모바일 환경에서 타임스탬프 트리 기반 캐시 무효화 보고 기법

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

정 성 원 † 이 학 주 ††
(Sungwon Jung) (HakJoo Lee)

요 약 이동 컴퓨팅(Mobile Computing)환경에서 빈번한 접속 단절은 클라이언트의 캐시 일관성(Consistency)문제로 직결된다. 이러한 캐시 일관성 문제를 해결하기 위해 무효화 보고(Invalidation Report)에 관한 연구가 진행되고 있다. 그러나 기존의 무효화 보고 기법은 서버 측의 데이타의 수가 많아지거나, 갱신(Update)되는 데이타의 양이 증가하게 되면 무효화 보고 기법의 크기의 증가 및 캐시 효율성의 저하를 가지고 온다. 또한 캐시 전체의 무효화를 확인하는 보고 기법은 선택적 청취(Selective tuning)를 지원하지 못한다. 본 논문에서는 이러한 기존 방식의 문제점을 해결하며 효율성을 개선한 무효화 보고 기법으로써 TTCI(Timestamp Tree-based Cache Invalidation scheme)를 제안한다. 갱신된 데이타의 타임스탬프를 이용하여 타임스탬프 트리를 구성하고 데이타의 아이디를 갱신된 순서대로 나열하여 무효화 보고를 만든다. 이를 활용하게 되면 각 클라이언트는 자신의 단절(Disconnect)된 시점에 맞는 정보를 확인하여 캐시의 이용률을 증가 시킬 수 있다. 또한 트리 구조를 이용하여 선택적 청취를 가능하게 함으로써 클라이언트의 자원 소비를 줄일 수 있다. 이러한 본 구조의 효율성을 검증하기 위해 기존의 기법인 DRCI(Dual-Report Cache Invalidation)와 성능비교를 실시한다.

키워드 : 무효화보고, 캐시 일관성, 데이타 브로드캐스트, 이동 컴퓨팅

Abstract Frequent disconnection is connected directly to client's cache consistency problem in Mobile Computing environment. For solving cache consistency problem, research about Invalidation Report is studied. But, existent invalidation report structure comes with increase of size of invalidation report structure and decline of cache efficiency if quantity of data become much, or quantity of updated data increases. Also, while existent method confirms whole cache, invalidation report doesn't support selective listening. This paper proposes TTCI(Timestamp Tree-based Cache Invalidation scheme) as invalidation report structure that solve problem of these existing schemes and improve efficiency. We can make TTCI using timestamp of updated data, composing timestamp tree and list ID of data in updated order. If we utilize this, each client can confirm correct information in point that become own disconnecting and increase cache utilization ratio. Also, we can pare down client's resources consumption by selective listening using tree structure. We experimented in comparison with DRCI(Dual-Report Cache Invalidation) that is existent techniques to verify such efficiency of TTCI scheme.

Key words : Invalidation Report, Cache Consistency, Data Broadcast, Mobile Computing

1. 서 론

휴대용 컴퓨터와 무선 네트워크의 발전은 모바일 컴퓨팅(Mobile Computing)이라는 새로운 패러다임을 만들어 냈다. PDA(Personal Digital Assistant)와 랩톱(Laptop)과 같은 모바일 장비와 무선 LAN, 위성 서비스, 이동 전화와 같은 무선 네트워크의 결합은 시간과 공간의 제약을 뛰어 넘은 정보의 접근을 가능하게 하고

· 이 논문은 부분적으로 한국과학재단 특정기초과제 No. R01-2006-000-10536-0(2007)와 2007년도 두뇌한국21사업에 의하여 지원되었음
† 종신회원 : 서강대학교 컴퓨터학과 교수
jungsung@sogang.ac.kr
†† 학생회원 : 서강대학교 컴퓨터학과
or4nge@lge.com
논문접수 : 2005년 2월 18일
심사완료 : 2007년 3월 20일

있다[1-4].

이러한 모바일 컴퓨팅 환경에서는 대역폭의 제한과 모바일 장비의 한정된 전원이라는 중요한 제약사항을 가지고 있다. 사용자는 유선환경에 비해 낮고 비대칭적인 대역폭을 이용하여 정보를 접근해야만 한다. 모바일 네트워크는 무선 주파수를 이용함으로써 유선에 비해 낮은 전송 속도를 가지게 되고 모바일 장비의 낮은 전력과 장비의 특수성으로 인해 수신에 비해 송신의 열등한 성능을 가지게 된다. 또한 모바일 장비의 이동성으로 인한 제한된 전원을 가지게 된다. 이러한 환경으로 인해 모바일 컴퓨팅 분야는 효과적인 데이타 전달이 가능한가를 중요한 문제로 자리매김 시키고 있다[2,5].

이에 따라 기존의 유선환경에서와 같은 서버 클라이언트의 관계는 심각한 자원 부족을 초래하게 된다. 클라이언트의 요구에 따라 서버 측의 데이타를 전송하는 것은 대역폭의 부족 현상에 직결되며 클라이언트 질의에 따른 전원의 부족 현상과 처리 시간의 증가를 초래하게 된다. 따라서 서버에서는 다수의 클라이언트가 요구하는 데이타를 주기적으로 브로드캐스트(Broadcast)하고 클라이언트는 원하는 데이타를 선별적으로 수신함으로써 이러한 단점을 극복하고 있다.

이러한 환경에서는 클라이언트가 요구한 데이타를 수신하기 위해 상당시간 브로드캐스팅을 기다려야만 하는 문제점을 내포하게 된다. 따라서 이와 같은 문제점을 개선하기 위해 클라이언트는 자주 이용하는 데이타를 캐시에 저장한다. 하지만 클라이언트는 불안전한 네트워크 환경으로 인한 빈번한 접속 단절이 발생하게 되고 서버의 데이타와 일치하지 않는 캐시의 데이타를 이용할 가능성이 존재하게 된다. 이러한 문제를 해결하기 위해 서버는 무효화 보고(Invalidation Report, IR)를 클라이언트에게 주기적으로 제공하여 서버의 데이타와 일관성(Consistency)을 유지할 수 있게 한다[6].

최근 이러한 데이타의 일관성을 유지하기 위한 무효화 보고 기법들이 많이 등장하고 있다. 하지만 기존의 기법들은 무효화 보고에 표시할 데이타의 증가 시 많은 문제점을 내포하게 된다. 예를 들면 데이타의 증가 시 급격한 캐시 사용율의 저하와 IR 크기의 증가를 가져올 수 있다. 이러한 점을 극복하기 위해 본 논문에서는 효과적인 무효화 보고 기법으로서 TTCI(Timestamp Tree-based Cache Invalidation scheme)를 제안한다. TTCI는 갱신된 데이타의 수 보다 적은 수의 타임스탬프(Timestamp)를 이용하여 트리를 구성하고 이를 이용해 갱신된 데이타의 ID를 확인할 수 있도록 유도하는 기본적인 구조를 가지고 있다. 이러한 방법은 동적인 그룹화, 무효화 보고 기법에 이용되는 타임스탬프 수의 한정, 선택적 청취의 가능이라는 세 가지 특징을 가지고 있게 된다.

본 논문의 2장에서는 기존의 무효화 보고 기법에 대해 살펴보고 기존의 구조가 가지고 있는 문제점에 대해 살펴본다. 3장에서는 이러한 단점을 극복하고자 제안하는 TTCI 구조에 대해 소개한다. 또한 4장에서는 제안한 무효화 보고 기법의 효율성을 검증하기위해 성능을 평가하며 5장에서 결론을 내리게 된다.

2. 관련연구

본 장에서는 기존의 무효화 보고 기법(Invalidation Report, IR)의 기본적인 전달 방식과 이때 캐시의 구성에 대해 살펴본다. 또한 이러한 환경에서 이용된 무효화 보고 기법을 살펴보고 기존 무효화 보고 기법이 가지는 문제점들을 살펴본다.

모바일 환경에서 서버로부터 클라이언트로의 데이타 전송은 일반적인 유선 환경의 서버 클라이언트환경과는 다른 방식을 취한다. 즉 클라이언트는 서버에게 특정 데이타를 요구하는 것이 아니라 서버의 데이타를 주기적으로 클라이언트에게 전송한다. 이러한 주기적인 전송을 브로드캐스트 디스크(Broadcast Disk)라고 하며 모바일 환경에서 데이타 전송의 특수성을 잘 표현하고 있다.

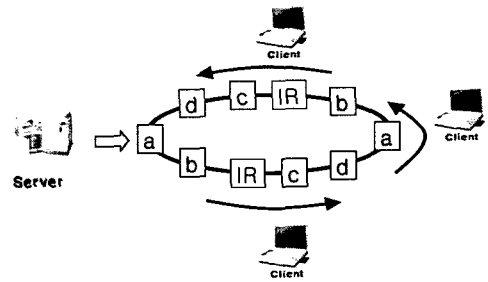

그림 1 데이타 브로드캐스트

이렇게 전송된 데이타는 각 클라이언트의 요구와 서버의 방송주기에 따라 자신이 원하는 데이타를 캐시에 저장하게 된다. 저장된 데이타가 유효한지 알려주기 위해 서버는 IR을 주기적으로 전송하게 된다. 따라서 클라이언트는 IR을 이용하여 캐시에 저장하고 있는 데이타가 유효한지 확인할 수 있다. IR의 기본적인 구성은 일정 기간동안 갱신된 데이타의 ID정보를 포함하고 있다. 이 정보는 $[T-w\times L, T]$ 시간 동안 발생한 갱신 정보를 제공하게 된다. 이때 T는 IR의 타임스탬프를 말하며, w는 갱신된 데이타의 로그 윈도우 크기를 나타낸다. 또한 L은 IR이 전송되는 간격을 말하며 w가 2인 IR을 나타내면 다음 그림과 같다[6].

