순수 P2P 네트워크 환경에서 에이전트 이주를 위한 자원 예약 기반 동적 부하 균형 기법

김 경 인[†] · 김 영 진[††] · 엄 영 익[†††]

요 약

이동 에이전트는 자율성을 가지고 비동기적으로 수행이 가능한 개체로, 네트워크상의 여러 호스트들을 이동하면서 사용자를 대신해 특정한 작업을 수행하고 다른 에이전트들과 상호 협력할 수 있는 프로세스로 정의된다. 현재 이동 에이전트는 전자 상거래, 이동 통신, 병렬처리, 정보의 검색 및 복구 등 여러 분야에서 사용된다. 그러나 순수 P2P 환경에서 컴퓨팅 자원을 사용하고자 하는 이동 에이전트들이 이주할 피어의 가용한 자원을 고려하지 않고 이주할 경우, 이주된 에이전트로 인해 해당 시스템의 부하가 증가 된다는 문제점이 발생하게 된다. 이러한 문제점을 해결하기 위해 본 논문에서는 각 피어들에 상주하는 자원 관리 에이전트를 이용하여 부하 정보를 모니터링 하고 이주할 에이전트와 목적지 피어를 결정하게 된다. 이동 에이전트가 이주하는 과정에서 특정 피어의 자원이 예약 선점 되었을 경우 해당 자원이 다른 에이전트에게 할당되는 것을 방지하는 자원 예약 기법을 기반으로 피어들 간의 균형 있는 이주 기법을 제안한다.

Dynamic Load Balancing Scheme Based on Resource Reservation for Migration of Agents in Pure P2P Network Environments

Kyung-In Kim[†] · Young-jin Kim[††] · Young-Ik Eom[†††]

ABSTRACT

Mobile agents are defined as processes which can be autonomously delegated or transferred among the hosts in a network in order to perform some computations on behalf of the user and co-operate with other agents. Currently, mobile agents are used in various fields, such as electronic commerce, mobile communication, parallel processing, search of information, recovery, and so on. In pure P2P network environment, if mobile agents that require computing resources rashly migrate to another peers without consideration on the peer's capacity of resources, the peer may have a problem that the performance of the peer is degraded due to lack of resources. To solve this problem, we propose resource reservation based load balancing scheme of using RMA(Resource Management Agent) that monitors workload information of the peers and that decides migrating agents and destination peers. In mobile agent migrating procedure, if the resource of specific peer is already reserved, our resource reservation scheme prevents other mobile agents from allocating the resource.

키워드 : P2P 네트워크(P2P Network), 이동 에이전트(Mobile Agent), 부하 균형(Load Balancing), 자원 예약(Resource Reservation)

1. 서 론

오늘날 이동 에이전트는 인터넷 정보 홍수 속에서 원하는 정보를 검색해 주고, 필요한 작업을 대신 수행해 주는 등 다양한 분야에서 유용하게 사용되고 있다. 또한 국가 시책으로 추진되고 있는 초고속 정보통신망과 연계하여 정보 검색뿐만 아니라 전자상거래나 이동 컴퓨팅 분야에서도 그 필요성은 대두되고 있다. 이러한 이동 에이전트는 과도한 중앙 서버에 대한 의존과 트래픽 집중 현상에서 벗어나 자율성과 이동성을 가지고 비동기적인 연산을 지원하며, 서비스의 분산 및 병렬 처리, 동적인 서버 인터페이스의 변경 지원, 네트워크 부하 감소 등 여러 장점들을 가지고 있다. 그러나 이동 에이전트들이 각 피어들의 가용 자원을 고려하지 않고 이주(migration)하게 되면 해당 시스템의 효율을 떨어뜨리는 문제점을 초래할 수 있다[1, 2].

본 논문에서는 순수 P2P 환경에서 자원을 필요로 하는 에이전트들이 충분한 자원을 가지고 있는 피어들의 자원을 예약함으로써, 에이전트 실행의 효율성을 높이는 기법을 제안한다. 본 제안 기법을 이용하는 에이전트들은 현재 작업을 수행중인 피어가 부하의 임계치를 넘을 경우 인접 피어들의 가용 자원과 부하 정보를 측정한 후, 수집된 부하 정

※ 이 논문은 2003년도 한국학술진흥재단의 지원에 의하여 연구되었음(KRF-2003-041-D20420).
† 준 회 원 : 성균관대학교 전기전자 및 컴퓨터 공학과
†† 준 회 원 : 삼성전자 연구원
††† 종신회원 : 성균관대학교 정보통신공학부 교수
논문접수 : 2004년 2월 19일, 심사완료 : 2004년 8월 4일

보에 따라 이주를 결정하게 된다. 이 기법은 한 피어로 집중될 수 있었던 이동 에이전트들을 여러 피어로 자원의 가용량에 따라 분산 시킬 수 있기 때문에 특정 피어로 에이전트가 집중되는 현상을 방지할 수 있다. 또한, 각 에이전트 플랫폼에 자원 예약 기법을 적용하여 효율적인 자원 사용 정책을 보장할 수 있다.

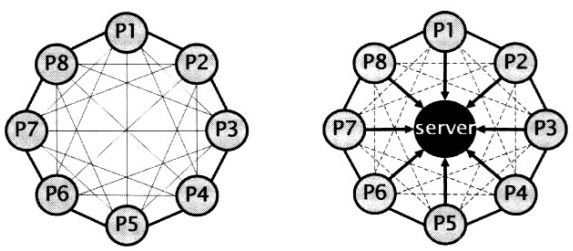
본 논문의 구성은 다음과 같다. 2장에서는 이동 에이전트 및 P2P 네트워크 환경의 개념과 종류 그리고 이동 에이전트를 이용한 부하 균형 기법을 설명한다. 3장에서는 본 논문에서 제안하는 시스템 환경과 메시지 형식, 알고리즘 그리고 동작 시나리오를 보인다. 4장에서는 제안한 시스템의 시뮬레이션 과정을 보이고 마지막으로 5장에서 본 논문의 결론 및 향후 연구 과제에 대해 기술한다.

2. 관련 연구

본 장에서는 P2P 네트워크와 이동 에이전트의 개념을 설명하고, 이동 에이전트를 이용하는 부하 균형 기법에 관해 알아본다.

2.1 P2P 네트워크

일반적으로 P2P 네트워크에서 각 피어는 서버와 클라이언트의 기능을 모두 수행한다. 기존의 중앙 집중식 네트워크 구조의 문제점인 트래픽 집중이라는 한계를 극복하기 위해 각 피어들은 상호 분산, 협력, 자원의 공유 등을 지원함으로써 그 효율성을 극대화하고 있다. 현재 운영되고 있는 P2P 네트워크 방식은 서버 운영의 유무에 따라 순수(pure) P2P 네트워크 방식과 하이브리드(hybrid) P2P 네트워크 방식으로 나뉘며, 이러한 네트워크 구성을 (그림 1)에서 보인다[3].



