최단 수행 순서 제공을 위한 에이전트 이주 정보 시스템 설계

박 홍 진†

요 약

이동 에이전트가 연속적으로 다른 노드로 이주될 때 총 작업수행 시간은 방문해야할 순서에 따라 변경될 수 있다. 효율적인 이주 알고리즘은 초기 상태에서 목적 상태로 수행되기 위해 최소한 수행 시간을 제공하는 수행 순서를 의미하며, 이동 에이전트의 효율적인 이주 알고리즘을 통해 개발자가 원하는 작업의 총 수행 시간을 최소화시킬 수 있다. 기존의 대표적인 이동 에이전트인 Aglets, Voyager, Odyssey 등은 이주할 때 네트워크 상태와 이주할 노드의 상태를 고려하지 못함으로써 효율적인 이주를 보장할 수 없다. 본 논문은 이동 에이전트에서 효율적인 이주를 위해 AMIS를 제안한다. AMIS는 이동 에이전트의 총 이주 시간을 최소화하며, 안전하고 견고한 이주를 위해 최단 수행 순서를 제공한다.

Design of the Agent Migration Information System for Shortest Migration Order

Hong Jin Park†

ABSTRACT

The total processing time may vary according to the order of visit when a mobile agent is sequentially migrated to another node. An effective migration algorithm is one in which the processing time is kept to its minimum from the initial state to the destination state by ordering the process. The total time spend for the process can be minimized by adopting an effective migration algorithm. Existing mobile agents such as Aglets, Voyager, and Odyssey do not guarantee the effectiveness by not taking the status of the network and the node to be moved into upon the migration. This paper proposes AMIS as the method used for the migration of the mobile agent. AMIS minimizes the total migration time of the mobile agent, and provides a firm and safe order for the migration of the mobile agent.

키워드 : 분산 시스템(Distributed System), 이동 에이전트(Mobile Agent), 이주 알고리즘(Migration Algorithm)

1. 서 론

이동 에이전트(mobile agent)는 네트워크를 통해 이주하면서 자신의 제어하에 유용한 수행을 할 수 있는 프로그램이다. 기존 클라이언트/서버 파라다임에서 네트워크를 통해 원하는 데이터를 전송하는 것과 다르게 이동 에이전트는 데이터가 있는 노드로 에이전트가 이주하여 작업을 수행하며 결과를 가지고 원래의 노드에 돌아온다[1, 2]. 따라서 네트워크 상에서 많은 데이터 전송이 필요할 경우 원격지에 있는 데이터를 직접적으로 연결하여 데이터를 접근하는 기존 클라이언트/서버 파라다임보다 이동 에이전트 파라다임은 네트워크의 낮은 대역폭으로 분산된 자원을 효율적으로 접근할 수 있는 장점이 있다. 이동 에이전트의 또 하나의 장점은 클라이언트 서버 사이에 지속적인 연결이 필요 없기 때문에 연결이 끊어지더라도 견고하게 작업을 수행 할 수 있다[3]. Code-on-Demand 파라다임은 클라이언트 측으로 서버의 프로그램이 이동 에이전트처럼 코드가 이주하나, 여러 노드로 계속적인 이주는 지원할 수 없다. 따라서 분산 컴퓨팅 파라다임에서 이동 에이전트 기술은 많은 노드로부터 정보를 수집하는 응용 프로그램에서는 다른 분산 파라다임에 비해 낮은 대역폭을 이용하면서 견고한 수행을 할 수 있다[4, 5].

이동 에이전트가 연속적으로 다른 노드로 이주 될 때 총 작업 수행 시간은 방문해야 할 노드의 순서에 따라 변할 수 있다[6, 15]. 효율적인 이주 알고리즘(migration algorithm)은 초기 상태에서 목적 상태로 수행되기 위해 최소한 수행 시간을 제공하는 수행 순서를 의미하며[7], 이동 에이전트의 효율적인 이주 알고리즘을 통해 개발자가 원하는 작업의 총 수행 시간을 최소화시킬 수 있다. 효율적인 이주 알고리

※ 이 논문은 2002년도 상지대학교 교내 연구비 지원에 의한 것임.
† 정 회 원 : 상지대학교 컴퓨터정보공학부 교수
논문접수 : 2002년 4월 23일, 심사완료 : 2002년 10월 7일

즘은 노드의 단절, 노드의 다운, 네트워크 구성 변경 등 네트워크 환경이 동적으로 변경되었을 경우는 더욱 중요하다.

현재 개발된 대표적인 이동 에이전트 시스템인 Aglets[8], Voyager[9], Odyssey[10] 등에서 에이전트 이주는 명시적으로 프로그램을 개발하는 개발자가 순서를 정하여 수행되고 있다. 이러한 이동 에이전트의 이주 순서는 최적이 아님은 물론 이주할 노드의 시스템 상태를 고려하지 못하고 있다.

본 논문은 이동 에이전트가 연속적으로 이주할 때 최소 이주 시간을 제공하기 위해 에이전트 이주 정보 시스템(AMIS : Agent Migration Information System)을 제안한다. 본 논문에서 제안하고 있는 이주 정보 시스템의 목적은 사용자에게 서비스의 위치 투명성을 제공함과 동시에 에이전트의 총 수행시간을 최소화하고 안전하고 견고한 이주 수행 순서의 제공에 있다.

