OODBMS 성능향상을 위한 객체 선인출 전략
(An Adaptable Object Prefetch Policy for Enhancing OODBMS Performance)

안 정 호 † 김 형 주 ††
(Jung-Ho Ahn)(Hyoung-Joo Kim)

요 약 객체지향 데이타베이스에서 객체 접근의 성능은 효율적인 객체 선인출을 통해 이루어질 수 있다. 본 연구에서는 고급의 객체 시맨틱을 사용하지 않고 페이지나 세그먼트를 단위로 선택적인 객체 선인출을 수행하는 동적 SEOF(Selective Eager Object Fetch) 방법을 고안하였다. 본 알고리즘은 객체 인출의 상관 관계와 빈도수를 모두 고려하였으며, 다른 기존의 객체 선인출 방법들과는 달리 시스템의 부하에 따라 선인출의 정도를 동적으로 조정함으로써 클라이언트의 메모리나 스왑 자원을 효율적으로 이용하여 시스템의 성능을 향상시킨다. 또한 제안된 방법은 객체 버퍼의 사용을 제한하여 자원의 고갈을 막을 수 있으며, 클러스터링의 정도나 데이타베이스의 크기에 대해 효과적으로 대응한다. 본 논문에서는 다양한 다중 클라이언트 환경에서의 시뮬레이션을 통해 제안된 알고리즘의 성능 평가를 실시하였다.

Abstract When accessing objects in an object-oriented database, performance can be greatly improved by prefetching objects efficiently. In this paper we present a new object prefetch policy, dynamic SEOF(Selective Eager Object Fetch), which prefetches objects only from selected candidate pages without using any high-level object semantics. Our policy considers both correlations and frequencies of fetching objects when selecting candidates to prefetch. Unlike existing prefetch policies, dynamic SEOF utilizes the memory and the swap space of clients efficiently by adjusting its behavior dynamically according to the working environment, and thus improves overall performance. Furthermore, the proposed policy prevents resource exhaustion by limiting the range of utilizations of objects in the buffer and has good adaptability to both the effectiveness of clustering and database size. We show the performance of the proposed policy through experiments over various multi-client system configurations.

1. 서 론

객체지향 DBMS는 객체지향 데이타 모델의 강력한 표현 능력을 바탕으로 CAD/CAM/CASE, 인공지능 전문가 시스템 쉘, 멀티미디어 사무 정보 시스템 등 차세대 응용 분야의 기반 DBMS로서 그 위치를 점차 확고히 하고 있다[1, 2]. 현재 O2[1]와 같은 대다수의 객체지향 DBMS는 고속의 근거리 통신망상에서의 데이타 전송을 기반으로한 클라이언트-서버 형태를 취하고 있다. 그러므로 전체적인 객체 접근 속도의 향상을 위해서는 클라이언트에서의 객체 적중률(hit ratio)을 높임으로써 서버에 대한 요구를 감소시키는 것이 필요하다. 즉, 객체 버퍼 관리의 효율성이 강조되고 있다.

그러나 객체 버퍼의 관리는 페이지 버퍼와는 달리 복잡하며 많은 문제점을 가지고 있어 효율적인 버퍼 관리를 어렵게 하고 있다. 객체 버퍼 관리의 어려움으로는 먼저 객체 버퍼 관리에서 요구되는 많은 메모리 할당과 반환 연산을 들 수 있으며, 더욱이 가변적인 크기의 객체를 다룸으로써 단편화(fragmentation) 문제도 발생한다. 이 밖에 클라이언트간 객체 버퍼의 일관성을 유지하는 문제도 객체 버퍼의 관리를 더욱 어렵게 하고 있다

· 본 연구는 학술진흥재단 자유공모과제 No. 97-3043 "WWW에서의 이질환경 DBMS 접근을 위한 미들웨어 연구"에 의해 수행되었습니다
† 비 회 원 : 삼성전자 네트웍사업부 연구원
jhahn@oopsla.snu.ac.kr
†† 종신회원 : 서울대학교 컴퓨터공학과 교수
hjkim@oopsla.snu.ac.kr
논문접수 : 1997년 10월 6일
심사완료 : 1998년 12월 11일

[3].

이러한 객체 버퍼 관리의 어려움으로 인하여 객체 버퍼 관리에 관한 연구에서는 효율적인 버퍼 교체 알고리즘보다는 주로 객체 선인출(prefetch)을 통해서 객체 히트(hit)를 높이기 위한 노력이 대부분이었다[2]~[5]. 그 예로 객체 사이의 관계 등 고급의 객체 시맨틱을 기반으로한 객체 선인출에 관한 연구[2]를 비롯하여 객체 접근 패턴의 프로화일링(profiling)이나 학습을 통해 객체를 선인출하고자 하는 연구[3, 5]가 있다.

그러나 고급의 객체 시맨틱을 기반으로한 선인출은 복합 객체의 검색에는 적합하나 메소드 내에서의 객체 접근과 같이 일반적인 객체 접근의 패턴 예측에는 곤란하다[6]. 또한 다양한 응용 프로그램의 지원을 위해서 객체를 단순히 바이트 스트림으로만 보는 바이트 서버 형태의 객체 관리자에서는 이러한 고급의 객체 시맨틱을 아는 것이 어렵다. 그리고 객체 접근 패턴의 프로화일링이나 학습은 모든 접근 문맥(context)마다 이를 각각 유지해야 한다는 어려움과 함께 효율적인 적용이 힘들다는 문제를 안고 있다. 또한 지금까지 객체 버퍼링이나 객체 선인출에 관한 연구[2, 3, 7, 8, 9]에서는 다중 클라이언트 환경에서의 성능 영향을 거의 고려하지 않고 있으며, 주로 성능 평가의 대상을 작업 공간이 메인 메모리 내에 포함될 수 있는 작은 규모의 데이타베이스로 한정함으로써 메모리나 스왑 자원의 고갈이나 경쟁(competition)도 고려하지 못하였다.

본 연구에서는 고급의 객체 시맨틱을 사용하지 않고 페이지나 세그먼트를 단위로 선택적인 객체 선인출을 함으로써 객체 버퍼의 공간 활용도를 높이고 객체 접근 성능을 향상시킨 *동적 SEOF (Selective Eager Object Fetch)* 방법을 고안하고, 다중 클라이언트 환경에서 성능 평가 실험을 실시하였다. 본 동적 SEOF는 두개의 FIFO 큐를 사용하여 한 페이지 또는 세그먼트 내의 객체들에 대한 접근의 빈도수와 이들 접근 사이의 상관 관계를 고려하여 선인출할 대상 객체를 선택하는 알고리즘이다. 이러한 선택적인 객체 선인출을 통해 본 알고리즘은 객체 버퍼의 적중률을 높임과 동시에 무분별한 자원의 낭비를 막는다. 동적 SEOF는 본 연구에서 실시한 다양한 성능 평가 실험에서 좋은 성능을 보였으며, 클러스터링의 변화와 데이타베이스의 크기 변화에도 효과적으로 대응하였다.

본 논문의 구성은 다음과 같다. 먼저 제 2 절에서는 객체 접근 속도를 향상시키기 위해 지금까지 연구된 객체 선인출에 관한 기법을 설명하고 관련 연구를 소개하였다. 그리고 제 3 절에서는 새로운 객체 선인출 방법인 동적 SEOF 방법을 제시하였으며, 구현에 관한 몇 가지 사항도 간단하게 살펴보았다. 제 4 절에서는 시뮬레이션을 통해 본 연구에서 제시한 방법에 대한 성능 평가를 실시하고 그 결과를 분석하였다. 끝으로 제 5 절에서는 앞으로의 연구 방향을 제시하였다.

2. 객체 선인출 기법

흔히 차세대 데이타베이스 응용이라고 불리는 CAD/CAM/CASE 등 객체지향 응용 프로그램은 많은 수의 객체를 메모리 버퍼 내에 캐쉬해놓고 이들 메모리 내의 객체에 대해서 많은 직접적인 연산을 행하는 특성을 보인다. 그런데 이들 객체를 버퍼 공간에 페이지 단위로 유지하는 것은 불필요한 객체들까지 함께 가지고 있어야 함으로 바람직하지 못하다. 특히 순수한 페이지 단위의 버퍼 관리는 클러스터링이 나쁜 경우 버퍼 공간의 활용도가 더욱 떨어진다[3, 8, 10].

