
2			empty
3	4	4	valid
4	1	1	valid
5	2	2	valid
6	5	5	valid
7	6	6	invalid
8	6	6	valid
9	7	7	valid
10	8	8	valid
11	9	9	valid
12	3	3	valid
13			empty
14			empty
15			empty

그림 10 Page Based FTL: change ppn

페이지를 옮긴 뒤에는 'change ppn' motion이 실행되어 PMT의 LPN 3에 대한 PPN 정보를 Free page의 PPN으로 변경시켜 줍니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel 물치기 파일 선택 선택된 파일 없음

Operate

**PMT**

LPN	PPN
0	none
1	4
2	5
3	12
4	3
5	6
6	8
7	9
8	10
9	11
10	none

**Block**

PPN	DATA	LPN	STATE
0			empty
1			empty
2			empty
3			empty
4	1	1	valid
5	2	2	valid
6	5	5	valid
7	6	6	invalid
8	6	6	valid
9	7	7	valid
10	8	8	valid
11	9	9	valid
12	3	3	valid
13	4	4	valid
14			empty
15			empty

그림 11 Page Based FTL: make empty & copy page2

네 번째 페이지도 세 번째 페이지와 마찬가지로 해당 페이지를 empty 시킨 뒤에 'copy page' 모션을 이용하여 Free 블록의 두 번째 페이지로 이동 시켜 줍니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel 물치기 파일 선택 선택된 파일 없음

Operate

**PMT**

LPN	PPN
0	none
1	4
2	5
3	12
4	13
5	6
6	8
7	9
8	10
9	11
10	none

**Block**

PPN	DATA	LPN	STATE
0			empty
1			empty
2			empty
3			empty
4	1	1	valid
5	2	2	valid
6	5	5	valid
7	6	6	invalid
8	6	6	valid
9	7	7	valid
10	8	8	valid
11	9	9	valid
12	3	3	valid
13	4	4	valid
14			empty
15			empty

그림 12 Page Based FTL: change ppn2

그런 뒤에는 LPN 4에 매핑되는 PPN 값을 13으로 변경시켜줍니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel 끝치기 파일 선택 선택된 파일 없음 Operate

**PMT**

LPN	PPN
0	none
1	4
2	5
3	12
4	13
5	6
6	8
7	9
8	10
9	11
10	14

**Block**

PPN	DATA	LPN	STATE
0			empty
1			empty
2			empty
3			empty
4	1	1	valid
5	2	2	valid
6	5	5	valid
7	6	6	invalid
8	6	6	valid
9	7	7	valid
10	8	8	valid
11	9	9	valid
12	3	3	valid
13	4	4	valid
14			empty
15			empty

그림 13 Page Based FTL: change ppn3

Garbage\_collection 함수를 모두 마친 뒤에는 원래 명령이었던 LPN 10에 대한 Write 연산을 실행시킵니다. 우선 위 그림처럼 LPN 10에 매핑되는 PPN을 새로 쓰여질 페이지인 14로 바꿔줍니다.

Page Based FTL Visualization

10 10 Write pause step back step forward delay\_up delay\_down 1600 cancel 끝치기 파일 선택 선택된 파일 없음 Operate

**PMT**

LPN	PPN
0	none
1	4
2	5
3	12
4	13
5	6
6	8
7	9
8	10
9	11
10	14

**Block**

PPN	DATA	LPN	STATE
0			empty
1			empty
2			empty
3			empty
4	1	1	valid
5	2	2	valid
6	5	5	valid
7	6	6	invalid
8	6	6	valid
9	7	7	valid
10	8	8	valid
11	9	9	valid
12	3	3	valid
13	4	4	valid
14	10	10	valid
15			empty

그림 14 Page Based FTL: write data & write spare & make valid

마지막으로 위 그림처럼 14번 페이지에 입력 값으로 들어온 LPN과 DATA를 적어주고 해당 페이지의 상태를 'valid'로 바꾼 뒤 모든 애니메이션을 종료하게 됩니다.



## ■ 결론

FTL Visualization 프로젝트를 통해 추후에 FTL에 대해 배우게 될 학생들의 이해를 도울 수 있을 것 같습니다. 학생들은 Write와 Read 버튼을 통해 직접 자신이 설정한 LPN과 DATA를 넣어가며 저장 과정을 애니메이션 형식으로 볼 수 있게 됩니다. step back / forward 버튼을 통해서 애니메이션의 진행 과정을 구분 동작으로 보며 코너 케이스의 동작에 대해서도 상세한 과정을 지켜 볼 수 있게 될 것입니다. 이는 분명히 FTL에 대한 이해도 향상으로 이어질 것이라고 생각합니다. 그리고 '파일 선택' 버튼을 통해 미리 저장되어 있던 명령들을 불러와, Operation 버튼을 통해 하나씩 실행 시켜볼 수도 있게 될 것입니다. 이를 통해 학생들은 직접 명령을 하나하나 입력하는 수고를 덜 수 있을 것이라고 생각합니다. 또한 수업 중에 학생들에게 미리 준비된 인풋들을 가지고 설명을 할 때에도 용이하게 사용될 수 있을 것 같습니다.

## ■ 참고문헌

- [1] Data Structure Visualizations,  
<https://www.cs.usfca.edu/~galles/visualization/Algorithms.html>
- [2] ESLAB, [http://nyx.skku.ac.kr/?page\\_id=2303](http://nyx.skku.ac.kr/?page_id=2303)
- [3] A.Gupta et al., "DFTL:A Flash Translation Layer Employing Demand-based Selective Caching of page-level Address Mappings", ASPLOS, 2009.