(a) 순수 P2P 네트워크 (b) 하이브리드 P2P 네트워크

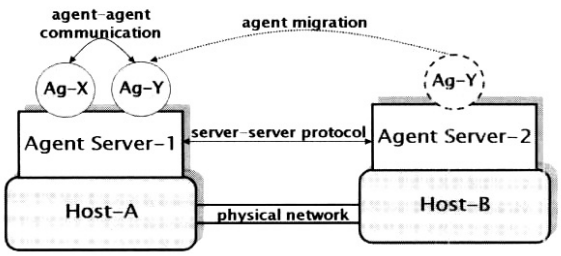
(그림 1) P2P 네트워크 구성

순수 P2P 네트워크 방식은 특정 서버의 운영 없이 피어들 간에 자유로운 연결이 보장된 형태로, 다른 피어들에게 연결하여 자원의 위치를 검색한 후 이를 공유하는 형태를 보인다. 따라서 이러한 방식은 특정 피어가 종료되어도 다른 피어들로부터 서비스를 받을 수 있기 때문에 높은 신뢰성과 확장성을 가진다. 대표적인 순수 P2P 네트워크 시스템으로는 Gnutella가 있다[4]. 하이브리드 P2P 네트워크 방식의 경우에는 특정 서버를 두게 되며 최초 각 피어들이 서로 통신하기 위해서는 일단 이 서버에 접속하게 된다. 이후에는 서버의 도움 없이 피어간 직접 원하는 정보를 전달하게 된다. 대표적인 하이브리드 P2P 네트워크 시스템으로는 Napster가 있다[5].

2.2 이동 에이전트

이동 에이전트는 이기종 분산 환경에서 자율적인 이동이 가능한 개체로 네트워크의 트래픽 감소, 비동기적인 상호작용, 서비스의 분산 및 병렬처리 등을 지원할 수 있다. 현재 여러 에이전트 시스템들이 교육과 상업적인 목적으로 구현되어 있으며 대표적인 에이전트 시스템으로는 Aglet, Aganta, Odyssey, Tacoma, Voyager 등이 있다[6-11].

이동 에이전트는 각각의 독립된 역할뿐만 아니라 정보와 자원을 서로 공유할 수 있기 때문에, 여러 에이전트간의 협업에 의해 한 에이전트가 해결하기 힘든 복잡한 문제들을 해결할 수 있으며, 충분한 자원을 보유한 호스트로 이주하여 계속 작업을 수행할 수도 있다. 이외에도 이동 에이전트는 통신 대역폭의 절감으로 네트워크 부하를 감소시키며, 네트워크에 발생할 수 있는 다양한 오류에 유연하고 능동적으로 대처할 수 있다[11, 12]. (그림 2)에서는 이동 에이전트의 이주와 관련된 일반적인 에이전트 시스템의 동작 과정을 보인다[14].



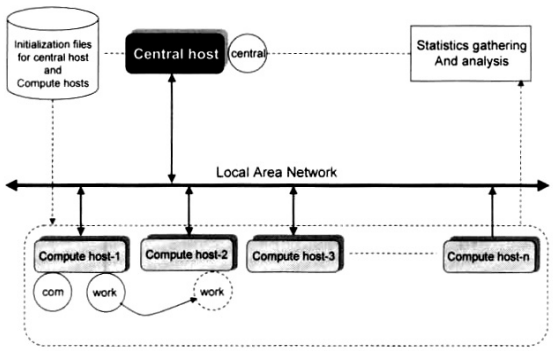
(그림 2) 이동 에이전트 시스템

호스트 A와 호스트 B는 물리적인 네트워크를 통해 연결되어 있으며 각 호스트에는 에이전트 서버(agent server) 프로세스가 실행된다. 에이전트 서버 프로세스는 안전하고 신뢰성 있는 실행 환경을 기반으로 에이전트를 실행하고 호스팅 할 수 있다. (그림 2)에서 호스트 B에 있는 에이전트(Ag-Y)는 호스트 A의 자원이 필요하거나 호스트 A의 임의의 에이전트(Ag-X)와 협업을 목적으로 호스트 A로 이주하여 작업을 계속 수행하는 과정을 보인다.

2.3 이동 에이전트의 부하 균형

에이전트 시스템에서 부하란 시스템에서 원하는 작업을

수행하기 위해 취하는 동작이나 자원을 말한다. 따라서 이동 에이전트의 부하 균형(load balancing)이란 부하가 많이 발생되는 호스트의 작업을 상대적으로 부하가 적은 호스트들에게 이동시킴으로써, 평균 응답 시간을 최소화하고 전체 시스템 처리량을 최대화하는 하나의 태스크 스케줄링 방법을 말한다. 일반적인 부하 균형 기법에는 정적 부하 균형과 동적 부하 균형이 있다[12, 13, 15]. 정적 부하 균형 기법은 작업이 실행되기 이전에 임의의 프로세서에게 미리 작업을 할당하는 방법을 말한다. 즉 시스템의 부하 정보를 고려하여 실행될 호스트를 미리 결정하는 방법을 말하며, 정적 부하 균형 기법을 이용하는 에이전트 플랫폼으로 IBM사의 Aglet이 있다. 동적 부하 균형의 경우 시스템 운영 중에 부하 정보를 고려하여 동적으로 작업을 할당할 수 있기 때문에 변화하는 시스템 상태에 유연하게 대응할 수 있다. (그림 3)은 동적 부하 균형 기법을 이용하는 Comet 이동 에이전트 시스템을 보인다[13].



(그림 3) Comet 시스템 환경

Comet은 동적 부하 균형 기법을 이용하여 이동 에이전트를 이주시키며, 부하 정보를 수집하기 위하여 각 호스트에 상주하는 com 에이전트를 이용한다. (그림 3)에서, Comet은 Central 호스트와 Compute 호스트로 구성 된다. Central 호스트는 전체 시스템을 기동, 정지, 종료하는 역할과 함께 각 호스트의 부하 정보를 수집한다. 부하 정보는 Central 호스트에 상주하는 Central 에이전트와 각 호스트마다 상주하는 com 에이전트 간의 통신에 의해 수집되며, 수집된 부하 정보를 토대로 Compute 호스트의 이주할 프로세스(work 에이전트)를 결정하고(selection policy) 해당 프로세스를 이주할 호스트로 스케줄링(location policy)한다.