이를 해결하기 위해 Itasca[11], Ontos[12], Versant[13] 등을 비롯한 많은 객체지향 DBMS에서는 객체 단위의 버퍼링을 취하고 있다. 즉, 페이지 버퍼 위에 객체 버퍼를 두고 필요한 객체를 페이지 버퍼로부터 객체 버퍼로 인출(fetch)하여 사용한다. 이 구조를 흔히 이원적-버퍼링(dual-buffering) 구조라 하며, 이원적-버퍼링 구조는 순수한 페이지 버퍼의 단점을 보완하여 좋은 버퍼 공간 활용도를 얻을 수 있다는 장점을 가지고 있다[3, 10]. 또한 효율적인 조각 모음(garbage collection)을 가능케 한다.

이러한 이원적-버퍼링 구조상에서 객체지향 데이타베이스 응용 프로그램의 객체 접근 속도는 객체 버퍼의 적중률에 가장 큰 영향을 받는다. 그러므로 객체 접근의 성능 향상을 위해서는 효율적인 객체 버퍼 관리가 요구된다. 그러나 이미 앞서 언급한 바와 같이 객체 버퍼 관리에는 페이지 버퍼 관리와는 달리 어려운 문제가 많이 존재한다. 특히 FIX/UNFIX 프로토콜[14]을 통해 데이타의 사용 기간을 알 수 있는 페이지 버퍼와는 달리 객체 버퍼에서는 객체 접근의 시간적 경계를 알기가 어렵다. 그러므로 객체를 접근할 때마다 접근 기록을 유지하고 이를 바탕으로 LRU(Least Recently Used)와 같은 알고리즘에 따라 버퍼 교체를 하는 것은 불가능하거나, 아니면 성능에 많은 부담을 줄 수 있다. 더욱이 *load-work-save*로 이루어지는 일반적인 객체지향 데이타베이스 응용 프로그램의 작업 주기(working cycle)에서는 버퍼 교체에 대한 중요성이 상대적으로 적다[15]. 이에 O2, Objectivity/DB[16], Versant, Mneme[15], EPVM[9] 등 많은 시스템에서는 별도의 객체 버퍼 관

러 없이 인출된 객체를 트랜잭션의 종료나 객체에 대한 참조가 종료되었음을 명시적으로 알릴 때까지 메모리 내에 유지한다.

그러므로 객체 접근 속도를 향상시키기 위한 객체 버퍼 적중률의 증가는 효율적인 객체 버퍼 관리보다는 효율적인 객체 선인출을 통해서 이루어질 수 있다. 특히 객체지향 데이타베이스 응용과 같이 읽기/쓰기 비율이 높고, 데이타의 공유도가 낮은 응용 환경에서는 선인출된 객체가 다른 클라이언트에 의해 무효화될 가능성이 적기 때문에 객체 선인출이 더욱 유리하다[2, 5, 15]

객체 선인출 방식에는 선인출할 객체를 선택하는 방식에 따라서 크게 세 가지 방식으로 나눌 수 있다. 먼저, 첫 번째 방식은 객체 미스(miss)가 발생할 경우 해당 객체뿐만 아니라 그 객체가 속한 페이지나 세그먼트 내의 모든 객체를 함께 인출하는 방법이다. 이러한 객체 선인출 방식을 취한 시스템으로는 ORION[10], *ENCORE*[17] 등이 있다. 적극적 객체 선인출(eager object prefetch) 방식은 객체 미스를 줄이게 되어 좋은 객체 접근 속도를 보장한다. 또한 결과적으로 객체 미스의 감소는 서버에 대한 페이지 요구를 줄여 서버의 부담을 작게 한다. 그러므로 다중 클라이언트 환경에서 전체적인 반응 시간의 향상을 기대할 수 있다.

그러나 페이지나 세그먼트를 단위로 무조건적인 객체 선인출을 하는 것은 불필요한 객체 인출이 발생할 수 있기 때문에 메모리 복사로 인한 오버헤드나 버퍼 공간의 낭비를 유발할 수 있다. 또한 버퍼 공간의 낭비는 스왑으로 인한 많은 페이지 폴트를 일으킬 수 있다. 그러므로 이 방식은 클러스터링이 좋거나 작업 공간이 작은 경우에는 우수하나, 그렇지 않은 경우에는 많은 성능 저하를 일으킬 수 있다.

다음 두 번째 방식은 객체 시맨틱을 사용하여 선인출할 객체를 결정하는 방식이다. 즉, 객체 미스시 객체와 재귀적으로 연결된 모든 객체를 선인출의 대상으로 하는 방식이다. Chang과 Katz[2]는 객체지향 데이타베이스 응용 프로그램의 접근 패턴을 조사 분석하고, 이들 응용 프로그램에게 만족할 만한 객체 접근 성능을 제공하기 위해서 상속이나 객체간 참조 관계 등을 이용한 클러스터링 방법과 객체 버퍼 관리 방법을 제시하였다. 이들은 사용자의 힌트와 객체간 참조 관계를 따라서 객체를 선인출하거나 버퍼내의 관계된 페이지의 우선 순위를 높임으로써 좋은 버퍼 응답 시간을 얻고자하였다.

이 방식은 클러스터링이 잘 되어 있는 복합 객체의 접근에는 매우 좋은 성능을 발휘한다. 그러나 클러스터링은 객체간 여러 관계가 존재할 경우, 객체 접근 양식에 따라 그 효용 가치를 갖지 못할 수 있다[3, 5] 즉, 저장된 클러스터링과 대립되는 방식으로 접근하거나 객체간의 연결 고리가 방대하게 이루어진 경우에는 적극적 객체 선인출 방식 이상으로 많은 양의 불필요한 객체가 선인출될 수 있으며, 잘못된 선인출로 인한 부담 또한 증가하게 된다. 그리고 이미 앞서 언급한 바와 같이 고급의 객체 시맨틱을 기반으로한 선인출은 일반적인 객체 접근의 예측이 곤란하며 바이트 서버 형태의 객체 관리자에서는 구현하기 어렵다.

세 번째 방식은 연관 메모리(*associative memory*)를 사용하여 객체 접근 패턴을 학습하거나 객체 접근을 프로화일링함으로써 객체 접근 패턴을 미리 예측하는 방식이다. 이들은 객체 클러스터링이 하나의 주 관계를 따라서만 가능하기 때문에 객체간 다중 관계가 존재할 경우 그다지 효과적이지 못하다는 점과 하위의 객체 관리자에서는 객체 시맨틱의 이용이 어렵다는 점에 근거를 두고 있다. 객체 접근의 프로화일링에 기반한 방법에서는 객체 접근 패턴이나 버퍼 관리에 필요한 여러 힌트를 프로화일에 기록하고 이를 객체 선인출에 이용함으로써 객체 버퍼의 히트를 높이고자 하였다[3]. 그리고 Palmer와 Zdonik[5]은 연관 메모리를 사용하여 객체 접근 패턴을 학습하고 이를 바탕으로 객체 선인출함으로써 객체 접근 속도를 향상시키고자 하였다. 그러나 같은 데이타를 사용하는 경우에도 응용 프로그램에 따라서 접근 패턴이 다를 수 있기 때문에 프로화일링이나 접근 패턴의 학습은 접근 문맥을 단위로 각각 이루어져야 하나, 모든 접근 문맥마다 객체 접근의 프로화일을 유지하거나 패턴을 인식하는 것은 관리나 정확도의 유지가 어렵다.

이러한 객체 선인출에 관한 연구 이외에 페이지 단위의 버퍼링과 객체 단위의 버퍼링을 동시에 취함으로써 보다 복잡하고 다양한 객체 참조 패턴의 변화에 쉽게 적응하여 버퍼 공간의 사용 효율성을 높이고자 한 노력이 있었다[8]. 여기서는 먼저 페이지 단위의 버퍼 관리를 통해 같은 페이지 내에 있는 다른 객체의 선인출 효과를 얻음과 동시에 버퍼 공간이 부족한 경우 사용중인 객체를 객체 버퍼로 옮김으로써 버퍼 공간의 사용 효율성을 높이자는 입장이다. 그러나 C나 C++ 언어와 같이 객체의 주소 값을 직접 사용하여 객체를 접근을 하는 경우에는 사용중인 객체의 이동이 어렵기 때문에 페이지 버퍼내의 객체를 후에 다시 객체 버퍼로 이동하는 것은 매우 어렵다.

