인터넷 응용을 위한 고장 감내 이동 에이전트 프레임워크와 레플리케이션 연구

박 경 모†

요 약

본 논문에서는 인터넷 환경에서 분산 이동 에이전트의 신뢰성에 관련된 이슈를 논한다. 인터넷 응용을 위해 이동 에이전트들에 고장-감내하도록 만든 구조 프레임워크를 제안한다. 에이전트 데이터를 복제하는 레플리케이션 기법은 인터넷을 통한 분산 시스템에서 고장 감내를 획득하기 위해 매우 중요하다. 본 연구의 초점은 제안된 고장-감내 이동 에이전트 프레임워크에 대한 레플리케이션 컴포넌트 부분이다. 레플리케이션 차수, 능동형과 수동형 레플리케이션 전략, 레플리케이션 규모에 따른 효과에 대해 시뮬레이션 연구를 통해 얻은 성능 분석 결과를 설명한다.

A Fault-Tolerant Mobile Agent Framework and Replication Study for Internet Applications

Kyeongmo Park†

ABSTRACT

This paper addresses the issue involved in dependability of distributed mobile agents in the Internet environment. We propose an architectural framework for the Internet applications making mobile agents into fault-tolerant. The replication of agents and data is of great importance to achieve fault tolerance in distributed systems over the Internet. This research focuses on the replication component for the proposed fault-tolerant mobile agent framework. We present and analyze the performance results obtained when doing simulation study on the effects of the degree of replication, the active and passive replication strategies, and the replication scale.

키워드 : 인터넷 응용(Internet Application), 고장 감내(Fault Tolerance), 레플리케이션(Replication)

1. 서 론

오늘날 정보통신산업에서 가장 활발한 영역 중의 하나는 모바일 컴퓨팅(mobile computing) 분야로 급성장하고 있다. 자동차, 휴대폰을 비롯하여 데스크탑(desktop), 랩탑(laptop), 팜탑(palmtop)에 이르기까지 여러 형태의 컴퓨터에서 사용자의 작업을 처리하기 위해 인터넷 서비스를 필요로 한다. 이러한 모바일 장치들은 일반적으로 신뢰할 수 없는 낮은 대역폭과 높은 지연대기시간을 갖는 무선 네트워크 접속을 한다. 근래 인터넷 서비스의 확장과 인터넷 정보의 급속한 증가로 인하여 많은 사람들은 언제 어디에서든 상당한 양의 정보에 액세스할 수 있다. 정부기관 사무실, 학교 도서관, 공항, 호텔과 같은 공공장소에서 인터넷 터미널을 볼 수가 있다. 어떤 터미널에서든 사용자는 웹 메일 서비스를 이용하여 자신의 e-메일을 체크할 수 있다. 궁극적으로 모바일 사용자들은 어떤 컴퓨터에서든 자신들의 파일과 애플리케이션에 액세스할 수 있을 것이다. 인터넷을 통한 분산 컴퓨팅에서 이동 에이전트(mobile agent)는 필수적인 도구이다. 거의 모든 주요 인터넷 사이트들에서 이동 에이전트를 사용하고 있다[1, 3, 4, 7, 8].

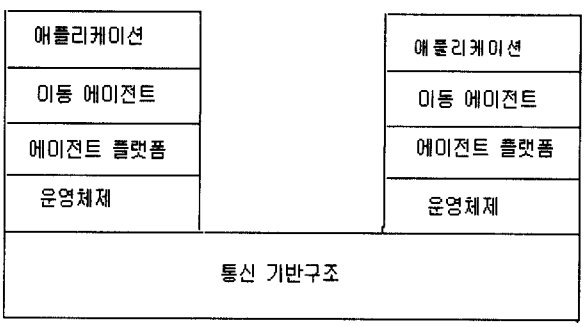
이동 에이전트는 이질적인(heterogeneous) 네트워크상의 호스트(host)로부터 호스트로 자신들이 선택한 시간과 장소에 따라 이동할 수 있는 자치적(autonomous) 성질의 컴퓨터 프로그램이다. 호스트들 사이에서 실행되는 프로그램의 상태는 저장되고 새로운 호스트로 이전된다. 실행 중에 다른 곳으로 갔다가 다시 돌아와서 실행 상 떠나기 전 프로그램 상태를 복원한다.

이동 에이전트는 전통적인 운영체제 프로세스와 는 다르다. 프로세스와 달리 이동 에이전트는 분산 시스템의 어느 시점에서 실행되고 있는 위치를 알고 있다. 이동 에이전트는 통신 네트워크를 알고 있으며, 실행하는 동안 한 노드에서 다른 노드로 독립적이고 비동기적으로 이동하기

※ 본 연구는 2003년도 가톨릭대학교 교비연구비의 지원으로 이루어졌음.
† 정 회 원 : 가톨릭대학교 컴퓨터정보공학부 교수
논문접수 : 2003년 7월 21일, 심사완료 : 2003년 12월 12일

위해 관련 정보를 가지고 필요한 결정을 한다. 이동 에이전트 시스템은 프로세스 이전 시스템과 다르다. 에이전트의 이동에 있어 *jump* 혹은 *go* 문장을 선택하여 움직이는 반면에 프로세스 이전 시스템에서는 시스템이 작업부하의 균형을 맞추기 위해 수행 프로세스를 언제 어디로 이동시킬 것인지를 결정한다. Milojicic[6]에 의하면 에이전트 이전이 전통적인 프로세스 이전보다 더 빠르다고 보고하고 있다.