그러나 Comet에는 몇 가지 문제점이 있다. 첫째, Comet에서는 전체 시스템의 부하 정보를 수집하기 위한 많은 네트워크 트래픽이 발생하게 된다. 둘째, Central 호스트가 종료될 경우, work 에이전트들은 다른 호스트로 이주할 수 없게 된다. 셋째, Comet은 중앙 집중식 구조이기 때문에 Compute 호스트가 늘어날 경우 Central 호스트가 관리해야 할 호스트가 많아지게 되며, 이로 인해 Central 에이전트가 각 호스트의 부하 정보를 수집하는 시간이 증가하게 된다.

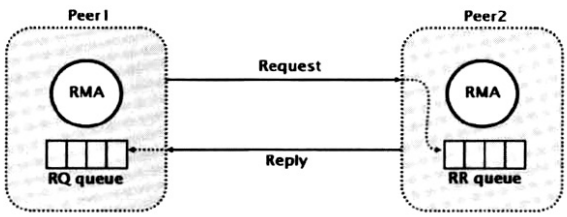
본 논문에서는 순수 P2P 네트워크 환경에서 에이전트로 하여금 동적 부하 균형 기법을 이용하게 함으로써 가용 자원이 있는 피어로 이주하는 기법을 제공한다. 본 기법을 이용하면 가용 자원을 가진 특정 피어가 종료되거나 네트워크 연결이 단절되어도 다른 피어로 이주할 수 있으며 또한 네트워크에 참여하는 피어들의 높은 확장성을 제공한다.

3. 제안 기법

본 장에서는 제안하는 시스템 환경과 피어 간에 송수신하는 메시지들의 형식, 알고리즘, 그리고 동작 시나리오 등을 소개한다.

3.1 시스템 환경

본 논문에서 제안하는 기법은 순수 P2P 네트워크 환경을 기반으로 하며, 각 피어의 자원은 CPU와 가용한 메모리 크기로 한정하였다. (그림 4)에서는 본 제안 기법의 시스템 환경을 보인다.



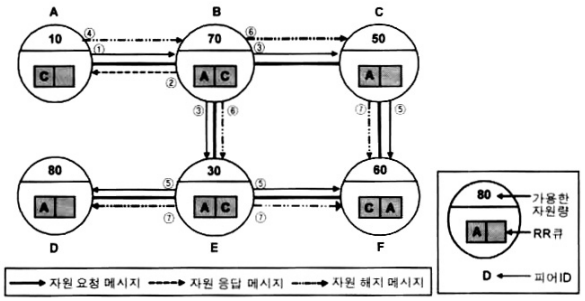
(그림 4) 시스템 환경

본 논문에서 제안하는 시스템에는 피어에 상주하면서 가용 자원을 모니터링하는 RMA(Resource Management Agent)와 자원을 요청하거나 이에 대해 응답하는 메시지들을 저장하는 큐가 있다. RMA는 시스템의 백그라운드에서 데몬으로 동작하면서 피어의 자원을 수집하는 작업과 이주할 이동 에이전트와 이주될 목적지 피어를 결정하는 정책을 수행한다. 또한 RMA는 다른 호스트로부터 자원 요청 메시지들이 수신될 경우, 특정 큐에 이 메시지들을 저장하여 자원을 예약하며, 자원 해지 메시지를 받게 될 경우, 큐에서 예약 메시지를 삭제하는 등 피어의 가용 자원을 바탕으로 필요한 메시지를 송수신하는 역할도 담당한다. 자원을 요청하고 응답하는 메시지들의 저장은 RQ(Resource reQuest) 큐와 RR(Resource Reservation) 큐가 담당한다. RR 큐는 필요한 자원의 종류와 크기로 설정된 메시지들을 저장하여 자원을 예약하는 역할을 담당하며, RQ 큐는 가용 자원을 보유한 피어들이 보내는 응답 메시지들을 저장함으로써 어떤 피어에 자원이 예약되었는지 확인할 수 있게 한다.

3.2 동작 과정

본 논문에서의 제안 기법은 자원 요청 단계, 자원 예약 단계, 자원 해지 단계로 구성되며 (그림 5)와 (그림 6)은 이 제안 기법의 3단계 동작 과정을 단계별로 보인다(단, (그림 5)와 (그림 6)의 동작 과정은 동시에 일어난 상황이며, 각기 상황에 맞는 주체를 중심으로 분류하여 설명한다).

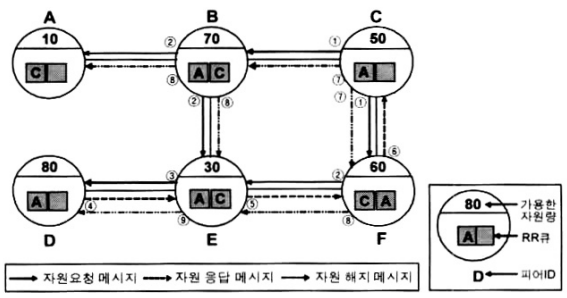
피어 A의 자원 요청 및 응답, 해지 과정을 (그림 5)에서 보인다.



(그림 5) 피어 A의 전체 동작 과정

(그림 5)에서 피어 A의 특정 에이전트가 자원 70을 원한다고 가정할 경우, 피어 A는 인접한 피어 B에게 "자원 요청 메시지"를 전송한다(①). 이 메시지를 받은 피어 B는 자신이 자원 70을 보유하고 있기 때문에 피어 A의 "자원 요청 메시지"를 RR 큐에 저장한 후, 피어 A에게 "자원 응답 메시지"를 회신한다(②). 또한, 피어 B는 피어 A를 제외한 나머지 인접 피어들에게 "자원 요청 메시지"를 전송하게 된다(③). 이와 같은 방법으로 "자원 요청 메시지"를 받은 피어들은 자원 요청에 대한 예약 메시지를 RR 큐에 저장하고 인접 피어들에게 메시지를 전송한다(③, ⑤). 피어 B로부터 "자원 응답 메시지"를 받은 피어 A는 피어 B를 제외한 나머지 피어들에게 예약된 자원을 해지하기 위한 "자원 해지 메시지"를 전송하게 된다(④, ⑥, ⑦). (그림 6)은 피어 C의 동작 과정을 보인다.

피어 C의 자원 요청 및 응답, 해지 과정을 (그림 5)에서 보인다.



(그림 6) 피어 C의 전체 동작 과정

(그림 6)에서 피어 C의 특정 에이전트가 자원 70을 원할 경우, 피어 C는 인접한 피어 B와 피어 F에게 "자원 요청 메시지"를 전송한다(①). 이 메시지를 받은 피어 B와 피어 F는 피어 C의 "자원 요청 메시지"를 RR 큐에 저장한 후, 인접한 피어들에게 "자원 요청 메시지"를 전송하게 된다(②, ③). 결국 "자원 요청 메시지"를 받은 피어 D는 피어 C가 요청하는 충분한 자원을 보유하고 있기 때문에, "자원 응답 메시지"를 피어 C에게 전송한다(④, ⑤, ⑥). 피어 D로부터 "자원 응답 메시지"를 받은 피어 C는 피어 D를 제외한 나머지 피어들에게 예약된 자원들을 해지하라는 "자원 해지 메시지"를 전송하게 된다(⑦, ⑧, ⑨).