한편 객체 선인출은 스위즐링 문제와 많은 관련이 있기 때문에 많은 스위즐링 관련 연구에서 함께 다루었다

[7, 9, 15, 18, 19]. 비록 이들 연구들은 주로 다양한 스위즐링 방식에 관한 실험 결과를 보여주고 있으나, 그 결과는 객체 선인출 방법에 대한 성능 평가도 간접적으로 보여주고 있다.

지금까지 설명한 기존의 선인출 알고리즘과는 달리, 본 논문에서 제안하는 동적 SEOF 알고리즘은 버퍼 사용을 제한하여 무분별한 객체 선인출로 인한 자원의 낭비를 막을 수 있으며, 객체간 참조 관계와 같은 고급의 객체 시맨틱을 사용하지 않기 때문에 구현이 용이하다. 또한 객체의 접근 패턴 및 시스템의 부하에 따라 선인출의 대상 및 정도를 동적으로 결정함으로써 이에 대한 적응성이 높다.

3. 동적 SEOF(Selective Eager Object Fetch) 기법

일반적인 이원적-버퍼링 구조에서 객체 접근은 다음과 같이 이루어진다. 먼저 객체 버퍼 내에 접근하고자 하는 객체가 없는 경우, 해당 객체가 있는 페이지를 버퍼 내에 고정시키고 여기서 해당 객체를 객체 버퍼로 복사한다. 마찬가지로 페이지가 버퍼 내에 없으면 해당 페이지를 서버나 디스크 장치로부터 읽어들인다. 따라서 평균 객체 접근 비용은 다음과 같이 계산할 수 있다.

$$\text{평균 접근 비용} = O_{hit} \times C_{O_{hit}} + O_{miss} \times (P_{hit} \times C_{P_{hit}} + P_{miss} \times C_{P_{miss}} + C_{O_{miss}})$$

이때 $O_{hit}(O_{miss})$와 $P_{hit}(P_{miss})$는 각각 객체 버퍼와 페이지 버퍼의 적중률(부재율)을 의미하며, $C_{O_{hit}}(C_{O_{miss}})$와 $C_{P_{hit}}(C_{P_{miss}})$는 각각 객체 버퍼와 페이지 버퍼의 히트(미스)시 드는 비용을 나타낸다. 그런데 $C_{P_{miss}}$는 클라이언트와 서버간의 미스된 페이지에 대한 요구 및 응답의 비용을 의미하므로 그 비용이 매우 높다. 따라서 객체 접근 속도의 향상은 P_{miss}와 O_{miss}를 줄임으로써 가능하다[1].

먼저 P_{miss}는 효율적인 페이지 교체 알고리즘을 채용함으로써 페이지 버퍼의 적중률을 높일 수 있다. 페이지 버퍼 관리 알고리즘으로는 일반적으로 가장 흔히 쓰이고 있는 LRU를 비롯하여 LRU-K[20]나 2Q[21] 등 많은 알고리즘이 제안되었으며 객체지향 DBMS에서도 이를 적용할 수 있다[2].

객체 버퍼의 경우에는 앞서 언급한 바와 같이 효율적인 객체 버퍼 교체 알고리즘을 적용하기 어렵기 때문에 O_{miss}를 줄이기 위해서는 효율적인 객체 선인출이 요구된다. 기존 객체 선인출 기법에 대해서는 이미 제 2 절에서 살펴보았다

이번 절에서는 먼저 정적 SEOF 알고리즘을 간단히 살펴본 후, 정적 알고리즘의 문제점을 토대로 이를 확장한 동적 SEOF 알고리즘을 소개하겠다.

3.1 정적 SEOF 알고리즘

일반적인 객체 접근 패턴에서 페이지내의 많은 객체가 사용되는 페이지는 일정한 간격으로 계속적인 페이지 접근이 일어난다. 반면 한, 두 번의 페이지 접근만 간헐적으로 이루어지는 페이지는 페이지내의 일부 객체만이 사용되는 페이지라 할 수 있다. 그러므로 페이지 접근의 빈도수를 통해서 일정한 값이 넘는 페이지를 객체 선인출의 대상으로 선택할 수 있다.

그러나 생성시 일정한 수의 객체가 연속적으로 저장되고, 탐색시에도 항상 연속적으로 같이 접근되는 객체 집단이 있을 수 있다. 이 경우에는 단순히 접근 빈도수만을 통해 객체 선인출의 대상을 선택하는 것은 오류로 나타날 수 있다. 이러한 페이지는 짧은 기간 내에 연속적인 객체 인출이 이루어진 후에 다시 접근되지 않을 가능성이 높기 때문이다.

이러한 문제를 해결하기 위해서는 객체 접근의 상관관계(correlation)를 배제시켜야 한다. 상관 관계 문제는 지금까지 페이지 버퍼를 대상으로 다루어져 왔다. Robinson과 Devarakonda[22]는 기존의 LFU와는 달리 일정한 기간 동안에는 페이지가 사용될지라도 참조 횟수를 증가시키지 않음으로써 참조의 순간적인 지역성을 배제시키는 방안을 제시하였다. 이로써 전체적으로 자주 사용되지는 않지만 짧은 기간 내에 많은 접근이 있는 데이타에 높은 우선 순위를 주는 것을 방지하는 것이다. 그 밖에 LRU-K[20]나 2Q[21]에서도 이와 비슷한 방식으로 참조의 상관 관계를 고려하고 있다.

이와 마찬가지로 페이지 내에 저장된 객체 중 일부만을 연속적으로 인출하는 경우에도 이를 객체 인출의 상관 관계로 보고 객체 선인출의 대상에서 제외시킬 수 있다. 상관된 접근 패턴을 보이는 페이지는 페이지내의 일부 객체만을 사용할 가능성이 높으므로 필요한 객체만을 인출함으로써 버퍼 공간의 낭비를 줄일 수 있다. 또한 이러한 페이지에 대해서는 객체 인출이 비교적 짧은 시간 동안 연속적으로 이루어지므로 페이지 미스가

---

1) 본 연구는 페이지 서버 형태의 클라이언트-서버 구조를 기반으로 하였다. 그러나 객체 서버 형태의 경우에도 이를 쉽게 적용할 수 있다.

2) 그러나 지금까지 객체지향 DBMS의 특성에 기반한 페이지 버퍼 관리 알고리즘에 대한 연구는 없었다.

발생할 가능성은 매우 낮다.

이들 접근의 빈도수와 인출의 상관 관계 배제가 SEOF(Selective Eager Object Fetch) 전략의 기본이 된다. 즉, SEOF 알고리즘은 다음 두 개념에 기반하고 있다.

- 한 페이지에 대해서 짧은 기간 내에 연속적인 객체 인출이 일어나는 경우에는 페이지의 일부 객체만을 사용할 가능성이 높다. 이를 객체 인출의 상관 관계로 간주한다.
- 객체 인출을 위한 비상관적 페이지 접근이 일정한 기간 연속적으로 이루어지면 그 페이지는 앞으로 사용될 많은 객체를 포함하고 있을 가능성이 높다.

이를 바탕으로 하여 SEOF는 다음과 같이 동작한다. SEOF에서는 그림 1과 같이 두개의 FIFO 큐, S_{in}과 S_{out}을 유지한다[3]. 이때 큐의 크기는 각각 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$이다. 먼저 객체 미스를 처리하기 위해 고정된 페이지가 S_{in}큐와 S_{out}큐에 모두 존재하지 않으면 S_{in}큐에 삽입시킨다. 이때 S_{in}큐의 크기가 $Thresh_{S_{in}}$보다 커지면 S_{in}큐에 삽입된 첫 번째 페이지를 삭제하고 삭제된 페이지는 다시 S_{out}큐에 삽입한다. 이때 S_{out}큐의 크기가 $Thresh_{S_{out}}$보다 커지면 S_{in}큐의 경우와 같이 첫 번째 삽입된 페이지를 삭제한다. 그러나 만약 고정된 페이지가 S_{out}큐에 있다면 그 페이지에 대해서 적극적 객체 선인출을 수행한다. 그리고 S_{in}큐에 있다면 무시한다.

