2001년 한국정보처리학회 추계 학술발표논문집 제8권 제2호

이동에이전트 기반 워크플로우 시스템에서 에이전트 위임모델의 부하분산

°유정준[1], 이동익[1], 정승욱[2], 김중배[2]
[1]광주과학기술원 정보통신공학과
[2]한국전자통신연구원 전자상거래연구부 SCM연구팀
{jjyoo, dilee}@kjist.ac.kr, {swjung, jjkim}@etri.re.kr

Load Distribution of Agent Delegation Model in Mobile Agent Based Workflow Systems

Jeong-Joon Yoo°, Dong-Ik Lee, Seung-Woog Jung, and Joong-Bae Kim
Department of Information and Communications
Kwang-Ju Institute of Science and Technology(K-JIST)
SCM Team, Department of EC, ETRI

요 약

에이전트 위임모델 기반 워크플로우 시스템은 기존 클라이언트 서버기반 워크플로우 시스템과 이동에이전트 기반 워크플로우 시스템보다 워크플로우 수가 증가하고 워크플로우 구조가 복잡해짐에 따라 보다 좋은 성능 및 확장성을 제공한다. 이는 에이전트 위임모델이 워크플로우 엔진에서 발생하는 병목현상을 제거하여 작업수행자(Task Performer)라 명명된 호스트들로 워크플로우 엔진의 부하를 자연스럽게 분산시키기 때문이다. 에이전트 위임모델은 워크플로우 엔진의 부하를 제거하지만, 작업수행자의 부하를 고려하지 않고 작업을 할당하기 때문에, 작업수행자에 병목현상을 발생시킬 수 있다. 이는 에이전트 위임모델의 목적인 워크플로우 시스템의 성능 및 확장성 향상을 저해하는 요인이다. 따라서, 워크플로우 시스템의 성능 및 확장성을 향상시키기 위해서는 작업수행자에 발생하는 병목현상을 제거해야 한다. 본 논문에서는 작업수행자의 병목현상을 제거하기 위해 응답리스트(Response List)를 제안하며 응답리스트 관리방법을 제시한다. 이러한 응답리스트가 에이전트 위임모델 기반 워크플로우 시스템의 성능 및 확장성 향상에 어떠한 영향을 미치는지 성능평가를 수행한다.