이동 에이전트 시스템을 구축하기 위한 많은 시스템들은 (그림 1)에서 보는 바와 같이 공통적으로 계층적 기반 구조를 갖고 있다. 애플리케이션은 에이전트 플랫폼 상에서 실행되는 이동 에이전트를 만든다. 이동 에이전트의 가능한 활동을 지원하는 에이전트 플랫폼은 기본 아키텍처의 운영체제와 네트워크에 의해 제공되는 자원과 통신 채널을 사용한다.



(그림 1) 일반적 이동 에이전트 기반구조

인터넷을 통한 분산 컴퓨팅을 하는데 이동 에이전트는 매우 흥미 있는 패러다임이다. 최근 많은 인터넷 응용에서 이동 에이전트의 사용은 급속도로 증가하고 있다[1, 3, 4, 7].

이동 에이전트는 네트워크 연결 불능에 대한 취약성을 감소시키고 클라이언트-서버 애플리케이션의 지연시간과 대역폭에서의 향상을 포함하는 여러 장점들이 있다[4]. 이동 에이전트는 클라이언트와 서버 사이에 데이터를 전송하는 것이 아니라 클라이언트로부터 서버로 애플리케이션 코드를 가지고 간다. 애플리케이션 코드의 크기는 클라이언트와 서버 사이에 상호 교환되는 데이터의 양보다 대개는 더 적기 때문에, 이동 에이전트 시스템은 클라이언트-서버 컴퓨팅을 통해 상당한 성능 개선을 가져온다. 따라서 이동 에이전트를 사용하는 많은 인터넷 응용이 급속히 증가하고 있다[1, 3, 13].

그런데, 인터넷을 통한 컴퓨팅은 신뢰성이 없다. 인터넷을 통해 연결된 노드들은 끊임없이 고장이 나고 복구된다. 과중한 통신 부하, 링크 실패, 소프트웨어 결함 등으로 인해 인터넷에서의 일시적인 통신 장애는 흔히 일어나는 것이다. 인터넷 상에 전송되는 정보는 안전하지 못하고 에이전트의 보안을 보장받지 못한다. 그러므로 신뢰성(dependability) 문제는 인터넷 응용의 중요한 연구주제가 되고 있다[7, 15].

고장 감내(fault tolerance) 기능이란 네트워크이나 호스트 노드에 고장이 나더라도 분산 소프트웨어 시스템이 중단되지 않고 계속 운영 될 수 있도록 보장하는 것을 말한다. 오류들은 발견되어 복구되어야만 한다. 따라서 이동 에이전트를 고장 감내 하도록 만드는 것이 중요하다.

본 연구에서는 인터넷 환경에서 분산 이동 에이전트의 신뢰성에 관련된 이슈를 논한다. 인터넷 응용을 위해 이동 에이전트들에 레플리케이션을 통해 고장-감내하도록 만든 프레임워크를 제안한다. 제안된 고장 감내 이동 에이전트 프레임워크와 이와 관련된 레플리케이션의 관리 및 성능 연구의 초점을 맞춘다. 데이터를 복제하는 레플리케이션(replication) 기법은 분산 시스템에서 고장 감내를 획득하기 위해 폭넓게 이용된다. 제안 프레임워크에 대한 레플리케이션 관리에 있어 레플리케이션 차수 및 능동적 레플리케이션, 수동적 레플리케이션 기법에 대한 효과를 모의실험을 통해 얻은 성능 결과를 비교한다.

본 논문의 구성은 다음과 같다. 2절에서는 이동 에이전트 시스템을 위한 고장 감내 아키텍처에 대한 기본 개념을 설명하고 에이전트 프로세스들에 대한 레플리케이션 결합을 이용한 고장 감내 방안을 기술한다. 3절에서는 레플리케이션 차수와 2가지 레플리케이션 기법에 대한 효과를 모의실험을 통해 얻은 성능 결과를 기술한다. 마지막으로 4절에서 결론과 추가 연구 방향을 제시한다.

2. 고장 감내 이동 에이전트 프레임워크

서론에서 설명하였듯이 이동 에이전트 모델을 인터넷에 응용하면 분산 컴퓨팅에서 전통적으로 사용하던 클라이언트-서버 모델의 제약점을 극복할 수 있다. 인터넷 응용 관련하여 이동 에이전트 시스템의 설계, 구현, 배치에 관한 중요한 연구 쟁점들이 있다. 주요 이슈로는 에이전트 고장 감내, 에이전트 보안, 에이전트간의 통신과 동기화 등이 포함된다. 본 연구에서는 인터넷 환경에서의 신뢰성 있는 이동 에이전트 시스템을 구축하기 위한 에이전트 레플리케이션을 통한 고장 감내 프레임워크와 어떻게 레플리케이션을 효율적으로 사용할 수 있는지에 대해 집중한다.

2.1 기본 개념

이동 에이전트에 대한 고장 감내 해결방안 관련하여 에이전트 시스템의 특성과 인터넷 응용에 대한 이해가 우선되어야 한다. 제시하는 고장 감내 이동 에이전트 프레임워크 구축 목적 근거로 요구사항들을 요약하면 다음과 같다.

