PMS: 모바일 임베디드 시스템의 소프트웨어 스트리밍 서비스를 위한 확률 기반 다중 접근 블록 선인출 알고리즘

(PMS: Probability-based Multi Successor Prefetch Algorithm for Software Streaming Services of Mobile Embedded Devices)

이 영 재 † 박 선 영 † 박 은 지 †
(Youngjae Lee) (Seonyeong Park) (Eunji Pak)
이 대 우 † 정 욱 † 김 진 수 ††
(Daewoo Lee) (Wook Jung) (Jinsoo Kim)

요 약 최근 PDA의 대중화, 텔레메틱스 산업의 발전에 따라 제한된 저장장치를 갖는 모바일 임베디드 시스템에서 PC와 같은 다양한 소프트웨어를 사용하고자하는 수요가 늘어나고 있다. 그에 따라 소프트웨어 스트리밍 서비스의 필요성이 증가하고 있으나 소프트웨어를 속도가 느린 무선 네트워크를 통해 블록 단위로 전송받아 실행속도가 느린 문제점이 있다. 그리하여 이를 보완해주는 선인출 알고리즘이 필요하다.

본 논문에서는 기존에 연구된 선인출 알고리즘인 최근 접근 블록 알고리즘(LS)과 PPM 기반 알고리즘을 소프트웨어 스트리밍 서비스에 적용시켜 성능을 측정하고 분석한 결과를 토대로 고안된 확률 기반 다중 접근 블록(PMS) 알고리즘을 제안한다. LS의 적중률은 60%정도로 낮지만 메모리 사용량이 적다. 그에 반해 PPM 기반 알고리즘은 메모리 사용량은 많지만 96%이상의 높은 적중률을 보인다. PMS는 블록 단위의 소프트웨어 스트리밍 서비스의 특징과 PPM 기반 알고리즘의 특성을 이용하여 LS의 단점을 보완해 N개의 접근 블록을 확률을 기반으로 저장하고 선인출에 이용한다. 이러한 PMS는 보다 적은 공간오버헤드를 가지면서 PPM 기반 알고리즘과 비슷한 적중률을 나타내 높은 메모리 효율을 나타낸다.

키워드 : 선인출, 접근블록, 스트리밍, 소프트웨어 스트리밍

Abstract *As the demand of employing various PC software on mobile embedded devices which* have limited storages has been increased, software streaming services are needed. However it takes too much time to launch software on them because it is transferred through wireless networks. To address this problem, prefetch algorithms are needed.

We examined 'Last successor (LS)' algorithm and PPM-based prefetch algorithm as prefetch algorithms for software streaming services. We present 'Probability-base Multi Successor (PMS)' algorithm which is contrived through analyzing evaluations of previous algorithms and characteristics of software streaming services. While LS has one successor per each block, PMS has N successors based on probability which is calculated by PPM-based prefetch algorithm. The hit rate of PMS is similar to that of PPM-base prefetch algorithm and the space overhead is similar to that of LS. We can get good efficiency at the point of memory usage when PMS is applied to software streaming services.

Key words : prefetch, successor, streaming, software streaming

· 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음 (IITA-2006-1090-0603-0020)

† 비 회 원 : 한국과학기술원 전산학과
yjlee@camars.kaist.ac.kr
parksy@camars.kaist.ac.kr
pakej@camars.kaist.ac.kr
dwlee@camars.kaist.ac.kr
wjung@camars.kaist.ac.kr
†† 종신회원 : 한국과학기술원 전산학과 교수
jinsoo@cs.kaist.ac.kr

논문접수 : 2006년 5월 4일
심사완료 : 2007년 4월 11일

1. 서 론

PDA나 휴대폰과 같은 모바일 임베디드 시스템이 널리 사용되면서 이러한 기기에서 PC와 같은 환경의 소프트웨어를 사용하고자하는 사용자들의 수요가 늘어나고 있다. 그러나 모바일 임베디드 시스템은 특성상 장착할 수 있는 저장장치의 용량에 한계가 있어 사용자가 원하는 소프트웨어들을 모두 저장하고 유지관리하는데에 어려움이 많다. 이에 따라 비디오, 오디오 파일 스트리밍 시스템과 유사한 소프트웨어 스트리밍 시스템이 제안되었다.

소프트웨어 스트리밍 시스템이란 소프트웨어를 수행할 때 필요한 실행 코드를 네트워크로 전송받아 실행하는 것을 말한다. 이러한 소프트웨어 스트리밍 시스템을 사용하면 모바일 임베디드 시스템의 제한된 저장공간을 이용하여 다양한 소프트웨어를 수행할 수 있고 사용자의 필요에 따라서 소프트웨어를 전송받아 사용할 수 있으므로 유지관리가 용이하다. 그러나 모바일 임베디드 시스템에서는 소프트웨어의 실행코드를 전송 속도가 느린 무선 네트워크를 통해 블록단위로 전송받아 실행하게 되어 소프트웨어의 수행 시간이 오래 걸리는 단점이 있다. 이러한 단점은 실행코드를 미리 전송받는 선인출 기술을 사용하여 보완할 수 있다. 시간적으로 연속되어 사용될 확률이 높은 실행 코드 블록들을 연관짓고 특정 데이타 구조로 모델링하여 앞으로 사용될 코드 블록들을 예측하고 미리 전송함으로써 소프트웨어의 수행 시간과 사용자 응답 시간을 줄일 수 있다.

본 논문에서 구현되어 사용된 소프트웨어 스트리밍 서비스의 구조와 동작방식은 다음과 같다. 소프트웨어 스트리밍 서비스는 여러 소프트웨어를 저장하고 있는 서버와 이 서버에 접속하여 소프트웨어를 수행하는 모바일 임베디드 시스템과 같은 클라이언트로 구성된다. 클라이언트는 소프트웨어를 수행하기위해 서버에 접속하고 자체 개발한 파일시스템으로 마운트된 디렉토리에서 소프트웨어를 실행한다. 소프트웨어가 실행되면 파일 시스템은 해당 소프트웨어가 클라이언트의 로컬 저장 장치에 이미 저장되어 있다는 가정하에 4Kbyte 단위로 이루어진 코드 블록을 로컬 저장 장치에 요청한다. 요청 블록이 로컬 저장 장치에 없다면 파일 시스템은 서버에 해당 코드 블록을 요청하고 네트워크로 전송받아 수행한다. 이와 동시에 서버는 클라이언트가 요청한 코드 블록 정보를 이용해 이후에 사용될 블록을 예측하여 클라이언트에 미리 전송하여 준다.