3.3 메시지 형식

본 기법에서 사용하는 메시지로는 자원 요청 및 예약(RREQ) 메시지, 예약 응답(RREP) 메시지, 예약 취소(RREV) 메시지 등이 있으며, (그림 7)에서는 이 메시지들의 형태를 보인다.

Type	Seq	S_Peer	R_Type	R_Size	TTL	R_Peer

(a) RREQ(Resource REQuest) message

Type	Seq	S_Peer	R_Peer	R_Type	R_Size

(b) RREP(Resource REPly) message

Type	Seq	S_Peer	E_Peer	TTL

(c) RREV(Resource REVoke) message

(그림 7) 메시지 형식

(그림 7)에서, 메시지들은 Type 필드의 값에 의해 그 형식이 구분되며, Seq 필드는 메시지의 중복 처리를 방지하는데 사용된다. S_Peer 필드는 자원을 요청하는 피어의 ID가 저장되고, R_Type 필드와 R_Size 필드는 요청하는 자원의 종류와 크기를 나타낸다. RREP 메시지의 R_Peer 필드는 가변적인 필드로써, 해당 메시지를 수신한 피어들의 ID가 추가되며, 결과적으로 RREP 메시지의 R_Peer 필드에는 해당 메시지가 이동하는 경로가 저장된다. 또한 RREV 메시지의 E_Peer 필드에는 이동 에이전트가 이주할 피어의 ID가 저장되며, RREV 메시지를 수신한 피어들은 E_Peer 필드가 자신의 피어 ID와 다르면 RR 큐의 엔트리 중 RREV 메시지의 S_Peer, Seq 필드에 해당하는 엔트리를 삭제한다.

3.4 알고리즘

생성된 메시지가 전송되는 최대 홉 수를 MH(Max Hop)라 하고, 한 홉 사이에 메시지의 평균 송수신 시간을 TT(Transfer Time)ms로 정의한다. 에이전트가 RREQ 메시지를 전송한 후 RREP 메시지를 받을 때까지 걸리는 최대 예상 시간은 2×TT×MH(ms)가 된다. 또한 자원이 예약된 피어에 이동에이전트가 이주를 완료 할 때까지 걸리는 예상 시간도 2×TT×MH(ms)

가 된다. 특정 피어에 자원이 부족할 경우 해당 RMA는 연결되어 있는 모든 피어들에게 RREQ 메시지를 전송하게 되며, 이러한 동작 과정은 (알고리즘 1)에서 보인다.

```
input : RT(Resource Type), RS(Resource Size)
output : reserved peer
{
   msg = Type(RREQ)+Seq(Seq)+S_Peer(myPeerID)+R_Type
         (RT)+R_Size(RS)+TTL(MH)+R_Peer(NULL) ;
   send the msg to the connected peers ;
   TimeOut = C1×2×TT×MH+C2 ;
   sleep(TimeOut) ;
   select the most appropriate entry in RQ queue ;
   make RREV message and send the message to the connected
         peers ;
}
```

(알고리즘 1) 자원 할당 요청 알고리즘

가용 자원을 모니터링하는 RMA는 필요한 자원의 종류와 크기를 RREQ 메시지에 설정한 후 인접한 피어들에게 전송한다. RREQ 메시지를 전송한 피어는 일정 시간(C1×2×TT×MH +C2 ms, C1, C2는 상수)동안 RREP 메시지를 기다린 후, 수신된 RREP 메시지들 중 에이전트를 이주시킬 최적의 피어를 결정하게 된다. RREQ 메시지를 수신한 피어에서는 QUEUE_ENTRY를 생성하여 RR 큐에 저장하게 되며, RR 큐에 저장되는 엔트리의 구조는 <표 1>에서 보인다.

<표 1> QUEUE_ENTRY

Field	설 명
S_Peer	자원을 할당 요청한 피어의 ID를 저장하는 플래그
Seq	Sequence Number
RS(Resource Size)	요청한 자원의 크기를 저장하는 플래그
RT(Resource Type)	요청한 자원의 종류를 저장하는 플래그
ReservedTime	예약 요청이 들어온 시간을 저장하는 플래그
ResFlag	예약 유무를(default : 0) 나타내는 플래그

ResFlag 필드는 초기에 0으로 설정되며 RREQ 메시지 수신 시 가용 자원이 있으면 1로 바뀌게 된다. 그리고 ReservedTime 필드에는 RREQ 메시지를 수신한 시간이 저장된다. 자원의 예약은 가용 자원이 있을 경우에만 이루어지며, RREQ 메시지를 수신한 피어의 작업 절차는 (알고리즘 2)에서 보이는 바와 같다.

RREQ 메시지를 수신한 피어는 QUEUE_ENTRY를 생성하여 RR 큐에 저장한다. 만약 RREQ 메시지를 수신한 피어가 가용 자원을 보유한 경우 자원을 예약하여 최초 자원을 요청한 피어에게 RREP 메시지를 송신하게 되지만 그렇지 않은 경우 RR 큐에 저장만 하고 자원이 반납되기를 기다린다. 이후 TTL 값을 확인하여 인접 피어들로의 RREQ 메시지 전송 여부를 결정한다. 또한 RMA는 이후에 수신되는 예약 메시지들에 대해 이전에 예약된 자원을 제외한 나머지 자원들로 예약할 수 있도록 한다. (알고리즘 3)에서는 RREP 메시지를 수신한 피어의 작업 절차를 보인다.

```
input : msg(RREQ message received)
output : none
{
   if (duplicate message)
      return ;
   record the Seq number of msg and S_Peer ;
   make a QUEUE_ENTRY ;
   insert the QUEUE_ENTRY into the RR queue ;
   if (GetResourceSize(msg.R_Type) >= msg.R_Size) {
      Reserve the requested resource ;
      msg.R_Peer += myPeerID ;
      msg2 = Type(RREP)+Seq(msg.Seq)+S_Peer(msg.S_Peer)+
             R_Peer(msg.R_Peer)+R_Type(msg.R_Type)
             +R_Size(msg.R_Size) ;
      send msg2 to the peer that sent msg ;
   }
   msg.TTL-- ;
   if (msg.TTL > 0) {
      msg.R_Peer += myPeerID ;
      send msg to the connected peers except the peer that sent
             msg ;
   }
}
```

(알고리즘 2) 수신된 RREQ 메시지 동작 알고리즘

```
input : msg(RREP message)
output : Reserved peer
{
   if (S_Peer field of msg != myPeerID) {
      send msg to the previous peer within R_Peer field of msg ;
      return ;
   }
   insert msg into RQ queue ;
}
```

