멀티미디어 저장 시스템을 위한 사용자 요청 통합
(Unifying User Requests for Multimedia Storage Systems)

황 인 준 †
(Eenjun Hwang)

요 약 대부분의 멀티미디어 저장 시스템은 사용자의 요청을 라운드-로빈(round-robin) 방식으로 수행한다. 즉, 사용자는 일정한 간격을 두고 정해진 시간 동안 저장 시스템의 서비스를 받게 된다. 더욱이, 여기에 사용되는 대부분의 알고리즘들은 시스템과 밀접하게 연관된 비용 함수를 바탕으로 평가를 받게 된다. 이러한 알고리즘의 예로서는 FCFS, SCAN, SCAN-EDF 등이 있다. 본 논문에서는 주어진 사용자 요청에 대해서 사용자 대기 시간이나 최대 디스크 대역폭과 같은 제약 조건들을 만족시키는 동시에 임의의 비용 함수에 대해 최적의 스케줄인 통합 읽기 요청(Unified Read Request)을 생성하는 요청 통합기(Request Unifier) 모듈을 설계한다. 그리고 이러한 사용자 요청 통합에 사용될 수 있는 세 개의 알고리즘을 제안하고 성능 평가를 통해 그들의 장단점을 비교해 본다.
키워드 : 멀티미디어 서버, 요청 통합, 비용 함수

Abstract Most work on multimedia storage systems has assumed that clients will be serviced using a round-robin strategy. The server services the clients in rounds and each client is allocated a time slice within that round. Furthermore, most such algorithms are evaluated on the basis of a tightly coupled cost function. This is the basis of well-known algorithms such as FCFS, SCAN, SCAN-EDF, etc.

In this paper, we describe a scheduling module called Request Unifier(RU) that takes as input, a set of client request, and a set of constraints on the desired performance such as client waiting time or maximum disk bandwidth, and a cost function. It produces as output a Unified Read Request(URR), telling the storage server which data items to read and when these data items to be delivered to the clients. Given a cost function, a URR is optimal if there is no other URR satisfying the constraints with a lower cost. We present three algorithms in this paper that can accomplish this kind of request merging and compare their performance through an experimental evaluation.
Key words : Multimedia Server, Request Unification, Cost Function

1. 서 론

최근 들어, 효과적인 멀티미디어 저장 서버의 구현에 관해서 많은 연구가 진행되어 왔다. 멀티미디어 저장 서버는 대개 사용자의 요청에 대해 순회적으로 서비스를 제공한다. 즉, 서버는 서비스 주기(cycle)를 설정하고 각 사용자에게 주기 내에서 정해진 시간(time slice) 동안 서비스를 제공하는 형태로 주기마다 반복적으로 수행한다.

본 논문에서는, 사용자의 어플리케이션에서 필요로 하는 여러 형태의 미디어 데이타 요청에 대해 주어진 제약 조건(constraints)의 범위 내에서 데이타 요청을 통합하여 서버의 부하를 줄일 수 있는 요청 통합기를 설계한다. 요청 통합기가 수행하는 기능은 사용자의 요청에 대해 주어진 비용 함수(cost function)에 최적의 형태로 사용자의 요청을 사전 처리(preprocessing)하여 실제 저장 시스템에 전달하는 것이다. 요청 통합기에 의해 사전 처리된 결과를 본 논문에서는 통합 읽기 요청이라고 부른다. 다시 말하면, 통합 읽기 요청은 서로 다른 여러 사용자들의 요청에 대해 공통된 부분들을 통합하여 서비스함으로써 저장 서버의 부하를 줄이고자 한다.

저장 서버에서 요청 통합기를 사용할 경우 얻을 수 있는 장점은 다음과 같다.

· 본 연구는 아주대학교 정착 연구 지원사업에 의한 것임.
† 정 회 원 : 아주대학교 통신전문대학원 교수
ehwang@madang.ajou.ac.kr
논문접수 2001년 4월 12일
심사완료 2001년 10월 8일

1. 요청 통합기는 사용자의 요청과 어플리케이션의 요구 사항을 고려하여 수행되도록 설계되었기 때문에 어떠한 저장 시스템과도 연동하여 사용 가능하다. 즉, 요청 통합기는 디스크 서버나 테이프 서버, 혹은 CD-ROM 서버 등과 같은 다양한 저장 장치에 활용될 수 있다. 대부분의 디스크 서버는 사용자의 요청에 대해 특정한 목적 함수(objective function)에 따른 알고리즘을 수행한다. 요청 통합기의 목적은 저장 서버로 전달되는 읽기 요청의 수를 줄이는 데에 있다. 여기서, 가정하는 것은 여러 사용자가 동시에 같은 데이타를 필요로 할 때 하나의 버퍼에 읽혀진 데이타를 공유하여 사용하게 한다는 것이다. 이러한 버퍼 데이타 공유는 멀티미디어 서버의 성능 향상을 위해 흔히 사용되는 기법이다.
2. 요청 통합기는 사용될 비용(목적) 함수가 입력의 하나로 간주되기 때문에, 필요에 따라 임의의 비용 함수와 연동하여 검색 스케쥴을 만들 수 있다. 검색 스케줄은 여러 가지 기준에 의해 평가될 수 있으며, 사용자의 평균 대기 시간이나 시스템에 걸리는 부하, 서비스의 질(QoS), 또는 그것들의 혼합된 형태 등이 대표적인 예가 될 수 있다.

본 논문에서는 우선 요청 통합기에 주어지는 매개 변수에 대해 정의하고, 주어진 비용 함수에 대하여 최적(optimal)의 통합 읽기 요청이 되기 위한 조건을 정의한다. 통합 읽기 요청을 생성하기 위해 사용자 요청을 조작하는 데 사용될 수 있는 이동(shift), 교체(swap), 분할(split)이라는 세 가지 통합 연산을 정의하고, 주어진 읽기 요청에 대해서 일련의 통합 연산 적용을 통해 최적의 것을 찾을 수 있음을 보인다. 더불어, 우리는 일반적으로 주어진 비용 함수에 대해 최소의 비용을 가지는 통합 읽기 요청을 찾는 문제가 NP-complete임을 보인다. 요청 통합기의 구현에 사용될 수 있는 첫번째 알고리즘인 OptURR은 최적의 통합 읽기 요청을 찾아낼 수 있지만 위에서 언급한 문제로 인해 실제 사용하기에는 문제가 있다. 따라서, 우리는 heuristics을 이용하여 비록 최적은 아닐지라도 빠른 시간 내에 실행 가능한 통합 읽기 요청을 찾아낼 수 있는 FastURR과 GreedyURR이라는 두 개의 알고리즘을 제안하고 여러 가지 실험을 통하여 그들의 성능을 비교 분석하였다.

2. 문제 정의

이 장에서는 본 논문에서 시스템 개발에 사용된 모델에 대해 서술한다.

2.1 통합 읽기 요청

비디오 구성 (Video Organization) : 비디오 라이브러리 VL은 비디오 세그먼트의 유한 집합으로 정의되며 비디오 세그먼트 v는 다시 $bnum(v)$개의 연속적인 단위 블록으로 구성된다.

사용자 요청 (User Request) : 만약 b_i와 b_j가 비디오 세그먼트 v에 속한 블록이라 하면 b_i에서 b_j까지의 블록에 대한 읽기 요청을 (v, b_i, b_j)로 나타내며, 사용자 C가 요청한 비디오 세그멘트들을 $req(C)$로 표현한다. 따라서, 만약 $(v, b_i, b_j) \in req(C)$이면 사용자 C가 비디오 세그먼트 v의 블록 b_i에서 b_j까지를 요청한 것이 된다. 특히, 우리는 $req(C)$에 속하는 비디오 세그먼트들이 중복되거나 바로 인접하여 표시되지 않는다고 가정한다. 이러한 가정은 특별한 것이 아니며 실제 사용자의 요구를 이러한 형태로 바꾸는 것은 아주 쉬운 작업이다. 예를 들어 {(m1,10,20), (m1,18,24), (m1,24,27), (m2,27,29)}로 주어진 세그먼트 요청은 {(m1,10,27), (m2,27,29)}와 같이 축약된 형태로 나타낼 수 있다.

