이동 컴퓨팅을 위한 토큰 기반 상호배제 알고리즘

하 숙정, 이 경숙, 배 인한

대구호성가톨릭대학교 전자정보공학부

A Token-Based Mutual Exclusion Algorithm For Mobile Computings

Sook-Jeong Ha, Kyeung-Sook Lee, Ihn-Han, Bae

Dept. of Computer Engineering, Catholic University of Taegu-Hyosung

요 약

이동 호스트를 갖는 이동 컴퓨팅 환경에서의 상호배제 알고리즘은 기존의 정적 컴퓨팅 환경과 여러 가지 다른 점이 있다 본 논문에서는 이동 컴퓨팅 환경에서 상호배제 문제를 효율적으로 해결하는 토큰 기반 상호배제 알고리즘을 제안하고, 이것의 성능을 메시지 송수신 비용으로 평가하였다.

1. 서론

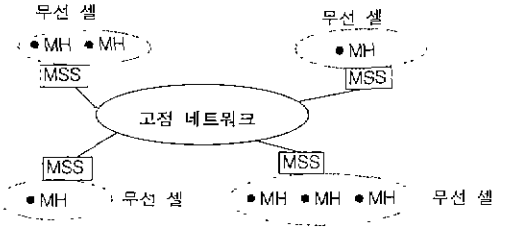
분산 알고리즘의 성능에 영향을 미치는 동기화 문제 중 대표적으로 고려되어야 할 요소로 상호배제 문제가 있다. 상호배제를 해결하기 위한 기존의 여러 알고리즘[1, 2, 3]은 정적 호스트로 구성된 모델에 기초를 두고 있다 그러나 최근 랩탑, 노트북과 같은 이동성을 가진 모빌 컴퓨터가 등장하여 이에 대한 사용이 급격히 확산되어 감에 따라 이를 기존의 분산 컴퓨팅 환경에 통합시키기 위한 연구가 이루어져 왔다[4]. 호스트가 이동할 수 있다는 것을 가정한다면 이러한 호스트는 임의의 시간에 임의의 다른 장소로 이동하므로 상호배제의 해결이 정적 호스트 모델보다 더욱 어려워지게 되고 이동 컴퓨팅 환경에 맞도록 재조정되어야 할 필요가 있을 것이다.

본 논문에서는 정적 호스트로 구성된 분산 환경에서의 Kumar[3]의 상호배제 알고리즘을 이동 호스트가 존재하는 이동 컴퓨팅 환경에 맞도록 수정한 토큰 기반 상호배제 알고리즘을 제안한다 본 논문의 구성은 다음과 같다 2장에서는 이동 컴퓨팅에 관한 개괄적 설명과 함께 본 논문에서 사용한 시스템 모델을 기술하고, 3장에서는 분산 상호배제에 대한 기존의 연구들을 살펴보고, 4장에서는 제안한 토큰 기반 상호배제 알고리즘을 설명하고, 5장에서는 제안하는 알고리즘의 정확성 및 성능을 평가한다. 그리고 끝으로 6장에서 결론을 맺는다

2. 이동 컴퓨팅

"이동"이라는 용어는 네트워크에 접속되어 있는 동안 이동할 수 있음을 내포하고 있다[4]. 네트워크에 접속되어 있는 동안 이동할 수 있는 호스트를 이동 호스트(MH)라고 하며 MH들과 직접적으로 통신할 수 있는 호스트를 모빌 지원국(mobile support station : MSS) 또는 고정 호스트라고 한다. MSS가 무선 통신을 통해 MH와 직접 통신할 수 있는 논리적 또는 지리적 수용 영역을 셀이라 한다 즉 MSS와 MSS의 셀 안에 위치하고 있는 MH간에만 직접 무선 통신을 할 수 있으며 이러한 MH를 MSS의 '로컬 MH'라 한다.

MH가 현재 셀에서 다른 셀로 이동할 때는 핸드오프 프로시져가 수행된다 이 때 MH는 이동한 셀의 새 MSS에게 GREETING 메시지를 통해 자신의 정보를 제공하며, GREETING 메시지를 수신한 MSS는 MH의 이전 MSS에게 DEREGISTER 메시지를 전송한다 DEREGISTER 메시지를 수신한 MSS는 REGISTER 메시지를 통해 MH에 대한 제반 정보를 새 MSS에게 전달한다.



