서비스 복제 및 이주를 이용한 서비스 확장성 향상 기법

(Methods to Enhance Service Scalability Using Service Replication and Migration)

김 지 원 † 이 재 유 † 김 수 동 ††
(Ji Won Kim) (Jae Yoo Lee) (Soo Dong Kim)

요 약 서비스 지향 컴퓨팅은 재사용 가능한 서비스를 사용하여 애플리케이션을 개발하는 효과적인 패러다임으로서 널리 각광받고 있다. 서비스 지향 컴퓨팅에서 서비스 소비자는 서비스를 관리하거나, 수정할 필요가 없이 서비스 제공자가 제공하는 서비스를 단지 사용하기만 하면 된다. 반면에, 서비스 제공자는 사용자가 원하는 때에 서비스를 사용할 수 있도록 자원 및 데이터 등을 관리하여야 한다. 하지만, 서비스 소비자는 불특정 다수의 특성을 갖기 때문에 서비스 제공자가 서비스 품질을 관리하기 힘들다. 따라서 서비스 수준 협약에 명시된 품질을 보장하면서 여러 명의 소비자에게 서비스를 제공하기 위한 서비스 확장성이 서비스 지향 컴퓨팅의 잠재적인 문제로 주목 받고 있다. 확장성에 대한 연구는 네트워크와 데이터베이스, 분산 컴퓨팅 등의 여러 분야에서 진행되었다. 하지만, 서비스 공학 분야에서는 아직 서비스 확장성의 정의와 관련 메트릭 등의 연구가 미흡한 실정이다. 본 논문에서는 서비스 환경을 다중 노드가 연결된 네트워크로 구성하고, 모든 노드의 자원을 통합 관리한다. 또한 동적인 서비스의 복제 및 이주 기법을 이용하여 서비스 확장성을 관리하기 위한 프레임워크를 제안한다. 3장에서는 확장성 관리 프레임워크의 구조와 기능성을 소개하고, 4장에서는 프레임워크의 기능성 실현에 필요한 확장성 향상 기법을 제안한다. 5장에서는 제안된 기법을 적용하여 프레임워크를 설계/구현하며, 6장에서는 구현된 프레임워크를 적용하여 실험을 수행한다. 실험을 통해 확장성 향상 기법의 실효성을 확인한다.

키워드 : 서비스 공학, 서비스 이주, 서비스 복제, 서비스 확장성 관리

Abstract Service-oriented computing, the effective paradigm for developing service applications by using reusable services, becomes popular. In service-oriented computing, service consumer has no responsibility for managing services, just invokes services what service providers are producing. On the other hand, service providers should manage any resources and data for service consumers can use the service anytime and anywhere. However, it is hard service providers manage the quality of the services because an unspecified number of service consumers. Therefore, service scalability for providing services with higher quality of services specified in a service level agreement becomes a potential problem in service-oriented computing. There have been many researches for scalability in network, database, and distributed computing area. But a research about a definition of service scalability and metrics of measuring service scalability is still not mature in service engineering area. In this paper, we construct a service network which connects multiple service nodes, and integrate all the resources to manage it. And we also present a service scalability framework for managing service scalability by using a mechanism of service migration or replication. In section 3, we, firstly, present

· 본 과제는 정보통신산업진흥원의 SW공학 요소기술 연구개발사업의 결과물임을 밝힙니다.

† 학생회원 : 숭실대학교 컴퓨터학과
jwkim@otlab.ssu.ac.kr
jylee81@otlab.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
sdkim777@gmail.com

논문접수 : 2010년 3월 24일
심사완료 : 2010년 5월 7일

정보과학회논문지: 소프트웨어 및 응용 제37권 제7호(2010.7)

the structure of the scalability management framework and basic functionalities. In section 4, we propose scalability enhancement mechanism which is needed to release functionality of the framework. In section 5, we design and implement the framework by using proposed mechanism. In section 6, we demonstrate the result of our case study which dynamically manages services in multi-nodes environment by applying our framework. Through the case study, we show the applicability of our scalability management framework and mechanism.

Key words : Service Engineering, Service Migration, Service Replication, Service Scalability Management

1. 서 론

서비스 지향 컴퓨팅은 재사용 가능한 서비스를 사용하여 애플리케이션을 개발하는 효과적인 패러다임으로서 주목 받고 있다. 서비스 지향 컴퓨팅은 (1)서비스가 소비자의 컴퓨터가 아닌 원격지의 서버에서 운영되고, (2)인터넷을 통해 서비스의 가능성을 제공하며, (3)서비스 운영에 필요한 모든 자원은 서비스가 배치된 서버의 자원을 사용하고, (4)서비스 업데이트 등의 유지보수는 서비스 제공자가 담당하고 소비자는 단순히 제공되는 서비스를 이용한다는 점에서 기존의 컴퓨팅과 다르다.

서비스 지향 컴퓨팅에서는 서비스 사용자 그룹이나 서비스 요청의 빈도가 미리 알려져 있지 않으므로, 예측하지 못한 많은 양의 서비스 요청은 서비스 서버의 자원 고갈과 네트워크의 집중문제를 야기시켜 서비스의 전반적인 품질을 저하시킬 수 있다. 이러한 문제는 서비스 확장성(Scalability)이 충분히 높지 않을 때 발생한다. 낮은 수준의 확장성을 가진 서비스는 사용자의 선호도를 낮추게 된다. 따라서, 서비스 지향 컴퓨팅에서는 소비자의 증가에 따른 서비스 품질의 저하를 최소화하며 보다 많은 소비자에게 서비스를 제공하기 위한 '서비스 확장성'에 대한 연구가 요구되고 있다[1,2].

확장성에 대한 연구는 네트워크와 데이터베이스, 분산 컴퓨팅 등의 여러 분야에서 진행되었다. 기존 연구에서는 동적 라우팅을 통하여 네트워크에 연결된 다중 노드에서 실행 중인 동일한 컴포넌트에 부하를 분산하거나, 데이터베이스의 복제와 동기화를 통한 부하분산, 혹은 태스크를 분할하여 병렬 처리하는 방식으로 확장성을 관리한다. 하지만, 서비스는 불특정 다수의 소비자를 대상으로 하기 때문에, 소비자의 서비스 이용에 따른 부하 발생량을 예측하기 힘들고, 부하량이 급격히 증가하는 피크타임이 존재한다. 이와 같은 서비스의 특성으로 인하여 서비스 확장성에 대한 정의와 관련 메트릭, 확장성 관리 기법 등이 새롭게 정의되어야 한다. 즉, 각 분야에서 필요한 확장성 연구와 연관 지어서 서비스 수준의 확장성을 확보하는 기법 등이 부족한 상황이다.

확장성을 높이기 위한 전통적인 접근방법은 새로운 하드웨어를 추가하여 시스템의 확장성을 높이는 것이다. 일반적으로 Scale-UP은 하나의 노드에 추가적인 하드웨어를 설치하여 해당 노드의 물리적인 처리능력을 향상시키는 것이며, Scale-Out은 시스템에 새로운 노드를 추가하여 대상 시스템의 처리능력을 향상시킨다. 하지만, 하드웨어적인 확장성 향상 기법은 하드웨어 설치 및 운영을 위해 시스템을 멈출 필요가 있고, 새로운 하드웨어를 추가할수록 추가되는 자원에 대한 실제 확장성 향상 비율이 감소하는 문제점을 갖는다.

본 논문에서는 전통적인 하드웨어적 기법이 아닌 소프트웨어 측면에서 서비스 확장성 향상 기법을 제안한다. 소프트웨어적인 확장성 향상 기법이란 기존의 하드웨어적인 기법이 아닌 동일한 자원 상황에서도 소프트웨어적인 처리를 통하여 자원의 활용성을 높이고 결과적으로 서비스의 확장성을 향상시키는 기법이다. 또한 전통적인 기법의 문제점인 자원의 정적 추가에 대한 해결 방안으로 서비스 운영 중에 새로운 노드를 추가하고, 해당 노드에 서비스를 배치/운영하는 동적 노드 관리기법을 제안한다. 제안된 기법은 소비자의 서비스 요청을 광역 확장성 관리기에 등록된 각 노드에 효율적으로 분배함으로써, 노드 추가에 따른 운영비용을 최소화하여 추가되는 자원에 대한 실제 확장성 향상 비율을 높인다.

본 논문에서는 소프트웨어적인 서비스 확장성 향상 기법을 이용한 실용적인 서비스 확장성 관리 프레임워크를 제안한다. 먼저, 3장에서는 서비스 확장성 관리를 위한 프레임워크의 구조를 소개하고 확장성 관리에 사용되는 주요 기능성에 대하여 기술한다. 또한 서비스 확장성을 측정하기 위한 관련 메트릭을 정의한다. 4장에서는 제안된 프레임워크에서 사용되는 확장성 향상 기법에 대하여 설명하고, 각 기법에 대한 알고리즘을 기술한다. 5장에서는 제안된 기법을 적용하여 서비스 확장성 관리 프레임워크를 설계/구현하며, 6장에서 구현된 프레임워크를 적용하여 다중 노드 환경에서 서비스를 동적으로 관리하여 확장성을 향상시키는 실험을 수행한다. 이 실험을 통하여 제안된 확장성 관리 프레임워크와 프레임워크에서 사용된 확장성 향상 기법의 실효성을 확인한다.

본 논문에서 제안된 서비스 확장성 관리 프레임워크와 기법을 사용하여 서비스 시스템의 모든 자원을 동적

으로 등록, 관리하고 부하량에 따라 새로운 서비스 호출을 동적 라우팅함으로써 서비스 시스템의 확장성을 극대화할 수 있다.

