순수 P2P 환경을 위한
이동 에이전트 기반 자원 검색 기법

김인숙[0], 김문정, 엄영익
성균관대학교 정보통신공학부 분산시스템연구실
{easy[0], tops, yieom}@ece.skku.ac.kr

Mobile Agent Based Discovery Mechanism for Pure P2P Environments

In-suk Kim[0] Moon Jeong Kim Young Ik Eom
Distributed Systems Lab., School of Information and Communication Engineering, Sungkyunkwan University

요 약

최근 인터넷의 급속한 성장과 초고속 통신망의 구축으로 인하여 다양한 멀티미디어 서비스들이 제공되고 있다. 그러나 현재 대부분의 멀티미디어 서비스들은 클라이언트/서버 모델을 기반으로 구축되어 있기 때문에, 중앙 서버로의 과도한 부하가 집중되는 문제점을 가지고 있다. 본 논문에서는 순수 P2P(Pure Peer-to-Peer) 환경으로 구축된 멀티미디어 서비스에서의 자원 검색 기법을 제안한다. 제안 기법은 자율성과 이동성을 지닌 이동 에이전트(mobile agent)를 기반으로 자원의 검색을 수행하여, 기존의 순수 P2P 환경에서의 검색 기법의 문제점을 해결한다. 또한, 시나리오를 통하여 제안 기법의 구체적인 동작 과정에 대해 알아본다.

1. 서 론

최근 인터넷의 급속한 성장과 초고속 정보통신망의 구축은 인터넷이 다양한 멀티미디어 서비스를 제공할 수 있게 하였다. 그러나 현재 대부분의 멀티미디어 서비스 환경은 몇 개의 대용량 서버를 구축하여 서비스를 제공하는 클라이언트/서버 모델을 택하고 있다. 그러나 기존 클라이언트/서버 모델은 서버 집중식으로 트래픽 집중의 한계 때문에 어려움을 겪고 있다. 이를 해결하기 위한 노력의 일환으로 P2P 컴퓨팅 기술이 각광을 받고 있는데, 이는 기존의 문제점을 클라이언트 상호간 분산 · 협력이라는 새로운 개념으로 풀어내려는 시도이다.

본 논문은 멀티미디어 서비스를 제공하는 서버들이 순수 P2P 환경하에서 이동 에이전트의 자율성과 이동성을 이용하여 클라이언트에 의해 요청받는 자원을 효율적으로 찾는 검색 메커니즘을 제안한다. 이 제안기법을 사용할 경우, 중앙서버를 이용한 다른 검색방식에서 나타나는 부하의 집중 현상을 제거할 수 있다. 또한, 기존의 순수 P2P 환경에서 검색 기법에 이동 에이전트의 사용으로 자원 검색 정보의 유실을 방지한다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 순수 P2P에서의 기존의 자원 검색 기법과 이동 에이전트에 대해 알아본다. 3장에서는 본 논문에서 제안하는 이동 에이전트 기반의 자원 검색 기법을 제시하고, 시나리오를 통해 제안 기법의 구체적인 동작 과정에 대해 알아본다. 마지막으로, 4장에서는 결론 및 문제점들을 지적하고 향후 연구 과제에 대해서 기술한다.

2. 관련연구

2.1 순수 P2P 환경에서의 검색 기법

P2P 컴퓨팅은 시스템들 사이에서 직접적인 교환을 통해 컴퓨터 자원들과 서비스를 공유하는 것이다. P2P 컴퓨팅 모델은 구성에 따라 크게 두가지로 나눌 수 있다. 중앙의 서버 유무로 구별되는 두가지 컴퓨팅 모델은 순수 P2P 컴퓨팅 모델과 혼합형 P2P 모델로 불리우고 있다[1].

순수 P2P는 자원 검색을 위해 너비 우선 탐색(breadth first search : Gnutella), 깊이 우선 탐색(depth first search : Freenet)등의 일반적인 검색 기법을 따른다. 이중 순수 P2P의 대표적인 예인 Gnutella의 경우, 기본적으로 BFS를 따르고 이를 위한 패킷의 전송은 검색 속도가 가장 빠른 broadcast를 사용한다. 그러나 순수 P2P에서 broadcast를 사용할 경우, 검색 결과가 요청이 발생한 곳으로 돌아갈 경로(자원 검색을 위해 거쳐온 경로로 검색 결과가 되돌아감)의 노드가 연결이 끊겼을 경우 이 검색 결과를 전달할 방법이 없다. 이런 문제점을 보안하기 위해 강력한 백본망을 통해 검색을 하도록 하는 KaZaA가 등장하였다[2][3][4][5].

