W-Lease: 무선 인터넷 시스템을 위한 강한 캐쉬 일관성 보장방법

김영한○, 이혁준, 정광수*
광운대학교 컴퓨터 공학과, 전자공학부*
oldkim@gwu.ac.kr, {hlee,kchung*}@daisy.gwu.ac.kr

W-Lease : The Strong Cache Consistency Mechanism for Wireless Internet System

Younghan Kim○, Hyukjoon Lee, Kwangsue Chung*
Dept. of Computer Engineering, School of Electronics Engineering*, Kwnagwoon University

요 약

데이터 캐싱은 웹 컨텐츠의 양적인 팽창에 대응하기 위한 대표적인 방법 중 하나이다. 데이터 캐싱에서 캐쉬 일관성 유지 기법은 매우 중요하다. 리스 알고리즘은 유선 망에서의 강한 캐쉬 일관성 보장 방법 중 하나로 서버와 클라이언트간의 부하를 적절히 분산시켜 주는 효과가 매우 높은 것으로 알려져 있다. 그러나 이 방식이 무선 인터넷 환경에 적용되기 위해서는 잦은 접속단절을 갖는 무선망의 특성이 고려되어야 한다. 본 논문에서는 기존의 리스 알고리즘을 보완하여 무선망에서 효율적으로 사용될 수 있도록 한 W-Lease 알고리즘을 제안한다. W-Lease 알고리즘은 접속단절 상태에 놓인 클라이언트들에 대하여 서버가 일관성 관리를 하게 함으로써, 강한 캐쉬 일관성을 효율적으로 유지할 수 있다. 제안된 알고리즘은 구현 및 실험을 통해 기존의 알고리즘과 비교하여 향상된 성능을 입증한다.

1. 서 론

최근 들어 이동 통신 기술 및 무선 랜 기술의 발전으로 이동 컴퓨팅 환경이 확산되고 있다. 그러나 배터리 수명의 제약과 통신 요금의 부담으로 인해, 이러한 이동 단말들은 자주 접속단절 상태에 놓일 수밖에 없다[1]. 따라서 클라이언트와 서버간에 효율적으로 캐쉬를 사용하기 위한 데이터 전송 메커니즘에 대한 연구가 활발히 진행되어 왔다. 캐쉬 일관성유지는 캐쉬 메커니즘에 있어서 가장 중요한 부분 중의 하나이다. 즉, 현재 클라이언트가 캐쉬에 저장해 놓은 데이터가 서버가 가지고 있는 원래의 데이터와 일치함을 보장해주는 기법을 말한다. 캐쉬 일관성 유지 기법은, 사용자에게 유효기간이 지난 데이터를 제공할 확률이 어느 정도 되는가에 따라 강한 캐쉬 일관성(strong consistency)과, 약한 캐쉬 일관성(weak consistency)으로 나뉘어 진다. 최근의 경향으로 볼 때, 현재 웹 서버에 의해 제공되는 데이터들은 지금까지와는 달리 갱신이 더욱 자주 이루어지고 있다. 어떤 종류의 데이터들은 (예를 들어, 뉴스 사이트나 주가 정보들) 몇 분 단위로 계속 바뀌고 있으며, 이에 따라 강한 캐쉬 일관성에 대한 요구가 더욱 커지고 있다[2]. 그러나 무선망에서는 빈번한 접속단절의 발생으로 인하여, 강한 캐쉬 일관성을 제공하기 위해서는 많은 작업 부하를 감수해야 한다.