1. 서 론

워크플로우 시스템이란, 하나 또는 그 이상의 워크플로우 엔진을 통해 워크플로우를 정의, 관리, 그리고 수행할 수 있는 시스템을 말한다[1]. 워크플로우 시스템의 성능 및 확장성을 향상시키기 위해 많은 연구가 진행되고 있다[2, 3, 4]. 기존 워크플로우 시스템으로는 클라이언트 서버 모델 기반 그리고 이동에이전트 기반 워크플로우 시스템이 있으며, 클라이언트 서버 모델 기반 워크플로우 시스템 내의 중앙 집중화된 서버는 워크플로우 시스템의 성능 및 확장성을 저하시키는 주요 원인으로 지적되었다[3]. 보다 향상된 확장성 향상을 위해 이동에이전트 기반 워크플로우 시스템이 DartFlow [5]에서 제안되었다. 이동에이전트란 네트워크로 연결된 컴퓨터들을 사용자 또는 다른 시스템을 대신하여 작업을 수행하며 이동위치를 스스로 결정할 수 있는 자율성(autonomy)과 이동성(mobility)을 가진 소프트웨어 프로그램이다[6]. 워크플로우 시스템 내에서 이동에이전트의 장점은 워크플로우 시스템의 확장성 향상, 동적변경의 용이 및 오류가 자주 발생하는 네트워크에 대해 보다 신뢰성 있는 원격연산(remote operation)을 제공한다는 것이다. 그러나, 이동에이전트 기반 워크플로우 시스템은 워크플로우 수행에 필요한 전체정보를 이동에이전트가 갖고 이동(에이전트 이동 오버헤드라 정의함)하기 때문에 이동하는데 많은 시간을 필요로 한다. 이는 결국 워크플로우 시스템의 성능을 저하시키는 요인으로 작용한다. 이를 해결하기 위해 에이전트 위임모델[3]이 제안되었다. 에이전트 위임모델은 클라이언트 서버 모델 기반 워크플로우 시스템이 갖는 확장성 부족의 문제와 이동에이전트 기반 워크플로우 시스템이 갖는 에이전트 이동 오버헤드(agent migration overhead) 에 의해 발생하는 성능 저하의 문제점을 해결하여 워크플로우 시스템의 성능 및 확장성 향상을 목표로 제안된 방법이다. 즉, 이동에이전트가 워크플로우 엔진에서 작업수행자로 직접 이동하여 작업을 수행함으로써 워크플로우 엔진과 작업수행자간의 원격 상호작용(remote interaction)의 수를 감소시켜 워크플로우 시스템의 확장성을 향상시켰으며, 워크플로우 전체구조를 특정 규칙에 의해 분할함으로써, DartFlow 시스템에서 발생했던 에이전트 이동 오버헤드를 감소시켜 성능향상을 꾀하였다. 에이전트 위임모델은 동시에 수행되는 워크플로우 프로세스의 수가 많을수록, 그리고 워크플로우 구조가 복잡할수록 기존 워크플로우 시스템에 비해 보다 향상된 성능 및 확장성을 제공한다. 그러나, 에이전트 위임모델은 작업수행자의 부하를 고려하지 않고 워크플로우 엔진의 부하를 작업수행자에게 분배함으로써 작업수행자에 병목현상을 발생시킬 수 있는 문제점을 갖고 있다. 작업수행자의 병목현상은 워크플로우 시스템의 성능 및 확장성을 저하시킬 수 있기 때문에, 보다 향상된 성능 및 확장성을 위해서는 작업수행자의 부하를 분산시킬 수 있는 방법이 필요하다. 본 논문에서는 에이전트 위임모델이 갖는 작업수행자에서의 병목현상 발생문제를 명확히 하고, 이러한 문제 해결을 위해 응답리스트(Response List)를 도입한다. 또한 응답리스트 도입이 워크플로우 시스템의 성능 및 확장성에 미치는 영향을 실험을 통해 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 에이전트 위임모델 기반 워크플로우 시스템과 그의 문제점을 설명한다. 3장에서는 에이전트 위임모델이 갖는 문제점 해결을 위해, 응답리스트 관리자를 정의하고 응답리스트 도입으로 인한 부가적인 고려사항을 설명한다. 4장에서는 응답리스트를 도입했을 경우 기존 에이전트 위임모델 기반 워크플로우 시스템의 성능 및 확장성에 어떠한 영향을 미치는지 실험을 통해

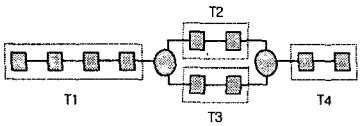


그림 1. 워크플로우 분할

보인다. 끝으로 5장에서는 본 논문의 결론을 맺는다.

2. 에이전트 위임모델 기반 워크플로우 시스템

에이전트 위임모델 기반 워크플로우 시스템[3]은 기존 이동 에이전트 기반 워크플로우 시스템[5]이 갖던 이동 오버헤드를 줄임으로서 워크플로우 시스템의 성능을 향상시키고자 제안된 방법이다. 즉, 워크플로우 전체를 하나의 이동에이전트에게 위임하지 않고, 특정 규칙에 의해 워크플로우를 분할함으로써 분할된 각 블록을 각각의 이동에이전트에게 할당 및 위임하는 방법이다.

2.1 에이전트 위임모델[3]

워크플로우를 분할하는 방법은 여러 가지가 있겠지만 다양한 워크플로우를 분할할 수 있는 정해진 규칙이 있어야 하며 이들은 알고리즘으로 구현 가능해야 한다. [3]에서 소개된 Maximal Sequence Model은 정해진 규칙에 의해 워크플로우 구조를 분할하는 방법으로 워크플로우 구조를 입력받아 Maximal Sequence Path 라 명명된 블록들을 반환하는 워크플로우 분할 규칙이다. 이러한 분할 규칙에 의해 생성된 각 Maximal Sequence Path들은 각각 하나의 이동에이전트에게 위임 및 수행된다.
그림1은 한 워크플로우가 여러 Maximal Sequence Path(T1, T2, T3, 및 T4)로 분할되는 예를 보이고 있다. 분할된 각 Maximal Sequence Path들은 하나의 이동에이전트에게 할당되며, 각 이동에이전트들은 적절한 워크플로우 수행순서에 따라 원격호스트에 존재하는 작업수행자로 직접 이동, 작업(task)을 수행 또는 수행을 요청하게 된다.