(알고리즘 3) 수신된 RREP 메시지의 동작 알고리즘

```
input : msg (RREV message)
output : none
{
   msg.TTL-- ;
   if (msg.TTL > 0)
      send msg to the connected peers except the peer that
            sent msg ;
   if (E_Peer field of msg == myPeerID)
      return ;
   remove the corresponding entry in RR queue ;
}
```

(알고리즘 4) 수신된 RREV 메시지 동작 알고리즘

만약 RREP 메시지를 수신한 피어가 자원을 요청한 피어가 아니라면 R_Peer 필드를 통해 이전 피어로 메시지를 전송하게 되며, 결국 RREP 메시지는 최초 자원을 요청한 피어까지 전송된다. RREP 메시지가 최초 자원을 요청한 피어

에 도착하게 되면, 이는 해당 피어의 RQ 큐에 저장된다. 최초 자원을 요청한 피어는 RQ 큐에서 최적의 피어 하나만 선정하여 이동 에이전트를 이주시키게 되고 나머지 피어들에게는 RREV 메시지를 보내게 된다(알고리즘 1). (알고리즘 4)는 RREV 메시지를 수신한 피어들의 작업 절차를 보인다.

RREV 메시지를 수신한 피어는 메시지의 TTL 값이 0보다 클 경우 메시지를 보낸 피어를 제외한 인접 피어들에게 메시지를 전송하게 된다. 만약 RREV 메시지의 E_Peer 필드와 해당 피어의 ID가 같을 경우 RREV 메시지를 수시한 피어는 RR 큐 내의 해당 예약 메시지를 삭제하지 않으며, 그렇지 않을 경우 RR 큐 내의 엔트리 중 S_Peer와 Seq 필드 값이 일치하는 예약 메시지를 삭제한다. 그리고 자원을 예약한 피어의 종료나 네트워크 연결의 단절로 인하여 RREV 메시지를 수신하지 못한 피어들은 RMA에 의해 일정 시간이 지나게 되면 예약된 자원이 자동으로 해지되며, 그 작업 절차는 (알고리즘 5)에서 보인다.

```
input : none
output : none
Thread {
    int _ms=2×TT×MH ;
    while(1) {
        a = (the number of entries in RR queue) ;
        while(a--) {
            que = get a-th entry in RR queue ;
            b = Current Time - que.ReservedTime ;
            if (b > _ms && que.ResFlag == 1)
                remove a-th entry from RR queue ;
        }
    }
    sleep(during regular period) ;
}
```

(알고리즘 5) RMA의 자원 해지 알고리즘

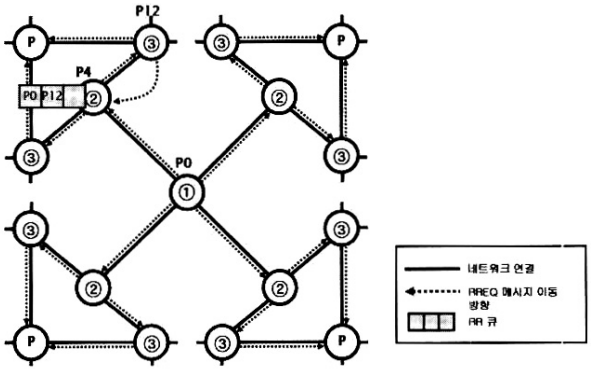
RMA는 일정 시간마다 RR 큐 내에 있는 예약 메시지들을 확인함으로써, RREV 메시지를 수신하지 못하여 RR 큐에서 영구히 존재할 수 있는 예약 메시지들을 삭제하게 된다.

3.5 동작 시나리오

본 절에서는 순수 P2P 네트워크 환경에서 작업을 수행하는 특정 피어에 부하가 집중되는 경우, 각 피어들의 자원에 대한 정보를 조사하여 충분한 자원을 보유한 피어로 에이전트를 이주시키는 시나리오를 보인다. 에이전트의 이주 과정에서 가용 자원이 이미 다른 에이전트에게 예약된 피어로 이주 할 수 없도록 하는 방식으로 동작 시나리오를 보인다.

3.5.1 RREQ 메시지 전송과정

현재 작업 중인 피어의 자원이 부족할 경우, 이동 에이전트는 충분한 자원을 보유한 피어로 이주하여 계속 작업하기를 원한다. 결과적으로 RMA는 자원을 예약하기 위해 RREQ 메시지를 전송하며, 메시지를 수신한 피어는 RR 큐에 RREQ 메시지를 저장한다. (그림 8)에서, 피어 P0은 자원을 예약하기 위해 인접한 피어들에게 RREQ 메시지를 전송하며, 이 메시지는 TTL 값(default : 3홉)에 의해 ③단계에 위치하는 피어들까지 전송된다.

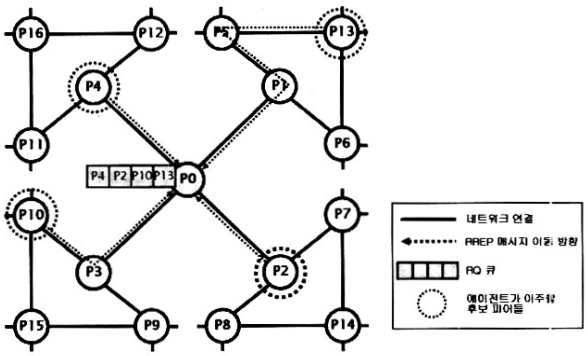


(그림 8) RREQ 메시지의 전송 과정

RREQ 메시지를 수신한 피어는 해당 메시지의 필드(R_Type, R_Size)를 검색해 예약 유무를 결정한다. 만약 예약이 된다면 최초 자원을 요청한 피어에게 RREP 메시지를 전송한다. (그림 8)에서, 피어 P0과 피어 P12는 동일한 크기의 자원을 예약하기 원하며, 피어 P4는 요청하는 자원의 크기만큼을 가지고 있고, 피어 P0은 피어 P12보다 먼저 자원 예약 시도를 하였다고 가정한다. 이 상태에서, 피어 P4의 RR 큐에는 먼저 자원을 예약한 피어 P0의 예약 엔트리와 이후에 예약한 피어 P12의 엔트리가 차례로 저장되게 된다. 이와 같은 경우 예약한 순서대로 자원 예약의 우선권이 주어지므로, 피어 P4는 피어 P0의 자원 예약을 처리한 후 피어 P12의 자원예약 요청을 처리하게 된다.

3.5.2 RREP 메시지 전송과정

자원 예약을 하게 된 피어의 RREP 메시지 전송 과정을 (그림 9)에서 보인다.