따라서 본 논문에서는 보다 효율적인 무선망에서의 강한 캐쉬 일관성 유지를 위해, 클라이언트의 접속단절상태를 서버에서 파악하여 클라이언트가 접속단절 상태에 놓였을 경우 서버가 전적으로 캐쉬 일관성 유지에 관련된 사항을 관리해줌으로써 불필요한 서버의 제어정보 전송을 줄이고, 클라이언트가 좀 더 빨리 사용자의 요청에 응답할 수 있도록 보완된 W-Lease 알고리즘을 제시한다

2. 기존 연구에 대한 고찰

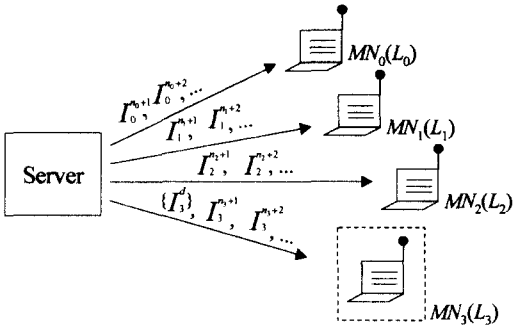
무선망에서의 캐쉬 일관성 유지에 대한 이전의 연구들은, 협소한 대역폭과 잦은 접속단절을 갖는 무선망의 특성을 고려하여 서버가 클라이언트에게 브로드캐스트를 통해 무효화 메시지를 전송하는 무효화(Invalidation)방식을 택해왔다[4,5]. 브로드캐스트를 이용한 무효화 방식을 사용하면, 효율적으로 캐쉬 일관성을 유지할 수 있다. 그러나 이 방식은 본질적으로 약한 캐쉬 일관성 유지 기법이다. 따라서 브로드캐스트를 이용한 무효화 방식은 무효화 메시지를 전파하는 각 주기 사이 동안 클라이언트가 자신이 가지고 있는 데이터의 캐쉬 일관성 여부를 확인할 수 있는 방법이 전혀 없다는 문제점을 갖고있다. 이러한 문제점을 해결하기 위해서 사용자에 대한 응답을 다음 브로드캐스트 주기까지 미루는 방법을 생각해 볼 수는 있으나, 이로 인해 평균적으로 응답시간이 길어지는 문제가 있다. 클라이언트로부터의 ACK를 완전히 배제하고 있는 것 또한 문제가 될 수 있다. 강한 캐쉬 일관성을 제공하기 위해서는 클라이언트로부터의 ACK가 필수적이다. 클라이언트로부터의 ACK는 서버가 보내준 무효화 메시지가 제대로 전달되었는지를 확인해주는 역할을 한다. 브로드캐스트를 이용한 무효화 방식에서는 ACK에 대한 고려를 하지 않고 있는데, 만약 서버로부터의 무효화 메시지가 중간에 유실된 후 이를 파악할 수 있는 방법이 없다면, 클라이언트는 사용자에게 잘못된 데이터를 제공하게 된다. 따라서 무선 환경에서 강한 캐쉬 일관성을 제공하기 위해서는 기존의 브로드캐스트를 이용한 무효화 방식과는 다른 새로운 방식이 요구되며, 이 방식은 무선 환경의 고유 특성에 잘 부합되는 것이어야 한다.

한편 유선 망에서의 강한 캐쉬 일관성 유지 보장에 관련된 최근의 연구성과 중 하나로 리스 알고리즘이 있다[3]. 리스 알고리즘에 따르면, 서버 측에서는 리스가 유효한 클라이언트들만 관리하면 되므로 작업부하를 감소시킬 수 있고, 클라이언트 측에서는 자신이 가지고 있는 캐쉬 데이터 중 리스가 유효한 것들은 사용자의 요청에 즉시 응답할 수 있어 응답시간을 줄일 수 있는 등, 성능향상을 도모할 수 있게 된다. 그러나 무선망에서는 캐쉬 일관성 유지에 대해 다른 양상을 갖는다. 무선망에서는 잦은 접속단절로 인하여, 캐쉬 일관성 유지에 어려움이 있다. 클라이언트는 매번 접속단절상태에서 벗어날 때마다 자신이 가지고 있는 캐쉬 데이터의 일관성 유지 여부를 확인하기 위하여 서버와 제어정보를 송수신하여야 한다. 자신이 접속단절에 빠진 동안에 어떠한 제어정보가 생성되었는지를 모르기

