무선 환경에서의 이동 클라이언트를 위한 효율적인 캐시 일관성 유지 방안

(An Efficient Cache Consistency Method for Mobile Clients in Wireless Environments)

송 원 민 † 정 성 원 ††
(Wonmin Song) (Sungwon Jung)

요 약 최근 이동 컴퓨팅(Mobile Computing)환경에서의 잦은 접속 단절로 인한 클라이언트 캐시 일관성(Consistency) 문제를 해결하기 위한 방법에 대한 연구가 진행되고 있다. 이러한 캐시 유지 방법의 한 분야로서, 무효화 보고(Invalidation Report)에 대한 연구가 진행 중이다. 그러나 기존의 무효화 보고에 대한 연구는 서버에서의 갱신 정도와 상관없이 전송되기 이전 일정 시간동안의 서버 갱신에 대한 정보만을 포함한다. 따라서 기존의 방법은 무효화 보고에 포함되어 있는 정보의 범위 이상의 시간 동안의 클라이언트 접속 단절이 발생할 경우 서버에서 어느 정도의 갱신이 발생하였는지와는 무관하게 클라이언트의 캐시 일관성을 유지하지 못한다. 따라서 본 논문에서는 무효화 보고에 포함되어 있는 정보의 범위를 초과하는 시간 동안 접속 단절된 클라이언트에 대하여 각 클라이언트가 캐시한 데이타 가운데 어느 정도의 데이타가 갱신 되었을지 예측함으로써 캐시 일관성을 유지할 수 있는 효율적인 CCI(Cost-based Cache Invalidation)기법을 제안한다. 또한 기존의 무효화보고 기법인 BT(Broadcasting Time stamp)과 CCI의 성능 비교를 실시하였다.

키워드 : 캐시 일관성, 무효화보고, 방송, 이동 컴퓨팅

Abstract Recently, there have been many research efforts reported in the literature that focus on the cache consistency problems of mobile clients resulting from their frequent disconnections with a server in mobile environments. To cope with this problem, a number of methods based on the invalidation report scheme has been proposed. However, these proposed methods are sensitive to the disconnection time of mobile clients and independent of the frequency of data updates in the server. As a result, although the number of data updated in the server is small, the traditional methods can not guarantee the cache consistency of mobile clients if their disconnection time is over the time period the invalidation report is allowed to cover. In this paper, we propose an efficient cache consistency method called CCI(Cost-based Cache Invalidation) for mobile clients that take into account not only the disconnection time but also the frequencies of data updates in the server. We also present an in-depth experimental analysis by comparing CCI method with BT(Broadcasting Time stamp) method based on Invalidation Report.

Key words : Cache consistency, Invalidation report, Broadcast, Mobile computing

1. 서 론

랩톱(Laptop), 팜톱(Palmtop)과 같은 휴대용 컴퓨터의 등장과 이동전화, 무선LAN, 위성 서비스와 같은 이동 통신기술의 발전은 고유 특성에 따른 상호 요구에 힘입어, 하나의 결합된 형태로 나타나게 되었다. 그 결과 휴대용 컴퓨터를 이용하는 사용자들은 무선 네트워크를 통하여 위치와 상관없이 자유로운 데이타 액세스를 할 수 있게 되었다. 이와 같은 새로운 컴퓨팅 패러다임을 이동 컴퓨팅(Mobile Computing)이라 한다.

이러한 이동 컴퓨팅 환경의 대표적인 특징으로는 이동성과 무선을 생각할 수 있다. 이러한 이동 컴퓨팅의 특징은 많은 사용자들로 하여금 언제 어디서나 원하는 데이타에 접근할 수 있다는 장점을 제공해 주지만 반면

† 비 회 원 : 펜택&큐리텔 연구원
song.wonmin@curitel.com
†† 종신회원 : 서강대학교 컴퓨터학과 교수
jungsung@ccs.sogang.ac.kr
논문접수 : 2003년 1월 3일
심사완료 : 2003년 8월 4일

이로 인하여 여러 가지의 문제점도 발생된다. 이러한 여러 가지 문제점 가운데 대표적인 것으로 대역폭의 부족이라는 문제점을 생각할 수 있다. 이동 컴퓨팅은 무선 환경이라는 특징으로 인하여 사용자의 이동성을 증가시켰으나 반면 유선에 비하여 매우 제한된 대역폭에서 통신을 수행해야 한다. 이러한 제한된 대역폭이라는 환경하에서는 기존의 클라이언트 서버와 같이 클라이언트가 필요할 때마다 원하는 데이타에 접근하는 것이 대역폭에 심각한 부족 현상을 초래할 수 있으며 클라이언트 내의 질의 처리 시간의 증가를 초래한다.

이에 따라 서버에서 다수의 클라이언트가 공통적으로 원하는 데이타들을 방송(Broadcast)하고 클라이언트는 이 중에서 원하는 데이타를 선별적으로 수신하는 방송환경이 제안되었다[1]. 그러나 이러한 방송환경은 다수의 클라이언트를 그 대상으로 하고 있기 때문에 특정 클라이언트가 원하는 데이타를 수신하기 위해서는 상당 시간 동안 방송채널(Broadcast Channel)에서 기다려야 한다는 문제점이 발생한다. 이러한 문제점을 해결하기 위하여 클라이언트는 자주 사용되는 데이타를 캐시에 저장하여 이러한 문제를 해결하고, 서버는 무효화 보고(Invalidation Report)를 클라이언트에 전송하여 서버에서 갱신된 데이타에 대한 일관성(Consistency)을 보장한다.

그림 1 서버와 클라이언트 데이타의 Inconsistency

결론적으로 현대 사회에서 무선 환경이 제안됨에 따라 대역폭의 부족 문제를 해결하기 위하여 방송(Broadcast)의 필요성이 제안되었고, 이에 따라 발생하는 특정 클라이언트에서의 데이타 접근 시간의 증가 문제를 해결하기 위하여 클라이언트 캐시 활용 방안이 제안되었다[2, 3]. 그러나 캐시를 이용하면서 클라이언트 캐시와 서버 데이타 사이의 일관성 문제가 다시 제기되어 무효화 보고를 이용하는 방법이 제안되었다[4,5].

최근 이러한 무효화 보고를 이용하여 클라이언트 캐시의 일관성을 유지하는 많은 방법들이 제안되었으나[6-8] 기존의 방법들은 클라이언트의 접속 단절이 일정 시간 이상 지속될 경우 서버에서의 갱신 정도와 상관없이 캐시의 모든 데이타를 버려야 한다는 문제점을 가지고 있다. 이에 따라 본 논문에서는 클라이언트의 접속 단절 시간 보다는 서버에서의 갱신 정도에 따라 클라이언트 캐시의 일관성을 유지할 수 있는 방안을 제시한다.

본 논문의 2장에서는 기존의 무효화 보고 연구에 대한 조사를 통하여 기존 연구의 한계점을 지적하고자 한다. 3장에서는 캐시 일관성 유지를 위해 제안하는 작업 모델을 클라이언트와 서버로 나누어 설명한다. 4장에서는 클라이언트 캐시 일관성을 유지할 수 있는 알고리즘을 설명한다. 5장에서는 제안한 알고리즘과 기존 알고리즘의 성능 평가를 통하여 본 논문에서 제안하는 알고리즘의 효율성을 실험하였다. 6장에서는 기존의 BT 알고리즘과 CCI 알고리즘의 장단점을 종합하여 전체적인 효율성을 비교하고 7장에서 결론을 내린다.

2. 관련 연구

기존의 무효화 보고를 이용한 클라이언트 캐시 일관성 유지 방안은 크게 stateless 서버와 stateful 서버를 가정한 방법 두 가지로 구분할 수 있다.

stateful 서버를 가정한 방법으로는 Andrew File System[9]과 AS(Asynchronous Scheme)[10] 기법 등이 있다. 이 기법에서는 클라이언트의 캐시 상태를 알고 있는 서버가 클라이언트의 임의의 접속 단절에 대하여 클라이언트 캐시를 유지할 수 있도록 하였다. 그러나 이러한 기법은 stateful 서버를 유지하기 위해 클라이언트가 특정 데이타를 캐시에 저장할 때마다 서버에 그 정보를 전송해야 한다는 면에서 대역폭의 낭비를 초래하게 된다.

stateless 서버를 가정한 방법은 크게 다시 두 부분으로 구분된다. 첫 번째는 클라이언트가 질의 처리를 할 때마다 서버에 캐시 데이타의 일관성 확인 요청 메시지를 전송하는 방법이다. Network File System[11]은 이러한 방법의 예이다. 그러나 이 방법은 질의 수행을 할 때마다 네트워크에 상당량의 데이타 전송을 요구하게 된다. 두 번째는 서버가 주기적 혹은 비 주기적으로 서버에서 일정 시간동안 갱신된 데이타들에 대한 정보를 방송하는 방법이다. 이 방법은 위의 두 방법에 비하여 대역폭의 오버헤드을 줄일 수 있어 클라이언트 캐시 일관성 유지를 위한 방법으로 가장 널리 사용되는 방법이다. 이러한 방법은 다시 크게 네 가지 방법으로 구분할 수 있다.

첫 번째 방법은 일정 시간 동안 갱신된 데이타들 각각의 목록을 전송하는 방법으로 이러한 기법에는 AT (Amnesic Terminals), BT(Broadcasting Timestamps) 등의 방법이 있다[12]. AT 기법은 이전 무효화 보고 이후 서버에서 갱신된 데이타에 대한 정보를 전송하여 무효화 보고를 수신한 클라이언트는 캐시의 일관성을 유지할 수 있도록 한 방법이다. 그러나 AT 방법은 클라이언트가 접속 단절로 인하여 하나 이상의 무효화 보고

를 수신하지 못하는 경우 캐시의 데이타 전체를 버려야 하는 문제를 지니고 있다. 이에 따라 BT 기법이 제안되었는데, BT 기법은 window size를 설정하고 이에 따라 클라이언트가 접속 단절에 의하여 수신하지 못한 무효화 보고의 수가 window size보다 적을 경우에는 클라이언트 캐시 일관성 유지가 가능하다.

두 번째 방법은 갱신된 데이타들 각각의 목록을 모두 전송하기 보다는 각 데이타를 그룹으로 나누어 그 그룹의 어떤 데이타가 최종적으로 갱신된 시각을 전송하는 방법으로 이러한 기법에는 GCI(Group-based Cache Invalidation)[13,14]가 있다.

세 번째 방법은 갱신된 데이타에 대한 정보만을 전송하기 보다는 서버 전체 데이타에 대하여 갱신된 데이타와 그렇지 않은 데이타를 구분하여 계층화된 비트열로 전송하는 방법으로 이러한 기법에는 BS(Bit Sequence)[15], BB(Bit sequence wiht Bit count)[16,17] 등의 방법이 있다. BS 기법은 서버 전체 데이타에 대하여 갱신 정보를 계층화된 비트열로 전송하였으며 BB 기법은 BS 기법에서 필요한 갱신 정보를 선택적으로 수신할 수 있도록 한다.

마지막 방법은 위의 세 가지 방법들 가운데 일부를 적절히 혼합하여 함께 사용한 방법이다. AAW (Adaptive invalidation report with Adjusting Window)[18]는 동적으로 BT 기법과 BS 기법의 사용을 결정하는 방법이며, SCI(Selective Cache Invalidation)[13,14]는 GCI와 BT 기법을 함께 사용하여 선택적인 무효화 보고 정보의 수신이 가능하도록 한 방법이다.

표 1은 기존의 캐시 일관성 유지 방안들을 분류한 것이다. 그러나 이러한 방법들은 모두 서버에서 갱신된 데이타의 수에 따라 클라이언트의 캐시 일관성 유지 여부가 결정되기 보다는 클라이언트가 접속단절된 시간에 따라 캐시 일관성 유지 여부가 결정되는 방법이다. 따라서 서버에서 갱신이 자주 발생하지 않은 경우 오랜 시간 접속 단절된 클라이언트라 할지라도 클라이언트의 캐시 대부분을 유지할 수 있음에도 불구하고 모든 데이타를 버려야만 하는 문제점이 있었다.

본 논문에서는 기존 알고리즘의 이러한 문제점들을 해결하기 위해 stateless 서버 환경에서 캐시 일관성 유지 기법을 제안한다. 본 논문에서 제안하는 기법을 위해 클라이언트와 서버에서는 각각 다음과 같은 문제를 해결하여야 한다. 우선 클라이언트 측에서 해결해야 하는 문제는 다음과 같다. Stateless 서버 환경에서는 클라이언트가 접속 단절 이후 재 연결이 되면 캐시 일관성의 확인을 위하여 서버에 자신의 캐시 상태를 나타내는 정보를 전송해야 한다. 그러나 캐시의 모든 상태 정보를 전송하는 것은 대역폭에 많은 오버헤드를 요구하므로 클라이언트의 특성을 정확히 나타낼 수 있으면서도 크기는 작은 정보의 전송이 요구된다.

다음으로 서버 측에서 해결해야 하는 문제는 다음과 같다. 클라이언트의 캐시 성향이나 접속 단절의 시기 및 기간은 매우 가변적이므로 이러한 가변적인 환경에서 클라이언트 캐시 일관성을 유지할 수 있는 적절한 무효화 보고 전송 알고리즘이 서버에서 수행되어야 한다. 본 논문에서는 이러한 다양한 접속 단절 상황의 클라이언트에 대한 캐시 일관성 유지를 위해 기존과 다른 CCI(Cost based Cache Invalidation) 기법을 제안하였다.

3. 작업 모델

본 논문에서 제안하는 클라이언트 캐시 일관성 유지 방법을 구현하기 위하여 기존의 시스템과는 다른 작업 모델이 필요하다. 그러므로 제안하는 작업 모델을 클라이언트 작업 모델과 서버 작업 모델로 구분하여 3.1절과 3.2절에서 설명한다. 본 논문에서 제안하는 모델은 좁은 지역 단위의 시스템을 가정하며 동시에 매우 많은 클라이언트가 접속 단절되지는 않는 경우를 가정한다.

3.1 클라이언트 작업 모델

이 절에서는 클라이언트에서 수행되는 작업 모델을 설명한다. 클라이언트의 작업은 크게 자주 사용되는 데이타를 클라이언트에 저장하는 '데이타 캐시', 클라이언트가 특정 데이타를 필요로 할 때 서버에 수행하는 '서버

표 1 기존의 클라이언트 캐시 일관성 유지 방안 분류

Stateful 서버 환경	Andrew File System		
	Asynchronous Scheme		
Stateless 서버 환경	질의 처리 시에 캐시 일관성 확인 요청	Network File System	
	서버에서 갱신된 데이타 정보 방송	갱신된 데이타 각각의 정보 방송	AT
			BT
		갱신된 데이타가 속한 그룹의 정보 방송	GCI
		서버 전체 데이타에 대한 정보 방송	BS
			BB
		혼합 사용	AAW
			SCI

로의 요청', 클라이언트 내에서 발생하는 '질의 처리'의 세 경우로 구분할 수 있다. 각각의 수행은 아래와 같다.

3.1.1 클라이언트의 데이타 캐시

서버는 클라이언트의 요구 경향에 따라 전체 데이타가 아닌 일부의 데이타를 방송 하는데 방송 채널에는 그림 2와 같은 정보가 저장되어 있다. 그림 2에서 SendTS는 전송 시점의 timestamp 값, DataID는 전송하는 데이타의 ID, DataValue는 전송하는 데이타의 실제 값, Freq는 서버에서 데이타를 방송 할 때 한 번의 전체 주기 동안의 해당 데이타의 방송 횟수를 나타내는 방송 빈도이다. 방송을 받은 클라이언트는 자주 사용하는 데이타를 캐시하는데 이 때 각 데이타의 DataID, DataValue, Freq, SendTS를 함께 캐시한다. 기존의 무효화 보고와 마찬가지로 일정 시간동안 서버에서 갱신된 항목에 대한 정보를 전송하는 무효화 보고를 CIR (Common Invalidation Report)이라 정의한다.