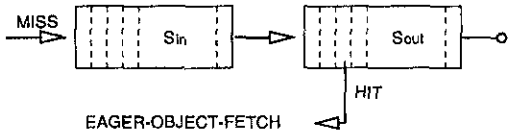


그림 1 SEOF의 S_{in}과 S_{out}큐

여기서 S_{in}큐는 일정 기간의 연속적인 페이지 접근을 하나의 페이지 접근으로 간주함으로써 객체 인출의 상관 관계를 제거하며, S_{out}큐는 정해진 기간 내에 계속적으로 사용되는 페이지를 선택한다. 이로써 SEOF는 페이지내의 객체가 많이 사용될 페이지에 대해서만 선택적으로 객체를 선인출할 수 있다.

자세한 알고리즘의 기술은 알고리즘 1과 같다.

3) 실제 성능 평가 결과 LRU 큐를 사용한 경우에도 비슷한 성능을 나타내었다.

알고리즘 1 정적 SEOF 기법

```
// when the reference to object o, which is located in page p, is invoked

if o is already in the object buffer then
   // do nothing (an object buffer hit)
else
  if p is in S_out then
    dequeue p from S_out
    fetch all uncached objects in p eagerly
  else
    if p is in S_in then
      // do nothing (a correlated access)
    else
      enqueue p into S_in
      if sizeof(S_in) > Thresh_Sin then
        dequeue the first-come entry e_in from S_in
        enqueue e_in into S_out
        if sizeof(S_out) > Thresh_Sout then
          dequeue the first-come entry e_out from S_out
          endif
        endif
      endif
      fetch object o
  endif
endif
return object o
```

3.2 동적 SEOF 알고리즘

SEOF 알고리즘에서의 두 파라미터 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$은 객체 선인출의 정도를 결정하기 때문에 성능에 중요한 변수가 된다. 정성적으로 기술하면, SEOF는 $Thresh_{S_{in}}$을 작게 하고 $Thresh_{S_{out}}$은 크게 할 수록 많은 페이지에 대해서 객체를 선인출하게 되며, 반대로 $Thresh_{S_{in}}$을 크게 하고 $Thresh_{S_{out}}$은 작게 할 수록 선인출의 대상이 줄어든다. 두 파라미터 값의 변화에 대한 SEOF 알고리즘의 행동 양식은 논문 [4]에서 밝힌바 있다.

이렇듯 정적 SEOF가 비-선인출 방식부터 적극적인 선인출 방식까지 다양한 형태의 객체 선인출 양식을 보일 수 있으나, 최적의 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$을 찾는 것은 쉬운 일이 아니다. 또한 특정한 $Thresh_{S_{in}}$이나 $Thresh_{S_{out}}$으로써 모든 동작 환경에서 최상의 성능을 보일 수 도 없다. 따라서 서버나 네트워크 등 시스템의 부하나 물리적 메모리와 스왑 공간 등 시스템 자원을 고려하여 동적으로 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$을 조정하는 것이 요구된다.

논문 [4]의 결과를 통해, 최적의 선인출 정도는 시스템의 부하에 따라 달라짐을 알 수 있다. 서버나 네트워크의 부하가 심한 경우에는, 적극적으로 많은 객체를 선인출하고 이를 클라이언트에 캐쉬함으로써 클라이언트-서버간 상호대화(interaction)를 줄이는 것이 유리하다. 즉, 적극적인 객체 선인출로 서버에 대한 페이지 요구의 응답 시간을 줄임으로써 전반적인 시스템 성능을 향상

시킨다. 그러나 클라이언트의 수가 작아서 시스템의 부하가 낮을 때에는 적극적 선인출이 스왑핑 오버헤드로 인하여 그 장점을 잃는다. 즉, 서버의 응답 시간이 작은 경우, 스왑 I/O 비용이 시스템 성능 저하의 주원인이 된다. 이 경우에는, 오히려 적은 수의 객체를 선인출함으로써 시스템 성능을 향상시킬 수 있다.

이를 반영하여 동적 SEOF는 시스템의 부하를 기반으로 선인출의 정도를 결정하고, 이에 따라 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$을 조정하게된다. 본 동적 SEOF 알고리즘의 동작은 다음과 같다. 먼저 정기적으로 또는 일정한 회수의 SEOF 알고리즘의 수행마다, 현 시스템 부하를 검사하여 어느 정도로 객체를 선인출할 것인가를 결정한다. 여기서 객체 선인출의 정도는 객체 버퍼의 이용도로서 수치화할 수 있다. 즉, (선) 인출된 객체에 대한 실제 사용된 객체의 비율은 선인출의 적극성 정도를 나타낸다. 예를 들면, 비-선인출 전략에서는 인출된 모든 객체를 사용하기 때문에 객체 버퍼의 이용도는 1.0이 된다. 그러나 객체를 적극적으로 선인출하면 할수록 객체 버퍼의 이용도는 떨어지게 된다. 따라서 다음과 같은 함수 F'을 생각할 수 있다.

새로운 버퍼 이용도의 목표 = F'(현 시스템 부하)

여기서 F'은 시스템 부하에 대한 반비례 함수로 현재 시스템 부하가 $LowThreshold_{workloaa}$보다 작으면 일정한 크기(이하 $\Delta_{utilization}$으로 표기)만큼 버퍼 이용도의 목표를 증가시키고, 반대로 현 부하가 $HighThreshold_{workloaa}$보다 크면 $\Delta_{utilization}$만큼 증가시킨다.

그러나, 이때 버퍼 이용도의 목표 값은 자원의 고갈을 막기 위해 제한되어야 한다. 이는 적극적 선인출이 무조건적으로 객체를 선인출함으로써 너무 많은 메모리 및 스왑 자원을 요구할 수 있기 때문이다. 그러므로 동적 SEOF에서는 클라이언트의 자원에 따라 버퍼 이용도의 하한 값을 정하고 그 이하의 버퍼 이용도는 허락하지 않음으로써 자원의 고갈을 막는다. 객체 버퍼 이용도의 제한에 대해서는 4절에서 좀 더 살펴보겠다.

버퍼 이용도의 목표치가 결정된 후, 동적 SEOF는 현재의 버퍼 이용도와 목표치를 비교하여 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$을 조정한다. 즉,

새로운{$Thresh_{S_{in}}$, $Thresh_{S_{out}}$} = F''(버퍼 이용도의 목표치, 현 버퍼 이용도)

함수 F'과 마찬가지로 함수 F''은 현 버퍼 이용도가 목표치보다 작으면 일정한 크기(이하 Δ_{queue}로 표기)만큼 $Thresh_{S_{in}}$은 증가, $Thresh_{S_{out}}$은 감소시킨다. 반대로 현 버퍼 이용도가 목표치보다 크면, $Thresh_{S_{in}}$은 감소, $Thresh_{S_{out}}$은 증가시킨다. 그러나 이때 동적 SEOF가 객체 버퍼 이용도에 너무 민감하게 반응하지 않도록 하기 위해서 그림 2와 같이 윈도우 개념을 사용할 수 있다. 즉, 함수 F''은 현 버퍼 이용도가 목표 이용도 윈도우를 벗어나는 경우에만 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$을 조정함으로써 어느 정도의 차이는 무시한다.

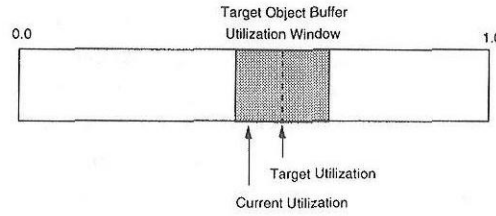


그림 2 객체 버퍼 이용도 윈도우

같은 방식으로 시작시 시스템의 부하를 검사하여 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$의 초기치, 그리고 버퍼 이용도의 초기 목표치를 결정한다.

마지막 고려대상은 클라이언트의 페이지 버퍼 크기이다. 클라이언트의 페이지 버퍼와 객체 버퍼는 동일한 메모리 자원에 대해서 경쟁하는 관계에 있다. 즉, 페이지 버퍼의 크기를 크게 할 수록 페이지 버퍼에 대한 적중률은 높아지나 반대로 스왑으로 인한 페이지 폴트는 증가한다. 따라서 적절한 페이지 버퍼의 크기를 선택하는 것이 중요하다.