본 논문에서는 위와 같은 소프트웨어 스트리밍 서비스에 기존에 연구된 과거 사용 기반 알고리즘을 적용하여 그것들의 장단점을 분석하고 새로운 알고리즘을 제안하는 것을 목적으로 하여 다음과 같은 두가지의 기존 선인출 알고리즘을 이용했다. 첫번째로 최근 접근 블록(LS; Last Successor) 알고리즘이다. 과거 사용 기반 모델 방법 중 가장 간단한 것으로 각 블록에 대해 바로 이전에 접근된 블록의 정보를 저장하여 선인출 블록 예측에 사용하는 방법이다. 이 알고리즘은 간단하고 메모리 사용량이 적은 장점이 있지만 상대적으로 선인출된 블록의 적중률이 낮다는 단점이 있다. 본 논문의 소프트웨어 스트리밍 서비스에 LS를 적용시켰을 때 소프트웨어 종류별로 평균 60%정도의 적중률을 나타냈으며 이는 아래에 기술할 다른 알고리즘 보다 낮은 적중률이다. 이는 LS의 특성이 기인한 것으로 선인출 해주는 블록의 개수가 적기 때문에 낮은 적중률을 나타낸다. 두번째 알고리즘으로 텍스트 압축 알고리즘인 PPM (Prediction by Partial Matching)에 기반한 알고리즘 (PPM 기반 알고리즘)이 있다[1-3]. 이 방법은 과거에 사용된 코드 블록들의 기록을 트라이 자료 구조 형태로 모델링하여 선인출 블록 예측에 사용한다. 이것은 최고 96%이상의 적중률을 나타내지만 사용되는 자료구조가 일반적으로 수십 메가 바이트 이상의 과도한 양의 메모리를 차지한다는 단점이 있다. 본 논문의 소프트웨어 스트리밍 서비스에 적용시켰을 경우 선인출된 블록의 적중률은 높았지만 소프트웨어에 따라서 수백메가에 이르는 메모리를 사용하는 경우도 있었다. 이러한 PPM 기반 알고리즘은 적중률은 높지만 코드 블록 단위로 선인출 해주는 소프트웨어 스트리밍 서비스에는 적합하지 않다.

본 논문에서는 위와 같은 기존 연구의 장단점을 바탕으로 확률 기반 다중 접근 블록(PMS; Probability-based Multi Successor) 알고리즘을 제안한다. PMS는 LS를 사용했을 때 선인출 블록의 개수가 적어 적중률이 낮고 PPM 기반 알고리즘은 적중률은 높지만 트라이(trie) 자료 구조를 이용하여 메모리 사용량이 너무 높다는 점에 착안하여 고안되었다. PMS는 LS의 자료구조를 기반으로하여 선인출될 수 있는 블록의 개수를 늘리고 PPM 기반 알고리즘에서 사용한 확률 계산 방법을 이용하여 자료구조를 구성한다. 이에 따라 PMS는 PPM이 기하급수적 공간 오버헤드를 갖는 반면 LS와 같은 선형적 공간 오버헤드를 갖으며 높은 적중률을 나타낸다. 본 논문 에서는 실험 통해 PMS는 PPM 기반 알고리즘의 메모리 사용량의 평균 3~4%정도의 메모리를 사용하지만 PPM 기반 알고리즘의 적중률에 근접한 적중률을 나타내어 메모리 사용량에 비해 좋은 성능을 보임을 나타내었다.

본 논문의 다음과 같이 구성되어 있다. 2장에서는 기존의 선인출과 관련된 연구들을 살펴보고, 3장에서는 다중 접근 블록 알고리즘의 착안점과 구조를 설명한다. 4장에서는 본 연구에서 사용된 기존의 선인출 알고리즘

과 다중 접근 블록 알고리즘의 실험결과를 비교, 분석하고 마지막으로 5장에서는 본 논문의 결론에 대해서 기술한다.

2. 관련 연구

기존의 선인출 알고리즘 관련 연구는 크게 세가지가 있다. 첫째, 순차적인 블록들을 한번에 여러 개씩 가져오는 알고리즘이다[8]. 기존의 리눅스와 같은 운영체제들은 디스크 I/O 시간을 줄이기 위하여 디스크의 내용을 메인 메모리로 읽어들일 때 미리 읽기를 수행한다. 미리 읽기는 요청된 블록과 근접한 연속 블록들을 한번에 읽어오는 방식으로 이러한 연속 블록 미리 읽기는 과거의 블록 접근 정보나 소프트웨어의 특성을 이용하지 않고 공간 지역성(spatial locality)만을 이용해 블록을 선인출하는 방법이다. 이 방법은 주로 순차적인 접근이 이루어지는 멀티미디어 스트리밍 서비스에 이용되는 방법이다. 그러나 소프트웨어 스트리밍 서비스에서는 멀티디미어 스트리밍 서비스와 달리 순차적 접근이 빈번히 일어나지 않으므로 적중률이 높지 않다. 본 연구의 소프트웨어 스트리밍 서비스에서는 네트워크를 이용하기 때문에 적중률이 높지 않은 블록들을 미리 읽어오는 것은 네트워크 대역폭을 낭비하여 실제로 사용되는 블록의 전송지연을 일으킬 수 있다. 또한 사용되지 않는 블록들을 미리 읽어와서 모바일 기기에 저장하게 되면 제한된 저장 공간을 낭비하게 되어 적중률이 떨어지게 된다. 이러한 이유로 인해 연속 블록 미리 읽기는 네트워크를 통한 소프트웨어 스트리밍 서비스에는 적합하지 않다.

두번째 방법으로 프로그래머가 앞으로 사용될 실행 코드를 예측할 수 있도록 정보를 주는 방법이 있다. 이러한 사전 기록 정보 기반 알고리즘은 프로그래머가 소프트웨어 작성 시, 소스 코드 내에 미리 정의된 함수를 이용하여 제공한 정보를 통해 선인출하는 것이다[5,6]. 소프트웨어 작성자가 기록한 정보이므로 적중률이나 정확도 면에서 일반적으로 다른 선인출 알고리즘보다 높은 성능을 나타낸다. 그러나 이 방법을 이용하기 위해서는 운영체제에서 선인출 정보 제공 관련 함수를 지원해야 하고 프로그래머가 소프트웨어의 소스코드에 선인출 정보 제공 관련 코드를 일일히 작성해야 된다는 단점이 있다. 그리고 기존의 소프트웨어를 수행할 때 이 알고리즘을 적용하여 성능을 높이려면 모든 소스 코드를 수정해야한다. 이와 같은 문제점으로 이 알고리즘은 본 논문에서는 고려하지 않는다.

세번째 방법은 과거에 소프트웨어를 수행할 때 사용된 코드 블록 사용 정보를 기록하고 이를 기반으로 하여 앞으로 사용될 코드 블록들을 예측하는 것이다. 이러한 과거 기록 기반 알고리즘은 이전에 해당 소프트웨어를 수행할 때 사용된 블록들의 접근 패턴 기록을 모델링하여 자료구조를 만들고 선인출 블록 예측시 이용하는 방법이다. 과거의 정보를 이용하기 때문에 위에서 기술한 연속블록 미리 읽기보다 높은 적중률을 나타낸다. LS와 PPM 기반 알고리즘이 이에 해당한다[1-3]. LS는 과거 기록을 분석하여 각 블록마다 가장 최근의 접근 블록 하나를 저장하여 모델링한다. 접근 블록(Successor)은 어떤 블록이 사용된 이후에 사용된 블록이다. 예를 들어, 블록 A 다음에 블록 B가 사용되었다는 과거 기록이 있다면 B는 A의 접근 블록이 되고 블록 A 이후에 사용될 블록으로 블록 B를 예측한다. 이 방법은 과거 기록을 바탕으로 각 블록마다 최근 접근 블록 하나씩의 정보를 배열 형식으로 저장하는 방법으로 구현이 가능하다.

