최단 수행 순서 제공을 위한 
에이전트 이주 정보 시스템 설계

박 홍 진

요 약

이동 에이전트가 연속적으로 다른 노드로 이주할 때 중 작업 수행 시간은 빈도에 따라 차이가 날 수 있다. 효율적인 이주 알고리즘은 초기 상태에서 목적 상태로 수행되기 위해 최소한의 수행 시간을 제공하는 순서를 의미하며, 이동 에이전트의 활성화로 이주 알고리즘을 통해 개발자가 원하는 작업의 중 수행 시간을 최소화시킬 수 있다. 기존의 대표적인 이동 에이전트로 Apllets, Voyager, Odyssey 등은 이주할 때 네트워크 상의 이주할 노드의 상태를 고려하지 못함으로써 효율적인 이주를 보장할 수 없다. 본 논문은 이동 에이전트에서 효율적인 이주를 위해 AMIS를 제안한다. AMIS는 이동 에이전트의 중 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 spent for the process can be minimized by adopting an effective migration algorithm. Existing mobile agents such as Apllets, 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. 서론

이동 에이전트(mobile agent)는 네트워크를 통해 이주하면서 자신의 제어하에 유용한 수행을 할 수 있는 프로그램이다. 기존 클라이언트/서버 패러다임에서 네트워크를 통해 전송하는 데이터를 전송하는 것과 다르게 이동 에이전트는 데이터가 있는 노드로 이동하여 이주하여 작업을 수행하며 결과를 가지고 원래의 노드로 돌아온다[1, 2]. 따라서 네트워크 상에서 많은 데이터 전송이 필요할 경우 시간적인 문제에 있는 데이터를 적절한 위치에 전송하여 데이터 접근을 첨단화하고 기존 클라이언트/서버 패러다임보다 이동 에이전트 패러다임은 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 수행할 수 있다[3]. Code-on-Demand 패러다임은 클라이언트 측으로 서버의 프로그램이 이동된 에이전트처럼 코드가 이주하므로, 여러 노드로 장기적인 이주의 지원할 수 없다. 따라서 분산 컴퓨팅 패러다임에서 이동 에이전트 기술은 네트워크의 효율성을 높이는 용용 프로그램에서는 다른 분산 패러다임에 비해 낮은 대역폭을 이용하면서 견고한 수행을 할 수 있다[4, 5].

이동 에이전트가 연속적으로 다른 노드로 이주될 때 중 작업 수행 시간은 빈도에 따라 차이가 날 수 있으므로 이주 알고리즘(migration algorithm)은 초기 상태에서 목적 상태로 수행되기 위해 최소한의 수행 시간을 제공하는 수행 순서를 의미하며, 이동 에이전트의 효율적인 이주 알고리즘을 통해 개발자가 원하는 작업의 종 수행 시간을 최소화시킬 수 있다. 효율적인 이주 알고리즘
증은 노드의 단결, 노드의 다중, 네트워크 구조 변경 등 네트워크 환경이 동적으로 변경되었을 경우에 더욱 중요하다.

현재 개발된 대표적인 이동 에이전트 시스템인 Aglets[8], Voyager[9], Odyssey[10] 등에서 에이전트 이주는 명백히 로 프로그램을 개발하는 개발자가 순서를 정하여 수행되고 있다. 이러한 이동 에이전트의 이주 순서는 최적의 이동을 목표로 하는 노드의 시스템 상태에 고려하지 않고 있다.