2. 기존 이동 에이전트 시스템의 이주

초창기의 이동 에이전트 시스템 개발은 Tcl, Scheme, Oblique, Rosette 같은 프로그래밍 언어를 이용하였으나 현재는 자바 가상 머신(JVM : Java Virtual Machine)으로 인해 하드웨어 독립성을 지원하는 자바 언어로 구현되고 있다. 자바의 직렬화(serialization), 원격 메소드 호출(RMI : Remote Method Invocation), 멀티 쓰레딩(multi-threading), 리플렉션(reflection) 기능은 이동 에이전트의 특성을 지원할 수 있는 자바 기능이다. 현재까지 개발된 대표적인 이동 에이전트 시스템은 제너럴 매직사에서 개발한 Odyssey와 IBM의 Aglet, 오브젝트스페이스의 Voyager이다[11].

2.1 Aglets

일본 IBM에서 개발한 Aglets는 에이전트(agent)와 애플릿(applet)을 합쳐서 생성된 합성어이다. Aglets는 다른 이동 에이전트 시스템과 다르게 자체적으로 개발된 ATP(Agent Transfer Protocol)을 사용한다[8]. ATP는 HTTP의 상위 프로토콜로 에이전트를 다른 노드로 이주할 때 사용되는 응용 계층의 프로토콜이다. Aglets는 자바로 구현되었으며, 에이전트 이주는 자바에서 제공하는 소켓(Socket)을 기반으로하고 있다. 에이전트 이주와 관련된 명령어는 두 가지를 제공하며 다음과 같다. "dispatch"는 에이전트를 전송하는 것이며, "retract"는 에이전트를 가져오는 것(patching)이다(그림 1).

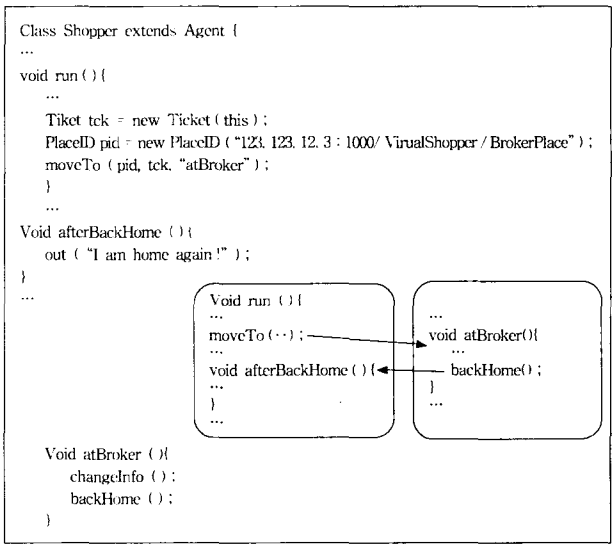
```
예) 단일 이주일 경우
dispatch URL ( "apt://some.host.com" ) ;
dispatch URL ( "apt://some.host.com" ) ;

예) 연속 이주일 경우
Vector itinerary = new Vector ( ) ;
try {
        itinerary.addElement ( new URL ( destination_1 )) ;
        itinerary.addElement ( new URL ( destination_2 )) ;
                .
                .
        itinerary.addElement ( new URL ( destination_n )) ;
} catch ( Exception e ) { Failed to creat itinerary }
..
dispatch ( itinerary ) ;
```

(그림 1) Aglets에서 에이전트 이주

2.2 Voyager

오브젝트스페이스(ObjectSpace) 사에서 만든 Voyager는 순수 자바와 100%로 호환되는 자바로 구현된 이동 에이전트 시스템이다[9]. Voyager는 원격 객체의 활성화 기능, CORBA에서 사용할 수 있는 IDL, IIOP등의 기능이 모두 지원된다. Voyager는 자바 메시지를 에이전트에게 전송할 수 있는 기능이 있다. Voyager 에이전트 이주는 다음과 같다. 첫째, 이동할 객체를 생성시킨다. 둘째는 생성된 객체를 얻는다. 이때 사용되는 메소드는 moveTo()이다. 셋째 지정된 노드로 프로그램을 이주시키며 사용되는 메소드는 moveTo(String URL)이다(그림 2).



(그림 2) Voyager에서 에이전트 이주

2.3 Odyssey

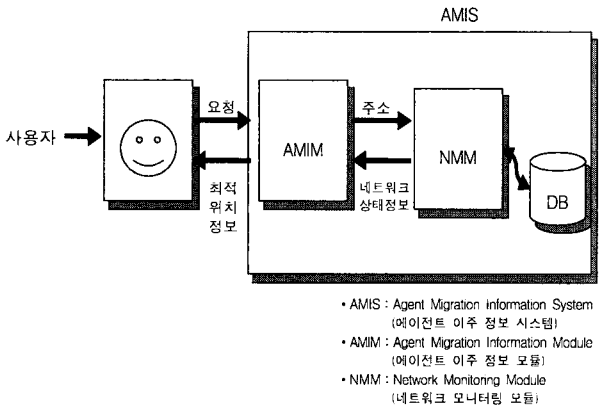
이동 에이전트를 처음으로 사용한 제너럴 매직(General Magic) 사에서 만든 Odyssey는 초창기 C++를 기반으로 한 이동 에이전트 개발 환경인 텔레스크립트(Telescript)의 영향을 받았으며, 현재는 자바 언어로 구현된 이동 에이전트 시스템이다[10]. 현재는 CORBA 객체를 원격지에서 접근과 JDBC를 통해 관계형 데이터 베이스를 지원하고 있다. 이동 에이전트 이주는 자바에서 제공하고 있는 RMI를 기반으로 하고 있다. 에이전트는 "go" 명령어를 사용하여 전향적인(proactive) 이주를 수행할 수 있다. 또한, "send" 명령어는

원격 전향적인 복사를 수행할 수 있다.