서버는 클라이언트의 캐시 일관성을 확인하기 위해

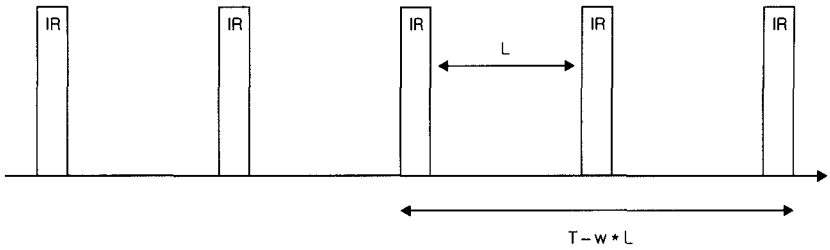

그림 2 IR 전송

IR을 구성하여 주기적으로 전송하게 된다. 이때 클라이언트는 다음 그림과 같은 캐시를 가지게 된다. 왼쪽에 나타난 타임스탬프는 가장 최근에 서버로부터 데이타를 확인한 시점의 타임스탬프이다. 즉 이 시간까지는 캐시에 저장된 데이타의 무결성을 보장할 수 있다. 이 값을 기준으로 캐시에 저장되어 있는 데이타가 현재 유효한지 그렇지 않은지 판단하게 된다. 즉 IR의 타임스탬프와 비교해서 유효한 데이타를 유지하게 된다. 그 외 데이타 ID와 데이타의 값은 캐시의 정보로써 저장되게 된다.

TimeStamp	Data ID	Data ID	...	Data ID
	Data value	Data value	...	Data value

그림 3 클라이언트 캐시의 구조

이러한 IR기법들이 가지고 있는 공통적인 문제점을 살펴보면 다음과 같다.

- 데이타의 수에 비례하여 IR의 크기의 증가 및 성능의 감소
- 데이타의 갱신 횟수가 증가함에 따른 IR의 크기 증가 및 성능의 감소
- Cache-Level의 선택적 청취 불가능

우선 데이타의 수가 증가 하게 되면 IR의 크기가 커지게 된다. 기존의 IR방법들은 많은 양의 데이타를 표현하는데 적절하지 않다. 대표적으로 GCORE(Group cache COld update-set REtention)[8], GCI(Group-based Cache Invalidation)[9]와 같은 그룹 기반의 IR 기법들은 데이타의 양이 많아지게 되면 IR에 포함되는 그룹 정보가 커지게 된다. 그룹에는 그 그룹에 속하는 데이타 중 가장 최근에 갱신된 데이타의 타임스탬프가 표시되게 된다. 따라서 그룹화 정도(Granularity)에 비례하여 IR의 길이를 필요로 한다. BS(Bit-Sequence)[10], MDBS (*MultiDimensional Bit-Sequence*)*[11]*, *MLBS*(*MultiLevel* Bit-Sequence)[11]와 같이 Bit-Sequence 기반의 기법에서도 데이타의 양이 많아짐에 따라 IR의 크기가 커지게 된다. BS는 기본적으로 모든 데이타에 대해 하나의 bit를 대응 시키고 있다. 따라서 데이타의 수를 N이라고 한다면 BS 계층마다 항상 N, $N/2$, $N/2^2$, ... bit가 소모되게 된다. 약 $2N$ bit가 소모되게 된다. 또한 이러한 BS에 $logN$ 계층만큼의 타임스탬프가 존재하게 된다. 이 방식은 데이타의 갱신이 빈번한 것과 상관없이 데이타의 전체 양에 비례하여 IR이 구성된다.

또한 데이타가 많아짐에 따라 IR의 성능이 감소하게 된다. 즉 거짓 무효화(False Invalidation)가 발생할 확률이 급격히 커지게 된다. 여기서 말하는 거짓 무효화란 실제 캐시에 저장된 데이타는 유효하나 데이타가 무효화 되는 상황을 이야기 한다. DRCI와 같이 그룹화를 기반으로 한 기법은 그룹화정도에 따라 전체 그룹의 요소들이 하나의 타임스탬프로 유효성이 판별되기 때문에 이러한 상황이 발생할 수 있다. 또한 BS와 같은 IR은 하나의 계층에서 최대 N/2개의 데이타에 대해 하나의 타임스탬프를 이용하므로 이러한 상황은 더욱 빈번히 발생할 수 있다. 이러한 단점을 극복하기 위해 MLBS와 MDBS가 등장하였다. MDBS는 중첩된 그룹화를 통해 이러한 단점을 극복하였고, MLBS는 특정 부분에 좀더 자세한 정보를 제공함으로써 이러한 단점을 극복하였다. 하지만 이 경우 모두 갱신되는 데이타가 고르게 분포하여 많아질 경우 BS방법에서 나타났던 문제점이 그대로 반복된다.

기존의 IR은 갱신 회수의 증가가 IR의 길이와 성능에 영향을 미친다. 그룹화를 기반으로 한 IR 중 DRCI(Dual-Report Cache Invalidation)[7]와 같은 경우 OIR(Object Invalidation Report)의 길이가 갱신된 데이타의 수에 비례하여 증가하게 된다. OIR은 데이타의 ID와 데이타가 갱신된 시점의 타임스탬프로 구성되어 있다. 따라서 갱신 횟수가 증가함에 따라 OIR은 데이타 ID의 길이와 타임스탬프 길이에 비례하여 증가하게 된다. 또한 GIR은 하나의 그룹으로 여러 개의 데이타의 무효성을 나타낸다. 즉 한 그룹의 타임스탬프가 그 그룹 전체의 데이타의 갱신 정보를 대변하게 된다. 따라서 갱신이 빈번해지면 유효한 데이타도 무효해질 확률이 더 커지게 된다. 즉 그룹화로 인한 효과가 반감되게 된다.

앞서 설명했듯이 IR의 기능을 분류해보면 크게 Cache-Level과 Query-Level로 나눌 수 있다. Cache-Level이란 한번의 IR을 전송하여 현재 저장하고 있는 캐시의 유효성을 판단하는 것을 말한다. 다시 말해 IR을 전송받게 되면 클라이언트에 저장하고 있는 모든 데이타의 유효성을 판단하게 된다. 유효하지 않은 데이타는 제거(Drop)하여 캐시를 유지하는 방법이다. 이러한 방법은 클라이언트가 어떠한 데이타를 요청하는지에 관계없이 한번의 IR로 사용자의 요구를 수용할 수 있다. 반면 Query-Level은 한번의 IR을 청취하면서 클라이언트가 질의하고자 하는 데이타의 유효성만을 판단하게 된다. 즉 클라이언트가 어떠한 질의를 발생시켰을 때 질의에서 요구하는 데이타의 유효성을 판단하게 된다. 이러한 방법은 요구하는 데이타에 대해 목적성을 가지고 접근함으로써 선택적 청취를 가능하게 한다.

위에서 살펴본 바와 같이 Cache-Level을 만족하며 선택적 청취를 가능하게 하는 IR방법은 찾기 어렵다. 선택적 청취가 모바일 환경에서는 매우 중요한 요소임을 고려하면 꼭 필요한 기능이다. 모바일 환경에서 클라이언트의 자원의 협소함은 자명한 사실이기 때문이다. 또한 IR의 효율성을 위해서는 한번의 IR로 전체 캐시의 유효성을 판단하는 것도 중요하다. 따라서 선택적 청취를 가능하게 하는 Cache-Level의 IR은 필요한 기능임에도 불구하고 이를 만족하는 IR을 찾기 어렵다.

이외에도 기존의 IR 방법을 사용하면서 캐시의 재사용율을 높이고자 클라이언트와 서버가 협력하여 캐시의 유효성을 판단하는 기법으로 CCI(Cost-Based Invalidation)[15]와 ECI[16] 등이 제안되었다. 하지만 다운링크 보다 큰 비용을 필요로 하는 업링크를 사용해야 하고, 클라이언트의 수가 많을 경우 서버의 과부하를 초래할 수 있기 때문에 확장성면에서도 문제점을 안고 있다.

따라서 본 논문에서는 업링크의 사용하지 않는 기존의 IR 방식의 단점을 보완하고 개개의 클라이언트의 접속단절시간에 따른 IR의 효과적인 이용을 유도하고자 TTCI(Timestamp Tree-based Cache Invalidation scheme) 기법을 제안한다. 본장에 이어 TTCI의 개념과 구성방식에 대해 살펴본다.

3. TTCI 알고리즘

여기서는 무효화 보고 기법(Invalidation Report, IR)의 한 방법인 TTCI(Timestamp Tree-based Cache Invalidation scheme)에 대해 설명한다. TTCI는 TT(Timestamp Tree)와 IS(Id liSt)로 구성되어 있다. TT는 데이타가 갱신된 타임스탬프의 트리로 구성되어 있다. 또한 IS는 갱신된 데이타의 ID 리스트로 구성되어 있다. 이러한 TTCI의 구성 방법에 대해 논의하고 TTCI 기법의 효율성에 대해 살펴본다. 또한 서버 측과 클라이언트 측의 TTCI 적용 방법에 대해 설명한다.

3.1 TTCI 기본 개념

TTCI는 TT와 IS로 구성되어 있다. TT는 브로드캐스트 데이타의 인덱스와 유사한 부분으로 캐시에서 저장하고 있는 마지막으로 검증된 타임스탬프를 기준으로 하여 자신이 단절된 이후에 갱신된 정보를 선택할 시점을 알려주게 된다. IS에는 갱신된 데이타의 ID가 저장되어 있고 이 정보를 이용하여 캐시의 무효성을 판단하게 된다.

