모바일 컴퓨팅 환경에서 높은 캐쉬 가용성을 위한 캐쉬 액세스 프로토콜

최재호 이상근
고려대학교 컴퓨터학과
{redcolor25, yalphy}@korea.ac.kr

A Cache Access Protocol for High Usability in a Mobile Computing Environment

Jae-Ho Choi and SangKeun Lee
Department of Computer Science and Engineering, Korea University

요 약

모바일 컴퓨팅 환경에서 Invalidation Report(IR)의 사용은 무선대역폭과 배터리 사용 측면에서 효율적인 방법임이 증명되어 왔다. 그러나, IR기반의 방법은 긴 응답시간과 낮은 캐쉬 가용성을 가지며, 또한 클라이언트가 충분한 캐쉬를 가지고 있다고 해도 짧은 응답시간을 요구하는 응용프로그램에는 적합하지 않다는 단점을 가진다. 본 논문에는 이러한 문제를 해결하기 위해 효율적인 캐쉬 사용이 가능한 프로토콜을 제안하였다. 제안한 기법에서 클라이언트는 수동적으로 IR을 기다려 캐쉬를 사용하지 않고 보다 능동적으로 캐쉬를 사용한다. 이러한 기법을 통해 우리는 필요 없는 응답시간 지연을 야기하는 " false alarm" 를 제거 할 수 있었다. 제안된 한계치 기반 방법을 통해 우리는 약간의 데이터 현재성을 손해 보았지만, 응답시간을 최적화 할 수 있었다. 제안한 기법을 평가하기 위한 시뮬레이션 결과는 우리가 제안한 기법의 아주 적은 현재성 손해를 보지만, 응답시간을 크게 줄일 수 있다는 것을 보여준다.

1. 서 론

최근 셀룰러 폰, PDA등과 같은 모바일 기기 사용증가에 따라 더욱 많은 서비스들의 무선환경을 통해 서비스될 것으로 예상되고 있다 [6].[15]. 그러나, 이러한 서비스들은 좁은 대역폭, 제한된 배터리 용량, 잦는 클라이언트의 접속단절 등의 모바일 환경으로 인한 제약 사항들을 가진다. 모바일 환경에서 자주 사용하는 데이터를 클라이언트 측에 저장하는 것은 제한된 배터리 사용을 효율적으로 하고 좁은 대역폭 사용을 위한 경쟁을 줄이기 위한 효율적인 방법이다 [4],[5]. 그러나 캐쉬에 저장된 데이터는 원본 데이터는 업데이트 되었지만, 복사본은 업데이트 되지 않을 수도 있기 때문에 무효화될 수 있으며[9], 그 결과 클라이언트는 자신의 캐쉬를 바로 사용할 수 없다. 최근 많은 연구에서 IR기반의 캐쉬무효화 방법들이 그 확장성 (Scalability) 때문에 매우 효과적인 기법임을 증명되었다 [2],[10],[12]. 그러나 IR기반의 방법은 긴 응답시간과 낮은 캐쉬 가용성을 가진다는 단점이 있으며, 많은 경우에 클라이언트는 특정한 시간 안에 데이터를 얻어야 한다 [8]. 다음의 예를 살펴보자.

> 예제1. 교통정보 시스템을 가정해보자. 그 시스템의 서버는 주기적으로 교통정보에 대한 IR을 방송한다. 어떠한 교차로 앞에서 운전자가 교차로에 연결된 도로에 대한 교통 정보를 요구하고, 그 요구한 정보가 운전자의 캐쉬에 있을 때, 기존의 IR기반의 방법을 사용하면, 데이터가 캐쉬에 있음에도 불구하고 그 운전자는 다음 IR 을 확인할 때까지는 캐쉬에 있는 데이터를 사용할 수 없다. 운전자는 교차로를 지나기 전까지 그 교차로에 대한 정보를 원하였지만, IR주기가 교차로 통과 이전에 오지 않아서 운전자는 교차로에 대한 정보가 캐쉬에 있음에도 불구하고 데이터를 사용하지 못했다.

예제 1에 나타난 시스템은 당연히 사용자의 요구를 만족시 키지 못할 것이다. 만약 사용자가 원하는 시간 안에 IR을 받았다 할지라도 데이터 사용시점 이후에 업데이트가 발생하여 불필요한 응답시간 지연을 일으키는 " false alarm" 문제가 발생할 수도 있다 [5]. 많은 연구들이 이러한 긴 응답시간을 줄이기 위해 이루어져 왔다 [3],[5]. 이러한 연구들에서는 Updated Invalidation Report (UIR)이 사용되었다. 하지만 이러한 연구에서도 역시 클라이언트는 수동적으로 IR을 기다리기만 했다.