그러나 기존 이동 에이전트의 이주는 명시적으로 프로그램을 개발하는 개발자가 순서를 정하여 수행되고 있다. 즉, 이동 에이전트를 이용하여 응용 프로그램 개발자는 에이전트의 계속적인 이주를 위해 자바에서 제공하고 있는 벡터(vector) 클래스를 이용하여 에이전트 이주의 순서를 정의하고 있다. 이러한 이주는 다음과 같은 문제점이 있다. 첫째 이동 에이전트가 이주할 때 네트워크 상태가 전혀 고려되지 않았다. 이는 이주 수행 시간이 최적이 아님은 물론 최악의 경우 최대 이주 수행 시간이 걸려서 수행될 수 있음을 의미한다. 둘째 이주할 노드의 상태를 고려하지 않았다. 만약 이동 에이전트가 연속적인 이주할 때 이주해야 할 노드의 시스템이 다운(down) 되었을 경우 이동 에이전트는 해당 노드에서 멈출 것이며, 이러한 상황에서 이동 에이전트는 계속적인 수행을 할 수 없다.

3. 에이전트 이주 정보 시스템

본 논문에서 제시하고 있는 효율적인 이주 방법은 위치 투명성을 제공함과 동시에 에이전트의 총 수행시간을 최소화하고 안전하고 견고한 이주를 위해 최단 수행 순서의 제공이 목적이다. 에이전트 이주 정보 시스템은 에이전트 이주 정보 모듈(AMIM : Agent Migration Information Module)과 네트워크 모니터링 모듈(NMM : Network Monitoring Module)로 구성되어 있다(그림 3).



(그림 3) 에이전트 이주 정보 시스템의 구조

사용자가 원하는 서비스가 로컬이 아닌 원격지에 있을 경우 요청 에이전트는 원격지의 최적 경로를 위해 AMIS에게 요청한다. AMIS는 네트워크 상황과 시스템의 부하 정보를 고려하여 사용자 에이전트가 요청한 서비스를 수행할 수 있는 서비스 에이전트가 존재하는 노드에 대한 최적의 목적지 경로를 제공하는 것을 목적으로 한다. 에이전트 이주 정보 모듈은 각 노가의 제공하는 서비스에 대한 정보를 가지고 있다. 따라서 사용자에게 요청된 서비스와 네트워크 실제 주소 사이를 매핑시켜야 한다. 네트워크 모니터링 모듈은 최단 경로 설정을 위한 전체 네트워크의 정보를 수집하며, 기존의 최단 경로 설정 알고리즘과 외판원 문제를 해결하는 알고리즘이 있다.

3.1 에이전트 이주 정보 모듈

AMIS의 구성 요소인 에이전트 이주 정보 모듈(AMIM)은 이주 알고리즘과 서비스 위치 정보 테이블을 유지하고 있다. 서비스 위치 정보 테이블은 사용자가 요청하는 서비스 이름과 해당 서비스를 수행할 수 있는 에이전트 이름, 네트워크에서 해당 에이전트가 존재하는 위치 정보, 결함 여부를 나타내는 가용성과 해당 노드의 부하 정보로 구성되어 있다<표 1>. 하나의 노드에는 여러 개의 에이전트가 존재할 수 있으며, 하나의 에이전트는 여러 개의 서비스를 제공할 수 있다. 가용성은 0혹은 1값을 지니고 있는데 1일 경우 해당 노드는 살아 있음을 나타내고, 0일 경우 해당 노드는 결함이 발생되었음을 나타낸다. 가용성과 시스템 처리율에 대한 정보는 네트워크 모니터링 모듈에서 주기적으로 정보를 얻는다.

〈표 1〉 서비스 위치 정보 테이블

Service Name	Agent Name	Location Information	Availability	Load Information
Service_A	AA	111.111.12.xxx	1	60%
Service_A1	AA	111.111.12.xxx	1	60%
Service_C	BB	111.111.12.xxx	1	60%
Service_A	CC	222.222.17.xyz	1	20%
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

단일 이주는 수신측이 하나로써 이동 에이전트는 수신측으로 이동한 후에 작업을 수행하며 수행이 끝나면 원래 송신 측으로 되돌아오는 경우이다. 연속 이주는 이주할 노드의 개수가 여러 개인 경우를 의미하며, 선택된 노드의 개수는 다음과 같이 수식으로 표현된다.

$$2 \le M \le N$$

여기서 M은 선택된 노드의 개수이며, N은 전체 시스템의 노드의 개수이다.