2.2 이동 에이전트

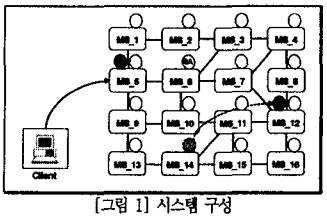
이동 에이전트란 컴퓨터 네트웍에서 사용자를 대표하며 노드에서 노드로의 자율적인 이동이 가능한 프로그램으로써 자신의 판단에 의하여 컴퓨터 시스템을 이동하며 작업을 수행한다. 이동 에이전트의 이점으로는 네트워크의 트래픽 감소, 비동기적인 상호 작용, 부하 균형, 서비스의 분산 및 병렬 처리 등이 있다[6][7].

3. 제안 기법

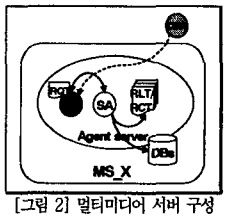
본 논문에서 제시하는 시스템의 구성과 각각의 자료 구조, 알고리즘, 동작 시나리오에 대해 살펴보겠다.

3.1 시스템 구성

본 논문에서 제안하는 기법의 시스템 모델은 그림 1에서 보인다.



[그림 1] 시스템 구성



[그림 2] 멀티미디어 서버 구성

멀티미디어 서버간의 구성은 순수 P2P이고 각 서버들은 물리적으로 가깝게 이웃한 서버의 위치를 서로 알고 있다고 가정한다. 클라이언트가 임의의 서버에 접속하여 서비스를 요청하게 되면 요청을 받은 서버가 이동 에이전트를 생성하여 자원의 위치를 검색한다. 각 멀티미디어 서버(MS)의 구성은 그림 2에서 보인다.

그림 2에서와 같이 MS(Multimedia Server)는 파일들, agent server, SA(Stationary Agent), MA(Mobile Agent), RCT(Resource Cache Table), RLT(Resource Local Table), RQT(Resource Query Table) 등으로 이루어진다.

멀티미디어 자원을 저장하는 서버는 이동 에이전트(MA)가 동작할

수 있는 환경(agent server)을 제공하여 SA와 MA가 자원에 대한 정보를 주고받을 수 있도록 한다.

SA는 RLT를 통해 local 상에 위치한 자원, 그리고 다른 서버 상에 존재하는 동일한 자원 위치 정보를 유지한다. 또한 SA는 RCT를 통해 해당 서버에 최근 요청이 들어온 자원 위치 정보를 유지한다.

MA는 클라이언트의 요청을 받은 서버에 의해 생성되어, 요청에 대한 RQT를 작성해 해당 자원을 검색한 후 사용자에게 결과를 통보한다.

3.2 자료 구조

본 논문에서 제안하는 기법에서 사용된 정보 테이블들의 자료구조는 그림 3에서 보인다.

RLT	RN	RT	MS	Mkr	Mky	Rloc	Rsz	Ub	Ut	Vb	SR	Co
RCT	RN	RT	MS	Mkr	Mky	RSip	SR					
RQT	RN	RT	MS	Mkr	Mky	RSip	Rsz	TTL	Cip			

[그림 3] 테이블의 구조

그림 3에서 RN(Resource Name), RT(Resource Type), MS(Main Star), Mkr(Maker), Mky(Make year)는 RLT와 RCT, RQT에 공통인 필드로서 사용자가 자원 요청시 키워드로 사용된다. RSip(Resource Server ip)는 자원을 보유한 서버의 ip, Rsz(Resource size)는 사용자에게 검색 결과로 돌려줄 항목이다. RLT의 Rloc(Resource location)는 자원 보유 서버 상 자원의 물리적인 위치를 나타낸다. Ub(Update bit), Ut(Update time), Vb(Valid bit), Co(Cowork)는 자원 정보의 변경시에 사용된다. SR(Similar Resource)은 해당 자원과 동일 자원을 보유한 서버의 IP 목록이다. RQT의 TTL(Time To Live)과 Cip(Client ip)는 사용자에게 검색 결과를 돌려주기 위해 사용된다.

3.3 알고리즘

본 장에서는 자원의 정보 관리, 유지와 검색을 처리하기 위한 알고리즘들을 소개하고 있다.