클라이언트는 가장 최근에 수신한 CIR의 timestamp 값을 저장하는데, 이를 LastTS라 정의한다. 클라이언트 캐시의 구조는 그림 3과 같다. 여기에서 CIR은 기존의 무효화 보고에서 window size가 1인 무효화 보고를 의미하며 이에 대한 자세한 정의는 정의 3.2에서 소개한다.

3.1.2 클라이언트의 서버로의 요청

클라이언트의 서버로의 요청은 크게 BR(Broadcast Request)과 VR(Validation Request)의 두 가지로 구분할 수 있다. 서버는 서버의 전체 데이타가 아닌 일부 데이타를 방송한다. 그러므로 특정 클라이언트가 원하는 데이타가 방송되지 않은 경우가 발생할 수 있다. 이러한 경우에 해당 클라이언트는 서버에 원하는 데이타를 방송해 달라는 요청을 전송하게 되는데 이것이 BR이다. 서버는 BR이 가장 많은 데이타부터 시작해서 일정 부분을 지속적으로 방송한다. 따라서 BR의 경향에 따라 방송 스케줄은 계속 변경될 수 있다.

본 논문에서는 기존의 무효화 보고 기법도 사용하면서 동시에 기존 CIR과는 다른 무효화 보고인 RIR (Requested Invalidation Report)과 DM(Drop Message)을 사용한다. 클라이언트가 일정 시간 이상 접속 단절된 경우 CIR 만으로 캐시 일관성 확인이 불가능하다. 이러한 경우 클라이언트는 캐시에 저장되어 있는 데이타의 전체적인 특성을 나타내는 몇 가지 정보를 서버에 전송한다. 이를 수신한 서버는 클라이언트의 접속 단절 시간동안 갱신된 항목에 대한 정보를 전송하여 캐시 일관성 유지를 하는데 이러한 무효화 보고를 RIR이라 정의한다.

RIR의 경우와 마찬가지로 클라이언트가 CIR 만으로 캐시 일관성 확인이 불가능한 경우를 가정하자. 그러나 클라이언트의 접속 단절 동안 갱신된 데이타의 대부분을 해당 클라이언트가 캐시하고 있을 것으로 판단되는 경우 RIR을 전송하여 클라이언트가 대부분의 데이타를 버리도록 하는 것은 비효율적이다. 따라서 이러한 경우 서버는 DM을 전송하는데 DM을 받은 클라이언트는 모든 데이타를 버림으로써 캐시 일관성을 유지한다. CIR, RIR과 DM에 대한 자세한 정의는 3.2절의 서버 작업 모델에서 다시 설명하도록 하겠다. 이와 같이 클라이언트가 일정 시간 이상 접속 단절되어 CIR 만으로 캐시 일관성 확인이 어려운 경우 서버에 캐시의 전체적인 특성을 나타내는 정보를 전송하는데 이것을 VR이라 정의한다.

정의 3.1 이전 방송 스케줄에 포함되어 있지 않은 데이타 가운데 클라이언트가 필요한 데이타 ID의 리스트인 BR과 접속 단절로 인해 클라이언트가 하나 이상의 CIR을 수신하지 못한 경우 일관성 확인을 위해 서버에 전송하는 정보인 VR을 다음과 같이 정의한다.

- BR : <RequestID>
  - RequestID : 방송되지 않은 데이타 중, 클라이언트가 필요한 데이타의 ID
- VR : <LastTS, Freq, NumCD(Freq), UpperTS (Freq), LowerTS(Freq)>
  - LastTS : 클라이언트가 접속 단절되기 이전에 마지막으로 수신한 CIR의 timestamp
  - Freq : 데이타의 방송 빈도

SendTS	DataID		DataID		...	DataID	
	DataValue	Freq	DataValue	Freq		DataValue	Freq

그림 2 방송 채널의 구조

LastTS	DataID	DataValue	Freq	SendTS

그림 3 클라이언트 캐시 구조

- NumCD(Freq) : 클라이언트 캐시에 저장된, 방송 빈도가 Freq인 데이타의 수
- UpperTS(Freq) : 클라이언트 캐시에 저장된, 방송 빈도가 Freq인 데이타 가운데, 가장 큰 timestamp
- LowerTS(Freq) : 클라이언트 캐시에 저장된, 방송 빈도가 Freq인 데이타 가운데, 가장 작은 timestamp

그림 4는 정의 3.1에 따라 VR을 구성하는 예이다. 그림 (a)와 같이 20의 timestamp 간격으로 CIR이 전송되고 있으며 클라이언트는 TS=42일 때 접속 단절되어 TS=53일 때 재 연결 되었으며 그림 (b)와 같은 캐시를 가진다고 가정하자. 그림 (b)와 같이 클라이언트 캐시에는 DataID, DataValue, SendTS, Freq가 함께 저장된다. 또한 클라이언트가 접속 단절 이전에 최종적으로 수신한 CIR의 timestamp 값을 나타내는 LastTS가 함께 저장되어 있다.

클라이언트는 재 연결 이후 그림 (c)와 같이 VR을 구성한다. 그림 (b)에서 Freq=4인 데이타는 D1, D3, D10으로 총 3개이다. 마찬가지로 Freq=2인 데이타는 D6, D8, D15, D20으로 4개, Freq=1인 데이타는 D4, D9, D22의 3개이다. 따라서 그림 (c)에서 NumCD(4)=3, NumCD(2)=4, NumCD(1)=3이 된다. 또한 그림 (b)에서 Freq=4인 데이타 D1, D3, D10 가운데 가장 작은 SendTS 값을 가지는 데이타는 D1이며 그 값은 10이고 가장 큰 값을 가지는 데이타는 D10이며 그 값은 40이다. 따라서 LowerTS(4)=10, UpperTS(4)=40이 된다. 마찬가지로 Freq=2, Freq=1인 경우에 대하여 계산하면 LowerTS(2)=20, UpperTS(2)=40, LowerTS(1)=10, UpperTS(1)=10이 된다. 이에 따라 그림 (c)와 같은 VR이 생성된다.

3.1.3 클라이언트 내의 질의 처리

클라이언트가 질의 처리를 위하여 특정 데이타를 필요로 하는 경우를 가정하자. 만약 필요한 데이타가 캐시에 없다면 클라이언트는 방송되는 데이타를 수신하여 질의를 처리해야 한다. 반대로, 저장되어 있다면 데이타의 일관성을 확인해야 한다. 그런데 이는 클라이언트의 접속 단절 여부와 접속 단절 동안 수신하지 못한 CIR이 있는지에 따라 다음과 같이 구분되어야 한다.

우선 접속 단절되지 않은 경우를 가정하자. 필요한 데이타가 캐시에 저장되어 있지 않다면, 클라이언트는 서버에서 방송되는 데이타를 수신하여 질의를 처리한다. 또한 필요한 데이타가 캐시에 저장되어 있다면, 질의 처리는 즉시 수행되지 않고 그림 5와 같이 다음 CIR을 수신할 때까지 기다린다. CIR 수신 결과 캐시의 데이타가 유효하다면 LastTS 값을 갱신한 후 해당 데이타를 이용하여 질의를 처리할 수 있다. 만약, CIR 수신 결과 캐시의 데이타가 유효하지 않다면 이러한 데이타는 버리고 방송되는 데이타를 수신하여 질의 처리를 수행해야 한다.

둘째로 접속 단절되었으나 수신하지 못한 CIR이 없는 경우를 가정하자. 재 연결된 클라이언트는 다음 CIR을 수신할 때까지 아무 작업도 하지 않는다. 재 연결 이후

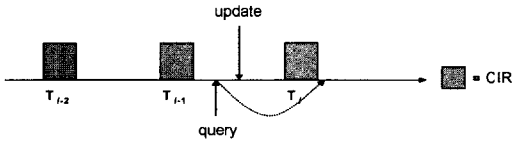
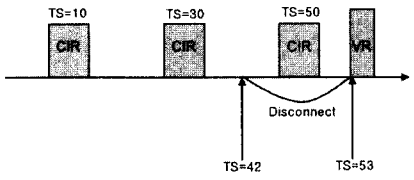


그림 5 클라이언트의 질의 처리



(a) Client Disconnect 및 VR 전송 시점

LastTS	DataID	DataValue	Freq	SendTS
30	D1	V1	4	10
	D3	V3	4	30
	D4	V4	1	10
	D6	V6	2	20
	D8	V8	2	40
	D9	V9	1	10
	D10	V10	4	40
	D15	V15	2	20
	D20	V20	2	20
	D22	V22	1	10

(b) VR 전송 시점에 Client의 Cache

LastTS	30		
Freq	4	2	1
NumCD(Freq)	3	4	3
LowerTS(Freq)	10	20	10
UpperTS(Freq)	40	40	10

(c) Reconnect 이후 VR

그림 4 클라이언트 캐시에 따른 VR 구성

처음 수신한 CIR의 timestamp와 LastTS의 차가 CIR의 방송 주기보다 작다면 이는 수신하지 못한 CIR이 없음을 의미한다. 따라서 위의 접속 단절되지 않은 경우와 동일한 과정으로 질의 처리가 가능하다.

끝으로 접속 단절되었으며 하나 이상의 CIR을 수신하지 못한 경우를 가정하자. 재 연결 이후 처음으로 수신한 CIR의 timestamp 값과 LastTS의 차가 방송 주기보다 크다면 이는 수신하지 못한 CIR이 하나 이상 존재함을 의미한다. 따라서 CIR 만으로 캐시 데이타 일관성을 확인할 수 없다. 이러한 경우에 클라이언트는 서버에 VR을 전송한다. VR을 수신한 서버는 적절한 알고리즘에 따라 RIR 혹은 DM의 전송을 결정한다.

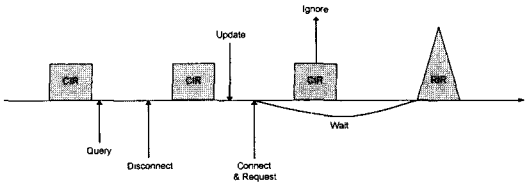


그림 6 클라이언트의 RIR 수신

3.2 서버 작업 모델

VR의 수신 여부와 클라이언트 특성, 접속 단절 동안의 갱신에 따라 서버는 CIR, RIR, DM의 전송을 결정해야 한다. 이러한 무효화 보고의 전송을 결정하기 위하여 서버에서는 방송된 데이타와 갱신된 데이타에 대한 리스트를 유지한다. 이처럼 서버에서 유지되는 정보에 관하여 3.2.1에서 설명하고, 제안하는 무효화 보고인 CIR, RIR, DM을 3.2.2에서 설명한다.

3.2.1 서버의 정보 관리 모델

서버는 VR 처리를 위하여 방송 리스트(Broadcast List)와 갱신된 데이타의 리스트(Update List)를 유지한다. 방송 리스트는 서버에서 각 데이타가 언제 방송 되었는지를 기록하는 리스트이다. 그림 7과 8은 방송 스케줄에 따른 방송 리스트 유지의 예를 보인 것이다. 서버에 D1~D1000의 1000개의 데이타가 있고 한 번의 방송 전체 주기 마다 500개의 데이타가 방송된다고 가정하자. 이러한 서버에서 timestamp가 1~10일 때는 D1~D500의 데이타가 방송되었고, timestamp가 11~20일 때는 D1~D450, D501~D550의 데이타가 방송되었으며 마찬

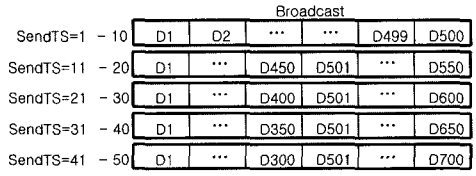


그림 7 각 시간별 Broadcast schedule의 예

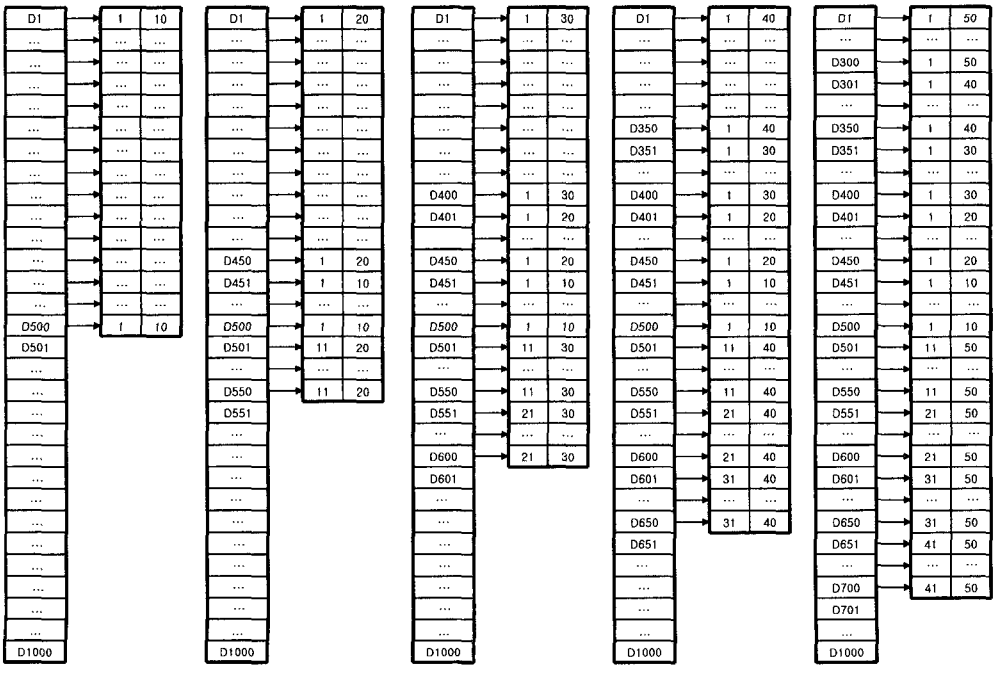


그림 8 Broadcast Schedule에 따른 Broadcast List 유지

가지로 그림 7과 같이 500개씩의 데이타가 계속 방송된 상황을 가정하자.

기본적으로 방송 스케줄은 전체적인 클라이언트의 데이타에 대한 관심도에 따라 결정되므로 많은 양의 정보가 자주 변경될 경우는 많지 않다. 따라서 그림 8과 같이 서버에 방송 리스트를 유지할 수 있다. 그림 7에서 SendTS 1부터 10까지는 D1부터 D500까지의 데이타가 방송 되었으므로 D1, D2, …, D499, D500 각각에 해당 데이타가 방송된 시각을 나타내는 1-10을 그림 8의 (a)에서와 같이 기록한다. SendTS 값 11부터 20까지는 D1부터 D450, D501부터 D550까지의 데이타가 방송되었다. 이에 따라 그림 8의 (b)를 보면 같이 D1부터 D450까지의 데이타는 1부터 20, D451부터 D500까지의 데이타는 1부터 10, D501부터 D550까지의 데이타는 11부터 20까지 방송되었음을 알 수 있다. 방송 스케줄에서 상당 부분은 이전의 방송 스케줄과 거의 동일하므로 이러한 방법을 이용하면 비교적 작은 크기의 정보를 이용하여 방송 리스트를 구성할 수 있다.

또한 서버는 각 데이타에 대하여 가장 최근에 갱신된 timestamp 값을 가지는 갱신 리스트를 유지한다. 그림 9는 각 시간에 갱신된 데이타에 대하여 유지해야 할 리스트를 시간별로 나타낸 것이다.