(그림 1) 이동 컴퓨팅 환경 모델

3. 관련 연구

분산 상호배제 알고리즘에 관한 기존의 연구들을 살펴보면 다음과 같다 Kumar[3]는 노드들 간에 이동하며 토큰 역할을 하는 동기화 큐(SQ)를 정의하고 SQ를 수신한 노드만이 CS를 실행할 수 있는 토큰 기반 상호배제 알고리즘을 제안하였다. CS를 실행하려는 노드는 SQ의 목적지에 CS 실행 요청 메시지를 전송하며 요청 메시지는 목적지 큐(DQ)에 저장된다. 특정 노드에 SQ가 도착하면 DQ의 내용이 SQ로 이동되고 SQ가 공백이 될 때까지 SQ에 들어 있는 요청 노드에게 순서대로 토큰이 전달된다. Badrinath[4]는 MSS들이 논리적으로 링을 형성하여 토큰이 논리 링을 따라 순회하며, MSS가 자신의 셀 내의 MH들의 CS 실행 요청을 관리하기 위해 요청 큐와 인가 큐를 사용하는 토큰 기반 상호배제 알고리즘을 제안하였다. 특정 MSS에 토큰이 도착하면 요청 큐의 내용이 인가 큐로 옮겨지고 인가 큐의 선두에 있는 MH에게 토큰이 전달된다. 이 때 토큰을 전송할 MH가 이미 다른 셀로 이동한 경우에는 탐색 과정을 통해 새로운 위치를 찾아 토큰을 전달한다 Walter[5]는 ad hoc 네트워크로 알려져 있는 이동 무선 네트워크에서의 토큰 기반 상호배제 알고리즘을 제안하였다. 노드는 논리적 포인터, 패스(path) 포인터라는 두 개의 자료구조를 가진다. 각 노드는 자신의 바로 옆 즉 1-hop인 이웃에 대해서만 논리적 포인터를 통해 정보를 유지하며, 토큰의 위치는 패스 포인터를 통해

정보를 유지한다. CS 실행 요청은 패스 포인터를 따라 전송되어 토큰을 가진 노드에 도착하게 되고 토큰을 전달받으면 CS를 실행한다. 토큰을 전송할 노드가 이동한 경우 즉 1-hop 이웃이 아닌 경우에는 Find 메시지를 통해 새로운 위치를 찾아 논리적 링크를 변경된 물리적 링크에 맞도록 수정하고 토큰을 전달한다

4. 상호배제 알고리즘 설계

본 논문에서는 다음과 같은 이동 컴퓨팅 환경을 가정하고, Kumar의 토큰 기반 상호배제 알고리즘을 이동 컴퓨팅 환경에 맞도록 수정한다.

- 고정 네트워크는 단순화를 위해 MSS만으로 구성되며 MSS는 논리적인 링을 구성한다.
- 모든 호스트와 링크는 신뢰할 수 있으며 메시지간의 순서화를 위해 모든 MSS간에 논리적 타임스탬프[6]가 관리된다.

상호배제 알고리즘을 위해 MSS가 유지하는 자료구조는 다음과 같다

- TokenExist(초기값:FALSE) : 토큰을 갖고 있다면 TRUE이다.
- TokenPass(초기화:NULL) : P_Token을 소유한 채 이동한 MH를 위해 필요한 것으로 MH로부터 수신한 P_Token을 전송할 MSS를 지정한다.
- CLQ(Current Local Queue, 초기화:empty) : 로컬 MH들이 보낸 CS 실행 요청 메시지가 저장된다.
- NLQ(Next Local Queue, 초기화:empty) : MSS에 토큰이 도착했을 때 CLQ로 옮겨진다. 즉 토큰이 다음 링 순회를 통해 도착하였을 때 서비스를 받을 수 있는 요청들이 들어있다
- SQ(Synchronization Queue, 초기화:empty) : 토큰이 도착했을 때 P_Token을 전송받을 호스트의 요청이 저장되는 큐로서 MSS에 토큰이 도착하면 CLQ의 내용이 SQ로 옮겨진다.
- W_Time$_k$(초기화:0) : MH$_k$가 CS 실행을 요청하여 P_Token을 받기 위해 대기한 시간

MH가 유지하는 자료구조는 다음과 같다.

- TokenHold(초기화:FALSE) : P_Token을 갖고 있다면 TRUE이다.
- CSRequest(초기화:FALSE) : CS 실행을 요청하였다면 TRUE이며 P_Token을 얻으면 FALSE가 된다.