- **선택적 고장감내(fault tolerance : FT)** : FT의 사용 여부에 따라 사용자, 에이전트 환경 또는 애플리케이션

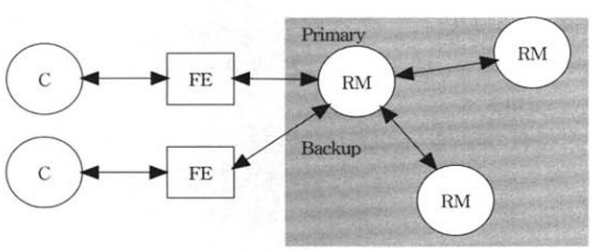
자체를 개별적으로 결정할 수 있어야 한다. 그래서 FT를 사용 하거나 사용하지 않는 애플리케이션을 병행적으로 실행이 가능해야 한다.

- **자치성(autonomy)** : 이동 에이전트가 네트워크 내에 투입되면 사용자는 에이전트의 실행에 대해 크게 제어하지 않는다. 만약 FT를 위한 활동을 감독 프로그램 요구에 따라 지시 받으면 자치성이 제한된다. 자치적인 에이전트에 의해 내려지는 모든 결정들을 감독관리자에 의해 조정할 필요가 있다.
- **사용자 투명성(transparency)** : 애플리케이션의 사용자들은 자기들의 문제를 해결하고 알고리즘을 코딩하는데 관심을 둔다. 일반 사용자 애플리케이션에 FT에 대해서는 관심을 두지 않는다. 따라서 고장 감내 실행을 하려면 애플리케이션 코드를 변경할 필요가 없어야 한다.
- **모듈식 구조(modular architecture)** : FT 실행을 하는 경우에 필요한 메모리와 계산하는데 걸리는 시간에 대한 요구를 충족시키기 위해 이동 에이전트들의 모듈식 구성이 필요하다. 여러 기능들을 실행시간에 교환될 수 있는 특정한 여러 모듈로 포함할 수 있어야 한다.
- **애플리케이션-에이전트커널 분리** : 사용자 투명성을 촉진하고 한 에이전트가 다른 모듈을 실행할 수 있도록 하기 위해 애플리케이션과 에이전트 커널 간에 분리시킨다. 애플리케이션은 에이전트 행동에 영향을 줄 수 있어야 한다.
- **확장성(scalability)** : 대규모의 분산 애플리케이션에서 에이전트의 수는 시간에 따라 상당한 변화를 줄 수 있다. 분산 멀티에이전트 시스템에서는 에이전트와 사용가능한 자원을 현저한 성능의 저하나 복잡성에 있어서 상당한 증가 없이 증감 조정할 수 있어야 한다.
- **비동기 상호작용(asynchronous interaction)** : 전통적인 클라이언트-서버 아키텍처에서는 두 편(클라이언트-서버)이 동기적인 상호작용으로 관계되고 각 편의 고장을 탐지하는 것이 아주 쉽다. 이에 비해 이동 에이전트는 비동기적인 방식으로 작용하므로 오류 탐지를 위한 추가 작업이 포함된다.
- **트랜잭션(transaction) 지원** : 이동 에이전트가 실행하는 동안 에이전트의 내부 상태와 호스트의 상태를 변경할 수 있다. 어떤 고장이 났을 경우 변경된 내용을 효과적인 방법으로 일관성 있게 보존해야 한다.
- **단절 컴퓨팅(disconnected computing)** : 통신 채널이 일시적인 고장에 의해 손상되더라도 이동 에이전트는 연결이 복구되는 동안 기다리면서 여전히 자신들의 작업을 수행할 수 있어야 한다.
- **자원 관리** : 이동 에이전트는 이따끔 현재 호스트의 고장으로 인해 한 호스트에서 다른 호스트로 이전할 필요가 있다. 이전(migration)작업은 다른 호스트가 제공해야 하는 자원을 고려하여 이동시켜야만 한다. 에이전트가 다른 호스트로 이동할 때, 어떤 관련 상태 정보를 에이전트와 함께 실려 보낼지 확인해야 한다.
- **적응성 (adaptivity)** : 실행시간 동안에 에이전트의 작동에 영향을 줄 수 있어야 한다. 만일 이것이 가능하지 않다면 사용자의 요구에 의해 FT 기능을 추가하는 것은 불가능하다.

2.2 분산처리 환경에서의 레플리케이션

분산 시스템의 데이터는 객체들로 이루어진다. 하나의 객체(object)는 파일이나 자바 객체일 수 잇다. 각각의 논리적 객체는 복제(replica)라고 불리는 수집된 물리적인 복사본에 의해 구현된다. 복제된 것은 시스템 연산의 일관성과 연결된 데이터와 움직임이 각 컴퓨터에 저장된 물리적 객체이다. 객체를 복제한 것은 특정한 시점에서 반드시 꼭 같지는 않다. 레플리케이션(replication)을 통해 여러 컴퓨터에 데이터를 복제하는 것은 분산 시스템에서 고장 감내, 고가용성, 성능 향상을 제공하기 위해 중요하다. 레플리케이션 기법은 분산 시스템에서 광범위하게 사용된다. 예를 들어, 브라우저로 웹 서버들과 웹 프락시 서버들에 있는 자원을 캐싱(caching)하는 것은 레플리케이션의 한 형태이다. 서버와 캐시에 있는 데이터는 서로 복사본이다. 레플리케이션은 인터넷 서버스를 향상시키기 위한 기법으로 신뢰성 관련해 고장감내하거나 서비스의 고가용성을 증가시키고 성능을 개선을 위해 최근 많이 연구되고 있다[11, 17].

