모바일 환경에서의 타임스템프 트리 기반 캐시 무효화 보고 구조

이 학 주° 정 성 원
서 강 대 학 교 컴 퓨 터 학 과
or4nge@sogang.ac.kr° jungsung@ccs.sogang.ac.kr

A Timestamp Tree-based Cache Invalidation Report Scheme in Mobile Environments

Hakjoo Lee° Sungwon Jung
Dept. of Computer Science, Sogang University

요 약

이동 컴퓨팅(Mobile Computing) 환경에서 빈번한 접속 단절은 클라이언트의 캐시 일관성(Consistency) 문제로 직결된다. 본 논문에서는 이를 해결하기 위한 무효화 보고(Invalidation Report, IR)중 하나를 제시한다. 기존의 IR방식은 대량의 데이터와 빈번한 데이터의 갱신(Update)시 IR 길이의 증가폭이 상당히 크다. 이는 타임스탬프가 IR에서 차지하는 비중이 큼으로써 오는 현상이다. 또한 기존방식은 캐시 레벨의 선택적 청취(Selective tuning)를 지원하지 못하는 문제점을 가지고 있다. 이러한 점을 극복하고자 B+Tree를 기반으로 시간 값을 구성하고 이에 따른 데이터 아이디를 재구성하는 IR 방식을 제안한다. 이를 활용하면 각 클라이언트가 자신의 단절(Disconnect)된 시간에 맞는 갱신 정보를 확인할 수 있게 된다. 이러한 방법은 기존 IR의 단점을 보완하며 선택적 청취를 유도할 수 있는 효과적인 방법이다.

1. 서론

이동 컴퓨팅(Mobile Computing) 환경은 대역폭의 비대칭과 클라이언트가 가진 자원의 협소함이라는 중요한 특징을 가지고 있다. 이동 컴퓨팅 환경에서 클라이언트가 어떠한 데이터를 접근하고자할 때마다 서버에 이러한 요구를 전달하는 것은 심각한 문제가 될 수 있다. 이러한 이동 컴퓨팅 환경에서는 데이터를 캐시(Cache)함으로써 이러한 문제점들을 보안하고 있다. 즉 클라이언트의 요구를 매번 서버 측에 전달하는 것이 아니라, 사용된 데이터를 저장함으로써 데이터 요청의 횟수를 줄이고자 한다. 이는 업스트림(upsteam) 대역폭의 절약과 그때 소모되는 전원의 절약을 유도할 수 있다. 그러나 이동 컴퓨팅 환경에서의 클라이언트들은 자의적인 선택과 타의적인 환경으로 인해 잦은 단절(Disconnection)이 발생한다. 따라서 캐시에 저장하고 있는 데이터의 일관성(Consistency)을 보장 할 수 없다. 이러한 문제를 해결하고자, 서버는 무효화 보고(Invalidation Report, IR)[1]를 클라이언트에게 주기적으로 전송한다.

이러한 IR을 보고하는 다양한 방법들이 연구 되어 왔다. 예를 들면 Bit-Sequence (BS), Broadcasting Timestamps (BT) [1]와 Dual-Report Cache Invalidation (DRCI) [2]와 같은 IR기법들이 있다. 이러한 방법은 공통적으로 데이터의 양이 증가함에 따라 IR의 길이도 급격히 증가함으로써 다량의 정보를 표현하기 적합하지 않다. 이러한 특징은 IR의 구성에서 타임스템프(TimeStamp)가 차지하는 비중이 큼으로써 오는 현상이다.

따라서 본 논문에서는 이러한 문제점을 극복하고 선택적 청취(Selective listen, Selective tuning)를 가능하게 함으로써 효율성을 높이는 방법을 제시한다. 이 방법은 타임스템프 Tree 구조를 이용함으로써 가능하게 되는데 이러한 구조를 활용하기 위해서는 전송해야할 데이터의 재구성(Information Reconstruction, IC)과 타임스템프 트리(TimeStamp Tree, TT)의 구성이 요구된다.