2. 관련 연구

Wang[3]의 연구는 서비스기반 그리드환경에서 에이전트를 이용한 부하분산 모델을 제시하고 있다. 제시된 모델은 여섯 개의 컴포넌트로 구성되어 있고, 각 컴포넌트는 Global Scheduler(GS), Local Scheduler(LS), Balancer Agent(BA), Fault Tolerance Agent(FTA), History Information Database(HID), Load Monitoring Agent(LM) 이다. GS와 LS는 부하를 분산하기 위한 역할을 수행하며, GS는 LS의 영역을 포함하고 있다. BA는 GS와 LS에 탑재되어 부하분산을 수행하며, FTA는 결함 허용(fault tolerant)을 수행한다. HID는 부하에 대한 데이터를 기록하며 데이터는 GS에 저장된다. LM은 서비스 인스턴스가 부담하고 있는 부하량을 감시한다. 여섯 개의 컴포넌트와 함께 부하 측정기법과 부하분산 기법을 제시하여 성능과 확장성을 향상시키고 있다. 그러나, 서비스노드의 확장성이 고려되지 않아 제한된 노드만을 활용해야 한다. 따라서, 기대 이상으로 발생하는 서비스 요청에 대응하기 위한 해결책이 요구된다.

Aversa[4]는 그리드시스템에서 모바일 에이전트 기반의 확장성 향상과 부하분산 기법을 제시하고 있다. 제시된 기법의 아키텍처는 서비스와 자원을 관리하고 배포하는 Directory Service, 성능관련 요소를 감시는 컴포넌트와 이를 관리하는 Resource Manager와 Resource Manager에 의해 세워진 계획을 수행하는 Agents Manager 으로 구성되어 있다. Agents Manager는 수집된 성능정보 데이터를 기반으로 동적인 부하분산 계획을 수립하고 수행한다. 또한 Agents Manager은 가용한 그리드 자원을 Directory Service를 통해 수집하고 있다. 동적인 그리드 자원의 활용과 동적 부하분산을 통해 성능과 확장성의 향상을 기대 할 수 있다. 하지만, 기법을 적용하기 위한 실질적인 방법이 제시되지 않았고, 웹서비스를 포함한 그리드 자원의 관리 방안이 제시되지 않아, 그리드 자원의 추가와 제거에 대한 기법이 묘연하다.

Chawathe[5]의 연구에서는 서비스 네트워크를 구성하여 확장성과 결함 회피를 제공하고 있다. 논문에서 제시하고 잇는 SNS(Scalable Network Services) 아키텍처는 중앙집중 형 관리자를 통해 부하분산과 노드 관리 그리고 결함에 대한 회피를 제공하고 있다. SNS에는 모든 노드를 관리하는 중앙관리자와 서비스의 기능을 수행하는 Worker, 그리고 Worker에 포함되어 있는 라이브러리로서 중앙관리자의 연결을 위한 Worker Driver으로 구성된다. 중앙관리자는 노드의 Worker Driver로부터 노드가 부담하는 부하에 대한 데이터를 수집하고 부하분산에 대한 계획을 수립하고 수행한다. 부하의 증가는 중앙관리자로부터 유휴노드를 Worker으로 참여시킨다. 이 때, 한시적인 부하의 증가로 인해 Worker을 추가 작업이 진행 되어서는 안 된다. SNS는 중앙집중형 관리로 인해 모든 Worker의 상황을 파악하고, 구성된 네트워크에서 효과적인 부하분산 및 확장성향상을 도모 한다. 그러나, 중앙집중 형 구성은 병목현상을 유발하기 쉽다. 서비스 사용자의 요청이 모두 집중되고 Worker간의 상호작용 또한 중앙관리자를 통하기 때문에, 서비스 중계과정에서 지연발생의 소지가 높다.

이와 같이 네트워크를 이용한 확장성 향상과 결함회피에 대한 많은 연구가 진행되었지만, 정적인 노드 관리와 서비스의 특성을 고려하지 못한 한계점이 존재한다. 따라서, 본 논문에서는 다중 노드로 구성된 서비스 시스템의 특성을 고려한 소프트웨어적인 확장성 향상 기법을 제안한다. 또한 기존의 하드웨어적인 기법의 문제점인 정적인 노드 관리를 해결하기 위한 동적인 서비스 노드 추가와 서비스 복제를 통한 동적 서비스 관리 기법을 제안한다.

3. 서비스 확장성 프레임워크

본 논문에서 제안하는 서비스 확장성 프레임워크는 등록된 노드의 자원을 통합하여 서비스의 운영에 필요한 자원을 동적으로 할당한다. 이 장에서는 새로운 노드를 등록하고, 등록된 모든 자원을 통합하여 동적으로 관리하기 위해 필요한 주요 기능과 등록된 서비스의 확장성을 향상시키기 위한 절차에 대해 기술한다.

3.1 서비스 확장성 프레임워크의 개요

서비스 확장성 프레임워크는 다중 노드 상의 다중 서비스 환경, 즉 서비스 그리드 환경을 대상으로 한다. 따라서 서비스를 제공하기 위한 복수 개의 노드가 존재하고, 서비스가 수행되는 동안 서비스의 확장성을 향상시키기 위한 목적을 가지며 그림 1의 구조로 이루어진다.

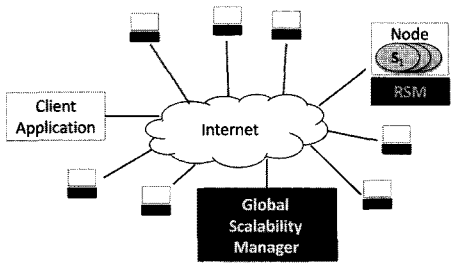

그림 1 서비스 확장성 프레임워크의 개략적인 구조

(1) 광역 확장성 관리기(Global Scalability Manager, GSM): 프레임워크에 등록된 모든 노드의 자원을 통합하여 관리하고, 실시간 노드의 상태를 고려하여 확장성 향상 계획을 수립 및 지시한다.

(2) 지역 확장성 관리기(Regional Scalability Manager, RSM): 각 노드에 위치하여 해당 노드의 자원 상황을 감시하고, 수집된 정보를 GSM으로 전달한다. 또한 GSM에서 수립된 계획에 따라 서비스 이주 및 복제 등의 서비스 확장성 향상 기법을 직접 수행한다. RSM이 설치된 각 노드에는 동적으로 서비스가 배치, 운영되고, GSM에 의해 관리된다.

서비스 확장성 프레임워크가 수행하는 서비스의 동적 관리는 하나의 서비스가 특정 노드에 정해진 자원에 정적으로 배치되어 운영되는 것이 아니라 운영 중인 서비스의 이용량, 자원 상황, 서비스에 부여된 부하량의 변화를 반영하여 동적으로 서비스가 노드에 배치, 운영되는 것이다. 서비스의 동적 관리를 위해 각 노드의 RSM은 해당 노드의 서비스 미들웨어에 서비스를 동적으로 배치하기 위한 Hot-Deploy[6] 기능을 제공하고 해당 노드에서 운영 중인 서비스의 현재 상태를 GSM으로 전달한다. 또한, GSM은 서비스 이용량의 변화와 각 노드의 RSM으로부터 전달받은 데이터를 기반으로 전체적인 부하분산을 진행하고, 필요에 따라 서비스를 다른 노드로 복제 혹은 이주하는 기능을 실행한다.

3.2 서비스 확장성 프레임워크의 주요 기능

서비스 확장성 프레임워크는 대상 서비스의 확장성을 향상시키기 위해 그림 2와 같은 네 가지 절차를 반복 수행한다.

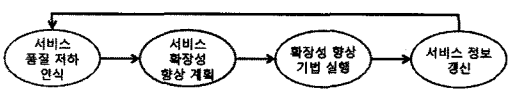

그림 2 서비스 확장성 관리 절차

'서비스 품질 저하 인식' 단계에서는 서비스가 서비스 수준협약(Service-Level Agreement, SLA)[7]에 명시된 품질을 보장하는지 감시하고, 품질의 저하를 인식한다. '서비스 확장성 향상 계획' 단계에서는 서비스 품질 저하를 야기하는 문제의 유형을 분석하고, 분석된 문제를 해결하기 위한 기법과 실행 계획을 수립한다. '서비스 확장성 향상 기법 실행' 단계에서는 이전 단계에서 수립된 계획에 의해 결정된 기법을 실행한다. 서비스 확장성 향상 기법에는 서비스 복제 혹은 이주 방법이 있으며, 자세한 사항은 4장에서 기술된다. '서비스 정보 갱신' 단계에서는 실행된 서비스 확장성 향상 기법에 따라 변경된 서비스에 대한 정보를 갱신한다. 서비스의 확장성 관리 절차를 수행하기 위해 프레임워크에서 제공되는 기능은 다음과 같다.

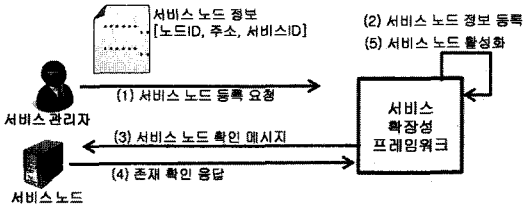

그림 3 서비스노드 등록 절차

새로운 노드 등록: 서비스 확장성 프레임워크에 새로운 노드를 등록하는 것은 그림 3에서 보여지는 절차로 진행된다.

첫째, 서비스노드의 서비스 관리자는 노드의 IP 주소와 배치된 서비스 이름, SLA 등 등록에 필요한 정보를 포함하여 서비스노드 등록 요청을 한다. 둘째, 서비스 확장성 프레임워크는 제공된 정보를 기반으로 노드와 서비스에 유일한 식별 ID를 부여하고 이를 '비활성화' 상태로 서비스노드 목록에 등록한다. 셋째, 실제 서비스 노드의 존재 유무를 확인 메시지를 발송한다. 넷째, 서비스노드에 설치된 RSM에서 메시지를 접수하여 응답 메시지를 전송한다. 다섯째, 확인 메시지를 받은 노드의 상태정보를 '활성화' 상태로 전환한다. 프레임워크는 활성화된 노드에 대한 통합 관리를 시작한다.