고장 감내를 위한 수동형(passive) 레플리케이션 모델(그림 2)에서, 제1의(primary) 복제 관리자와 하나 또는 둘 이상의 제2의(backup) 복제 관리자들이 특정한 시간에 존재한다. 서비스를 받기위해서는 프론트엔드(front-end)는 오직 제1의 복제 관리자와 통신한다. 제1의 복제 관리자는 연산을 수행하여 업데이트한 데이터를 복사본을 백업 복제 관리자들에게 전송한다. 만일 제1 관리자가 고장나면 백업 관리자중의 하나가 제1 관리자 역할을 위해 승격된다.



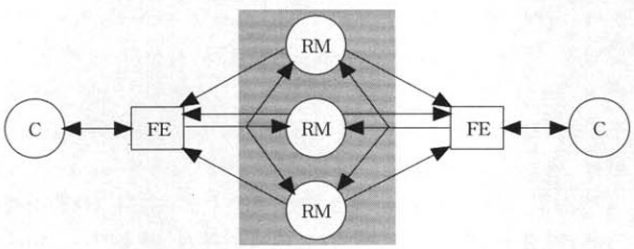
C : Client FE : Front-End RM : Replica Manager

(그림 2) : 수동형 레플리케이션 모델

능동형(active) 레플리케이션 모델 (그림 3)에서는 복제

관리자들은 동등한 역할을 다하는 상태 프로세스들이며 그룹으로 구성된다. 프론트엔드는 복제 관리자들의 그룹에 요청을 멀티캐스트하고 모든 복제 관리자들 받은 요청을 동일하지만 독립적으로 처리하여 응답한다. 여기에서는 어떤 복제 관리자가 고장이 나도 서비스의 성능에 영향을 주지 않는다. 그 이유는 나머지 복제 관리자들이 정상적인 방법으로 응답을 계속할 수 있기 때문이다.

능동형 레플리케이션에서는 요청 받은 모든 복제(복제 관리자)를 병행적으로 처리한다. 능동형 전략을 사용하면 높은 오버헤드가 따른다. 레플리케이션 차수(degree)가 n이라면 n개 복제된 것이 있다는 뜻이고 하나의 결과를 만들기 위해 n번의 처리가 필요하다.



C : Client FE : Front-End RM : Replica Manager

(그림 3) 능동형 레플리케이션 모델

수동형 레플리케이션에서는 모든 요청들은 하나의 복제(master 복제 관리자)에 의해 처리되며 이 복제는 일관성 유지를 위해 주기적으로 자기의 현재 상태를 다른 복제들(slave 복제 관리자)에게 전송한다. 수동형 전략에서는 고장 난 경우에만 중복 복제를 활성화함으로써 프로세서 사용율을 향상시킬 수 있다. 현재 활동중인 복제에 오류가 있으면 새로운 복제를 미활동적인 복제 세트 중에서 선택하고 마지막으로 저장된 상태에서 재실행을 시작 한다. 수동형 기법은 능동형 기법 보다 CPU 자원을 덜 필요로 한다. 하지만 처리 시간과 기억공간에 있어 비용 부담이 많은 체크포인트 관리가 요구된다. 능동형 레플리케이션 기법은 빠른 복구하는데 지연시간이 비교적 짧다. 따라서 실시간(real-time) 제약조건을 갖는 애플리케이션에 주로 사용된다. 반면에 수동형 방법은 고장 없는 실행 중에는 오버헤드가 낮은 장점을 갖지만 복구지연시간이 길다는 단점이 있다.

적합한 기법을 선택하기 위해서는 시스템 환경, 고장률과 복구지연시간, 시스템 오버헤드를 참작하여 애플리케이션에서 필요한 요구사항들을 종합적으로 고려하여 결정한다. 고장률이 아주 높거나 애플리케이션 설계에 엄격한 실시간 제약사항이 있으면 능동형 레플리케이션 기법을 선택해야 한다. 다른 대부분의 경우에는 수동형 기법이 바람직하다.

〈표 1〉 능동형, 수동형 레플리케이션의 비교

	능동형 기법	수동형 기법
장점	• 복구지연시간 짧다 • 실시간 응용 사용	• 프로세서 사용율 좋다 • 부담이 비교적 적다
단점	• 부담이 크다 • CPU 자원 증가	• 복구지연시간이 길다 • 체크포인트 관리에 따른 처리 시간 및 공간 비용

2.3 고장 감내 에이전트 어프로치

기본적으로 에이전트 애플리케이션은 에이전트 사이의 협력에 의존한다. 관련된 에이전트중의 하나가 기능에 있어 고장이 생기면 전체 계산 작업이 종료되는 문제가 발생할 수 있다. 이 경우 애플리케이션에 중요한 특정 에이전트를 복제하는 레플리케이션 기법을 이용하여 문제를 해결할 수 있다.

그러나 경우에 따라서 레플리케이션 작업은 처리 및 통신에 부담을 주고 애플리케이션 진행하는 시점에서 매우 중요한 상태를 놓칠 수 있기 때문에 유의하여 사용해야 한다. 본질상 에이전트 시스템은 유연하고 역동적이므로 상황에 따른 급속한 변경이 가능한 고장 감내 메커니즘을 구성하는데 강점이 된다. 또한 에이전트 플랫폼으로부터 독립적인 솔루션을 유지해야 플랫폼이 급격한 변경에도 불구하고 유효하게 사용될 수 있으며 결과적으로 에이전트 시스템간의 상호운영성(interoperability)을 제공한다.