본 논문은 다음과 같은 구성으로 새로운 IR의 구성(Scheme)을 설명한다. 2장에서는 기존 연구에서 가지고 있는 문제점들을 생각해보고, 3장에서는 이러한 문제점을 해결하고자 제시하는 IR의 구성을 설명한다. 4장에서는 제시한 방법의 분석과 기존의 DRCI와의 비교를 통해 성능을 평가해 본다. 또한 5장에서는 현 연구의 결론과 앞으로의 방향을 다시 한번 짚어 본다.

2. 관련 연구

클라이언트의 캐시의 일관성(Consistency)을 유지하는 방법은 크게 stateful방법과 stateless방법으로 구분할 수 있다. stateful방법은 서버가 클라이언트의 정확한 상태를 서버가 알고 거기에 따른 캐시의 일관성을 유지하는 방법으로 Andrew File System[3]과 Asynchronous Scheme(AS)[4]가 있다. 반면 stateless방법으로는 타임스템프를 이용한 BT, 데이터를 그룹으로 나누어 표현하는 Group-based Cache Invalidation(CGI)과 이를 기반으로 한 DRCI가 있다. 이에 반해 데이터를 계층화된 Bit-Sequence로 표현하는 BS와 이를 기반으로 하여 다량의 데이터를 효율적으로 표현하고자 유도하는 방식인 Multidimensional Bit-Sequence(MDBS)[5], 접근빈도가 높은 데이터의 표현을 강조하는 Multilevel Bit-Sequence(MLBS)[5]가 있다. 상기의 방법들은 다음과 같은 공통적인 문제점들을 내포하고 있다.

첫째, 과도한 데이터의 양에 따른 문제를 생각할 수 있다. 기존의 IR 방법들은 많은 양의 데이터를 표현하는데 적합하지 않다. DRCI와 같은 경우 데이터의 양이 증가함에 따라 GIR(Group invalidation Report)의 크기가 증가하게 된다. 또한 이러한 GIR정보에 대한 각 클라이언트가 부담해야하는 자원의 소모 또한 무시할 수 없는 부분이다. MDBS와 MLBS는 BS를 기반으로 하고 있기 때문에 표현해야하는 데이터가 증가하게 되면 비례적으로 많은 계층의 Bit-Sequnce를 필요로 하게 된다. 그와 함께 MDBS와 MLBS는 데이터의 양이 많아짐으로서 False Invalidation이 발생할 확률이 높아지는 구조적인 문제점이 가시화될 가능성이 커지게 된다.

둘째, 데이터의 갱신이 증가함에 따른 문제를 고려할 수 있다. 데이터의 빈번한 업데이트는 IR에 많은 영향을 미친다. DRCI와 같은 경우 OIR(Object Invalidation Report)의 크기가 데이터 아이디(Data ID)와 타임스템프 크기에 비례하여 증가하게 된다. 또한 Bit-Sequence를 기반으로 한 알고리즘에서는 표현할 수 있는 업데이트 데이터의 수가 한정되어 있다는 문제점을 내포하고 있다.

셋째, 선택적 청취(Selective Tuning)의 지원이 불가능하다. IR의 방식은 크게 캐시 레벨(Cache-Level, CL)과 커리 레벨(Query-Level, QL)로 분류할 수 있다.[2] CL이란 한번의 IR의 청취로 전체 캐시 내용의 유효성(validation)을 판단할 수 있는 방법을 말하고, QL은 한번의 IR청취로 하나의 질의(Query)에서 요구하는 데이터의 유효성을 판단할 수 있는 방법을 이야기한다. 그런데 기존의 방법을 활용하면 CL을 지원하면서 동시에 선택적 청취가 불가능하다. 선택적 청취는 이동 컴퓨팅 환경에서 클라이언트에게 중요한 요구사항임을 생각해보면 간과할 수 없는 문제점이다.