서비스 품질 실시간 감시: GSM은 RSM으로부터 전달받은 분산된 서비스의 품질을 분석하고, 이를 통합하여 해당 서비스의 전체 품질을 측정함으로써 서비스의 전체적인 품질을 감시한다. 서비스 확장성 프레임워크는 이미 개발된 서비스의 품질을 대상으로 하기 때문에, 감시하는 품질은 ISO 9126[8]에 정의된 사용 품질 특성(Quality in Use)을 대상으로 한다. ISO 9126에 정의된 네 가지 사용 품질 특성은 유효성, 생산성, 안정성, 만족도이며, 안정성은 서비스 개발 시 고려해야 하는 상황이기 때문에 감시 대상에서 제외하고, 유효성과 만족도가 생산성에 의해 많은 영향을 받고 비례하기 때문에, 생산성을 기준으로 서비스의 품질을 명시한다.

생산성은 자원, 시간 등의 주어진 상황 하에 산출된 결과에 대한 측정치이다. 즉, 일정 시간 동안 정해진 자원을 소모하여 서비스가 처리할 수 있는 처리량(Throughput)이다[9-11]. 서비스의 처리량은 응답시간[12,13]과 밀접한 관계를 가지며, 다음과 같이 정의된다.

$$Throughput(s) = \frac{n}{\sum_{i=1}^{n} \text{ResponseTime}_i}$$

각 노드의 RSM은 서비스의 응답시간을 GSM으로

전달하고, GSM은 전달받은 응답시간을 기준으로 각 노드에서 운영 중인 서비스의 품질을 측정한다. 서비스의 이상 유무를 판단하기 위해 SLA에 명시된 서비스의 최대응답시간을 역수로 하여 최소처리량을 계산하고, 실제 측정된 서비스의 처리량과 비교한다. 만약, 실제 처리량이 작다면 해당 서비스에 문제가 있다고 판단한다.

문제 상황 분석: 서비스의 확장성에 발생한 문제를 인식한 GSM은 현재 상황을 분석하여 서비스 확장성을 향상시키기 위한 적절한 기법을 선택한다[14,15]. 문제를 인식한 시점에서 해당 노드로의 서비스 호출이 더 이상 증가하지 않도록 조치한 후, 분석을 시작한다. 먼저, 서비스의 발생 호출 수와 현재 서비스가 처리 중인 호출의 수를 비교하여, 문제가 되는 노드와 발생한 문제의 유형을 식별한다. 발생한 문제의 유형에 따라 부하분산의 비율을 재조정하거나 서비스를 복제 혹은 이주 중에서 적용 가능한 해결 기법을 선택한다.

후보 노드 조회: 후보 노드는 확장성 문제를 해결하기 위해 서비스의 복제 혹은 이주 기법을 적용할 수 있는 대상 노드이다. 서비스 확장성 프레임워크에 등록된 모든 노드와 서비스는 GSM으로부터 유일한 ID를 부여받고 표 1과 같이 관리된다.

GMS은 '문제 상황 분석'에서 식별된 문제 서비스의 ID를 기반으로 서비스가 배치된 노드들의 목록, 각 노드에 분산된 부하량, 평균응답시간을 수집하고 이를 이용하여 '문제 상황 분석'에서 결정된 확장성 향상 기법을 적용 할 후보 노드를 선출한다. 서비스의 부족한 처리량과 다른 노드의 가용한 처리량에 따라서 후보 노드의 수가 결정된다.

서비스 전달: 서비스 전달은 하나의 RSM이 다른 노드의 RSM으로 서비스 컴포넌트를 전달하는 것이다. 서비스를 전달하는 RSM은 서비스 컴포넌트를 SOAP[16] 메시지에 첨부하여 GSM으로부터 전달된 후보 노드 목록에 존재하는 각 노드의 RSM에게 전달한다. 엔드포인트 주소는 모든 RSM이 표 2와 같이 동일한 형태를 가진다.

서비스를 전달받은 RSM은 서비스 미들웨어의 Hot-Deploy 기능을 이용하여 전달받은 서비스를 동적으로 배치한다.

서비스 제거: RSM이 서비스 이주를 위해 서비스 컴포넌트를 GSM에게 전달 한 후 노드에 배치된 서비스를 제거한다. 서비스 제거는 노드의 RSM이 해당 서비스 컴포넌트를 삭제하여 수행된다. 동시에 더 이상 서비스가 제거된 노드로 부하를 분산하지 않도록, GSM에서 해당 서비스에 대한 부하분산테이블을 갱신한다.

표 2 서비스 전달을 위한 엔드포인트 주소

번호	노드	엔드포인트 주소
1	Node1	http:// *{Domain or IP}-1* / RSM /UploadService
2	Node2	http:// *{Domain or IP}-2* / RSM /UploadService
…	…	…
N	Noden	http:// *{Domain or IP}-N* / RSM /UploadService

서비스 확장성 측정: 문제가 발생한 서비스에 대하여 확장성 향상 기법을 적용하기 전과 후의 품질을 비교하여 해당 서비스의 확장성을 측정한다[17]. 따라서 서비스 확장성은 다음과 같이 계산된다.

$$Scalabilit\ y(s) = \frac{Throughput_{after}}{Throughput_{before}}$$

분모는 서비스 확장성 향상 기법을 적용하기 전의 문제가 발생한 서비스 처리량이다. 분자는 기법 적용 후에 측정된 처리량이다. 기법이 성공적으로 적용되면 서비스의 처리량이 증가하여 최소 품질을 만족해야 한다. 즉, 확장성이 1 이상의 값을 가져야 한다.

4. 확장성 향상 기법

이 장에서는 서비스 확장성 프레임워크에서 관리 대상 서비스의 확장성을 향상시키기 위한 기법과 관련된 알고리즘을 설명한다.

4.1 확장성 향상 계획 기법

'확장성 향상 계획' 단계에서는 '문제 인식' 단계에서 발견된 문제 상황을 분석하여, 상황에 맞는 확장성 향상 기법을 적용하기 위한 계획을 수립한다.

먼저 발생한 문제를 분석하기 위해, 해당 서비스의 전체 가용한 호출의 수(Total available number of requests)와 발생한 서비스 호출의 수(number of service requests), 하나의 노드에서 처리할 수 있는 가용한 호출의 수(Available number of requests), 해당 노드에 할당된 서비스 호출의 수(number of distributed request)를 가져온다. 두 자원량의 관계에 따라 다음과 같이 두 가지 경우로 구분된다;

표 1 노드 정보 관리 테이블

번호	노드 ID	서비스 ID	서비스 주소	평균응답시간 (ms/req)	처리량 (req/sec)
1	Node1	Service1	node1.mssec.ssu.ac.kr/Service1	0.9	22
2	Node2	Service1	node2.mssec.ssu.ac.kr/Service1	0.6	33
…	…	…	…	…	..
N	Noden	Service2	nodeN.mssec.ssu.ac.kr/Service2	0.5	18

전체 가용한 호출의 수가 충분한 경우: 노드 전체에서 가용한 서비스 호출의 수가 발생한 서비스 호출의 수보다 많지만, 각 노드 별로 살펴보면 확장성이 부족한 노드가 존재하는 상황이다. 즉, 문제가 발생한 노드로 과도한 부하가 집중된 상태이며, 각 노드에서 가용한 호출 수를 비교하여 부하분산 테이블을 갱신을 통해 해결된다.

전체 가용한 호출의 수가 부족한 경우: 서비스 호출의 증가로 모든 노드가 문제 노드인 상황과 특정 노드에 문제가 발생하여 더 이상 서비스를 제공할 수 없는 두 가지의 상황에 의해 발생할 수 있다. 두 번째의 경우, 하나의 노드가 사용 불가능한 상태로 전환되어 해당 노드의 가용한 호출의 수가 0이 된다. 첫 번째 상황은 새로운 노드에 서비스를 복제하고, 두 번째 상황은 비슷한 조건의 새로운 노드에 문제 노드에서 제공되던 서비스를 이주시키는 것으로 해결이 가능하다. 확장성 향상 계획을 수립하기 위한 알고리즘은 다음 알고리즘 1과 같다.

위의 알고리즘은 문제가 발생한 서비스를 입력 값으로 받고, 해당 서비스에 발생한 문제의 유형에 따라 결정된 확장성 향상 기법을 출력 값으로 반환한다. Analyze-Fault에서 발생한 문제의 유형을 판단하고, 이를 기반으로 PlanActuation에서 적용 가능한 확장성 향상 기법을 결정하고 관련 모듈을 호출한다.

4.2 서비스 복제 기법

서비스 복제는 노드에 배치되어 제공 중인 서비스를 동적으로 다른 노드에 복제/배치하는 활동이며 기존의 서비스 운영에 영향을 끼치지 않고 진행된다. 복제된 서비스는 새로운 노드에서 제공하는 자원을 추가적으로 확보하여 처리 가능한 서비스 호출의 수를 확대할 수 있다. 그림 4는 $Node_1$에 배치된 서비스 SVC_{1-1}이 $Node_2$로 복제되는 과정을 보여준다.