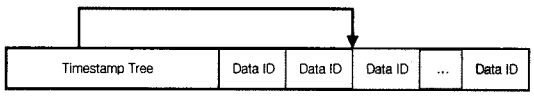

그림 4 TTCI의 방송

TT는 데이타 ID 리스트의 읽어야할 시점을 결정하는 역할을 한다. 그림에서 볼 수 있듯이 TT를 이용해 IS의 읽을 시점을 결정하고 그 부분부터 마지막 까지 데이타 ID를 읽게 된다. IS의 색깔이 있는 부분이 읽혀진 부분을 나타낸다. 즉 TT는 IS의 인덱스와 같은 역할을 한다. 이와 같은 기법은 선택적 청취를 가능하게 한다. 우선 TT는 트리 구조를 기본으로 하고 있다. 따라서 트리의 같은 레벨의 정보 중 자신이 원하는 정보만을 선택 청취한다. 이것은 계층적 구조를 가진 정보의 일반적 특징이기도 하다. 또한 TT를 이용하여 데이타 ID 리스트 즉 IS의 일부분만을 접근하므로 선택적 청취를 유도할 수 있게 된다.

TT의 구성의 기본 아이디어는 다음과 같다. TT는 트리 구조를 기본으로 하며 서버 측에서 갱신된 데이타들의 정보를 트리로 구성하고 있다. 이때 트리를 구성하는 방식은 Top-Down 개념을 활용한다. B-Tree 같은 경우 데이타의 삽입될 위치를 검색한 후에 삽입하는 과정에서 노드의 fanout에 따라 노드가 분할되거나 병합되는 과정을 거치게 된다. 이때 삽입은 리프 노드에서부터 발생하므로 트리의 구성은 Bottom-Up으로 변경이 되게 된다. 하지만 본 논문에서 제안하는 TT는 한정된 정보를 바탕으로 트리를 구성하여 효과적으로 전송하는 것이 목적이므로 Top-Down 방식을 이용한다. 즉 트리의 추가적인 삽입과 삭제가 없는 고정된 정보를 가지고 상위 계층부터 트리를 구성한다. 또한 TT에 사용되는 값을 리프 노드의 정보를 이용해 구성하는 것이 아니라 이미 주어진 타임스탬프 정보만을 이용하여 구성한다. 이러한 방식은 클라이언트에서 무효화 판단을 위해 TT의 이용 시 실제 갱신된 타임스탬프정보를 가지고 검색을 하기 때문에 좀더 정확한 정보를 제공할 수 있다. 즉 트리의

표 1 데이타 갱신 정보

'Timestamp	Data ID
1	d5,d3
2	d2
3	d6
4	d4,d8
5	d1
6	d9
7	d6

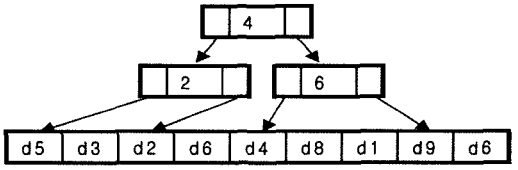

그림 5 TT의 구성

내부에서 이용하는 정보가 리프 노드를 찾기 위한 인덱스의 역할 뿐만 아니라 분류의 기능을 줄 수 있다.

TT의 구성은 타임스탬프를 fanout의 크기를 기준으로 나누는 것을 원칙으로 한다. 즉 시간 순으로 정렬된 타임스탬프를 fanout 만큼씩 묶어 경계 값을 그 노드의 정보로 이용한다.

위의 그림 5는 1부터 7까지의 타임스탬프를 fanout 2로 가정하여 TT를 구성한 예이다. 예제에서 알 수 있듯이 1부터 7까지의 타임스탬프를 2등분하여 경계 값이 되는 4를 첫 번째 노드로 구성하고 분류된 영역을 다시 2등분하여 경계가 되는 2와 6을 그 다음 레벨의 노드로 결정하게 된다. 이와 같은 방법으로 TT를 구성하게 된다.

이때 IS의 구성은 갱신된 데이타의 ID를 나열하여 구성한다. 그림 5와 같다. 각 클라이언트가 청취해야하는 시점은 TT를 이용하여 알 수 있고, IS에서는 이 정보를 기반으로 갱신된 데이타의 ID 정보를 제공한다. 이때 IS는 갱신된 시간을 기준으로 정렬되어 있다. 따라서 TT에서 제공하는 포인터 이후의 모든 데이타 ID가 갱신된 정보임을 알 수 있다.

3.2 TTCI 특징

모바일 환경에서 클라이언트의 캐시 관리 방법은 수동적이며 많은 제약 사항들을 가지고 있다. 즉 서버와의 주기적인 통신을 보장할 수 없으며 데이타의 요청을 이용한 데이타의 접근은 고 비용을 요구한다. 따라서 모바일 환경에서 캐시를 유지하기 위해서 IR이 이용된다.

이러한 IR을 연구한 선행 논문에서는 크기와 효율성을 최적화하기 위해 노력해 왔다. 본 논문에서도 이러한 효율성과 크기 문제를 고려하며, 추가적으로 선택적 청취를 고려하여 다음과 같은 세 가지 방법을 이용한다.

- 동적인 그룹화
- 일정수의 타임스탬프 이용
- 선택적 청취

기존의 IR 기법은 그룹화 방법을 이용하여 IR의 윈도우 크기를 크게 할 수 있으며 IR의 크기를 줄일 수 있다. 하지만 그룹기반의 IR같은 경우 데이타 ID와 그룹화에 대한 정보를 가지고 있어야 한다. 즉 IR에 앞서 정해진 그룹화 방법에 대한 정보를 클라이언트는 알고 있어야만 한다. 여기서 그룹화 방법에 대한 정보란 어떤 데이타 ID가 어떤 그룹에 속해 있는지에 대한 정보를 말한다. 이러한 그룹화 정보를 매번 클라이언트에게 알려 주거나 클라이언트가 사전에 알고 있어야만 IR의 활용이 가능하다. 이러한 그룹 정보 자체는 클라이언트와 서버 모두에게 큰 오버헤드로 작용하게 된다. 또한 이와 같은 그룹화는 데이타의 성격 즉 일반적인 특성을 기준으로 묶여지게 된다.

또한 BS(Bit Sequence) 계열의 IR도 한 bit가 지시하고 있는 데이타 ID정보를 클라이언트가 알고 있어야 BS정보를 해석할 수 있다. 즉 BS의 각 비트가 의미하는 데이타의 정보를 미리 청취한 상태에서 클라이언트의 데이타의 무결성을 검사할 수 있다.

반면 본 논문에서 제안한 TTCI는 선행된 정보를 사용하지 않는다. 즉 IR의 구성에 관한 정보를 미리 가지고 있지 않아도 된다. IR에 구성되어 있는 정보만을 이용하여 캐시의 유효성을 확인하게 된다. 이는 TTCI가 동적인 그룹화를 이용하기 때문이다. DRCI(Dual-Report Cache Invalidation)와 같이 데이타의 성격에 따라 그룹화를 하는 것이 아니라 IR의 구성 단계에서 갱신된 타임스탬프 값을 기준으로 그룹화를 한다. 즉 전체 갱신된 데이타 ID를 타임스탬프를 기준으로 정렬하여 그룹을 구성한다. 이러한 방식을 적용하여 클라이언트가 속한 그룹이 어딘지 TT에 명시 하게 된다. 클라이언트는 가장 최근에 전송 받은 타임스탬프를 기반으로 자신이 속한 그룹을 찾게 된다. IR에서 가장 중요한 변수로 고려되는 타임스탬프를 이용하여 그룹화를 동적으로 구성한다.

두 번째로 IR의 구성에서 큰 비중을 차지하는 타임스탬프의 갱신된 데이타의 수에 비해 적은 양은 이용한다. 앞서 살펴본 바와 같이 기본적인 캐시는 데이타의 ID와 마지막으로 확인된 타임스탬프, 데이타의 값으로 구성되어 있다. 이때 데이타의 유효성을 판단하기 위해서 IR은 기본적으로 데이타의 ID와 타임스탬프가 요구 된다. 이때 데이타의 ID는 최소 $\log n$ bits를 가지고 표현 할 수 있다. n은 전체 데이타의 수를 말한다. 즉 n가지 상황을 나타낼 수 있는 최소의 bit만 존재하면 데이타의 ID를 표현 할 수 있다. 반면 타임스탬프는 서버와 클라이언트 간의 절대적인 시간을 표시해야한다. 따라서 데이타 ID에 비해 많은 정보를 포함해야 하므로 상대적으로 많은

bit를 차지해야만 한다. 기존의 IR기법들은 그룹화를 이용해 타임스탬프의 수를 줄였다. 본 논문에서 제안하는 TTCI는 일정한 타임스탬프 수를 보장하여 IR의 길이가 갱신된 데이타의 수에 민감하지 않도록 유도한다. 다시 말해 IR에서 일정 비중을 차지하고 있는 타임스탬프의 수를 고정함으로써 IR의 길이를 어느 정도 예측할 수 있고, 조절할 수 있도록 하고자 한다.

마지막으로 TTCI는 선택적 청취를 이용한다. 모바일 환경에서 선택적 청취는 클라이언트의 전력의 소비를 줄일 수 있는 중요한 변수이다. 일예로 AT&T에서 제작한 Hobbit Chip과 같은 경우 활동모드(Active mode)에서 250mW를 소비한다. 하지만 대기모드(Doze mode)에서는 50mW를 소비한다[12]. 이러한 이유로 모바일 환경에서 데이타를 전송하기 위한 방법으로 사용되는 브로드캐스트 디스크에서는 여러 가지 인덱스 구조를 이용한 선택적 청취를 유도하고 있다.

본 논문에서 제안한 IR도 이와 같은 모바일 환경에서 동작해야하므로 선택적 청취 기능은 중요한 요소이다. TTCI 기법에서 TT에 해당하는 부분은 트리 구조로써 선택적 청취가 가능하다. 트리의 구조상 각 레벨의 하나의 노드만을 청취하면 원하는 정보를 얻을 수 있게 된다. 또한 TT에 의해 각 클라이언트가 수신해야하는 데이타의 ID 리스트의 시점을 지적해 줌으로써 전체 데이타 ID 리스트 즉 전체 IS가 아닌 개개의 클라이언트가 청취해야하는 시점을 지정해 주게 된다. 이는 TTCI에서 사용하는 특징 중 하나가 된다. 뿐만 아니라 이러한 선택적 청취는 Cache-Level에서 동작하므로 클라이언트의 전체 캐시를 관리하는데 더욱 효과적이라 할 수 있다.