본 논문은 이동 에이전트가 연속적으로 이주할 때 최적 이주 시간을 제공하기 위해 에이전트 이주 정보 시스템(AMIS: Agent Migration Information System)를 제안한다. 본 논문에서 제안하고 있는 이동 정보 시스템의 목적은 사용자에게 서비스의 위치 정보를 제공함과 동시에 에이전트의 효율적 실행을 최소화하고 안정한 고속 이동 수행 순서를 제공한다. 에이전트 시스템의 이주

2. 기존 이동 에이전트 시스템의 이해

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

2.1 Aglets


2.2 Voyager

오브젝트스페이스(ObjectSpace) 사에서 만든 Voyager는 순수 자바의 100% 호환된 자바로 구현된 이동 에이전트 시스템이다[9]. Voyager는 원격 객체의 활성화 기능, CORBA에서 사용할 수 있는 IDL, IOP 등의 기능이 모두 지원된다. Voyager는 자바 메시지를 에이전트에게 전송할 수 있는 기능이 있다. Voyager 에이전트 이주는 다음과 같다. 첫째, 이동할 객체를 생성한다. 둘째는 생성된 객체를 얻는다. 이때 사용하는 메소드는 moveTo()이다. 셋째, 에이전트를 이주시키며 사용하는 메소드는 moveToStoringURL이다(그림 2).

(그림 1) Aglets에서 에이전트 이주

(그림 2) Voyager에서 에이전트 이주

2.3 Odyssey

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

본 논문에서 제시하고 있는 효율적인 이주 방법은 위치 정보를 제공함과 동시에 에이전트의 종 수행시간을 최소화하고 안전하게 겪을 수 있도록 이주를 위해 최적 수행 순서의 제공이 목적이이다. 에이전트 이주 정보 시스템은 에이전트 이주 정보 모듈(AMIM: Agent Migration Information Module)과 네트워크 모니터링 모듈(NMM: Network Monitoring Module)으로 구성되어 있다(그림 3).

![Diagram](image)

(그림 3) 에이전트 이주 정보 시스템의 구조

사용자가 원하는 서비스가 로봇이 아닌 위치에 있을 경우 요청 에이전트는 원격의 최적 경로를 위해 AMIS에 요청한다. AMIS는 네트워크 상황과 시스템의 부하 정보를 제공하여 사용자가 에이전트가 요청한 서비스를 수행할 수 있는 서비스 에이전트가 존재하는 노드에 대한 최적의 목록을 제공하는 것을 목적으로 한다. 에이전트 이주 정보 모듈은 각 노드가 제공하는 서비스에 대한 정보를 제공하고 있다. 따라서 사용자에게 요청된 서비스와 네트워크

정상 수행 순서 제공을 위한 에이전트 이주 정보 시스템 설계 557

지정 주소 사이를 매핑시키려 한다. 네트워크 노선의 모듈은 최단 경로 설정을 위한 전체 네트워크의 정보를 수집하며, 기존의 최단 경로 설정 알고리즘과 관련된 문제를 해결하는 알고리즘을 있다.

3.1 에이전트 이주 정보 모듈

AMIS의 구성 요소인 에이전트 이주 정보 모듈(AMIM)은 이주 알고리즘과 서비스 위치 정보 데이터를 유지하고 있다. 서비스 위치 정보 테이블은 사용자가 요청하는 서비스 이름과 해당 서비스를 수행할 수 있는 에이전트 이름, 위치 정보에서 해당 에이전트가 존재하는 위치 정보, 실행 여부를 나타내는 가용성과 해당 노드의 부하 정보로 구성되어 있다. 하나의 노드에는 여러 개의 에이전트가 존재할 수 있으며, 하나의 에이전트는 여러 개의 서비스를 제공할 수 있다. 가용성은 0이 1을 지니고 있는데 1일 경우 해당 노드는 상태가 아님을 나타내고, 0일 경우 해당 노드는 잠재가 발생되었음을 나타낸다. 가용성과 시험 결과에 대한 정보는 네트워크 모니터링 모듈에서 추적적으로 정보를 얻는다.