데이터 일관성 문제에 대해서도 많은 연구들이 보고되어 왔다 [9], [13], [16]. 우리는 보다 효율적인 캐쉬 사용을 위해 우리는 기존의 기법에서 사용된 일관성 모델보다 약한 일관성 모델을 채택하였으며, 새로운 프로토콜인 Direct Cache Access Protocol (DCAP)와 Threshold-based Cache Access Protocol (TCAP)를 제안하였다. 우리가 제안한 기법에서는 적절한 한계치(Threshold)를 사용하면 불필요한 응답지연을 일으키는 " false alarm" 이 발생하지 않는다.

다음 장에서는 배경지식을 간단히 소개하고, 3장에서는 제안한 기법을 설명할 것이다. 4장에서는 제안한 프로토콜에 대한 실험결과를 제시하고, 5장에서는 결론을 맺을 것이다.

2. 배 경

데이터 일관성 문제는 여러 분산처리 시스템에서 연구보고 되어 왔다 [7],[14]. IR기반의 대부분의 기법들은 latest value 일관성 모델을 사용하여 왔다 [1],[4],[5]. 이 일관성 모델에서 클라이언트는 항상 가장 최근의 값을 사용하게 된다.

우리는 기존의 IR기반 연구에서 사용된 시스템 모델과 유사한 시스템 모델을 사용하였다. 이 모델은 두 개의 채널, 방송채널과 클라이언트가 업로드 할 수 있는 채널이 있고, 클라이언트는 로컬 캐쉬를 가진다. 하나의 서버는 다수의 클라이언트에게 서비스를 하며, 서버는 주기적으로 클라이언트에게 IR과 클라이언트가 요청한 데이터를 방송채널을 통해 보낸다. [2]는 IR기반의 캐쉬 무효화 기법에 대해 소개했다. IR은 다음과 같이 정의되는 정보를 클라이언트에게 주기적으로 보낸다.

- $IR_i = \{<d_x, t_x> | (d_x \in D) \wedge (T_i - w*L < t_x \leq T_i)\}$

D는 데이터 아이템집합, w는 윈도우 사이즈, L은 IR방송주기를 나타낸다. 서버는 $w*L$ 만큼의 업데이트 정보를 보내기 때문에 클라이언트는 그동안 접속단절되어 있어도 자신의 캐쉬를 재사용할 수 있다. 그러나 이러한 보장을 받기 위해 클라이언트는 자신의 캐쉬에 데이터가 있어도 반드시 다음 IR을 들은 후에 자신의 캐쉬에 데이터를 사용할 수 있다. 이러한 문제를 해결하기 위해 UIR을 이용한 기법이 제안되었다 [5][19]. UIR은 *(1,m)* 색인[11]에 사용된 아이디어를 이용하여 IR을 m 번 복사해서 IR주기 안에 넣음으로 긴 응답시간을 줄였다. 그러나, 전체 IR을 모두 반복하기에는 데이터가 크기 때문에 다음과 같은 UIR을 사용하였다.

- $UIR_{i,k} = \{d_x | (d_x \in D) \wedge (T_{i,k-1} < t_x \leq T_{i,k})\}$ 이때, $(0<k<m-1)$,