이러한 TT의 구성은 서버 측 오버헤드로 작용할 수 있다. 하지만 모바일 환경은 서버 측의 오버헤드 보다 클라이언트 자원의 협소함이 더 중요한 변수이므로 이러한 구성상의 오버헤드는 간과 할 수 있다. MDBS와 MLBS같은 기법도 이러한 서버 측의 오버헤드는 언급하지 않고 있다.

3.3 구성 알고리즘

서버는 주기적으로 IR을 구성하여 클라이언트에게 전송하게 된다. 본 절에서는 앞서 소개한 TTCI의 구체적인 구성 방법을 소개한다.

정의 3.1. 갱신된 데이타의 타임스탬프를 중복된 값을 배제하여 증가하는 순서로 정렬하였을 때 i번째 타임스탬프를 TS_i라고 한다. 또한 TS_i에 갱신된 데이타의 아이디를 $ID_1,ID_2,...,ID_q$라고 한다. 이때 TSL_i는 TS_i와 그때 갱신된 데이타의 아이디를 저장하는 리스트를 나타낸다. 즉 TSL_i = $<TS_i,ID_1,ID_2,ID_3,...,ID_q>$와 같다. OLIST는 이러한 TSL_i를 구성원으로 하는 리스트이다. 즉 OLIST의 TSL수를 N이라 할 때 OLIST = $<TSL_1, TSL_2,TSL_3,...,TSL_N>$와 같이 정의한다. 여기서 TSL_i의 TS를 LTS_i라고 할 때 다음과 같은 성질을 만족한다.

1. $LTS_1 < LTS_2 < LTS_3 ... < LTS_N$. 즉 LTS의 값은 중복되는 값을 배제하여 증가하는 순서로 정렬되어 있다.

예를 들어 표 2의 갱신된 리스트를 이용하여 표 3와 같이 OLIST를 구성한다. 표에서 알 수 있듯이 갱신된 리스트인 ULIST는 데이타의 아이디 순서로 정렬되어 있다. 즉 데이타가 갱신 될 때의 타임스탬프를 저장하는 역할을 하게 된다. 이러한 정보를 이용하여 타임스탬프를 정렬하고 동일한 타임스탬프를 가진 데이타는 하나의 타임스탬프를 공유하도록 OLIST를 구성한다. 예를 들어 d13과 d17은 타임스탬프 8일 때 갱신된 데이타로 하나의 TSL에 속하게 된다.

정의 3.2. OLIST의 i번째 TSL인 TSL_i의 아이디 리스트를 TSLID(i)라고 한다. 즉 TSLID(i) = $<ID_1,ID_2, ID_3,...,ID_q>$이다. 이때 전체 갱신된 아이디 리스트를 IS라고 하며 다음과 같이 정의한다. IS = <TSLID(1), TSLID(2), ..., TSLID(N)>. 이때 N은 TSLID의 수를 말한다.

정의 3.3. TT는 F의 fanout과 D의 depth를 갖는다. TT의 노드는 포인터와 타임스탬프 값으로 구성된다. 따라서 노드의 i번째 포인터를 P_i라고하며 그 노드의 k번째 타임스탬프를 TS_k라고 한다($1 \le i \le F$, $1 \le k < F$). 이때 레벨이 L이며 트리의 왼쪽에서 N번째인 노드는 다음과 같이 정의 된다. NODE(L,N) = $<P_1,TS_1,P_2,TS_2,P_3,..., TS_{F-1},P_F>$. ($1 \le L \le D$, $1 \le N \le F^{L-1}$) 이때 $TS_1<TS_2<...<TS_k$를 만족한다. 또한 P_i는 다음과 같이 결정된다.

1. NODE(L,N)이 내부 노드일 경우 P_i는 NODE(L+1, (N-1)*F+i)를 가리키고 있다.
2. NODE(L,N)이 리프 노드일 경우 P_i는 다음과 같은

표 2 ULIST

Data ID	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	d12	d13	d14	d15	d16	d17	d18	d19	d20
Timestamp	24	16	10	6	22	18	26	32	2	20	14	30	8	4	12	28	8	14	24	20

표 3 OLIST

Timestamp	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
Data ID	d9	d14	d4	d13, d17	d3	d15	d11, d18	d2	d6	d10, d20	d5	d1, d19	d7	d16	d12	d8

성질을 만족한다.

- i) N=1, i=1 일 때, P1은 IS의 첫 번째 아이디를 가리키다.
- ii) N>1, i=1 일 때, 부모 노드가 NODE(L,N)을 가리키는 포인터를 P_j라고 할 때 P_1은 부모 노드의 TS_{j-1}에 갱신된 데이타의 아이디를 가리킨다. 즉 TS_{i-1}이 OLIST의 k번째 TSL에 속할 때 P_1은 IS의 TSLID(k)의 첫 번째 아이디를 가리킨다.
- iii) N≥1, i>1일 때, P_i는 TS_{i-1}에 갱신된 아이디 리스트를 가리키게 된다. 즉 i-1번째 타임스탬프인 TS_{i-1}은 OLIST의 j번째 TSL에 속한다고 한다. 이때 P_i는 IS의 TSLID(j)의 첫 번째 아이디를 가리킨다.

그림 5 트리의 NODE(2,1)은 <P_1,2,P_2>로 이루어지게 되며 이 노드가 내부 노드일 경우 P_1은 NODE(3,1)을 P_2는 NODE(3,2)을 가리키게 된다. 리프 노드일 경우는 다음의 예제에 포함된다.

TTCI의 구성 알고리즘은 네 단계를 이용한다. 첫 번째 단계에서는 정의 3.1을 이용하여 OLIST를 생성한다. OLIST는 갱신된 데이타를 타임스탬프를 기준으로 정렬하여 구성된다. 이렇게 OLIST는 정의에서 밝힌 바와

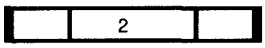

그림 6 정의 3.3 예

같이 중복된 타임스탬프를 하나로 정렬하여 아이디 리스트를 갖게 한다. 두 번째 단계는 이러한 OLIST를 이용하여 TT를 구성한다. TT를 구성하기 위해 TT에서 사용할 타임스탬프의 수인 NTS와 fanout인 FA를 이용하여 TT의 depth인 DE를 구하게 된다. 이렇게 구해진 DE를 이용하여 TT를 구성한다. TT의 구성은 FA 크기로 나누어 경계가 되는 값을 취하는 방식이다. OLIST를 FA로 등분한 값을 gap이라고 하고 이를 이용하여 각 노드에 사용할 타임스탬프를 구하게 된다. 세 번째 단계에서 IS를 구성하고 네 번째 단계에서 TT와 결합하게 된다. IS는 갱신된 데이타의 아이디 리스트를 말하는데 첫 번째 단계에서 구성한 OLIST의 아이디를 순서대로 나열하게 된다. 이렇게 구성된 IS를 정의 3.3에서 정의한 방식대로 노드의 포인터를 결합하게 되면 TTCI 기법이 완성된다. 자세한 구성은 그림 7과 같다.

이와 같은 과정을 거쳐서 TTCI를 구성하는데. 간단

```
begin
/* INPUT :
   ND = IR의 윈도우 사이즈 동안에 갱신된 데이터의 수
   ULIST = { <D1,TS1>, <D2,TS2>,....,<DND,TSND>} D는 데이터의 아이디를 TS는 타
임스탬프를 나타낸다.
   NT = OLIST의 원소의 개수. Step1에 의해 결정된다.
   FA = TTCI의 fanout, NTS = TTCI에서 사용되는 타임스탬프의 수.
OUTPUT :
   TTCI 구조 */
Step 1: OLIST를 생성한다.
for (i=1; i≤ND; i++) ULIST를 이용하여 <정의 3.1>의 TSi를 구한다.
이때 OLIST는 NT개의 원소를 갖게 된다.
Step 2: TTCI의 TT를 생성한다.
level=1; gap=NT; DE = ⌈logFANTS⌉; /*TTCI의 depth*/
while (level≤DE) {
        node=1; tmax=gap; tmin=1; cluster = gap;
        while (node≤FA^(level-1)) {
                gap = ⌈(t max − t min)/FA⌉;
              /*<정의 3.3>를 이용하여 NODE(level,node)의 TS를 구한다*/
                for ( i=1; i<FA; i++)
                TSi = OLIST의 (gap * i + tmin)번째 TSL의 타임스탬프;
                tmin = tmax+ 1;
                tmax = tmax + cluster - 1;
                if ( tmax > NT ) tmax = NT;
                node++; }
        level++; }
Step 3: TTCI의 IS를 생성한다.
for (i=1; i≤NT; i++) <정의 3.2>의 TSLID(i)를 구성한다.
Step 4: TT의 포인터를 연결한다.
for ( TT의 모든 노드 ) <정의 3.3>을 이용해 노드의 포인터를 구한다.
end
```

그림 7 TTCI 구성 알고리즘

한 예를 통해 TTCI의 구성절차를 살펴본다. 갱신된 데이타는 20개이고 데이타의 ID와 타임스탬프에 대한 정보는 표 2와 같은 ULIST에 저장되어 있다. 이때 TT의 fanout은 3, TT에서 이용하는 타임스탬프의 수인 NTS는 8로 지정한다.

Step1을 이용하여 OLIST를 구한다. OLIST는 정의한 바와 같이 타임스탬프의 오름차순 정렬을 따르고 있다. 표 2의 ULIST를 이용하여 표 3과 같은 OLIST를 구할 수 있다. 이때 OLIST의 원소의 개수인 NT는 16이 된다.