<table>
<thead>
<tr>
<th>SERVICE NAME</th>
<th>Agent Name</th>
<th>Location Information</th>
<th>Availability</th>
<th>Load Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service A</td>
<td>AA</td>
<td>111.111.12.xxx</td>
<td>1</td>
<td>60%</td>
</tr>
<tr>
<td>Service B</td>
<td>BB</td>
<td>111.111.12.xxx</td>
<td>1</td>
<td>60%</td>
</tr>
<tr>
<td>Service C</td>
<td>CC</td>
<td>222.222.17.xyz</td>
<td>1</td>
<td>20%</td>
</tr>
</tbody>
</table>

단일 이주는 수신측이 하나로써 이동 에이전트는 수신측으로 이동한 후에 재작업을 수행하며 수행이 끝난 후 원래 송신 채로 되돌아오는 경우이다. 연속 이주는 사용한 노드 의 개수가 여러 개인 경우를 의미하며, 선택된 노드의 개수는 다음과 같이 수식으로 표현된다.

\[ 2 \leq M \leq N \]

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

| RSIN = Request Service Names |
| BN = Request Node |
| SN = Service Node |
| ASIN = 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;  
else 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;  

연속이주는 이동 이전트 이주할 때 이주할 수가 정해진 경우와 순서가 정해지지 않는 경우로 구분될 수 있다. 이동 이전트가 연속적으로 이주할 때 알고리즘은 〈표 2〉와 같다. 〈표 2〉는 이동 이전트의 이주순서가 고정된 경우와 고정되지 않은 경우를 각각 고려하였으며, 이에 대한 설명은 아래와 같다.

1. 이주할 수가 정해지면서 이주 노드가 여러 개인 경우  
   하나의 요청 이전트가 이주하는 노드가 하나인 경우  
   이주할 순서는 여러 개인 경우
2. 이주할 순서가 정해지지 않으면서 이주 노드가 여러 개인 경우
   이 경우는 ①의 경우와 비슷하여 연속이주의면 이주할 순서가 정해지지 않은 경우이다. 이주할 순서가 정해지지 않은 경우는 예는 정보 수집이나 점검 및 편리성할 경우 특정 사이트에 대한 순서가 정해지지 않은 경우다.

3.1.1.1 단일 이주 수행 시간
   이동 이전트에서 이주되는 정보는 이전트 코드와 상태 정보와 전송되기 전에 마샬링을 해야한다[5]. 마샬링 (marshalling)은 상태 정보와 데이터를 상태론으로 전송될 수 있는 메시지로 재정기시 하는 것을 의미한다. 마샬링된 형태로 전송된 이동 이전트는 수행 중에서 서비스를 수행하기전에 마샬링된 정보를 원래 상태로 복원하는 언마샬링 (unmarshalling)이 필요하다. 즉, 수신측에서 언마샬링된 후 서비스 수행 작업을 수행한다. 수행될 결과는 송신측에서 전송된 이전트에 추가하여 송신측으로 되돌아간다. 이때 결과 역시 마샬링이 필요하다. 이전트가 단일 노드에 이주하여 수행하는데 걸리는 총 수행 시간을 표현하기 위해 사용된 기호는 〈표 3〉과 같다.

〈표 3〉 이주에서 사용된 기호

<table>
<thead>
<tr>
<th>의미</th>
<th>기호</th>
</tr>
</thead>
<tbody>
<tr>
<td>이전트 코드</td>
<td>Acode</td>
</tr>
<tr>
<td>상태 정보</td>
<td>Astat</td>
</tr>
<tr>
<td>네트워크 처리율</td>
<td>r</td>
</tr>
<tr>
<td>마샬링/언마샬링 시간</td>
<td>μ</td>
</tr>
<tr>
<td>수행 시간</td>
<td>ω</td>
</tr>
</tbody>
</table>

이동 이전트는 다음과 같이 표현될 수 있다.