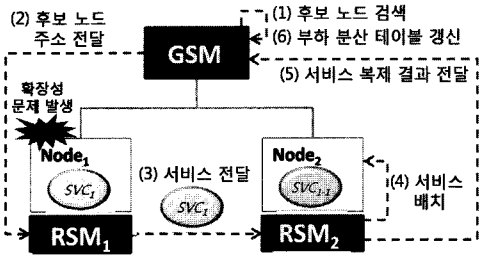

그림 4 서비스 복제

```
1:    Algorithm PlanActuation
2:      Input: Service SVC
3:      Output: Scalability Enhancing Method
4:      Set Scalability Enhancing Method to null
5:      AnalyzeFault(SVC)
6:        Retrieve TotalAvailableNumberofRequests and NumberOfServiceRequest of the problematic service SVC
7:        if TotalAvailableNumberofRequests > NumberOfServiceRequest then
8:          for each node n_i from NodesList of SVC
9:      Retrieve AvailableNumberOfReqeusts, NumberOfDistributedRequests from n_i
10:     if AvailableNumberOfReqeusts < NumberOfDistributedRequests then
11:       Set FaultType to 1
12:       Add n_i into ProblematicNodesList with FaultType
13:     end if
14:   end for
15:   else
16:     for each node n_i from NodesList
17:       Retrieve AvailableNumberOfReqeusts, NumberOfDistributedRequests from n_i
18:       if AvailableNumberOfReqeusts < NumberOfDistributedRequests then
19:         Set FaultType to 2
20:         Add n_i into ProblematicNodesList with FaultType
21:       end if
22:     end for
23:   end if
24:   if ProblematicNodesList is not null then
25:     Set FaultType to 3
26:     Add NodesList into ProblematicNodesList with FaultType
27:   end if
28:   return ProblematicNodesList
29: PlanActuation(FaultType)
30:   switch FaultType
31:     case 1: Scalability Enhancing Method is Update Load Balancing Table
32:       break
33:     case 2: Scalability Enhancing Method is Service Replication
34:       break
35:     case 3: Scalability Enhancing Method is Service Migration
36:       break
37:   return Scalability Enhancing Method
```

알고리즘 1

GSM은 서비스 복제 기법을 적용할 후보 노드를 검색한다. 후보 노드는 GSM에 등록된 노드 중에서 동일한 서비스를 배치 중이지 않고 여유 자원을 보유하고 있는 노드가 대상이 되고, 부족한 처리량에 따라 하나 혹은 그 이상의 후보 노드를 선택한다. 선택된 후보 노드의 주소를 문제가 발생한 노드의 RSM_1에게 전달한다. 대상 노드의 주소를 전달받은 RSM_1은 각 대상 노드의 RSM_2로 서비스 SVC_1을 전달한다. 이 때, 서비스 SVC_1의 서비스 컴포넌트는 SOAP 메시지에 첨부되어 전달된다. 서비스를 전달받은 RSM_2는 메시지에 첨부된 서비스 컴포넌트를 $Node_2$에 동적으로 배치한다. 성공적으로 서비스 SVC_{1-1}가 배치되면 성공 여부를 GSM에게 전달하고, GSM은 $Node_2$를 부하분산테이블의 정보에 추가한다. GSM은 각 노드의 자원 비율을 고려하여 부하분산 비율을 다시 설정한다. 서비스 복제 알고리즘은 아래 알고리즘 2와 같이 정의된다.

서비스 복제 알고리즘은 Service Status을 입력 값으로 받는다. Service Status은 서비스의 자원 상황 정보와 관련 노드 목록, 문제 노드 목록을 포함한다. FindCandidateNodes는 부족한 자원량을 기준으로 서비스를 복제할 새로운 노드의 후보 목록을 반환한다. 후보 노드 목록은 NotifyCandidateNodes를 이용하여 GSM에서 RSM으로 전달된다. 후보 노드 목록을 전달받은 RSM은 TransmitService를 이용하여 대상 노드의 RSM에 서비스를 전달한다. 서비스를 전달받은 RSM은 해당 노드의 서비스 미들웨어에 HotDeployService를 이용하여 전달받은 서비스를 동적으로 배치하며, 배치 결과를 NotifyReplicationResult를 이용하여 GSM에 전달한다. GSM은 전달받은 서비스 복제 결과에 따라서 UpdateLoadBalancingTable을 이용하여 서비스에 대한 부하분산 테이블을 갱신한다.

4.3 서비스 이주 기법

```
1: Algorithm ServiceReplication
2:   Input: Service Status
3:         Output: null
4:   FindCandidateNodes(RequiredThroughput)
5:         // 부족한 처리량을 기준으로 서비스를 복제할 새로운 노드의 후보 목록을 반환한다.
6:           Retrieve NodesList
7:           while RequiredThroughput is greater than 0
8:             Retrieve AvailableNumberOfRequests of services in n_i from NodesList
9:             if AvailableNumberOfRequests is greater than RequiredThroughput then
10:        Add n_i to CandidateNodes
11:      else
12:        Add n_i to CandidateNodes
13:        Subtract AvailableNumberOfRequests from RequiredThroughput
14:      end if
15:    end while
16:    return CandidateNodes
17: NotifyCandidateNodes(CandidateNodes)
18: // 문제가 발생한 노드에 CandidateNodes를 전달한다.
19: TransmitService(CandidateNodes)
20:    Retrieve SVCPackage
21:    for each node n_i from CandidateNodes
22:      SendSOAPMessage(n_i, SVCPackage)
23:    // 서비스 컴포넌트를 첨부한 SOAP 메시지를 노드 n_i의 RSM으로 전달한다.
24:    end for
25:    return
26: HotDeployService(SVCPackage)
27: // 전달받은 서비스 컴포넌트를 노드의 서비스 미들웨어에 동적 배치하고, 결과를 ReplicationResult에 저장한다.
28: NotifyReplicationResult(ReplicationResult, NodeAddress)
29: // 서비스 복제 결과 ReplicationResult와 해당 노드의 주소 NodeAddress를 GSM으로 전달한다.
30: UpdateLoadBalancingTable(ReplicationResult, NodeAddress)
31:   if ReplicationResult is true
32:     Retrieve NodesList, TotalAvailableResources
33:     Set TotalAvailableResources to 0
34:     Add node n with same NodeAddress into NodeList
35:     for each node n_i from NodeList
36:       Add available resources amount on n_i into TotalAvailableResources
37:     end for
38:     for each node n_k from NodeList
39: //서비스에 대해 배정된 전체 자원량에 대한 각 노드의 자원량의 비율을 기준으로 상대적인 부하 할당량을 부여
40:     end for
41:   end if
42:   return
```

알고리즘 2

서비스 이주는 노드에 배치된 서비스를 새로운 노드로 이동시켜서 동일한 역할을 수행하도록 하는 활동이다. 이 것은 불안정한 노드로의 서비스 호출 분배를 우선적으로 중지함으로써 전체적인 서비스 품질저하를 줄이기 위한 방법이다.

그림 5에서 RSM_1 은 $Node_1$ 에 직접 발생한 문제로 인해 서비스 제공이 힘들어지는 상황을 인식하고, 문제 상황에 대한 정보를 서비스 컴포넌트와 함께 GSM으로 전달한다. GSM은 RSM_1 으로부터 서비스 이주 요청을 받는 즉시 $Node_1$ 으로 연결된 모든 서비스의 연결을 종료하고 부하분산 테이블의 정보를 갱신한다. $Node_1$ 이 제공하던 자원량을 기준으로 후보 노드를 검색하여, 결정된 RSM_2 에게 RSM_1 으로부터 전달받은 서비스 컴포넌트를 다시 전달한다. RSM_2 는 전달받은 서비스 컴포넌트를 $Node_2$ 의 서비스 미들웨어에 동적으로 배치하고 이주 결과를 GSM에게 전달한다. 이주 결과에 따라 GSM은 $Node_2$ 를 부하분산 테이블의 정보에 추가하고, 서비스와 관련된 모든 노드의 가용한 처리량 비율을 고려하여 부하분산 비율을 다시 설정한다. 서비스 이주 알고리즘은 다음 알고리즘 3과 같이 정의된다.

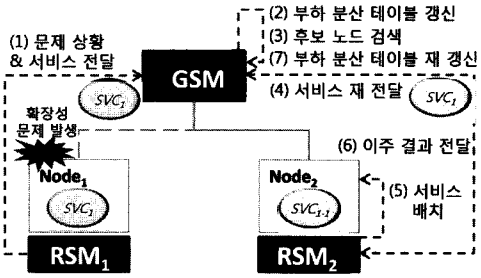

그림 5 서비스 이주

노드에 문제가 발생하여 더 이상의 서비스 제공이 불가능할 때, 해당 노드의 RSM은 NotifyProblemStatus를 이용하여 노드의 현재 상황과 이주할 서비스를 GSM으로 전달한다. GSM은 이주신청을 한 노드의 정보를 RemoveNodeFromList를 이용하여 부하분산 테이블에서 우선 제거한 후, FindCandidateNodes를 이용하여 이주할 후보 노드를 검색하고, TransmitService를 이용하여 직접 대상 노드의 RSM으로 서비스를 전달한다. 전달받은 RSM은 HotDeployService를 이용하여 서비스를 동적 배치하고 결과를 다시 GSM으로 전달한다. GSM은 서비스 이주 결과에 따라 부하분산 테이블을 갱신한다.

5. 서비스 확장성 프레임워크 설계 및 구현

본 논문에서 제안하는 기법은 JAVA환경에서 구현된다. 따라서 Java Runtime Environment 구성된 플랫폼과 OS 환경 하에서는 운영이 가능하다. 또한 서비스운용을 위한 미들웨어가 고려되었고, 미들웨어는 Hot-Deploy를 지원해야 한다.

5.1 설계 모델

그림 6과 같이 GSM은 서비스 사용자요청을 수집하여 분산하는 노드에 설치되고, RSM은 서비스가 배치되는 서비스노드에 설치된다.