〈표 2〉 연속 이주 알고리즘

```
RSN = Request Service Names
RN = Request Node
SN = Service Node
ASN = Array Service Node
SLIT = Service Location Information Table
CB = Check_bit
(if CB == 1 then fixed migration
        else random migration )
```

```
NMM = Network Monitoring Module
ASP = Array Shortest Path
SLI = System Load Information


wait RSN ;
get RN ;

find SN matched RSN in SLIT ;

if ( SN is empty ) {
        // can't find location
        reply error message ;
}

if ( SN is not empty and SN > 1) {

        // fault check
        check availability ;

        // fixed migration
        if ( CB == 1 ) {
                pass ASN & RN to NMM ;
                get ASP ;
                return ASP ;

        } else if ( CB == 0 ){ //random migration
                pass ASN & RN to NMM ;
                get ASP ;
                checking ASP & SLI ;
                return optimal ASP ;
                }
        }
}
```

연속 이주는 이동 에이전트 이주할 때 이주할 순서가 정해진 경우와 순서가 정해지지 않는 경우로 구분될 수 있다. 이동 에이전트가 연속적으로 이주할 때 알고리즘은 <표 2>와 같다. <표 2>는 이동 에이전트의 이주순서가 고정된 경우와 고정되지 않은 경우를 각각 고려하였으며, 이에 대한 설명은 아래와 같다.

① 이주할 순서가 정해지면서 이주 노드가 여러 개인 경우

하나의 요청 에이전트가 이주하는 노드가 하나이상인 경우이면서 이주할 순서는 미리 정의된 경우이다. 예를 들어 사용자가 요청한 노드의 순서가 A, B, C라면 이를 수행하는 에이전트는 A, B, C 순서로 수행해야 한다. 이 경우에는 단일 이주와 비슷하며, 서비스를 수행하는 각 노드의 최단 경로를 고려하여 최적의 경로를 서비스 요청 에이전트에게 전송시킨다. 이주할 순서가 정해진 경우의 예로 작업을 병렬적으로 처리를 하는 경우이다.

② 이주할 순서가 정해지지 않으면서 이주 노드가 여러 개인 경우

이 경우는 ①번 경우와 비교하여 연속 이주이면서 이주할 순서가 정해지지 않은 경우이다. 이주할 순서가 정해지지 않은 경우의 예는 정보 수집이나 검색 및 필터링할 경우 특정 사이트에 대한 순서가 정해지지 않은 경우이다.

3.1.1 이주 수행 시간

3.1.1.1 단일 이주 수행 시간

이동 에이전트에서 이주되는 정보는 에이전트 코드와 상태 정보이며 전송되기 전에 마샬링을 해야한다[5]. 마샬링(marshalling)은 상태 정보와 데이터를 상대편으로 전송될 수 있는 메시지로 패킹시키는 것을 의미한다. 마샬링된 형태로 전송된 이동 에이전트는 수신 측에서 서비스를 수행하기전에 마샬링된 정보를 원래 상태로 복원하는 언마샬링(unmarshalling)이 필요하다. 즉, 수신측에서 언마샬링된 후 서비스 수행 작업을 수행한다. 수행된 결과는 송신측에서 전송된 이동 에이전트에 추가하여 송신측으로 되돌려 진다. 이때 결과도 역시 마샬링이 필요하다. 에이전트가 단일 노드에 이주하여 수행하는데 걸리는 총 수행 시간을 표현하기 위해 사용된 기호는 <표 3>와 같다.

〈표 3〉 이주에서 사용된 기호

의 미	기 호
에이전트 코드	A_{code}
상태 정보	A_{state}
네트워크 처리율	τ
마샬링/언마샬링 시간	μ
수행 시간	ω

이동 에이전트는 다음과 같이 표현될 수 있다.

$$A = (A_{code}, A_{state})$$

위치 L1에서 L2로 에이전트 이주에 대한 네트워크 부하는 다음과 같이 계산되어 진다.

$$A_{Mig}(L1, L2, A) = A_{code} + A_{state}$$

네트워크 처리율 τ, 마샬링의 오버헤드 μ을 포함하는 L1에서 L2로 에이전트 이주에 대한 총 실행 시간은 다음 식 (1)과 같이 나타낼 수 있다.

$$T_{Mig}(L1, L2, A) = \frac{A_{Mig}(L1, L2, A)}{\tau(L1, L2)} + 2\mu(A_{code} + A_{state}) \quad (1)$$

이동 에어전트가 L1에서 L2로 이주한 후에 언마샬링을 한 다음, 목적지 노드에서 작업을 수행한 후 다시 L1으로 전송되는 총 수행 시간은 다음 식 (2)와 같이 나타낼 수 있다.

$$T(L1, L2, A, \tau, A_{result}) = T_{Mig}(L1, L2, A) + \omega + 2\mu A_{result} + \frac{A_{result}}{\tau(L1, L2)} \quad (2)$$

식 (2)를 풀면 다음과 같다.

$$\frac{(A_{code}+A_{state})+A_{result}}{\tau(L1,L2)}+\omega + 2\mu(A_{code}+A_{state})+2\mu A_{result}$$

3.1.1.2 연속 이주 수행 시간

노드의 개수가 3개인 경우에는 이주할 수 있는 경우의 수는 2가지로 단순하다. 그러나, 이주 노드가 증가하면 할수록 이주할 수 있는 경우의 수는 지수적으로 증가하게 된다. 이주할 순서가 정해지지 않은 경우 이주할 수 있는 경우의 수는 원래 노드를 제외하고 목적지 노드의 개수를 놓아놓은 경우로써 서로 다른 n개중에서 n-1개를 택하여 일렬로 나열하는 순열(permutation)이다.