A = (Acode, Astat)

위치 L1에서 L2로 이전트 이주에 대한 네트워크 부하는 다음과 같이 계산되어진다.

\[ A_{net}(L1, L2, A) = A_{code} + A_{state} \]

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

\[ T_{net}(L1, L2, A) = \frac{A_{code}(L1, L2, A)}{r(L1, L2)} + 2μ(A_{code} + A_{state}) \]  

(1)

이동 이전트가 L1에서 L2로 이주한 후에 언마샬링한 다음, 목적지 노드에서 작업을 수행한 후 다시 L1으로 전송되며 총 수행 시간은 다음 식 (2)와 같이 나타낼 수 있다.

\[ T(L1, L2, A, r, A_{net}) = T_{net}(L1, L2, A) + ω \]

\[ + 2μA_{net} + \frac{A_{net}(L1, L2, A)}{r(L1, L2)} \]  

(2)
식 (2)를 풀면 다음과 같다.
\[
\frac{(A_{\text{code}} + A_{\text{new}}) + A_{\text{net}}}{\tau(L1, L2)} + \omega + 2\mu(A_{\text{code}} + A_{\text{new}}) + 2\mu A_{\text{net}} = 0
\]

3.1.1.2 연속 이후 수행 시간
노드의 개수가 3개인 경우에는 이주할 수 있는 경우의 수는 2가지로 단순하다. 그러나, 이주 노드가 증가함에 따라 수록 이주할 수 있는 경우의 수는 지수적으로 증가하게 된다. 이주 후 순서가 정해지지 않은 경우 이주할 수 있는 경우의 수는 원래 노드를 제외하고 따라서 노드의 개수로 이주가 먼저로 서초 다개중에서 n-1개를 대하여 일별로 나열하는 순열(permutation)이다.
노드의 집합 S = \{1, 2, 3, ..., n\}일 때 이주할 수 있는 경우의 수는 다음과 식 (3)과 같이 나타낼 수 있다.
\[
P_0 = n(n-1)(n-2)(n-3) \times \cdots \times 2
\]

따라서, 연속 이주에서 목적지가 없을 경우에는 이주한 목적지 노드를 최소 경로로 이주하는 문제로 집합 S의 최소 경로로 이주에 대한 문제이며 다음과 같은 식 (4)로 나타낼 수 있다.
\[
C(n) = \sum_{i=1}^{n-1} d_{\text{eff}, i, i+1} + d_{\text{eff}, n, 1}
\]

식 (4)에서 \(d_{\text{eff}, i, i+1}\)는 노드 \(i\)에서 \(i+1\)까지의 네트워크 거리를 나타낸다. 본 논문에서는 노드 \(i\)에서 \(i+1\)까지의 네트워크 거리를 의미하기도 한다. 최소 경로로 이주할 수 있는 \(d_{\text{eff}, i, i+1}\)와 \(d_{\text{eff}, n, 1}\)의 최소값을 구하는 것이다. 주, 최소 경로 수행 시간을 구하는 것은 \(C(n)\)의 최소 값과 시스템의 부하 정보를 고려하여 최적의 경로를 찾는 문제이다. 식 (4) 최적의 이주 경로를 결정하는 것은 외관 문제(Travelling Salesman Problem)를 해결하는 문제와 비슷하다. 외관 문제는 NP-complete 문제로 알려져 있다.

단일 이주는 하나의 수신자만 있는 경우지만 연속 이주는 이동 예상자가 하나 이상의 사이트로 이주하는 것을 의미한다. n개의 노드를 이주한다면 점차는 시간을 계산하기 위해 n개의 노드에서 일의 노드를 \(i\)라고 하자 (1 \leq i \leq n). n개의 노드에서 효율적인 이주 방법은 이주 시간을 최소화함으로써 총 수행 시간을 최소화시키는데 목적이 있다.

Si번째의 수행되는 시간은 다음과 같다.
\[
S_i = S_{i-1} + A_{\text{net}}(L_{i-1}, L_i, A) + 2\mu(A_{\text{new}} + A_{\text{net}})
\]
\[
+ 2\mu A_{\text{net}}(i-1) + \frac{A_{\text{net}}(i-1)}{\tau(L_{i-1}, L_i)}
\]

3.2 네트워크 모니터링 모듈
네트워크 모니터링 모듈 <표 4>와 같이 다음 3가지로 분류된다.[11]