사용자의 요청은 부하분산노드의 GSM에 집중되고, GSM은 연결된 하나 이상의 서비스노드에게 서비스 요청을 전달한다. 서비스노드에게 전달되는 요청은 서비스가 수신하여 처리하고 결과 값을 GSM에게 반환한다. 그림 6에서 부하분산노드$_1$ 은 서비스$_1$ 에 대한 부하분산

```
1:      Algorithm ServiceMigration
2:  Input: System Status
3:  Output: null
4:      NotifyProblemStatus(CurrentStatus, SVCPackage)
5:      // 문제 상황과 노드에 배치된 서비스 컴포넌트를 첨부하여 GSM에게 서비스 이주를 신청한다.
6:      RemoveNodeFromList(NodeAddress)
7:        Retrieve NodeList, TotalAvailableNumberOfRequests
8:        Set TotalAvailableNumberOfRequests to 0
9:        Remove node n with same NodeAddress from NodeList
10:   for each node n_i from NodeList
11:     Add AvailableNumberOfRequests of n_i into TotalAvailableNumberOfRequests
12:   end for
13:   return
15: FindCandidateNodes(RequiredThroughtput)
16: // 부족한 자원량을 기준으로 서비스를 이주할 새로운 노드의 후보 목록을 반환한다.
17: HotDeployService(SVCPackage)
18: // 전달받은 서비스 컴포넌트를 노드의 서비스 미들웨어에 동적 배치하고, 결과를 ReplicationResult에 저장한다.
19: NotifyResult(Result, NodeAddress)
20: // 서비스 복제 결과 Result와 해당 노드의 주소 NodeAddress를 GSM으로 전달한다.
21: UpdateLoadBalancingTable(Result, NodeAddress)
22: //서비스에 대해 배정된 전체 자원량에 대한 각 노드의 자원량의 비율을 기준으로 상대적인 부하 할당량을 부여
```

알고리즘 3

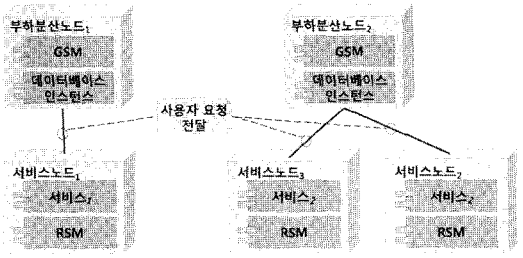

그림 6 플랫폼 모델

을 수행하고 부하분산노드$_2$는 서비스$_2$에 대한 부하 분산만을 수행한다. 서비스노드$_2$에서 제공되던 서비스$_2$는 확장성에 문제가 발생하고 발생한 문제를 극복하기 위해 서비스노드$_3$의 여유자원을 할당받은 예이다. 이 때, 부하분산노드$_2$에 배치 되어 있는 GSM은 사용자의 요청을 두 개의 노드에 적절히 분산하여 전달함으로써, 서비스노드$_2$과 서비스노드$_3$의 자원을 균형적으로 소모하도록 유도한다. GSM이 연결된 서비스노드들의 자원의 그룹화하고 하나의 통합된 자원처럼 동적으로 활용함을 의미한다[18]. 그리고, 본 논문에서 제안하는 프레임워크는 그림 6과 같이 하나의 GSM은 하나의 서비스 확장성 향상만을 고려하고 있음을 확인할 수 있다.

그림 7은 프레임워크를 구성하는 GSM과 RSM의 컴포넌트 모형을 나타낸다. GSM은 세 개, RSM은 세 개의 구성요소를 가지고 있다.

그림 7에서 GSM의 부하분산실행기(Load Distributor-LD)는 등록된 서비스노드들의 정보와 각 노드에서 부담하고 있는 부하를 계산한 데이터를 기반으로, 사용자의 요청이 특정노드에 집중되지 않도록 분산하여 전달 한다. 전역 확장성 플래너(Scalability Inter Planner-SIP)는 확장성 향상을 계획하고 관련 기법을 수행한다. 서비스노드의 RSM이 노드의 문제를 인지하고 GSM에게 알리면 4장의 알고리즘과 기법 기반으로 위기를 극복할 확장성 계획을 수립하고 기법을 선정한다. 수립된 계획과 기법은 RSM에게 전달되어 수행을 지시한다. GSM에 등록된 노드의 정보는 노드관리자(Node Manager-NM)에 의해 관리 된다. 관리목록의 갱신은 RSM과 상호작용을 통해 수행된다. NM은 RSM이 전송한 서비스노드의 정보를 수신하여 서비스노드의 추가 혹은 삭제, 서비스의 추가 또는 삭제를 수행한다. 또한 서비스의 응답시간과 서비스 요청의 처리량을 수집하여 서비스노드가 부담하는 부하를 계산한다. 서비스의 응답시간과 서비스 요청량을 수집하며, 노드에서 소모되고 있는 자원의 양을 함께 수집하여 서비스노드의 상태를 확인 할 수 있는 데이터를 생성한다.

RSM의 브로드캐스터(Broadcaster-BC)는 서비스노드의 존재를 알린다. RSM이 구동될 때, BC는 서비스노드의 정보를 네트워크에 브로드캐스팅[19]하고 GSM이 정보를 수신한다. 서비스노드의 IP와 노드ID를 포함하고 있는 정보는 GSM의 NM이 수집하고, 서비스노드가 새로운 노드일 경우 서비스노드목록에 추가하고, 기존에 인지 되었던 노드의 경우 변경된 정보만을 갱신처리 한다. 서비스 전송기(Service Transporter-ST)는 서비스 컴포넌트를 전송 또는 수신하는 역할을 한다. ST는 서비스노드의 서비스와 서비스 컴포넌트 파일의 물리적 위치정보를 포함하는 서비스 컴포넌트의 정보를 관리하고, 이 데이터를 이용하여 서비스 컴포넌트를 외부로 전송하거나, 또는 서비스 컴포넌트를 수신하여 배치한다. 자원 모니터(Resource Monitor-RM)은 서비스노드의 자원사용률을 감시하여 통계 데이터를 생성하고 일정 주기로 GSM에게 생성한 데이터를 전송한다.

그림 8은 프레임워크의 클래스 다이어그램을 나타낸다. 그림 7에서 컴포넌트를 구성하는 요소를 클래스로 구성하여, GSM과 RSM은 각 세 개의 클래스로 구성된다.

GSM의 NM은 실시간으로 서비스의 부하를 측정하여 서비스노드의 확장성을 계산하기 위한 accumulateData()와 노드가 부담하고 있는 부하량을 제공하기 위한 getLoad()으로 구성된다. 그리고, 서비스노드의 목록을 관리하기 위한 목적으로 노드의 추가와 삭제 그리고 노드의 목록을 반환하는 메소드로 구성된다. 반환되는 노드의 목록에 포함되는 노드는 적은 양의 부하를 부담하고 있는 노드들로 구성된다. BC에서 발송하는 메시지 수신을 위해 receiveNodeInfo()가 사용된다. NM은 노드의 확장성 저하를 감지하기 위해 checkNodeCondition()을 사용한다. 확장성이 낮아 질 때 SIP에게 통보하고 확장성 향상을 위한 절차를 수행하도록 유도 한다. LD는 사용자의 서비스 사용요청을 수신하여 서비스노드의 서비스로 전달하기 위한 메소드만을 가진다. 따라

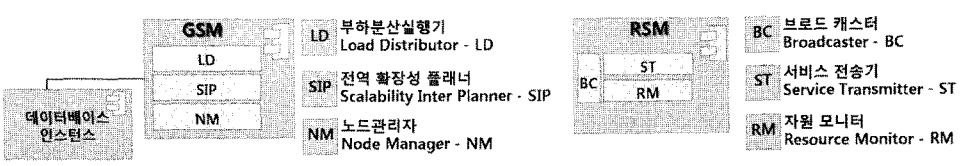

그림 7 GSM와 RSM의 컴포넌트 모형

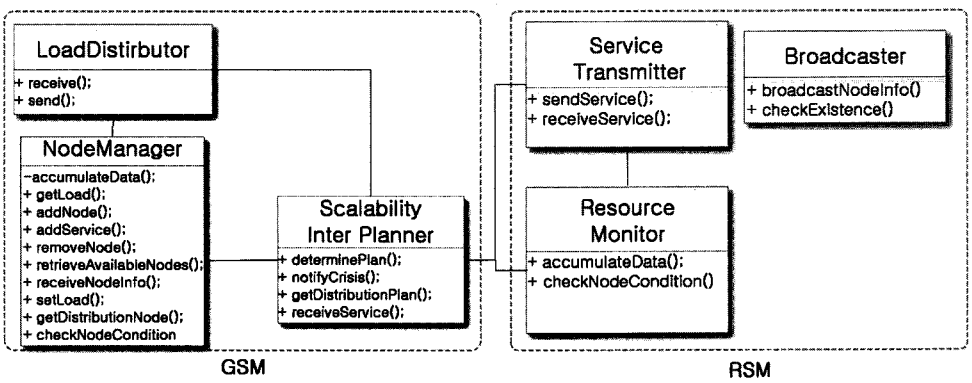

그림 8 프레임워크의 클래스다이어그램

서, receive()와 send()만을 가진다. 사용자의 요청을 중계하는 LD는 SIP가 세운 계획과 NM에서 측정한 서비스노드의 부하에 대한 통계를 바탕으로 부하분산을 실행한다.