알고리즘의 개괄적인 부분은 다음과 같다.

(1) MH$_k$가 CS 실행을 요청하는 경우(Wireless Module)

MH$_k$는 자신이 속한 셀의 MSS$_i$에게 CS 실행 요청 메시지를 전송하고, MSS$_i$로부터 P_Token을 받으면 CS를 실행한 다음, 자신이 속한 셀의 MSS$_i$에게 P_Token을 반환한다. 그리고 핸드오프를 수행할 때 GREETING 메시지를 새로운 MSS$_j$에게 전송한다.

```
On requesting CS :
  CSRequest←TRUE
  Send REQUEST_k to local MSS_i
Wait for P_Token to arrive
On receiving P_Token :
  TokenHold←TRUE;
  CSRequest←FALSE;
  Execute CS;
  Send P_Token to local MSS_i;
  TokenHold←FALSE;
On moving to a new cell
  MH_k send GREETING(TokenHold, CSRequest) to MSS_j
  that is new local MSS;
```

(2) CS 실행 요청 메시지가 MH$_k$로부터 MSS$_i$에 도착한 경우 (Request_Receive Module)

MH$_k$로부터 CS 실행 요청 메시지가 도착하면, MSS$_i$는 그 요청 메시지에 타임스탬프를 부여하고 타임스탬프가 부착된 요청 메시지를 CLQ의 적절한 위치에 넣는다.

```
On receiving REQUEST_k
  Assign TS_k to REQUEST_k;
  Insert (REQUEST_k, TS_k) in CLQ;
```

(3) MSS$_i$에 토큰이 도착하는 경우(Token_Receive Module)

MSS$_i$에 토큰이 도착하면, MSS$_i$의 CLQ에 있는 요청 메시지들을 SQ로, NLQ의 요청 메시지들을 CLQ로 옮긴 후, SQ의 선두에 있는 MH$_k$에게 P_Token을 전송하고, MH$_k$로부터 P_Token이 도착하기를 기다린다. 이 과정을 SQ가 공백일 때까지 반복한다. MSS$_i$의 SQ가 공백이면 토큰을 논리 링에서의 다음 MSS인 MSS$_{i+1}$에게 전송한다.

```
On receiving TOKEN :
  TokenExist←TRUE;
  Refill SQ from CLQ;
  Refill CLQ from NLQ;
  While SQ is not empty
    Send P_Token to the mobile host MH_k that is at the head of SQ;
    Delete (REQUEST_k, TS_k) from SQ;
    Wait for P_Token from MH_k;
  EndWhile
  Send TOKEN to MSS_i+1 that is the next MSS in logical ring;
```

(4) MSS$_i$가 MH$_k$ 또는 MSS$_j$(i≠j)로부터 P_Token을 받은 경우 (Ptoken_Receive Module)

MSS$_i$가 MH$_k$ 또는 MSS$_j$로부터 P_Token을 받으면, MSS$_i$는 이 P_Token이 자신의 것인지 아니면 다른 MSS의 것인지를 판단하기 위해 TokenPass를 조사한다 TokenPass가 NULL이 아니면 그 MH$_k$는 P_Token을 갖고 핸드오프를 하였다는 것을 나타낸다. 따라서 P_Token을 TokenPass에 지정된 MSS로 전송한다

```
On receiving P_Token :
  If TokenPass is not NULL
    Send P_Token to TokenPass;
    TokenPass←NULL;
  Endif
```

(5) MH$_k$가 MSS$_i$에서 MSS$_j$로 핸드오프하는 경우(Wired Module)