노드의 집합 S = {1, 2, 3, ···, n}일 때 이주할 수 있는 경우의 수는 다음 식 (3)과 같이 나타낼 수 있다.

$$_nP_{(n-1)} = n(n-1)(n-2)(n-3)\times\cdots\times 2 \quad (3)$$

따라서, 연속 이주에서 목적지가 없을 경우에 이주할 목적지 노드를 최소 경비로 이주하는 문제로 집합 S의 최소 경비로 이주에 대한 문제이며 다음과 같은 식 (4)로 나타낼 수 있다.

$$C(\pi)=\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)}+d_{\pi(n),\pi(1)} \quad (4)$$

식 (4)에서 $d_{\pi(i),\pi(i+1)}$는 노드 i에서 i+1까지의 네트워크 거리를 나타내며, 본 논문에서는 노드 i에서 i+1까지의 네트워크 처리율을 의미하기도 한다. 최소 경비로 이주할 수 있는 $d_{\pi(i),\pi(i+1)}$와 $d_{\pi(n),\pi(1)}$의 최소값을 구하는 것이다. 즉, 최소 작업 수행 시간을 구하는 것은 $C(\pi)$의 최소 값과 시스템의 부하 정보를 고려하여 최적의 경로를 찾는 문제이다. 식 (4) 최적의 이주 경로를 설정하는 것은 외판원 문제(Travelling Salesman Problem)을 해결하는 문제와 비슷하다. 외판원 문제는 NP-complete 문제로 알려져 있다.

단일 이주는 하나의 수신자만 있는 경우지만 연속 이주는 이동 에이전트가 하나 이상의 사이트로 이주하는 것을 의미한다. n개의 노드를 이주하는데 걸리는 시간을 계산하기 위해 n개의 노드에서 임의 노드를 Si라고 하자 ($1 \le i \le n$). n개의 노드에서 효율적인 이주 방법은 이주 시간을 최소화함으로써 총 수행 시간을 최소화시키는데 목적이 있다.

Si번째의 수행되는 시간은 다음과 같다.

$$S_i = S_{i-1}+\frac{A_{Mig}(L_{i-1},L_i,A)}{\tau(L_{i-1}+L_i)}+2\mu(A_{code}+A_{state}) + 2\mu A_{result(i-1)}+\frac{A_{result(i-1)}}{\tau(L_{i-1}+L_i)}$$

Si번째에서 수행한 후 다시 원래의 시스템으로 돌아와서 처리하는 시간은 다음과 같다.

$$T = S_i+\omega_i+2\mu(A_{result,}+A_{code}+A_{state}) + \frac{A_{result(i)}+A_{Mig}(L_i,L_0,A)}{\tau(L_i,L_0)}$$

3.2 네트워크 모니터링 모듈

네트워크 모니터링은 <표 4>와 같이 다음 3가지로 분류된다[11].

<표 4> 네트워크 모니터링 모듈 분류

분류 기준	모니터링 기법	설 명
모니터링 작업 동안에 생성되어지는 트래픽의 양에 따라	수동적인 모니터링	존재하는 메시지에게 피기백(piggyback) 상태 정보를 추가하여 모니터링한다.
	능동적인 모니터링	추가적인 상태 제어 정보를 전송시킴으로써 모니터링한다.
주기적이거나 요구 메시지에 따라	주기적인 모니터링	주기적으로 네트워크를 모니터링하여 어플리케이션이 상태 정보를 요구하였을 때 미리 처리된 정보를 제공한다.
	요구 메시지를 통한 모니터링	어플리케이션이 특정 노드에 대한 네트워크 모니터링 요청이 있을 때 해당 노드에 모니터링 메시지를 전송시켜 특정 자원에 대한 상태 정보를 제공한다.
중앙 집중형과 분산형	중앙 집중형	전체 네트워크의 정보를 중앙 시스템에서 모으고 관리한다.
	분산형	각 노드에서 네트워크 상태정보를 모니터링한다. 다른 네트워크 상태 정보를 알기 위해서는 다른 노드에게 요청하여 정보를 얻는다.

수동적인 모니터링은 네트워크 트래픽을 적게 차지 하지만 두 노드간에 전송되는 메시지 횟수가 적으면, 충분한 네트워크 정보를 획득하지 못한다. 또한, 노드간 전송되는 메시지가 많을 경우 불필요한 메시지의 추가 정보가 많이 발생된다. 사용자가 노드에 대한 모니터링 요청을 했을 경우 요구 메시지를 통해 모니터링하는 방법은 주기적인 모니터링 방법보다 많은 시간이 소요될 수 있다. 그러나 필요할 때마다 네트워크 모니터링 정보를 획득함으로 효율적일 수 있다. 중앙 집중형 모니터링 방법은 확장성과 견고함에 문제가 발생될 수 있다. 만약 정보를 수집하는 중앙 시스템이 고장이 발생되면 전체 네트워크 모니터링 정보는 활용될 수 없다. 따라서 일반적으로 중앙 집중형 모니터링 방법에서 수집된 네트워크 모니터링 정보는 미러링된다. 분산형은 중앙 집중형보다 견고하나 복잡한 제어가 필요하다.