Step2를 이용하여 TT를 생성한다. 우선 level = 1, node = 1, tmin = 1, tmax = 16, cluster = 16으로 설정한다. 첫 번째 루프에서 $gap = \left\lceil \frac{16-1}{3} \right\rceil = 5$이다. 즉 16개의 OLIST의 타임스탬프를 3등분하면 간격이 5가 된다. 따라서 NODE(1,1)은 <P_1,TS_1,P_2,TS_2,P_3>로 구성되어 있고 TS_1은 TSLT(5*1+1)=12, TS_2는 TSLT(5*2+1)=22로 구해진다.

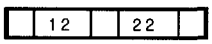

그림 8 NODE(1,1)

node는 2가 되어 루프의 조건을 만족하지 않게 되므로 level = 2, node = 1, t_{min} = 1, t_{max} = 5, cluster = 5로 설정된다. 또한 $gap = \left\lceil \frac{5-1}{3} \right\rceil = 2$이다. 따라서 NODE(2,1)은 <$P_1$,$TS_1$,$P_2$,$TS_2$,$P_3$>로 구성되어 있고 TS_1은 TSLT(2*1+1)=6, TS_2는 TSLT(2*2+1)=10으로 구해진다.

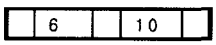

그림 9 NODE(2,1)

NODE(2,2)와 NODE(2,3)도 위와 같은 방법을 이용하므로 자세한 계산 과정은 생략한다. 반복된 과정을 통해 각 노드에 적용될 타임스탬프의 값을 구하게 된다.

Step3을 이용하여 IS를 구성한다. IS는 OLIST를 이용하여 데이타 ID를 정렬된 순서대로 리스트를 구성하면 된다. 아래의 그림에 IS의 구조가 나타나 있다.

Step4의 과정을 통해 노드의 포인터가 가리키는 지점을 정한다. NODE(1,1)은 내부노드 이므로 정의 3.3의 성질 1에 의해 P_1은 NODE(2,1)을 P_2는 NODE(2,2)을 P_3는 NODE(2,3)을 가리킨다. 또한 NODE(2,1)은 리프노드이므로 P_1은 IS의 첫 번째 데이타 ID인 d9을 가리키게 되고 P_2는 TS_1인 6에서 갱신된 TSLID인 d4를 가리키게 된다. 또한 P_3는 TS_2인 10에서 갱신된 TSLID d3을 가리키게 된다. NODE(2,2)의 P_1은 부모노드의 포인터가 P_2이므로 부모노드의 TS_1인 12일 때 갱신된 TSLID인 d15를 가리키게 된다. 나머지 경우의 포인터는 앞서 소개한 NODE(2,1)의 내용과 중복되므로 생략한다. 최종결과는 그림 10과 같다.

이와 같이 구성된 트리는 브로드캐스트 채널을 통해 전달 할 수 있다. 즉 TT 부분을 선행해서 보내고 후에 IS 부분을 브로드캐스트 함으로써 하나의 IR을 전달 할 수 있다. 또한 TT에서 사용되는 노드의 포인터는 다음 노드 혹은 IS가 전달될 시간 값을 이야기한다.

3.4 수신 알고리즘

서버 측에서 구성한 TTCI 기법의 IR을 클라이언트가 전송 받게 된다. TTCI는 윈도우내의 정보를 나타내고 있다. 따라서 마지막으로 무효화를 확인한 시간이 윈도우 범위에 있는지를 확인하게 된다. 그 이후 TTCI의 TT를 이용하여 마지막으로 무효화를 확인한 시점을 기준으로 탐색하는 과정을 거치게 된다. 이러한 과정으로 얻게 되는 IS의 포인터를 이용하여 클라이언트가 마지막으로 무효화를 확인한 시간 이후에 갱신된 정보를 수신하게 된다. 자세한 내용은 다음과 같은 알고리즘을 이용한다.

위와 같은 방법을 이용하면 클라이언트는 선택적 청취가 가능해진다. 즉 최근에 데이타의 유효성을 검증한 시간을 기준으로 TT를 검색하게 된다. 이때 트리의 구조상 자신이 원하는 노드를 선택 청취하는 것이 가능해진다. 또한 TT를 통해서 결정된 포인트를 이용하여 IS 중 자신이 수신해야할 정보만을 듣게 된다.

앞 절에서 소개한 예를 이용해 본 절에서는 클라이언트가 청취하는 방법을 살펴본다. 이때 클라이언트의 캐시 상태는 다음 같다. 또한 클라이언트가 마지막으로 유

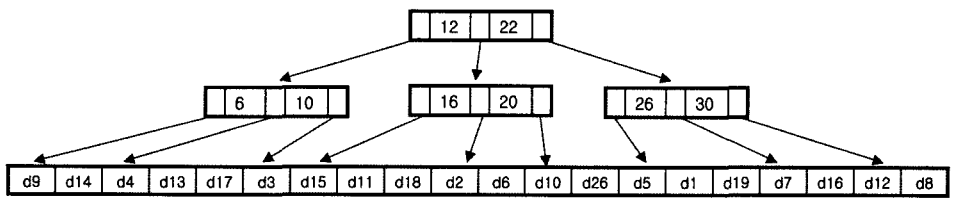

그림 10 TTCI 예제

```
begin
/* INPUT :
   T = 현재 리포트의 타임스탬프
   T^C=클라이언트 캐시의 마지막으로 유효함을 확인한 타임스탬프
   L = TTCI의 방송 주기, w = TTCI의 윈도우 크기
   C = 현재 캐시의 데이터 아이디 집합
OUTPUT :
무효화된 데이터가 제거된 캐시*/
if (T - T^C > wL) {
        캐시의 모든 데이터를 제거;
}
else{
      /* TT 수신 */
      while (true){
        현재 리포트의 타임스탬프 T를 기준으로 TT의 노드를 탐색;
        if ( 검색된 노드가 IS의 노드 ){
                po = 현재 탐색된 포인터;
                break;}
      }
      /* IS 수신 */
      I = po부터 수신한 IS의 데이터 아이디 집합;
      for ( d ∈ I 인 모든 원소 ) {
                if ( d ∈ C )
                C에서 d에 해당하는 아이디와 값 제거;
}}
T^C = T;
end
```

그림 11 TTCI 수신 알고리즘

	d2	d3	d4	d7	d8	d9	d10	d11	d13	d16
17	10203	20404	29289	39492	39485	72050	49483	39304	38204	25603

그림 12 캐시의 현재 상태

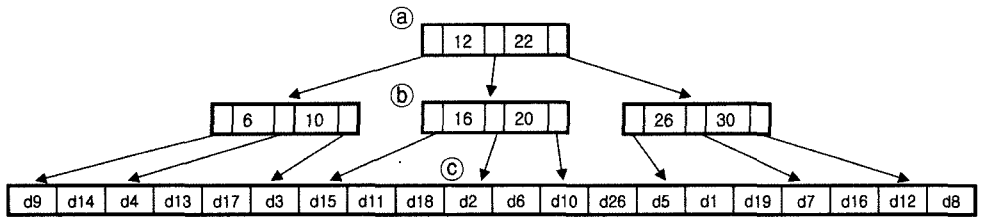

그림 13 클라이언트가 청취한 TTCI

효화를 확인한 시간인 T^C는 17이고 현재 TTCI가 전송된 시간인 T는 32라고 가정한다. 또한 IR 방송주기인 w는 2이고 윈도우 크기인 L은 5라고 가정한다.

알고리즘에 의해 전송된 IR의 윈도우 크기 내에 포함되어있는지 확인한다. 본 예제 에서는 w가 2이고 L이 5이므로 전송된 IR로 무효화를 확인 할 수 있다. 따라서 TT부분을 청취하게 된다. 클라이언트는 TT의 루트에 해당하는 노드를 전송 받게 된다. 이때 전송된 노드는 그림 13의 노드 ⓐ와 같고 이를 가장 최근에 확인된 시간인 17을 기준으로 다음에 전송될 포인터를 찾는다. 즉 17은 12보다 크고 22보다 작으므로 가운데 존재하는 포인터를 참조하게 되고 대기모드로 전환하게 된다. 포인터가 지시한 시간에 활동모드로 전환하게 되고 ⓑ와 같은 노드를 수신하게 된다. 현재 수신 받은 노드는 IS가 아니므로 위의 과정을 반복하게 된다. 수신한 노드에 의해 16보다 크고 20보다 작은 17은 중간에 위치한 포인터를 선택하게 된다. 선택한 노드가 IS의 노드이므로 po에 현재 청취해야할 포인터를 저장하고 루프를 벗어나

32	d3	d4	d9	d11	d13					
	20404	29289	72050	39304	38204					

그림 14 IR 전송에 따른 캐시의 상태

게 된다. 포인터가 지정한 시점 부터 청취하게 되는 IS의 아이디를 I라고 하며 그 리스트의 내용은 그림 13의 ⓒ와 같다. 이를 이용해 현재 캐시의 무효화된 데이타를 제거하게 된다. I의 원소를 d라고 할 때 d와 현재 캐시에 저장되어 있는 데이타 ID를 비교하여 중복되는 경우 제거한다. 예를 들어 I의 원소인 d2는 현재 캐시에 저장되어 있으므로 제거 된다. 현재 수신한 TTCI의 정보를 표시해보면 다음과 같고 색깔이 칠해진 부분이 클라이언트가 수신한 데이타가 된다.

IS의 값을 이용하여 캐시의 무효한 값들을 제거하면 다음과 같은 캐시의 상태로 변경이 된다. 이때 T^C의 값은 T값으로 대체되어 32가 된다. 즉 마지막으로 클라이언트 캐시의 유효화를 확인한 시간인 32를 저장하게 된다.

위 그림 14와 같이 d2, d7, d8, d10, d16의 데이타가 무효화 되면서 캐시에서 정보가 사라진 것을 알 수 있다.

4. 성능 비교

본 장에서는 몇 가지 실험을 통해 본 논문에서 제시하는 무효화 보고 기법(Invalidation Report, IR)방법이 효과적임을 보인다. 앞 장에서 기술하였던 바와 같이 TTCI(Timestamp Tree-based Cache Invalidation scheme) 기법은 많은 양의 데이타가 존재하는 경우와 갱신이 빈번한 경우에 효율성을 높이기 위한 방법이다. 따라서 이러한 환경에서 얼마나 효과적인지를 실험을 통해 비교한다.