[알고리즘 1] MA 알고리즘

```
copy & send CMA to neighbor;
request SA resource with keywords;
receive result from SA (resource location info);
if (resource location info is in RLT){
   inform resource location to client;
   copy & send CMA (location in SR);
}
else if (resource location info in RCT)//
      copy & send CMA (location in RCT);
while (User_request_terminate is not happen) {
   if (CMA_return) {
      while (RLT or RCT's SR is not full) {
         allow communicate with SA;
      }
      don't allow communicate with SA;
   }
   else wait for CMA_return;
}
request SA to send terminate message;
terminate;
```

알고리즘 1은 사용자 요청의 발생부터 종료까지 검색을 총괄하는 MA의 동작 과정을 보인다. 사용자가 자원 요청 시, 요청을 받은 MS 상에 MA가 생성된다. MA는 자신을 복제하여 이웃한 서버들에게 브로드캐스트한다. MA는 로컬 상의 SA가 유지하는 테이블에서 해당 자원 정보를 검색하여 사용자에게 공지하고, 자원이 위치하는 서버로 자신을 복제하여 보낸다. 이 MA는 사용자의 검색 완료까지 다른 서버로 전송되었던 복제 MA(CMA)들이 돌아오기를 기다린다. 돌아온 CMA들은 먼저 SA에게 검색 결과를 알리기 위해 MA에게 승인을 받는다. MA는 사용자의 검색 완료 시 SA에게 CMA들이 종료하도록 다른 SA들에게 종료 메시지 전송을 요청하고 종료한다.

[알고리즘 2] CMA 알고리즘

```
send req# to SA;
if (req# is not valid)
   terminate;
else {
   copy & send CMA (neighbor);
   request SA resource with RQT's info;
   receive result from SA (resource location info)
   if (resource location info is in RLT) {
      inform resource location to client;
      copy & send CMA (location in SR);
   }
   else if (resource location info in RCT)
      copy & send CMA (location in RCT);
   else   //no location info in RCT
      terminate;
}
return home;
ask communication's validation; // communication with SA
if (communication is valid)
   report result & terminate;
else
   terminate;
```

알고리즘 2는 자원 검색을 위해 복제된 CMA의 동작 과정을 보인다. 다른 서버에서 복제되어 이동된 CMA는 현 서버 상에서 자신이 유효한 지 SA에게 묻는다. req#로 유효 여부를 판단하는데, TTL값(한 홉을 지날 때마 복제될 때 하나씩 감소)이 0이 아니고, 서버에 같은 req#를 가진 MA가 방문하지 않았고(동일한 MA에 의해 복제된 CMA들은 같은 req#를 가짐), CMA 종료 메시지가 SA에게 미도착 시 유효이다. 이를 위해 각 서버에 SA가 관리하는 blackboard라는 테이블을 둔다. 이 테이블에는 방문하는 MA들의 req#를 기록해 유효 검사 시 사용한다. CMA는 유효성을 인정받으면 자신을 복제하여 이웃한 서버로 보낸 후, SA에게 자원의 검색을 요청한다. 자원 위치 정보가 있으면 사용자에게 공지하거나 자원이 있는 서버로 자신을 복제하여 보낸 후, MA가 생성되었던 서버(home)로 돌아가 home의 SA에게 검색 결과를 공지한 후 종료한다. 만약 위치 정보를 못 찾았을 경우는 해당 서버 상에서 종료한다.

[알고리즘 3] SA의 자원 요청 처리 알고리즘

```
receive message;
if (message is terminate message) //receive message from other SA
   record to blackboard;
if (message is req#) { //receive message from CMA for validation
   search blackboard;
   notify to CMA;
   record req# of CMA to blackboard;
}
if (message is resource request) //receive message is resource request
   notify resource info in RLT, RCT;
if (message is terminate message) //receive message from MA
   broadcast terminate message to SA;
```

알고리즘 3은 SA가 자원 요청에 참여하는 동작 과정을 보인다. SA가 메시지를 받으면, 이를 분석해서 각 메시지를 구분한다. 이런 메시지에는 SA, CMA, MA, 다른 SA로부터 오는 메시지가 있다. 클라이언트의 자원 검색 완료에 의한 CMA 종료 요청을 받은 SA가 다른 SA들에게 종료 메시지를 전송하는 동작 과정, CMA의 유효성 여부 검사 요청과 자원 검색 요청에 대해 처리 결과를 넘겨주는 SA의 동작 과정을 보인다.