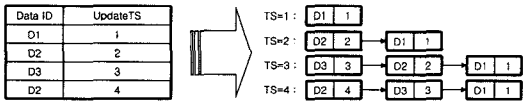


그림 9 서버의 갱신 리스트 유지

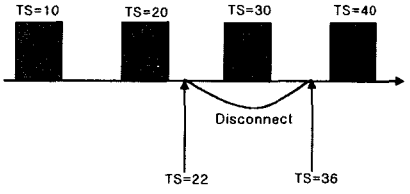
3.2.2 무효화보고의 종류

무효화 보고란 주기적, 혹은 비주기적으로 일정 시간 동안의 서버 갱신 정보를 방송하여 해당 데이타를 캐시하고 있는 클라이언트가 무효화 보고에 명시되어 있는 데이타를 버림으로써 캐시 데이타의 일관성을 유지할 수 있도록 하는 방법이다. 그러나 기존의 무효화 보고는 서버에서의 갱신된 데이타의 수와 상관없이 전송되기 이전 일정 시간동안의 서버 갱신에 대한 정보만을 포함한다. 따라서 무효화 보고에 포함되어 있는 정보의 범위 이상의 시간 동안의 클라이언트 접속 단절이 발생할 경우 서버에서 어느 데이타에, 몇 번의 갱신이 발생하였는지와 무관하게 클라이언트 캐시 일관성을 유지하지 못한다.

예를 들어 기존의 방법에서 그림 10과 같이 무효화 보고가 timestamp 10의 주기로 10, 20, 30, … 의 시각에 전송되고 있으며 무효화 보고는 전송되기 이전 10의 시간 동안 서버에서 갱신된 정보를 가지고 있는 환경을 가정하자. 어떤 클라이언트가 timestamp 값 22에서 접속 단절된 후 36에서 재 연결되었다고 가정하자. 이러한 환경에서 클라이언트는 timestamp 값 40인 무효화 보고를 이용하여 캐시 일관성을 유지할 수 없으며, 따라서 모든 캐시 데이타를 버려야 한다. 그런데 만약 서버에서 그림 (b)와 같이 갱신이 발생한 상황을 가정한다면 클라이언트가 수신하지 못한 CIR에는 D2와 D10의 데이타에 대한 갱신 정보가 포함되어 있을 것이다. 그러나 만약 클라이언트가 이러한 데이타를 저장하고 있지 않다면 클라이언트는 캐시의 데이타가 모두 일관성을 유지하고 있음에도 불구하고 모든 데이타를 버리는 문제가 발생하게 된다.

이처럼 클라이언트의 일관성 유지를 위해서는 클라이언트의 접속 단절 시간 뿐 아니라 클라이언트에 어떤 데이타가 캐시되어 있으며, 접속 단절동안 서버에서 어떤 데이타에 갱신이 발생되었는지까지 함께 고려되어야 함을 알 수 있다. 이에 따라 본 논문에서는 기존의 일반적인 무효화 보고와 함께 클라이언트의 요구에 따라 전송되는 새로운 무효화 보고를 함께 사용함으로써 클라이언트의 캐시 일관성을 유지하는 방안을 제시한다. 본 논문에서 제안하는 무효화 보고는 크게 세 종류로 나누어지며 각각은 다음과 같이 정의된다.

정의 3.2 L의 방송 주기를 가지고, 방송되기 이전 L 만큼의 시간동안에 서버에서 갱신된 데이타의 ID 정보를 가지고 있는 CIR (Common Invalidation Report)를



(a) Client Disconnect

Data ID	UpdateTS
D1	2
D5	7
D6	13
D11	14
D10	25
D2	28
D8	31
D2	37
D7	39

(b) 각 데이터의 업데이트 timestamp

그림 10 캐시 일관성과 서버 갱신의 관계

CIRTS	CIRUpdateID	CIRUpdateID	•••	CIRUpdateID	CIRUpdateID

그림 11 CIR의 구조

LastTS=10

Data ID	D1	D2	D3	D4	D5
Broadcast timestamp	10	30	20	10	40
Update timestamp	20	40	30	8	8

(a) 브로드캐스트 리스트와 업데이트 리스트의 정보

RIRTS	50
MinTS	10
MaxTS	30
RIRUpdateID	D1
	D2
	D3

(b) 서버의 RIR 구성

그림 12 RIR 구성의 예

다음과 같이 정의한다.

<CIRTS, CIRUpdateID>

- CIRTS : 서버에서 CIR이 전송되는 시점의 timestamp
- CIRUpdateID : 전송되기 이전 L 시간동안 서버에서 갱신된 데이타 ID

CIR은 서버에서의 갱신 횟수와 상관없이 L 시간 동안 갱신된 데이타의 리스트를 나타내므로 접속 단절로 인하여 수신하지 못한 CIR이 없는 클라이언트나 접속단절 되지 않은 클라이언트라면 CIR을 이용하여 캐시 일관성 유지를 할 수 있다. 그림 11은 CIR의 구조를 나타낸다.

정의 3.3 서버가 클라이언트에게 VR을 수신한 경우 캐시의 많은 부분을 유지할 수 있다고 판단될 때 전송하는 무효화 보고인 RIR(Requested Invalidation Report)을 다음과 같이 정의한다.

<RIRTS, MinTS, MaxTS, RIRUpdateID>

- RIRTS : 서버에서 RIR이 전송되는 시점의 timestamp
- MinTS : RIRUpdateID에 표현된 데이타 중, 가장 작은 방송 timestamp
- MaxTS : RIRUpdateID에 표현된 데이타 중, 가장 큰 방송 timestamp
- RIRUpdateID : 클라이언트 접속단절 이후 현재까지 서버에서 갱신된, MinTS와 MaxTS 사이에 방송되었던 데이타의 ID 리스트

그림 12는 정의 3.3에 따른 RIR 구성의 예이다. 클라이언트가 마지막으로 수신한 CIRTS 값이 10(즉, LastTS=10)이라 하고, 방송 리스트와 갱신 리스트의 값이 그림 (a)와 같으며, 서버의 timestamp 값이 50일 때 그림 (b)와 같은 RIR을 전송한다고 가정하자. LastTS=10이므로 클라이언트는 timestamp가 10일 때까지는 접속 단절되지 않았음을 의미한다. 따라서 timestamp 8에 갱신된 데이타 D4와 D5의 갱신 사실은 클라이언트가 알고 있으므로 RIR에 포함될 필요가 없다. 따라서 RIR에는 데이타 D1, D2, D3의 정보만 포함된다. 그런데 이 중에서 가장 작은 방송 timestamp 값을 가지는 것은 D1이며, 그 값은 10이므로 MinTS=10이 된다. 마찬가지로 데이타 D1, D2, D3 가운데 가장 큰 방송 timestamp 값을 가지는 것은 D2이며, 그 값은 30이 되어 MaxTS=30이 된다. 따라서 그림 (b)와 같은 RIR이 구성된다.

RIR은 클라이언트의 접속 단절 시간동안 서버에서 갱신된 데이타 가운데 클라이언트가 캐시하고 있는 데이타가 적을 것으로 판단되는 경우 전송된다. 이러한 경우, 비록 클라이언트는 장시간 접속 단절되었다 할지라도 클라이언트 캐시의 데이타 대부분을 유지할 수 있으므로 캐시의 모든 데이타를 버리는 것 보다 유지하는 것이 효율적이기 때문이다. RIR을 전송받은 클라이언트는 RIRUpdateID 중에서 캐시에 저장되어 있는 것은 버리고 그 외의 데이타는 그대로 사용함으로써 캐시 데이타의 일관성 유지가 가능하다.

정의 3.4 서버가 클라이언트에게 VR을 전송 받았으며 클라이언트 캐시의 대부분을 유지할 수 없다고 판단될 경우 클라이언트에게 캐시의 모든 데이타를 버리라는 메시지를 전송하는데 이를 DM(Drop Message)이라 정의한다.

DM을 수신한 클라이언트는 캐시에 저장한 모든 데이타가 일관성을 보장하지 못한다고 생각하고 모든 데이타를 버린다. 이러한 경우 실제는 클라이언트 캐시에 일관성을 보장하고 있는 데이타가 존재할 수 있으나 그 유지비용이 크므로 모든 데이타를 버리는 것이다.

4. 무효화보고 결정 알고리즘

서버의 방송 수행은 데이타 방송, CIR 전송, RIR 전송, DM 전송으로 구분된다. 이러한 무효화 보고의 전송 결정을 위한 알고리즘을 기술하기에 앞서 4.1절에서는 서버가 수신한 VR을 분할하는 정의 및 정리, 4.2절에서

DM과 RIR의 전송 오버헤드를 계산하기 위한 정의 및 정리, 4.3절에서 전송 무효화 보고 결정 알고리즘을 설명한다.

4.1 서버의 VR 분할

일반적으로 클라이언트의 접속 단절은 임의의 시간에 임의의 수 만큼의 클라이언트에서 발생한다. 따라서 서버가 동시에 다수의 VR을 전송받은 경우 이러한 다수의 VR을 어떻게 처리해야 하는지에 관한 문제가 발생한다.

예를 들어 그림 13과 같이 3개의 VR이 있을 때 첫 번째는 LowerTS(1)=10, UpperTS(1)=50이고, 두 번째는 LowerTS(1)=10, UpperTS(1)=20, 세 번째는 LowerTS(1)=90, UpperTS(1)=100인 경우를 가정하자. 이와 같은 경우에 10부터 20사이의 구간은 두 개의 클라이언트가 이용하고 있으며 다른 구간은 하나의 클라이언트가 이용하고 있다. 이러한 경우에 10부터 20사이 구간의 정보를 전송하지 않는다면 2개의 클라이언트가 이에 해당하는 캐시 데이타를 삭제해야 하지만 다른 구간의 정보를 전송하지 않는다면 하나의 클라이언트가 이에 해당하는 캐시 데이타를 삭제해야 한다. 이처럼 각 구간을 이용하는 클라이언트의 수가 다른 경우 그 구간을 전송하기 위한 비용이 동일하다 해도 그 중요도는 다름을 알 수 있다. 따라서 이러한 구간은 서로 다르게 취급되어야 한다.

그러므로 본 절에서는 서버가 전송받은 전체 VR에 대하여 구간을 분할하고 각 구간을 서로 다르게 취급하는 방안을 제시한다. VR은 LastTS, Freq, LowerTS(Freq), UpperTS(Freq), NumCD(Freq)의 정보를 가지고 있다. 그러므로 이러한 값의 분할 방법이 필요하다. 이에 따라 정의 4.1에서는 LowerTS(Freq)와 UpperTS(Freq)를 분할한 SubLowerTS(Freq)와 SubUpperTS(Freq)를, 정의 4.2에서는 LastTS를 분할한 SubLastTS를, 정리 4.1에서는 NumCD(Freq)를 분할한 SubNumCD(Freq)를 설명한다. 이하의 정의에서 x개의 클라이언트가 VR을 전송한 경우라면 각 클라이언트의 VR 정보를 $LowerTS_1(Freq)$, …, $LowerTS_x(Freq)$와 같이 아래첨자를 이용하여 표현하도록 하겠다. 그리고 분할된 VR은 SubVR이라 표현하기로 한다. 또한 x개의 클라이언트가 전송한 VR을 분할한 결과 y개의 SubVR이 생성되었다면 그 각각을 $SubVR_1$, …, $SubVR_y$와 같이 표현하도록 하겠다.

정의 4.1 x개의 클라이언트가 동시에 VR을 전송한 경우, 전체 $LowerTS_{1\le i\le x}(Freq)$, $UpperTS_{1\le i\le x}(Freq)$ 중 서로 다른 값들을 작은 값에서 큰 값의 순서대로 정렬한다. 정렬 결과 $k(1\le k\le 2x)$개의 timestamp 값이 순서대로 정렬되면 이러한 timestamp를 작은 것에서 큰 것 순으로 $SortedTS_1(Freq)$, $SortedTS_2(Freq)$, …, $SortedTS_k(Freq)$라 하자. $1\le y\le k-1$일 때 $SubLowerTS_y(Freq)=SortedTS_y(Freq)$와 $SubUpperTS_y(Freq)=SortedTS_{y+1}(Freq)$이라 정의한다.

그림 15는 그림 14와 같이 4개의 클라이언트가 동시에 VR을 전송한 경우 구간을 분할한 예이다. 그림의 예는 Freq=4인 경우만을 보이고 있으며 다른 값인 경우도 마찬가지로 계산할 수 있다.

정의 4.1에 따라 모든 LowerTS(4)와 UpperTS(4)를 작은 값에서 큰 값의 순으로 정렬하면 3, 5, 6, 8, 10, 12가 된다. 그러므로 $SortedTS_1(4)=3$, $SortedTS_2(4)=5$가 된다. 따라서 $SubLowerTS_1(4)$와 $SubUpperTS_1(4)$는 다음과 같이 정의된다.

$SubLowerTS_1(4)=SortedTS_1(4)=3$

$SubUpperTS_1(4)=SortedTS_2(4)=5$

정의 4.2 x개의 VR을 분할한 결과 y개의 SubVR이

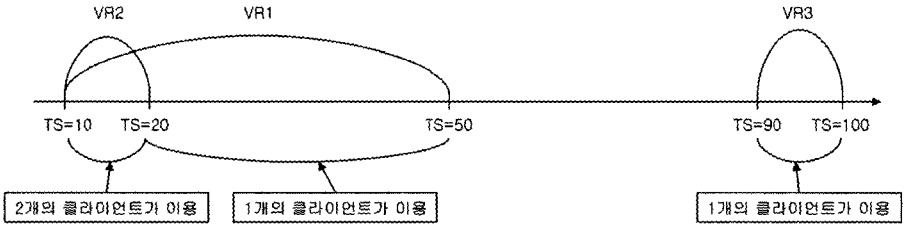


그림 13 VR의 구간별 중요도 차이

C1의 VR		C2의 VR		C3의 VR		C4의 VR	
$LastTS_1$	1	$LastTS_2$	11	$LastTS_3$	1	$LastTS_4$	11
$LowerTS_1(4)$	5	$LowerTS_2(4)$	8	$LowerTS_3(4)$	3	$LowerTS_4(4)$	10
$UpperTS_1(4)$	10	$UpperTS_2(4)$	12	$UpperTS_3(4)$	6	$UpperTS_4(4)$	12
$NumCD_1(4)$	15	$NumCD_2(4)$	16	$NumCD_3(4)$	9	$NumCD_4(4)$	10

그림 14 각 클라이언트의 VR

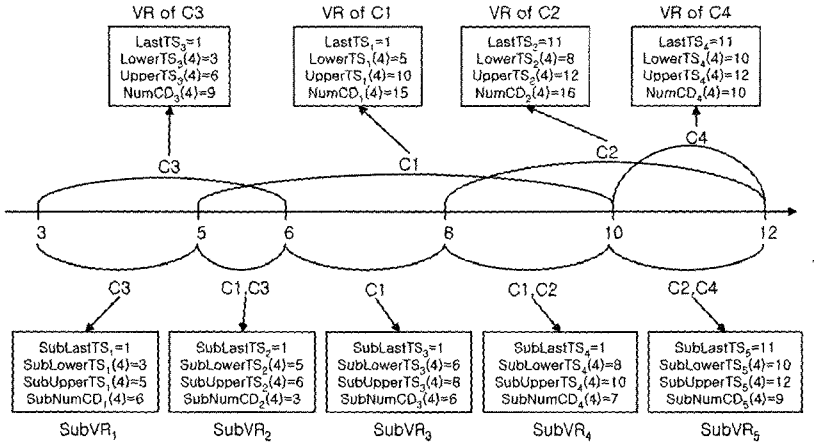


그림 15 VR의 분할

생성되었다고 가정하자. 이 때 $1 \le j \le y$인 모든 정수 j 각각에 대해서 $LowerTS_i(Freq) \le SubLowerTS_j(Freq) \le SubUpperTS_j(Freq) \le UpperTS_i(Freq)$를 만족하는 $1 \le i \le x$인 모든 정수 i를 선택하고 이러한 정수 i의 집합을 I라 할 때 $SubLastTSj = \min(LastTS_{i \in I})$라 정의한다.