MH$_k$가 상호배제를 요청하고 대기중인 상태로 핸드오프한다면, MH$_k$가 새 MSS인 MSS$_j$에게 TokenHold, CSRequest를 포함하는 GREETING 메시지를 전송하고, MSS$_j$는 이전 MSS인 MSS$_i$에게 CSRequest를 포함하는 DEREGISTER 메시지를 전송한다. MSS$_i$는 MH$_k$가 어떤 큐에 들어 있는지를 알기 위해 NLQ, CLQ, SQ로부터 (REQUEST$_k$, TS$_k$)를 찾고, (REQUEST$_k$, TS$_k$, QType)를 포함하는 REGISTER 메시지를 MSS$_j$로 전송한다 요청 MH$_k$의 대기시간(W_Time$_k$)을 이용하여 CLQ와 NLQ에 있던 MH$_k$가 핸드오프하였을 때는 대기시간이 토큰의 순회시간(δ_i)보다 클 경우에는 CLQ에 있던 것은 SQ로, NLQ에 있던 것은 CLQ로 옮긴다. MH의 요청이 SQ에 있다는 것은 MSS에 토큰이 있을 때 이동한 한 것이다. 따라서 새 MSS에서도 역시 SQ로 옮김으로써 꼭 P_Token을 받을 수 있도록 한다

```
On receiving GREETING(TokenHold, CSRequest) :
  If TokenHold is TRUE
    TokenPass←MSS_i;
    Send DEREGISTER() to MSS_i;
    Wait for REGISTER() from MSS_i;
  Else If CSRequest is TRUE
    Send DEREGISTER(CSRequest) to MSS_i;
    Wait for REGISTER(REQUEST_k, TS_k, QType) from MSS_i;
  Endif
On receiving DEREGISTER message :
  If CSRequest is TRUE
    Search for from SQ, CLQ and NLQ;
    Send REGISTER(REQUEST_k, TS_k, QType) to MSS_j;
    Delete (REQUEST_k, TS_k) from the Queue;
```

```
  Else
    Send REGISTER() to MSS;
  Endif
On receving REGISTER message :
  IF (REQUEST_k, TS_k, QType) is not NULL
    If QType is SQ
      Insert (REQUEST_k, TS_k) in SQ in increasing order with timestamp;
  Else
    If W_Time_k > δ_l
      Case QType of
        CLQ : Insert (REQUEST_k, TS_k) in SQ in increasing order with
              timestamp;
        NLQ   Insert (REQUEST_k, TS_k) in CLQ in increasing order
              with timestamp;
      Endcase
    Else
      Insert (REQUESTk, TS_k) in NLQ in increasing order with
              timestamp
  Endif
```

5. 평가

상호배제 알고리즘의 정확성은 상호배제 보장, 교착상태 방지, 무한연기 회피로 평가된다. 본 논문에서 제안하는 상호배제 알고리즘은 CS의 실행 권한을 부여하는 토큰이 단 한 개 존재하며 토큰을 가진 호스트가 CS 실행을 마친 후에만 다른 특정 호스트에게 토큰이 전달되므로 두 개 이상의 호스트가 동시에 토큰을 가지는 경우는 발생하지 않으므로 상호배제를 보장한다. 토큰은 MSS들로 구성된 논리적 링을 따라 차례로 전달되며, 호스트와 링크에는 결함이 없는 것으로 가정하였으므로 토큰의 유실로 인한 교착상태는 발생하지 않는다. 토큰을 가진 MSS의 SQ가 공백이 되면 논리적 링의 다음 MSS로 토큰이 자동으로 전달되므로 토큰이 특정 MSS에 계속 머무르고, CS의 요청을 인지하지 못하여 시스템이 무한 대기에 빠지는 일은 발생하지 않는다. CLQ나 NLQ에 있던 요청을 대기시간을 고려하여 SQ나 CLQ로 이동시킴으로써 요청 MH가 계속적인 이동으로 인해 NLQ에만 들어가게 되어 무한정 대기하는 경우는 발생하지 않는다.

알고리즘의 성능은 메시지 전송으로 인한 통신 비용으로 평가한다. 임의의 두 MSS간에 점대점 메시지 전송 비용은 C_{fixed}, 무선 링크를 통한 MH와 로컬 MSS간의 메시지 전송 비용은 $C_{wireless}$, 그리고 이동한 MH의 새로운 MSS를 찾아서 메시지를 전송하는데 드는 비용은 C_{search}라고 할 때 Badrinath의 알고리즘과 제안 알고리즘의 성능을 평가하면 표 1과 같다. 여기서 ℓ은 MH의 평균 이동 회수, t는 토큰을 갖고 이동한 요청 호스트의 평균 개수, m은 토큰을 받지 않고 이동한 요청 MH의 평균 개수, 그리고 k는 CS 실행을 요청한 호스트의 수이다.

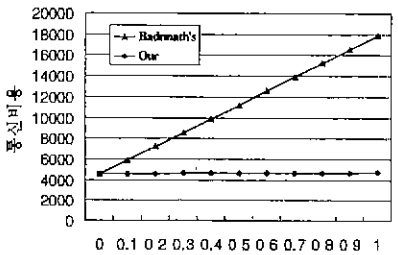
<표 1> Badrinath와 제안한 알고리즘의 성능 비교.