$(m-1)$은 하나의 IR주기안에서 복사된 UIR의 개수

3. 제안하는 프로토콜

3.1 동기와 기본 아이디어

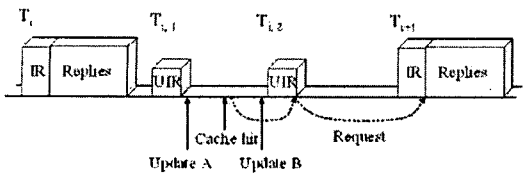


그림 1. 불필요한 응답지연의 예

그림 1은 불필요한 응답지연의 예를 보여준다. d_x가 캐쉬에 있으며, A에서 업데이트 되었을 때와 B에서 업데이트 되었을 때, 모두 $T_{i,2}$에서 방송되는 UIR은 d_x가 업데이트 되었다는 정보를 포함하게 된다. 만일 A와 B사이에서 캐쉬에 있는 d_x를 사용하게 된다면, B에서 업데이트가 일어났을 경우에 기존의 UIR기반의 방법에서는 UIR을 보고 업데이트가 되었음을 알고 서버에 데이터를 요청하게 되고, 다음 IR 뒤에서 데이터를 받게 된다. 그러나, 실제 캐쉬힛이 일어난 시점에는 업데이트는 일어나지 않았다. 즉, 캐쉬에 있는 데이터는 그 시점에는 최신의 데이터 이다. 따라서, UIR은 " false alarm" 을 클라이언트에게 주게 되는 것이다. 만약 캐쉬에 있는 데이터를 바로 사용한다면, 이러한 문제는 없어지게 된다. 또한, 이러한 " false alarm" 문제 때문에 기존에 사용되는 일관성 모델인 latest value 일관성 모델은 빠른 접근 시간을 얻기 위해 적합하지 않다. 일관성 모델은 응용프로그램에 따라 독립적이기 때문에 [1],[5],[9], 본 논문에서는 캐쉬 가용성을 높이기 위해 기존보다 약한 일관성 모델을 사용하였다. 우리의 일관성 모델은 latest IR 일관성을 사용한다. 이 일관성 모델에서 클라이언트는 가장 최근의 IR/UIR에 의해 일관성을 보장 받는다.

응답시간을 최소화 하기 위해서 우리는 DCAP와 TCAP를 제안하였다. 기존의 IR/UIR 기반의 알고리즘들에서는 클라이언트는 캐쉬사용을 위해 수동적으로 IR/UIR을 기다리지만, 우리가 제안한 프로토콜에서는 능동적으로 캐쉬를 사용한다. DCAP에 서는 클라이언트는 자신의 캐쉬에 데이터가 있으면 다음 IR/UIR까지 기다리지 않고 곧바로 그 데이터를 사용한다. 클라이언트가 곧바로 자신의 캐쉬를 사용했을 때 그 데이터가 최신의 데이터가 아니기 위해서는 첫째, 최근의 IR/UIR까지는 그 데이터가 최신의 데이터여야하고, 둘째, 최근의 IR/UIR 부터 클라이언트의 캐쉬힛이 일어난 시점사이에 업데이트가 일어나야 하며, 셋째, 그 업데이트가 일어난 데이터를 클라이언트가 사용해야만 한다. 세가지 조건이 동시에 일어날 확률은 실제로 매우 적을 것으로 예상되기 때문에, DCAP에서는 캐쉬를 곧바로 사용하더라도 현재성의 손해는 크지 않을 것으로 예상하였다. TCAP에서는 일정한 시간으로 한계치를 정하고 그 시점 이전에는 캐쉬를 곧바로 사용하고 한계치 이후에는 기존의 IR/UIR처럼 다음 IR/UIR을 기다린다. 실제로 오랜 응답지연을 가져오는 한계치 이전의 경우에 캐쉬를 바로 읽음으로써 DCAP와 근사한 응답시간을 가지게 되지만, 위에 설명한 조건 중 두 번째 조건이 한계치에 비례하여 줄어들게 되므로, 현재성의 손해는 이에 비례하여 줄어들 수 있다. TCAP의 경우 한계치는 클라이언트에 의해 조절되어 질 수 있으며, 한계치가 0 일 경우에는 기존의 IR/UIR기법과 같기 때문에 가장 큰 데이터 현재성을 보장받을 수 있고, 한계치가 한 IR/UIR주기일 경우 TCAP는 DCAP와 동일하다. 즉, DCAP와 UIR/IR방법은 모두 TCAP의 특수한 경우이다.

3.2 알고리즘

제안하는 시스템의 서버 알고리즘은 기본적으로 IR/UIR기반의 서버 알고리즘[5]과 동일하다. 서버는 IR/UIR을 IEEE 802.11 [18]과 유사한 방법으로 방송한다. 그림 2는 TCAP의 클라이언트 알고리즘이다. DCAP알고리즘은 TCAP의 특수한 경우이기 때문에 생략한다.

* Notations
- $Q_i = \{d_x \mid d_x$ has been queried from the threshold to $T_i\}$.
- $Q_{i,k} = \{d_x \mid d_x$ has been queried in the interval $[T_{i,k-1}, T_{i,k}]\}$.
- t_x^s: the timestamp of cached data item d_x.
- T_l: the timestamp of the last received IR.
- T_c: the current time.
- $T_{threshold}$: the time threshold, $0 \le T_{threshold} \le L$.
- L_{data}: an id list of the data items that a client requested from the server.