때문에, 클라이언트는 일단 자신이 가지고 있는 캐쉬 정보가 모두 불확실하다는 가정을 하게된다. 다시 말해서, 클라이언트의 캐쉬에 저장되어 있는 모든 유효한 리스를 가진 데이터들에 대해서, 접속단절 기간 중에 서버에 의해 갱신되었는지 여부를 일일이 확인을 해 주어야 한다. 이러한 과정이 접속단절 발생 시마다 되풀이된다면 네트워크 전반에 걸쳐 과중한 부담을 안겨주게 된다. 또한 서버는 접속 단절에 빠진 클라이언트들에 대해 소모적으로 캐쉬 무효화 메시지를 전송하게 되는데, 이 또한 접속단절의 빈도가 높아지면 높아질수록 서버에 심각한 성능저하를 가져오게 된다. 따라서 리스 알고리즘을 무선망에 적용시켜 강한 캐쉬 일관성 유지를 제공하기 위해서는 클라이언트의 접속단절 상태를 고려한 알고리즘상의 수정이 필요하다.

3. 제안하는 알고리즘

[그림 1]은 W-Lease의 개념을 그림으로 나타낸 것이다. $I_j^{n_j+k}$는 j 번째 이동 노드(Mobile Node)에 대한 무효화 메시지 중 이전까지의 전송(n_j) 이후로 k 번째 메시지를 나타내며, $\{I_j^d\}$는 j 번째 이동 노드의 접속단절 기간 동안의 무효화 메시지들의 집합을 나타낸다. 각 이동 노드들은 lease L을 가지고 서버에게서 자신에게 해당되는 무효화 메시지 $I_0^{n_0+k}, I_1^{n_1+k}, I_2^{n_2+k}, I_3^{n_3+k}$ 을 전송 받고 있다. 이때 MN_3가 무효화 메시지 $I_3^{n_3+1}$을 전송 받은 이후에 접속단절 상태에 들어가면 서버는 해당 호스트가 접속단절에 들어갔음을 파악하고 $I_3^{n_3+2}$부터는 따로 관리하여 집합$\{I_3^d\}$에 포함시킨다. 캐쉬 무효화 메시지를 전송하는 경우, 서버는 먼저 이 집합의 리스트를 검색하여, 전송 대상인 클라이언트가 이 집합에 속해 있을 경우, 해당 클라이언트를 메시지 전송대상에서 제외하고, 이 메시지를 따로 보관한다. 이후 MN_3가 재 접속하였을 경우 서버는 $\{I_3^d\}$의 내용을 전송한다. 이 때 서버는 각각의 메시지를 한번에 보내주어 각 메시지에 대한 ACK를 따로 받아들이기 보다, 한번만 받을 수 있게 하여 제어정보의 전송량을 최적화시킨다.



[그림 1] W-Lease 개요: 서버가 클라이언트(MN_3)의 접속단절 기간 동안 무효화메시지를 관리한다.