2.2 에이전트 위임모델 기반 워크플로우 시스템

그림2는 3계층의 에이전트 기반 워크플로우 시스템의 구조[7]를 보여주고 있다. 빌드타임(build-time)동안, 정의된 모든 워크플로우 마다 Maximal Sequence Model를 적용하여 그림2의 계층1에 존재하는 워크플로우 정의 저장소(workflow definition repository)에 저장한다. 계층2에는 여러 워크플로우 엔진이 존재하며, 워크플로우 일부(Maximal Sequence Path)를 이동에이전트에게 분할하여 주는 역할을 담당한다. 계층2에서 작업을 할당받은 이동에이전트는 작업수행자가 존재하는 계층3으로 이동하여 작업을 수행 후 그 결과를 자신이 생성된 워크플로우 엔진에게 통보한다. 각 계층의 구조 및 제시된 에이전트의 기능은 [7]에서 언급되었다.

2.3 다른 워크플로우 시스템과의 성능 및 확장성 비교

이동에이전트 기반 워크플로우 시스템에 위임모델을 적용하면, 그림2의 계층2와 계층3간의 상호작용의 수를 줄이고 이동에이전트의 크기를 줄임으로써, 에이전트 이동 오버헤드를 줄일 수있어 성능과 확장성이 개선된다.즉, 에이전트 위임모델은 워크플로우 엔진의 부하를 작업수행자들에게 자연스럽게 분산시킴으로써 부하분산의 효과를 볼 수 있다.

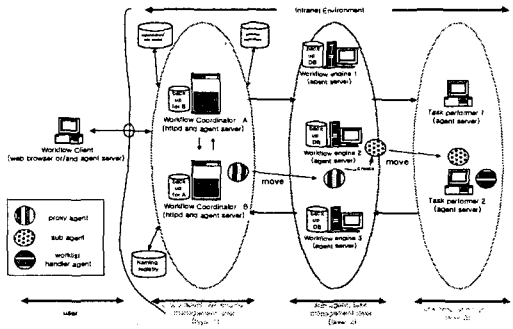
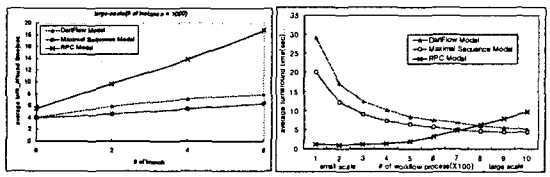


그림 2. 에이전트 위임모델 기반 워크플로우 시스템 구조



(a) (b)

그림 3. 워크플로우 시스템의 성능 및 확장성 비교:(a) # of workflow branch VS. turnaround time, (b) # of workflow process VS. turnaround time

그림3은 에이전트 위임모델기반 워크플로우 시스템과 기존(클라이언트 서버 기반, 이동에이전트 기반) 워크플로우 시스템의 성능 및 확장성을 비교한 결과를 보여준다. 실험결과에 의하면 에이전트 위임모델 기반 워크플로우 시스템이 클라이언트 서버모델 기반 워크플로우 시스템(RPC Model)과 이동에이전트 기반 워크플로우 시스템(DartFlow)에 비해 워크플로우 구조가 더 복잡할수록, 그리고 프로세스의 수가 증가할수록, 보다 좋은 성능 및 확장성(그래프의 기울기)에 있어서 보다 나은 결과를 나타낸다[3]. 그러나, 다음 장에서 제시된 것처럼, 에이전트 위임모델은 작업수행자의 부하를 고려하지 않고, 단지 워크플로우 엔진의 부하를 작업수행자에게 전달함으로써, 어느 경우 작업수행자에서 병목현상을 발생시킬 수 있다.

3. 에이전트 위임모델에 의한 부하분산: 문제 및 해결

본 장에서는 2장에서 설명된 에이전트 위임모델의 문제점을 설명하고 문제점 해결을 위해 응답리스트 관리자를 제안한다.