위에서 도입한 정의는 비단 비디오 데이타베이스의 검색뿐 만 아니라 일반 VoD(Video-on-Demand) 환경에서도 사용자의 요청을 구체화시키는 데 유용하게 사용될 수 있다. VoD 환경에서는 사용자의 요청이 일반적인 재생일 경우 $\{(v, b_i, b_j)\}$와 같이 표현된다. 반면, 비디오 데이타베이스에서의 검색일 경우, 사용자의 질의에 대한 결과가 여러 개의 비디오 세그먼트로 구성될 수 있다.

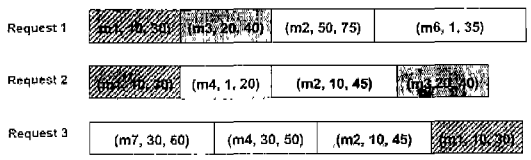

그림 1 사용자 읽기 요청의 예제

그림 1은 세 명의 사용자가 비디오 서버에 일련의 비디오 세그먼트들을 요청한 경우를 보여 준다. 그림에서 사용자 1의 요청은 다음과 같이 표시될 수 있다: {(m1, 10,30), (m3,20,40), (m1,50,75), (m6,1,35)}.

다중 읽기 요청 (Multi-Read Request): n개의 사용자 요청 $req(C_1), \ldots, req(C_n)$과 초당 S1≥1개의 데이타 블록을 읽을 수 있는 저장 서버를 가정할 경우, 시간 t에 대한 사용자 C_i의 다중 읽기 요청 $MR(C_i, t)$는 그

시간에 사용자 C_i에게 읽혀져야 하는 데이타 블록을 의미한다. $MR(C_i, t)$는 수용해야 하는 제약 조건이 있다. 예를 들면, 대역폭 제약 조건이나 사용자 대기 시간 등이 여기에 해당된다. 또한 다중 읽기 요청은 사용자 요청에 대한 결과들이 끊어짐이 없이 연속적으로 전달될 수 있게 구성되어야 한다. 이러한 제약 조건들은 나중에 다시 정의하겠다.

사용자 $C_1, \ldots, C_n$에 대한 다중 읽기 요청을 설명하기 위해 다음과 같은 기호를 정의한다.

$st(C_i) = \min\{t \mid MR(C_i, t) \neq 0\}$
$et(C_i) = \max\{t \mid MR(C_i, t) \neq 0\}$
$pt(C_i, m, b) = \min\{t \mid \exists (m, r, s) \in MR(C_i, t) r \leq b \leq s\}$.

여기서, $st(C_i)$는 C_i의 요청에 대해 처음으로 결과가 나타나기 시작하는 시간을 나타내며, C_i의 요청에 대한 대기 시간을 나타낸다. $et(C_i)$는 C_i의 요청에 대한 마지막 결과가 보이는 시간을 말한다. $pt(C_i, v, b)$는 클라이언트 C_i가 신청한 비디오 v의 블록 b가 저장 장치에서 읽혀지는 시간을 의미한다.

통합 읽기 요청 (Unified Read Request) : 통합 읽기 요청 U는 다음과 같은 조건을 만족시키는 다중 읽기 요청으로 정의할 수 있다:

1. (Gap-free Representation of Client Request) $st(C_i) \leq t \leq et(C_i)$인 모든 시간 동안에 $U(C_i, t) \neq \emptyset$이다. 이 조건은 사용자 C_i에 대해 일단 프리젠테이션이 시작되면 끝날 때까지 중간에 끊어짐이 없어야 함을 말한다. 비디오와 같은 연속 매체에 대해서 위와 같은 조건은 필수적이다.
2. (Completeness of Client Request) 임의의 사용자 C_i와 요청한 비디오 v에 대해서 $(v, r, s) \in req(C_i)$이고 $r \leq h \leq s$이면 $(v, r', s') \in U(C_i, t)$이면서 $r' \leq h \leq s'$를 만족시키는 시간 t가 $st(C_i)$와 $et(C_i)$사이에 존재한다. 이 조건은 사용자가 요청한 모든 블록은 통합 요청 읽기에 포함되어야 함을 나타낸다.
3. (Irredundancy) 임의의 사용자 C_i와 요청한 비디오 v에 대해 $(v, r, s) \in U(C_i, t)$이고 $(v, r', s') \in U(C_i, t')$이며 $t \neq t'$일 경우, $[r, s] \cap [r', s'] = \emptyset$이다. 이 조건은 동일한 비디오가 한 프리젠테이션에서 중복되어 나타나지 말아야 함을 나타낸다.
4. (Order Preservation within a Video) 임의의 사용자 C_i와 비디오 세그먼트 v에 대해 $(v, r, s) \in U(C_i, t)$이고 $(v, r', s') \in U(C_i, t')$이며 $t < t'$일 경우, $s < r'$이다. 이 조건은 한 비디오에 속한 블록의 순서가 전체 프리젠테이션에서도 유지되어야 함을 규정한다. 이는 통합 과정에서 생길 수 있는 한 비디오내의 블록들의 순서가 뒤바뀌는 오류를 방지한다.

예를 들어, 그림 1에 보여진 사용자 요청에 대한 가능한 통합 읽기 요청의 한 예는 그림 2와 같다. 그림에서 통합된 사용자 요청은 위에서 언급한 모든 조건들을 만족시키면서도 공통된 비디오 세그먼트들이 최대한 공유될 수 있도록 같은 시간대에 정렬되어 있음을 볼 수 있다. 결국 요청 통합기는 여러 사용자의 요청에서 중복되어 나타나는 비디오 세그먼트들을 공유가 가능하게 재정렬함으로써 저장 서버의 부하를 줄여주는 모듈이다.

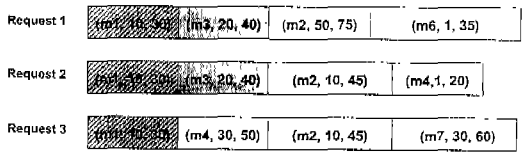

그림 2 사용자 요청 통합의 결과

2.2 통합 읽기 요청의 제약 조건

통합 읽기 요청 U에 대해, $U(t)$는 $\bigcup_{C_i} U(C_i, t)$를 의미한다. 즉, $U(t)$는 t라는 시간에 서버에서 읽혀지는 모든 데이타 블록의 집합이라고 할 수 있다. 통합 읽기 요청을 생성할 때 여러 가지 제약 조건들을 고려해야 하며, 대개 제약 조건은 사용자가 명시하거나 시스템에 의해 주어진다. 본 논문에서 고려한 제약 조건은 다음과 같다.

- 대역폭 제약 조건 (Bandwidth Constraint): 대역폭 제약 조건은 저장 서버가 단위 시간에 최대 개의 데이타 블록을 읽을 수 있다는 것을 나타낸다. 따라서, 대역폭 제약 조건은 임의의 시점 에서 모든 통합 읽기 요청 U에 필요한 대역폭의 합이 저장 서버의 최대 대역폭 보다는 작아야 함을 규정한다.

$$U(t) \leq S$$

- 소비율 제약 조건 (Consumption Rate Constraint): 사용자들은 서로 다른 소비율을 가질 수 있기 때문에 각 사용자가 필요로 하는 예상 소비율을 고려할 필요가 있다. 만약 사용자 C_i가 단위 시간에 W_i개의 블록을 소비한다고 할 때, 소비율 제약 조건은 다음과 같이 표현된다.

$(\forall t) st(C_i) \leq t \leq et(C_i)$
$\rightarrow card(\{(m, b) \mid (v, h, r) \in U(C_i, t) \& h \leq b \leq r\}) = w_i$

즉, 사용자 C_i의 요청이 수락되어 저장 서버로부터 읽혀지기 시작하면, 단위 시간당 w_i블록만큼 계속 읽혀져야 한다는 것을 나타낸다.