(A) When a client C_j receives IR_a and L_{bcast}:
  if $T_l < (T_i - L*w)$ then drop the entire cache entry;
  for each data item $< d_x, t_x^s >$ in the cache do
    if $((d_x,t_x) \in IR_a) \wedge (t_x^s < t_x)$ then invalidate d_x;
  for each $d_x \in L_{bcast}$ do
    if $(d_x \in L_{data})$
      then download d_x into local cache;
      Use d_x to answer the previous query;
    if d_x is an invalid cache item
      then download d_x into local cache;
      $T_l = T_i$;
  if $(L_{data} \neq \emptyset)$
  then query (Q_i).
  $L_{data} = \emptyset$;
(B) When a client receives a $UIR_{i,k}$:
  if missed IR_a
    then break;
    wait for the next IR;
  for each data item $< d_x, t_x^s >$ in the cache do
    if $(d_x \in UIR_{i,k})$
    then invalidated d_x;
  if $(L_{data} \neq \emptyset)$ then query $(Q_{i,k})$.
    $L_{data} = \emptyset$;
(C) When a client receives a query (d_x):
  if $(T_c - T_l < T_{threshold})$
    if d_x is a valid entry in the cache
      then use the cache's value to answer the query immediately;
    else send request(d_x) to the server;
  else $L_{data} = L_{data} \cup d_x$;

(D) Procedure query(Q)
  for each $d_x \in L_{data}$ do
    send request(L_{data}) to the server;

그림 2. TCAP알고리즘

4. 성능 평가

성능평가를 위해 우리는 이산 시뮬레이션 패키지인 CSIM [17]을 사용하였다. 우리의 시뮬레이션 모델과 파라미터는 [4],[5]에서 사용된 것과 유사하다. 사용된 파라미터는 표 1과 같다. 우리는 현재성의 손실측정을 위해 시간과 데이터의 버전을 사용하였다. 손실측정에 사용된 식은 다음과 같다. N_{total}은 클라이언트가 서버 또는 로컬캐쉬로부터 받은 데이터의 총 개수를 나타내며, N_s는 최신데이터가 아닌 데이터의 개수, $Diff_v$와 $Diff_t$는 각각 가장 최신데이터와 관심있는 데이터를 받은 버전과 시간의 차이를 나타낸다.

$$\text{평균 현재성 손해량} = \frac{Ns * \frac{\sum Diff_t\ (Diff_v)}{Ns}}{Ntotal} = \frac{\sum Diff_t\ (Diff_v)}{Ntotal}$$

4.1 평균응답시간 측정

그림 3은 TCAP와 DCAP 그리고 IR/UIR의 평균응답시간을 측정한 그래프이다. 응답시간에서는 DCAP가 가장 좋은 성능을 보였으며, TCAP의 한계치가 4/L일때, TCAP는 거의 DCAP와 유사한 응답시간을 보였다. 한계치가 3/L일때 역시 TCAP는 UIR그래프보다 DCAP쪽에 근접한 것을 볼 수 있는데, 이는 TCAP의 경우 더 오랜 응답시간을 요구하는 한계치 이전에 데이터가 캐쉬에 있으면 곧바로 사용하기 때문이다. IR은 우리의 프로토콜과 UIR과는 다르게 캐쉬사이즈가 증가해도 응답시간이 크게 좋아지지 않는 것을 볼 수 있는데 이는 캐쉬에 있는 데이터가 많아지더라도 그 데이터가 업데이트 될 확률도 같이 증가하기 때문이다. UIR이나 우리의 프로토콜의 경우에는 IR보다 자주 유효한지 검사하기 때문에 그런 현상이 발생하지 않는다. 캐쉬사이즈가 커지면 커질수록 우리의 프로토콜이 UIR보다 효율적인 이유는 캐쉬에 있는 데이터가 증가할수록 우리의 프로토콜이 효율적이기 때문이다.