<table>
<thead>
<tr>
<th>분류 기준</th>
<th>모니터링 기법</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>주기적이나 요구 시기에 따라</td>
<td>주기적으로 네트워크를 모니터링하여 애플리케이션 상태 정보를 요구하였을 때 미리 처리한 정보를 제공한다.</td>
<td></td>
</tr>
<tr>
<td>요구 메시지를 통해 모니터링</td>
<td>애플리케이션의 특성 노드에 대한 네트워크 요구 정보에 대한 정보를 제공한다.</td>
<td></td>
</tr>
<tr>
<td>장점 집중형과 분산형</td>
<td>장점 집중형 전체 네트워크 정보를 수집 시스템에서 모두 관리한다. 장점 분산형 각 노드에서 네트워크 상태 정보를 모니터링한다. 다른 네트워크 상태 정보를 얻기 위해서는 다른 노드에게 요청하여 정보를 얻는다.</td>
<td></td>
</tr>
</tbody>
</table>

수동적인 모니터링은 네트워크 트래픽을 적재 차지하지만 두 노드간에 전송되는 메시지 횟수가 적으면, 충분한 네트워크 정보를 획득하지 못한다. 또한, 노드간 전송되는 메시지가 많음에 따라 많은 수의 네트워크 정보를 획득할 수 있다. 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 획득함으로써 효율적일 수 있다. 장점 집중형 모니터링 방법은 확장성과 고밀도에 문제를 발생할 수 있다. 만약 정보의 수렴이 충분장 시스템의 고장이 발생하면 전체 네트워크 모니터링 정보가 활용될 수 없다. 따라서 일반적으로 장점 집중형 모니터링 방법에서 수집된 네트워크 모니터링 정보는 사용된다. 분산형은 장점 집중형보다 간단하다고 복잡한 경계가 필요하다. 이동 예상자의 효율적인 이주 방법을 고려할 때 본 논문에서 사용하는 네트워크 모니터링 모듈은 위에서 기술한 모니터링 방법중 위치 집중형 특성을 가지면서 추가적으로 모니터링 상태 정보를 전송시키는 능동적인 모니터링 기술이 필요하다.
<표 5> 네트워크 모니터링 알고리즘

```java
while (true) {
    // wait until the next collecting time
    sleep (Freq);
    // send msg to all node
    sendto (all node);
    // wait to get reply msg from all slave machine
    wait (1);
    check failure;
    get reply msg;
    store (adjacent_latency);
    store (system_load);
    store (free_memory_size);
}
```

4. 성능 평가


평가는 기존 방식의 사용자가 임의로 수행하는 수행 시간에서 평균이, 최대 수행 시간을 기준 방식으로 하여 측정하였다. 여기에서 기존 방식은 Aglet과 Voyager를 의미하며 이들은 분산형의 이중 방식 시스템[7]이다. 단일 이중 경우와 연속 이중 경우를 구분하여 실험하였으며, 10,000회를 수행하여 평균하였다. 이중 에이전트가 네트워크 노드를 이주할 때 소요되는 시간은 \( M(10,5) \)인 정규 분포(normal distribution)를 이용하였으며, 이중된 노드에서 처리되는 수행 시간은 TRIA(5,10,15)의 삼각 함수를 사용하였다.

(그림 4)은 이중 에이전트가 단지 하나의 노드로만 이동할 경우 즉, 단일 이중 체계이다. 본 논문에서 제시하고 있는 에이전트 이중 정보 시스템에서 소요되는 시간과 이 주 후 목적지 노드에서 수행되는 시간은 무시하였으며, 단지 이중 에이전트가 목적지 노드에서 왼쪽 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 이중 알고리즘 을 이용하여 수행되는 시간을 보면 다락 28% 정도가 증가 되었는데 이는 본 논문에서 제시하는 에이전트 이중 정보 시스템으로 이주한 후에 최대 거리로 목적지 노드로 이주한 결과이다. 그러나, 단일 사용자가 아닌 이중 순서가 최악의 경우로 최대 수행 시간이 점차 대물 본 논문에서 제시하고 있는 이중 알고리즘 이 약 18%정도가 빠르게 수행되었다.