실험을 위해서 두 가지 비교요소를 이용한다. 첫 번째는 캐시 이용률(Cache Usage)을 통해 이루어진다. IR의 목적은 캐시에 저장되어 있는 데이타의 이용률을 높이는데 그 목적이 있다. 즉 캐시에 저장되어 있는 데이타가 유효한지 확인하여 저장된 데이타를 최대한 이용하게 하는 것이 IR의 목적이다. 따라서 IR의 성능은 캐시의 이용률로 확인될 수 있다. 즉 캐시에 저장된 데이타는 IR에 의해 무효화를 검사하여 유효하지 않은 데이타는 제거된다. 이때 IR의 성능에 따라 캐시에 유효하게 남아 있는 데이타의 수가 다르게 된다. 이를 Cache Usage Ratio 라고 하며 다음과 같이 계산된다. 이때 NumofData는 캐시에 저장될 수 있는 전체 데이타의 수를 말하며 NumOfValidData는 IR을 통해 유효하지 않은 데이타가 제거된 후 현재 캐시에 저장된 유효한 데이타의 수를 말한다.

$$Cache\ \ Usage\ \ Ratio = \frac{NmCfValidData}{NmCfCachedData} \times 100$$

두 번째 비교 요소로 IR의 길이를 이용한다. 모바일 환경에서 대역폭은 유선환경에 비해 고비용의 자원이다. 따라서 주기적으로 전송되는 IR의 길이는 대역폭의 소비와 직결된다. 또한 Cache-Level의 IR 길이는 클라이언트의 IR의 정보를 확인하기 위한 Tuning Time과도 직접적인 연관이 있게 된다. 따라서 모바일 클라이언트의 Tuning Time은 배터리의 이용을 의미하게 되고 이는 모바일 기계의 한정된 자원인 전력소비를 의미한다. 이러한 이유로 여기서는 IR의 길이를 성능 판단의 다른 기준으로 삼는다.

본 장에서 이용하는 대조군은 DRCI(Dual-Report Cache Invalidation)[7]이다. 이 IR 구성 방식은 기본적인 타임스탬프구조에 그룹을 이용하여 IR의 효율성을 유도하는 방법이다. 이러한 방법은 갱신된 데이타의 정보를 모두 포함함으로써 단절이 되지 않는 상황에서는 최상의 캐시 효율성 제공한다. 이러한 DRCI는 본 논문에서 제안하는 TTCI와 같은 Cache-Level의 IR로써 TTCI의 효율성을 확인하기에 적합하다. 또한 BS(Bit Sequence)[10]에 비해 DRCI가 효과적이라는 것은 증명된바있다[7]. 따라서 본 논문에서 제안한 IR의 특성을 비교하기 위해 DRCI를 실험 대조군으로 선택한다. 또한 IR의 기본적인 기법인 TS(Broadcasting Timestamp)를 이용하여 캐시의 재사용성과 길이를 비교한다[6].

4.1 실험 환경

본장에서 이용하는 시스템의 구성은 다음과 같다. 클라이언트는 데이타의 요구와 브로드 캐스트된 데이타를 캐시로 저장한다. 또한 서버 측은 데이타의 갱신과 브로드캐스트를 하게 된다. 이러한 모바일 환경의 서버와 클라이언트를 가상화하여 구성하였다. 클라이언트는 서버로 데이타를 요청하게 된다. 서버는 브로드캐스트 주기 안에 갱신된 데이타를 이용해 IR을 일정간격으로 전송하게 된다. 클라이언트는 이러한 IR을 기준으로 캐시의 무효화된 데이타를 판단하여 제거하게 된다. 또한 일정간격으로 갱신된 데이타를 전송한다. 다음 그림의 번호와 같은 순서가 반복해서 일어나게 된다. 미리 정의된 비율을 이용하여 클라이언트는 네트워크 단절을 수행한다. 이때 클라이언트는 IR을 전송 받지 못하게 되고 이후에 받게 되는 IR을 이용하여 캐시의 무효화된 데이타를 판단하게 된다.

실험을 위해 다음 표 4와 같은 환경 변수를 정의한다. 실험에서 클라이언트의 접근 경향과 서버의 전체 데

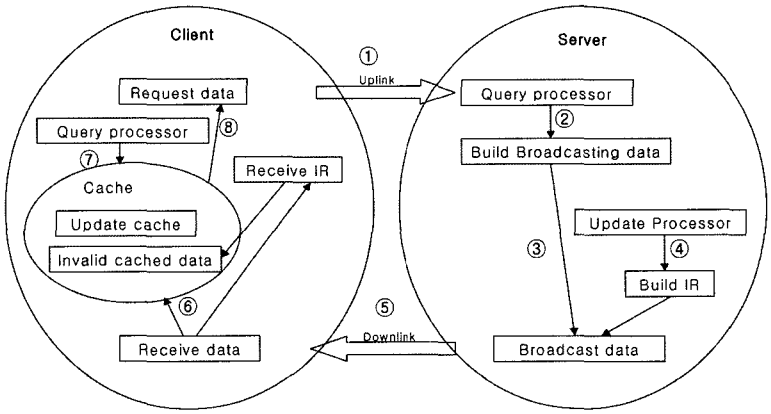

그림 15 실험 구성 모델

표 4 실험 환경 변수 정의

항 목	정 의	기 본 값
공통으로 사용되는 변수		
Total_Data	전체 데이타 수	1000 obj
ID_Size	데이타의 ID의 크기	32 bits
Timestamp_Size	타임스탬프의 크기	64 bits
NumOfCache	캐시에 저장될 수 있는 데이타 수	100 obj
IR_Broadcast_Term	IR이 방송되는 주기	50 sec
Disconnect_Freq_R	클라이언트가 연결 단절되는 빈도	0.2
Update_Ratio	갱신된 데이타 수 / 전체 데이타 수	0.2
Skewed_Update_R	갱신되는 데이타의 편향정도	0.9
Skewed_Demand_R	요구되는 데이타의 편향정도	0.9
NumOfClient	클라이언트의 수	100 obj
TTCI에서 사용되는 변수		
TTCI_Ws	TTCI의 IR Window 크기	5
Point	TT의 구성에 필요한 포인터의 크기	16 bits
TTCI_TS	TTCI에서 사용된 타임스탬프의 수	82 obj
Fanout	TTCI에서 사용된 Fanout의 크기	3
DRCI에서 사용되는 변수		
DRCI_Group	DRCI에서 GIR의 Group수	100 obj
GIR_Ws	DRCI의 GIR Window 크기	5
OIR_Ws	DRCI의 OIR Window 크기	2
TS에서 사용되는 변수		
TS_Ws	TS Window 크기	2

이타의 갱신 경향을 모델링 하기 위해 Zipf distribution을 이용하였다[13-15]. zipf distribution은 균일하지 않은(non-uniform) 접근 경향을 모델링 하기 위해 많이 사용되는 분포로서 브로드 캐스트에 사용될 데이타를 온도에 따라 정렬했을 때 θ값에 따라 온도가 높은 데이타가 생성될 확률이 온도가 낮은 데이타가 생성될 확률보다 높게 되는 분포를 보인다.

4.2 갱신되는 데이타의 증가에 따른 캐시의 효율성 비교

본 논문에서 제안하는 방법은 갱신되는 데이타 수의 증가에 유연하게 대처하기 위해 고안되었다. 따라서 갱신되는 데이타의 양의 증가에 따른 캐시의 효율성을 비교해본다. 즉 기존이 기법보다 데이타의 양의 증가해도 효과적임을 보인다.

실험을 위해 위와 같은 환경을 설정하게 된다. 즉 갱

표 5 갱신 빈도에 따른 성능비교를 위한 환경 변수 설정

항 목	내 용	
	실험 1	실험 2
Total_Data	1,000	10,000
NumOfCache	100	1,000
DRCI_Group	100	1,000
TTCI_TS	82	754
Update_Ratio	0 - 0.5	

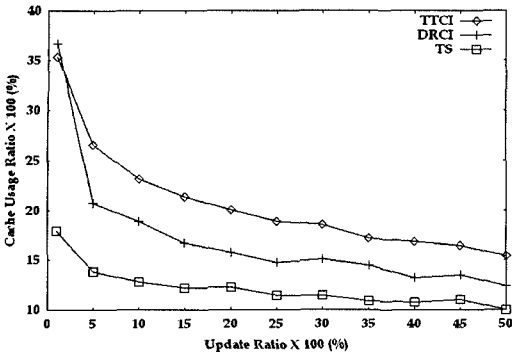

그림 16 1,000일 때 캐시의 이용률

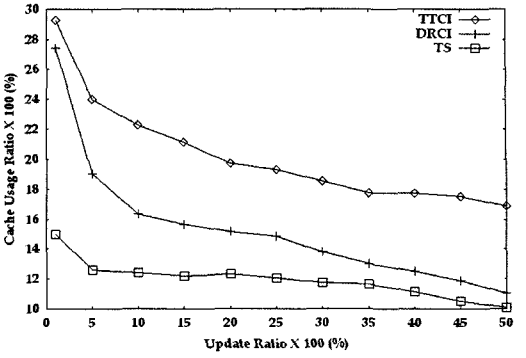

그림 17 10,000일 때 캐시의 이용률

신되는 데이타의 수를 증가시킬 때 단절 빈도에 따른 캐시의 이용률을 알아본다. 실험은 갱신 빈도에 따라 Zip Factor를 이용하여 데이타를 갱신한다. 이러한 데이타를 IR에 반영하게 된다. 이때 IR을 통해 캐시의 무효화된 데이타를 제외한 캐시의 이용률을 확인한다.