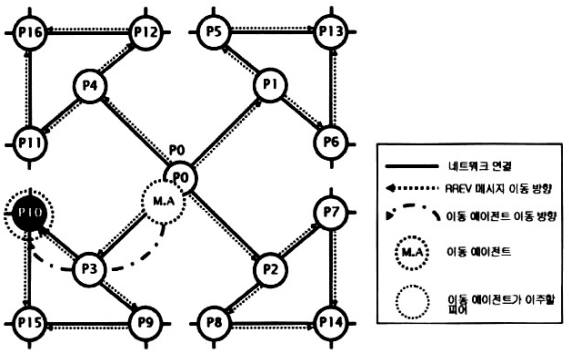


(그림 9) RREP 메시지 전송 과정

(그림 9)에서, 피어 P0이 RREQ 메시지를 전송할 경우 충분한 자원이 있는 피어 P2, P4, P10, P13은 RREP 메시지를 피어 P0에게 전송하게 된다. RREP 메시지를 수신한 피어 P0은 RQ 큐에 RREP 메시지들을 저장하게 되며, 이 중 특정 피어를 선정하여 에이전트를 이주시키게 된다.

3.5.3 RREV 메시지 전송과정

RREV 메시지의 전송 과정을 (그림 10)에서 보인다.



(그림 10) RREV 메시지의 전송 과정

(그림 10)에서, 피어 P0은 RQ 큐에 있는 메시지를 토대로 에이전트가 이주될 피어를 선정한다. 만약 피어 P0이 피어 P10을 이주할 피어로 결정했다면 RREV 메시지의 E_Peer 필드에는 P10의 ID가 저장된다. 이후 RREV 메시지를 수신한 피어들은 피어 P10을 제외하고 모두 예약이 취소되거나 RR 큐에서 해당 엔트리를 삭제한다.

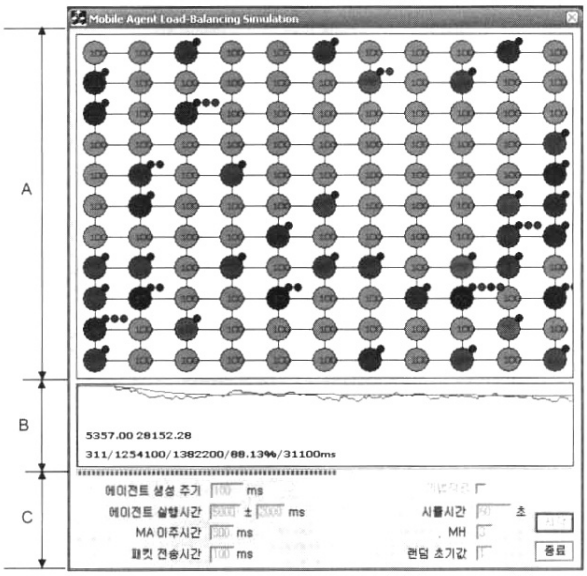
4. 시뮬레이션

본 장에서는 시뮬레이션 모델을 정의하여 앞에서 제시한 알고리즘들의 성능을 측정하고 그 결과를 비교, 분석한다.

4.1 시뮬레이션 개요 및 과정

본 논문에서 제안하는 기법을 테스트하기 위해 Windows 2000 플랫폼을 사용하였으며, 시뮬레이터는 Microsoft사의 Visual C++ 6.0을 사용하여 구현하였다. 본 기법을 이용한 시뮬레이션 프로그램의 화면 구성을 (그림 11)에서 보인다.

(그림 11)에서, 시뮬레이션 화면 구성은 크게 A, B, C 패널로 구성된다. 패널 A는 121개의 피어에서 동작하는 에이전트들의 실행하는 과정을 보인다. 이때 각 피어의 자원은 100으로 설정되어 있고 에이전트가 실행된 피어에 자원이 없을 때 필요한 자원이 확보된 피어로 이주하게 되는 시뮬레이션 동작 과정을 보인다. 패널 B에서는 시뮬레이션 과정에서 보이는 통계치를 나타내며 각각의 숫자가 나타내는 의미는 왼쪽부터 차례대로 생성된 에이전트 수, 전체 에이전트 실행시간, 전체 에이전트 실행시간+전체 에이전트 sleep 시간, 효율성, 시뮬레이션 실행 시간을 나타낸다. 패널 C에서는 여러 가지 상황을 고려해 성능을 평가할 수 있는 제어 패널을 보인다. 여기에서는 에이전트의 생성 주기, 실행 시간, 이주 시간, 메시지 전송 시간, 시뮬레이션 시간, 홉수 등을 제어하여 성능을 비교할 수 있게 한다. 에이전트의 생성주기는 1초당 생성되는 에이전트 수와 반비례하며, 1초당 생성되는 에이전트 수는 "1,000ms/에이전트 생성 주기"가 된다.



(그림 11) 시뮬레이션 화면 구성

본 논문에서 성능평가의 비교 대상이 되는 모델로 에이전트 이주를 하지 않는 기법을 선정하였다. 이 기법에서는, 새로운 에이전트의 실행 시 자원이 부족하면 이를 sleep 상태로 전이시켜 자원을 기다리게 하며, 이후 충분한 자원이 확보되면 작업을 수행할 수 있게 한다. 본 제안 기법은 자원이 부족할 경우 다른 피어의 자원 상태를 고려해 동적으로 이주를 시키게 되므로 에이전트의 sleep 시간을 줄여 시스템의 효율성을 높일 수 있게 된다.

4.2 시뮬레이션 실행 결과

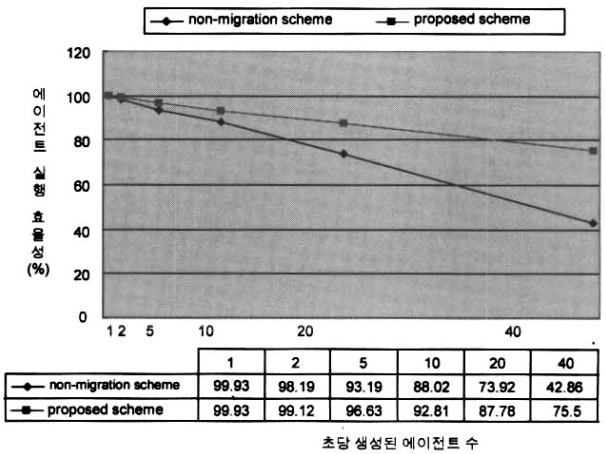
본 논문에서는 패널 C의 다양한 제어 옵션들을 성능 평가의 요소(factor)로 이용하여 시뮬레이션을 수행하였다. 성능 평가의 기준이 되는 에이전트 실행 효율성은 식 (1)에서와 같이 설정하였다.

$$\frac{service\ time}{sleep\ time + service\ time}$$

식 (1) 에이전트 실행 효율성

에이전트들은 실행에 필요한 자원이 부족할 경우 sleep하게 되므로 이 시간을 줄이면 전체 효율성을 높일 수 있게 된다. 위 식에서 service time이 항상 동일하다면 sleep하는 시간을 단축시킬수록 효율성이 높아짐을 알 수 있을 것이다. 시뮬레이션에서 사용되는 기본 설정은 에이전트 생성주기를 100ms, 에이전트 실행 시간은 5,000 ±2,000ms, 에이전트 이주시간을 300ms, 패킷 전송 시간을 100ms, 시뮬레이션을 실행하는 시간은 60초, 에이전트가 이주할 최대 홉 수는 3홉으로 가정하였다.