그림 14, 15의 예에서 정의 4.2에 따라 아래와 같이 $SubLastTS_2$가 정의된다. 본 예에서는 4개의 VR을 이용하여 5개의 SubVR 구간이 생성되므로 x=4, y=5가 된다. 따라서 $1 \le j \le 5$가 된다. 따라서 j=1, 2, 3, 4, 5인 모든 경우에 대하여 위의 정의를 적용할 수 있으나 여기에서는 j=2인 경우만을 설명하도록 하겠다. 다른 j값에 대해서는 동일하게 적용이 가능하다.

$SubLowerTS_j(4) = SubLowerTS_2(4) = 5$

$SubUpperTS_j(4) = SubUpperTS_2(4) = 6$

그러므로 $LowerTS_i(Freq) \le 5 \le 6 \le UpperTS_i(Freq)$이 되어 이를 만족하는 $1 \le i \le 4$인 정수는 1과 3이 존재한다. 따라서 I={1, 3}이 된다. 그러므로 $SubLastTS_2$는 $LastTS_1$과 $LastTS_3$ 중에서 작은 값이 되고, 따라서 $SubLastTS_2 = 1$로 정의된다.

이런 과정을 거쳐 생성된 $SubLastTS_j$는 해당 $SubVR_j$를 포함하고 있는 $VR_{i \in I}$의 $LastTS_{i \in I}$ 값 중에서 가장 작은 값을 가지게 된다. 그러므로 $SubLastTS_j$ 이전에 갱신된 데이타들의 갱신 사실은 $VR_{i \in I}$를 전송한 클라이언트가 이미 알고 있음을 의미한다. 따라서 $SubLastTS_j$ 이후에 갱신된 데이타의 목록만을 전송하면 클라이언트 캐시의 일관성이 유지될 수 있다.

그림 15와 같이 다수의 클라이언트가 VR을 전송한 경우를 생각해 보자. 그림의 예에서 4개의 클라이언트가 전송한 VR에 의하여 5개의 SubVR이 생성되었다. 이와 같은 경우 클라이언트 $SubVR_1$은 클라이언트 C3에만 해당되는 구간이며 $SubVR_3$는 클라이언트 C1에만 해당되는 구간임을 알 수 있다. 그러나 $SubVR_2$와 같은 구간의 경우 C1과 C3가 함께 연관되어 있는 구간임을 알 수 있다. 이처럼 하나의 SubVR 구간에 대하여 다수의 클라이언트가 연관되어 있는 경우를 표현하기 위하여 본 정리에서는 $SubVR_2$에서 클라이언트 C1과 연관되어 있는 경우만을 생각한 SubNumCD(Freq) 값을 $SubNumCD_{(2,1)}(Freq)$, $SubVR_2$에서 C3와 연관되어 있는 경우만을 생각한 SubNumCD(Freq) 값을 $SubNumCD_{(2,3)}(Freq)$와 같이 나타내며, 이러한 두 값의 평균값을 $SubNumCD_2(Freq)$와 같이 표현하기로 하겠다. 이와 같이 정의된 $SubVR_j$에 대한 $SubNumCD_j(Freq)$를 계산하기 위하여 아래와 같은 정리 4.1을 이용할 수 있다.

정리 4.1 x개의 VR을 분할한 결과 y개의 SubVR이 생성 되었다고 가정하자. 이 때 $1 \le j \le y$인 모든 j 각각에 대해서 $LowerTS_i(Freq) \le SubLowerTS_j(Freq) \le SubUpperTS_j(Freq) \le UpperTS_i(Freq)$를 만족하는 $1 \le i \le x$인 모든 i를 선택하고 이러한 i의 집합을 I라 할 때 각각에 대하여 클라이언트 i의 $SubVR_j$ 구간에 해당하는 $SubNumCD(Freq)$를 $SubNumCD_{(j,i)}(Freq)$로 나타낼 때 다음과 같이 계산할 수 있다.

$$SubNumCD_{(j,i)}(Freq) \approx \frac{NumCD_i(Freq) \times (SubUpperTS_j(Freq) - SubLowerTS_j(Freq))}{UpperTS_i(Freq) - LowerTS_i(Freq)}$$

이 때, $SubVR_j$ 구간에 해당하는 클라이언트의 SubNumCD(Freq)의 평균 값인 $SubNumCD_j(Freq)$을 아래와 같이 계산할 수 있다.

$$SubNumCD_j(Freq) \approx \frac{\sum_{i \in I} SubNumCD_{(j,i)}(Freq)}{|I|}$$

여기에서 |I|는 집합 I의 원소의 개수를 의미하며, 이후 특정 집합에 | | 표시를 한 것은 해당 집합의

원소의 개수를 나타내는 것으로 한다.

증명) x개의 VR을 분할한 결과 y개의 SubVR이 생성 되었다고 가정하자. 이 때 $1 \le j \le y$인 모든 j 각각에 대해서 $LowerTS_i(Freq) \le SubLowerTS_j(Freq) \le SubUpperTS_j(Freq) \le UpperTS_i(Freq)$를 만족하는 $1 \le i \le x$인 모든 i를 선택하면 i는 $SubVR_j$가 포함되는 VR을 전송한 모든 클라이언트를 나타낸다. 그런데 위와 같은 결과는 동일한 방송 빈도를 가지는 데이타에 대한 결과이므로 비슷한 정도의 클라이언트 접근 빈도를 가진다고 할 수 있다. 따라서 이러한 동일한 방송 빈도를 가지는 데이타들을 대상으로 할 때, i 각각에 대한 $SubNumCD_{(j,i)}(Freq)$는 다음과 같은 비례식이 성립한다.

$NumCD_i(Freq) : UpperTS_i(Freq) - LowerTS_i(Freq)$
$= SubNumCD_{(j,i)}(Freq) : SubUpperTS_j(Freq) - SubLowerTS_j(Freq)$

그러므로 $SubNumCD_{(j,i)}(Freq)$는 다음과 같이 계산할 수 있다.

$$SubNumCD_{(j,i)}(Freq) = \frac{NumCD_i(Freq) \times (SubUpperTS_j(Freq) - SubLowerTS_j(Freq))}{UpperTS_i(Freq) - LowerTS_i(Freq)}$$

또한 위의 조건을 만족하는 모든 i의 집합 I는 $SubVR_j$가 포함되는 VR을 전송한 모든 클라이언트의 집합을 의미하므로, $|I|$는 이러한 클라이언트의 수를 나타낸다.

그러므로 전체 $SubNumCD_{(j,i)}(Freq)$의 평균은 다음과 같이 계산할 수 있다.

$$SubNumCD_j(Freq) = \frac{\sum_{i \in I} SubNumCD_{(j,i)}(Freq)}{|I|}$$

그림 14, 그림 15의 예에서 정리 3.1에 따라 아래처럼 $SubNumCD_1(4)$이 정의된다.

$$SubNumCD_{(1,3)}(4) = \frac{9 \times (5-3)}{6-3} = 6$$

$$\therefore SubNumCD1(4) = 6/1 = 6$$

4.2 RIR과 DM의 전송 오버헤드 판단

본 논문에서는 RIR을 전송할 경우 발생하는 오버헤드와, DM을 전송할 경우 발생하는 오버헤드를 비교하여 RIR 전송으로 인한 오버헤드가 적다면 RIR을 전송하고 그렇지 않다면 DM을 전송하는 방법을 사용한다. 따라서 이러한 각각의 오버헤드를 계산하는 방법이 필요하다. 이를 위하여 정의 4.3, 4.4, 정리 4.2에서는 기본 정의 및 정리를 설명하고, 정의 4.5 정리 4.3, 4.4에서는 DM과 RIR 전송 오버헤드를 결정하기 위한 정의 및 정리를 설명하도록 하겠다. 이하의 정의 4.3, 4.4에서는 데이타 D가 방송된 timestamp의 집합을 B(D), 서버에서 최종적으로 갱신된 시각을 U(D), 방송 빈도를 F(D)라고 가정한다.

정의 4.3 VR의 분할 결과 y개의 SubVR이 생성되었으며 $1 \le j \le y$라고 가정하자. 이 때 서버에 존재하는 임의의 데이타 D에 대해서, Freq의 방송 빈도를 가지면서 $SubLowerTS_j(Freq)$와 $SubUpperTS_j(Freq)$ 사이의 시간에 한 번 이상 방송된 데이타들의 집합 $SubSD_j(Freq)$를 다음과 같이 정의한다.

$SubSD_j(Freq)$
$= \{D | F(D)=Freq \wedge \max\{B(D)\} \ge SubLowerTS_j(Freq) \wedge \min\{B(D)\} \le SubUpperTS_j(Freq)\}$

이렇게 계산된 $SubSD_j(Freq)$는 현재까지 서버에서 Freq의 방송 빈도로 방송된 데이타 가운데 j번째 SubVR을(즉, $SubVR_j$) 포함하는 VR을 전송한 클라이언트가 캐시하고 있을 가능성이 있는 데이타의 집합을 의미한다. 반대로 $SubSD_j(Freq)$에 포함되지 않는 데이타는 캐시하고 있을 가능성이 전혀 없는 데이타를 의미한다. 따라서 서버는 서버의 모든 데이타의 갱신 여부를 확인할 필요 없이 $SubSD_j(Freq)$에 포함되는 데이타에 대해서만 갱신 여부를 확인함으로써 클라이언트의 캐시 일관성을 보장할 수 있다.

정의 4.4 VR의 분할 결과 y개의 SubVR이 생성되었으며 $1 \le j \le y$라고 가정하자. $SubSD_j(Freq)$에 포함되는 데이타 D 중에서 $SubLastTS_j$ 이후 갱신이 발생된 데이타를 나타내는 집합 $SubSU_j(Freq)$을 다음과 같이 정의한다.

$$SubSU_j(Freq) = \{D | D \in SubSD_j(Freq) \wedge U(D) > SubLastTS_j\}$$

이렇게 계산된 $SubSU_j(Freq)$는 $SubVR_j$를 포함하는 VR을 전송한 클라이언트가 갱신 여부를 모르고 있을 가능성이 있는 데이타의 집합을 의미한다. 실제 클라이언트가 필요한 갱신 정보는 서버 내의 모든 데이타에 대한 갱신 정보가 아니라 자신이 캐시하고 있는 데이타 가운데 갱신 여부를 모르고 있는 데이타에 대한 정보가 필요하다. 따라서 여기에서 계산된 $SubSU_j(Freq)$만이 클라이언트 캐시 일관성 유지를 위하여 필요한 정보임을 알 수 있다. 실제로 뒤에서 RIR이 전송되어야 할 것으로 판단되는 경우 서버는 $SubSU_j(Freq)$에 해당하는 데이타 목록을 전송한다.

정리 4.2 VR의 분할 결과 y개의 SubVR이 생성되었으며 $1 \le j \le y$라고 가정하자. 실제 클라이언트가 캐시하고 있는 데이타 가운데 접속 단절 동안 갱신가 발생한 데이타의 수를 나타내는 $SubNumCU_j(Freq)$를 다음과 같이 계산할 수 있다.

$$SubNumCU_j(Freq) = SubNumCD_j(Freq) \times \frac{|SubSU_j(Freq)|}{|SubSD_j(Freq)|}$$

증명) 정의 4.3에서 $SubSD_j(Freq)$를 $SubVR_j$를 포함하는 VR을 전송한 클라이언트가 캐시하고 있을 가능성이 있는 데이타들의 집합으로 정의하였다. 또한 정의

4.4에서는 $SubSU_j(Freq)$를 $SubSD_j(Freq)$가운데 클라이언트의 접속 단절 이후 갱신이 발생한 데이타의 집합으로 정의하였다. 또한 정리 4.1에서는 $SubNumCD_j(Freq)$가 $SubVR_j$를 포함하는 VR을 전송한 클라이언트가 캐시하고 있는 데이타의 수를 나타냄을 증명하였다. 그런데 이러한 값은 모두 같은 방송 빈도를 가지는 데이타에 대하여 계산된 값이므로 전체 클라이언트에 의하여 비슷한 정도의 캐시 비율을 가짐을 예측할 수 있다. 따라서 실제 클라이언트가 캐시하고 있는 데이타 중에서 접속 단절 동안 갱신된 데이타의 수를 나타내는 $SubNumCU_j(Freq)$를 다음의 비례식을 이용하여 계산할 수 있다.

$$|SubSD_j(Freq)| : |SubSU_j(Freq)| = SubNumCD_j(Freq) : SubNumCU_j(Freq)$$

따라서 아래와 같은 식이 성립한다.

$$SubNumCU_j(Freq) = SubNumCD_j(Freq) \times \frac{|SubSU_j(Freq)|}{|SubSD_j(Freq)|}$$

클라이언트가 $SubVR_j$에 해당하는 RIR을 전송받을 경우 일부의 데이타는 그대로 재사용이 가능하고, 일부의 데이타는 버려야 한다. 이 때 캐시에 그대로 유지되어 있는 데이타의 경우, 질의응답을 위해 캐시의 데이타의 사용 여부를 결정하기 위해서는 CIR을 수신할 때까지 기다려야 한다. 또한 서버가 RIR을 전송할 경우 RIR에는 클라이언트가 접속 단절된 동안 서버에서 갱신된 모든 데이타에 대한 정보가 포함되므로 이를 수신하는데 어느 정도의 시간이 필요하다. 이에 따라 서버가 $SubVR_j$에 해당하는 RIR을 전송할 경우 클라이언트가 이러한 RIR을 수신하기 위하여 방송 채널에서 기다려야 하는 길이를 $CostWait_j$라 정의하고, 이에 대한 내용을 정리 4.3에서 설명한다.

정리 4.3 방송 스케줄에서 하나의 데이타의 크기와 RIR에서의 하나의 데이타의 크기 비율을 α로 나타내면 $SubVR_j$에 해당하는 $CostWait_j$는 다음과 같다.

$$CostWait_j = \frac{\sum_{f \in Freq} |SubSU_j(f)|}{\alpha}$$

증명) 서버에서 RIR을 전송할 것으로 판단한 경우 서버는 갱신된 모든 데이타의 리스트를 전송할 필요가 없다. 대신 클라이언트가 캐시하고 있을 가능성이 있는 데이타 중에서 클라이언트가 갱신 사실을 모르고 있는 데이타의 리스트만을 전송하면 된다. 앞의 정의 4.5에서 이러한 데이타의 집합을 $SubSU_j(Freq)$라 정의했으므로 실제 서버가 전송해야 할 데이타의 리스트는 이 집합에 해당하는 데이타이다. $SubSU_j(Freq)$는 방송 빈도가 Freq인 데이타만을 나타내므로 서버가 전송해야 하는 데이타는 각 방송 빈도에 해당하는 $SubSU_j(Freq)$가 된다. 따라서 서버가 전송해야 하는 데이타의 전체 수는 이러한 집합의 원소들의 합으로 표현되어 다음과 같다.

$$\sum_{f \in Freq} |SubSU_j(f)|$$

그런데 일반적으로 하나의 데이타에 대한 정보를 나타내는 방송 스케줄에서의 크기는 무효화 보고에서의 크기보다 크다. 따라서 이러한 길이의 비율을 α로 나타내면, 실제 클라이언트가 RIR을 수신하기 위하여 기다려야 하는 길이의 비율은 다음과 같다.

$$CostWait_j = \frac{\sum_{f \in Freq} |SubSU_j(f)|}{\alpha}$$

정의 4.5 $SubVR_j$에 해당하는 RIR을 전송할 경우 의 오버헤드를 나타내는 값인 $CostRIR_j$를 다음과 같이 정의한다.

$$CostRIR_j = CostWait_j + NmCIRData$$

위에서 NumCIRData는 위의 $CostRIR_j$를 계산하기 바로 이전에 전송된 CIR에 포함되어 있는 데이타의 수를 의미한다.