상기의 그림과 같이 데이타의 갱신되는 확률이 많아짐에 따라 TTCI와 DRCI 모두 캐시의 이용률이 떨어지는 것을 확인 할 수 있다. 하지만 DRCI에 비해 TTCI는 갱신되는 비율이 50%에 이르러도 일정 수준의 캐시 이용률을 보이는 것을 알 수 있다. 즉 갱신 빈도가 증가하여도 TTCI가 보다 완만한 기울기로 캐시 이용률을 감소시키는 것을 확인 할 수 있다. 결론적으로 TTCI가 갱신되는 데이타의 양에 보다 효과적으로 대처한다고 판단할 수 있다. 또한 TS는 DRCI의 OIR(Object Invalidation Report)과 같은 정보를 전송하므로 DRCI의 OIR 정보에 해당하는 캐시 이용률을 보여주고 있다.

데이타의 수가 1,000인 경우, 데이타의 갱신 빈도가 낮을 경우 DRCI의 성능이 일시적으로 좋게 나오는 점을 볼 수 있다. 이는 갱신되는 데이타의 수가 적을 경우 클라이언트는 DRCI의 OIR 정보만을 이용하므로 정확한 갱신 정보를 제공할 수 있기 때문이다. 반면 GIR (Group Invalidation Report)은 하나의 그룹에 대해 마지막으로 갱신된 데이타에 대한 타임스탬프 정보만을 표현하고 있다. 따라서 갱신 빈도가 증가하게 되었을 경우 하나의 그룹이 마지막으로 갱신된 타임스탬프 값에 모두 의존하게 되므로 TTCI가 DRCI보다 20%정도 효과적인 성능을 갖는 것을 알 수 있다.

데이타의 수가 10,000개일 때도 이와 유사한 결과를 얻고 있다. 이는 데이타의 수에 비례하여 캐시의 크기를 증가하였기 때문이다. 하지만 10,000개의 데이타를 환경으로 한 실험에서는 갱신되는 비율이 낮아도 IR에 표현되는 데이타의 양이 많게 되므로 TTCI가 항상 좋은 성능을 보임을 알 수 있다. 평균적으로 20%의 성능향상을 볼 수 있다.

4.3 갱신되는 데이타의 증가에 따른 IR의 크기 비교

모바일 환경에서 중요한 자원인 대역폭의 소비와 직결되는 IR의 크기를 비교함으로써 성능을 비교해본다. 본 논문에서 제안한 기법인 TTCI는 방송되는 IR의 길이보다 적은 양의 Tuning Time을 요구하고 있다. 하지만 기존 Cache-Level의 기법은 IR의 길이만큼의 Tuning Time을 요구한다. 따라서 본 절에서는 전체 IR의 길이를 비교하여 Tuning Time의 성능을 예측하고자 한다. 따라서 다음과 같은 실험 환경을 이용하여 DRCI와 비교한다.

표 6 IR의 길이 비교를 위한 환경 설정

항 목	내 용	
	실험 3	실험 4
Total_Data	1,000	10,000
NumOfCache	100	1,000
DRCI_Group	100	1,000
TTCI_TS	82	754
Update_Ratio	0 ~ 0.5	

그래프에서 보는 바와 같이 갱신되는 데이타의 수가 증가함에 따라 IR의 길이가 커지는 것을 알 수 있다. 즉 표현되어야할 데이타의 수가 증가함에 따라 DRCI는 OIR의 길이가 증가하게 되고 TTCI와 같은 경우 IS의 길이가 증가하게 된다. DRCI와 같은 경우 OIR에 갱신된 데이타의 ID 정보와 타임스탬프 정보가 포함되게 되고, TTCI와 같은 경우 데이타의 ID정보가 추가 되게 된다. 따라서 갱신정보가 증가하게 되면 그 길이에 비례하여 선형적인 증가를 가지게 됨을 예상할 수 있다. 하지만 갱신되는 분포가 고르지 않은 실험환경을 가정하기 때문에 중복하여 갱신되는 데이타의 증가에 따른 IR의 길이의 증가가 위와 같이 곡선을 그리게 된다. 또한 TS의 크기는 서버 측의 갱신 빈도에 비례하여 증가하

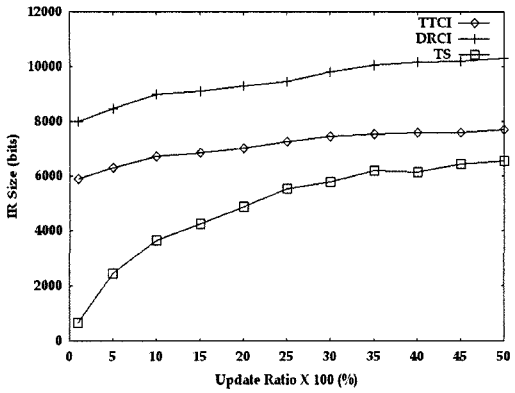

그림 18 1,000일 때 IR의 길이

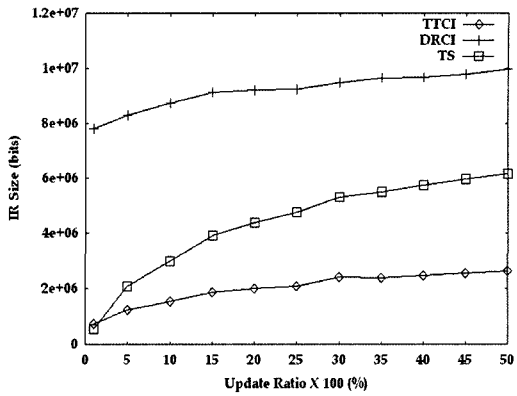

그림 19 10,000일 때 IR의 길이

는 것을 확인 할 수 있다. 데이타의 수가 1,000개일 때와 10,000개일 때를 비교해 보면 데이타의 양이 많을수록 증가하는 비의 차가 커지는 것을 확인 할 수 있다. 1,000개의 데이타를 이용할 때는 DRCI보다 25%이상의 길이 감소를 확인할 수 있었고 10,000개의 데이타를 이용할 때는 DRCI의 25%정도의 길이로 IR을 구성할 수 있었다. 또한 DRCI는 전체 IR을 들어야 하므로 TTCI가 Tuning Time에서 더 좋은 성능을 보일 것은 쉽게 예측할 수 있다.

4.4 IR의 방송 간격에 따른 캐시의 효율성 비교

모바일 환경에서 단절은 일반적인 환경 변수로 고려된다. 즉 불안정한 네트워크 환경은 모바일을 이용한 통신에서 통상적인 특성이다. 따라서 본 절에서는 IR의 방송 간격이 증가함에 따른 IR의 성능을 비교해 보고자 한다. 실험 환경은 다음과 같다.

표 7 IR의 방송 간격에 따른 성능비교를 위한 환경 설정

항 목	내 용
IR_Broadcast_Term	0 - 120 sec

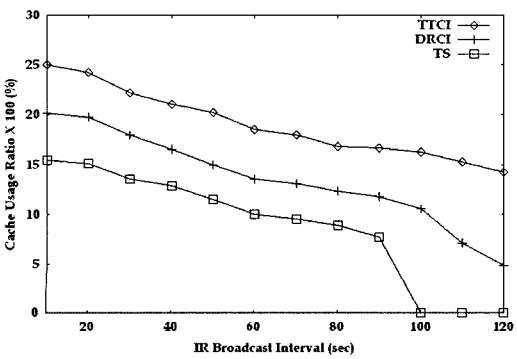

그림 20 IR의 방송 주기에 따른 캐시의 이용률

그림에서 알 수 있듯이 IR의 방송 주기가 커짐에 따라 TTCI와 DRCI 모두 캐시 이용률이 감소하는 것을 알 수 있다. 이때 전반적으로 DRCI에 비해 25%정도의 성능향상을 기대할 수 있다. 또한 윈도우 크기인 120에 근접한 IR 방송주기를 가져도 TTCI 방법을 이용하면 일정 비율의 캐시 이용률을 유지함을 알 수 있다. 반면 DRCI와 같은 경우 OIR 윈도우 크기인 100을 넘어설 경우 GIR의 의존도가 커지게 되어 캐시 이용률의 급격한 감소를 가져오게 된다. 또한 윈도우 크기가 100을 넘게 되면 TS는 IR을 이용할 수 없게 되므로 캐시 이용률이 0이 된다. 이러한 성능은 TTCI의 타임스탬프를 기반으로 한 동적인 그룹화 기법이 이와 같은 캐시의 이용률을 가져오게 하는 것을 예측할 수 있다. 즉 전체 윈도우 크기에 균등하게 정보를 배분함으로써 IR을 듣는 시점에 덜 영향을 받기 때문이다. 이러한 점은 DRCI의 OIR과 GIR의 차등적인 윈도우 크기와 비교되는 부분이다.

4.5 편향된 데이타의 갱신 및 요청에 따른 캐시의 효율성 비교

모바일 환경에서 데이타의 갱신과 요구는 모든 데이타에 골고루 분포되지 않는다. 즉 특정 데이타에 집중되는 현상을 갖게 된다. 이렇게 데이타가 집중되는 부분을 Hot data set이라고 하며 그렇지 않은 부분을 Cold data set이라고 한다. 이러한 Hot data set의 존재는 모바일 데이타의 일반적인 패턴이다. Skewed_Update_P는 서버 측의 갱신되는 편향성을 나타낸다. 또한 Skewed_Demand_P는 클라이언트 측의 캐시되는 데이타의 편향성을 말한다. 따라서 여기서는 Hot한 정도에 따른 캐시의 성능을 비교하고자 한다. 데이타 갱신의 편향성의 실험 시 캐시는 50%의 편향되지 않은 성향을 유지한다. 데이타의 요구의 편향성을 실험할 경우도 갱신의 편향성을 배제한다.