이동 에이전트의 효율적인 이주 방법을 고려할 때 본 논문에서 사용되는 네트워크 모니터링 모듈은 위에서 기술한 모니터링 방법중 중앙 집중형 특징을 가지면서 추가적으로 모니터링 상태 정보를 전송시키는 능동적인 모니터링 기술이 필요하다.

〈표 5〉 네트워크 모니터링 알고리즘

```
while ( true ) {
        // wait until the next collecting time
        sleep ( Freq ) ;
        // send msg to all node
        sendto ( all node ) ;
        // waite to get reply msg from all slave machine
        wait ( ) ;
        check failure ;
        get reply msg ;
        store ( adjacent_latency ) ;
        store ( system_load ) ;
        store ( free_memory_size ) ;
                .
                .
}
```

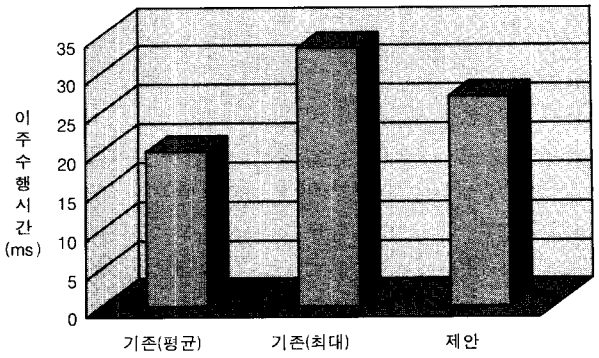
<표 5>은 네트워크 모니터링 모듈에서 각 노드의 상태를 알아보기 위한 알고리즘이다. 이 알고리즘은 주기적으로 수행이 되며 각 노드의 다운여부, 인접 노드간의 지연시간, 시스템 부하 정보, 기타(에러율 등)를 모으는 작업을 수행한다. 또한, 네트워크 환경이 동적으로 변경될 때에는 네트워크 모니터링 알고리즘에서 주기를 동적으로 변경시킬 수 있다. 즉, 네트워크 환경이 자주 변경될 때에는 상태 정보 메시지 주기를 짧게 하여 보다 정확한 네트워크 상태 정보를 얻을 수 있을 것이다.

4. 성능 평가

최단 수행 시간을 이용한 이주 알고리즘의 평가를 위해 시뮬레이션하였다. 시뮬레이션 도구로는 SIMAN(SIMulation ANalysis) 기반의 ARENA 3.0을 이용하였다. SIMAN는 이산형 및 연속형 시스템의 성능을 평가할 수 있는 시뮬레이션 언어이며[12], 미국 System Modeling사에서 1993년에 개발한 ARENA[13, 14]는 SIMAN 지원하는 GUI환경 기반의 시뮬레이션 도구이다.

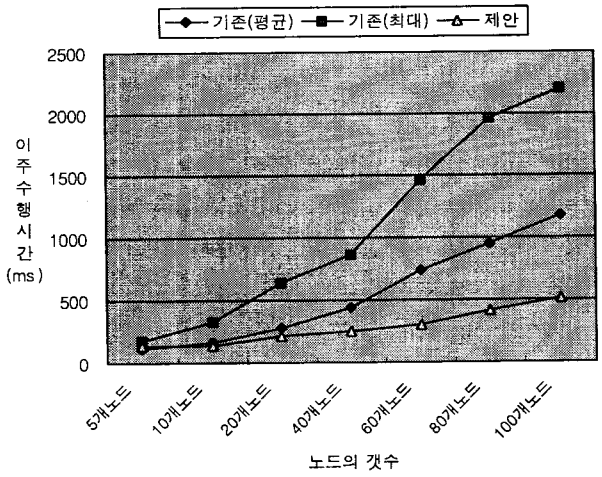
평가는 기존 방식의 사용자가 임의로 이주하기 때문에 임의의 수행 시간에서 평균이나, 최대 수행 시간을 기존 방식으로 하여 측정하였다. 여기에서 기존 방식은 Aglet과 Voyager를 의미하며 이들은 명시적인 이주 방식 시스템[7]이다. 단일 이주 경우와 연속 이주 경우를 구분하여 실험하였으며, 10,000회를 수행하여 평가하였다. 이동 에이전트가 네트워크 노드를 이주할 때 소요되는 시간은 $N(10, 5^2)$인 정규분포(normal distribution)를 이용하였으며, 이주된 노드에서 처리되는 수행 시간은 TRIA(5, 10, 15)의 삼각 함수를 사용하였다.

(그림 4)은 이동 에이전트가 단지 하나의 노드로만 이동할 경우 즉, 단일 이주인 경우이다. 본 논문에서 제시하고 있는 에이전트 이주 정보 시스템에서 소요되는 시간과 이주 후 목적지 노드에서 수행되는 시간은 무시하였으며, 단지 이동 에이전트가 목적지 노드에서 원래 노드를 이주할 때 걸리는 네트워크 수행 시간을 나타낸 것이다.