3. Timestamp Tree를 이용한 무효화 보고(Invalidation Report)

여기서는 2장에서 제시한 문제점들을 극복할 수 있는 방법으로 B+Tree를 이용한 IR을 제안한다. 이 방법은 기존의 구성과 달리 업데이트된 데이터를 주체로 하지 않는다. 즉 업데이트된 시간을 기준으로 하여 데이터를 분류하여 재구성함으로써 시간을 기준으로 한 그룹화(Grouping)를 유도한다. 이러한 방식은 기존의 방식에 비교해 다음과 같은 특징을 가진다.

우선 기존의 IR 구성의 정적인 그룹화를 유도한다면 여기서 제시하

는 방법은 동적인 그룹화를 유도한다. 즉 기존의 방식은 데이터가 입력됨에 따라 하나의 IR 정보에 많은 데이터를 포함시키는 그룹화를 유도하는데 이러한 방식은 데이터의 성격을 기준으로 인해 선행되어진다. 즉 동종이거나 유사한 데이터를 하나의 그룹으로 묶게 되는데, 이런 접근은 2장에서 이야기한 것과 같이 여러 문제점을 내포하고 있다. 반면 여기서 제시한 방법은 동적인 그룹화를 유도하고 있다. 즉 TT를 이용하여 재 정렬된 데이터들을 그룹화 한다. 이러한 묶음은 기존의 방식과 달리 갱신된 시간에 따른 그룹화로서 다른 시간에 전송된 IR에 따라 다른 그룹 구조를 가지게 된다. 이러한 방식은 업데이트된 시간을 보고하는 것이 IR의 기능임을 고려할 때 가장 적절한 그룹화 방법이라 할 수 있다.

또한 고정된 크기의 타임스탬프를 이용하고 있다. 여기서 제안한 방법은 기존의 방법과 달리 업데이트된 시간을 기준으로 하여 데이터를 재구성하고 이를 이용하여 TT를 구성한다. 이때 TT의 팬아웃(Fan-out)과 깊이(Depth)는 데이터의 양에 따라 선정되게 된다. 이러한 구성은 데이터의 증가에 따른 타임스탬프 크기의 증감을 막게 된다. 반면 갱신된 데이터의 양이 증가하게 되면 정보의 정확도가 떨어지게 되고 갱신된 데이터가 적으면 데이터에 대한 보다 정확한 정보를 전달할 수 있게 된다. 즉 TT의 크기를 고정 시킴으로서 IR의 크기에 많은 비중을 차지하는 시간 값의 부하의 증감을 없애고 IR의 정확도와 그 크기의 동적인 절충을 유도하는 방식이다. 이러한 유도로 인해 데이터의 갱신이 증가하는 문제의 완충작용을 기대할 수 있다.

마지막으로 본 논문에서 제시하는 방법은 선택적 청취가 가능하다. 기존의 방식은 선택적 청취와 CL을 동시에 지원하지 못하고 있다. 이동 컴퓨팅 환경에서 선택적 청취는 클라이언트의 자원 소모에 중요한 변수임을 고려할 때 절실한 기능이다. 또한 캐시의 내용의 유효성을 판단하기 위해 CLO 나 QL보다 효과적이라는 것은 자명한 사실이다. 따라서 이 두 가지 요구 사항을 TT와 IC를 이용하여 얻을 수 있다. TT의 트리 구조는 자신이 원하는 노드의 선택적 청취를 가능하게 하고 있으며, 재구성된 데이터 정보가 IR의 모든 데이터를 청취하는 것이 아니라 클라이언트의 단절 시점 이후의 정보만을 청취하도록 하고 있다. 서버와 클라이언트의 자세한 구성과 동작은 다음과 같이 이루어진다.

3.1 서버측 알고리즘

서버는 주기적으로 IR을 구성하여 클라이언트에게 보내준다. 여기서는 이러한 IR을 구성하는 과정을 다음과 같이 소개한다.