(그림 5) 연속 이주 평가

(그림 5)에서 이중의 노드가 5개인 경우 제안한 이중 알고리즘의 기존 평균과 비교하여 높은 28% 정도 늦게 수행되었으며, 10개인 경우에는 15%정도 빠르게 수행되었다(그림 5)에서는 나타나 있지 않으나, 노드가 8개에서 9개 사이에 에서 제한된 이중 알고리즘과 기존 평균과 비교한 결과를 나타내었다). 노드가 20개인 경우 기존 평균과 제한한 경우 약 24% 정도 제한한 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 방법이 각각의 수행 시간을 비교하면 1/2배 정도로 차이가 발생함을 알 수 있다. 또한, 사용자의 이주 손서가 최악인 경우에 비해 보면, 본 논문에서 제안한 노드의 방법과 노드의 개수 가 20개인 경우 대략 67%, 40개인 경우 70% 정도, 60개에서 100개인 경우는 약 80% 정도로 제한한 이주 알고리즘의 수행 시간이 빨라졌다.

(그림 6) 100개의 노드에서 연속 이주 평가

(그림 7) 연속 이주 평가 - 노드 수행 시간 고려

시간이 기존 최대 수행 시간보다도 약 18% 정도 늦게 수행 되었으며, 노드의 개수가 10개에서 11개 사이일경 제한된 방식의 수행 시간이 기존 최대 수행 시간과 비슷하게 수행되었다. 노드의 개수가 20개인 경우에는 기존 방식의 평균 수행 시간보다 22% 정도 늦게 수행되었다. 노드의 개수가 40개인 경우 본 논문에서 제시한 방법과 기존 평균 수행 시간과 비슷하였다. 그러나 노드의 개수가 60개인 경우에는 16%, 80개인 경우 17%, 100개인 경우에는 20% 정도 빨라졌다. 제한 방식과 기존의 최대 수행 시간을 비교한 결과 이주 수행 시간을 고려한 결과 노드의 개수가 20개인 경우에는 약 14% 정도, 40개인 경우에는 약 30%, 60개에서 100개까지는 평균 45% 정도로 빠르게 수행되었다.

5. 결론

네트워크 상에 분산되어 있는 정보를 수집할 필요가 있는 경우에 이동 에이전트 사용은 기존 분산 파티다임의 RPC와 비교하여 성능이 높다. 네트워크상에서 연속적으로 이주 하는 이동 에이전트의 효율적인 이주 알고리즘이 사용자의 작업의 종 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. ARENA는 성능 평가 결과 노드 수행 시간을 고려한 단일 이주 이주 경우에는 제한된 방법이 더 평균보다 40%, 최대보다 20% 정도 더 빠르게 수행되었다. 이는 이동 에이전트가 단일 이주인 경우 제한된 방법으로 수행하면, 제한한 AMIS로 이동 에이전트 이주 순서를 수행하여 총 수행 시간을 향상시킨다. 노드 개수가 20개인 경우 이주 수행 시간을 비교하여 약 24% 정도 제한된 알고리즘의 수행 시간이 크게 줄었다. 노드의 개수가 40개에서 100개인 경우에는 평균 이주 수행 시간과 비교하
교하여 1/2배 정도로 차이가 발생하였다. 또한 이내 계속적으로 최대 이상 순서로 수행하는 경우와 수행 시간과 비교하여 70% ~ 80%로 빠르게 수행되었다.

참고 문헌


박홍진

e-mail : hjpark1@mail.sangji.ac.kr
1993년 한성대학교 컴퓨터공학과 졸업 (공학사)
1995년 중앙대학교 컴퓨터공학과(공학 서열)
2001년 중앙대학교 컴퓨터공학과(공학 박사)
2003년 ~ 현재 상지대학교 이공과대학 컴퓨터정보공학부 전임교수
관심분야 : 에이전트, 실시간 시스템, 분산 시스템