- 대기 제약 조건(Wait Constraint): 사용자 C_i에 대한 대기 제약 조건은 다음과 같이 표현될 수 있다.

$$st(C_i) \le t_s$$

이것은 사용자 C_i의 요청에 대해 최대 t_s라는 시간 전에 서비스가 시작되어야 함을 나타낸다. 이와 같은 대기 시간은 사용자가 직접 명시할 수도 있고 실제 응용에서 사용자에게 제시하는 조건의 하나가 될 수도 있다.

- 마감 제약 조건 (Deadline Constraint): 사용자 C_i의 마감 제한은 다음과 같이 표현된다.

$$et(C_i) \le t_e$$

이 조건은 사용자 C_i가 요구한 모든 데이타는 적어도 t_e라는 시간 내에 모두 저장 장치에서 읽혀져야 함을 말한다.

2.3 실행 가능 통합 읽기 요청

사용자 $C_1, \ldots, C_n$과 각 사용자에 대해 설정된 소비율이나 대기/마감 시간 등의 제약 조건 그리고 대역폭 제약 조건 등을 포함하는 제약 조건의 집합을 $CONS$라 하면, $CONS$에 명시된 모든 제약 조건을 만족하는 통합 읽기 요청 U를 실행 가능(feasible) 통합 읽기 요청이라고 한다.

사용자의 요청에 대해 실제 여러 개의 통합 읽기 요청이 존재할 수 있다. 그러한 경우, 이들 간의 정량적인 비교를 하기 위해서 우리는 비용 함수(cost function)를 사용한다. 비용 함수 cf는 입력으로 주어진 어떤 통합 읽기 요청에 대해 정량적인 값을 결과로 돌려주는 함수로 정의할 수 있으며, 본 논문에서는 다음과 같은 비용 함수를 고려하였다.

1. 실제 읽어야 하는 데이타 블록의 총계 (Total number of data blocks read) : 이 비용 함수는 주어진 통합 읽기 요청에 대해 저장 장치에서 실제 읽혀져야 하는 데이타 블록의 총수를 계산한다.

$$cf_{br}(U) = \sum\left(card\left(\bigcup_{i=1}^{n} \bigcup_{v \in VL} \bigcup_{(v,h,r) \in U(C_i,t)} \{(v,b) \mid (v,h,r) \in U(C_i,t) \& h \le b \le r\}\right)\right)$$

이 공식은 모든 사용자의 요청에 대해서 데이타 공유를 고려하여 주어진 시간 동안에 실제 읽혀져야 하는 데이타 블록의 수를 계산한다. 따라서, 만약 비디오 v의 블록 b_5를 세 명의 사용자가 같은 시간에 원할 경우 서버는 한번의 디스크 액세스로 처리할 수 있으므로 실제 읽히는 블록은 하나로 간주된다.

2. 우선 순위를 고려한 평균 대기 시간 (Prioritized average wait time) : 주어진 응용이 사용자에 대해 서로 다른 우선 순위를 부여한다면 그 우선 순위를 고려한 평균 대기 시간을 측정함으로써 통합 읽기 요청을 비교할 수도 있다. 높은 우선 순위를 가진 사용자는 그렇지 않은 사용자에 비해 덜 기다리게 스케줄 되어야 한다. 우선 순위를 고려한 비용 함수는 다음과 같이 정의된다.

$$cf_{pwt}(U) = \frac{\sum_{i=1}^{n}(p_i \times st(C_i))}{n}$$

3. 복합 방식 (Hybrid strategy) : 복합 방식은 하나 이상의 비용 함수를 동시에 고려한다. 예를 들어, 위의 두 비용 함수에 서로 다른 가중치로 고려한 새로운 비용 함수는 다음과 같이 정의된다. 여기서, α_1, α_2는 비용 함수에 대해 서로 다른 가중치를 반영하는 값이다.

$$cf_{hyb}(U) = \alpha_1 \times cf_{br}(U) + \alpha_2 \times cf_{pwt}(U)$$

다음은 일반적으로 실행 가능한 통합 읽기 요청을 찾는 문제의 복잡도는 NP-complete임을 보여준다.

Theorem 2.1 우선 K를 사용자가 요청한 모든 블록의 수라고 하자. 그러면 비용 함수 $cf_{br}(U) = K$에 대해 실행 가능 통합 읽기 요청을 찾는 문제는 NP-complete이다. 이것은 대역폭 제한 조건이나 마감 시간 제한 조건 등이 없는 상황에서 각 사용자 C_i가 오직 세 개의 블록만 요구하는 경우에도 해당된다.

증명: 이것은 3-coloring 그래프를 이용한 reduction으로 증명할 수 있다. G를 그래프라고 할 때, 일반적인 경우라면 G에 있는 모든 모서리는 삼각형 안에 있다고 가정하자. 만약 삼각형 안에 들어가지 않은 모서리(edge) (x, y)가 있을 경우에는 그 모서리에 대해서는 더미 노드 $v_{(x,y)}$를 추가하여 두 모서리 $(x, v_{(x,y)})$와 $(y, v_{(x,y)})$을 연결하여 삼각형을 구성함으로써 위의 가정을 실현할 수 있다. 여기서, 처음 그래프가 3-colorable이면 이렇게 만들어진 새로운 그래프도 3-colorable이다.

그래프의 각 정점(vertex)은 비디오의 한 블록에 해당한다. 여기서 우리는 그래프 G의 모든 삼각형 (u, v, w)을 사용자 $C_i = \{u, v, w\}$로 적용시켜서 통합 읽기 요청 문제의 개체를 생성했다. 사용자는 세 정점으로 나타내어지는 결과 블록에만 관심이 있다.

완벽한 스케줄이 존재한다면 그래프는 3-colorable이다. 그래프 G가 3-colorable이고 Y_1, Y_2, Y_3 가 색의 종류이며 $C_i = \{u, v, w\}$가 그래프 G의 삼각형 (u, v, w)

의 집합이라고 하면, u, v, w는 각각 다른 색을 가진다. C_i의 요소는 $a_j \in Y_j$일 때 $\{a_1, a_2, a_3\}$과 같이 정렬될 수 있다.

이제, $U(1)=Y_1$, $U(2)=Y_2$, $U(3)=Y_3$가 feasibility 문제의 해답이라고 하자. $i \neq j$인 경우에는 $Y_i \cap Y_j = \varnothing$이다. (u, v)가 그래프 G의 모서리이고 u, v는 집합 Y_i에서 나왔다고 하자. 그래프 G는 어떤 정점 w에 대해서 삼각형 (u, v, w)을 가져야만 한다. 따라서, feasibility 문제는 $S_j = \{u, v, w\}$라는 집합을 가지게 된다. Y_1, Y_2, Y_3는 feasibility 문제에 대한 답이기 때문에 u, v, w역시 다른 집합 Y_1, Y_2, Y_3에 나타나야 한다. 이것은 요소 u, v 중 적어도 하나는 다른 집합인 Y_i와 Y_j에 나타나야 함을 의미하는 데 이는 모순이다. 그렇기 때문에, 각 Y_i는 그래프 G에 독립적인 집합이고 Y_i는 그래프 G의 정점분할을 야기시킨다. 따라서, 그래프 G는 3-colorable이다.

최적(Optimal) 통합 읽기 요청: 사용자 요청에 대해 관련된 제약 조건들의 집합 *CONS*와 비용 함수 cf가 주어져 있다고 가정하자. 만약 통합 읽기 요청 U가 실행 가능하고 다른 어떠한 통합 읽기 요청 U'에 대해서도 $cf(U) < cf(U')$일 경우 U를 최적 통합 읽기 요청이라고 한다.

다음의 결과는 비록 비용 함수 cf_{br}만을 고려하고 모든 사용자의 요청이 오직 두개의 블록으로만 이루어져 있어도 최적의 통합 읽기 요청을 구하는 문제는 NP-complete임을 말해준다.