3.1 서버 알고리즘

[그림 2]는 서버 측 알고리즘의 주요 부분이다. 클라이언트(Client) 오브젝트는 각기 식별자(ID)와 해당 클라이언트가 접속단절 기간동안 갱신된 오브젝트들의 리스트(Updated ObjectList)를 담고 있으며, 서버에 접근한 적이 있는 클라이언

```
Server Side

- Server writes object o
  for all <client,leaseTimeout> ∈ o.clientList
   if (leaseTimeout > currentTime)
     sendingList ← sendingList ∪ client
  for all <client> ∈ sending_list
   if (client ∈ disconnectList) then
     client.updatedObjectsList ← client.updatedObjectsList ∪ {o}
          sendingList ← sendingList - {c}
  send(INVALIDATE, o.id) to all clients in sendingList
  T ← max(o.leaseTimeout, MSGTIMEOUT) + currentTime
    while (T ≥ currentTime) and (sendingList ≠ ∅) do
      receive (INVALIDATEACK, c.id, o.id) from c ∈ sendingList
      sendingList ← sendingList - {c}
      disconnectList ← disconnectList ∪ sendingList
      o.version ← o.version + 1
  write o

- Server re-establishes contact with client c,
   which is awaken from disconnection
  recoverDisconnectedClient(c)
   if (c.updatedObjectList ≠ ∅) then
     send to c with all <o.id> in c.updatedObjectList
   T ← currentTime + MSGTIMEOUT
     while (T ≥ currentTime) do
       if (receive (INVALIDATEACK, o.id) from c) then
         c.updatedObjectList ← ∅
   if(c.updatedObjectList ≠ ∅) then
     c.updatedObjectList ← ∅
     disconnectList ← disconnectList ∪ {c}

- Server grants lease for object o with o.id = objID
  receive(LEASEREQUEST, objID, version) from c
  let o be the object such that o.id = objID
  o.leaseTimeout ← currentTime + LEASETIMEOUT
  o.clientList ← o.clientList - {<client, X>}  // delete old leases
  o.clientList ← o.clientList ∪ {<client, o.leaseTimeout>}
  if(o.version > clientVersion) then
    send(LEASEGRANT, o.id, o.version, o.leaseTimeout, o.data)
  else if(o.version = clientVersion) then
    send(LEASEGRANT, o.id, o.version, o.leaseTimeout)
```

[그림 2] 서버 측 알고리즘

```
Client Side

- Client reads object o
  if (o.leaseTimeout < currnetTime) then
       read local copy of o
  else
       request lease for object o
       read local copy of o

- Client c requests lease for object o
  send(LEASEREQUEST, o.id, o.version) to server

- Client c is granted lease for object o
  receive(LEASEGRANT, o.id, o.version, o.leaseTimeout[, o.data]) from ser
  let obj be the object for which ID is obj.id = o.id
  obj.version = o.version
  obj.leaseTimeout = o.leaseTimeout
  [obj.data ← o.data]

- Client receives object invalidation message for object o
  receive(INVALIDATE, o.id) from server
  let obj be the object for which obj.id = o.id
  obj.leaseTimeout = -1; delete obj.data; obj.data ← NULL
  send(INVALIDATEACK, CLIENTID, obj.id) to server
```

[그림 3] 클라이언트 측 알고리즘

트들의 개수만큼 생성되어 유지된다. 이 리스트는 이후에 이 클라이언트가 재 접속하였을 경우 접속단절 기간동안의 제어정보를 보내 주기 위해, 캐쉬 무효화 메시지를 보관하는 목적으로 사용된다.

3.2 클라이언트 알고리즘

클라이언트는 서버에게 리스를 요청하고 데이터를 서비스 받으며, 서버에서 갱신이 발생하였을 경우 캐쉬 무효화 메시지를 전송 받는다. 사용자의 요청이 있을 경우, 자신이 가지고 있는 데이터의 리스가 유효한 경우 그대로 서비스하고, 그렇지 않은 경우 서버에서 가져다가 사용자에게 서비스한다. 접속단절 후에는 서버로부터 접속단절 동안의 무효화 메시지를 전송 받아 캐쉬 일관성을 지속적으로 유지해 나간다.

4. 성능 평가

4.1 실험 환경

본 논문의 시뮬레이션은 단일 서버-다중 클라이언트 모델을 기반으로 하며, 데이터 베이스는 서버만이 갱신 할 수 있고, 서

버와 클라이언트간에 오가는 메시지에 대한 요청은 클라이언트에서 비롯됨을 가정한다.