(그림 4) 단일 이주 평가

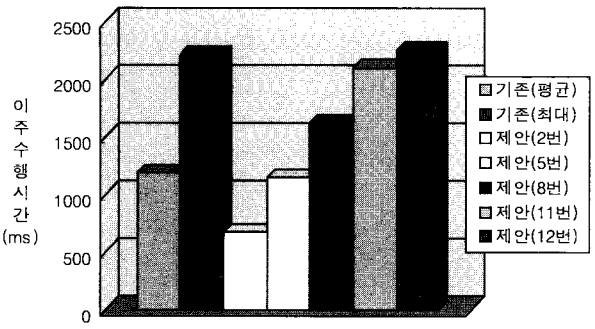
(그림 4)처럼 기존 사용자가 임의로 수행 순서를 정할 경우의 평균 소요 시간과 본 논문에서 제안한 이주 알고리즘을 이용하여 수행되는 시간을 보면 대략 26%정도가 증가되었는데 이는 본 논문에서 제시하는 에이전트 이주 정보 시스템으로 이주한 후에 최단 거리로 목적지 노드로 이주한 결과이다. 그러나, 만약 사용자가 정한 이주 순서가 최악의 경우로 최대 수행 시간이 걸린다면 본 논문에서 제시하고 있는 이주 알고리즘이 약 18%정도가 빠르게 수행되었다.



(그림 5) 연속 이주 평가

(그림 5)에서 이주할 노드가 5개인 경우 제안한 이주 알고리즘이 기존 평균과 비교하여 대략 28% 정도 늦게 수행되었으며, 10개인 경우에는 15%정도 빠르게 수행되었다((그림 5)에서는 나타나 있지 않으나, 노드가 8개에서 9정도사이에서 제안한 이주 알고리즘과 기존 평균값과 비슷한 결과를 나타내었다.). 노드가 20개인 경우 기존 평균과 제안한 경우 약 24% 정도 제안한 경우의 수행 시간이 적게 걸렸다. 40개인 경우는 대략 42%정도이며 60개에서 100개인 경우는 약

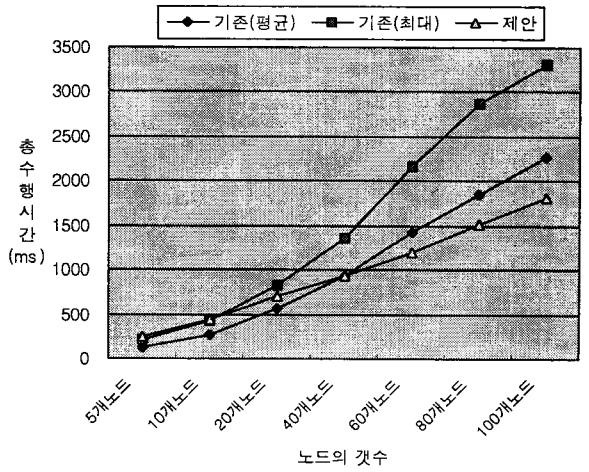
60% 정도 속도 차이가 발생되었다. 노드가 40개 이상인 경우에서 본 논문에서 제시하고 있는 알고리즘과 기존 평균 이주 수행 시간을 비교하면 1/2배 정도로 차이가 발생함을 알 수 있다. 또한, 사용자의 이주 순서가 최악인 경우와 비교해 보면, 본 논문에서 제안한 노드의 방식과 노드의 개수가 20개일 경우 대략 67%, 40개일 경우 70%정도, 60개에서 100개일 경우는 약 80%정도로 제안한 이주 알고리즘의 수행 시간이 빨랐다.



(그림 6) 100개의 노드에서 연속 이주 평가

(그림 6)는 노드의 개수를 100개로 정한 후 기존 사용자가 임의로 선택할 경우의 평균과 최악의 경우인 최대 수행 시간과 본 논문에서 제안한 이주 알고리즘이 에이전트 이주 정보 시스템에 최단 거리를 요청한 횟수와의 비교이다. 예를 들어 제안(2번)은 이동 에이전트가 이주하기 전에 에이전트 이주 정보 시스템에 최단 거리를 요청하고, 중간에(50번째 노드에서) 다시 최단 거리를 요청한 경우이다. 네트워크 트래픽은 시간에 따라 변경된다. 따라서 네트워크 노드의 최단 거리도 시간에 따라 다르므로 이에 대한 평가는 (그림 6)과 같다. (그림 6)와 같이 본 논문이 제안하고 있는 에이전트 이주 정보 시스템의 요청 횟수가 많으면 많을수록 이주 수행 시간이 증가하고 있으며, 이는 최단 이주 순서를 요청한 노드에서 AMIS로 요청 메시지가 송신되는 시간과 AMIS로 응답 메시지가 요청 노드로 전달되는데 걸리는 시간 때문이다. 즉, 수행되어할 노드에서 이동에이전트가 다음 노드로 이주할 때 AMIS로 최단 이주 순서를 요청하면 요청할수록 메시지의 송수신 많아지기 때문에 점점 이주 수행 시간이 길어짐을 의미한다. 본 논문의 평가에서는 100개의 노드에서 대략 5번 정도 요청하면 평균적으로 이주하는 수행 시간과 비슷함을 알 수 있으며, 12번 이상 최단 수행 시간을 AMIS에게 요청하면, 최대 수행 시간을 걸려서 수행 한 경우보다 더 수행 시간이 길어짐을 알 수 있다. 이는 최단 수행 순서 메시지가 요청 노드와 AMIS간에 송수신 할 때 걸리는 시간 때문이다.