RMI(remote method invocation) 기반의 자바로 작성되는 프로토타입 시스템은 JVM을 이용한 아키텍처 독립적인 특징을 갖는다. 우선 고려된 시스템 모델은 동기적인 것으로 비동기적 모델은 추후에 확장될 수 있다고 가설을 두었다. 이것은 고장 탐지에 따른 통신 지연에 대한 상한값을 취하는 것을 의미한다. 본 논문에서는 신뢰성 있는 인터넷 응용을 위한 프레임워크를 제안한다.

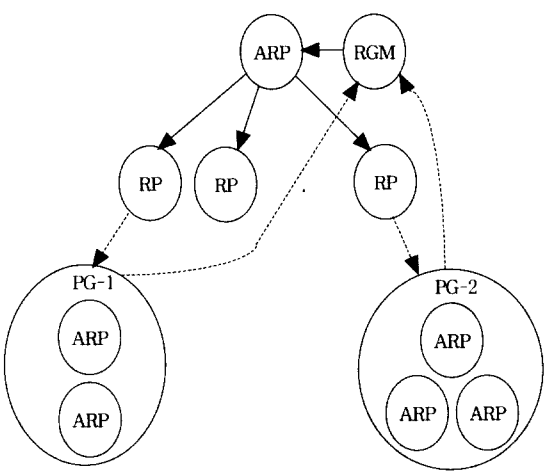
2.4 고장 감내 이동 에이전트 아키텍처

본 절에서는 앞서 설명한 이동 에이전트 시스템에 레플리케이션을 이용한 고장 감내 어프로치로써 FTMAR(Fault-Tolerant Mobile Agent Replication)프레임워크에 대하여 기술한다. 각각의 프로세스는 다양한 레플리케이션 전략(2.5절에 상세히 기술된다)을 사용하여 무제한적으로 복제될 수 있다. 시스템 상태에 따라 복제하는 방법을 변화시킬 수 있다. 복제한 것을 증감할 수 있는 프로세스 그룹 회원 관리 기능을 포함한다.

(그림 3)에 보듯이 레플리케이션 그룹내에서 순서를 갖는 멀티캐스트 기능이 있다. 레플리케이션 그룹은 모든 응용 프로세스에 내재하는 불투명한 것이다. 복제한 수자와 특정한 프로세스에 사용되는 내부적 기법에 관한 사항은 다른 응용 프로세스들에 대해 감추어진다. 레플리케이션 그룹마다 한 개의 주축 마스터 프로세스가 있어 다른 프

로세스들과 통신 연락한다. 마스터는 전체적으로 순서 있는 멀티캐스트 및 고장탐색 기능을 갖는 시퀀서(sequencer) 역할을 한다. 마스터가 고장이 생긴 경우 나머지 복제된 것 중에서 새로운 마스터를 선출한다.

FTMAR 구조에서는 고장 감내 능력으로 각 에이전트는 에이전트 레플리케이션 프로세스(ARP) 객체의 여러 기능을 계승하고 기반 시스템은 에이전트 실행과 통신 관리할 수 있다. 그리하여 이 프레임워크 (그림 4)를 이용하면 에이전트에 대한 중재자 역할이 가능하다. 또한 에이전트가 시동, 정지, 중단 및 재개되는 시점과 메시지 데이터 수령 시점을 결정한다. 각 프로세스는 자신의 프로세스 쉘과 관련되어 있고 레플리케이션 그룹 관리자(RGM)의 역할을 한다. 받은 메시지를 레플리케이션 그룹의 모든 멤버들에게 배달한다. RGM이 입력 메시지를 인터셉트 하고 캐싱을 할 수 있다. 따라서 모든 메시지는 레플리케이션 그룹 내의 동일한 순서대로 처리되어 진다. 에이전트를 복제할 때 에이전트의 레플리케이션 그룹이 중단되고 실행이 재개되기 전에 해당되는 ARP가 새로 만든 RGM 로 복사된다. ARP 프로세스는 단독 에이전트이든 레플리케이션 그룹이든지 간에 무시하고 원격 프로세스(RP)와 통신할 수 있다. 통신에는 RP 인터페이스에 의한 지역적 프락시를 이용된다.



ARP : Agent Replication Process
RGM : Replication Group Manager
RP : Remote Process,
PG : Process Group

(그림 4) 고장 감내를 위한 FTMAR 구조

2.5 레플리케이션 관리

2.3과 2.4절에서 설명대로 본 연구에서는 에이전트 프로세스들에 대한 레플리케이션을 결합하여 고장 감내를 이루고자 한다. 리플리케이션 사용 관련 구체적인 전략을 기술한다. FTMAR 구조에서는 능동형과 수동형 레플리케이션 기법이 모두 다 가능하다. 사용하는 기법이 하나로 고정되어 있지 않고 유연하다. 각 에이전트는 실행시간에 레플리케이션 프로토콜을 변경할 수 있다., 수동형 레플리케이션 경우에 복제 수자, 백업 시간 주기를 조정할 수 있다. 각 에이전트에 대한 레플리케이션은 여타 애플리케이션 프로세스에 대해 모두 투명하다는 특성이 갖는다. 레플리케이션 관리자는 각 에이전트와 결합되어 있고 다음과 같은 4가지 기능을 제공한다.