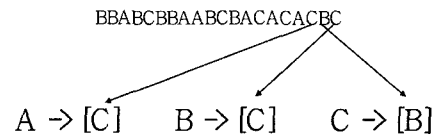


그림 1 최근 접근 블록 기반 알고리즘(LS)의 'BBABC BBAABCBACACACBC' 모델링

그림 1은 알파벳 A, B, C를 하나의 블록이라 생각하고 'BBABCBBAABCBACACACBC'의 과거 기록을 LS를 이용하여 모델링 한 것 이다. 위 패턴에서 A 다음에 마지막으로 나온 접근 블록이 C이고 B의 마지막 접근 블록은 C, C의 마지막 접근 블록은 B이므로 A -> [C], B -> [C], C -> [B]와 같이 모델링 된다. 그리고 각 블록마다 하나씩의 접근 블록을 저장하고 있으므로 A와 B의 접근 블록으로 C를, C의 접근 블록으로는 B를 1의 확률로 예측한다. 이전의 연구[2]에서 이루어진 파일 단위의 선인출 성능 측정에서 PPM 기반 알고리즘에 비해 낮은 70%의 적중률을 나타냈지만 알고리즘이 간단하며 메모리 사용량이 적다는 장점이 있다.

또 다른 과거 사용 기록을 분석하여 모델링하는 방법으로 텍스트 압축 알고리즘인 PPM(Prediction by Partial Matching)을 응용한 연구들이 있다[4,7]. PPM 기반 알고리즘의 동작 방법은 다음과 같다. 단어 내의 각 문자는 다음에 이어지는 문자의 컨텍스트가 되어 특정 문자 패턴이 나왔을 때 다음에 이어지는 문자를 예측할 수 있다. 예를 들어, "prefetch"라는 단어가 있을 때, 문자 h는 "prefetc" 다음에 이어지는 문자가 된다. 이를 소프트웨어 실행 코드 블록의 선인출 방법에 사용될 때 각 문자는 파일 이름과 블록 번호가 되고 문자열은 파일과 블록 번호의 스트림이 되어 소프트웨어 실행 코드

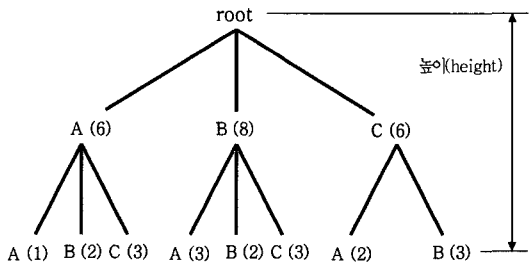
블록의 과거 사용기록이 된다[1-3]. 이러한 과거 사용기록을 트라이 형태로 모델링하고 다음에 사용될 파일 혹은 블록을 예측하는데 사용한다.

그림 2는 알파벳 A, B, C를 하나의 블록이라 생각하고 'BBABCBBAABCBACACACBC'의 과거 기록을 트라이(trie)를 이용하여 모델링한 것이다. 각 알파벳에 해당하는 각 노드는 블록을 나타내며 괄호 안의 숫자는 각 블록의 참조 회수를 나타낸다. 그림 2(a)의 트라이는 높이가 2인 트라이로 과거 기록에서 길이가 2인 서브 과거 기록의 정보를 모두 담고 있다. 즉, BA는 위 과거 기록의 서브 과거 기록이 되는데 그림 2(a)의 트라이는 BA에서 알 수 있는 정보인 B와 A의 사용, 그리고 B이후 A의 사용이 한번 있었다는 기록을 모두 담고 있다. 그림 2(b)의 트라이는 높이가 3인 트라이로 위와 같은 논리로 길이가 3인 서브 과거 기록의 정보를 모두 담고 있다.

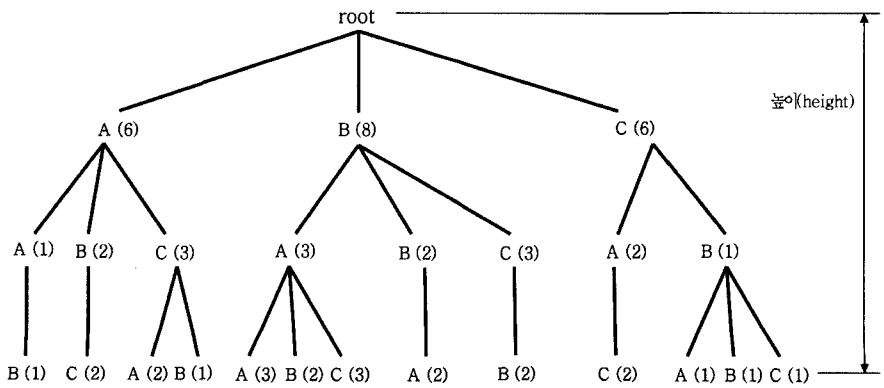
일반적으로 높이가 h인 트라이는 길이가 h인 서브 과거 기록 정보를 모두 담고 있으며 이러한 높이 값에 따라 다양한 순서(order), 깊이(depth), 확률(probability)의 변수에 의해서 예측 방식을 달리 할 수 있다. 순서는 몇 개의 블록을 가지고 다음 블록을 예측할 것인지를 나타내고 깊이는 몇 개의 블록을 예측할 것인지를 나타내며 확률(probability)은 특정 블록이 미래에 나타날 확률을 나타낸다. 만약 그림 2(b)의 트라이를 이용하여 트라이에서 순서 값이 2이고 깊이가 1인 예측을 한다면 AC이후의 접근 블록으로 A, C를 예측할 수 있고 A는 66%의 확률, C는 33%의 확률로 접근될 가능성이 있다는 것을 알 수 있다.

PPM 기반 알고리즘을 응용한 다양한 예측 방법에 관한 연구들이 기존에 이루어졌다. FMOC(Finite Multi-Order Context)[3]는 순서 값에 여러 단계를 두어 사용하는 방법으로 모델링된 트라이의 높이(height) 값에 따라 순서와 깊이값을 다양하게 하여 예측할 수 있다는 장점이 있는 반면, 높이 값이 커질수록 트라이의 크기가 기하 급수적으로 커진다는 단점이 있다. N을 코드 블록의 총 개수, m을 트라이의 높이값 이라고 했을 때, 트라이를 저장하기 위한 공간 오버헤드는 $O(N^m)$가 되고 N과 m이 증가함에 따라 트라이를 저장하기 위한 오버헤드는 급속도로 증가한다.

PCM(Partitioned Context Model)[2]은 FMOC의 공간 오버헤드를 줄이기 위한 목적으로 고안 되었다. 트라이의 크기를 줄이기 위해서 루트의 첫 번째 자식 노드를 중심으로 트라이를 파티션하고 각 파티션의 크기를 일정하게 제한한다. 그 파티션 내의 노드를 삭제하



(a) 높이값이 2인 트라이



(b) 높이값이 3인 트라이

그림 2 PPM 기반 알고리즘의 'BBABCBBAABCBACACACBC' 모델링

기 위해서 다음과 같은 방법이 사용된다. 각 노드는 과거의 접근 횟수에 대한 카운트를 가지고 있는데, 파티션이 가득 찬 경우에 이 카운트를 절반으로 나누고 그 값이 0이 되는 노드는 파티션에서 삭제한다. 이때, 파티션 내의 모든 노드 카운트가 0가 되지 않으면 노드는 추가되지 않는다. 이러한 노드 삭제 방법은 트라이가 최근의 사용 패턴에 따라 빠르게 변하도록 하기 위한 것이다. PCM 방법의 공간 오버헤드는 파티션의 크기에 따라 다르며 $O(N^m)$보다 작고 $O(n)$보다 크다. 실험을 통해 분석한 결과, PCM 방법은 FMOC에 비해 성능이 크게 떨어지지 않지만 공간 오버헤드는 크게 줄일 수 있다[2].