3.1 작업수행자에서의 병목현상 및 부하분산

그림4는 에이전트 위임모델에 따라 작업이 작업수행자에게 할당되었을 때, 각 작업수행자에게 할당되는 작업의 수를 그래프로 보여주고 있다. 특정 작업수행자(task performer)에게는 많은 작업들이 할당되어 수행을 위해 대기중인 작업수가 고르지 않음을 볼 수 있다. 이는 현재 작업수행자의 부하를 고려하지 않고 단지, 워크플로우 엔진의 부하를 작업수행자로 분배시키려고 한데에서 기인한다. 이로 인해 발생하는 문제는 그림5에서 보는 바와같이, 작업수행자의 수를 늘려도 워크플로우 수행시간이 기대만큼 증가하지 않는다는 것이다. 따라서, 작업수행자의 증가에 따른 성능향상정도를 향상시키기 위해서는 작업수행자의 부하를 고려하여 작업이 할당되어야 한다.

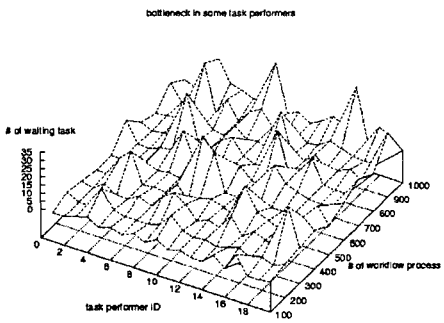


그림 4. 에이전트 위임모델에 의한 작업수행자의 병목현상

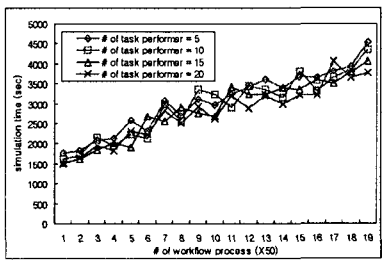


그림 5. 작업수행자 증가가 성능향상에 미치는 영향

분산시스템에서의 부하분산 방법[8]으로는 프로세스를 이동시키는 방법, 데이터를 복사하여 분산시키는 방법이 있다. 프로세스를 부하가 적은 노드로 이동시키기 위해서는 각 노드내의 자원을 사용하는 작업수, 이용률, 큐(Queue)길이 등의 정보가 필요하다. 프로세스를 이동시키는 방법은, 부하가 적은 노드가 부하가 많은 노드로부터 프로세스 이동을 요청(receiver-initiated strategy) 하거나, 부하가 많은 노드가 부하가 적은 노드로 프로세스를 이동(sender-initiated strategy) 시키는 동적부하분산(dynamic load distribution) 방법이 있으며, 한 서버에게 다른 노드로 일방적으로 프로세스를 할당해주는 정적부하분산(static load distribution) 방법이 있다. 데이터를 복사하여 분산시키는 방법은 자원에 대한 가용성(availability)을 높이는 반면 일관성 유지가 보장되어야 한다.

워크플로우 시스템 내의 작업수행자들은 상호연결이 존재하지 않기 때문에 서로간의 부하정보를 교환하기가 어렵다. 그리고, 워크플로우의 특성상 워크플로우 엔진과 작업수행자는 작업을 수행하고 수행한 작업에 대한 결과를 반환하는 특성상 작업수행자간의 동적부하분산 방법보다는 워크플로우 엔진과 작업수행자간의 작업할당방법인 정적부하분산이 쉽게 구현될 수 있다. 따라서, 우리는 정적부하분산 방법을 사용하며, 작업수행자의 부하를 관리하기 위해 그림2의 계층2에 응답리스트 관리자(Response List Manager)를 두어, 효율적으로 부하를 분산시킬 수 있도록 한다.

3.2 응답리스트 관리자(Response List Manager)

응답리스트 관리자란 그림2의 계층3에 존재하는 작업수행자들이 처리해야 할 작업의 수를 계층2에서 관리하기 위한 자료구조로 리스트의 길이는 작업을 수행 가능한 작업수행자들의 수와 같으며 작업수행자의 증감에 따라 동적으로 변경 가능해야 한다. 응답리스트 관리자는 계층2에 존재하는 모든 워크플로우 엔진에 존재하며, 워크플로우 엔진에서 처음으로 작업수행자에게 작업을 할당할 때, 해당 작업수행자에