항 목	변 수
window size	w
Data ID	d_x
Time stamp	t_x
Data	D
Current Time	T
Fan out	F
Depth	De

<표 1> 변수 정의

-Step 1
업데이트된 데이터의 아이디를 갱신된 시간과 함께 리스트에 저장한다. 이렇게 구성된 리스트를 Lo라고 하며 다음과 같이 정의한다.
$Lo = \{< d_x, t_x > \mid (d_x \in D) \wedge (T - w < t_x < T)\}$
-Step 2
Lo를 t_x로 오름차순 정렬한다. 이렇게 구성된 리스트를 Lo'이라고 한다.
-Step 3
구성된 Lo'에서 t_1값과 t_{max}값을 t_{from}과 t_{to}에 대응시킨다. 이때 t_{from}과 t_{to} 기준으로 하여 $F+1$ 등분했을 때의 값과 t_x의 근사 값을 $t_{x1}, t_{x2} \ldots t_{xF}$라고 결정한다. 이렇게 결정된 값이 현재 트리 깊이의 노드가 된다. 각 인접한 노드의 값을 t_{from}과 t_{to}에 대응시켜 위의 과정을 반복하여 트리의 깊이가 De인 노드 값들을 결정한다.
-Step 4
Step3에서 결정된 노드들을 이용하여 B+Tree를 구성한다. 이렇게 구성된 트리를 TT라고 한다. Lo'의 d_x을 TT의 리프 노드 값에 대응시켜 오프셋(offset)을 연결한다.
-Step 5
구성된 IR을 클라이언트에게 브로드캐시팅(Broadcasting)한다.

<그림 1> 서버측 알고리즘

다음의 간단한 예시를 이용하여 새롭게 제시하는 IR의 구성을 다시 한번 살펴본다. 데이터의 크기는 16이고 데이터의 아이디와 변경된 시간은 다음과 같이 숫자로 표현한다. 또한 F와 De가 2라고 가정한다. 이러한 데이터를 Step1에 의해 구성된 리스트는 다음과 같다.

ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Timestamp	24	16	10	6	22	18	26	32	2	20	14	30	8	4	12	28

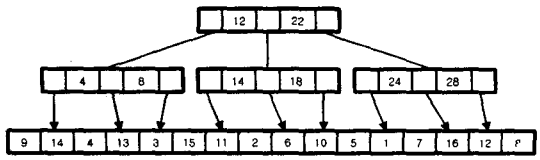
<그림 2> Step1에 따라 구성된 ID와 Timestamp

Step2를 이용하면 다음과 같은 Lo'가 나오게 된다. <그림 3>에서 알 수 있듯이 타임스탬프를 기준으로 오름차순 정렬하고 있다.

ID	9	14	4	13	3	15	11	2	6	10	5	1	7	16	12	8
Timestamp	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32

<그림 3> Step2 에 따라 재구성된 ID와 Timestamp

정렬된 Lo'을 이용하여 Step3을 적용한다. 처음 선택되는 t_{from}과 t_{to}는 2와 32이므로 다음과 같이 계산된다. $t_{from} = 2$, $t_{to} = 32$, $F = 2$이므로 $t_{x1} = 12, t_{x2} = 22$가 결정될 수 있다. 이러한 결정을 반복하여 Step4를 적용 시키면 <그림 4>와 같은 트리가 구성된다.



<그림 4> Step3과 Step4에 따른 트리 구성

<그림 4>와 같이 구성된 트리를 Step5에 의해 클라이언트로 브로드캐스트한다. 브로드캐스팅 방법은 싱글 채널과 멀티 채널에 따라 적절하게 구성할 수 있으나, 여기서는 싱글 채널에서 상위 노드부터 선형적으로 전송되는 방법을 고려한다.

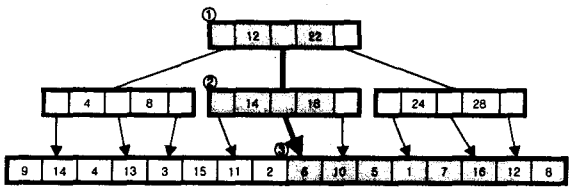
3.2 클라이언트측 알고리즘

-Step 1
노드의 값을 청취하여 자신이 단절된 시간이 속하는 오프셋을 결정한다. 현재 상태를 대기 모드(Doze mode)로 전환한다.
-Step 2
선택된 오프셋을 이용하여 자신이 원하는 노드가 방송되는 시간에 활동 모드(Active mode)로 전환한다. 현재 청취하는 노드가 리프 노드(Leaf node)가 아니라면 Step1부터의 과정을 반복한다.
-Step 3
선택한 오프셋을 이용하여 단절된 이후에 갱신된 데이터의 아이디 정보를 청취한다.
-Step 4
청취한 데이터의 아이디를 이용하여 자신의 캐시에 저장하고 있는 데이터의 무결성을 검사한다.