Theorem 2.2 각 사용자 C_i가 오직 두 개의 데이타 블록을 요청하는 경우에도 비용 함수 $cf_{br}(U)$에 대해 최적의 해를 찾는 문제는 NP-complete이다.

Proof: 우리는 주어진 그래프를 나누는 최소 노드 집합을 찾는 문제는 각 클라이언트의 cardinality가 2인 경우의 최적화 문제로 축약(reduced)될 수 있음을 보일 수 있다.

G가 주어진 그래프이고 최적화 문제의 예가 그래프 G에 있는 모든 모서리 (u, v)에 대한 집합 $\{u, v\}$으로 이루어져 있다고 하자. 그리고 $U(1)=Y_1$, $U(2)=Y_2$가 $cf_{br}(U)$를 최소화하는 최적화 문제에 대한 해라고 하자. 만약 $Y=Y_1 \cap Y_2$라 하면, $C=K+|Y|$이고 $|Y|$는 최소이다. G에서 만약 Y와 그것에 연결된 모서리를 제거하면 G는 양분(bipartite)된다. 이것은 남아있는 모든 모서리의 두 끝 점이 서로 다른 집합 $Y_1 \backslash Y$와 $Y_2 \backslash Y$에 속해 있기 때문이다.

마찬가지로, 만약 제거되면 두개의 색깔 그룹 Y_1과 Y_2로 그래프 G를 양분하는 정점의 집합을 Y라고 하면, $U(1)=Y_1 \cup Y$, $U(2)=Y_2 \cup Y$가 최적화 문제에 대한 실행 가능한 해이다.

3. 통합 읽기 요청 계산을 위한 알고리즘

앞에서 요청 통합기의 역할과 실행 가능 통합 읽기 요청 및 임의의 비용 함수에 대해 최적의 통합 읽기 요청을 정의하였다. 이 장에서는 실제 통합 읽기 요청을 구하는 데 사용되는 기본적인 통합 연산자를 정의하고 이것을 이용하여 최적의 통합 읽기 요청을 구하기 위한 알고리즘을 서술한다.

3.1 기본적인 통합 연산자

사용자 요청에 적용될 기본적인 통합 연산자는 앞에서 언급했듯이 임의의 다중 읽기 요청을 다른 형태의 다중 읽기 요청으로 변화시킨다. 여기서 주의할 점은 비록 입력으로 주어진 다중 읽기 요청이 실행 가능 통합 읽기 요청이라 할지라도 통합 연산의 결과는 아닐 수도 있다는 것이다. 이러한 점에서 어떤 통합 연산자가 실행 가능한 통합 읽기 요청에 적용됐을 때 그 결과도 역시 실행 가능 통합 읽기 요청이면 이 통합 연산자를 MR $-safe$ 하다고 한다.

이제 통합 읽기 요청에 적용될 세가지 주요 기본 통합 연산자를 정의하겠다.

- 이동(shift) : 사용자 C_i의 다중 읽기 요청 $MR(C_i, t)$가 (v, k, l)이면, $shift(C_i, t, \delta)$로 표현되는 이동 연산은 비디오 세그먼트의 읽기 요청을 δ만큼 지연시킨다. 따라서, $shift(C_i, t, \delta)$를 $MR(C_i, t)$에 적용한 결과인 MR'는 $MR'(C_i, t) = MR(C_i, t-\delta)$인 점을 제외하고는 MR과 같다. 그림 3은 다중 읽기 요청에 대한 이동 연산의 결과를 보여준다.

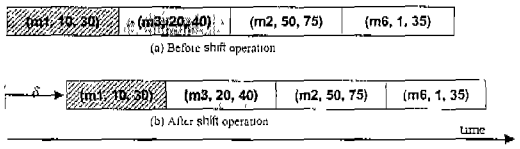

그림 3 이동 연산

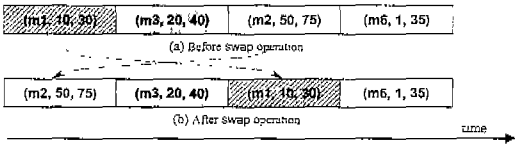

그림 4 교환 연산

- 교환(swap) : 교환 연산은 사용자 C_i에 전송되는 비디오 세그먼트들의 순서를 바꾸는 연산이다. 구체적으로 사용자 C_i에 대한 다중 읽기 요청 $MR(C_i, t)$이 (v, k, l)로 표시된다면 교환 연산 $swap(C_i, v, k, l, v', k', l')$에 의해 만들어지는 결과 다중 읽기 요청 MR'은 $MR'(C_i, t) = MR(C_i, t')$이고 $MR'(C_i, t') = MR(C_i, t)$인 점을 제외하고는 주어진 MR과 같다. 그림 4는 두 비디오 세그먼트 m1과 m2가 교환 연산에 의해 다중 요청에서 그 순서를 바꾼 것을 보여준다. 이것은 일반 비디오 데이타베이스에 대한 검색의 결과에 대해 사용자가 특정한 순서를 명시하지 않는다면 적절한 재정렬을 통해 세그먼트에 대한 공유의 가능성을 높여줄 수 있다.
- 분할(split) : 저장 서버가 사용자가 요청한 비디오 세그먼트 $(v, k, l) \in MR(C_i, t)$의 연속된 블록을 가져오기 시작하는 초기 시간을 t라고 가정한다. 분할 연산은 이 요청을 다시 두 개의 하위 요청으로 나눈다. 따라서, $k \le mid \le l$인 임의의 블록 mid에 대해 $split(C_i, v, k, l, mid)$은 그 결과로 세그먼트 v의 블록 k에서 l까지의 읽기 요청을 두 개의 요청, 즉 k부터 mid까지의 부분과 $(mid+1)$부터 l까지에 대한 요청으로 분할하는 점을 제외하고는 입력 MR과 같은 새로운 다중 읽기 요청을 반환한다. 그림 5는 세그먼트 m1에 대한 읽기 요청 (m1,50,75)을 블록 70을 기준으로 두 개의 하위 요청으로 분할하는 예를 보여준다.

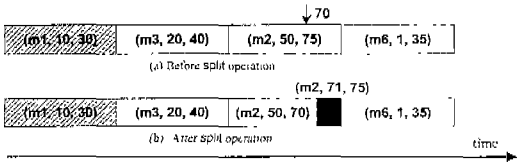

그림 5 분할 연산

앞에서 우리는 $MR-safe$에 대해서 언급하였다. 만약, 실행 가능 통합 읽기 요청인 MR에 대해, 통합 연산 op의 적용 결과인 $op(..., MR, ...)$도 실행 가능 통합 읽기 요청이면 op는 적용 안전하다고 한다. 통합 연산 중에서 이동과 교환 연산은 적용 안전하지 않을 수 있지만 분할 연산은 항상 적용 안전하다. 이동과 교환 연산은 비록 $MR-safe$는 아니지만, 다양한 형태의 사용자 요청에 적용할 수 있다. 이제 이 세 가지의 통합 연산을 이용하여 사용자의 다중 읽기 요청을 주어진 비용 함수를 최소화시키는 방향으로 통합하는 알고리즘에 대해서 알아보겠다.

3.2 OptURR 알고리즘

OptURR 알고리즘은 주어진 비용 함수에 대해서 사용자의 요구를 만족시키는 최적의 실행 가능 통합 읽기 요청을 찾을 수 있으며, 전체적인 구조는 그림 6에 나타나 있다. 여기서 evaluateURR 함수는 주어진 비용 함수에 대해서 입력된 통합 읽기 요청의 비용을 계산한다.

OptURR 알고리즘은 genInitURR 함수에 의해 생성된 초기 통합 읽기 요청에 대해 적용 가능한 통합 연산을 반복적으로 수행하면서 최적의 통합 읽기 요청을 찾아간다. 이 과정은 하나의 거대한 탐색 트리를 형성하게 되는 데, 탐색 트리에서 최상위 노드에는 초기 통합 읽기 요청이 위치하게 된다. 만약 N이 트리의 한 노드이고 op가 위에서 서술한 통합 연산 중에서 N에 적용 안전한 통합 연산자라면, $op(N)$은 노드 N의 자식 노드로 탐색 트리에 나타난다.