(그림 7)은 이동 에이전트가 각 노드 수행 시간 및 AMIS의 수행 시간을 고려한 연속이주의 총 수행 시간을 나타낸 것이다. 노드의 개수가 5개일 경우에는 제안한 방식의 수행



(그림 7) 연속 이주 평가 - 노드 수행 시간 고려

시간이 기존 최대 수행 시간보다도 약 18%정도 늦게 수행되었으며, 노드의 개수가 10개에서 11개 사이일경 제안한 방식의 수행 시간이 기존 최대 수행 시간과 비슷하게 수행되었다. 노드의 개수가 20개일 경우에는 기존 방식의 평균 수행 시간 보다 23%정도 늦게 수행되었다. 노드의 개수가 40개일 경우 본 논문에서 제시한 방법과 기존 평균 수행 시간과 비슷하였다. 그러나 노드의 개수가 60개일 경우에는 16%, 80개일 경우 17%, 100개일 경우에는 20%정도 빨랐다. 제안 방식과 기존의 최대 수행 시간을 비교한 결과 이주 수행 시간을 고려한 경우 노드의 개수가 20개일 경우에는 약 14%정도, 40개일 경우에는 대략 30%, 60개에서 100개까지는 평균 45%정도로 빠르게 수행되었다.

5. 결 론

네트워크 상에 분산되어 있는 정보를 수집할 필요가 있는 때 이동 에이전트 사용은 기존 분산 파라다임의 RPC와 비교하여 성능이 높다. 네트워크상에서 연속적으로 이주하는 이동 에이전트의 효율적인 이주 알고리즘은 사용자의 작업의 총 수행 시간을 줄일 수 있다.

본 논문은 기존 이동 에이전트 이주의 문제점을 해결하기 위해 AMIS를 제안하였다. AMIS는 이주 정보가 필요한 이동 에이전트에게 네트워크의 전체적인 이주 정보 제공이 목적이다. ARENA로 성능 평가 결과 노드 수행 수간을 고려한 단일 이주일 경우에는 제안한 방법이 더 평균보다 40%, 최대보다 20%정도 더 걸려서 수행되었다. 이는 이동 에이전트가 단일 이주인 경우 제안된 방법으로 수행하면, 제안한 AMIS로 에이전트 이주 순서를 요청하여 생기는 메시지 전송(송/수신) 시간 때문이다. 노드 개수가 20개인 연속 이주의 경우 평균 수행 시간과 비교하여 약 24%정도 제안한 알고리즘의 수행 시간이 적게 걸렸다. 노드의 개수가 40개에서 100개일 경우에는 평균 이주 수행 시간과 비

교하여 1/2배 정도로 차이가 발생하였다. 또한 이때 계속적으로 최대 이주 순서로 수행하는 경우와 수행 시간과 비교하면 70%~80%로 빠르게 수행되었다.

참 고 문 헌

[1] Gray, R., Kotz, D., Nog, S., Rus, D., Cybenko, G., "Mobile agents : the next generation in distributed computing," Parallel Algorithms/Architecture Synthesis, 1997.

[2] Vu Anh Pham, Karmouch, A., "Mobile software agents : an overview," IEEE Communications Magazine, Vol.36, Issue 7, July, 1998.

[3] C. G. Harrison., D. M. Chess and A. Kershenbaum, "Mobile Agents : Are they a good idea?," IBM Research Division, Number RC 19887, 1995.

[4] Moridera, A., Murano, K., Mochida, Y., "The network paradigm of the 21st century and its key technologies," IEEE Communications Magazine, Vol.38, Issue 11, Nov., 2000.

[5] D. B. Lange, Mitsuru Oshima, "Mobile Agents with Java : The Aglet API," WWW Journal, 1998.

[6] Misikangas, P., Raatikainen, K., "Agent migration between incompatible agent platforms," Distributed Computing Systems, 2000.

[7] Schulze, B., Madeira, E. R. M., "Migration transparency in agent systems," Autonomous Decentralized Systems, 1999.

[8] IBM's Aglets, http://www.trl.ibm.com/aglets.

[9] Object Space's Voyager, http://www.objectspace.com/Voyager.

[10] Genral Magic's Odyssey, http://www.genmagic.com/agents.

[11] Caripe, W., Cybenko, G., Moizumi, K., Gray, R., "Network awareness and mobile agent systems," IEEE Communications Magazine, Vol.36, Issue 7, July, 1998.

[12] Sturrock, D. T., Pegden, C. D., "Introduction to SIMAN," Simulation Conference, 1990.

[13] ARENA, http://www.eos.ncsu.edu/software/software_index/arena.html.

[14] W. David Kelton, Randall P. Sadowski, Simulation with ARENA, McGraw-Hill Com., 1998.

[15] Ichiro Satoch, "Adaptive Protocols for Agent Migration," Proceedings of IEEE International Conference on Distributed Computing Systems(ICDCS '2001), 2001.

박 홍 진

e-mail : hjpark1@mail.sangji.ac.kr

1993년 원광대학교 컴퓨터공학과 졸업(공학사)

1995년 중앙대학교 컴퓨터공학과(공학석사)

2001년 중앙대학교 컴퓨터공학과(공학박사)

2001년~현재 상지대학교 이공과대학 컴퓨터정보공학부 전임강사

관심분야 : 에이전트, 실시간 시스템, 분산 시스템