<그림 5> 클라이언트측 알고리즘

클라이언트는 3.1절에서 소개한 서버가 구성한 IR을 수신하게 된다. 이때 일반적인 IR의 수신과 달리 트리를 이용한 선택적 청취를 하게 된다. 자세한 방법은 <그림 5>와 같다.

3.1절에서 제시한 예시를 이용하여 클라이언트의 IR의 활용을 좀 더 자세히 알아보면 다음과 같다. 즉 여기서는 클라이언트가 17초에 단절되었고 32초 이후에 이 IR을 전송 받았다고 가정한다. 이때 클라이언트가 청취하는 노드와 그 순서는 다음과 같다.



<그림 6> 클라이언트에서 IR의 청취

<그림 6>에서 보인 것과 같이 Step1을 적용하여 클라이언트는 루트 노드를 청취하게 된다. 이때 자신의 단절 시간인 17초가 12보다 크고 22보다 작음을 확인하고 중간의 오프셋을 이용하여 다음에 자신이 들어야할 노드가 방송될 시간을 확인 한다. 대기모드로 전환한 클라이언트는 Step2를 적용하여 ②번 노드가 방송될 때 활동모드로 전환하게 된다. 노드②에서 자신의 단절된 시간을 기준으로 다음 청취할 노드를 결정하게 된다. ②노드는 리프 노드이므로 Step3 과정에 의해 자신이 단절된 이후에 갱신된 데이터의 아이디를 청취하게 된다. 이를 이용해 Step4의 과정인 캐시내용의 무결성 검사를 시행하게 된다. 위의 트리에서 색이 칠해진 부분이 클라이언트가 직접 청취를 하는 부분이다. 즉 트리의 모든 내용을 클라이언트가 청취하는 것이 아니라, 클라이언트가 필요한 정보를 선택 청취하게 유도하고 있다.

4. 성능 평가

본 논문에서 제안한 IR 구성의 성능을 DRCI방법과 비교해보기 위해 다음과 같은 환경을 설정한다.

항 목	변 수	값 설정
Total data	M	255, 65535
Updated data / sec	N	
Time stamp	T bytes	4
Data ID	O bytes	1, 2
Group ID	G bytes	1, 2
Object window size	Ow	50
Number of group	K	10, 60
Fan-out	F	2, 3
Depth	D	3, 4
B+ window size	Bw	50
Offset	E	1

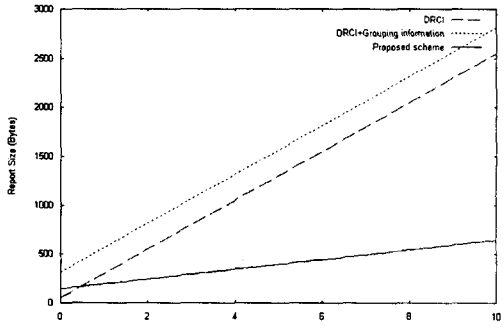
<표 2> 성능 평가를 위한 변수 설정

데이터의 증가에 따른 실험을 위해 데이터의 양을 $255(2^8)$와 $65535(2^{16})$를 이용한다. 또한 그에 따른 DRCI의 그룹 수와 본 논문에서 제시하는 구조의 팬 아웃 및 트리의 깊이를 미리 결정한다. 이때 초당 갱신되는 데이터의 수를 증가 시키며 IR의 길이를 비교 해본다.