또한 최적의 페이지 버퍼 크기는 객체 선인출의 정도와도 관련이 있다. 즉, 선인출이 적극적으로 이루어지면 - 객체 버퍼 이용도가 낮으면 – 작은 페이지 버퍼만이 요구된다. 그 이유는, 적극적 선인출을 통해 이미 많은 객체들이 객체 버퍼에 캐쉬되어 있고, 또한 작은 페이지 버퍼가 상대적으로 큰 객체 버퍼와의 자원 경쟁을 줄일 수 있기 때문이다. 예를 들면, 적극적 선인출 방법은 페이지 내의 모든 객체를 무조건 인출하여 객체 버퍼 내에 캐쉬하기 때문에 단지 하나의 페이지 버퍼 프레임만이 요구된다. 반면, 선인출되는 객체의 수가 작을 때에는 큰 페이지 버퍼가 페이지 버퍼 적중률을 증가시키기 때문에 전체적인 시스템 성능을 향상시킬 수 있다. 이에 동적 SEOF는 객체 선인출의 정도에 따라서 동적으로클라이언트의 페이지 버퍼 크기를 조정한다. 앞서 언급한 바와 같이, 선인출의 정도는 객체 버퍼 이용도로 표현되

며, 따라서 동적 SEOF는 현 객체 버퍼 이용도를 기준으로 페이지 버퍼의 크기를 조정한다. 즉,

새로운 페이지 버퍼의 크기 = F'''(현 객체 버퍼 이용도)

F'''는 현 버퍼 이용도에 비례하여 새로운 페이지 버퍼의 크기를 계산한다. 자세한 동적 SEOF 알고리즘은 알고리즘 2에 기술되어 있다[4]

알고리즘 2 동적 SEOF 기법

```
동적 SEOF
run the static SEOF algorithm
if adjusting Thresh_Sin and Thresh_Sout is required then
  calculate the new target utilization of objects in the buffer ( F' )
  calculate the new Thresh_Sin and Thresh_Sout ( F' )
  resize the S_in and S_out queue according to the new Thresh_Sin and
  Thresh_Sout
  calculate the new page buffer size ( F''' ) and resize the page buffer
end if

Function F'
// Calculate the new target utilization of objects in the buffer
get the current system workload
if the current system workload > HighThreshold_workload then
   move the current target object buffer utilization window left by
Δ_utilization
else if the current system workload < LowThreshold_workload then
   move the current target object buffer utilization window right by
Δ_utilization
end if

Function F''
// Calculate the new Thresh_Sin and Thresh_Sout
if the current utilization is in the right side of the target utilization
window then
  decrease Thresh_Sin and increase Thresh_Sout by Δ_queue
else // the current utilization is in the left side
  increase Thresh_Sin and decrease Thresh_Sout by Δ_queue
end if

Function F'''
// Calculate the new page buffer size
a new page buffer size ⇐ (the current utilization × α + β) ×
      the maximum page buffer size
```

3.3 구현

본 SEOF 알고리즘은 기존 객체 관리자에 쉽게 구현할 수 있다. SEOF 알고리즘은 S_{in}과 S_{out} 등 단지 두 개의 FIFO 큐만을 요구하며, 이들을 관리하기 위한 몇 개 인터페이스만을 구현하면 된다. 또한 SEOF 알고리즘은 객체 버퍼에서만 사용되기 때문에 기존 객체 관리자에 쉽게 이식할 수 있다. 그림 3은 SEOF 알고리즘의 한 구현 예와 객체 버퍼에서의 호출 순서를 보인 것이다

동적 전략의 경우 크기 조정을 위한 함수가 추가되어야 하며, 이 함수는 주기적, 또는 일정한 횟수의 CHECK 함수 호출마다 수행하면 된다. 이 함수는 시스

4) 여기서 함수 F', F'', F'''는 본 실험에서 사용된 함수이다. 그러나 이들 함수는 다른 임의의 좋은 함수로 대치하여 사용할 수 있다.

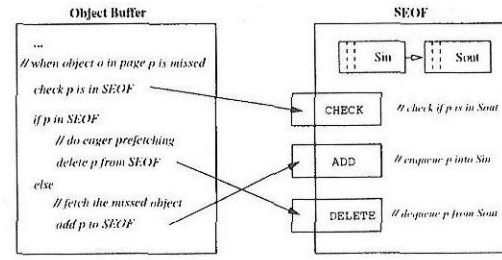


그림 3 SEOF 알고리즘의 호출 순서

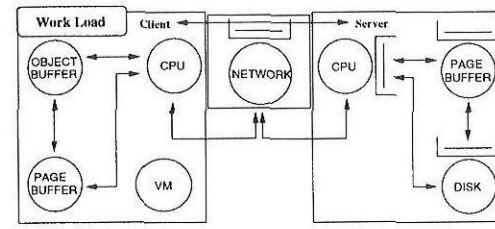


그림 4 시뮬레이션 모델

템의 부하에 따라서 앞서 제시한 알고리즘대로 큐 크기를 재조정하면 된다. 이때 시스템의 부하는 운영체제의 시스템 호출을 사용하거나 클라이언트 수 또는 서버에 대한 요구 횟수로부터 계산할 수 있다.

4. 성능 평가

4.1 시뮬레이션 모델

SEOF 알고리즘의 성능 평가는 이중-버퍼링 구조를 취하는 페이지 서버 형태의 클라이언트-서버 구조를 기반으로 한다[5]. 성능 평가에서 사용한 시뮬레이션 모델은 그림 4와 같다. 서버는 PAGE BUFFER와 DISK 그리고 CPU로 구성되며 각 클라이언트는 OBJECT BUFFER와 PAGE BUFFER, CPU, VM 등으로 구성된다. 클라이언트와 서버사이에는 NETWORK으로 연결되어 있다. 그림 4에서 화살표는 각 구성 요소간 작업 요구(또는 자원 사용 요구)와 이에 대한 응답 흐름을 나타낸 것이다.

각 클라이언트는 불필요한 객체 선인출로 인해서 발생하는 스왑 I/O에 의한 페이지 폴트를 모델링하기 위

5) 그러나 본 성능 평가의 결과는 객체 서버의 경우에도 적용할 수 있다. 객체 서버(또는 객체-그룹 서버)인 경우에는 서버 내에서의 페이지 버퍼와 클라이언트의 객체 버퍼사이에서 취할 수 있는 객체 선인출에 대한 성능 평가로 해석할 수 있다.

해서 VM을 갖는다. VM은 Page Aging[23]에 기반하여 사용자의 가상 메모리 공간과 물리적 메모리를 관리한다. VM은 일정한 시간 간격으로 물리적 메모리 내에 존재하는 가상 메모리 블럭의 AGE를 하나씩 증가시키고 메모리 블록을 접근할 때 그 블럭의 AGE를 0으로 리셋시킨다. 그리고 물리적 메모리가 필요한 경우 AGE가 큰 순서대로 일정한 수의 메모리 블럭을 스왑-아웃시킴으로써 물리적 메모리를 확보한다. 본 실험에서 가상 메모리 블럭의 크기는 4 K 바이트이다.

다음 PAGE BUFFER는 LRU를 기반으로 페이지 버퍼 관리를 담당하며, 모든 인출된 객체는 OBJECT BUFFER 내에서 트랜잭션 종료까지 유지된다.

표 1은 본 실험에서 사용한 주요 파라미터의 값이다. 이 값들은 본 연구실에서 개발한 페이지 서버 형태의 객체 저장 시스템을 프로화일링하여 얻은 값이다.

표 1 시뮬레이션에서 사용한 주요 파라미터

파라미터		값 (*msec*)
Object copy time		0.003/object
Avg Object buffer processing time		0.077
Avg Page buffer processing time		0.025
Avg. Swap I/O time		16.0
Avg. Disk access time		17.0/8 K byte
Network processing time		1.1 + 0.00075/byte
Network transfer time		45 M bits/sec
OO1	Part processing time	5.0/Part
	Connection processing time	0.5/Connection
OO7	Object access time	0.5/object

본 실험에서는 OO1 벤치마크[24]의 traverse 연산을 성능 평가에서의 작업 부하(work load)로 사용하였다. 이때 작업 공간의 크기 변화에 따른 성능 평가를 위해서 Part 객체 50,000개로 구성된 중간 데이타베이스와 200,000개로 구성된 큰 데이타베이스에 대해서 실험을 실시하였다.데이타베이스의 크기는 각각 16 M 바이트와 64 M 바이트이다. 또한 클러스터링의 우수성 정도에 따른 성능 변화를 보기 위해서 90%-1% 규칙으로 생성한 데이타베이스 외에 80%-5% 규칙의 클러스터링을 따르는 데이타베이스에 대해서도 실험을 실시하였다. 여기서 90%-1% 규칙이란 Part 객체 사이의 연결 중 90%는 1%이내의 가까운 Part 객체와 연결시키는 규칙을 말한다.

traverse 연산은 각 클라이언트마다 7 단계까지 재귀적으로 데이타베이스를 탐색해가며, 각 탐색은 10회 수행하였다. 그리고 다중 클라이언트 환경에서의 성능 변화를 비교하기 위해서 클라이언트의 수를 1에서 20까지 변화시키면서 실험하였다.