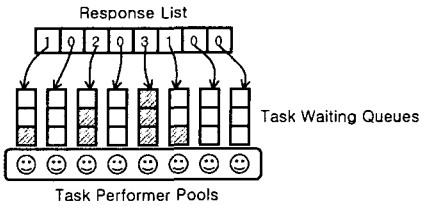


그림 6. 응답리스트

대한 응답리스트(그림 6)를 생성한다. 워크플로우 엔진에서 작업을 작업수행자에게 할당해야 할 경우 할당대상을 응답리스트 관리자에게 문의하며, 응답리스트 관리자는 응답리스트를 참조하여 숫자가 가장 적은 작업수행자에게 작업을 할당하며 응답리스트 내의 해당 필드의 숫자를 1만큼 증가시킨다. 작업 수행이 끝나면 해당 결과가 반환되어 돌아오기 때문에, 결과반환을 이용하여 해당 작업수행자와 관련된 응답리스트 내의 필드를 1만큼 감소시킨다. 응답리스트는 계층2에 존재하는 모든 워크플로우 엔진 내에 존재해야 하기 때문에 응답리스트 비일관성 (Response List Inconsistency) 문제가 발생할 수 있으며, 작업수행자가 많아질수록 응답리스트의 수가 무수히 많아져 응답리스트 과잉(Response List Overflow)문제를 발생시킬 수 있다.

3.3 응답리스트 비일관성 문제

각 워크플로우 엔진마다 갖고 있는 응답리스트 정보를 상호 교환하지 않는다면 작업을 할당하는 대상인 작업수행자를 서로 모르기 때문에 응답리스트의 값이 서로 달라질 수 밖에 없다. 이로 말미암아 응답리스트 비일관성 문제가 발생하며 이는 작업수행자의 부하를 잘못 판단하여 작업수행자 내의 병목현상을 발생시킬 가능성이 있다. 이를 해결하기 위한 방법으로는 워크플로우 내의 응답리스트 정보를 수시로 상호 교환하는 방법이 있지만, 이는 워크플로우 엔진의 수가 많아질수록 응답리스트 일관성 유지비용은 크게 증가할 수 밖에 없다. 따라서, 이를 대신하여, 작업수행자가 작업수행결과를 워크플로우 엔진에게 반환할 때, 자신의 작업 대기큐(Task Waiting Queue)에서 대기하고 있는 작업의 수를 함께 반환하여 워크플로우 엔진내의 응답리스트를 최신 정보로 갱신하게 된다.

3.4 응답리스트 증가문제

작업수행자 수가 증가할수록 워크플로우 엔진 내에서 관리해야 할 응답리스트의 수는 증가할 수 밖에 없다. 모든 작업수행자에 대한 응답리스트를 관리하는 것은 비효율적이다. 따라서, 모든 응답리스트를 관리하지 않고 응답리스트들의 상태를 관찰하여 상대적으로 많은 수행시간을 요구하는 작업수행자에 대한 응답리스트, 또는 응답리스트내의 필드값이 상대적으로 커지는 응답리스트(그림7의 List3, List6)만을 관리할 수 있다. 이를 결정하기 위한 리스트내의 필드에 대한 경계값은 시스템 관리자에 의해 결정되어야 한다.

4. 성능 평가

본 장에서는 응답리스트를 적용했을 경우 위임모델 기반 워크플로우 시스템의 성능 및 확장성에 어떠한 영향을 미치는지를 실험을 통해 보인다. 실험은 Stochastic Petri-nets simulation 도구인 UltraSAN[9]을 이용하여 수행되었다.

그림8은 응답리스트 관리자를 도입함으로써 작업수행자들의 부하가 고르게 분산됨을 보여주고 있다. 그림4에서는 부하가 특정 작업수행자에게 집중되었지만, 응답리스트 관리자를 도입함으로써 특정 작업수행자에서의 병목현상을 제거하