RIR 전송으로 인하여 발생하는 오버헤드는 크게 클라이언트가 RIR 수신을 위해 클라이언트가 기다려야 하는 시간, RIR 수신 이후 유지된 데이타를 이용하여 질의 처리를 하기 위하여 기다리는 시간으로 나누어 생각할 수 있다. 정리 4.3에서는 RIR 수신을 위해 걸리는 시간으로 CostWaitj를 계산하였다. 또한 RIR 수신 이후 유지된 데이타를 이용하여 질의 처리를 하기 위해서는 CIR을 수신할 때까지 기다려야 한다. 일반적으로 CIR의 크기는 서버에서 갱신된 데이타의 수에 따라 달라지지만 여기에서는 바로 이전에 전송된 CIR에 포함되어 있는 데이타의 수를 기준으로 계산하였다. 따라서 CIR을 수신하기 위해서는 바로 이전에 전송된 CIR에 포함되어 있는 데이타의 수 만큼의 데이타를 수신해야 한다. 따라서 이러한 두 값의 합을 RIR 전송에 따른 오버헤드로 정의하여 위와 같이 계산하였다.

서버가 DM을 전송하는 경우 DM은 그 크기가 매우 작으므로 수신에 시간이 오래 걸리지 않는다. 그러나 DM을 전송받은 클라이언트는 모든 캐시 데이타를 버리고 다시 방송 채널을 통하여 데이타를 캐시해야 하므로 이에 따른 추가적인 오버헤드가 발생한다. 이와 관련하여 클라이언트가 $SubVR_j$에 해당하는 DM을 전송받을 경우 질의 처리에 필요한 데이타를 다시 수신하는데 걸리는 시간을 $CostDM_j$라 정의하고 정리 4.4에서 설명한다.

정리 4.4 f의 방송 빈도를 가지는 데이타가 N_f개 있다고 가정하면 $SubVR_j$에 해당하는 $CostDM_j$는 다음과 같이 계산할 수 있다.

$$CostDM_j = \sum_{g \in Freq} \left(\frac{1}{g} \times \sum_{f \in Freq} (N_f \times f) \times \frac{SubNumCD_j(g) - SubNumCU_j(g)}{SubNumCD_j(g) - SubNumCU_j(g) + 1} \right)$$

증명) f의 방송 빈도를 가지는 데이타 D_f를 가정하자. 하나의 D_f는 한 번의 주기 동안 f번 반복된다. 따라서 한 번의 방송 주기에 f의 방송 빈도를 가지는 데이타가 N_f개 있다고 가정하면 전체 주기에서 f의 방송 빈도를 가지는 데이타의 총 길이는 $N_f \times f$이다. 따라서 전체 방송 스케줄의 길이는 다음과 같다.

$$\sum_{f \in Freq} (N_f \times f)$$

마찬가지로 한 번의 전체 방송 스케줄에서 g번 반복되는 데이타 D_g를 가정할 때, 하나의 D_g는 방송 채널에서 g번 반복되므로 최악의 경우 Dg를 수신하기 위해 클라이언트가 방송 채널에서 기다려야 하는 길이는 다음과 같다.

$$\frac{1}{g} \times \sum_{f \in Freq} (N_f \times f)$$

클라이언트가 RIR을 전송 받은 경우와 DM을 전송 받은 경우 질의 처리를 위한 시간 차이는 각 경우 캐시에 유지된 데이타에 대해서 발생하므로, 이 만큼의 데이타를 재수신하기 위한 시간을 계산해야 한다. 일반적으로 클라이언트가 원하는 데이타가 방송 스케줄에 어떻게 분포되어 있는지에 따라 그 시간은 달라지지만 본 논문에서는 평균적으로 균일하게 분포되어 있는 상황을 가정하기로 한다. 클라이언트가 원하는 데이타의 수가 N_L이고, 원하는 데이타 전체의 수신에 걸리는 최악의 경우 시간을 T_W라 가정하면 다음을 알 수 있다.

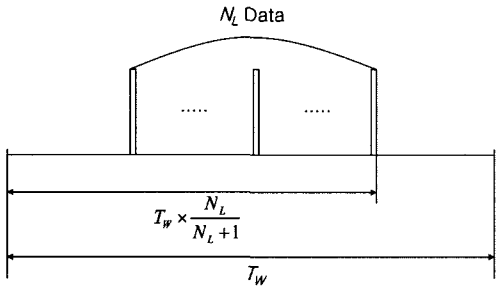


그림 16 데이타 수신을 위한 소요 시간

그림 16에서와 같이 N_L개의 데이타를 수신하기 위해서는 $T_W \times \frac{N_L}{N_L+1}$만큼의 시간이 소모됨을 알 수 있다. 그런데 클라이언트가 RIR을 수신하는 경우 캐시에는 $SubNumCD_j(g)-SubNumCU_j(g)$만큼의 데이타가 유지될 수 있고, DM을 수신하는 경우에는 유지될 수 있는 데이타가 전혀 존재하지 않는다. 따라서 클라이언트가 DM을 수신하는 경우 RIR을 수신하는 경우보다 질의 처리를 위해 추가적으로 걸리는 시간은 $SubNumCD_j(g)-SubNumCU_j(g)$만큼의 데이타를 수신하는 시간임을 알 수 있다. 따라서 위의 식에서 $NL=SubNumCD_j(g)-SubNumCU_j(g)$가 되어 아래의 식이 성립한다.

$$T_W \times \frac{SubNumCD_j(g) - SubNumCU_j(g)}{SubNumCD_j(g) - SubNumCU_j(g)+1}$$

그런데 위에서 최악의 경우 걸리는 시간을 $\frac{1}{g} \times \sum_{f \in Freq} (N_f \times f)$이므로, 다음과 같다.

$$\frac{1}{g} \times \sum_{f \in Freq} (N_f \times f) \times \frac{SubNumCD_j(g) - SubNumCU_j(g)}{SubNumCD_j(g) - SubNumCU_j(g)+1}$$

이러한 값은 각 방송 빈도를 가지는 데이타들에 대하여 모두 적용되므로 전체적으로, 클라이언트가 DM을 전송받을 경우 RIR을 전송 받은 경우보다 질의 처리를 위해 상대적으로 더 걸리는 시간인 $CostDM_j$는 다음과 계산할 수 있다.

$$MAX_{g \in Freq} \frac{1}{g} \times \sum_{f \in Freq} (N_f \times f) \times \frac{SubNumCD_j(g) - SubNumCU_j(g)}{SubNumCD_j(g) - SubNumCU_j(g)+1}$$

4.3 알고리즘

CCI 기법은 짧은 접속단절에 대하여 기존대로 CIR로 캐시 일관성을 유지한다[2]. 그러나 장기간의 접속단절에 대해서는 클라이언트가 접속 단절 시점과 캐시의 전체적인 정보를 포함하는 VR을 전송한다. 이 때 VR은 클라이언트의 모든 캐시 데이타에 대한 정보를 가지고 있는 것이 아니고 전체적인 캐시의 상태를 나타내는 정보이므로 그 크기가 작아 대역폭이 제한되어 있는 무선 환경에 적용이 가능하다. VR을 수신한 서버는 클라이언트의 접속단절 시간 동안의 갱신을 확인하여 클라이언트의 캐시에서 어느 정도의 데이타를 유지할 수 있는지를 예측하여 유지비용이 적은 경우라면 RIR을, 그렇지 않다면 DM을 전송한다.

이렇게 RIR 혹은 DM을 전송 받은 클라이언트는 전송 받은 무효화 보고의 종류에 따라 캐시 일관성 유지가 가능하다. DM을 받은 경우 모든 캐시 데이타가 일관성을 보장하지 못하는 것으로 간주하고 모든 데이타를 버린다. 또한 RIR을 받은 경우 RIR에 명시되어 있는 데이타를 버림으로써 클라이언트 캐시의 일관성 유지가 가능하다. 그림 17은 본 논문에서 제안하는 CCI 알고리즘의 전체적인 흐름을 나타낸다. 그림에서 어두운 부분은 서버에서 수행되는 부분이며 그렇지 않은 부분은 클라이언트에서 수행되는 부분을 나타낸다.

그림 18과 19는 본 논문에서 제안된 캐시 데이타 일관성 유지 알고리즘을 클라이언트와 서버로 분리하여 나타낸 것이다. 그림 18의 Step 1에서 클라이언트는 CIR을 수신하게 되면 CIRTS와 LastTS를 비교한다. 비교 결과 두 값의 차가 L보다 작거나 같다면 이는 클라이언트가 접속 단절되지 않았거나 아주 짧은 시간 접속 단절되어 수신하지 못한 CIR이 없음을 의미하므로

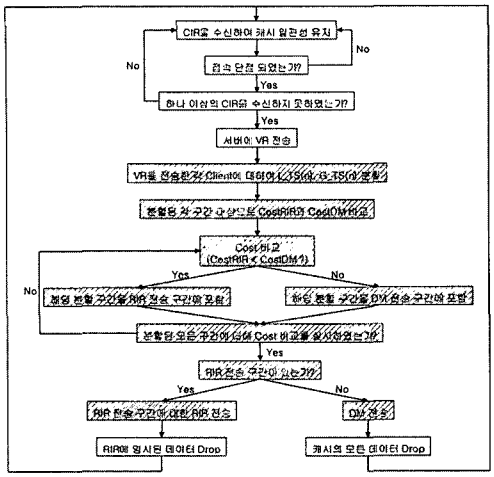


그림 17 CCI 알고리즘 수행 흐름도

```
Step 1 :
if(CIRTS-LastTS≤L) {
    for(모든 cached data) {
        if(cached data가 CIR에 있다)
            해당 cached data 삭제;
    }
    LastTS=CIRTS;
}
Step 2 ;
else {
    VR 전송;
    if(RIR 수신) {
        for(모든 cached data) {
            if(cached data가 RIR에 있다 && MinTS≤cached data의 SendTS≤MaxTS)
                해당 cached data 삭제;
            else if(cached data의 SendTS>MaxTS || cached data의 SendTS<MinTS)
                해당 cached data 삭제;
        }
        LastTS=RIRTS;
    }
    else if(DM 수신) {
        모든 cached data 삭제;
        LastTS=DMTS;
    }
}
```

그림 18 클라이언트측 알고리즘

```
Step 1 :
전체 VR에 해당하는 서로 다른 LowerTS(Freq)와 UpperTS(Freq) 정렬;
for(모든 방송 빈도) {
    for(정렬된 전체 값) {
        SubLowerTS_i(Freq)=SortedTS_k(Freq);
        SubUpperTS_i(Freq)=SortedTS_k-1(Freq);
        SubLastTS_i=min(LastTS_i<=I);
        for(해당 구간을 포함하는 모든 클라이언트)
            SubNumCD_i(Freq)-SubNumCD_i(Freq)+SubNumCD_(i,i-1)(Freq);
        SubNumCD_i(Freq)-SubNumCD_i(Freq)/|I|
    }
}
Step 2 :
for(모든 VR 분할 구간) {
    if(CostRIR_i<CostDM_i)
        해당 분할 구간을 RIR 전송 구간에 포함;
    else
        해당 분할 구간을 DM 전송 구간에 포함;
}
Step 3 :
if(RIR 전송 구간의 수==0)
    DM 전송;
else
    RIR 전송;
```

그림 19 서버측 알고리즘

수신한 CIR을 이용하여 캐시 일관성 유지가 가능하다. 따라서 CIR에 포함된 항목을 캐시에서 삭제하고 LastTS 값을 CIRTS 값으로 바꾸어 준다.

Step 2는 CIRTS와 LastTS의 차가 L보다 큰 경우이므로 이는 클라이언트가 접속 단절로 인하여 하나 이상의 CIR을 수신하지 못하였음을 의미한다. 따라서 이러한 경우 CIR 만으로 캐시 일관성 유지가 불가능하다. 그러므로 정의 2.1에 따라 자신의 캐시 특성을 반영하는 VR을 구성하여 서버에 전송한다. VR 전송 이후 서버로부터 RIR을 수신하였다면 MinTS와 MaxTS 사이의 SendTS 값을 가지면서 RIR에 포함된 데이타 항목을 삭제한다. 이 경우 MinTS와 MaxTS 사이의 SendTS 값을 가지면서 RIR에 포함되지 않은 데이타 항목은 클라이언트 접속 단절 동안 서버에서 갱신 되지 않았음을 의미하므로 그대로 사용이 가능하다. 그러나 RIR의 MinTS보다 작거나 MaxTS보다 큰 SendTS 값을 가지는 캐시 데이타는 전송된 RIR로 갱신 여부를 확인할 수 없으므로 모두 버려야 한다. 클라이언트가 VR 전송 이후 DM을 수신한 경우에는 모든 캐시 데이타를 삭제하고 단지 LastTS를 DM의 timestamp 값으로 바꾸어 준다.

그림 19는 클라이언트에게 VR을 수신한 서버가 RIR 혹은 DM 가운데 무엇을 전송할지 결정하는 알고리즘이다. 그림의 Step 1에서 서버는 클라이언트에게 수신한 VR 구간을 정의 4.1, 4.2, 정리 4.1에 따라 분할한다. 일반적으로 둘 이상의 클라이언트가 VR을 전송한 경우 VR의 LowerTS와 UpperTS는 서로 중복되는 부분이 발생하므로 독립적으로 이 구간에 대한 RIR을 구성하여 전송하면 대역폭의 낭비가 발생한다. 따라서 이와 같은 분할을 통하여 RIR 전송의 중복을 피하고 전송 여부를 판단할 수 있다.

Step 2는 Step 1에서 분할된 구간을 대상으로 CostRIR과 CostDM을 비교하는 단계이다. Step 1의 과정을 거쳐 생성된 각 SubVRj에 대한 정보인 SubLowerTS$_j$(Freq), SubUpperTS$_j$(Freq), SubLastTS$_j$를, SubNumCD$_j$(Freq)를 대상으로 정의 4.5와 정리 4.4와 같이 CostRIR$_j$와 CostDM$_j$를 계산할 수 있다. 계산 결과 CostDM$_j$ < CostRIR$_j$이라면 RIR 전송으로 인한 비용이 모든 데이타를 버리는 것 보다 크다는 의미이므로 DM 전송 구간에 포함 시킨다. 반대의 경우는 RIR 전송으로 인한 비용이 모든 데이타를 버리는 것 보다 작다는 의미이므로 RIR 전송 구간에 포함 시킨다. 이와 같은 과정은 분할된 모든 구간에 대해서 수행된다.

마지막 단계인 Step 3는 Step 2의 결과에 따라 RIR 혹은 DM을 전송하는 단계이다. 분할된 모든 구간에 대하여 알고리즘을 수행한 결과 RIR 전송 구간이 없다면 이는 어느 한 구간도 전송될 수 없음을 의미한다. 따라서 이러한 경우에는 클라이언트에게 모든 데이타를 버리라는 DM을 전송한다. 그러나 RIR 전송 구간에 일부의 부분이라도 포함되어 있는 경우 해당 구간의 데이타에 대한 갱신 정보를 포함하는 RIR을 전송한다. 이를 전송받은 클라이언트는 전송받은 RIR로 유지할 수 있는 데이타에 대해서는 캐시 일관성 유지를 하고, RIR로 캐시 일관성을 확인할 수 없는 데이타에 대해서는 이러한 모든 데이타를 버림으로써 캐시 일관성을 유지한다.