- **레플리케이션 그룹 정보관리** : 레플리케이션 그룹에 포함된 모든 복제들과 현재 사용 전략에 대한 정보를 끊임없이 변경 유지한다.
- **레플리케이션 중단과 재개** : 레플리케이션 그룹을 일시적으로 중단시키거나 재시동하는 일을 맡는다. 일시적 중단은 그룹 멤버의 수자나 사용하는 내부 전략이 바뀔 때 필요로 한다. 이 기능은 현재 사용 전략에 달려있다. 능동형 기법에서는 그룹 중단은 각 복제된 것에게 중단 명령을 전송을 의미한다. 수동형에서는 오직 마스터가 중지된다.
- **메시지 데이터 확산** : 메시지 데이터 확산 기능을 통해 레플리케이션 그룹 내의 통신을 확산시킨다. 이 기능은 현재 사용 전략에 따라서 다양하다. 예를 들어, 능동형에서는 메시지 데이터가 복제들에게 브로드캐스트 하여 제공되지만 수동형에서는 레플리케이션 그룹의 마스터에 의해서만 메시지가 처리된다. 능동형, 수동형 기법에 따라 통신하는 대상이 달라진다.
- **레플리케이션 변경** : 레플리케이션 관리자는 현재 사용하는 레플리케이션 정책의 변경하고 레플리케이션 파라미터의 조정한다. 능동형에서 수동형으로 변경될 때, 필요한 연산은 단지 복제된 것들에게 전략의 변경을 알려주는 것이다. 반면에, 수동형에서 능동형으로 변경할 때에는 전체 그룹들을 일시 정지시키고 실행을 재개하기 전에 복제된 모든 것들을 새로 갱신한다.

고장복구의 처리를 위해서 레플리케이션 관리자를 이용한다. 레플리케이션 그룹의 능동형 복제에 고장 이 나면 그룹 내의 다른 복제중의 하나를 새로운 능동형 복제로 선출한다.

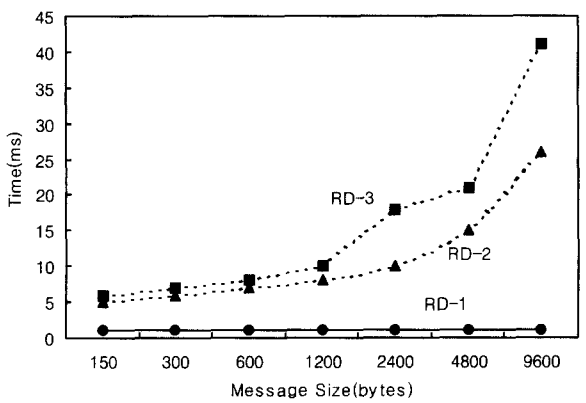
3. 성능 연구

2장에서 기술한 FTMAR 구조 프레임워크의 중요 요소인 레플리케이션 성능 관련 파라미터인 레플리케이션 차수와 능동형과 수동형 레플리케이션 전략에 대한 효과를 분석한다. 성능 측정 결과는 Sun Ultra Sparc II, JDK 1.1.6 시스템에서 모의실험을 통해 얻었다.

3.1 능동형과 수동형 레플리케이션 성능 비교

본 실험에서는 능동형 레플리케이션 기법을 사용하여

하나의 레플리케이션 그룹에 동기형 메시지 한 개를 보내는데 드는 비용을 평가했다. (그림 5)는 서로 다른 레플리케이션 차수(replication degree : RD)를 갖는 3가지 시스템 구성(RD-1, RD-2, RD-3)에 대해 차별적인 성능결과를 나타낸 것이다. (그림 5)에서 보여주는 시간은 하나의 메시지를 보내서 응답을 받을 때까지의 시간 간격을 측정한 결과이다. 가로좌표 x축은 메시지의 사이즈를 표시한다. 각각의 메시지를 복제들에게 보낸다. 실험에 사용되는 시스템 형상 구성에 있어, RD-1에서는 프로세스는 로컬(local)로 지역적으로 두고 복제되지 않는다. RD-2에서는 1개의 원격 호스트에서 프로세스를 복제한다. RD-3에서는 3개의 복제가 있는데, 메시지를 보내는 호스트에 있는 1개의 마스터 복제와 2개의 원격 호스트들에 있는 2개의 백업 복제들을 말한다.



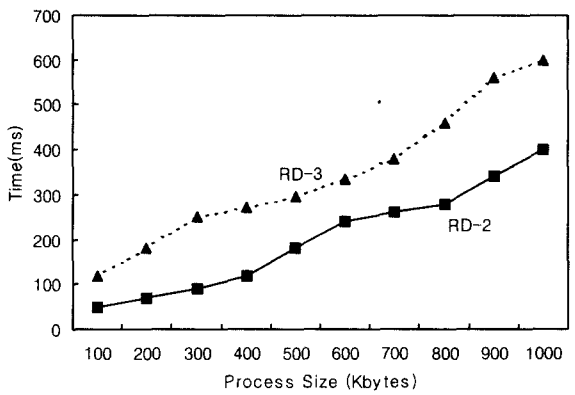
(그림 5) 능동형 레플리케이션에서 3가지 RD에 따른 통신비용 비교

RD-1 로컬에서는 메시지 사이즈에 관련 없이 아주 적은 시간으로 일양 분포를 한다. RD-2, RD-3 곡선 경우 1200 사이즈까지 큰 차이를 보이지 않지만 9600인 경우 시간간격이 커져서 호스트와 원격 호스트간의 통신 소요시간 및 복제 비용 증가로 지수분포로 RD-3가 RD-2 보다 더 급격히 상승한다.