RSM의 ST는 서비스 컴포넌트를 발송하고 수신하기 위한 메소드를 가진다. 따라서, 서비스 컴포넌트 전송을 위한 sendService()와 수신을 위한 receiveService() 메소드를 가진다. RM은 실시간으로 자원사용량에 측정하여 데이터를 축적하는 accumulateData()을 포함한다. BC는 서비스노드의 정보를 브로드캐스팅하기 위한 목적으로 사용된다. 노드의 정보만을 다루는 BC는 다른 클래스의 데이터나 기능성을 요구하지 않기 때문에 여타의 클래스와 관계를 맺지 안는다. BC는 오직 broadCastNodeInfo()만을 가진다.

그림 9는 서비스 복제를 위한 시퀀스 다이어그램을 나타낸다. SIP, NM은 GSM에 포함된 클래스이고, ST는 RSM에 포함된 클래스이다. 그러나, 두 ST는 동일한 RSM에 포함되지 않는다. Nodex: ST에서 $Node_x$는 자원부족 문제가 발생한 서비스노드를 의미하고, $Node_k$는 4장에서 제시한 알고리즘을 바탕으로 서비스 복제를 위해 SIP가 선택한 서비스노드이다. 따라서 $Node_k$는

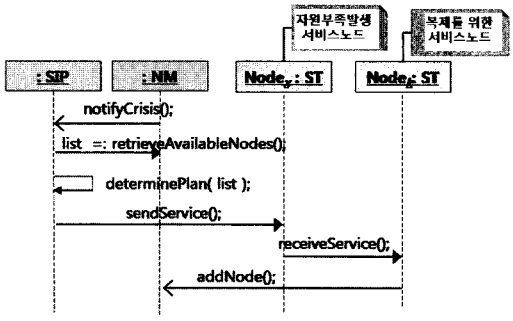

그림 9 서비스 복제 시퀀스 다이어그램

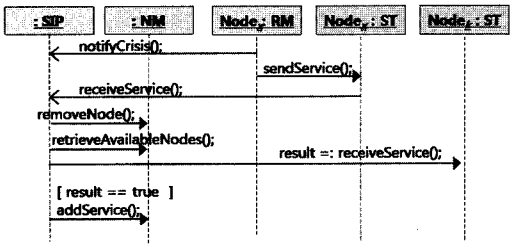

그림 10 서비스 이주 시퀀스 다이어그램

서비스 컴포넌트를 수신하여 배치하고, 부하를 분담할 노드가 된다.

그림 10은 서비스 이주를 위한 시퀀스 다이어그램을 나타낸다. $Node_x$는 문제가 발생한 노드를 나타내며, $Node_k$는 GSM에 등록이 되어 있는 노드이고 서비스 전이를 위해 충분한 자원을 보유 하고 있는 서비스가 이주할 서비스노드이다. 그림에서 retrieveAvailableNodes()를 통해 $Node_k$가 선정된다. 서비스 이주는 서비스 복제와 달리 RSM간 서비스 컴포넌트를 송/수신하지 않고, GSM이 서비스 컴포넌트를 수신하여 적절한 노드검색 후 선정된 노드에게 전달하기 때문에, SIP와 $Node_k$의 ST가 서비스 컴포넌트 전송을 위해 상호작용한다. 서비스의 배치의 결과에 따라 SIP는 NM에 서비스 정보를 갱신하거나 또는 하지 않는다.

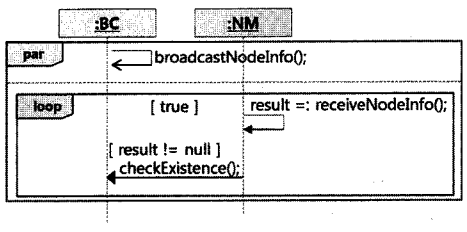

그림 11 노드등록

그림 11은 서비스노드의 RSM이 구동과 동시에 서비스노드의 정보를 네트워크에 브로드캐스팅하고, GSM의 NM이 정보를 수신하여 노드를 등록하는 절차를 나타낸다. BC는 수신자가 누군지, 수신이 성공했는지 고려하지 않고 서비스노드의 정보를 전송한다. NM은 BC가 전송하는 정보를 수신하기 위해 별도의 프로세스를 활용한다. 정보를 수신한 NM은 서비스노드를 인증하기 위해 서비스노드에게 질의를 한다. 서비스노드가 확인되면 NM은 새로운 서비스노드를 등록하거나 기존에 등록된 서비스노드 정보를 갱신한다. broadcastNodeInfo()가 노드 정보를 브로드캐스팅하며, receiveNodeInfo()가 메시지를 수신한다. 메시지 수신을 대기 하기 위해 loop를 사용하였고, loop는 checkExistence()가 수행 되더라도 프로세스가 종료되지 않고 다음 메시지수신을 위해 대기 한다.

5.2 프레임워크 구현

본 논문의 프레임워크는 Springframework와 Quartz 그리고 CXF를 활용하여 구현한다. 다양한 설정환경을 제공하는 Springframework는 소스의 수정 및 컴파일 과정을 줄여주고, 다양한 유틸리티를 제공함으로써 개발 편의를 제공한다. Quartz는 프레임워크에서 사용되는 예약작업과 반복작업등을 위해 사용된다. CXF는 웹서비스 구현에 대한 편의를 제공하여 웹서비스 개발기간을 단축시켜 주기 때문에 프레임워크를 위한 추가 라이브러리로 사용한다.

코드리스트 1은 확장성이 떨어진 노드에 대한 정보를 통보 받은 SIP가 서비스 복제를 수행하는 부분이다. 서비스를 복제하기 위한 후보노드조회가 가장먼저 수행되고, 조회된 후보 노드를 바탕으로 SIP는 부하분산에 대한 계획을 수정한다. 007번에서 SIP는 서비스의 부하의 분산에 선정된 노드를 포함한다. 013번은 서비스 복제를 명령하는 메소드이다. 메소드의 인자로 후보 노드의 정보를 함께 전달한다.

```
001: public void notifyCrisis(ServiceNodeVO serviceNode){
002:     ...
003:
004:     availableNode = nodeManagerService.retrieveAvailableNodes();
005:
006:
007:     this.determinePlan(availableNode);
008:
009:     ClientFactory factory = new ClientFactory();
010:     ...
011:     factory.setClass(ServiceTransmitterImpl.class);
012:     st = (ServiceTransmitterImpl) factory.getClient();
013:     st.sendService(availableNode, "Service1");  }
```

코드리스트 1 서비스 복제수행

코드리스트 2는 SIP의 receiveService() 메소드를 나타낸다. 서비스 컴포넌트를 수신한 SIP가 이주노드를 선정하고 해당 노드에게 서비스 컴포넌트를 전달하는 코드를 포함한다. 003번에서 SIP는 NM의 removeNode()를 호출하여 서비스관리목록에서 문제가 생긴 서비스노드를 제거한다. 005번에서 데이터베이스를 조회하여 서비스를 제공하기에 충분한 자원을 보유하고 있는 서비스노드를 선정하고 012번 줄에서 SIP는 선택된 서비스노드에게 서비스 컴포넌트를 전송하고 있다. 서비스 전송이 완료 후 SIP는 노드관리목록에 새로운 서비스노드를 등록한다. 문제가 발생한 서비스노드는 003번과 같이 제거되고, 서비스 정보는 갱신된다.

```
001: public void receiveService(ServicePackageVO servicePackage) {
002:     ...
003:     nm.removeNode(nodeVO);
004:
005:     svcNode = this.getCandidateNode( nm.retrieveAvailableNodes() );
006:
007:     ClientFactory factory = new ClientFactory();
008:     ServiceTransmitter  serviceTransmitter;
009:     serviceTransmitter = (ServiceTransmitter) factory.getClient();
010:     ...
011:
012:     if( serviceTransmitter.receiveService(servicePackage) ){
013:         nm.updateNode(svcNode);       }   }
```

코드리스트 2 서비스 이주

6. 실험 및 결과

이 장에서 수행되는 실험은 제안된 서비스 복제 및 이주 기법에 대해 다양한 실험을 적용하고, 그 결과를 기반으로 제시된 기법의 적용성과 우수성을 검증을 목적으로 한다.

실험을 위한 환경은 2.59GHz의 CPU와 2기가바이트의 메모리를 똑같이 보유한 세 개의 PC와 사용자 역할을 수행하기 위한 하나의 PC가 그림 12와 같이 구성된다. 두 개의 PC는 서비스노드 역할을 하며, 서비스컴포넌트와 RSM이 설치된다. 남은 하나의 PC는 GSM이 설치되어 부하분산노드 역할을 수행한다. 사용자 역할의 PC에는 실험을 진행하기 위한 에뮬레이터가 설치된다.

실험에 사용한 PC는 모두 Windows XP를 운영체제로 사용하며, JAVA 1.6이 설치되었다. 서비스 기반의 실험을 진행하기 위해 서비스 시스템을 구성하는 세 대의 PC에는 Apache-Tomcat 6.0이 설치되어 서비스 배치에 사용된다. 본 실험에서는 사용자로부터 외화액수와

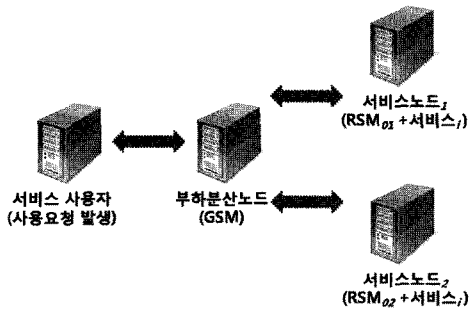

그림 12 실험을 위한 노드의 구성

대상 국가를 입력받아서 원화로 계산하여 반환하는 간단한 형태의 환율 제공 서비스를 이용한다. 서비스의 SLA에 포함된 응답시간의 임계값은 45ms로 정의한다. 에뮬레이터는 대상 국가와 외화금액을 포함한 서비스 사용 메시지를 GSM으로 전달한다. 서비스 사용자와 GSM, 그리고 GSM과 서비스간의 메시지 전송은 SOAP을 활용하고 RSM 간 메시지 전송 및 서비스 컴포넌트 전송을 위해서는 SOAP-MTOM[20]을 사용한다. 실험을 위해 사용된 서비스는 코드리스트 3의 코드로 구성된다.

```
001: @Override
002: public long exchangeRateToWon(MyService.COUNTRY_CODE countryCode,  int input) {
003:       long result = 0l;
004:       switch(countryCode){
005:             case USA:
006:                         result = (long)MyService.COUNTRY_CODE.USA.getRate() *
007:                         (long)input;
008:                         break;
009:             ....}
010:       return result;}
```

코드리스트 3. 환율 조회 프로그램

본 실험은 복제와 이주 기법에 대한 실험으로 크게 나누어지며, 각각에 대하여 다양한 서비스 부하에 대한 응답시간을 측정한다. 실험에 사용되는 용어 및 메트릭은 다음과 같다.