4.1.1 시뮬레이션 매개변수

데이터 베이스 안에는 1000개의 데이터 아이템이 있는데, 이들은 두 가지 부류로 나뉘어 진다. 하나는 핫 데이터 부분 집합이고 나머지 하나는 콜드 데이터 부분 집합이다. 핫 데이터 부분 집합은 데이터 아이템 집합의 1부터 50까지를 포함하며, 콜드 데이터 부분 집합은 데이터 베이스의 그 나머지 아이템들을 포함한다. 클라이언트가 핫 부분 집합에 접근할 확률은 높게(80%) 책정되어 있고, 콜드 부분 집합에 접근할 확률은 낮게(20%) 책정되어 있다. 이렇게 하는 이유는 시뮬레이션의 환경을 최대한 실제의 웹 접근 환경과 유사하게 하기 위한 것이다. 이상과 같은 내용을 정리하면, [표 1]과 같다.

[표 1] 시뮬레이션 매개 변수의 초기값

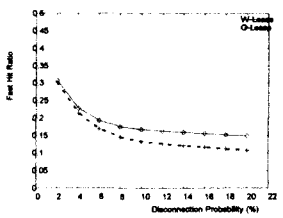
매 개 변 수	초 기 값
클라이언트의 수	100
데이터 베이스의 크기	1000
리스 기간의 크기	5000
시뮬레이션 시간	100,000
핫 데이터 아이템	1 to 200
콜드 데이터 아이템	데이터베이스의 나머지
핫 데이터 접근 확률	0.8
콜드 데이터 접근 확률	0.2
평균 요청 생성 시간	100
평균 업데이트 도착 시간	1 to 10000
접속단절 발생 확률	2 to 20 (%)
평균 접속 단절 시간	200 to 8000

4.1.2 시뮬레이션 모델의 구성

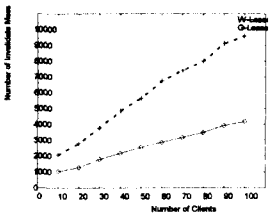
시뮬레이션 모델의 구성은 크게 단일 서버와 다중 클라이언트, 그리고 하나의 데이터베이스로 나뉘어 질 수 있다. 업데이트 생성 모듈은 데이터 베이스에 자리하고 있으며, 접속단절 생성 모듈과 데이터 요청 생성 모듈은 각 클라이언트마다 내재되어 있다. 따라서 모든 클라이언트들은 서로 다른 데이터 요청 패턴과 접속단절 발생 패턴을 가진다

4.2 실험 결과

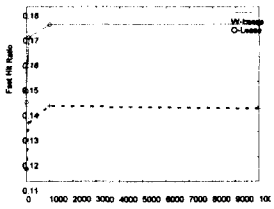
[그림 4], [그림 5], [그림 6], [그림 7]은 접속단절 발생확률, 클라이언트의 수, 평균 업데이트 도착 시간, 그리고 리스 기간 등의 값을 변화시키면서 알고리즘의 성능을 비교한 결과이다. [그림 4]는 접속단절 발생확률에 따른 캐쉬 적중률의 변화를 나타내고 있다. W-Lease 알고리즘에서는 각 클라이언트가 접속단절 상태에 놓였다가 재 접속하였을 경우, 서버가 접속단절 기간 동안의 캐쉬 무효화 메시지를 보관하였다가 전송해주기 때문에 클라이언트가 자신이 저장하고 있는 리스 정보를 항상 최신으로 유지할 수 있다. 따라서 사용자로부터의 요청이 있을 경우, 리스가 아직 만료되지 않은 데이터에 대해서 빠른 캐쉬 적중의 확률이 보다 높게 된다. [그림 5]은 클라이언트의 수를 증가시키면서 두 알고리즘의 무효화메시지의 수의 발생을 비교한 것이다. 원래의 리스 알고리즘은 클라이언트 수의 증가나 감소에 따라 항상 일정한 비율로 무효화 메시지 또한 증가하거나 감소한다. 그러나 W-Lease알고리즘은 서버의 관리를 통해 무효화 메시지를 줄일 수 있기 때문에, 원래의 리스 알고리즘보다 무효화 메시지의 수를 줄일 수 있으며, 이를 [그림 5]의 그래프를 통해 확인할 수 있다. [그림 6]은 평균 업데이트 도착 시간의 증가에 따른 캐쉬 적중률의 변화를 나타내고 있다. 평균 업데이트 도착 시간이 짧아지면 서버 쪽에서 갱신이 더욱 자주 일어나게 되는데, 이에 따라 캐쉬 무효화 메시지의 개수는 많아지고, 캐쉬가 더욱 자주 갱신되기 때문에, 캐쉬 적중률은 떨어지게 된다. 그러나 W-Lease알고리즘은 접속단절이 발생