EPCM(Extended PCM)[1]은 PCM의 확장 알고리즘으로 한번에 여러 파일 혹은 블록을 예측하기 위한 방법이다. EPCM 트라이의 각 노드는 접근 횟수를 근거로 다음에 접근될 확률을 구할 수 있는데, 확률 값이 일정 값 이상인 자식 노드들을 한번에 선인출한다.

3. 확률 기반 다중 접근 블록(Probability-based Multi Successor)

본 절에서는 기존의 선인출 알고리즘의 비교 분석 결과를 바탕으로 소프트웨어 스트리밍 서비스에 적합한 효율적인 선인출 알고리즘인 확률 기반 다중 접근 블록 알고리즘 제안한다.

3.1 기존 선인출 알고리즘 성능 측정 결과 분석

앞서 기술한 LS와 PPM 기반 알고리즘을 소프트웨어 스트리밍 시스템의 선인출 알고리즘에 적용시켜 다음과 같은 분석 결과를 얻었다. LS를 사용한 경우에는 알고리즘이 간단하고 메모리 사용량이 적지만 상대적으로 낮은 적중률을 나타냈다. PPM 기반 알고리즘을 사용한 경우에는 알고리즘이 복잡하고 메모리 사용량이 많지만 높은 적중률을 나타냈다. PPM 기반 알고리즘의 순서값 증가에 따른 선인출 적중률의 변화를 측정한 결과, 순서값이 증가할수록 적중률이 떨어지거나 변화하지 않아 순서값이 1일 때 최고 적중률을 얻을 수 있다. 즉, 소프트웨어 실행 코드의 선인출 예측 시에는 순서값을 증가시킬수록 필요한 트라이의 높이 값이 증가하여 메모리 사용량이 늘어나므로 순서값이 1일 때 가장 효율적인 적중률을 얻을 수 있다.

LS와 PPM 기반 알고리즘을 이용한 소프트웨어 실행 코드 블록 선인출 적중률 결과를 바탕으로 다음과 같이 정리할 수 있다.

- 실행 코드 블록의 선인출시에는 순서값은 1이면 충분하므로 과거기록을 메모리 사용량이 많은 트라이 구조로 모델링할 필요가 없다.
- LS는 예측 블록으로 선택 가능한 코드 블록의 개수가 적어 적중률이 낮으므로 각 블록마다 예측 블록으로 선택 가능한 코드 블록의 개수를 늘려서 적중률을 높일 수 있다.

위 2가지 결론을 바탕으로 다음과 같은 알고리즘을 고안하였다.

3.2 다중 접근 블록(MS: Multi Successor)

MS는 예측 블록으로 선택 가능한 코드 블록의 개수가 적다는 LS의 단점을 보완하기 위해 각 블록마다 버켓 수(N)만큼의 최초 접근 블록을 저장하는 방법으로 과거 기록을 모델링한다. 각각의 소프트웨어나 과거 기록의 특성에 따라 적절한 N을 정해주고 소프트웨어 스트리밍 서비스의 선인출 알고리즘에 적용하면 LS보다 높고 PPM 기반 알고리즘 기반 방법에 근접한 적중률을 얻을 수 있다.

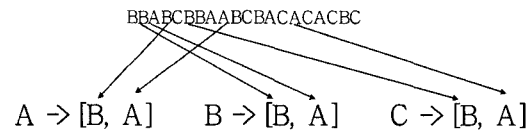


그림 3 MS의 'BBABCBBAABCBACACACBC' 모델링

그림 3은 알파벳 A, B, C를 하나의 블록이라 생각하고 'BBABCBBAABCBACACACBC'의 과거 기록을 N이 2인 누적 다중 접근 블록 알고리즘으로 모델링 한것이다. A의 최초 접근 블록 2개는 B와 A이므로 A에 대해서는 B와 A 정보를 가지고 있고 이와 비슷하게 B는 B, A, C도 B와 A 정보를 가지고 있다. A 이후에 나타날 접근 블록으로는 B와 A 모두 같은 확률로 예측한다.

LS의 적중률 결과에서 알 수 있듯이 N값이 너무 적으면 예측 블록으로 선택할 수 있는 코드 블록의 개수가 적기 때문에 높은 적중률을 얻을 수 없고 N값을 너무 크게 하면 높은 적중률을 얻을 수는 있겠지만 그에 반해 데이타 구조를 저장하는 메모리를 낭비하게 되어 적절한 N의 값을 알아내야한다. 적절한 N은 어느 수준의 적중률을 얻기 위해 저장해야할 각 코드 블록의 접근 블록 개수이며 이는 PPM 기반 알고리즘으로 과거기록을 모델링한 트라이를 분석함으로서 알 수 있다. 트라이에서 레벨 3에 위치한 노드가 나타내는 블록이 그 부모 노드 블록의 접근 블록이 된다. 그래서 레벨 2의 노드들의 평균적인 자식 노드의 수를 파악하면 적절한 N을 구할 수 있다. 즉, 그림 2(b)에서 C의 접근 블록은 A, B가 되므로 N이 2인 다중 접근 블록 알고리즘을 사용한다면 C의 모든 접근 블록을 모델링 할 수 있다. 이 방법은 N*블록개수 만큼의 블록 정보를 가지고 있으면 되므로 O(n) (n: 소프트웨어의 실행 코드 블록 수)의 공간 오버헤드를 갖는다.

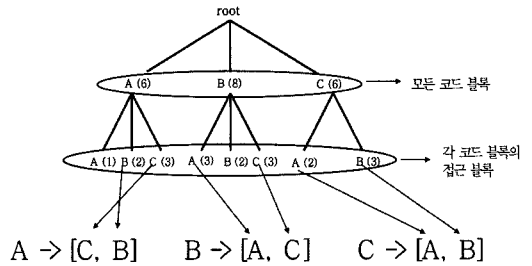


그림 4 PMS의 'BBABCBBAABCBACACACBC' 모델링

3.3 확률 기반 다중 접근 블록(PMS: Probability-base Multi Successor)

위에서 언급한 MS는 특정 블록의 여러 개의 접근 블록들 간의 확률을 고려하지 않은 방법이다. 확률을 고려하지 않았기 때문에 트라이 분석을 통해 찾은 적절한 N값을 이용해도 PPM 기반 알고리즘에 근접한 적중률을 얻을 수 없다. 이를 해결하기 위해 PMS는 확률이 높은 N개의 접근 블록을 이용하여 모델링한다. 확률이 높은 N개의 접근 블록을 구하기 위해서 PPM 기반 알고리즘으로 생성한 높이값이 2인 트라이를 이용한다. 그림 4는 'BBABCBBAABCBACACACBC'의 과거 기록을 그림 2(a)에서 언급한 높이가 2인 트라이를 이용하여 N이 2인 PMS로 모델링 한것이다. 그림 4의 트라이에서 볼수 있듯이 같이 과거 기록에 나타난 블록들은 모두 레벨이 2인 노드로 모델링 되고 레벨 3인 노드는 부모 노드가 나타내는 블록의 접근 블록을 모델링 한 것이 된다. 그러므로 각 블록들마다 확률이 높은 N의 접근 블록을 구하기 위해서는 레벨 2인 노드의 자식 노드 중에서 확률이 높은 N개의 노드를 이용하면 된다.