서비스 부하(Load): 주어진 기간 안에 발생하는 서비스 요청 개수이며, 본 실험에는 500ms의 기간을 고정하고, 서비스 요청 개수는 10개에서 1500개 사이에서는 100개의 단위로 증가하였고, 1500과 2000의 사이에서는 500개로 증가하였다. 2000개 이후에는 1000개를 증가하여 서비스의 사용요청 개수는 3000까지 증가한다.

응답시간(Response Time): 서비스 사용자가 서비스 사용을 위해 GSM에게 메시지를 전달했을 때, GSM이 메시지를 전달받은 시간과 GSM에서 서비스 사용자에게 메시지를 반환한 시간과의 차이를 나타낸다. 본 실험에서는 오차를 줄이기 위해 평균 값을 사용한다.

$$ResponseTime = ProcessingTime + WaitingTime + Replication\ (or\ Migration)\ Time + LoadDistributionTime$$

$$AverageResponseTime = \frac{\sum_{i=1}^{n} ResponseTime_i}{n}$$

이처럼 정의된 응답시간에는 요청된 서비스를 실제 처리(Processing)하는데 걸리는 시간과 서비스 대기 큐(Waiting Queue)에서 대기하는 시간 및 큐 처리에 따른 시간적 오버헤드도 포함된다. 복제의 경우, 서비스를 적합한 노드에 배정(Allocation, Forwarding)하는 오버헤드도 포함된다. 그러나, 본 실험에서는 라운드 로빈(Round Robin)이 방식을 사용하였으므로, 이 오버헤드

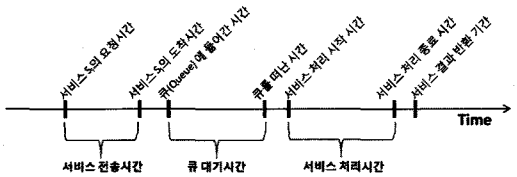

그림 13 서비스 처리 시간

는 응답시간에 거의 영향을 미치지 않는 것으로 측정되었다. 따라서, 서비스가 큐에 들어가지 않고, 바로 처리되는 경우는 응답시간이 그 서비스를 처리하는 시간과 거의 일치한다. 그림 13은 서비스의 요청에서부터 결과 반환에 이르는 과정을 나타내며 서비스의 응답시간에 포함되는 시간들을 도식화하고 있다.

사용자의 요청이 사용자로부터 서비스에 접수되기까지가 서비스 전송시간이며, 서비스에 접수된 사용자 요청이 큐에 적체된 후 처리를 위해 큐에서 나온 시간까지가 '큐 대기시간'이다. 사용자 요청이 많지 않은 시점에는 '큐 대기시간'이 무시할 수 있을 정도로 빠른 시간이 되지만, 사용자 요청이 증가할수록 한번에 처리하는 서비스처리량을 넘어서게 되어 큐에 적체되는 사용자 요청도 증가하게 된다. 큐에 사용자 요청이 적체될수록 큐에 사용자 요청이 머무는 시간이 증가하여 결국 '큐 대기시간'이 증가하게 된다. 서비스 처리에 필요한 시간 역시 '큐 대기시간'과 마찬가지로 사용자 요청이 많지 않을 때는 서비스 처리를 위해 필요한 자원이 충분하지만, 사용자 요청이 증가함에 따라 소모되는 자원도 증가하여 결국 자원의 부족으로 인해 '서비스 처리시간'이 증가 한다. 따라서, 사용자로부터 서비스에 대한 사용요청을 받아 적절한 응답을 전달하는데 걸리는 시간은 세 가지 주요 시간에 의해 결정이 된다. 본 실험에서 서비스를 부하를 과도하게 부여했을 때, 발생하는 지연 역시 같은 이유에서 발생한다.

복제 실험의 결과는 표 3과 같다. 실험은 요청량을 17회의 경우로 나누어 수행하였으며, 각 10회에서 3000회까지 실험을 수행하였다. 실험은 오차를 줄이기 위해 같은 환경에서 5회의 반복 실험을 수행하였으며, 실험을 통해 획득한 응답시간을 모두 활용하여 평균응답시간을 도출 한다. 표에서 서비스 부하는 사용자 요청을 수를 의미하고 사용자 요청은 500ms 이내에 모두 GSM에 전달된다.

표 3에서 RE-01과 RE-02에서 복제 후 응답시간이 증가하였지만, RE-16까지 SLA를 준수하였다. 표에서 RE-10에서부터 복제 이전과 이후가 차이가 확연히 나타나기 시작하여 그림 14와 같이 3000에 가까워질수록 두 선이 차이가 커짐을 확인할 수 있다.

표 3 서비스 복제 실험 결과

실험ID	서비스 부하	복제 전 평균 응답시간(ms)	복제 후 평균 응답시간(ms)	향상된 시간(ms)
RE-01	10	11.96	12.96	-1
RE-02	100	16.95	17.95	-1
RE-03	200	29.148	28.268	0.88
RE-04	300	35.4847	35.4546	0.0301
RE-05	400	36.2275	35.3452	0.8823
RE-06	500	40.47	39.291	1.179
RE-07	600	41.2457	39.321	1.9247
RE-08	700	41.5297	40.129	1.4007
RE-09	800	41.9813	40.13	1.8513
RE-10	900	43.0039	40.234	2.7699
RE-11	1000	45.9874	40.432	5.5554
RE-12	1100	45.7055	40.557	5.1485
RE-13	1200	47.8391	40.912	6.9271
RE-13	1300	46.6424	41.001	5.6414
RE-14	1400	48.6932	41.124	7.5692
RE-15	1500	49.7995	41.145	8.6545
RE-16	2000	51.3759	43.5412	7.8347
RE-17	3000	52.4973	45.3618	7.1355

표 4 서비스 이주 실험 결과

실험ID	서비스 부하	이주 전 평균 응답시간(ms)	이주 후 평균 응답시간(ms)	향상된 시간(ms)
MI-01	10	22.4	12.29	3.33
MI -02	100	26.154	14.75	3
MI -03	200	38.1	27.452	3.234
MI -04	300	39.665	34.154	4.1226
MI -05	400	41.5	37.124	5.8025
MI -06	500	47.14	40.846	8.21
MI -07	600	50.24	41.8564	9.0959
MI -08	700	54.246	41.954	9.33
MI -09	800	56.54	42.554	9.065
MI -10	900	58.154	43.5124	10.7501
MI -11	1000	59.854	44.876	9.98
MI -12	1100	60.56	45.514	12.456
MI -13	1200	59.156	47.547	11.592
MI -13	1300	61.134	47.8984	13.6032
MI -14	1400	62.354	48.8452	13.3108
MI -15	1500	64.17	48.99	13.908
MI -16	2000	66.249	51.846	11.011
MI -17	3000	67.6	52.44	10.56

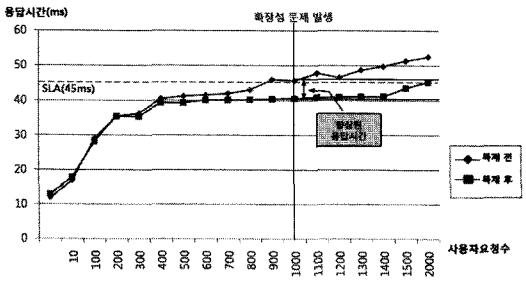

그림 14 서비스 복제 실험의 응답시간 그래프

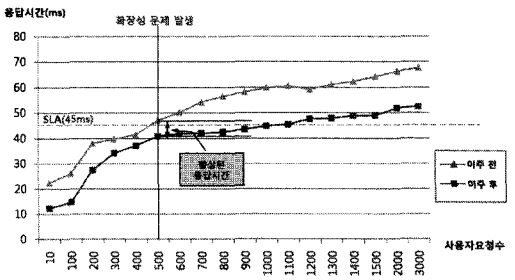

그림 15 서비스 이주 실험의 응답시간 그래프

표 4는 이주 실험의 결과를 나타낸다. 복제 실험에서와 같이 17번의 실험을 수행하였으며 사용자 요청수도 동일하게 수행하였다. 서비스 이주를 위한 실험에서 서비스는 보다 가까운 거리에 있는 서비스 노드에게로 이주하여 서비스를 제공하는 시나리오를 가진다.

표 4에서 표에서 이주 전과 후의 응답시간의 차이는 시간이 사용자 요청이 증가할수록 함께 증가하는 것을 확인할 수 있다.

그림 15에서 이주 전 응답시간을 나타내는 녹색 선은 500개의 사용자 요청이 발생하였을 때 SLA를 넘어 선다. 반면 이주를 완료한 서비스는 SLA에 명시한 응답시간보다 빠른 응답시간을 보이며 더 많은 사용자 요청을 처리하고 있다.