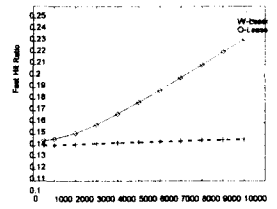
[그림 4] 접속단절 발생확률에 따른 캐쉬 적중률의 변화



[그림 5] 클라이언트 증가에 따른 캐쉬무효화메시지의 수



[그림 6] 평균 업데이트도착시간의 증가에 따른 캐쉬 적중률의 변화



[그림 7] 리스 기간의 증가에 따른 캐쉬 적중률의 변화

하더라도 클라이언트들이 항상 최신의 캐쉬 정보를 유지할 수 있으므로, 원래의 리스알고리즘 보다 높은 캐쉬 적중률을 가지게 된다. [그림7]은 리스 기간의 증가에 따른 캐쉬 적중률의 변화를 나타낸다. 리스 기간이 증가하면, 서버에 의해 캐쉬 일관성을 유지하게 되는 클라이언트들의 수가 늘어나므로, 이로 인해 서버의 작업부하는 커지지만 클라이언트는 별다른 요청 없이도 현재 자신이 캐쉬하고 있는 데이터가 유효한지 여부를 파악할 수 있기 때문에, 캐쉬 적중률은 높아지게 된다. W-Lease 알고리즘은 접속단절 발생 시에도 서버로부터 자신이 캐쉬하고 있는 데이터의 일관성 유지 여부에 대한 정보를 전달받을 수 있으므로, 원래의 리스알고리즘보다 더 높은 캐쉬 적중률을 가지게 된다.

5. 결론

본 논문에서 제안한 W-Lease 알고리즘은 무선 환경의 특징인 클라이언트의 접속단절상태를 고려하여 무선 환경에 적합하게 보완된 알고리즘으로, 서버가 접속단절 상태에 놓인 클라이언트들을 관리하여, 자신의 작업부하를 줄임은 물론, 제어 메시지 수의 감소, 클라이언트 응답시간의 감소 등, 여러 가지 측면에서 성능 향상을 도모하고 있다. 또한, 최대한 실제 환경과 유사하게 구성된 환경의 시뮬레이션을 통해 이러한 성능 향상에 대한 예측을 실제로 확인하였다.

6. 참고 문헌

[1] T. Imielinski, S. Viswanathan, and B. Badrinath, "Energy Efficient Indexing on Air: Organization and Access", IEEE Transactions on Knowledge and Data Engineering, 9(3):353-372, May/June 1997

[2] P. Barford, A.Bestavros, A. Bradley, and M. E. Crovella, "Changes in Web Client Access Patterns: Characteristics and Caching Implications", World Wide Web Journal, 1999

[3] Venkata Duvvuri, Prashant Shenoy, and Renu Tewari, "Adaptive Leases : A Strong Consistency Mechanism for the World Wide Web", IEEE INFOCOM 2000, 1999

[4] D. Barbara, and Tomaz Imielinski, "Sleepers and workaholics: Caching strategies for mobile environments", ACM SIGMOD, pages 1~12, 1994

[5] Qinglong Hu, and Dik Lun Lee, "Adaptive Cache Invalidation Methods in Mobile Environments", Cluster Computing pp 39-48, 1998