알고리즘 4-1과 4-2는 SA가 RLT와 RCT를 유지/관리하기 위한 동작 과정을 보인다. SA는 로컬의 자원의 추가/삭제를 모니터링하여 RLT에 작성하고 SA간 RLT 동기화를 위해 주기적으로 RLT의 변경 여부를 검색한다. 자원 추가의 경우, RLT의 SR필드를 채우기 위해서 사용자의 자원 요청 발생시와 동일하게 로컬의 MS에게 추가된 자원의 요청을 발생시킨다. CMA들이 검색 결과(동일 자원이 위치하는 MS의 위치 정보)를 가지고 오면 RLT의 SR필드를 작성해서 해당 자원이 이 서버에도 존재함을 SR필드에 존재하는 서버의 SA들에게 반영한다

[알고리즘 4] SA의 자원 정보 관리 및 유지를 위한 알고리즘

```
1. monitoring local resource
monitor resource add/remove;
if (resource add is find) {
   write resource info to RLT;
   draw Ub, Ut, vb for add;
   request resource to local MS;  //MA is create for resource search
   receive result from CMA (resource location info); //search result
   draw RLT's SR & release Vb;
}
else
   draw Ub & Ut for remove; //resource remove
2. check RLT's change regular
check RLT's Ub regular;
if (find change in Ub) {
   if (Ub is add) {
      if (Vb is valid)
         notify resource's add to servers in RLT's SR;
   }
   else {
      notify resource's remove to servers with Co filled in RLT's SR);
      remove resource info in RLT;
   }
}
3. receive message from other SA
receive message from other SA;
if (message is resource's add notification) {
   if (RLT's SR is not full)
      reflect & notify to the SA; //to message sender SA
}
if (message is response of add notification) {
   if (RLT's SR is not full)
      check Co in RLT;
}
if (message is resource's remove notification)
   remove resource info in RLT's SR;
4. receive message from CMA
receive message from search result;
if (result's resource is in RLT) {
   if (RLT's SR is not full)
      add to RLT's SR;
   else
      table is full message to MA;
}
else if (RCT is not full)
   add to RCT's SR;
else
   table is full message to MA;
```

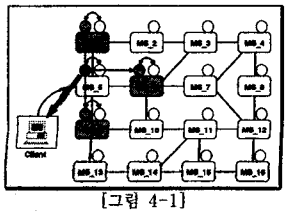
3.4 동작 시나리오

다음 시나리오는 클라이언트가 요청하는 자원이 MS5, MS16, MS4에 있고 클라이언트는 MS5에게 자원을 요청하는 경우를 보인다.

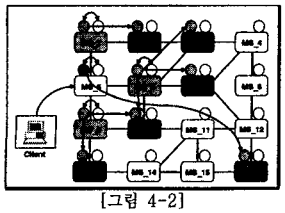
그림 4-1과 같이 MS5에서 MA가 생성되고 MA는 이웃한 서버들에게 자신을 복제하여 보낸다.

MS5 상의 MA는 SA에게 자원을 요청하고 SA는 자신의 RLT, RCT를 검색결과를 MA에게 알린다. MA는 로컬에 자원이 있음을 보고 사용자에게 MS5의 위치 정보와 자원에 대한 간략한 정보를 클라이언트에게 보낸다. MA는 RLT를 통해 알아낸 동일 자원을 가지고 있는 다른 MS(MS16)로 CMA를 복제하여 보낸다.

이웃한 MS로 복제된 CMA들은 자신이 도착한 각 MS 상에서 자신이 유효한지를 먼저 체크한다. 유효하다면 역시 이웃한 서버로 자신을 복제하여 보낸 후, 해당 MS의 SA에게 자원을 요청한다. CMA가 중복되어 유효하지 않은 경우, 해당 CMA가 종료된 것을 MS_10상의 CMA를 통해 볼 수 있다. 이 과정을 그림 4-2에서 보인다.