PMS의 자료구조를 생성하는 방법을 간단한 코드로 나타내면 그림 5와 같다. 사전에 소프트웨어 실행을 통해서 각 블록들의 참조 정보가 기록된 과거 기록을 얻고 이를 통해 PPM 기반 알고리즘을 사용하여 높이 값이 2인 트라이를 생성한다. 생성된 트라이 통해 그림 4와 같은 방법으로 모든 각 블록들의 확률이 높은 N개의 블록을 구하고 이 N개의 블록을 배열형태로 저장하여 자료 구조를 구성한다[1-3]. 이러한 자료구조는 각 블록마다 N개의 블록 정보를 가지고 있으므로 O(n) (n: 소프트웨어의 실행 코드 블록수)의 공간 오버헤드를 갖는다. 실제 소프트웨어 스트리밍 서비스에 PMS를 사용할 때는 그림 5의 자료구조 생성과정이 시간이 많이 걸리는 작업이므로 오프라인으로 따로 수행하여 자료구조를 만들어 실제 서비스에 사용해야한다.

자료구조를 이용하여 블록을 예측할 때는 MS와 같은 방법을 사용한다.

4. 실 험

4.1 실험환경

본 연구에서는 일반적인 서버/클라이언트 구조를 가진 자체 개발한 소프트웨어 스트리밍 시스템을 이용하여 LS, PPM 기반 알고리즘, MS, 본 논문에서 제안한 PMS를 구현하여 각각의 적중률을 측정했다. 서버는 자바로 구현되었으며 Pentium 4, 1GB SDRAM, 윈도우 XP환경에서 구동했다. 클라이언트는 하이버스- PXA270 임베디드 시스템(FWPXA270C5 (520MHz) CPU, 64 SDRAM)의 Montavista 커널 2.6 환경에서 로컬 저장장치와 네트워크 파일 전송을 관리하는 파일 시스템, 이 파일 시스템으로 마운트된 디렉토리에서 원하는 소프트웨어를 수행시켜주는 프로그램으로 이루어져 있다. 서버와 클라이언트는 100Mbps 이더넷 환경에서 통신한다.

서버/클라이언트의 동작 방식은 그림 6과 같다. 먼저 소프트웨어 수행이 시작되면 클라이언트의 선인출 파일 시스템에서 서버에 코드 블록을 요청한다. 그러면 서버에서는 클라이언트 별로 이미 전송된 코드 블록들을 기록한 비트맵과 현재 선인출되어 전송되고 있는 코드 블

```
// 사전에 소프트웨어 실행을 통해 과거 기록(trace.log)을 얻는다.

Trie trie = make_Trie("trace.log", height = 2);
// 사전에 얻은 과거 기록과 PPM 알고리즘을 이용하여 높이값이 2인 트라이를 생성한다.

Successors successors = new Successors[numberOFblocks][#_buckets];
// 각 블록별로 N개의 접근 블록을 저장.
// 각 블록의 접근 블록들은 successor변수와 블록의 인덱스로 접근한다.

for(each blocks){
        temp_successors = retrieve_N_blocks(trie);
        // 각 블록마다 확률이 높은 N개의 접근 블록을 구한다.
        successors[idx_block] = temp_successors;
}
```

그림 5 PMS 자료구조 생성 pseudocode

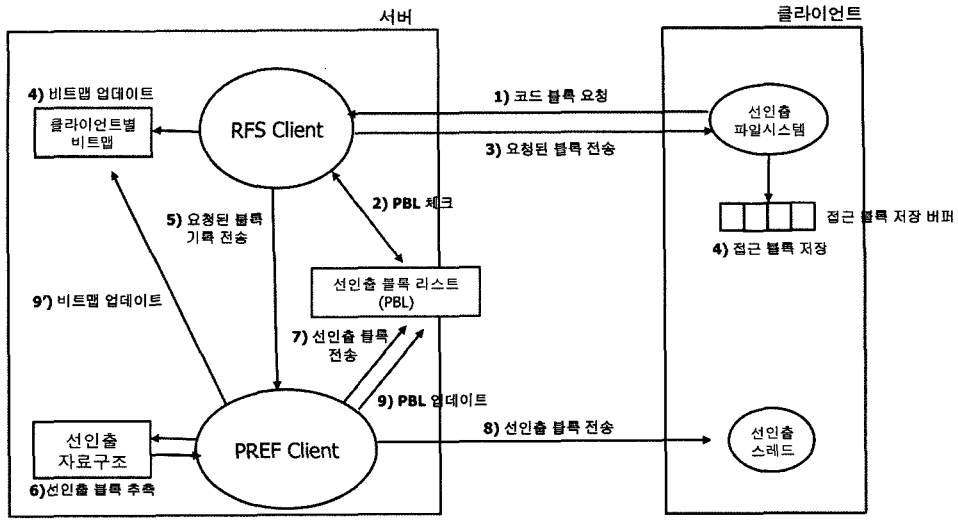


그림 6 본 논문에서 사용된 소프트웨어 스트리밍 서비스의 블록 다이어그램

록을 저장하고 있는 PBL을 참조하여 해당 블록을 전송해 준다. 그리고 서버의 다른 부분에서는 요청된 블록의 기록을 바탕으로 선인출 자료구조를 참조하여 앞으로 사용될 코드 블록을 예측하여 선인출 해준다. 해당 선인출 블록은 클라이언트의 선인출 스레드에 의해 수신된다.

선인출 알고리즘에 사용된 각 소프트웨어의 과거기록은 소프트웨어별로 정해진 시나리오의 수행에 의해 만들어졌다. 과거 기록은 텍스트 파일 형식이며 [파일 이름, 파일 블록 숫자] 로 구성되어 있다. 이들은 클리이언트에서 소프트웨어를 수행할 때 서버에 요청하는 코드 블록과 서버에서 선인출 하여 네트워크로 전송해준 코드 블록중 실제로 사용된 블록들의 정보로 구성되어있다. 네트워크의 상황에 따라 소프트웨어들을 같은 시나리오로 수행한다고 해도 서버로 요청되는 코드블록과 선인출된 코드 블록중 사용된 블록들이 달라지기 때문에 다양한 과거기록이 생성되며 과거 기록의 양이 많을수록 다양한 패턴의 정보가 저장되어 있다고 할 수 있다.

성능 측정에 사용된 기준은 적중률이다. 이는 소프트웨어를 수행하면서 서버로부터 네트워크로 전송되어 사용된 전체 블록 개수에 대한 선인출에 성공하여 클라이언트가 필요함에도 불구하고 직접 네트워크로 요청되지 않은 블록의 비율이다.

실험에는 표 1과 같은 4개의 소프트웨어가 사용되었다. Doom2는 게임 소프트웨어, Abiword 2.2는 문서작업 소프트웨어, PDFviewer는 pdf형식의 문서 뷰어 소프트웨어이고 Siag는 스프레드시트 소프트웨어이다. 각 소프트웨어는 하이버스- PXA270임베디드 시스템의 Montavista 커널에서 수행되도록 수정된 게임/문서 소프트웨어이며 특성은 표 1에 나타나 있다.

표 1 실험에 사용된 소프트웨어

이름	전체 블록 수(개)	시나리오에서 사용된 블록수(개)	과거 기록 파일 크기(Kbyte)
Doom2	4977	2734	1419
Abiword 2.2	4637	1240	129
PDFviewer	209	144	38
Siag	2259	312	95

LS, MS, PMS는 각 블록의 접근 블록을 기반으로 하여 다음 사용 블록을 예측하므로 순서값과 깊이값이 1인 예측만 가능하다. 그러나 PPM 기반 알고리즘과의 예측 성능 비교를 위해 재귀적으로 블록을 예측함으로서 특정 블록 다음에 사용될 임의의 개수만큼의 블록들을 예측할 수 있게 하였다. A -> [C], B -> [C], C -> [B]와 같이 모델링 되어있을 때 A 이후에 나타날 블록 2개를 예측할 때에는 A -> [C]에 의해 C, 그리고 A 이후에 C를 예측 했으므로 그 다음에 사용될 블로으로 C->[B]를 이용하여 B를 예측한다. 즉, A 이후에 나타날 2개의 블록으로 CB를 예측하는 것이다. 확률(probability)에 관련하여 각 블록마다 하나의 블록만 과거 기록으로서 가지고 있기 때문에 모든 경우에 확률은 1로 계산된다.