기본 설정을 기반으로 한 환경에서, 초당 생성된 에이전트 수에 따른 에이전트 실행 효율성을 (그림 12)에서 보인다.



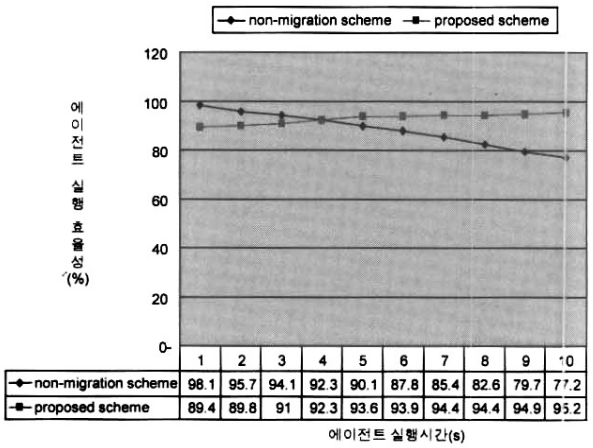
	1	2	5	10	20	40
non-migration scheme	99.93	98.19	93.19	88.02	73.92	42.86
proposed scheme	99.93	99.12	96.63	92.81	87.78	75.5

(그림 12) 에이전트 생성률에 따른 효율성의 변화

(그림 12)에서, non-migration scheme은 각 피어에서 에이전트의 이주 없이 작업을 처리하는 기법을 나타내며, proposed scheme은 제안하는 모델로써 동적인 부하 균형을 적용한 기법을 나타낸다. 초당 생성된 에이전트의 수를 배로 증가함에 따라 두 모델 모두 효율성은 떨어지는 것을 알 수 있다. 그러나 제안된 모델의 경우 부하를 고려하지 않은 모델보다 효율성의 감소 비율이 현저히 낮은 것을 알 수 있다. 결국 제안된 기법을 이용하면 시스템 전체의 작업 효율성을 높일 수 있게 된다.

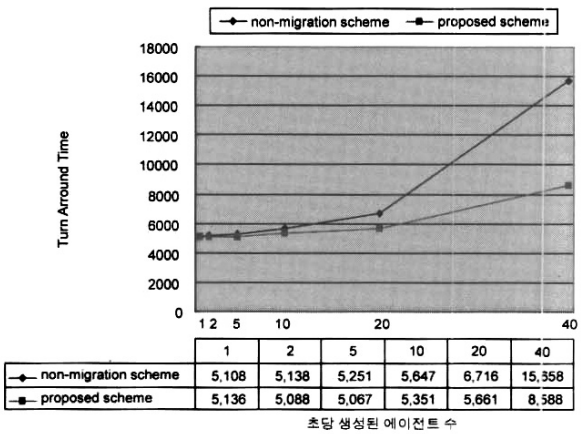
다음으로 1초에 10개의 에이전트가 생성되는 실행 환경에서 에이전트 실행 시간에 따른 에이전트 실행 효율성을 (그림 13)에서 보인다.

(그림 13)에서, 1초에서 3초 정도까지는 non-migration scheme이 proposed scheme보다 효율성이 더 좋음을 볼 수 있다. 이는 non-migration scheme의 에이전트들이 sleep하는 시간이 거의 없이 작업을 처리함으로써 자원을 예약하고 해지하는 proposed scheme 보다 효율성이 좋게 측정됨을 나타낸다. 하지만 그 후부터 non-migration scheme은 각 에이전트의 실행 시간이 늘어나면서 자원을 할당 받기 위해 기다리는 에이전트들이 점점 증가하게 되므로 그 효율성이 계속 감소되는 것을 볼 수 있다. 제안 기법에서는 충분한 자원이 있는 피어로 에이전트들을 이주시키므로 proposed scheme의 효율성이 non-migration scheme의 효율성보다 높아지는 것을 볼 수 있다.



	1	2	3	4	5	6	7	8	9	10
non-migration scheme	98.1	95.7	94.1	92.3	90.1	87.8	85.4	82.6	79.7	77.2
proposed scheme	89.4	89.8	91	92.3	93.6	93.9	94.4	94.4	94.9	95.2

(그림 13) 에이전트 실행시간에 대한 효율성의 변화



	1	2	5	10	20	40
non-migration scheme	5,108	5,138	5,251	5,647	6,716	15,358
proposed scheme	5,136	5,088	5,067	5,351	5,661	8,588

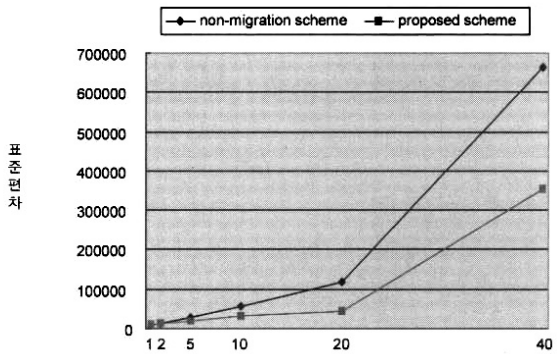
(그림 14) 에이전트 생성률에 따른 에이전트 평균 실행 시간의 변화

기본 설정을 기반으로 한 환경에서, 초당 생성된 에이전트 수와 에이전트들이 실행되는 평균 실행 시간의 관계를 (그림 14)에서 보인다.

(그림 14)에서, 초당 생성된 에이전트 수가 증가하게 되면 에이전트의 평균 실행 시간이 증가함을 알 수 있다. 하지만 proposed scheme은 non-migration scheme보다 증가율이 낮은 것을 알 수 있다. 이는 proposed scheme이 동적으로 작업의 부하를 균등하게 분배하여 에이전트의 sleep 시간을 줄였기 때문이며, 이에 따라 평균 에이전트 실행 시간이 보

다 완만하게 증가된 것을 알 수 있다.

기본 설정을 기반으로 한 환경에서, 초당 생성된 에이전트 수와 에이전트 실행 시간에 관련된 표준 편차를 (그림 15)에서 보인다.