이와 같은 과정을 거쳐 수행되는 CCI 알고리즘은 클라이언트 캐시 중에서 유지 비용이 적은 부분에 대해서만 캐시를 유지하고 그렇지 않은 부분에 대해서는 캐시의 데이타를 버림으로써 모든 데이타의 일관성을 확인하거나 모든 데이타를 버리는 기존의 방법에 비하여 클라이언트의 접속 단절 시간과 서버의 갱신된 데이타의 수에 따라 더욱 효율적인 캐시 일관성 유지를 할 수 있다.

5. 성능 평가

이 장에서는 제안된 알고리즘의 성능 평가를 위하여 4가지 측면에서 성능 평가를 실시하였다. 첫째는 RIR의 전송 확률이다. CCI 방법은 서버의 갱신이 많고, 온도가 높은 데이타(즉, 클라이언트의 접근빈도가 높은 데이타)를 캐시한 클라이언트일수록 RIR의 수신 비율이 낮아져야 하며, 그렇지 않은 클라이언트일수록 RIR의 수신 비율이 높아져야 한다. 따라서 본 논문에서 제안한 RIR 전송 여부의 결정 알고리즘이 이러한 사항을 충분히 반영하고 있는지를 확인하기 위하여 클라이언트의 데이타 캐시 성향과 서버에서의 갱신 비율에 따른 RIR의 전송 확률을 측정하였다.

둘째는 클라이언트의 캐시 유지 비율이다. 기존의 무효화 보고 알고리즘은 서버에서의 데이타 갱신 비율과 무관하게 클라이언트의 접속 단절 시간에 따라 클라이언트 캐시의 데이타를 유지할 수 있는지의 여부를 판단한다. 따라서 일정 시간 이상의 접속 단절이 발생할 경우 본 논문에서 제안된 알고리즘은 기존의 알고리즘에 비하여 우수한 캐시 유지 비율을 나타낼 것으로 기대할 수 있다. 따라서 서버의 갱신 비율과 클라이언트의 접속 단절 시간에 따라 아래와 같이 클라이언트 캐시 유지 비율을 측정하였다.

$$\sum_{i \in Client} \frac{NVD_i}{CacheSize_i}$$

위 식에서 CacheSize$_i$는 클라이언트 i의 캐시 크기를 의미한다. NVD$_i$는 클라이언트 i에서 접속 단절 이후 무효화 보고 전송 과정을 거쳐 올바른 데이타로 판단된 데이타의 수를 의미한다. 즉 NVD$_i$는 접속 단절 이후 클라이언트가 그대로 사용할 수 있는 데이타의 수를 의

미한다. 따라서 위의 식은 접속 단절 이후 같은 시간에 재 연결된 모든 클라이언트를 대상으로 전체 캐시 크기에 대한 유지할 수 있는 캐시된 데이타의 수의 비율을 의미한다.

셋째는 대역폭의 오버헤드이다. 본 논문에서 제안된 알고리즘에서 서버는 클라이언트가 어떤 데이타를 캐시하고 있는지 정확히 모르는 상태에서 무효화 보고를 전송하므로 RIR에는 클라이언트 접속 단절 이후 갱신된 모든 데이타가 포함되어야 하고, 따라서 기존의 무효화 보고 알고리즘에 비하여 그 크기가 상당히 커질 수 있다. 따라서 전체 대역폭에서 무효화 보고가 차지하는 비중을 아래와 같이 계산하여 대역폭의 오버헤드를 측정하였다.

$$\frac{SizeIR}{TotalBW}$$

위의 식에서 SizeIR은 무효화 보고의 크기를 의미하며 TotalBW는 한 번의 전체 주기 동안 방송되는 데이타 전체의 크기와 무효화 보고의 크기를 더한 전체 대역폭의 크기를 의미한다.

넷째는 서버 측에서 발생하는 오버헤드이다. 제안된 CCI 알고리즘의 경우 BT 알고리즘에서는 없던 BL을 유지해야 하므로 이에 따른 추가적인 비용이 필요하다.

본 논문에서는 기존의 알고리즘 가운데 대표적인 알고리즘이라 할 수 있는 Broadcasting Timestamp (BT) 알고리즘과 그 성능을 측정하였다. BT 알고리즘 이후 발표된 많은 무효화 보고 알고리즘이 있으나 대부분의 무효화 보고 알고리즘은 BT 알고리즘에 기반하여 제안된 알고리즘이므로 BT 알고리즘을 비교의 대상으로 선택하였다. 또한 본 논문에서 제안된 알고리즘은 특정 시간 동안의 접속 단절에 따른 클라이언트 캐시 유지 방안이 아닌 서버의 갱신 비율에 따른 캐시 유지 방안이므로 기존에 제안된 많은 알고리즘들은 적절한 비교의 대상이 될 수 없을 것으로 생각된다. 따라서 본 논문에서는 BT 알고리즘의 window size 변화에 따라 클라이언트의 캐시를 유지할 수 있는 window size를 가진 BT알고리즘과 RIR을 이용한 방법이 어느 정도의 성능 차이를 보이는지에 대한 성능 평가를 실시하였다. 성능 평가에 사용된 변수는 표 2와 같으며 각 경우에 대하여 10000회씩의 실험 결과에 대한 평균값을 측정하였다. 이하의 모든 실험은 표 3과 같은 환경에서 실시하였다.

5.1 RIR 전송 비율 측정

본 논문에서 제안된 방법은 서버의 갱신이 많고, 온도가 높은 데이타를 캐시한 클라이언트일수록 RIR 전송 비율이 낮아져야 한다. 따라서 이러한 상황에서 RIR 전송 비율이 효율적으로 결정되는지 판단하기 위하여 표 4와 같은 환경에서 성능 평가를 실시하였다. 본 논문에서는 서버의 일부 데이타가 방송되는 환경을 가정하므로 이에 따라 Total_SD, Bcast, Change_access_ pattern을 설정하였다. 일반적으로 시간이 흐르면 서버의 전체 데이타가 한 번 이상씩 방송된다고 생각할 수 있으므로 이 값은 RIR의 전송 여부에 큰 영향을 미치지 않는다. Bcast_freq는 4:2:1로 가장 온도가 높은 데이타가 4번 방송될 동안 가장 온도가 낮은 데이타는 한 번 방송됨을 의미한다. 클라이언트의 캐시 크기는 임의로 100을 설정하였다. 접속 단절 시간은 접속 단절 동안 클라이언트가 수신하지 못한 CIR의 수로 표현하였는데, 이 수에 따라 RIR 전송 비율이 달라질 것이나 접속 단절 시간이 길어짐은 Update가 증가함과 같은 의미로 해석할 수 있으므로 여기에서는 Update 만을 변경시켜 성능 평가를 수행하였다.

Client_access_pattern은 클라이언트가 어떤 데이타를

표 2 성능평가를 위한 변수

항 목	내 용
Total_SD	서버의 전체 데이타 수
Bcast	한 번의 스케줄에 의하여 방송되는 전체 데이타 수
Change_access_pattern	클라이언트의 데이타 접근 빈도 변화 비율
Update	매 방송 주기마다의 전체 서버 데이타에 대한 갱신된 데이타의 비율
Bcast_freq	방송 빈도
Size_cache	클라이언트 캐시 크기
IRmiss	접속 단절 동안 miss한 무효화 보고의 수
L	방송 주기
w	window size (BT 알고리즘)

표 3 실험 환경

Machine	Processor	Memory	Operating System
Sun Blade 1000	750MHz UltraSPARC-Ⅲ	1G	Solaris 8

표 4 RIR 전송 확률 측정을 위한 변수 설정

항 목	변수 값 설정
Total_SD	1000개
Bcast	500개
Change_access_pattern	10%
Update	10%, 20%, …, 80%, 90%
Bcast_freq	4:2:1
Size_cache	100
IRmiss	10
L	10
w	-

주로 캐시하는지를 나타내는 값으로 6:3:1은 방송 빈도 4인 데이타를 빈도 2인 데이타에 비하여 2배, 빈도 1인 데이타에 비하여 6배 많이 캐시하고 있음을 의미한다.

본 논문에서 제안된 기법은 클라이언트가 어느 방송 빈도를 가지는 데이타를 주로 캐시하는지에 따라 RIR 전송 여부에 큰 영향을 미친다. 따라서 온도가 높은 데이타를 주로 캐시하는 클라이언트와 낮은 데이타를 주로 캐시하는 클라이언트, 전체적으로 고르게 캐시하는 클라이언트에 따라 RIR 전송 비율을 측정하기 위하여 6:3:1, 1:3:6, 3:4:3의 캐시 성향에 대하여 그 변화를 살펴보았다. 표 3과 같은 경우 그림 20의 결과를 나타내었다.

위의 결과에서도 볼 수 있듯이 본 논문에서 제안된 RIR 전송에 따른 클라이언트 캐시 일관성 유지 기법은 같은 데이타를 캐시하고 있는 클라이언트의 경우, 서버에서 갱신이 많을수록 RIR 전송 비율이 낮아진다. 또한, 같은 갱신 비율인 경우 방송 빈도가 높은 데이타를 캐시하고 있는 클라이언트가 그렇지 않은 클라이언트보다 RIR을 전송받는 비율이 낮아진다. 이는 서버에서 갱신이 많이 발생할수록 확률적으로 클라이언트 캐시에서 유지할 수 있는 데이타가 적어지며, 방송 빈도가 높은 데이타를 캐시하고 있는 클라이언트일수록 캐시한 데이타를 버린 이후 다시 캐시하기 위해 걸리는 시간이 적게 걸린다는 면에서 적절한 결과임을 알 수 있다.

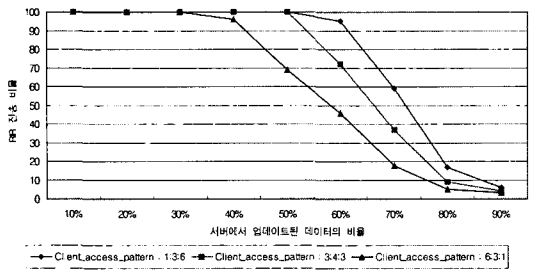


그림 20 RIR 전송 비율

5.2 클라이언트 캐시 유지 비율

그림 21은 window size가 2인 BT알고리즘과 CCI 알고리즘에서 클라이언트 접속 단절에 의하여 1개의 무효화 보고를 수신하지 못한 경우의 클라이언트 캐시 유지 비율을 측정한 것이다. 여기에서 BT알고리즘의 경우 window size가 얼마로 설정되는지와 상관없이 해당 window size보다 적은 수의 CIR을 수신하지 못한 클라이언트는 모두 캐시 일관성을 유지할 수 있으므로, 그림 21에서 BT알고리즘과 동일한 캐시 유지비율을 보일 것으로 예측할 수 있다. 그러므로 여기에서는 window size가 2이고 클라이언트가 접속 단절로 인하여 수신하지 못한 CIR의 수가 1개인 경우에 대해서만 실험하였다.

클라이언트가 1개의 무효화 보고를 수신하지 못하였으나 window size가 2이므로 BT 알고리즘에서 클라이언트는 올바른 캐시 데이타를 유지하는 것이 가능하다. CCI 알고리즘에서는 1개의 무효화 보고를 수신하지 못하였으므로 클라이언트가 서버에 VR을 전송하고 이에 따라 서버가 RIR을 전송하여 클라이언트의 데이타를 유지한다. 결과적으로 매우 짧은 시간의 접속 단절의 경우 BT와 CCI는 거의 동일한 캐시 유지 비율을 보이는 것을 알 수 있다. 그러나 서버의 갱신이 매우 많은 경우 CCI가 BT보다 조금 낮은 캐시 유지 비율을 보이는데, 이는 CCI 알고리즘의 경우 서버 갱신이 많을 때 DM을 전송함에 따라 캐시 유지 비율이 낮아지는 것으로 생각할 수 있다. 그러나 서버의 갱신이 매우 많이 발생하였으므로 DM을 전송하여 클라이언트의 전체 데이타를 버리는 것과 BT 알고리즘에서 무효화 보고를 전송하여 캐시를 유지하는 것 사이에 캐시 유지 비율의 차이가 크게 나타나지 않는 것을 알 수 있다.

그림 22는 수신하지 못한 CIR의 수를 0에서 50까지의 범위 안에서 임의로 설정하였을 때 각 알고리즘의 캐시 유지 비율을 측정한 것이다. 임의의 시간 동안의 접속 단절이 발생하므로 BT알고리즘에서는 클라이언트가 window size 이상의 수 만큼의 무효화 보고를 수신하지 못하게 되면 클라이언트 캐시의 유효성을 판단할

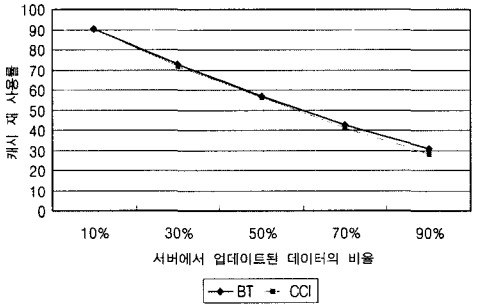


그림 21 일정 시간 접속 단절에 따른 캐시 유지 비율

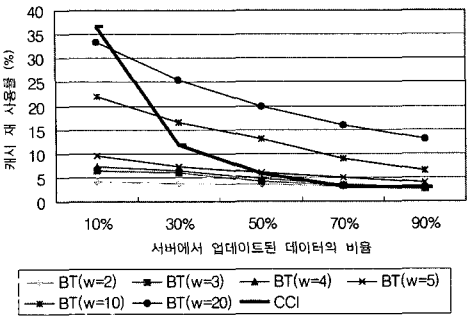


그림 22 임의시간 접속 단절에 따른 캐시 유지비율

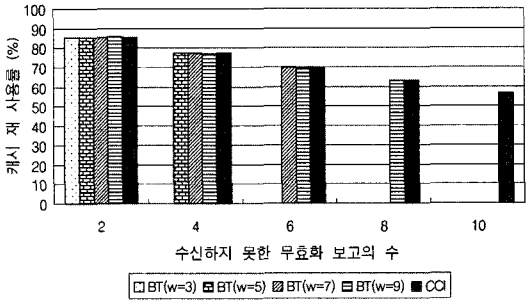


그림 23 수신하지 못한 무효화 보고의 수에 따른 캐시 유지 비율

수 없고 따라서 모든 데이타를 버리게 된다. 그러나CCI 알고리즘의 경우 임의 시간 동안의 접속 단절이 발생하더라도 서버에서의 갱신이 적다면 클라이언트의 캐시 데이타를 유지하는 것이 가능하므로 서버의 갱신이 적을 때 CCI는 BT 알고리즘보다 매우 우수한 성능을 보이는 것을 알 수 있다. 그러나 서버의 갱신이 매우 많이 발생하는 경우에 CCI 알고리즘에서는 대체로 RIR을 전송하기 보다는 DM을 전송하여 클라이언트 캐시의 모든 데이타를 버리게 하므로 갱신이 매우 많이 발생한 경우에는 CCI가 BT보다 낮은 캐시 유지 비율을 나타낸다. 그러나 서버의 갱신이 매우 많아 실제 BT 알고리즘에 의하여 유지된 캐시의 데이타 역시 매우 적으므로 이러한 경우 CCI가 BT보다 낮은 캐시 유지 비율을 나타내는 것은 큰 문제가 되지 않는다.

그림 23은 수신하지 못한 무효화 보고의 수를 2, 4, 6, 8. 10으로 하였을 때 캐시 유지 비율을 측정한 것이다. 그림에서 알 수 있듯이 BT 알고리즘에 의하여 클라이언트 캐시가 유지될 수 있는 정도까지의 접속 단절이 발생한 경우 window size가 다른 BT, CCI 알고리즘 모두 거의 비슷한 성능을 보인다. 그러나 장시간의 접속단절의 경우 BT알고리즘은 캐시를 전혀 유지할 수 없으나 CCI알고리즘은 상당부분의 캐시를 유지할 수 있다.