표 1. 시뮬레이션 파라미터

The parameters (unit)	Contents
Database size (data item)	1000
Number of clients	100
Broadcast interval (sec)	200
Broadcast bandwidth (bits/sec)	10000
Cache size (data item)	1 to 500
Broadcast window (broadcast interval)	4
UIR replicate times	4
HOT data items	5% of DB
Cold data items	95% of DB
Hot data update probability	33%
Client's zipf parameter	0.95
Query generate time (data item)	1 to 200

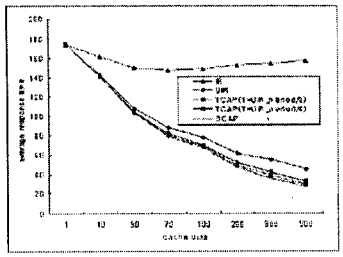


그림 3. 평균응답시간(L=40)

4.2 현재성 손실량

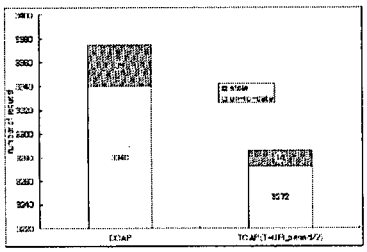
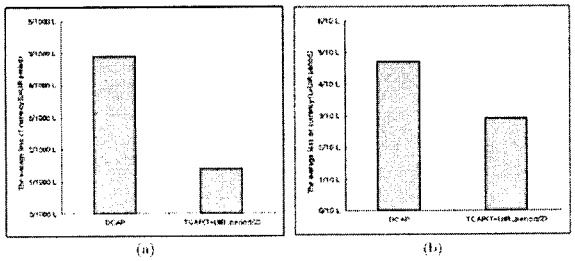


그림 4. 클라이언트의 총 데이터 요청평균

그림 4는 클라이언트의 총 데이터 요청을 관찰한 결과이다. 클라이언트는 서버와 캐쉬로부터 데이터를 가져오는데 전체 요청에서 데이터가 가장 최신의 것이 아닌 경우, 다시 말해, 현재성을 손해 보는 경우는 DCAP에 경우 1%, TCAP(한계치=L/2)의 경우 0.4%에 불과했다. 실제 업데이트가 많이 일어나는 데이터의 경우 가장 최근 IR이전에 업데이트가 일어날 확률도 동시에 높아지고, 업데이트가 자주 안 일어나는 데이터의 경우에는 최근 IR부터 데이터 사용시점까지 중에 업데이트가 일어날 확률도 적다. 이러한 사실은 앞서 언급한 현재성을 잃을 조건이 성립되는 것을 더욱 어렵게 한다.



(a) (b)

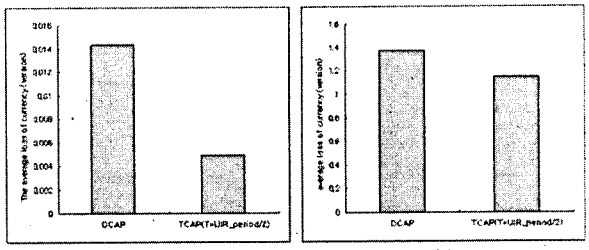
그림 5. 시간측면에서 측정한 현재성 손실량

그림 5는 시간측면에서 측정한 데이터의 현재성 손실량을 보여준다. 그림5(a)는 전체 데이터에 대한 평균 손실량이고, 그림5(b)는 실제 현재성을 잃은 데이터를 받았을 경우에 평균 손실량이다. DCAP에 경우 전체 데이터에 대해 평균적으로 약 0.005L (L=UIR주기)만큼 예전의 데이터를 받는다는 의미이다. 실제 현재성을 잃은 경우에도DCAP에 경우 0.5L 정도 이전 데이터를 읽었다. 이 실험결과는 시간의 측면에서 평균적으로 잃는 현재성의 손해가 크지 않은 것을 보여준다.

그림 6은 버전측면에서 살펴본 데이터의 현재성 손실량이다. 그림 6(a)는 전체 데이터에 대한 평균이며, 그림 6(b)는 실제 손실된 데이터를 받았을 때이다. 이 실험에서도 역시 DCAP에 경우 0.014정도 예전 버전 데이터를 받는 것으로 조사되었다. 버전역시 시간과 유사하게 매우 작은 데이터의 현재성 손실량을 보여준다.