그림 7은 이러한 탐색 과정의 한 단면을 보여준다. 각 노드에서 적용 가능한 모든 통합 연산을 적용하면서 새로운 탐색 노드를 생성한다. 이러한 일련의 과정 속에서 최소 비용을 가지는 최적의 노드를 찾아 나가게 된다. 여기서 탐색이 어떠한 단조 증가(monotonicity)의 성질을 가지지 않으므로 최적의 노드를 찾기 위해서는 트리 전체를 탐색해야 한다.

```
REQUESTS : set of client requests;
COST : cost function for schedules;
CONSTRAINTS : set of constraints for each request;
OPSET : {shift, swap, split};

OptURR ( REQUESTS, COST, CONSTRAINTS )
{
    initURR = genInitURR ( REQUESTS, CONSTRAINTS )
    optURR = initURR;
    optCost = evaluateURR ( initURR, COST );
    candidateList = append ( emptyList, initURR );

    while ( candidateList is not empty ) do {
        caddidate = choose a schedule from candidateList;
        if ( there exists op ∈ candidate.opList ) do {
            candidate.opList = remove ( candidate.opList, op ) ;
            newURR = applyOpToURR ( candidate, op)
            candidateList = append ( candidateList, newURR)
        }
        else {
            candidateList = remove ( candidateList, candidate ) ;
            if ( newURR is valid with respect to CONSTRAINTS ) do {
                candidate.cost = evaluateURR ( candidate, COST ) ;
                if ( candidate.cost < optCost ) do {
                    optURR = candidate ;
                    optCost = candidate.cost ;
                }
            }
        }
    }
    return ( optURR );
}
```

그림 6 OptURR 알고리즘

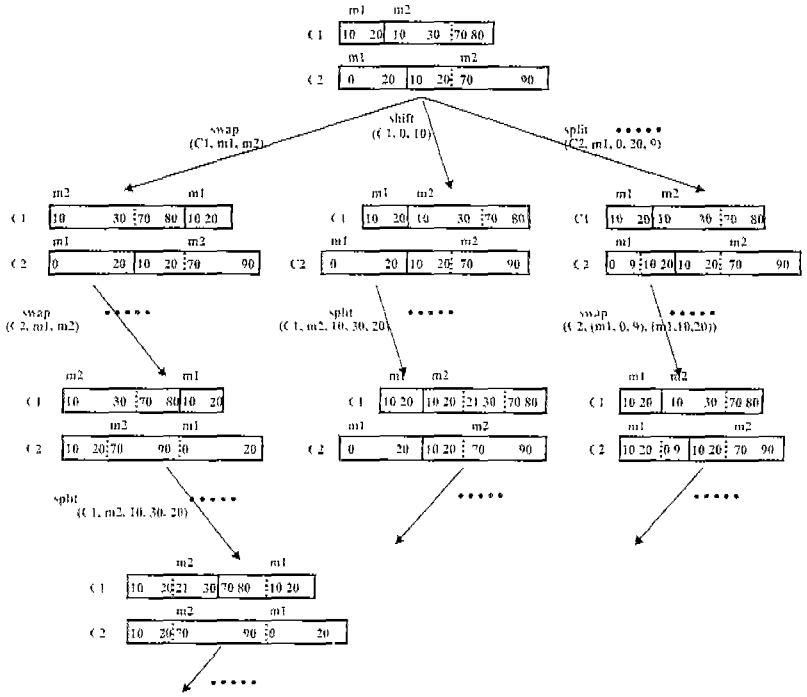

그림 7 OptURR에 의한 탐색 트리의 예

OptURR 알고리즘이 만드는 탐색 공간은 잠정적으로 매우 광범위할 수 있으며 그 과정에서 실제 많은 불필요한 노드에 대한 탐색이 이루어짐을 알 수 있다. 이것은 결국 최적의 통합 읽기 요청을 찾는 문제가 NP-Complete인 것과 밀접한 관계가 있다. 따라서 최적성과 탐색 시간의 효율성을 타협하여 빠른 시간 내에 적은 비용의 통합 읽기 요청을 찾을 수 있는 알고리즘이 필요하다.

3.3 GreedyURR 알고리즘

GreedyURR 알고리즘은 OptURR 알고리즘이 생성하는 탐색 트리에서 불필요한 노드의 생성을 줄이기 위한 조처를 취한다. 구체적으로, 탐색 트리의 각 노드 N에서 적용 안전한 통합 연산에 의해 생성 가능한 노드 중에서 최소의 비용을 가지는 노드만 N의 자식 노드로 만든다. 따라서, 탐색 트리를 만들어 가는 각 단계에서 GreedyURR 알고리즘은 지역적인(local) 최적의 노드를 제외한 다른 모든 노드들은 더 이상 고려하지 않는다. 대략적인 GreedyURR 알고리즘이 그림 8에 보이고 있다. 여기서, findBestURR 함수는 입력으로 주어진 후보 노드들의 리스트(후보 통합 읽기 요청들로 구성된 Candidate List)에서 최적의 후보를 찾아주는 기능을 한다.

결과적으로, GreedyURR 알고리즘은 OptURR 알고리즘과 연관된 트리에서 최소의 비용을 가지는 노드를 따라 오직 하나의 경로만 만들기 때문에 탐색에 필요한 노력을 상당히 줄일 수 있다. 물론 최종 결과가 최적이 아닐 수도 있다. 그림 9는 OptURR 알고리즘에 의해 그림

```
FindBestURR (candidateList, bestURR, bestCost)
{
    while ( candidateList ) {
        candidate = choose a schedule from candidateList ;
        if ( there exists op ∈ candidate.opList ) {
            candidate.opList = remove ( candidate.opList, op ) ;
            newURR = applyOpToURR ( candidate, op)
            if ( newURR is valid with respect to CONSTARAINTS ) {
                newURR.cost = evaluateURR ( newURR, COST ) ;
                if ( newURR.cost < bestCost)
                    candidateList = append ( candidateList, newURR)
            }
        }
        else {
            candidateList = remove ( candidateList, candidate ) ;
            if ( candidate cost < bestCost ) {
                bestURR = candidate ;
                bestCost = candidate.cost ;
            }
        }
    }
    return ( bestURR );
}

GreedyURR ( REQUESTS, COST, CONSTRAINTS )
{
    initURR = genInitURR ( REQUESTS , CONSTRAINTS ) ,
    bestURR = initURR ;
    bestCost = bestURR.cost = evaluateURR ( initURR , COST ) ,
    candidateList = append ( emptyList , initURR ) ;
    bestURR = findBestURR ( candidateList , bestURR , bestCost);
    return ( bestURR );
}
```

그림 8 GreedyURR 알고리즘

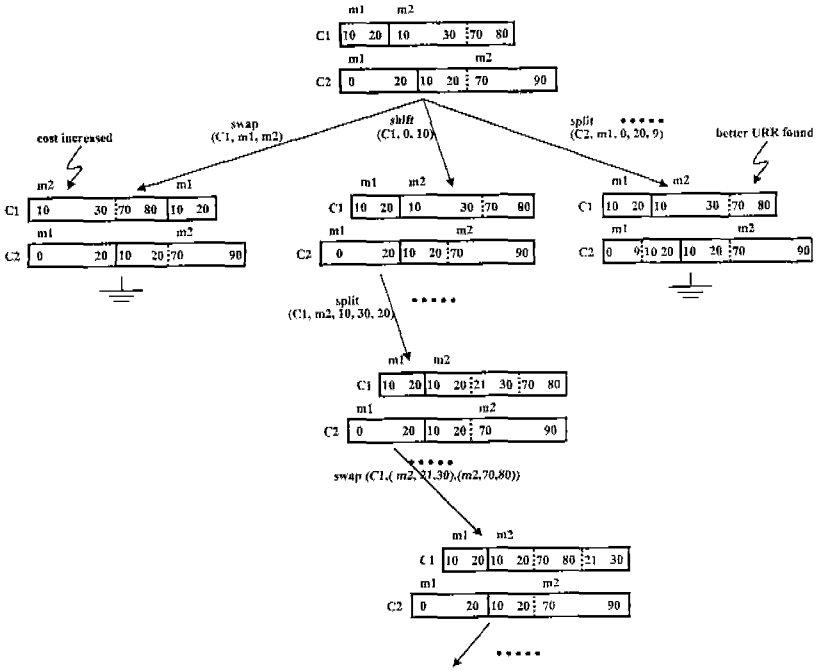

그림 9 GreedyURR에 의한 탐색 트리의 예

7에서 나타난 경우에 대해 GreedyURR 알고리즘의 수행 상태를 보여준다. 결과적으로, OptURR 알고리즘에 비해 만들어지는 탐색 범위가 훨씬 줄었음을 알 수 있다.