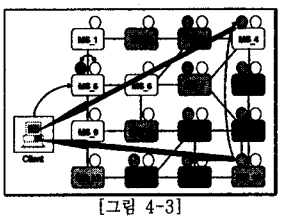
[그림 4-1]



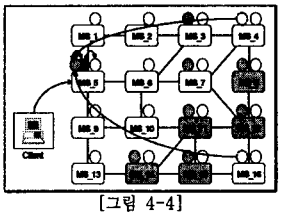
[그림 4-2]

MS16으로 간 CMA는 MS16의 SA에게 자신이 이 MS에서 유효한지 확인 후 자신을 복제해 이웃한 서버들에게 보낸 후 SA에게 자원을 요청한다. SA가 해당 자원이 로컬에 존재함을 CMA에게 알려주고, CMA는 바로 MS16의 위치 정보와 자원에 대한 간략한 정보를 클라이언트에게 보낸후, MS5에서 MA와 같이 RLT상의 정보를 보고 동일 자원을 보유한 MS4에게 CMA를 복제하여 보낸다. MS4에 도착한 CMA는 MS16에서의 CMA와 동일한 과정을 거친다. 한편, 자원이나 자원의 위치 정보를 보유하고 있지 않는 서버로 복제된 CMA들은 이웃한 서버로 자신을 복제하여 보낸 후, 자원 정보가 없음을 발견하면 해당 MS상에서 종료한다. 이 과정을 그림 4-3에서 보인다.

사용자에게 정보를 보낸 후 MS16상에 있던 CMA는 home인 MS5로 돌아와서 MA에게 검색 결과를 반영할 테이블 공간이 있는 지를 확인한 뒤, SA에게 MS16상에 자원이 있음을 보고하고 종료한다. SA는 해당 자원 위치 정보가 RLT 상에 있는 것을 확인하고 Ut를 갱신한다. MS4상의 CMA역시 MS16과 동일한 과정을 거친다. 이 과정을 그림 4-4에서 보인다.



[그림 4-3]



[그림 4-4]

마지막으로 사용자가 MS5, MS16, MS4상의 위치 정보를 받고 자원을 다운받을 서버를 선택해서 자원 검색을 종료하면 MS5상의 MA는 SA에게 req#와 함께 종료를 요청 후 종료한다. SA는 다른 SA들에게 req#에 대한 종료 요청을 브로드캐스트한다. 남아있던 MA들은 다른 MA로 이동해서 자신이 유효한 지 확인을 하는 과정에서 종료한다.

4. 결론 및 향후 과제

본 논문에서는 순수 P2P로 구성된 멀티미디어 서버 환경에서 이동 에이전트 기반의 검색기법을 제안하고 알고리즘과 간단한 동작 시나리오를 통해 이를 살펴보았다. 이동 에이전트의 이동성과 자율성을 이용하여 순수 P2P 상의 자원 검색 정보의 유실을 막도록 하였다. 빠른 검색을 위해서 이웃한 서버와 해당 자원을 소유한 서버로 MA를 복제하였고, MA 복제로 인한 트래픽 증가 억제를 위해 TTL과 blackboard를 사용했다. 또한 빠른 응답 속도와 home의 부하 감소를 위해 검색결과는 자원이 발견된 서버 상의 CMA가 직접 클라이언트에게 전송하도록 하였다. 그러나 이 논문에서 제시한 모델은 특정 자원에 대한 사용자의 요구가 쇄도할 경우 해당 자원을 보유한 서버에 걸리는 과부하는 막을 수 없다. 향후 각 멀티미디어 서비스를 제공하는 서버들 간에 멀티미디어 데이터를 고르게 분산시키는 기법이 제안되어야 하겠다.

참고 문헌

[1] White Paper: Intel Corp. "Peer to Peer Computing: P2P file-sharing at work in the Enterprise," Mar. 2001.

[2] M.A. Jovanovic, F.S. Annexstein and K.A. Berman, "Scalability issues in large peer to peer networks - A case study of Gnutella", Tech. Report, Univ. of Cincinati, 2001.

[3] Beverly Yang and Hector Garcia-Molina, "Improving search in peer-to-peer networks," ICDCS 2002, pp.5-14

[4] G. Kan. Gnutella. In A. Oram, editor, Peer-to-Peer: Harnessing the Benefits of aDisruptive Technology, chapter 8. O'Reilly, Mar. 2001.

[5] http://matrix.netsoc.tcd.ie/~neo/4ba2/p2p/

[6] K. Rothermel and R. Popescu-Zeletin (eds.), "Mobile Agents," 1st Int. Workshop on Mobile Agents (MA'97) LNCS 1219, Springer-Verlag, pp. 123-135

[7] Vu Anh Pham and Ahmed Karmouch, Mobile Software Agents: An Overview, Vol. 367, IEEE Communication Magazine, July, 1998.