5. 결 론

지금까지 대부분의 IR기반의 연구들에서 클라이언트들은 수동적으로 기다리는 역할을 했다. 본 논문에서는 클라이언트들이 능동적일 필요가 있음에 주목했다. 이런 생각을 바탕으로, 우리는 DCAP 를 제안했다. 제안한 프로토콜을 사용하여, 클라이언트는 자신의 캐쉬를 즉시 사용할



(a) (b)

그림 6. 버전측면에서 측정한 현재성 손실량

수 있게 되었고, " false alarm" 문제를 제거할 수도 있게 되었다. DCAP를 사용할 때 잃을 수 있는 현재성을 보완하기 위해 우리는 또한 DCAP의 보다 일반적인 경우인 TCAP를 제안하였다.

우리는 클라이언트의 응답시간과 캐쉬 가용성을 높일 수 있는 몇 가지 방법들에 대해 실험을 했다. 실험결과는 우리가 제안한 방법이 약간의 현재성 손실은 있지만, 응답시간을 최적화 할수 있는 것을 보여준다. 앞으로 우리는 클라이언트의 다양한 데이터 요청 패턴과 접속단절에 대해서 더 연구할 계획을 가지고 있다.

6. 참조문헌

[1] S. Acharya, M. J. Franklin, and S. B. Zdonik. Disseminating updates on broadcast disks. VLDB, pages 354-365, 1996.

[2] D. Barbar and T. Imielinski. Sleepers and workaholics: Caching strategies in mobile environments. SIGMOD, pages 1-12, 1994.

[3] G. Cao. On improving the performance of cache invalidation in mobile environments. Mobile Network and Applications, 7(4):291-303, 2002.

[4] G. Cao. Proactive power-aware cache management for mobile computing systems. IEEE Transaction on Computers, 51(6):608-621, 2002.

[5] G. Cao. A scalable low-latency cache invalidation strategy for mobile environments. IEEE Transaction on Knowledge and Data Engineering, 15(5):1251-1265, 2003.

[6] B. Y. Chan, A. Si, and H. V. Leong. A framework for cache management for mobile databases: Design and evaluation. Distributed Parallel Databases, 10(1):23-57, 2001.

[7] A. Datta, D. E. VanderMeer, A. Celik, and V. Kumar. Broadcast protocols to support efficient retrieval from databases by mobile users. ACM Transaction on Database Systems, 24(1):1-79, 1999.

[8] J. Fernandez and K. Ramamritham. Adaptive dissemination of data in time-critical asymmetric com- munication environments. Euromicro Real-Time Systems Symposium, pages 195-203, 1999.

[9] H. Guo, P.-A. Larson, R. Ramakrishnan, and J. Goldstein. Relaxed currency and consistency: How to say "good enough" in SQL. SIGMOD, pages 815-826, 2004.

[10] Q. Hu and D. L. Lee. Cache algorithms based on adaptive invalidation reports for mobile environments. Cluster Computing, 1(1):39-50, 1998.

[11] T. Imielinski, S. Viswanathan, and B. Badrinath. Data on air: Organization and access. IEEE Transaction on Knowledge and Data Engineering, 9(3):353-372, 1997.

[12] J. Jing, A. K. Elmagarmid, A. Helal, and R. Alonso. Bit-sequences: An adaptive cache invalidation method in mobile client/server environments. Mobile Networks and Applications, 2(2):115-127, 1997.

[13] A. Kahol, S. Khurana, S. K. Gupta, and P. K. Srimani. A strategy to manage cache consistency in a disconnected distributed environment. IEEE Transaction on Parallel Distributed Systems, 12(7):686-700, 2001.

[14] B. Nizeberg and V. Lo. Distributed shared memory: A survey of issues and algorithms. IEEE Computer, 24(8):52-60, 1991.

[15] F. J. Ovalle-Martinez, J. S. Gonzalez, and I. Stojmenovic. A parallel hill climbing algorithm for pushing dependent data in clients-providers-servers systems. Mobile Networks and Applications, 9(4):257-264, 2004.

[16] D. J. Ram, M. U. Mahesh, N. S. K. C. Sekhar, and C. Babu. Causal consistency in mobile environment. In SIGOPS, pages 34-40, 2001.

[17] H. Schwetman. CSIM user's guide(version 18). Mesquite Software,Inc., http://www.mesquite.com.

[18] IEEE. WG. Draft supplement to part 11: Wireless medium access control (MAC) and physical layer (PHY) specifications. Technical report, IEEE Std 802.11e/D4.3, 2003.

[19] M. K. H. Yeung and Y.-K. Kwok. Wireless cache invalidation schemes with link adaptation and downlink traffic. Mobile Computing, 4(1):68-83, 2005.