또한 본 실험에서는 다양한 형태의 객체 접근 패턴에 대한 성능 평가를 위해 OO7 벤치마크[25]의 traversal 1 연산을 사용한 실험도 함께 하였다. 여기서는 4 M 바이트와 25 M 바이트 크기의 작은 데이타베이스와 중간 데이타베이스를 사용하였다.

모든 실험에서는 동적 SEOF 방법에서의 큐 크기와 버퍼 이용도의 목표치의 초기 값을 구하기 위해 실제 측정에 앞서 두 번씩의 예비 수행을 실시하였다.

본 성능 평가를 위한 시뮬레이션은 C++SIM[26]을 사용하여 구현하였다.

4.2 시뮬레이션 결과

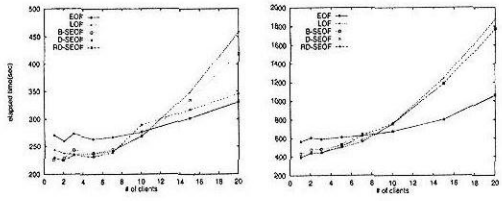
시뮬레이션은 각 벤치마크 데이타베이스에 대해서 페이지를 단위로 객체를 적극적으로 선인출하는 전략(EOF)과 필요한 객체만을 인출하는 방법(LOF), 그리고 SEOF에 대해서 수행하였다. EOF와 LOF의 경우에는 클라이언트에서의 페이지 버퍼 크기를 각각 8 K 바이트와 4.3 M 바이트로 하였다. 이 값은 본 실험에서 가장 좋은 성능을 보이는 페이지 버퍼 크기를 택한 것이다. 그리고 SEOF에 대해서는 정적 SEOF(B-SEOF)와 동적 SEOF(D-SEOF), 그리고 제한된 동적 SEOF(RD-SEOF)에 대해서 실험하였다. 여기서 RD-SEOF는 D-SEOF와 동일하나, 단지 자원의 고갈을 막기 위해 객체 버퍼 이용도의 하한 값에 제한을 둔 것이다. 먼저 B-SEOF에 대해서는 다양한 $Thresh_{S_m}$과 $Thresh_{S_{ext}}$에 대해서 실험을 하였으나, 본 논문에서는 객체 버퍼 공간의 사용 크기가 약 10 M 바이트 내외가 되면서 전반적으로 좋은 성능을 보인 것 중 하나의 결과를 보였다. 이때 $Thresh_{S_m}$과 $Thresh_{S_{ext}}$의 값은 모두 120이었으며, 2.7 M 바이트 크기의 클라이언트 페이지 버퍼를 사용하였다. 동적 버전인 D-SEOF와 RD-SEOF에 대해서는 네트워크 부하만을 고려하였는데, 그 이유는 본 실험에서 네트워크가 절대적인 성능 저하 요인이었기 때문이다. 상위와 하위 시스템 부하 임계값은 각각 0.55와 0.3이었으며, $\Delta_{utilization}$은 0.1이었다. 실험에서 현 객체 버퍼 이용도가 목표치의 20%를 벗어나는 경우 $Thresh_{S_m}$과 $Thresh_{S_{ext}}$을 조정하였으며[6] Δ_{queue}는 8이었다[7].

6) 본 실험에서는 이용도의 목표 값에 따라 이용도의 변화도가 다르다고 생각하였기 때문에 목표 윈도우의 크기를 가변적으로 하였다. 그러나 고정된 크기의 윈도우하에서도 비슷한 성능을 보였다.

7) $\Delta_{utilization}$ 및 Δ_{queue}의 크기에 따라 시스템 부하의 변화에 따른

RD-SEOF의 경우 약 12 M 바이트 정도까지 버퍼를 사용하도록 버퍼 이용도의 하한 값을 조정하였다. 모든 실험에서 서버 페이지 버퍼의 크기는 5.5 M 바이트이고, 클라이언트의 물리적 메모리 크기는 5 M 바이트이다.

먼저 90%-1%의 클러스터링 규칙으로 생성한 OO1 데이타베이스에 대한 실험 결과는 다음과 같다.



(a) 중간데이타베이스 (b) 큰 데이타베이스

그림 5 평균 수행 시간 (초) (OO1, 90%-1%)

그림 5는 OO1 traverse 연산을 총 10회 수행했을 때 걸리는 평균 시간을 보인 것인데, 기대한 바와 같이 클라이언트 수가 10이하인 경우 비적극적 선인출 전략이 적극적 선인출 전략보다 우수하다. 그 이유는 클라이언트 수가 작을 경우에는 많은 객체를 선인출함으로써 발생하는 스왑 및 페이지 폴트 오버헤드가 서버에 대한 페이지 요구 감소로 인한 이득보다 크기 때문이다. 그러나 클라이언트의 수가 증가하여 서버에 대한 페이지 요구가 많아짐에 따라서 서버의 반응 시간이 점차 느려지는데, 이때 비적극적 선인출 전략의 경우 그 증가폭이 훨씬 크다. 즉, 서버에 대한 페이지 요구 감소로 인한 이득이 페이지 폴트로 인한 손실보다 커져 적극적 선인출 전략이 비-선인출 전략보다 우수하게 나타난다. 그림 5에서는 또한 클라이언트의 수가 1인 경우에 비해 2나 3인 경우가 다소 좋은 성능을 보이고 있다. 그 이유는 클라이언트의 수가 늘어남에 따라 다른 클라이언트가 서버에게 요구한 페이지를 자신이 사용할 수 있기 때문에 서버의 페이지 버퍼 적중률이 높아지기 때문이다. 그러나 점차 클라이언트 수가 늘어남에 따라서 서버의 처리 한계를 넘어서기 때문에 성능이 다시 떨어진다.

SEOF의 경우를 살펴보면, 먼저 중간 데이타베이스에서는 B-SEOF가 EOF와 LOF의 중간 정도의 성능을 보이고 있다. 그러나 큰 데이타베이스에서는 LOF와 유사한 형태를 보이고 있는데, 그 이유는 작업 공간이 커짐에 따라서 선인출의 대상이 줄고 객체 적중률이 떨어져서 서버에 대한 페이지 요구가 늘어나기 때문이다. 그러나 D-SEOF는 거의 모든 경우에서 가장 좋은 성능을 보이고 있다. D-SEOF는 시스템의 부하가 작을 때는 적은 수의 객체만을 선인출하고, 시스템의 부하가 커질수록 더 적극적으로 객체를 선인출한다. 이러한 D-SEOF의 행동 양식은 수행중 사용한 객체 버퍼의 크기를 나타낸 표 2에서도 분명하게 나타난다. 이 결과는 시스템의 부하에 따라서 동적으로 큐의 크기를 조정함으로써 메모리나 스왑 자원을 효율적으로 이용하는 동적 SEOF의 성질을 잘 보여준 것이다.

표 2 객체 버퍼 크기 (M 바이트) (OO1, 90%-1%)

전략 \ 클라이언트 수		1	2	3	5	7	10	15	20
중간 DB	EOF	13.0	12.8	13.1	13.0	12.9	13.0	13.1	12.9
	LOF	2.1	2.1	2.2	2.2	2.1	2.2	2.1	2.1
	B-SEOF	4.9	5.4	5.5	5.3	5.4	5.3	5.4	5.2
	D-SEOF	2.1	2.1	2.2	2.1	2.2	13.0	13.0	13.0
	RD-SEOF	2.1	2.1	2.2	2.1	2.2	12.8	13.0	12.8
큰 DB	EOF	37.0	39.0	37.7	38.0	37.9	38.0	37.6	37.9
	LOF	2.8	2.8	2.8	2.8	2.8	2.8	2.7	2.8
	B-SEOF	11.3	10.0	10.6	10.5	10.2	10.4	10.2	10.3
	D-SEOF	2.8	2.8	2.8	4.1	5.6	38.0	37.5	38.2
	RD-SEOF	2.8	2.8	2.8	4.3	7.0	9.1	9.6	10.3

중간 데이타베이스에서 클라이언트 수가 작을 때 D-SEOF와 RD-SEOF는 모두 LOF보다 좋은 성능을 보이는데, 그 이유는 이들이 사용한 객체 버퍼의 크기는 서로 비슷한데 반해 LOF가 다른 동적 전략들 보다 더 큰 페이지 버퍼를 사용함으로써 좀더 많은 스왑핑을 유발하기 때문이다.