	1	2	5	10	20	40
non-migration scheme	9,351.2	12,923	27,395	54,920	118,407	662,163
proposed scheme	9,209	12,715	19,746	30,194	43,593	353,058

(그림 15) 에이전트 생성률에 따른 에이전트 실행 시간의 표준 편차

(그림 15)에서, 초당 생성된 에이전트의 수가 증가함에 따라 표준 편차도 증가함을 볼 수 있다. 표준 편차는 부하 균형 정도를 나타내는 가장 중요한 척도인데, 표준 편차의 수치가 낮은 proposed scheme은 non-migration scheme에 비해 에이전트 작업의 부하 균형이 잘 되었음을 알 수 있다. 따라서 본 논문에서 제안하는 기법은 초당 생성되는 에이전트 수와 에이전트의 실행 시간이 증가하여도 자원이 충분한 피어로 이주하여 작업을 처리하기 때문에 전체 시스템의 효율성을 높일 수 있음을 알 수 있다.

5. 결론 및 향후 연구계획

현재, 이동 에이전트 및 P2P 환경에 대한 다양한 연구와 개발이 진행되고 있으며, 앞으로 인터넷기반 P2P 기술 및 에이전트 기술 개발이 지속 될 것으로 예측된다. 본 논문에서는 기존의 중앙 집중식 네트워크 구조의 문제점인 트래픽 집중이라는 한계를 극복하기 위해 상호 분산, 협력, 자원의 공유 등을 지원하는 P2P 네트워크 환경을 기반으로 하는 에이전트 이주 기법을 제안하였다. 작업을 수행하는 피어의 가용 자원이 임계치에 미달 할 경우, 각 피어들의 자원에 대한 정보를 참조하여 충분한 자원을 보유한 피어로 에이전트를 이주시킨다. 자원의 수집은 각 피어들에 상주하는 에이전트가 담당하고, 이동 에이전트가 이주할 때 신뢰성과 실효성을 높여주기 위해서 피어의 자원을 예약하면 다른 피어가 예약한 피어로 먼저 이주할 수 없도록 한다. 따라서 이 기법은 더욱 안전한 에이전트의 이주와 각 피어의 부하에 대한 실시간 모니터링기반의 에이전트 이주 정책을 수행하기 때문에 작업의 효율성을 높일 수 있게 된다.

향후 변화하는 네트워크 환경과 좀 더 다양한 자원의 종류를 고려하는 에이전트 이주를 설계가 필요할 것이다. 더불어, 에이전트의 이동성은 보안 측면에서의 취약점을 드러내고 악의 있는 호스트(malicious host)나 악성 이동 에이전트(malicious mobile agent)로부터의 공격과 같은 새로운 문제를 발생시키므로 에이전트 시스템에 내·외부의 공격을 견딜 수 있는 보안성을 부여하기 위한 보안 모델에 대한 고려도 필요할 것이다.

참 고 문 헌

[1] C. Harrison, D. Chess and A. Kershenbaum, "Mobile Agents : Are They a Good Idea?," Research Report 1987, IBM Research Division, 1994.

[2] S. Berkovits, J. Guttman and V. Swarup, "Authentication for Mobile Agents," Lecture Notes in Computer Science #1419 : Mobile Agents and Security, Springer-Verlag, 1998.

[3] D. Barkai, "An Introduction to Peer-to-Peer Computing," Developer Update Magazine, Intel Corporation, Feb., 2000.

[4] CLIP2, "The Gnutella Protocol Specification v0.4," Technique Report, http://www.clip2.com.

[5] A. Oram, Peer-To-Peer, O'Reilly, Mar., 2001.

[6] J. Baumann, et. al., "Communication Concepts for Mobile Agent Systems," Lecture Notes in Computer Science, Vol. 1219, Springer-Verlag, 1997.

[7] A. Gopalan, S. Saleem and D. Andresen, "Bablets : Adding Hierarchical Scheduling to Aglets," The 8th IEEE International Symposium on High Performance Distributed Computing, Redondo Beach, California, Aug., 1999.

[8] N. M. Karnik and A. R. Tripathi, "Security in the Ajanta mobile agent system," Technical Report, University of Minnesota, Minneapolis, MN 55455, U.S.A, May, 1999.

[9] Glass G., "Voyager Core Package Technical Overview," White Paper, ObjectSpace, 1999.

[10] D. Johansen, R. van Renesse and F. B. Schneider, "An Introduction to the TACOMA Distributed System," Technical Report, Department of Computer Science University of Tromso, Jun., 1995.

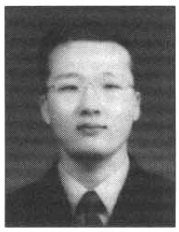
[11] N. Karnik and A. Tripathi, "Agent Server Architecture for the Ajanta Mobile Agent System," Proc. International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA'98), Jul., 1998.

[12] J. Gomoluch and M. Schroeder, "Information Agents on the Move : A Survey on Load-Balancing with Mobile Agents," Software Focus, Vol.2, No.2, Wiley, 2001.

[13] K. P. Chow and Y. K. Kwok, "On Load Balancing for Distributed Multiagent Computing," IEEE Transactions on Parallel and Distributed Systems, Vol.13, No.8, Aug., 2002.

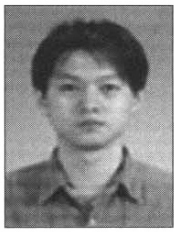
[14] N. Karnik, "Security in Mobile Agent Systems," Ph.D. Dissertation, University of Minnesota, 1998.

[15] D. Gupta and P. Bepari, "Load Sharing in Distributed Systems," Proc. National Workshop on Distributed Computing, Jadavpur University, Calcutta, Jan., 1999.



김 경 인

e-mail : kimki95@ece.skku.ac.kr
2002년 인제대학교 컴퓨터공학과(학사)
2002년~현재 성균관대학교 대학원 전기전자 및 컴퓨터공학과 석사과정
관심분야 : P2P, 이동 에이전트, 유비쿼터스 컴퓨팅 등



김 영 진

e-mail : yj21c.kim@samsung.com
2002년 대전대학교 컴퓨터공학과(학사)
2004년 성균관대학교 대학원 전기전자 및 컴퓨터 공학과 졸업(석사)
2004년~현재 삼성전자 연구원
관심분야 : P2P, 이동 에이전트, 분산 컴퓨팅 등

엄 영 익

e-mail : yieom@ece.skku.ac.kr
1983년 서울대학교 계산통계학과(학사)
1985년 서울대학교 대학원 전산과학과(석사)
1991년 서울대학교 대학원 전산과학과(박사)
2000년~2001년 Dept. of Info. and Comp. Science at UCI 방문교수
현재 성균관대학교 정보통신공학부 교수
관심분야 : 분산 컴퓨팅, 이동 컴퓨팅, 이동 에이전트, 시스템 소프트웨어, 시스템 보안 등