3.4 FastURR 알고리즘

앞의 두 알고리즘과는 달리, FastURR 알고리즘은 반복적으로(iteratively) 작동한다. 이 알고리즘은 정렬된 각 사용자의 요청에 대해 차례로 최적의 통합 읽기 요청을 찾아가는 방식이다. 우선 정렬에 의해 처음에 위치한 사용자의 요청에 대해서만 통합 읽기 요청을 생성한다. 그 다음부터는 이제까지 만들어진 통합 읽기 요청에 대해 현재 고려중인 특정 사용자의 요청이 가장 적은 비용으로 수행될 수 있는 통합 읽기 요청을 찾아가면서 궁극적으로 전체 사용자에 대한 통합 읽기 요청을 완성한다. FastURR 알고리즘은 그림 10과 같다.

FastURR 알고리즘도 탐색 공간을 줄이기 위해 통합 읽기 요청의 최적성을 희생한다. 비록 첫번째 요청에 대해 얻어진 초기 통합 읽기 요청은 최적일지라도, 점차적으로 다른 사용자 요청을 고려하면서 최종적으로 만들어진 전체 사용자에 대한 통합 읽기 요청은 최적이 아닐 수 있다. 하지만 최종의 통합 읽기 요청을 찾기까지 걸리는 시간이 다른 알고리즘에 비해 월등히 짧다. 이러한 특성을 좀 더 정량적으로 알아보기 위해 4장에서는 실험을 통해 이들 알고리즘의 성능을 분석해 본다.

```
FastURR ( REQUESTS, COST, CONSTRAINTS )
    {
        currURR =   0;
        while ( REQUESTS ) {
            ci_Request = remove one request from REQUESTS ,
            bestURR = genInitURR ( currURR, ci_Request ) ,
            bestCost = bestURR.cost = evaluateURR ( bestURR, COST ) ,
            candidateList = append (emptyList, bestURR ) ;
            currURR = findBestURR ( candidateList, bestURR , bestCost ) ,
        }
        return ( currURR) ,
    }
```

그림 10 FastURR 알고리즘

4. 실 험

본 장에서는 앞에서 제안한 알고리즘의 성능을 비교하고 분석하기 위해 수행한 실험과 그 결과에 대하여 기술한다. 성능 평가는 Sun Sparc Ultra1 상에서 수행되었으며 운영체제로는 Sun OS version 5.6이 사용되었다. 우선, 사용자 요청의 집합 $CL=\{C_1,...,C_n\}$에 대해 비디오 블록이 전체 사용자 요청에서 얼마나 중복되어 있는 지를 정량적으로 평가하기 위하여 다음과 같은 요

청 중복 비율 (Request Repetition Ratio)을 정의한다.

$$\frac{card(\{m,b) \mid (\exists C_i \in CL)(m,k,l) \in req(C_i) \& k \le b \le l\})}{\sum_{\forall i}(card(\{(m,b) \mid (m,k,l) \in req(C_i) \& k \le b \le l\}))}$$

예를 들어, 사용자 요청에 포함된 각 블록에 대해서 오직 한 사용자만이 요청을 하였다면 블록의 중복 비율은 1이 된다. 요청 중복 비율이 증가할수록 많은 사용자가 같은 데이타 블록을 원하기 때문에 요청 통합으로 얻을 수 있는 이점이 증가할 것임을 쉽게 추측할 수 있다. 실험에서는 0.5의 차이를 주며 1.5에서 3.0까지 다양한 요청 중복 비율을 고려하였다.

저장 시스템에는 모두 15명의 사용자가 동시에 접근하며, 각 사용자의 요청은 평균적으로 6개의 비디오 세그먼트로 구성되어 있고 각 세그먼트는 다시 50개의 단위 블록으로 구성되어 있다고 가정한다. 사용자의 비디오 요청은 실제 비디오 대여 가게[20]에서 관측된 데이타를 사용했으며, 실제 이 데이타는 Zipf 분포에 근접한 형태를 보여준다. 또한 실험에서는 같은 조건 하에서 적용할 통합 연산자의 조합을 달리하면서 그러한 조합이 결과에 미치는 영향을 측정하였다. 이것은 응용에 따라 적용할 통합 연산자를 제한할 수 있기 때문이다. 예를 들면, 사용자의 대기 시간을 없애기 위해서는 이동 연산을 제한하면 된다.

그림 11에서 그림 16은 사용자의 대기 시간에 아무 제약이 없는 가정 하에서 실시한 실험의 결과를 보여준다. 구체적으로, 그림 11과 그림 12는 FastURR과 GreedyURR 알고리즘에 의해 생성된 통합 읽기 요청에 대해 실제 저장 장치에서 읽혀져야 하는 데이타 블록의 양을 보여준다. 전체 사용자의 요청에 대해 적용된 통합 연산의 조합과 그에 따른 데이타 블록의 수가 그림에서 통합 연산의 조합에 따라 표시되어 있다. 그림에서 우리는 더 다양한 통합 연산이 사용될수록 더 효과적인 사용자 요청 통합이 이루어짐을 볼 수 있다. 또한 중복 비율이 증가함에 따라 읽혀져야 하는 데이타 블록의 수가 감소한다. 이것은 비디오 세그먼트가 사용자 요청에서 더 자주 중복되어 나타날수록 통합에 의해 더 많은 데이타 공유가 이루어질 수 있기 때문이다. 그리고 이것은 적은 수의 특정 데이타에 많은 요청이 집중되는 멀티미디어 응용의 특성을 고려할 때 읽기 요청 통합에 의해 얻을 수 있는 이점이 크다는 것을 알 수 있다. 사용자 요청을 통합함으로 얻어지는 이점을 정량적으로 비교하기 위해서 다음과 같은 데이타 공유 비율을 정의하였다.

$$Data\ Sharing\ Rate(\%) = \frac{(Data\ Blocks\ In\ Requests - Data\ Blocks\ Retrieved)}{Data\ Blocks\ In\ Requests} \times 100$$

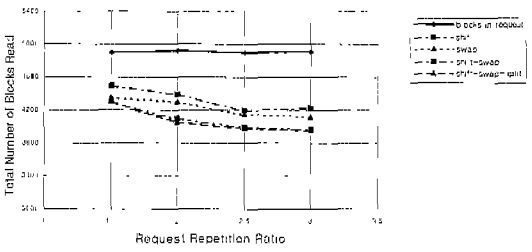

그림 11 GreedyURR에 의해 읽힌 데이타 블록의 합

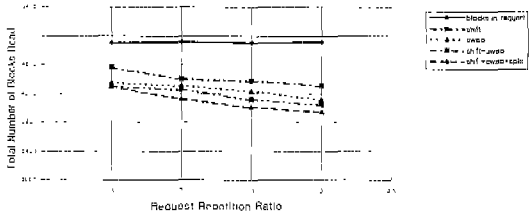

그림 12 FastURR에 의해 읽힌 데이타 블록의 합

표 1은 그림 11과 12에 나타난 결과에 대한 데이타 공유 비율을 보여준다. 표에서 GreedyURR과 FastURR 알고리즘 모두 데이타 공유 측면에서는 많은 이점을 보여주고 있다. 좀 더 자세히 살펴 보면 GreedyURR 알고리즘이 FastURR 알고리즘보다 약간 더 나은 결과를 보여주는 데, 그 이유는 GreedyURR 알고리즘이 그림 9에서 보았듯이 일반적으로 FastURR 알고리즘보다 더 광범위한 탐색 트리를 형성하여 보다 효율적인 통합 읽기 요청을 찾을 가능성이 크기 때문이다. 결국 이러한 데이타 공유를 통해서 한번의 디스크 액세스로 동시에 여러 사용자를 만족시킬 수 있으므로 더 높은 데이타 공유 비율로 나타난다.