마지막으로 제한된 동적 버전은 표 2에 나타난 바와 같이 버퍼 공간의 사용에 제한을 받는다. 따라서 시스템의 부하가 클 경우 D-SEOF보다 성능이 나쁘다. 그러나 시스템의 부하가 작은 경우에는 동적 SEOF들이 모두 LOF와 비슷한 성질을 보이기 때문에 버퍼 공간 크기에 대한 제한에 영향을 받지 않는다. 한편 중간 데이타베이스에서 RD-SEOF가 클라이언트 수가 10일 때 다소 성능이 떨어지는 것은 EOF나 D-SEOF 등 다른 전략에 비해 더 큰 클라이언트 페이지 버퍼를 사용하여 더 많은 스왑핑을 유발하기 때문이다.

본 실험에서 비록 시스템 부하가 심한 경우 적극적 선인출이 객체 접근 속도를 크게 향상시킬 수 있었으나, 객체 선인출로 인해서 요구되는 스왑 공간의 크기가 너무 커서 현실적으로 이를 적용하기 어려울 수 있다. 즉,

반응의 민감성이 결정된다. 이에 대한 연구는 현재 진행 중에 있다.

스왑 공간의 크기를 물리적 메모리 공간의 2 배정도로 하는 것이 일반적이기 때문에 5 M 바이트의 물리적 메모리를 가지고 있는 상황에서 약 40 M 바이트 정도의 스왑 공간을 사용하는 것은 무리가 있다. 이것이 본 연구에서 클라이언트의 자원에 따라 객체 선인출의 정도를 제한할 수 있는 RD-SEOF 전략을 포함한 이유이다.

본 실험에서 D-SEOF와 RD-SEOF에서 사용하는 시스템 부하의 임계값은 다소 적극적인 선인출을 하도록 조정하였다. 그 이유는 본 실험에서 시스템 부하, 특히 네트워크 부하가 클라이언트의 수가 증가함에 따라 매우 급격하게 증가했기 때문이다. 따라서, 동적 SEOF 전략들은 다소 급격한 행동의 변화를 보이고 있다.

다음은 클러스터링 변화에 따른 성능의 변화를 알아보겠다.

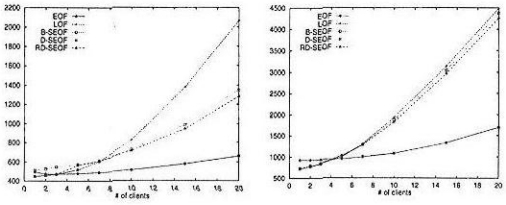
먼저 그림 6은 80%-5% 클러스터링 규칙으로 생성한 데이타베이스에 대한 실험 결과이다. 이 그래프는 전체적으로 90%-1% 클러스터링의 경우와 비슷한 양상을 보이고 있다. 그러나 80%-5% 클러스터링의 경우, 클라이언트 수가 증가함에 따라 나타나는 LOF의 성능 하락은 90%-1% 클러스터링의 경우에 비해 더욱 큰 폭으로 떨어지고 있다. 그 이유는 클러스터링이 나빠짐으로 해서 LOF의 적중률은 급격히 떨어지고, 따라서 서버에 대한 페이지 요구는 더욱 많이 나타나기 때문이다. 그리고 이로 인한 성능 저하는 EOF에서 페이지 폴트 증가로 인한 성능 저하의 폭보다 더욱 크다. 따라서 LOF와 EOF의 성능 교차점 역시 조금 왼쪽으로 이동하였다.

B-SEOF의 경우 중간 데이타베이스에 대해서는 이전 결과에 비해 EOF에 좀더 가깝게 나타난다. 그 이유는 중간 데이타베이스에 대한 traverse는 작업 공간이 작아서 80%-5%의 규칙으로 클러스터링된 경우에도 여전히 한 페이지 내에 사용할 객체의 비율이 높은 반면, 클러스터링이 낮아짐에 따라서 데이타베이스 생성시 생기는 사이클은 줄기 때문이다. 이로써 B-SEOF에 의해서 객체 선인출의 대상으로 선정되는 페이지는 늘어난다. 그러나 큰 데이타베이스에 대해서는 B-SEOF가 90%-1% 클러스터링의 경우에 비해 LOF에 더 가까운 형태로 나타난다. 이는 클러스터링이 나빠짐에 따라서 한 페이지 내에 사용될 객체의 수가 줄고 그만큼 B-SEOF에 의해서 객체 선인출의 대상이 되는 페이지가 줄기 때문이다.

동적 SEOF 알고리즘의 경우에는 이전 실험과 마찬가지로 좋은 성능을 보이고 있다. 그런데 RD-SEOF의 경우, 90%-1% 클러스터링의 경우와 비교하였을 때 중간 데이타베이스에서는 좀더 B-SEOF와 비슷한 양상을 보이나 큰 데이타베이스에는 두 전략의 차이가 더 벌어진다. 그 이유는 먼저 중간 데이타베이스에서 B-SEOF는 클러스터링이 나빠짐으로 인해서 좀더 많은 버퍼 공간을 사용하는데 반해, 큰 데이타베이스에서는 RD-SEOF가 높은 부하로 인해 더 많은 객체를 선인출하기 때문이다.

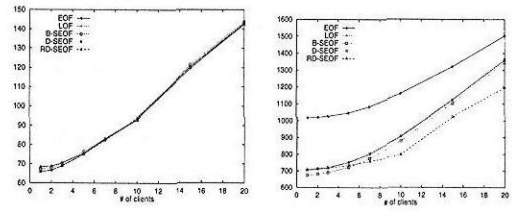
본 연구에서는 또한 OO7 traversal 1 연산을 사용하여 성능 평가를 실시하였다.

그림 7은 traversal 1 연산을 수행하였을 때의 시간을 보인 것이다. 작은 데이타베이스에서는 traversal 연산의 높은 지역성으로 인해서 모든 전략의 성능이 비슷하다. 또한 작업 공간 역시 모두 클라이언트의 메모리에



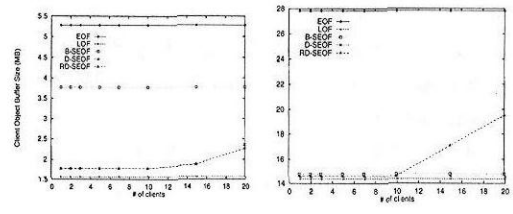
(a) 중간 데이타베이스 (b) 큰 데이타베이스

그림 6 평균 수행 시간 (초) (OO1, 80%-5%)



(a) 작은 데이타베이스 (b) 중간 데이타베이스

그림 7 평균 수행 시간 (초) (OO7)



(a) 작은 데이타베이스 (b) 중간 데이타베이스

그림 8 객체 버퍼 크기 (M 바이트) (OO7)

캐쉬될 정도로 작기 때문에 페이지 요구에 대한 응답 시간 역시 전략간 차이가 거의 없다. 그러나, 그래프에는 잘 나타나지 않지만 EOF의 성능이 다른 전략에 비해 다소 나쁘다. 그 이유는 다음과 같다. 작은 데이타베이스의 경우 거의 모든 데이타베이스가 클라이언트에 캐쉬되기 때문에 대부분의 객체 인출이 클라이언트 내에서 이루어지며, 따라서 객체 인출의 응답 시간이 전략에 상관없이 거의 동일하다. 그러나 EOF의 경우에는 많은 불필요한 객체를 선인출함으로써 생기는 복사 오버헤드를 피할 수 없으며, 이 때문에 성능의 저하가 생긴다. 같은 이유로 중간 데이타베이스에서도 LOF가 EOF에 비해 유리하다. 그러나 EOF의 경우 너무 많은 객체를 선인출함으로써 많은 스왑핑이 발생하고, 이에 성능 저하가 더욱 심해진다 (그림 8 참조).