위의 세 가지 실험에서 알 수 있듯이 BT 알고리즘에서는 window size를 조절함에 따라 장시간 동안의 클라이언트 접속 단절에 대비할 수 있으나 CCI는 window size의 조절 없이 BT 알고리즘과 거의 비슷한 성능을 보일 수 있으며 BT 알고리즘에 의하여 클라이언트의 캐시 데이타 유지가 불가능한 정도의 장시간의 접속 단절이 발생하는 경우에도 CCI 알고리즘은 클라이언트 캐시 데이타를 상당 부분 유지할 수 있음을 알 수 있다.

5.3 대역폭 오버헤드

그림 24는 5.2절의 그림 21과 동일한 환경에서 BT 알고리즘과 CCI 알고리즘의 대역폭 오버헤드를 측정한 결과이다. 그림 21에서 알 수 있듯이 이러한 환경에서 캐시 유지 비율은 거의 동일하나 그림 24를 보면 대역폭의 오버헤드는 CCI가 훨씬 적은 것을 알 수 있다. 이는 평균적으로 BT의 경우 클라이언트가 접속 단절되지 않아도 window size 만큼의 무효화 보고 정보가 계속 전송되므로 필요 없는 무효화 보고 정보가 계속 전송되어 대역폭 오버헤드가 증가되지만 CCI의 경우 클라이언트 접속 단절이 발생한 경우에만 RIR을 전송하고 접속 단절이 발생하지 않은 경우에는 이전 무효화 보고 이후 갱신된 데이타에 대한 정보만을 전송하므로 평균적으로 대역폭의 오버헤드는 CCI가 BT보다 더 적다는 것을 알 수 있다. 또한 서버 갱신이 매우 많이 발생한 경우에 CCI 알고리즘은 RIR 보다는 DM을 주로 전송한다. 이러한 사실은 서버 갱신이 적은 경우보다 많은 경우에 BT와 CCI의 대역폭 오버헤드의 차가 점점 커지는 사실로 확인할 수 있다.

그림 25는 5.2절의 그림 22와 동일한 환경에서 대역폭 오버헤드를 측정한 결과이다. 그림 24의 결과와 마찬가지로 CCI 알고리즘은 클라이언트 접속 단절이 발생한 경우에만 RIR을 전송하므로 평균적으로 BT보다 적은 대역폭 오버헤드를 보인다. 또한 서버의 갱신이 매우 많은 경우 DM을 전송하므로 두 알고리즘의 대역폭의 오버헤드 차가 점차 커지는 것을 알 수 있다. 이는 그림

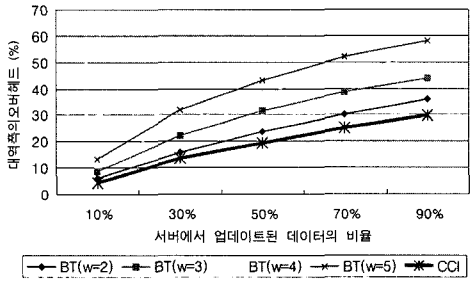


그림 24 일정시간 접속 단절에 따른 대역폭 오버헤드

22에서 서버 갱신이 매우 많이 발생한 경우 캐시 유지 비율이 낮아지는 이유가 될 수 있다. 그림 22의 BT 알고리즘에서 window size를 매우 크게 잡는 경우 CCI 알고리즘보다 상당히 높은 캐시 유지비율을 보였으나 그림 25에서 알 수 있듯이 BT 알고리즘은 window size를 증가시킴에 따라 대역폭 오버헤드가 매우 커서 실제 전체적인 효율은 CCI가 더욱 우수할 것을 예측할 수 있다.

또한 본 논문에서 제안한 알고리즘은 VR을 전송한 전체 클라이언트가 만족할 수 있는 RIR혹은 DM을 전송하므로 클라이언트 수에 따른 대역폭 오버헤드의 변화가 크지 않다. 이는 그림 25에서 보여주는 것과 같이 동시에 접속 단절된 클라이언트수가 100과 500일 경우 그 둘 간의 대역폭의 오버헤드가 차이가 최대 10%미만으로 적게 분석된 것으로 확인할 수 있다. 이와 함께 제안하는 알고리즘의 기본 환경은 동시에 매우 많은 클라이언트가 접속 단절되지 않는 것을 가정하고 있으며, 실제적으로 많은 수의 클라이언트가 동시에 접속 단절될 확률은 매우적다고 생각한다.

그림 26은 5.2절의 그림 23과 동일한 환경에서 BT 알고리즘과 CCI 알고리즘의 대역폭 오버헤드를 측정한 결과이다. 수신하지 못한 IR의 수에 따라 CCI 알고리즘과 window size가 다른 여러 BT 알고리즘의 대역폭 오버헤드를 측정한 것이다. 이와 같은 환경에서 그림 23의 결과를 보면 window size가 충분히 크다면 장시간

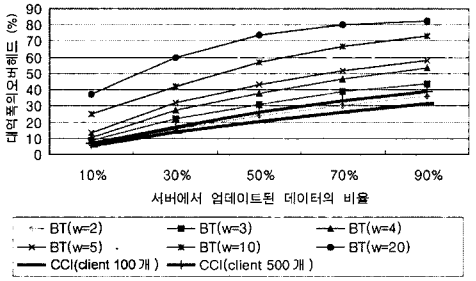


그림 25 임의시간 접속 단절에 따른 대역폭 오버헤드

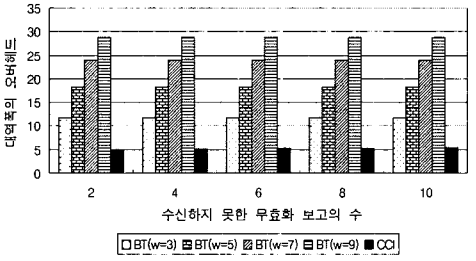


그림 26 수신하지 못한 무효화보고의 수에 따른 대역폭 오버헤드

의 접속 단절에 대하여 CCI 알고리즘과 BT 알고리즘은 거의 동일한 성능을 보임을 알 수 있다. 그러나 window size를 매우 크게 잡은 경우 클라이언트가 접속 단절되지 않은 경우에도 클라이언트가 이미 수신한 데이타 갱신 정보가 중복적으로 전송되므로 대역폭의 오버헤드는 매우 커지는 것을 알 수 있다. 그러나 위의 그림 24, 그림 25와 마찬가지로 CCI 알고리즘의 경우 클라이언트 접속 단절이 발생한 경우에만 RIR을 전송하므로 평균적으로 BT보다 매우 낮은 대역폭의 오버헤드를 보임을 알 수 있다. 또한 그림 25에서는 CCI의 클라이언트의 수에 따른 대역폭 오버헤드를 함께 측정하였다. 그러나 제안된 CCI 알고리즘은 다수의 클라이언트의 VR 요청에 대하여 개별적으로 RIR 혹은 DM을 전송하지 않고 VR을 전송한 모든 클라이언트가 만족할 수 있는 RIR 혹은 DM을 전송하므로 클라이언트의 수에 따른 대역폭의 변화가 크지 않은 사실을 확인할 수 있다.

위의 세 가지 실험에서 알 수 있듯이 CCI 알고리즘은 평균적으로 BT 알고리즘보다 훨씬 적은 대역폭 오버헤드를 나타내며 이는 BT 알고리즘의 경우 클라이언트가 접속 단절되지 않은 경우에도 클라이언트가 이미 수신한 갱신 정보를 중복적으로 전송하기 때문인 것으로 판단된다. 또한 서버의 갱신이 매우 많이 발생한 경우 CCI 알고리즘은 RIR 보다는 DM을 전송하므로 대역폭의 오버헤드를 상당히 감소시킬 수 있음을 알 수 있다.

5.4 서버측 오버헤드 측정

5.2절과 5.3절의 성능 평가에서 알 수 있듯이 서버의 갱신 비율에 따라 CCI는 그 효율성의 차이를 보이지만 기존의 BT 알고리즘 보다는 전체적으로 우수한 효율을 보임을 알 수 있다. 그러나 CCI 알고리즘은 BT 알고리즘에서는 필요 없었던 UL을 서버에 유지해야 하므로 이에 따른 서버측 오버헤드가 발생한다. 그러므로 본 절에서는 BT알고리즘과 CCI알고리즘의 서버측 오버헤드를 측정하였다.

알고리즘 수행을 위하여 서버측에서 유지해야 하는 정보는 방송 리스트와 갱신 리스트가 있다. 이 중에서 갱신 리스트의 경우 BT 알고리즘에서도 필요한 정보이지만, 방송 리스트의 경우에는 CCI 알고리즘에서만 필요한 정보이다.

갱신 리스트는 각 데이타의 가장 최근 갱신 시각만을 저장하므로 최대 서버에 존재하는 데이타의 수만큼의 데이타에 대한 최종 갱신 시각이 저장되어야 하며 이에 따른 저장 공간의 오버헤드는 BT와 CCI 모두 동일하다. 반면 방송 리스트의 경우에는 클라이언트의 데이타 접근 성향에 따라 달라지며, 이에 따라 방송 리스트에서 각 데이타에 연결되는 리스트의 길이가 달라질 수 있다.

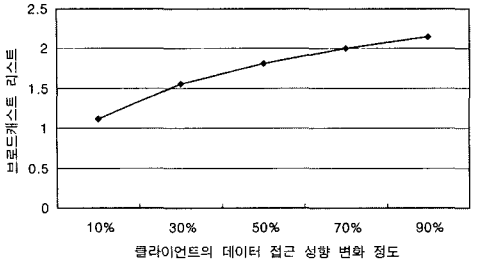


그림 27 클라이언트 데이타 접근 성향 변화에 따른 방송 리스트의 평균 길이

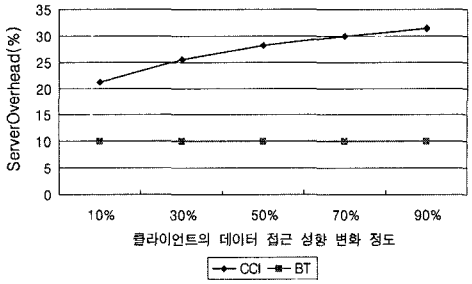


그림 28 서버측 오버헤드 비교

그러나 3.2.1절에서도 언급했듯이 방송 스케줄은 전체적인 클라이언트의 데이타 접근 성향에 따라 변화하므로 그 변화 정도가 심하지는 않다.

그림 27은 클라이언트의 데이타 접근 성향의 변화에 따른 방송 리스트의 평균 길이를 나타낸다. 여기에서 클라이언트의 데이타 접근 성향의 변화는 클라이언트가 이전까지 요구하던 데이타와는 다른 데이타를 요구하는 비율이 어느 정도인지를 퍼센트로 나타내는 값이다. 그림에서 알 수 있듯이 클라이언트의 데이타 접근 성향이 매우 크게 변화하는 경우에도 실제 방송 리스트의 길이는 소폭 증가하는 사실을 알 수 있다. 그림 27에서 나타난 것과 같은 방송 리스트의 평균 길이를 LengthBlist라고 할 때, 서버에서 유지해야 하는 방송 리스트의 전체적인 길이는 서버 전체 데이타에 대하여 각각 LengthBlist 만큼씩 존재한다. 따라서 서버에 존재하는 데이타의 수를 NumSD라고 하면 방송 리스트의 크기는 다음과 같이 계산할 수 있다.

$$NmSD \times LengthBlist$$

따라서 CCI와 BT 알고리즘 수행을 위해 서버에서 유지하는 정보의 오버헤드를 ServerOverhead라 하고, CCI와 BT 각각의 알고리즘의 ServerOverhead 값을 ServerOverheadCCI, ServerOverheadBT라고 할 때, 그 값은 다음과 같이 계산할 수 있다. 아래의 식에서 ServerOverheadCCI의 NumSD×LengthBlist+NumSD와 ServerOverheadBT의 NumSD는 데이타의 수를 의미하며 Storage Capacity of the Server는 바이트 단위의 서버 저장 용량을 의미한다. 그러므로 분모와 분자의 단위를 맞추기 위하여 분자에 β를 곱하는데, 여기에서 β는 하나의 데이타의 크기를 의미한다. 이 β 값은 서버에 저장된 데이타의 종류에 따라서 달라질 수 있을 것이다. 아래의 식에서도 알 수 있듯이, CCI 알고리즘은 BT 알고리즘보다 항상 더 큰 ServerOverhead를 가진다.

$$ServerOverhead_{CCI} = \frac{\beta(NmSD \times LengthBlist + NmSD)}{Storage \cap acity\ of\ the\ Server}$$

$$ServerOverhead_{BT} = \frac{\beta \times NmSD}{Storage \cap acity\ of\ the\ Server}$$

그림 28은 클라이언트의 데이타 접근 성향의 변화 비율에 따른 BT알고리즘과 CCI알고리즘의 Server-Overhead를 비교한 것이다.

위의 실험 결과에 따르면 BT 알고리즘의 경우 방송 리스트를 유지하지 않고 갱신 리스트만을 유지하므로 클라이언트의 데이타 접근 성향의 변화와 서버측 오버헤드와는 무관하게 항상 같은 결과를 보인다. 위의 결과는 갱신 리스트의 오버헤드를 10%로 설정했을 때의 값을 나타낸다. 그러나 CCI 알고리즘의 경우에는 갱신 리스트 이외에 방송 리스트를 유지해야 한다. 그런데 방송 리스트의 크기는 최소 갱신 리스트만큼의 크기를 가지게 되며 이에 따라 BT 알고리즘에서 서버측 오버헤드 값인 10%의 두배인 20% 이하로는 작아질 수가 없다. 실험 결과에서도 알 수 있듯이 클라이언트의 데이타 접근 성향의 변화 정도가 작을 때는 서버측 오버헤드가 실제로 20%에 가까운 값을 가짐을 알 수 있다. CCI 알고리즘의 서버측 오버헤드는 클라이언트의 데이타 접근 성향의 변화 정도가 커지면서 함께 커지게 되는데, 이는 방송 리스트의 크기가 커지면서 서버측 오버헤드가 증가하기 때문이다. 그러나 클라이언트의 데이타 접근 성향의 변화 정도가 매우 심한 90%의 경우에도 약 32% 정도의 오버헤드를 나타내어, 실제 거의 대부분의 경우에서 BT 알고리즘에 비하여 3배 이상의 오버헤드는 보이지 않을 것으로 예측할 수 있다.

6. CCI와 BT의 효율성 비교

5장의 성능 평가에서 알 수 있듯이 클라이언트 캐시 유지 비율과 대역폭 오버헤드 측면에서는 CCI가 BT보다 우수한 성능을 보이며, 서버측 오버헤드 면에서는 BT가 CCI보다 우수한 성능을 보인다. 이렇게 두 알고리즘은 서로 장단점을 가지고 있으므로 무조건 어느 하나의 알고리즘이 우수하다고는 할 수 없다. 그러므로 본 절에서는 5.2, 5.3, 5.4절에서 실험하였던 3가지의 요소를 종합하여 BT와 CCI알고리즘이 어떠한 환경에서 어느 정도 우수한지를 확인하고자 한다.

5.2절의 캐시 유지 비율을 CacheReuseability, 5.3절의 대역폭 오버헤드를 BWOverhead라고 하고 5.4절의 서버측 오버헤드를 ServerOverhead라 하고 BT와 CCI 알고리즘의 이러한 값을 첨자로 구분할 때, 각 알고리즘의 Performance를 정리 6.1에서 설명하겠다. 여기에서 각 알고리즘의 Performance는 알고리즘의 절대적인 효율성을 나타내는 값이 아니라, 서로 간에 어느 정도의 성능 차이를 보이는가를 상대적으로 비교하기 위한 수치이다.