4.2 이전 연구 결과

4.2.1 최근 접근 블록(LS)

LS의 소프트웨어별 적중률 측정 결과는 표 2와 같다. Abiword 2.2를 제외한 다른 소프트웨어들의 적중률은 60% 내외의 값을 보이고 있다. Abiword 2.2의 적중률만 84.78%로 비교적 높은 값을 나타내고 있다. 표 2에서 보는 것과 같이 Abiword 2.2의 각 블록의 평균 접근 블록의 수가 1.3개이고 표준편차가 0.79로 대부분 블

표 2 소프트웨어별 평균 접근 블록 수

	Doom2	Abiword 2.2	PDFviewer	Siag
LS의 적중률(%)	62.98	84.78	58.70	60.52
평균 접근 블록 수(개)	6.41	1.3	4.85	2.14
접근 블록 수의 표준편차	3.21	0.79	6.13	1.56

록의 접근 블록 개수가 1~2개임을 알 수 있고 더 나아가서 2개인 블록보다 1개인 블록이 더 많음을 알 수 있다. 이러하기 때문에 하나의 접근 블록만 저장할 수 있는 LS로 다른 소프트웨어에 비해 충분한 수의 예측 가능한 블록이 확보 가능하여 비교적 높은 적중률을 나타낸다.

4.2.2 PPM 기반 알고리즘

PPM 기반 알고리즘의 순서값 증가에 따른 소프트웨어별 적중률 측정 결과는 그림 7과 같다.

Siag를 제외한 나머지 소프트웨어에 대해서 순서값이 1일 때 95%이상의 적중률을 나타냈다. Siag경우에는 85%정도의 적중률을 나타냈다. 모두 최근 접근 블록 방법보다 높은 적중률을 나타냈다. 이는 LS보다 선인출되는 블록의 개수가 더 많기 때문이다.

순서값이 증가하면 Doom2와 Abiword 2.2의 적중률은 하락했고 PDFviewer와 Siag의 적중률은 크게 변하지 않았다. 순서값이 증가할수록 적중률이 하락한 이유

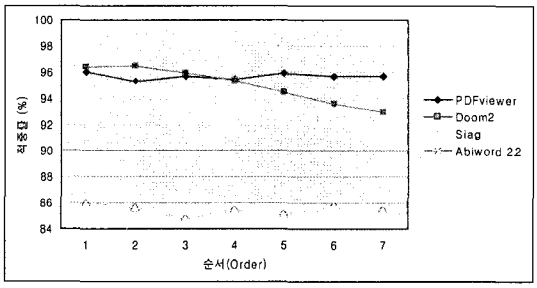


그림 7 PPM 기반 알고리즘의 순서값 변화에 따른 적중률 변화

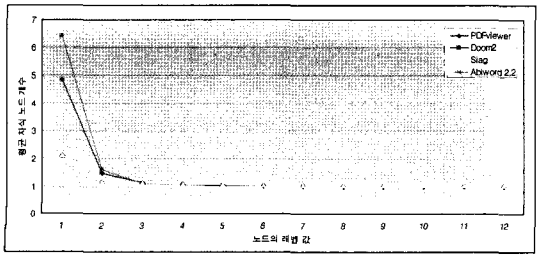


그림 8 모델링된 트라이의 레벨 증가에 따른 자식 노드 수의 변화

는 그림 8에서와 같이 트라이에서 노드의 레벨값이 증가할수록 평균 자식 노드수가 줄어들어 선인출 블록으로 선택 가능한 코드 블록의 개수가 줄어들고 이에 따라 선인출되는 코드 블록의 개수가 줄어들기 때문이다. 따라서 필요한 블록들을 네트워크로 직접 요청되는 블록의 개수가 증가하여 적중률이 떨어진다. PDFviewer와 Siag의 적중률이 하락하지 않고 크게 변화하지 않는 이유는 전체적으로 사용되는 블록의 개수가 적어 선인출되는 코드 블록 개수 감소의 영향이 다른 소프트웨어에 비해 상대적으로 적기 때문이다.

4.3 확률 기반 다중 접근 블록(Probability-based Multi Successor)

4.3.1 다중 접근 블록(Multi Successor)

MS의 버켓의 수 증가에 따른 소프트웨어별 적중률 측정 결과는 그림 9와 같다.

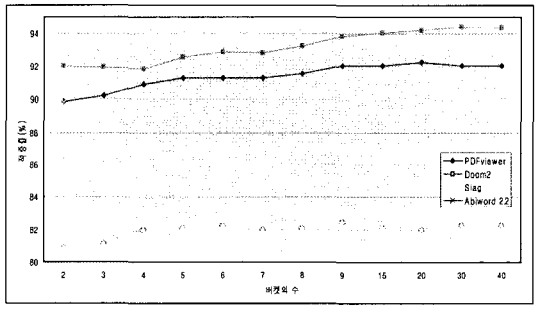


그림 9 버켓 크기 증가에 따른 MS의 적중률 변화

PDFviewer, Doom2, Abiword 2.2 같은 경우 90% 근접하거나 90% 이상의 적중률의 보여 LS보다 PDFviewer는 1.56배, Doom2는 1.5배, Abiword 2.2는 1.06배 높은 적중률을 나타냈다. Siag 경우는 80%이상의 적중률을 나타냈으며 LS 보다 1.36배 높은 적중률을 나타냈다.

Abiword 2.2를 제외하고 다른 소프트웨어의 경우에는 버켓의 수가 증가할수록 적중률의 값이 증가함을 알 수 있다. 이는 버켓의 수가 증가할수록 선인출 코드 블록으로 선택 가능한 코드 블록의 개수가 증가하여 전체적으로 선인출되는 코드 블록의 개수가 증가하기 때문이다. PDFviewer 경우에는 버켓의 크기가 9일 때 약 92%의 적중률, Doom2의 경우에는 버켓의 크기가 15개일 때 약 94%의 적중률, Siag 경우에는 버켓의 크기가 4개일 때 약 82%의 적중률로 수렴했다. 버켓의 수가 그림에서 알 수 있는 각 블록의 평균 접근 블록의 수에 가까울 때 적중률이 수렴할 것이라 예상했지만 결과는 그렇게 나오지 않았다. 그 이유는 버켓의 수만큼의 접근 블록을 저장할 때 각 접근 블록의 확률을 고려하지 않아 불필요한 블록 정보가 저장되었기 때문이다.

Abiword 2.2의 경우 버켓이 증가하여도 적중률이 증가하지 않는 이유는 앞서 설명했듯이 각 코드 블록의 접근 블록의 개수가 1~2개로 버켓의 수가 2개일 때에도 충분한 수 코드블록을 선인출 할 수 있기 때문이다.