수동형 레플리케이션 비용을 추정하기 위해 원격 복제들을 갱신하는 시간을 측정한다. 실험을 통해 얻은 응답시간이 너무 적어 의미가 없기 때문에 로컬 복제을 갱신하는 것은 무시하였다. (그림 6)에서는 다른 레플리케이션 차수(RD-2, RD-3 비교)를 사용해 하나의 레플리케이션 그룹을 갱신하는데 드는 비용을 측정한 성능 결과를 보여준다. RD-3의 갱신비용은 (그림 5)의 결과가 보여주듯이 호스트와 원격 호스트간의 통신 및 복제 비용 증가에 따른다. 레플리케이션 전략 변경에 따른 시간은 RD-2 경우가 능동형에서 수동형으로 바꾸는데 3.8ms가 걸렸고 RD-3 경우 같은 연산을 하는데 3.85ms로 둘 사이의 차이는 통신에 따른 비용보다 상대적으로 매우 낮다.

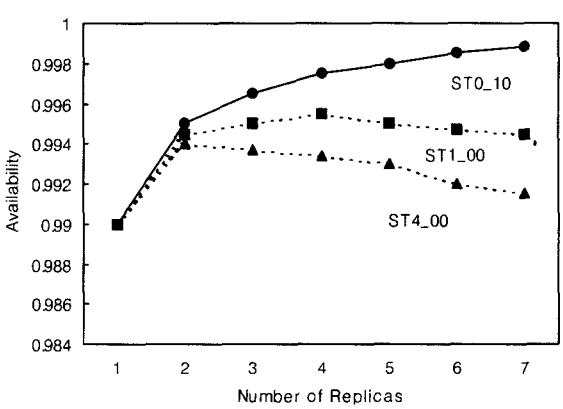
3.2 레플리케이션 규모에 따른 효과

이 실험에서 클라이언트 가용성이란 네트워크 서비스에 의해 수용되는 최종 사용자 액세스의 백분율로써 정의되며, 클라이언트 가용성에 대해 복제 수에 따른 효과를 떼놓고 생각한다. 본 연구의 목적은 네트워크 가용성을 최대화하고 서비스를 중앙 집중화하여 일관성 유지 부담을 최소화하기 위해 복제 규모를 어떻게 유지해야 하는지를 살펴보고자 한다.



(그림 6) 수동형 레플리케이션에서 RD에 따른 갱신 비용 비교

이 연구와 관련하여 복제 수가 변화되기 때문에 네트워크 가용성을 알아야 한다. 서비스가 미치는 범위인 네트워크 가용성은 하나의 복제에 도달할 수 있는 액세스의 백분율로 나타낸다. 서비스 가능범위를 측정할 수 없으므로, 어떤 복제에 도달할 수 없는 클라이언트 집단의 백분율을 $1\%/n$ (n : 복제 수)이 되도록 임의로 선정한다. 본 실험에 사용된 고장부하(faultload) 세트 : ST0_,10, ST1_00, ST4_00에 대한 평균고장률은 각각 0.11%, 1.05%, 4.12%이다.



(그림 7) 레플리케이션 규모에 따른 클라이언트 가용성 효과

(그림 7)은 서비스 가용성 관련하여 레플리케이션 차수의 변경에 따른 효과를 보여준다. 각각의 그래프는 서로 다른 고장부하와 서로 다른 계산 에러를 나타낸다. 1%/n 성장률과 높은 네트워크 고장률(ST4_00 그래프)에서는 최

적의 복제 수는 2에서 최고이다. 더 낮은 고장률(ST1_00)과 계산 에러가 없는 경우에 가장 좋은 가용성은 4개 복제를 가질 때가 나온다. 낮은 고장률을 갖는 모델(ST0_10)을 채택하면 가용성은 8개 복제될 때까지 계속 상승한다. 복제 수에 대한 함수에 관한 실험 결과는 추가적인 복제들을 사용하는 것이 서비스 가용성을 반드시 개선시키지 않고 감퇴시킬 수도 있다는 것을 정량적으로 보여준다. 그러므로 시스템 설계에 있어 주어진 고장부하와 일관성 수준에서 일관성 유지비용과 대비하여 추가적인 복제에 따른 최저의 가용성 이점이 얼마나 있는지를 면밀하게 평가해야 한다.

4. 결론 및 추가 연구

본 연구에서는 분산 환경에서 신뢰성 있는 인터넷 응용을 위한 고장 감내 에이전트에 대해 살펴보았다. 이동 에이전트 시스템을 위한 고장 감내 구조 프레임워크를 제안했다. 분산 멀티 에이전트 시스템의 확장성과 가용성에 유의하면서 고장 감내 이동 에이전트 레플리케이션(FTMAR) 구조와 설계에 관해 기술하였다. 특별히 제안 프레임워크의 레플리케이션 컴포넌트에 연구의 초점을 두어 레플리케이션 차수, 능동형과 수동형 레플리케이션 전략, 레플리케이션 규모에 따른 효과를 모의실험을 통해 성능향상 결과를 분석하고 설명하였다. 인터넷 사용의 신뢰성을 제고하기 위해 제안된 고장 감내를 위한 구조와 이에 관련된 레플리케이션 연구는 효율적인 인터넷 기반 분산 시스템 연구로 중요한 의미를 갖는다.