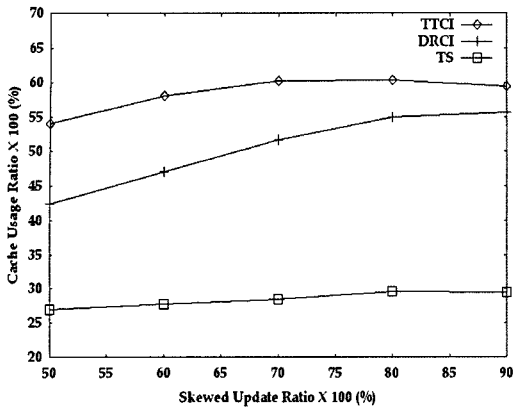

그림 21 편향된 갱신 시 캐시의 이용률

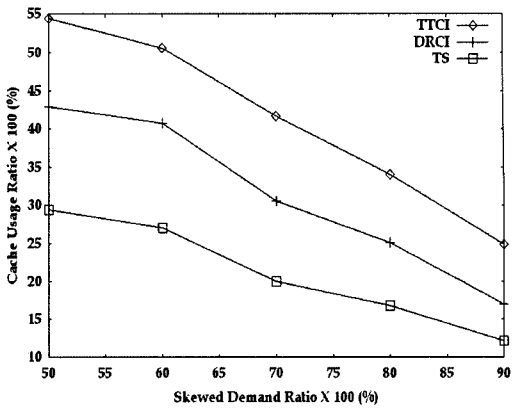

그림 22 편향된 요구 시 캐시의 이용률

표 8 편향된 데이타의 성능비교를 위한 환경 설정

항 목	내 용	
	실험 6	실험 7
Skewed_Update_R	0.5 - 0.9	0.5
Skewed_Demand_R	0.5	0.5 - 0.9

실험결과에서 알 수 있듯이 TTCI가 더 좋은 성능을 보임을 알 수 있다. 데이타의 갱신이 편향되어질 경우 TTCI와 DRCI 모두 갱신된 데이타의 양이 감소하므로 성능의 향상을 기대할 수 있다. 하지만 전반적으로 TTCI의 편향된 갱신에 대해 20%정도의 성능 향상을 기대할 수 있다. 또한 클라이언트의 접근 패턴이 치우친 경우에도 TTCI가 20% 정도 성능의 향상을 확인 할 수 있다.

5. 결 론

본 논문에서는 기존의 무효화 보고 기법(Invalidation Report, IR)들이 가지고 있던 문제점을 개선하여 새로운 IR기법을 제안하였다. 기존의 IR기법들은 데이타의 수가 증가함에 따라 IR크기가 증가하고 이와 더불어 IR의 성능이 감소하게 된다. 또한 데이타의 갱신 횟수가 증가할 때에도 이러한 현상은 똑같이 발생하게 된다. 이는 IR에 포함되어야할 데이타의 양이 증가함에 따라 미리 정의된 그룹 방법을 이용함으로써 오는 현상이다. 즉 기존의 IR은 미리 결정된 구조를 이용하여 포함되어야할 데이타의 양에 능동적으로 대처하지 못하기 때문이다. 또한 기존의 IR기법에서는 Cache-Level의 선택적 청취(Selective tuning)를 지원하는 기법을 찾기 어렵다. 기존 기법에서는 선택적 청취를 지원하기 위해 Query-Level의 기법을 이용하고 있다.

따라서 본 논문에서는 이러한 단점을 보안하고 효과적인 IR을 구성하기 위해 TTCI(Timestamp Tree-based Cache Invalidation scheme)를 제안하였다. 이러한 TTCI는 타임스탬프 트리를 이용하여 갱신된 데이타의 아이디를 동적으로 그룹화 하고 있다. 또한 이러한 타임스탬프 트리의 수를 갱신된 수에 비해 적게 함으로써 IR길이를 줄일 수 있다. 이와 함께 Cache-Level의 선택적 청취를 가능하게 함으로써 캐시 전체의 내용의 무효성을 검증함과 동시에 선택적 청취를 가능하게 하고 있다.

또한 본 논문에서 제안한 TTCI의 성능은 데이타의 갱신되는 양이 증가할 때 Cache Usage Percentage와 IR의 크기로서 비교 될 수 있다. 즉 IR의 효율성이라고 말할 수 있는 캐시의 사용율과 IR의 크기를 비교했을 경우 대조군으로 선택한 DRCI에 비해 25%의 크기감소와 20%의 캐시 이용률의 증가를 얻을 수 있었다. 이것은 타임스탬프를 이용한 동적인 구조를 이용하게 됨으로써 오는 현상이다. 또한 IR의 방송 간격이 커질 때 즉 단절(Disconnection)로 인한 IR의 수신 간격이 커질 때도 25%정도의 성능 향상을 알 수 있었다. 이는 IR 윈도우 전반에 걸쳐 갱신된 정보를 편향되지 않게 IR을 구성함으로써 얻을 수 있는 결과이다. 마지막 실험은 모바일 환경의 특성이라 할 수 있는 편향된 갱신, 요구 패턴에서 TTCI의 성능을 살펴보았다. 이 경우 서버 측의 데이타 갱신이 편향되었을 경우와 클라이언트의 접근 패턴이 편향되었을 경우 모두에서 TTCI가 20%의 성능 향상을 기대할 수 있었다. 이는 갱신 정보를 타임스탬프를 기반으로 구성하기 때문이며 TTCI가 편향성 자체에 영향을 덜 받는 구조임을 확인 할 수 있었다.

결과적으로 기존의 기법에 비해 TTCI는 IR의 크기와 캐시의 재사용 측면에서 좋은 성능을 보였다. 특히 많은 데이타의 빈번한 갱신과 클라이언트의 다양한 접근 패턴 그리고 연결 단절에 좀더 효과적으로 대처하는 것을 확인할 수 있었다.

참 고 문 헌

[1] T. Imielinski and B.R. Badrinath, "Data Management for Mobile Computing," SIGMOD RECORD, vol.22, no.1, pp. 34-39, (1993).

[2] T. Imielinski and B.R. Badrinath, "Wireless Mobile Computing: Challenges in Data Management," Comm. ACM, vol.37, no.10, pp. 18-28, (1994).

[3] J. Jing, A. Helal, and A. Elmagarmid, "Client-Server Computing in Mobile Environments," ACM Computing Surveys, vol.31, no.2, pp. 117-157, (1999).

[4] K-L. Tan and B.C. Ooi, Data Dissemination in Wireless Computing Environments, Kluwer Academic, 2000.

[5] G.H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," IEEE Computer, vol.27, no.6, pp. 38-47, (1994).

[6] D. Barbara and T. Imielinski, "Sleepers and Workaholics: Caching in Mobile Distributed Environments," Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 1-12, (1994).

[7] K-L. Tan, B.C. Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments," IEEE Transactions on parallel and Distributed systems, vol.12, no.8, pp. 789-807, (2001).

[8] K-L. Wu, P.S. Yun, and M.S. Chen, "Energy-Efficient Caching For Wireless Mobile Computing," Proc. 12th Int'l Conf. Data Eng., pp 336-343, (1996).

[9] J. Cai, K-L Tan, "Energy-Efficient Selective Cache Invalidation," Wireless Networks 5, pp. 489-502, (1999).

[10] J. Jing and A.Elmagarmid and A.Helal and R. Alonso, "Bit-Sequences : An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," ACM Mobile Networks and applications vol 2, pp. 115-127, (1997).

[11] A. Elmagarmid, J. Jing, A. (Sumi) Helal, And C. Lee, "Scalable Cache Invalidation Algorithms for Mobile Data Access," IEEE Transactions on knowledge and data engineering, vol 15, no 6, pp. 1498-1511 (2003).

[12] P.V. Argade, S. Aymeloglu, A.D. Berenbaum, M.V. DePaolis Jr.,R.T. Franzo, R.D. Freeman, D.A. Inglis, G. Komoriya, H. Lee, T.R.Little, G.A. MacDonald, H.R. Mclellan, E.C. Morgan, H.Q. Pham, G.D. Ronkin, R.J. Scavuzzo, and T.J. Woch, "Hobbit: A High-Performance, Low-Power Microprocessor," Proc. COMPCON'93 Int'l Computer Conf., pp. 88-95, (1993).

[13] D. Knuth, The Art of Computer Programming Second Edition, Vol Ⅲ, Addison Wesley, 1998.

[14] G.K Zipf, Human Behaviour and the Principle of Least Effort : An Introduction to Human Ecology, Addison Wesley Press, Cambridge, Massachusetts, 1949.

[15] J. Gray, P. Sundaresan, S. Englert, K. Baclawski and P. J. Weinberger, "Quickly generating billion-record synthetic databases," in ACM SIGMOD Record, Proc. ACM SIGMOD international conference on Management of data, pp. 253-252, (1994).

[16] S. Yi., W. Song, S. Jung, "A Cost Effective Cache Consistency Method for Mobile Clients in Wireless Environments," Database Systems for Advanced Applications, LNCS 2973, pp. 908-915, 2004.

[17] S. Yi., H. Shin, S. Jung, "Enhanced Cost Effective Cache Invalidation for Mobile Clients in Stateless Server Environments," Embedded and Ubiquitous Computing, LNCS 3207, pp. 387-397, 2004.

정 성 원

1988년 서강대학교 전자계산학 학사. 1990년 M.S. in Computer Science at Michigan State Univ. 1995년 Ph.D. in Computer Science at Michigan State Univ. 1997년~2000년 한국전산원 선임연구원. 2000년~현재 서강대학교 컴퓨터공학과 부교수. 관심분야는 Mobile Computing Systems, Mobile Databases, Telematics, Spatial DB, Mobile Agents, Streaming Data Processing in Ubiquitous Computing Environments, Distributed Databases

이 학 주

1996년 3월~2003년 2월 한양대학교 전자.컴퓨터공학부 학사. 2003년 3월~2005년 2월 서강대학교 컴퓨터공학과 석사 2005년 3월~현재 LG전자 DTV연구소 AIM Gr. 주임연구원. 관심분야는 Mobile Database, Mobile Data Caching, Data Carousel / Object Carousel