4.3.2 확률 기반 다중 접근 블록(Probability-based Multi Successor)

PMS를 적용했을 때 사용된 블록의 개수가 적은 PDFviewer, Siag경우에는 확률 적용에 따른 적중률의 의미있는 변화가 없었다. Doom2의 경우에는 MS를 사용했을때보다 적은 버켓의 수에서 최고 적중률이 수렴했고 Abiword 2.2경우에는 MS를 사용했을때보다 약간 높은 적중률을 나타냈다.

버켓의 크기에 따른 Doom2와 Abiword 2.2의 적중률 측정 결과는 그림 10, 그림 11과 같다.

Doom2의 경우에는 버켓의 크기가 모든 블록의 모든 접근 블록을 저장 할 수 있는 크기인 20 이상에서는 MS와 PMS 두 알고리즘 모두 비슷한 최고 적중률에 수렴했다. MS를 사용했을때는 버켓의 크기가 15일 때 최고 적중률로 수렴하기 시작했지만 PMS에서는 버켓의 크기가 Doom2 코드 블록의 평균 접근 블록 수인 6.41에 근접한 6일 때 최고 적중률로 수렴하기 시작했다. 이는 버켓의 크기만큼의 접근 블록을 저장할 때 높은 확률 순으로 저장하여 불필요한 블록을 선인출 해주는 경우가 줄어들기 때문이다.

Abiword 2.2 경우에는 버켓의 크기가 2일 때 코드 블록의 대부분 접근 블록을 저장할 수 있으므로 MS와 PMS 두 알고리즘에서 최고 적중률에 수렴했다. PMS의 적중률이 전체적으로 약 0.2% 높은 적중률을 나타냈

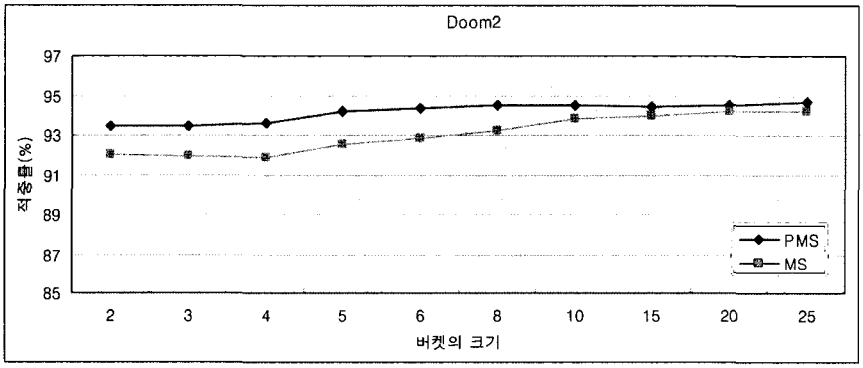


그림 10 PMS의 버켓 크기 증가에 따른 Doom2의 적중률 변화

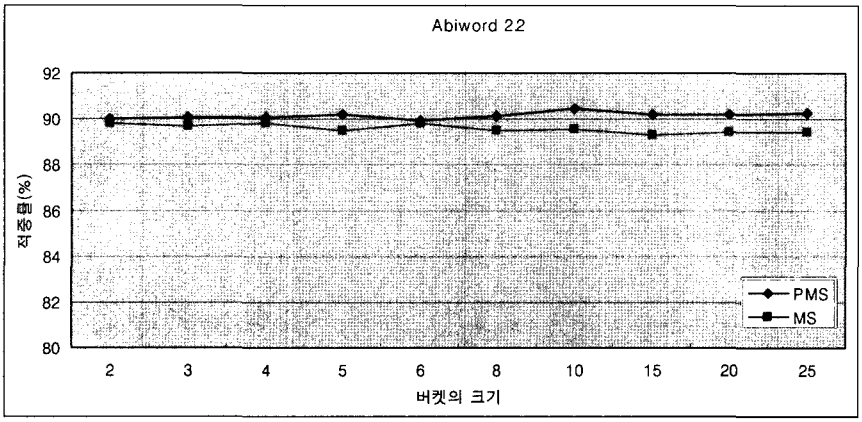


그림 11 PMS의 버켓 크기 증가에 따른 Abiword 2.2의 적중률 변화

표 3 소프트웨어별 선인출 방법에 따른 최고 적중률을 나타낼 때의 메모리 사용량(bytes)

	PPM 기반 알고리즘	LS	MS	PMS
PDFviewer	31725744	11248	85472	79128
Doom2	567030816	258592	2436880	1379512
Siag	29157744	601256	4222144	3937462
Abiword 2.2	30960640	3429616	19443088	18774608

다. 이는 네트워크 상황에 따른 오차 범위내의 값으로 Abiword 2.2 과거기록의 특성상 PMS와 MS간의 차이가 존재하지 않으므로 큰 의미는 없다.

4.4 적중률 비교와 메모리 사용량

소프트웨어별 각 방법으로 얻을 수 있는 최고 적중률을 나타내면 그림 12와 같다.

모든 소프트웨어에 대해서 PPM 기반 알고리즘을 사용했을 때 가장 높은 적중률을 나타냈다. 그 다음으로 PMS, MS순으로 높은 적중률을 나타냈다. LS는 가장 낮은 적중률을 나타냈다.

그림 12의 각 소프트웨어 별로 방법별 최고 적중률을 나타낼 때 사용되는 메모리 양은 표 3과 같다.

각 소프트웨어 별로 4가지 방법으로 실험할 때 순서값은 1, 깊이값은 25를 사용한 결과이다. 따라서, PPM 기반 알고리즘에 사용된 트라이의 높이값은 26이며 표 3에 나타난 메모리 사용량은 높이값이 26인 트라이를 구성하기 위해 사용된 메모리 양이다. PDFviewer 경우 MS의 버켓의 크기는 9이며 Doom2는 15, Siag는 4, Abiword 2.2는 2이다. PMS를 사용할 때 Doom2의 경우에는 버켓의 크기가 6이고 Abiword 2.2경우에는 2이다.

PPM 기반 알고리즘은 수십 메가에서 수백메가에 이르는 메모리를 사용하고 LS 비롯한 나머지 MS, PMS는 Abiword 2.2를 제외하고 수백킬로 바이트 수준의 메모리를 사용한다. MS와 PMS는 PPM 기반 알고리즘보다 5% 내외의 낮은 적중률은 보이지만 수백 분의 1의 메모리만을 사용하여 높은 메모리 사용 효율을 나타냈다.

5. 결 론

본 논문에서는 모바일 임베디드 시스템을 위한 소프트웨어 스트리밍 서비스에서 사용될 수 있는 선인출 알고리즘 방법을 제안한다. 기존에 연구된 방법들의 특성을 이용해서 제안된 PMS를 이용하여 적은 양의 메모리를 사용하여 이전 방법과 비슷한 수준의 적중률을 얻어 메모리 사용 효율을 높였다.

PPM 기반 알고리즘은 과거기록 파일을 이용하여 일정 범위 내의 코드 블록 사용 패턴을 모두 트라이로 모델링하여 다른 방법에 비해 96%에 가까운 높은 적중률을 나타냈지만 수십 메가 바이트에서 수백 메가 바이트에 이르는 과도한 양의 메모리를 사용하여 실제 소프트웨어 스트리밍 서비스에서 사용할 수 없는 문제점이 있다. LS는 PPM 기반 알고리즘에 비해 수백분의 1에 달하는 매우 적은 메모리만을 사용하지만 평균적으로 60% 내외의 낮은 적중률을 나타내는 단점이 있다.