반면 모든 SEOF 전략은 LOF보다 다소 성능이 우수하다. 그 이유는 SEOF 전략이 적당한 양의 객체를 선인출함으로써 (스왑핑의 유발 없이) 페이지 요구의 수를 줄였기 때문이다. SEOF 전략 중, 특히 클라이언트 수가 작은 경우 B-SEOF가 다른 동적 SEOF에 비해 성능이 우수한데, 그 이유는 객체 버퍼의 사용량을 표시한 그림 8에 나타난 바와 같이, B-SEOF가 좀 더 많은 객체를 선인출하도록 조정되었기 때문이다. 한편 특이하게도 RD-SEOF는 7 이상의 클라이언트 수에서 매우 좋은 성능을 보이고 있다. 이는 RD-SEOF의 경우, D-SEOF의 경우보다 시스템의 부하가 높아서 적극적으로 좀더 많은 객체를 선인출하였기 때문이다[8].

5. 결론 및 향후 연구

본 연구에서는 고급의 객체 시맨틱을 사용하지 않고 페이지나 세그먼트를 단위로 선택적인 객체 선인출을 함으로써 객체 버퍼의 공간 활용도를 높인 동적 SEOF(Selective Eager Object Fetch) 방법을 고안하고, 다중 클라이언트 환경에서 성능 평가 실험을 하였다. 동적 SEOF는 사용할 객체가 많은 것으로 보이는 페이지만을 대상으로 선택적인 객체 선인출을 함으로써 객체 버퍼의 적중률을 높임과 동시에 무분별한 자원의 낭비를 막는다. 또한 시스템의 부하에 따라서 그 행동 양식을 적절히 조절함으로써 성능 향상을 꾀하며, 또한 구현이 용이하다.

본 실험에서 동적 SEOF (D-SEOF)는 거의 모든 경우에서 가장 좋은 성능을 보였으며, 제한된 동적 SEOF (RD-SEOF) 경우에도 객체 버퍼 사용을 효율적으로 제한할 수 있음을 보여주었다. 클러스터링의 변화와 데이타베이스의 크기 변화에 따른 실험결과를 통해 동적 SEOF 전략이 이에 잘 대응함을 알 수 있으며, 또한 OO7 실험에서도 좋은 성능을 보여주었다. 이러한 결과를 통해 본 동적 SEOF 알고리즘은 다양한 실제 환경에서도 좋은 성능을 발휘할 수 있으리라 기대할 수 있다. 더욱이 본 알고리즘은 두개의 FIFO 큐의 관리만을 요구함으로 실행에 따른 오버헤드가 작다.

앞으로의 연구 과제로는 선인출한 객체 중 일정 기간 사용되지 않는 객체에 대해서 효율적인 객체 버퍼 교체를 함으로써 객체 공간의 활용도를 높이는 방법을 찾는 것과 시스템 부하는 물론 객체 접근 패턴에 따라 동적으로 $Thresh_{S_{in}}$과 $Thresh_{S_{out}}$을 조절하는 방안을 고안하는 것이다. 이들 방법은 동적 SEOF의 성능을 더욱 높일 것이다.

8) 중간데이타베이스의 실험에서 순수 사용 객체의 양이 14 M 바이트를 넘기 때문에 RD-SEOF의 객체 버퍼 사용 하한선은 20 M 바이트로 하였다.

참 고 문 헌

[1] F. Bancilhon, C. Delobel and P. Kanellakis, editors, *Building an Object-Oriented Database System: The Story of O2*, Morgan Kaufmann Publishers, Inc., 1992.

[2] E. E. Chang and R. H. Katz, "Exploiting Inheritance and Structure Semantics for Effective Clustering and Buffering in an Object-Oriented DBMS," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1989.

[3] J. R. Cheng and A. R. Hurson, "On The Performance Issues of Object-Based Buffering," In *Proceedings of International Conference on Parallel and Distributed Information Systems*, Dec. 1991.

[4] J.-H. Ahn and H.-J. Kim, "SEOF: An Adaptable Object Prefetch Policy For Object-Oriented Database Systems," In *Proceedings of the International Conference on Data Engineering*, Birmingham, U.K, Apr. 1997.

[5] M. Palmer and S. B. Zdonik, "Fido: A Cache That Learns to Fetch," In *Proceedings of the International Conference on Very Large Data Bases*, Sept. 1991.

[6] D. J. Dewitt and D. Maier, "A Study of Three Alternative Workstation-Server Architectures for Object-Oriented Database Systems," In *Proceedings of the International Conference on Very Large Data Bases*, Aug. 1990.

[7] A. L. Hosking and J. E.B. Moss, "Object Fault

Handling for Persistent Programming Languages: A Performance Evaluation," In *Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications*, Oct. 1993.

[8] A. Kemper and D. Kossmann, "Dual-Buffering Strategies in Object Bases," In *Proceedings of the International Conference on Very Large Data Bases*, Sept. 1994.

[9] S. J. White and D. J. DeWitt, "A Performance Study of Alternative Object Faulting and Pointer Swizzling Strategies," In *Proceedings of the International Conference on Very Large Data Bases*, Aug. 1992.

[10] W. Kim, J. F. Garza, N. Ballou and D. Woelk, "Architecture of the ORION Next-Generation Database System," *IEEE Transactions on Knowledge and Database Engineering*, Vol. 2, No. 1, Mar. 1990.

[11] IBEX Object Systems, Inc., "ITASCA Technical Summary Release 2.3", 1995.

[12] Ontos, Inc., "ONTOS Product Description," 1996.

[13] Versant Object Technology Corp., "Versant OODBMS Release 4," 1996.

[14] W. Effelsberg and T. Haerder, "Principles of Database Buffer Management," *ACM Transactions on Database Systems*, Vol. 9, No. 4, Dec. 1984.

[15] J. E. B. Moss, "Working with Persistent Objects: To Swizzle or Not to Swizzle," *IEEE Transactions on Software Engineering*, Vol. 18, No. 8, Aug. 1992.

[16] Objectivity, Inc., "Objectivity/DB Technical Overview Version 3," 1995.

[17] M. F. Hornick and S. B. Zdonik, "A Shared, Segmented Memory System for an Object-Oriented Database," *ACM Transactions on Office Information Systems*, Vol. 5, No. 1, Jan. 1987.

[18] A. Kemper and D. Kossmann, "Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis," In *Proceedings of the International Conference on Data Engineering*, Apr. 1993.

[19] P. R. Wilson and S. V. Kakkad, "Pointer Swizzling at Page Fault Time: Efficiently and Compatibly Supporting Huge Address Spaces on Standard Hardware," In *Proceeding of the International Workshop on Object Orientation in Operating Systems*, Sept. 1992.

[20] E. J. O'Neil, P. E. O'Neil and G. Weikum, "The LRU-K Page Replacement Algorithm For Database Disk Buffering," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1993.

[21] T. Johnson and D. Shasha, "2Q: A Low Overhead High Performance Buffer Management Replacement Algorithm," In *Proceedings of the International Conference on Very Large Data Bases*, Sept. 1994.

[22] J. T. Robinson and M. V. Devarakonda, "Data Cache Management Using Frequency-Based Replacement," In *Proceedings of the ACM SIGMETRICS Conference*, May 1990.

[23] M. J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall Inc, 1986.

[24] R. G. G. Cattell and J. Skeen, "Object Operations Benchmark," *ACM Transactions on Database Systems*, Vol. 17, No. 1, Mar. 1992.

[25] M. Carey, D. J. DeWitt and J. F. Naughton, "The OO7 benchmark," In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Washington, DC, May 1993.

[26] Department of Computing Science, University of Newcastle upon Tyne, "C++SIM User's Guide," public release 1.5 edition. http://cxxsim.ncl.ac.uk.

안 정 호

1991년 2월 서울대학교 컴퓨터공학과 졸업. 1993년 2월 서울대학교 컴퓨터공학과 공학석사. 1998년 8월 서울대학교 컴퓨터공학과 공학박사. 1998년 3월 ~ 현재 삼성전자 네트웍사업부 선임연구원. 관심분야는 객체지향 시스템, 데이타베이스, 분산 시스템.

김 형 주

제 26 권 제 1 호(B) 참조