정리 6.1 접속 단절 이전의 클라이언트 캐시의 전체 데이타와 재연결 이후 클라이언트 캐시에서 일관성이 보장되는 것으로 확인된 데이타 사이의 비율을 CacheReuseability, 전체 대역폭 사용량 중에서 클라이언트의 재연결 이후 클라이언트 캐시 데이타의 일관성 확인을 위하여 서버가 클라이언트에 전송하는 무효화 보고에 의한 사용량의 비율을 BWOverhead, 서버의 전체 크기 중에서 방송(Broadcast) 리스트와 갱신 리스트의 크기 비율을 ServerOverhead라 가정하자. 이때 BT알고리즘과 CCI 알고리즘의 상대적인 효율성을 나타내는 값인 PerformanceCCI와 PerformanceBT는 다음과 같이 계산된다.

$$Performance_{CCI}=\frac{W_C\times CacheReuseability_{CCI}}{W_B\times BWOverhead_{CCI}\times W_S\times ServerOverhead_{CCI}}$$

$$Performance_{BT}=\frac{W_C\times CacheReuseability_{BT}}{W_B\times BWOverhead_{BT}\times W_S\times ServerOverhead_{BT}}$$

여기에서 W_C, W_B, W_S는 각각 시스템에서 CacheReuseability, BWOverhead, ServerOverhead의 상대적인 중요성을 나타내는 값으로 시스템에 따라 다르게 결정된다.

증명) 위의 Performance는 CacheReuseability, BWOverhead, ServerOverhead의 변화에 따른 각 알고리즘의 상대적인 효율성을 나타내는 값이므로 이러한 세 값의 변화의 비율을 Performance가 그대로 반영하고 있음을 증명하도록 하겠다.

먼저 다음을 가정하자.

$CacheReuseability_{CCI}=a\times CacheReuseability_{BT}$

$BWOverhead_{CCI}=b\times BWOverhead_{BT}$

$ServerOverhead_{CCI}=c\times ServerOverhead_{BT}$

여기에서 $CacheReuseability_{CCI}=a\times CacheReuseability_{BT}$는 CCI 알고리즘의 CacheReuseability가 BT 알고리즘의 CacheReuseability보다 a배 높음을 의미하므로 이러한 비율을 Performance가 그대로 반영하고 있다면 CCI 알고리즘의 Performance 값은 BT알고리즘의 Performance 값보다 a배만큼 커야 한다.

또한 $BWOverhead_{CCI}=b\times BWOverhead_{BT}$는 CCI 알고리즘의 BWOverhead가 BT 알고리즘의 BWOverhead보다 b배 높음을 의미한다. 그런데 BWOverhead는 그 값이 작아질수록 전체 효율성이 좋아진다고 할 수 있으므로 이러한 비율을 Performance가 그대로 반영하고 있다면 CCI알고리즘의 Performance는 BT알고리즘의 Performance보다 b배만큼 작아야 한다.

마찬가지로 $ServerOverhead_{CCI}=c\times ServerOverhead_{BT}$ 이므로 CCI알고리즘의 Performance 값은 BT알고리즘의 Performance 값보다 c배만큼 작아야 한다.

그러므로 CCI 알고리즘의 Performance는 BT 알고리즘의 Performance보다 $\frac{a}{b\times c}$배 만큼 커야 함을 알 수 있다. 그러므로 다음이 성립한다.

$$Performance_{CCI}=\frac{a}{b\times c}\times Performance_{BT}$$

그런데 위에서 가정한 조건을 $Performance_{CCI}$계산식에 대입하면 다음과 같다.

$$Performance_{CCI}=\frac{W_C\times a\times CacheReuseability_{BT}}{W_B\times b\times BWOverhead_{BT}\times W_S\times c\times ServerOverhead_{BT}}$$

$$\therefore \frac{b\times c}{a}\times Performance_{CCI}=\frac{W_C\times CacheReuseability_{BT}}{W_B\times BWOverhead_{BT}\times W_S\times ServerOverhead_{BT}}$$

$$\therefore \frac{b\times c}{a}\times Performance_{CCI}=Performance_{BT}$$

$$\therefore Performance_{CCI}=\frac{a}{b\times c}\times Performance_{BT}$$

그러므로 정리의 Performance 계산식은 CacheReuseability, BWOverhead, ServerOverhead의 변화에 따른 각 알고리즘의 상대적인 효율성을 그대로 잘 반영하고 있다.

위에서 W_C, W_B, W_S는 시스템에 따라 위의 세가지 요소가 서로 다른 중요도를 가지는 경우 그 중요도를 구분하기 위한 값이다. 그러나 이 값은 특정 시스템에 따라 달라지는 값이므로 본 논문에서는 $W_B=W_S=W_C$로 가정하고 실험을 하였다.

그림 29는 임의 시간동안 접속 단절된 클라이언트에 대해서 각 알고리즘의 효율성을 비교한 것이다.

그림 (a)는 $\frac{ServerOverhead_{CCI}}{ServerOverhead_{BT}}=2$일 경우,

그림 (b)는 $\frac{ServerOverhead_{CCI}}{ServerOverhead_{BT}}=3$일 경우,

그림 (c)는 $\frac{ServerOverhead_{CCI}}{ServerOverhead_{BT}}=4$일 경우

의 Performance를 나타낸다. 그림 28에서도 알 수 있듯이 RIR의 서버측 오버헤드는 BT의 서버측 오버헤드를 4배 이상 넘지 않으므로 이러한 실험으로 전체적인 알고리즘의 효율성을 충분히 파악할 수 있다. 실험 결과 CCI는 BT에 비하여 좋은 효율을 보인다. 또한 서버의 용량이 충분하여, 서버측 오버헤드가 큰 문제가 되지 않는

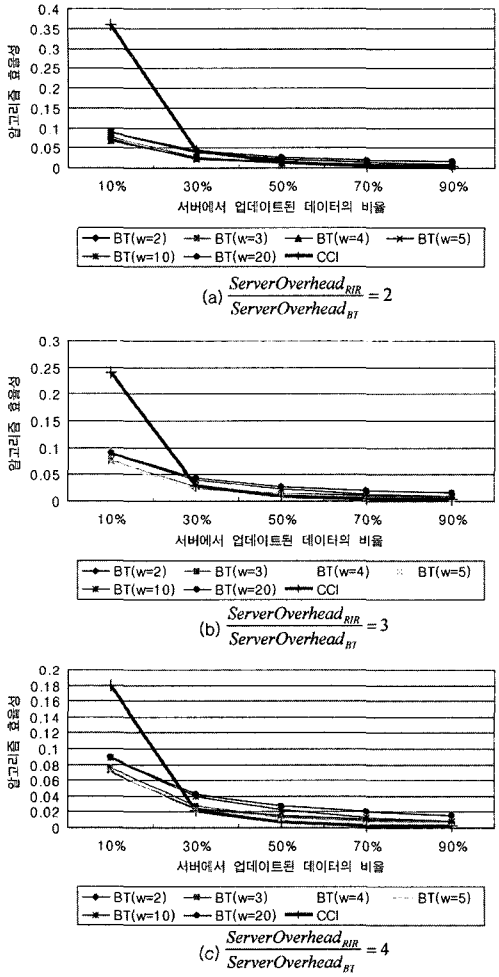


그림 29 BT와 CCI의 서버측 오버헤드에 따른 효율성 비교

환경에서 CCI가 BT에 비하여 우수한 성능을 보인다.

7. 결 론

본 논문에서는 stateless 서버 환경에서 장시간 접속 단절된 클라이언트의 효율적인 캐시 일관성 유지 기법을 제안하였다. CCI는 클라이언트 캐시 일관성 유지를 위하여 접속단절 동안의 서버에서의 갱신 비율과 클라이언트의 데이타 캐시 성향을 함께 고려한다. 또한 기존의 무효화보고 알고리즘은 일정 시간 이상의 클라이언트 접속 단절이 발생할 경우 서버에서의 갱신 정도와 상관없이 캐시의 데이타를 모두 버려야 했으나 본 논문에서 제안한 CCI 알고리즘은 장시간의 접속단절이 발생하는 경우에도 서버에서 갱신이 적다면 클라이언트 캐시의 데이타 대부분을 유지할 수 있는 장점을 가지고 있다.

이를 위하여 서버에서는 갱신 리스트와 방송(Broadcast) 리스트를 유지해야 하는 오버헤드가 생기지만 장시간의 클라이언트 접속 단절에 대해서도 갱신 상황에 따라 클라이언트 캐시를 유지할 수 있다. 따라서 장시간의 접속단절이 발생할 경우 모든 캐시 데이타를 버려야 하는 기존 방법에 비하여 높은 캐시 유지 비율을 보인다. 또한 RIR 전송시 갱신된 모든 데이타가 아니라 클라이언트가 캐시하고 있을 가능성이 있는 데이타만을 전송하며, 기존의 BT 알고리즘은 클라이언트의 접속 단절과 상관없이 항상 window size 만큼의 갱신 정보를 전송하는데 반해, 본 논문에서 제안하는 CCI 알고리즘은 클라이언트의 접속 단절이 발생한 경우에만 RIR을 전송한다. 이렇게 함으로써 대역폭의 절약을 기대할 수 있다.

이와 같은 특징은 성능 평가를 통해 확인할 수 있는데, 성능 평가 결과 서버의 갱신이 많고, 온도가 높은 데이타를 캐시한 클라이언트일수록 RIR 전송 비율이 낮아짐을 확인할 수 있다. 이렇게 RIR 전송 비율이 낮아지면 이에 따라 클라이언트 캐시의 데이타 유지 비율이 낮아질 수 있다. 그러나 성능 평가에서도 확인할 수 있듯이 서버에서 매우 많은 데이타가 갱신된 경우 실제 모든 갱신 항목을 클라이언트에 전송해도 결과적으로 클라이언트에서 유지할 수 있는 데이타는 거의 존재하지 않으므로 갱신된 모든 데이타 항목에 대한 정보를 모두 전송하는 기존의 알고리즘에 비하여 큰 성능 저하를 보이지는 않는다.

반면 갱신이 많은 경우 RIR을 전송하지 않음으로써 발생하는 대역폭의 절약은 기존의 알고리즘에 비하여 매우 뛰어나 전체적으로 효율적인 결과를 나타낸다. 그러나 CCI 알고리즘은 BT 알고리즘에 비하여 서버측 오버헤드가 커서 서버의 용량이 매우 재한되어 있고 클라이언트의 데이타 접근 성향의 변화가 심한 경우에는 적절치 않음을 알 수 있다.

결과적으로 CCI는 기존에 비하여 대역폭을 절약하고 장시간의 클라이언트 접속 단절에 대비할 수 있다는 장점을 가지고 있다. 특히 장시간의 접속 단절이 빈번하고 서버의 갱신이 매우 많이 발생하는 환경에서 window size를 크게 설정한 BT보다 매우 우수한 대역폭의 절약 효과를 나타내며, 장시간의 접속 단절이 빈번하고 서버의 갱신이 적은 경우는 우수한 캐시 유지 비율을 보인다. 또한 장시간의 접속 단절이 거의 발생하지 않는 환경이라 할지라도 기존과 거의 유사한 캐시 유지 비율을 보이면서 큰 대역폭의 절약 효과가 있다.

참 고 문 헌

[1] Imielinski T., Viswanathan S., Badrinath B.R.,

"Data on Air : Organization and Access," IEEE Transactions on Knowledge and Data engineering, Vol. 9, No. 3, May/June 1997.
[2] Boris Y. Chan, Antonio Si, Hong V. Leong, "Cache Management for Mobile Databases : Design and Evaluation," Proceedings of the Fourteenth International Conference on Data Engineering, Orlando, Florida, USA, February 23-27, 1998.
[3] Swarup Acharya, Michael Franklin, Stanley Zdonik, "Prefetching from a Broadcast Disk," Proceedings of the International conference on Data Engineering, New Orleans, LA, Feb. 1996.
[4] Swarup Acharya, Michael Franklin, Stanley Zdonik, "Disseminating Updates on Broadcast Disks," Proceedings of the 22nd VLDB Conference, 1996.
[5] D. Barbara, R.J. Lipton, "A Class of Randomized Strategies for Low-Cost Comparison of File Copies," IEEE Transactions on Parallel and Distributed Systems, Vol. 2, No.2, pp. 160-170, 1991.
[6] Qun Ren, Margaret H. Dunham, "Using Clustering for Effective Management of a Semantic Cache in Mobile Computing," Proceedings of the International Workshop on Data Engineering for Wireless and Mobile Access, 1999.
[7] Leandros, Tassiulas and Chi-Jiun Su, "Optimal Memory Management Strategies for a Mobile User in a Broadcast Data Delivery System," IEEE Journal on Selected areas in Communications, Vol. 15, No. 7, September 1997.
[8] A. Prasad Sistla, Ouri Wolfson, Yixiu Huang, "Misimization of Communication Cost Through Caching in Mobile Environments," IEEE Transactions on Parallel Distributed systems, Vol. 9, No. 4, April 1998.
[9] Satyanarayanan M., Howard J.H., Nichols D.N., Sidebotham R.N., Spector A.Z., and West M.J., "The ITC Distributed File System: Principles and Design," Proceedings of the 10th ACM Symposium on Operating Systems Principles, pp. 35-50, December 1985.
[10] Khurana, S., Kahol, A., Gupta, S.K.S. and Srimani, P.K., "An efficient cache maintenance scheme for mobile environment," Proceedings of the 20th International Conf. pp. 530-537, 2000.
[11] Sandberg R., Goldberg D., Kleiman S., Walsh D., and Lyon B., "Desing and Implementation of the Sun Network Filesystem," Proceedings of the USENIX Summer Conference, pp. 119-130, June 1985.
[12] Barbara D. and Imielinski T., "Sleepers and workaholics: caching strategies in mobile environments," Proceedings of the ACM SIGMOD, 1994.
[13] Jun Cai, Kina-Lee Tan, "Energy-efficient selective cache invalidation," Wireless Networks 5, pp. 489-502, 1999.
[14] Jun Cai, Kian-Lee Tan, Beng Chin Ooi, "On Incremental Cache Coherency Schemes in Mobile Computing Environments," Proceedings of the 13th International Conf on Data Engineering, pp. 114-123., 1997.
[15] Jing J., Elmagarmid A., Helal A. and Alonso R., "Bit-Sequences : An Adaptive Cache Invalidation Method in Mobile Client/Server Environments," ACM Mobile Networks and applications Vol 2, pp.115-127, 1997.
[16] Kian-Lee Tan, "Organization of Invalidation Reports for Energy-Efficient Cache Invalidation in Mobile Environments," ACM Mobile Networks and Application 6, pp. 279-290, 2001.
[17] Kian-Lee Tan, Jun Cai, Beng Chin Ooi, "An Evaluation of Cache Invalidation Strategies in Wireless Environments," IEEE Transactions on Parallel and Distributed systems, Vol. 12, No. 8, August 2001.
[18] Qinglong Hu and Dik Lun Lee, "Adaptive cache invalidation methods in mobile environments," Proceedings of the 6th IEEE International Symposium, pp.264-273, 1997.

송 원 민
1994년 3월~2001년 2월 서강대학교 수학과 학사. 2001년 3월~2003년 2월 서강대학교 컴퓨터학과 석사. 2003년 2월~현재 팬택&큐리텔 연구원. 관심분야는 이동통신, 모바일컴퓨팅, 모바일데이타베이스

정 성 원
정보과학회논문지 : 데이타베이스
제 30 권 제 3 호 참조