다양한 변수를 이용하여 이루어진 위 두 방법의 실험결과를 토대로 소프트웨어의 스트리밍 서비스에서 과거기록을 모델링하여 선인출 블록을 예측할 때 순서값은 1이면 충분하고 LS는 선인출 가능한 블록의 개수가 적어 적중률이 낮음을 알 수 있다. 이를 바탕으로 LS와 달리 각 코드 블록마다 N개의 접근 블록을 저장하여 예측에 사용하는 MS, N개의 접근 블록을 저장할 때 확률이 높은 순으로 저장하여 예측에 사용하는 PMS를 제안한다. PMS를 사용할 때 각 소프트웨어의 과기기록 분석을 통해 각 코드 블록의 접근 블록 개수를 구하여 이를 버켓의 크기로 사용하면 PPM 기반 알고리즘에 근접하는 적중률을 얻어 메모리 사용 효율을 높였다.

이와 같이 메모리 사용량이 적고 적중률이 높은 PMS가 적용된 소프트웨어 스트리밍 서비스가 이루어 진다면 다른 선인출 알고리즘을 사용했을 때보다 적중률 대비 사용되는 메모리양을 줄일 수 있고 소프트웨어의 수행 시간이 줄어들어 모바일 임베디드 환경에서 보다 나은 소프트웨어 수행 환경이 제공된다.

앞으로는 본 논문의 실험환경과는 다른 무선 네트워크를 사용하는 모바일 임베디드 시스템을 이용한 연구를 통해 적중률 및 응답속도를 파악하여 실제 상용서비스의 적용 가능성을 알아보아야한다. 또한, 선인출 알고리즘을 사용한 소프트웨어 스트리밍 서비스의 소프트웨어의 구조를 소프트웨어 스트리밍 서비스에 맞게 최적

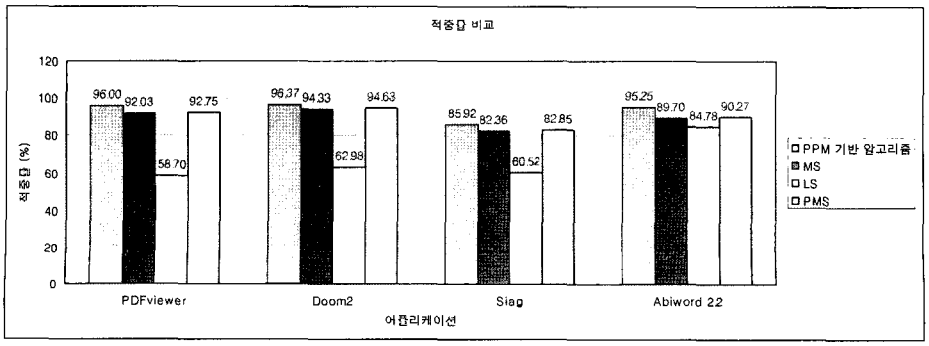


그림 12 소프트웨어별 선인출 알고리즘에 따른 최고 적중률

화하기 위한 연구가 진행되어야한다. 본 논문에 사용된 소프트웨어들은 대부분 스트리밍 서비스를 고려하지 않고 만들어진 것들이다. 그래서 소프트웨어가 처음 실행할 때 해당 소프트웨어의 대부분이 필요한 경우에는 시작이 완료될 때까지 모두 전송 받아야하므로 시간이 매우 오래 걸리고 스트리밍의 특성을 살리지 못하는 문제점이 있다. 소프트웨어가 처음 시작할 때 필요한 부분을 가능한한 적게 만든다면 단시간 내에 소프트웨어의 시작을 완료하고 사용자에게 입력을 받을 준비가 되었음을 알릴 수 있고 사용자의 입력이 이루어지는 동안 나머지 부분을 전송받을 수 있어 좀 더 좋은 사용자 반응 시간을 가질 수 있고 스트리밍의 특성을 최대한으로 활용할 수 있을 것이다.

참 고 문 헌

[1] Thomas M. Kroeger, Darrell D. E. Long, "Design and Implementation of a Predictive File Prefetching Algorithm," In Proc. of the 2001 USENIX Annual Technical Conf., pp. 105-118, 2001.

[2] Thomas M. Kroeger and Darrell D. E. Long, "The Case for Efficient File Access Pattern Modeling," in Proceedings of the SeventhWorkshop on Hot Topics in Operating Systems (HotOS-VII), IEEE, 1999.

[3] T. M. Kroeger and D. D. E. Long, "Predicting filesystem actions from prior events," in Proceedings of the USENIX 1996 Annual Technical Conference, pp. 319 - 328, USENIX, 1996.

[4] Jasmine Y.Q. Wang, Joon Suan Ong, Yvonne Coady, Michael J. Feeley, "Using Idle Workstations to Implement Predictive Prefetching," hpdc, p. 87, Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC-9 '00), 2000.

[5] Dorota M. Huizinga, Saurabh Desai, "Implementation of Informed Prefetching and Caching in Linux," The International Conference on Information Technology: Coding and Computing (ITCC'00), p. 443, 2000.

[6] Patterson, H. R., Gibson, G. A., Ginting, E., Stodolsky, D., Zelenka, J., "Informed Prefetching and Caching," Proceedings of the 15th ACM Symp. on Operating System Principles, pp. 79-95, 1995.

[7] K.M.Curewitz, P.Krishnan, and J.S.Vitter. "Practical Prefetching via Data Compression," Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD '93), 257-266, 1993.

[8] R.J.Feiertag and E.I.Organisk, "The Multics Input/Output System," In Proc. Third Symp. on Operating System Principle, pp. 35-41, 1971.

이 영 재

2005년 한국과학기술원 전자전산학과 전산학전공 학사. 2005년~현재 한국과학기술원 전자전산학과 전산학전공 석박사통합과정 재학. 관심분야는 운영체제, 분산 시스템, 스토리지 시스템

박 선 영

1999년 충남대학교 컴퓨터공학과 학사 2001년 한국과학기술원 전자전산학과 전산학전공 석사. 2003년~현재 한국과학기술원 전자전산학과 전산학전공 박사과정 2001년~2002년 한국전자통신연구원 컴퓨터시스템연구부 연구원. 관심분야는 운영체제, 클러스터 컴퓨팅, P2P시스템

박 은 지

2002년 한국과학기술원 전자전산학과 전산학전공 학사. 2004년 한국과학기술원 전자전산학과 전산학전공 석사. 2004년~현재 한국과학기술원 전자전산학과 전산학전공 박사과정. 관심분야는 운영체제, 분산 시스템, 스토리지 시스템, 병렬 처리

이 대 우

2002년 한국과학기술원 전자전산학과 전기및전자공학전공 학사. 2003년~현재 한국과학기술원 전자전산학과 전산학전공 석박사통합과정 재학. 관심분야는 운영체제, 분산시스템, Virtual Machine

정 욱

2004년 한국과학기술원 전자전산학과 전산학전공 학사. 2004년~현재 한국과학기술원 전자전산학과 전산학전공 석박사통합과정 재학. 관심분야는 운영체제, 스토리지 시스템, 분산 시스템

김 진 수

1991년 서울대학교 컴퓨터공학과 학사 1993년 서울대학교 컴퓨터공학과 석사 1999년 서울대학교 컴퓨터공학과 박사 1998년~1999년 IBM T. J. Watson Research Center 방문연구원. 1999년~2002년 한국전자통신연구원 선임연구원 2002년~현재 한국과학기술원 전자전산학과 부교수. 관심분야는 운영체제, 내장형 시스템, 스토리지 시스템, 컴퓨터 시스템