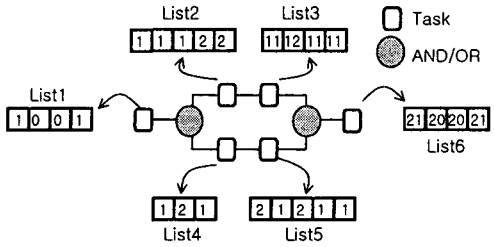


그림 7. 응답리스트 관리대상

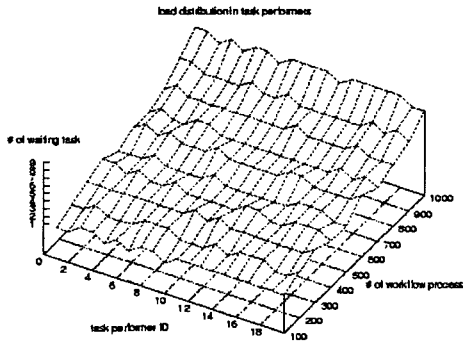


그림 8. 작업수행자의 부하분산

였다. 이러한 병목현상 제거는 그림9와 같이 작업수행자를 증가시켰을 때 워크플로우 시스템의 성능 및 확장성 향상효과가 뚜렷해지는 결과를 가져왔다.

5. 결론

워크플로우 시스템의 성능 및 확장성 향상을 위해 제안된 에이전트 위임모델은 워크플로우 엔진의 부하를 작업수행자들에 자연스럽게 분산시킴으로써 워크플로우 엔진의 병목현상을 제거, 결과적으로 워크플로우 시스템의 성능 및 확장성 향상을 가져왔다. 그러나, 에이전트 기반 워크플로우 시스템은 작업수행자의 부하를 고려하지 않고 작업을 할당하기 때문에 작업수행자에 병목현상을 발생시켰으며, 작업수행자의 수를 증가시켜도 성능 및 확장성 측면에서 보다 향상된 결과를 얻을 수 없었다. 본 논문에서는 이러한 에이전트 위임모델의 문제해결을 위해 응답리스트를 도입하였으며, 그 결과 작업수행자 수의 증가에 따라 워크플로우 시스템의 보다 나은 성능 및 확장성 향상을 얻을 수 있었다.

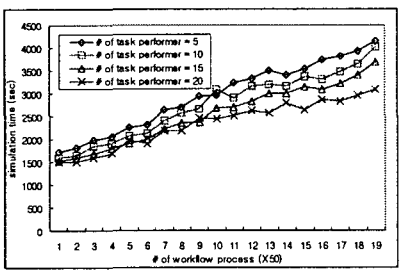


그림 9. 작업수행자 병목제거에 따른
워크플로우 시스템의 성능

감사의 글

이 연구는 일부 한국과학재단(KOSEF)의 특정기초연구(98-0102-11-01-3)에 의해 지원되었습니다.

참고 문헌

[1] WfMC, Workflow Management Coalition Terminology and Glossary- WfMC Specification, 1999.
[2] Heinl P. and Schuster H, Towards a highly scalable architecture for workflow management systems, In Proc. of International Workshop on Database and Expert Systems Applications, pp. 439 - 444, 1996.
[3] Jeong-Joon Yoo, Doheon Lee, Young-Ho Suh and Dong-Ik Lee, Scalable Workflow System Model Based on Mobile Agents, Lecture Note in Artificial Intelligence Vol. 2132, Springer Publisher, pp. 222-236, 2001.
[4] Kwang-Hoon Kim and Dong-Soo Han, Performance and scalability analysis on client-server workflow architecture, In Proc. of International Conference on Parallel and Distributed Systems, pp. 179-186, 2001.
[5] Ting Cai, Peter A. Gloor, and Saurab Nog, DartFlow: A Workflow Management System on the Web using Transportable Agents, Technical report, Dartmouth College, 1997.
[6] Stan Franklin and Art Graesser, Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents, In Proceedings of 3rd International Workshop on Agent Theories, Architectures, and Languages, Jan., 1997.
[7] Jeong-Joon Yoo, Young-Ho Suh, Sang-Bum Song, and Dong-Ik Lee, Agent-Based Workflow System Architecture and Mobile Agent Requirements, In Proc. of KIPS Fall Conference, 1999.
[8] A. Hac, T. Johnson, A Study of Dynamic Load Balancing in a Distributed System, Proc. ACM SIGCOMM Symposium on Communications, Architectures and Protocols, Stowe, Vermont, 1986.
[9] D. D. Deavours, W. D. Obal II, M. A. Qureshi, W. H. Sanders, and A. P. A. van Moorsel., UltraSAN Version 3 Overview, In Proc. of International Workshop on Petri Nets and Performance Models, 1995.