성능 / 알고리즘	통신비용
Badrinath's	$(k-t-m)\times 3C_{wireless} + t\times(3C_{wireless}+C_{search}) + m\times(3C_{wireless}+C_{fixed}+C_{search}) + N_{MSS}\times C_{fixed}$
Our	$(k-t)\times 3C_{wireless} + t\times(3C_{wireless}+C_{fixed}) + N_{MSS}\times C_{fixed}$

표 2와 같은 이동 컴퓨팅 환경에서 본 논문에서 제안하는 알고리즘과 Badrinath의 알고리즘의 성능을 시뮬레이션을 통하여 평가한다. Badrinath의 알고리즘에서 위치관리는 기본 IS-41(Interim Standard 41) 스킴[8]에 기반한다고 가정한다. IS-41 스킴에서의 탐색비용(C_{search})은 2A+4B이다. 여기서 A는 MSS와 등록 영역내의 LS간의 위치 질의/응답 메시지를 전송하는 평균비용이고, B는 어떤 위치 서버와 홈 위치 서버간의 위치 질의/응답 메시지를 전송하는 평균비용이다. 시뮬레이션 결과, Badrinath의 알고리즘은 MH의 이동율이 증가함에 따라 통신비용이 선형적으로 증가하지만 본 논문에서 제안하는 알고리즘은 MH의 이동율에 관계없이 일정한 통신비용이 발생하여 성능이 우수함을 알 수 있다(그림 2).

<표 2> 시뮬레이션 환경

파라미터	값
N_{MSS}	50
N_{MH}	500
MH의 CS 요청율	0.3
MH의 이동율	0.0~1.0
토큰을 소유한 MH의 이동율	0.02
C_{fixed}	2 *ms*
$C_{wireless}$	10 *ms*
C_{search}	35 *ms*
l	2



(그림 2) MH의 이동율에 따른 통신 비용

6. 결론

본 논문에서는 Kumar의 상호배제 알고리즘을 이동 컴퓨팅 환경에 맞도록 수정한 토큰 기반 상호배제 알고리즘을 제안하였다. CS 실행을 요청한 후 다른 셀로 이동한 MH에 대해서는 핸드오프 과정에서 이전 MSS에 저장된 요청 메시지를 새로운 셀의 MSS로 옮겨 새 위치에서 토큰을 기다리도록 하였다. 이처럼 이동한 MH에게 토큰을 전달하기 위해 MH의 이동 위치를 찾아내어 토큰을 전달하는 방법대신 이동한 셀에서 대기하도록 함으로써 메시지 전송 비용을 감소시켰다. 그러나 토큰을 요청하고 이동함으로써 다음 링 순회시에 토큰을 받게 되므로 대기 시간은 약간 증가하였다 향후 연구 과제는 이동 컴퓨팅 환경에서 링크와 호스트의 결함 감내 기능을 갖는 상호배제 알고리즘과 인가 기반 상호배제 알고리즘의 설계이다.

참고문헌

[1] B A Sanders, "The Information Structure of Distributed Mutual Exclusion Algorithms", ACM Transaction on Computer Systems, Vol 5, No. 1, pp. 284-299, August 1987

[2] M Maekawa, "A Sqrt(N) Algorithm for Mutual Exclusion in Decentralized Systems", ACM Trans Computer Systems, Vol 3 No 2. pp. 145-159, May 1985.

[3] V. Kumar, J. Place, and G Yang. "An Efficient Algorithm for Mutual Exclusion Using Queue Migration in Computer Networks", IEEE Trans on Knowledge and Data Engineering, Vol. 3, No. 3 pp 380-284, Sept 1991.

[4] B. R Badrinath, Arup Acharya, and Tomasz Imielinski, "Structuring Distributed Algorithms for Mobile Hosts", The 14th International Conference on Distributed Computing Systems, pp 21-28, 1994

[5] Jennifer Walters and Savita Kini "Mutual Exclusion on Multihop, Mobile Wireless Networks", Texas A&M University, Technical Report, p. 26, Dec 1997

[6] L. Lamport, "Time, Clocks, and the Ordering of Events in Distributed System", Comm of the ACM, Vol 21, No 7, pp 558-565, July 1978