표 1 중복 비율에 따른 데이타 공유 비율

Algorithm	Operations	Repetition Ratio(R)			
		R=1.5	R=2.0	R=2.5	R=3.0
GreedyURR	shift	10.8	12.6	14.9	15.5
	shift/swap	12.4	16.2	17.9	19.0
	shift/swap/split	12.6	17.3	18.4	19.4
FastURR	shift	9.6	11.6	13.8	15.3
	shift/swap	11.8	12.9	15.3	16.0
	shift/swap/split	12.0	15.2	16.2	17.2

그림 13과 14는 사용자의 초기 대기 시간을 초 단위로 보여준다. 그림에서 GreedyURR 알고리즘은 Fast

URR 알고리즘보다 약간 더 짧은 대기 시간을 보여주며 또한, 더 다양한 통합 연산을 적용할 때 사용자 대기 시간이 더 짧아지는 것을 볼 수 있다. 앞에서와 마찬가지로 GreedyURR 알고리즘이 좀 더 광범위한 탐색 트리를 형성함으로써 더 나은 통합 읽기 요청에 이를 가능성이 높기 때문에 더 짧은 사용자 대기 시간을 보여준다. 또한 중복 비율이 증가함에 따라 사용자 요청에 나타나는 서로 다른 비디오 세그먼트의 수는 줄어들고(사용자 요청에 포함된 전체 비디오 세그먼트의 수는 일정하게 했으므로) 자원 공유의 가능성이 증가하여 사용자 대기 시간의 감소로 나타난다.

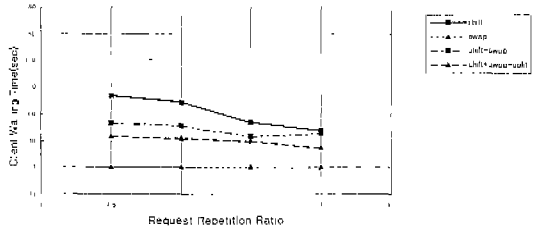

그림 13 GreedyURR 하에서 초기 대기 시간

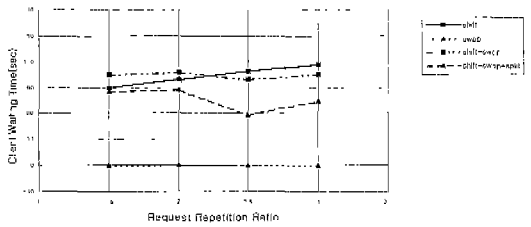

그림 14 FastURR 하에서 초기 대기 시간

그림 15와 16은 GreedyURR 알고리즘과 FastURR 알고리즘이 최종 통합 읽기 요청을 계산하기까지 걸린 실행 시간을 보여준다. 그림에서 FastURR 알고리즘은 GreedyURR 알고리즘에 비해 계산하는 데 걸린 시간이 훨씬 작다는 것을 알 수 있다. 이것은 앞에서 살펴본 내용과 밀접한 관계가 있는 데 GreedyURR 알고리즘이 훨씬 더 광범위한 탐색 트리를 생성하기 때문에 실제 읽어야 되는 블록의 수나 초기 사용자 대기 시간과 같은 측면에서는 FastURR 알고리즘 보다 나은 결과를 보이지만 통합 읽기 요청을 계산하는 데 걸리는 시간은 반비례해서 더 길어진다. 또한, 두 알고리즘 모두 중복 비율이 증가함에 따라 계산 시간이 늘어나는 것을 볼 수 있는 데 이것은 그만큼 중복 비율이 증가할수록 통합 가능한 경우의 수가 증가하여 전체적인 탐색 공간이 늘어나는 데 기인한다.

요약하면, FastURR 알고리즘은 스케줄링 시간으로 보면 GreedyURR 알고리즘에 비해 명백하게 더 나은 성능을 보인 반면 사용자의 초기 대기 시간이나 저장 장치에서 실제 읽어야 하는 블록의 수에서 보면 근소한 차이로 GreedyURR 알고리즘이 더 나은 성능을 보였다.

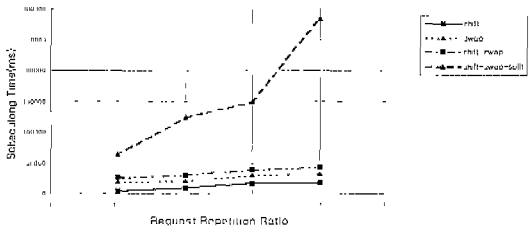

그림 15 GreedyURR의 스케줄링 시간

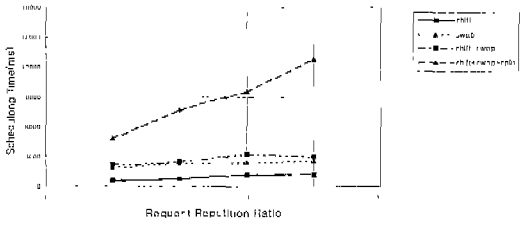

그림 16 FastURR의 스케줄링 시간

5. 관련 연구와 결론

이제까지 Video-on-Demand(VoD) 서버의 구현과 관련하여 많은 논문들이 발표되었으며[1, 2, 3, 4, 5, 6], 효과적인 디스크 스케줄링 방법이나 디스크 스트라이핑, 자원 관리, admission 제어와 같은 다양한 문제들을 다루고 있다. Ng와 Yang은 버퍼 공유와 prefetching을 이용한 버퍼 관리 기법을 제안하고 분석하였다[5]. 또한 빠른 응답을 위해 비디오 객체를 캐싱하는 기법이 [7]에 서술되어 있다. 우리는 실험에서 버퍼 이용을 개선시키기 위해 이 논문에서 제안한 *interval* 캐싱 기법을 이용하였다. 저장 시스템의 부하를 줄이기 위한 또 하나의 방법으로 [8]에서는 VoD 시스템을 위한 비디오 스트림의 *piggybacking* 기법에 대해서 언급하고 있다. 그러나 이 기법은 비디오가 상당히 길 때 효과적이며 사용자 요청에 의해 여러 개의 비디오 세그먼트가 되돌려지는 경우를 고려하지 않았다. [9]에서는 News-on-Demand

시스템에서 객체 검색을 최적화하는 기법을 제안하고 있다. 그러나 그들은 같은 크기의 비디오 세그먼트만을 고려하였으며 스케줄이 확정된 사용자의 요청에 대해서는 재정렬을 허용하지 않은 경우만을 고려하였다.

[10]에서는 중복되는 객체에 대해 객체를 통째로 prefetching하는 대신에 중복되는 부분만을 부분 prefetching하면서 복합 객체(Composite object)를 검색하는 기법을 제안했다. 그래서 객체가 가능한 한 많이 중복되게 스케줄 되었다면, 이 기법은 더 좋은 성능을 보여줄 것이다. 본 논문에서는 사용자가 여러 개의 비디오 클립을 보고자 하는 상황에서 다양한 비용 함수에 대해 최적의 데이타 검색을 지원할 수 있는 방법에 초점을 맞췄다. 또한 비디오 세그먼트들이 다른 크기를 가지는 것을 허용하고 있으며 사용자를 위한 비디오 세그먼트들은 이미 스케줄 되었음에도 나중에 다른 요청에 의해 그 순서가 재정렬될 수도 있다. 이러한 우리의 기법은 비디오 클립이 일치되는 경우뿐 아니라 겹치는(overlap) 경우도 허용될 수 있도록 쉽게 확장할 수 있다.