향후 관련 연구로 추가적인 연구 작업이 요구되는 사항은 다음과 같다. 첫째, 실험 간소화를 위해 동기형 시스템 모델을 가정했던 것에 더하여 비동기형 고장 탐지 기능을 통합하는 작업이 필요하다. 둘째, 효과적인 레플리케이션 기법 선택에 있어 응용설계자 단독으로 사전에 결정하는 것을 에이전트 모니터 의사결정 시스템을 이용한 좀 더 지능화되고 자동화시킬 수 있는 방안을 연구해야 한다. 셋째, 기존 일관성에 대한 최적화 연구로써 복제 서비스에 의한 가용성을 얼마만큼 개선할 수 있는지에 대한 성능평가 노력이 이루어져야 한다.

참 고 문 헌

[1] P. Dasgupta, "A Fault Tolerance Mechanism for MAg NET : A Mobile Agent based E-commerce System," *Proceedings of the Sixth International Conference on Internet Computing,* Las Vegas, NV, pp.733-739, June, 2002.

[2] E. Gendelman, L. F. Bic and M. B. Dillenourt, "An Application-Transparent Platform-Independent Approach to Rollback-Recovery for Mobile Agent Systems," *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2000)*, Taipei, Taiwan, pp. 564-571, April, 2000.

[3] D. Kotz and R. S. Gray, "Mobile Agents and the Future of the Internet," *ACM Operating Systems Review,* Vol. 33, No.3, pp.7-13, August, 1999.

[4] D. B. Lange and M. Oshima, "Seven Good Reasons for Mobile Agents," *Communications of the ACM,* Vol.42, No. 3, pp.88-89, March, 1999.

[5] P. Cao and C. Liu, "Maintaining Strong Cache Consistency in the Internet," *IEEE Transactions on Computer*, Vol. 37, No.4, pp.445-457, April, 1998.

[6] D. S. Milojicic, S. Guday and R. Wheeler, "Old Wine in New Bottles Applying OS Process Migration Technology to Mobile Agents," *Proceedings of the 3rd ECOOP'1998 Workshop on Mobile Object Systems*, July, 1998.

[7] S. Mishra, X. Jiang and B.Yang, "Providing Fault Tolerance to Mobile Intelligent Agents," *Proceedings of the 8th ISCA International Conference on Intelligent Systems,* Denver, CO, June, 1999.

[8] S. Mishra, "Agent Fault Tolerance Using Group Communication," *Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Application (PDPTA 2001)*, Las Vegas, NV, June, 2001.

[9] H. Pals, S. Petri and C. Grewe, "FANTOMAS, Fault Tolerance Mobile Agents in Clusters," *Proceedings of the 2000 International Parallel & Distributed Processing Symposium (IPDPS 2000) Workshop*, Cancun, Mexico May, 2000.

[10] V. A. Pham and A. Karmouch, "Mobile Software Agents : An Overview," *IEEE Communication Magazine*, pp.26-37, July, 1998.

[11] S. Pleisch and A. Schiper, "FATOMAS A Fault-Tolerant Mobile Agent System Based on the Agent-Dependent Approach," *Proceedings of the IEEE International Conference on Dependable Systems and Networks,* Goteborg, Sweden, pp.215-224, July, 2001.

[12] L. M. Silva, V. Batista and J. G. Silva, "Fault-Tolerant Execution of Mobile Agents," *Proceedings of the IEEE International Conference on Dependable Systems and Networks,* New York, pp.135-143, June, 2000.

[13] A. Sood, R. Simon and J. Noland, "A Light-Weight Agent Architecture for Collaborative Multimedia Systems," *Information Sciences,* Elsevier Science, Vol.140, Issues 1-2, pp.53-84, January, 2002.

[14] A. D. Stefano and C. Santoro, "Locating Mobile Agents in a Wide Distributed Environment," *IEEE Transactions on Parallel and Distributed Systems*, Vol.13, No.8, pp. 844-864, August, 2002.

[15] M. Strasser and K. Rothermel, "Reliability Concepts for Mobile Agent," *International Journal of Cooperative Information Systems*, Vol.7, No.4, pp.355-382, December, 1998.

[16] N. Wijngaards, M. Steen and F. Brazier, "On MAS Scalability," *Proceedings of the International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, University of Montreal, Canada, pp.121-126, May, 2001.

[17] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz and J. D. Kubiatowicz, "Bayeux : An Architecure for Scalable and Fault-tolerant Wide-area Data Dissemination," *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Port Jefferson, New York, June, 2001.

박 경 모

e-mail : kpark@catholic.ac.kr

1980년 중앙대학교 전산학과(학사)

1983년 서울대학교 계산통계학과 전산과학 전공(이학석사)

1990년 미국 New Jersey Institute of Technology, CIS학과(공학석사)

1994년 미국 George Mason University, IT공학부(공학박사)

1983년~1985년 삼성전자 컴퓨터소프트웨어 개발팀 연구원

1991년~1995년 GMU IT&E Lab. 연구원

1995년~1996년 한국전자통신연구소(ETRI) 컴퓨터연구단 연구원

1996년~현재 가톨릭대학교 컴퓨터정보공학부 부교수

2001년~2002년 조지메이슨대 컴퓨터학과 방문교수

관심분야 : 컴퓨터 시스템 성능평가, 분산 및 병렬처리, 고장감내