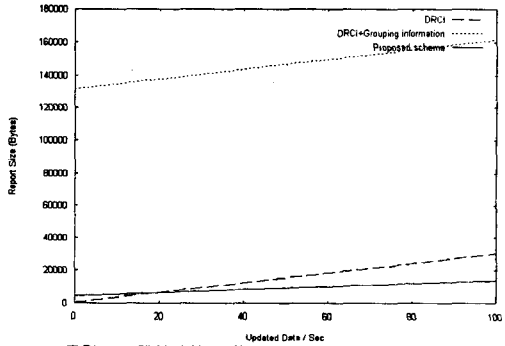
<그림 7>은 데이터의 수가 255일 경우를 비교하고 있다. 이 그림은 DRCI 방식으로 IR을 전송했을 때와 DRCI의 구성을 위해 그룹 정보를 함께 전송했을 때를 표시하고 있다. 또한 본 논문에서 제시한 방식의 IR을 표시하고 있다. 실험 결과에서 알 수 있듯이 갱신 빈도가 낮을 경우 기존 DRCI 방식이 유리하나 일정 빈도 이상일 때 더 좋은 성능을 보임을 알 수 있다.

<그림 8>은 데이터의 수가 65535로 많은 양의 데이터일 경우를 실험 하고 있다. 이 경우에서는 DRCI의 그룹화에 따른 비용이 상당히 큼을 알 수 있다. 이에 비해 여기서 제안한 방식은 갱신 빈도가 증가 할수록 더욱 좋은 성능을 보임을 알 수 있다.

이 실험에서 유추할 수 있듯이 여기서 제시한 방법은 M과 N이 증가로 인한 IR의 변화가 적은 것을 알 수 있다. 이것은 TT를 이용한 타임스탬프 크기를 고정 시킴으로써 얻을 수 있는 결과 이다. 즉 더 적은 양의 타임스탬프를 이용하여 비슷한 양의 정보를 표현하는 이 방식의 장점이라 할 수 있다.



<그림 7> 데이터의 수가 255일 경우 DRCI와의 비교



<그림 8> 데이터의 수가 65535일 경우 DRCI와의 비교

5. 결론 및 향후 연구

본 논문에서는 stateless 서버 환경에서 클라이언트의 효율성을 위해 사용하는 캐시의 일관성을 유지하는 방법을 제시하였다. 이러한 방법은 타임스탬프 트리와 시간순으로 정렬된 데이터 아이디를 바탕으로 많은 양의 갱신된 데이터의 전송 및 선택적 청취를 가능하게 한다. 또한 이러한 사항은 기존 IR과의 비교를 통해 확인 할 수 있었다.

현재 이러한 IR의 구성은 기존 IR에 비해 상대적으로 더 정확하고 많은 양의 정보를 표현하기 위해 연구되었다. 따라서 IR의 window size의 증가를 유도 할 수 있으나, 이와 함께 Latency의 증가도 수반하게 된다. 그러므로 이러한 단점을 극복하기 위한 연구가 필요하다.

6. 참고 문헌

[1] D.Barbara and Tomasz Imielinski, "Sleepers and workaholics: caching strategies in mobile environments", ACM SIGMOD Record, Proceedings of the 1994

[2] Kian-Lee Tan, Jun Cai, Ben Chin Ooi, "An evaluation of cache invalidation strategies in wireless environments" ,IEEE Transactions on Parallel and Distributed Systems, Volume 12 Issue 8, August 2001

[3] M. Satyanarayanan, J. H. Howard, D. N. Nichols, R. N Sidebotham, A. Z Spector, and M. J. West. "The ITC Distributed File System: Principles and Design", In Proceedings of the 10th ACM Symposium on Operating Systems Principles, December 1985.

[4] Khurana, S. and Kahol, A. and Gupta, S.K.S. and Srimani, P.K. "An efficient cache maintenance scheme for mobile environment", Distributed Computing Systems, 2000. Proceedings. 20th International Conference on, 2000.

[5] Elmagarmid, A. Jin Jing; Helal, A. Choonhwa Lee, "Scalable cache invalidation algorithms for mobile data access",Knowledge and Data Engineering, IEEE Transactions on , Volume: 15 , Issue: 6 , Nov.-Dec. 2003