[11]에서는 하드 실시간 환경에서 다중 프로그래밍을 위한 선점(preemption)과 우선 순위에 기반한 스케줄링 알고리즘을 제안하였다. 반면, 복합 미디어(mixed-media) 작업 요청을 위한 스케줄링 알고리즘이 [12, 13]에 제안되어 있다. 특히, 비디오와 오디오 같은 연속 미디어 객체의 검색을 주로 다루는 서버에서 텍스트와 이미지 같은 불연속 미디어 검색을 효과적으로 다룰 수 있는 기법을 제안하고 있다. 그들은 불연속 미디어 객체의 검색은 연속 미디어 객체의 검색과는 독립적인 것으로 간주하였다. 또한 [12, 13]에서 불연속적 미디어 객체의 검색 요청에 있어 그들의 검색 시간에 어떠한 제한도 두고 있지 않다. 본 논문에서는 비디오 세그먼트의 순서가 유동적일 때 비디오 세그먼트 검색 기법을 제안하고 있다. [14]에서는 버퍼 관리와 멀티미디어 프리젠테이션을 위한 admission 제어를 위한 알고리즘을 제안하고 있다. 특히, 디스크 서버와 네트워크 자원에 대한 추상적 모델을 사용하며, 서버 버퍼 관리를 기반하여 admission 제어에 초점을 맞추었다. [15]은 저장 매체의 효율을 높이는 방법으로 zoned-disks에 기반한 데이타 설계 방법을 제안하고 있다. 본 논문에서는 연속 미디어 서버의 저장 관점이 아닌 응용에 내재되어 있는 유연성(flexibility)을 이용하여 연속 미디어 서버로부터의 검색 스케줄링에 대해서 살펴 보았다.

결론적으로, 본 논문에서는 연속 미디어에 대한 사용자의 읽기 요청에 대해 임의의 비용 함수에 최적인 통합 읽기 요청을 생성하는 요청 통합기 모듈을 소개했다. 비용 함수로는 실제 환경에서 고려될 수 있는 다양한 요소가 가능하다. 예를 들면, 실제 디스크에서 읽혀야 할 데이타 블록의 수나 평균 사용자 대기 시간, 버퍼 이용도 등이 일반적인 비용 함수의 예가 될 수 있다. 또한 주어진 비용 함수에 최적인 통합 읽기 요청을 찾는 문제가 NP-complete임을 증명하였으며 비록 부분적으로 최적화된 통합 읽기 요청이지만 빠른 시간 내에 계산할 수 있는 FastURR과 GreedyURR 이라는 두 알고리즘을 제안하고 실험을 통해 그들의 성능을 비교하였다.

참 고 문 헌

[1] S. Berson, S. Ghandeharizadeh, R. Muntz, and X. Ju, *"Staggered Striping in Multimedia Information Systems,"* Proc. of ACM SIGMOD Conference, pp. 79-90, 1994.

[2] M. Budhikot, G. Parulkar, and J.R. Cox Jr, *"Design of a Large Scale Video Server,"* Journal of Computer Networks and ISDN Systems, pp. 504-517, Dec. 1994.

[3] H.J Chen, A. Krishnamurthy, T.D.C. Little, and D. Venkatesh, *"A Scalable Video-On-Demand Service for the Provision of VCR-like Functions,"* Proc. of Int'l. Conference on Multimedia Computing and Systems, pp. 65-72, 1995.

[4] M.S. Chen, D.D. Kandlur, and P.S. Yu, *"Support for Fully Interactive Playout in a Disk-Array-Based Video Server,"* Proc. of ACM Multimedia, pp. 391-398, 1994.

[5] R.T. Ng and J. Yang, *"Maximizing Buffer and Disk Utilizations for News On-Demand,"* Proc. of the 20'th VLDB Conference, Chile, 1994.

[6] P.V. Rangan and H. Vin, *"Designing File Systems for Digital Video and Audio,"* Proc. of ACM Symposium on Operating Systems Principles, pp. 69-79, 1991.

[7] A. Dan and D. Sitaram, *"A Generalized Interval Caching Policy for Mixed Interactive and Long Video Workloads,"* Multimedia Computing and Networking, San Jose, Jan. 1996.

[8] L. Golubchik, J. Lui, and R. Muntz, *"Reducing I/O Demand in Video-On-Demand Storage Servers,"* Proc. of ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, pp. 25-36, 1995.

[9] R.T. Ng and Paul Shum, *"Optimal Clip Ordering for News On-Demand Queries,"* Int'l Workshop on Multimedia Information Systems, New Work, 1996.

[10] S. Chaudhuri, S. Ghandeharizadeh, and C. Shahabi,

"*Avoiding Retrieval Contention for Composite Multimedia Object,*" Proc. of the VLDB Conference, Zurich, Switzerland, 1995.

[11] C.L. Liu and J.W. Layland, "*Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment,*" Journal of the ACM, Vol. 20, No. 1, pp. 46-61, Jan. 1973.

[12] G. Nerjes, P. Muth, M. Paterakis, Y, Rompogiannakis, P. Triantafillou, and G. Weikum, "*Scheduling Strategies for the Mixed Workloads in Multimedia Information Servers,*" Int'l Workshop on Research Issues in Data Engineering, Orlando, Florida, Feb. 1998.

[13] Y. Rompogiannakis, G. Nerjes, P. Muth, M. Paterakis, P. Triantafillou, and G. Weikum, "*Disk Scheduling for Mixed-Media Workloads in a Multimedia Server,*" Proc. of ACM Multimedia, pp. 297-302, 1988.

[14] N.H Balkir and G. Ozsoyoglu, "*Delivering Presentations from Multimedia Servers,*" VLDB journal, Dec. 1998.

[15] Y.M. Huang and S.L. Tsao, "*An Efficient Disk Layout Scheme for Multi-Disks Continuous Media Servers,*" Multimedia Tools and Applications Journal, Kluwer Academic Publishers, Vol. 9, No. 2, pp. 147-166, Sep. 1999.

[16] A.L. Drapeau, D.A. Patterson, and R.H. Katz, "*Toward Workload Characterization of Video Server and Digital Library Application,*" Proc. Of ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, Nashville, May 1994.

[17] M. Garey and D. Johnson, "*Computer and Intractability: A Guide to the Theory of NP-Completeness,*" Freeman and Company, NY, 1979.

[18] E. Hwang and B. Prabhakaran, "*Merging Retrieval Requests for Multimedia Storage Server,*" Int'l Workshop on Intelligent Multimedia Computing and Networking, New Jersey, Feb. 2000.

[19] J.Y. Leung and M.L. Merrill, "*A Note on Preemptive Scheduling of Periodic Real-Time Tasks,*" Information Processing Letters, Vol. 11, No. 3, pp. 115-118, Nov. 1980.

[20] "*Video Store Magazine,*" Dec. 13, 1992.

[21] E. Hwang, V. S. Subrahmanian, and B. Prabhakaran, "*Distributed Video Presentation,*" Int'l Conference on Data Engineering, pp. 268-275, Florida. Feb. 1998.

황 인 준

1988년 서울대학교 컴퓨터공학과(학사). 1990년 서울대학교 컴퓨터공학과(석사). 1998년 Univ. of Maryland at College Park 전산학과(박사). 1998년 6월 ~ 1998년 8월 Hughes Research Lab. 연구교수. 1998년 8월 ~ 1999년 8월 Bowie State Univ., Assistant Professor. 1999년 9월 ~ 현재 아주대학교 정보통신전문대학원 조교수. 관심분야는 데이타베이스, 멀티미디어 시스템, 정보 통합, 전자 상거래, XML 응용