그림 15에서 500개의 사용자 요청이 발생하였을 때, 확장성 문제가 발생하였고 이주 기법을 적용함으로써, 응답시간을 SLA에 명시한 값 이하로 감소 시키고 있음을 나타내고 있다.

결론적으로, 서비스는 복제 기법을 적용 후 서비스 부하가 표 3의 RE-11에서와 같이 1000일 때, 13%의 응답시간 향상을 가져 왔고, 이주 기법을 적용 후 표 4의 MI-06에서와 같이 부하가 500일 때, 11%의 응답시간 향상을 가져왔다.

7. 평가 및 결론

제안된 프레임워크는 다수의 관련 연구로부터 추출된 특징에 기준하여 표 5와 같이 평가표에 적용하여 비교 평가 된다. 표 5와 같이 각 논문에서 제시한 방법이 소프트웨어적인 기법인지를 확인하기 위한 '소프트웨어 기법', 서비스제공을 위한 서비스노드를 동적으로 추가 또는 제거가 가능한 기법을 평가하기 위한 '동적 노드 추가/제거', 서비스 제공을 위해 참여 중인 서비스노드들에서 서비스를 동적으로 배치 또는 제거함으로써 서비스

표 5 관련연구와 비교평가

연구 / 평가기준	Wang [3]	Aversa [4]	Chawathe [5]	제안된 기법
소프트웨어 기법	○	○	○	○
동적 노드 추가/제거	×	×	×	○
동적 자원 구성	×	×	△	○
동적 부하분산	○	○	○	○
결함회피	×	×	○	○

를 위해 사용하는 자원을 동적으로 구성하는지 평가하기 위한 '동적 자원 구성', 변화하는 자원의 구성에 따라 또는 환경의 변화에 따라 서비스에 대한 부하를 동적으로 분산하는지 평가하기 위한 '동적 부하분산', 기법이 결함회피에 대한 기능을 포함하고 있는지 평가하기 위한 '결함 회피'를 비교평가하기 위한 평가 기준으로 제시한다. 평가는 제시된 기준에 각 연구의 적합여부를 판단하여 Support(O), Semi-Support(△), Not-Support(X)의 세 단계로 구분지어 수행한다.

제안된 기법을 포함해서 관련연구는 모두 소프트웨어적인 기법을 제안하고 있다 따라서, 모든 연구는 'O'으로 평가 된다. 본 논문을 제외한 관련 연구들은 정적인 노드를 활용하고, 정적으로 구성된 자원을 활용하고 있기 때문에 '동적 노드 추가/제거'와 '동적 자원 구성'항목에서 Wang의 연구와 Aversa의 연구는 'X'로 평가된다. Chawathe의 연구에서는 제한된 노드 환경에서 동적 자원할당을 시도하고 있기 때문에 '△'으로 평가된다. 관련 연구와 제안된 기법모두가 동적인 부하분산 기법을 포함하고 있기 때문에 '동적 부하분산' 항목에서는 모두 'O'으로 평가된다. '결함회피' 항목에서는 Chawathe의 연구와 본 논문에서 제안하는 이주 기법이 이를 다루고 있기 때문에 두 연구에 대해서만 'O'으로 평가 한다.

서비스 확장성은 서비스 제공자가 서비스수준협약에 명시된 품질을 보장하면서 서비스를 제공할 수 있는 처리량이다. 하지만 서비스는 한정된 자원에 따른 제한된 서비스 처리량을 가진다. 그리드 컴퓨팅이나 분산 컴퓨팅분야에서는 이와 같은 노드의 자원제약을 해결하기 위해 다수의 노드를 활용하는 자원공유 방식으로 시스템의 처리능력을 높이는 많은 연구가 수행되고 있다.

본 논문에서는 실용적인 서비스 확장성 관리가 가능한 프레임워크를 제안하고 있다. 서비스 확장성 프레임워크는 그리드 컴퓨팅이나 분산 컴퓨팅에서 노드의 동적 관리에 초점을 두고 있는 반면, 공유된 자원을 보다 효율적으로 이용하기 위한 동적 서비스 관리 기법을 정의한다. 또한 서비스의 확장성 저하를 인식하고, 이를 향상시키기 위해 네 단계의 절차와 각 절차를 지원하기 위한 주요 기능을 정의하고, 정의된 주요기능을 기반으로 하는 프레임워크를 제안하고 있다. 그리고 확장성 프레임워크를 설계, 구현하여 서비스의 복제와 이주에 관한 사례 연구를 통해 서비스 확장성 프레임워크의 모델 및 알고리즘의 구현 가능성 및 유효성을 보여 준다.

참 고 문 헌

[1] Hill, M. D., "What is scalabilty?," *ACM SIGARCH Computer Architecture News*, vol.18, Issue 4, pp. 18-21, Dec. 1990.

[2] Rana, O. F. and Stout, K., "What is Scalability in Multi-Agent Systems?," *In proceedings of the fourth international conference on Autonomous agents(AGENTS '00)*, pp.56-63, Barcelona, Spain, June 3-7, 2000.

[3] Wang, J., Wu, Q., Zheng, D., and Jia, Y., "Agent based Load Balancing Model for Service based Grid Applications," *In proceedings of Computational Intelligence and Security (CIS'2006)*, pp487-491, Guangzhou, China, Nov. 3-6, 2006.

[4] Aversa, R., Martino, B. D., Donini, R., and Venticinque, S., "Load Balancing of Mobile Agents Based Applications in Grid Systems," *In proceedings of the 17th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, pp.6-11, Rome, Italy, June 23-25, 2008.

[5] Chawathe, Y. and Brewer, E. A., "System Support for Scalable and Fault Tolerant Internet Services," *In proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp.71-88, London, UK, Sep. 1-1, 2009.

[6] Xue, W., Huai, J., and Liu,Y., "Access control policy negotiation for remote hot-deployed grid services," *In proceedings of First International Conference on e-Science and Grid Computing 2005 (e-Science'05)*, pp.378-386, Melbourne, Australia, Dec. 5-8, 2005.

[7] Ward, C., Buco, M.J., Chang, R.N., Luan, L.Z., So, E., and Tang, C., "Fresco: a Web services based framework for configuring extensible SLA management systems," *In proceedings of 2005 IEEE International Conference on Web Services(ICWS 2005)*, pp.237-245, Florida, USA, July 11-15, 2005.

[8] Software Engineering-*Product Quality-Part 3: Quality in Use. ISO/IEC TR 9126-4*, July, 2003.

[9] Friedman, R. and Mosset, D., "Load Balancing Schemes for High-Throughput Distributed Fault-Tolerant Servers," *In proceedings the sixteenth symposium on reliable distributed systems(SRDS'97)*, pp.107-115, North Carolina, USA, Oct. 22-24, 1997.

[10] Jogalekar, P. P. and Woodside, C. C., "Evaluating the Scalability of Distributed System," *In Proceedings of the 31st Hawaii International Conference System Science*, vol.7, pp.524-524, Jan. 1998.

[11] Jogalekar, P. P. and Woodside, C. C., "Evaluating the Scalability of Distributed System," *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, vol.11, no.6, June 2000.

[12] Younas, M., Chao, K. M., Griffiths, N., Anane, R., and Awan, I., "Quality driven Web services in mobile computing," *In proceedings 24th International Conference on Distributed Computing Systems Workshops(ICDCS 2009)*, pp.216-221, Québec, Canada, March 23-24, 2004.

[13] Bondi, A. B., "Characteristics of Scalability and Their Impact on Performance," *In proceedings of the 2nd international workshop on Software and performance(WOSP2000)*, pp.195-203, Ottawa, Canada, September 17-20, 2000.

[14] Vogler, C. and Metaxas, D., "Toward Scalability in ASL Recognition: Breaking Down Signs into Phonemes," *In proceedings of the International Gesture Workshop on Gesture-Based Communication in Human-Computer Interaction(GW'99)*, Lecture Notes In Computer Science; vol.1739, pp. 211-224, Gif-sur-Yvette, France, March 17-19, 1999.

[15] Carzaniga, A., Rosenblum, D. S., and Wolf, A. L., "Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service," *In proceedings of the nineteenth annual ACM symposium on Principles of distributed computing (PODC 2000)*, pp.219-227, Oregon, United States, July 16-19, 2000.

[16] World Wide Web Consortium(W3C) Recommendation, *Simple Object Access Protocol(SOAP)*, Version 1.2, World Wide Web Consortium(W3C), Apr. 27, 2007, http://www.w3.org/TR/soap12-part1/ (accessed Mar. 13, 2010).

[17] Jogalekar, P. and Woodside, M., "Evaluating the Scalability of Distributed Systems," *IEEE transactions on parallel and distributed systems*, vol.11, no.6, June 2000.

[18] Allison, C., Harrington, P., Huang, F., and Livesey, M., "Scalable Services for Resource Management in Distributed and Networked Environments," *In Proceedings of the 3rd Workshop on Services in Distributed and Networked Environments (SDNE '96)*, pp.98-105, RSMcau, China, June 3-4, 1996.

[19] Vadhiyar, S. S., Fagg, G. E., and Dongarra, J., "Automatically Tuned Collective Communications," *In proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 3-3, Texas, USA, Nov. 04-10, 2000.

[20] World Wide Web Consortium (W3C) Recommendation, SOAP Message Transmission Optimization Mechanism, Latest version, World Wide Web Consortium (W3C), Jan. 25, 2005, http://www.w3.org/TR/soap12-mtom/ (accessed Mar. 13, 2010).

김 지 원

2004년 영동대학교 컴퓨터학과 학사. 2008년~현재 숭실대학교 컴퓨터학과 석사과정. 관심분야는 객체지향 S/W공학, 서비스 지향 아키텍처(SOA), 클라우드 컴퓨팅(Cloud Computing)

이 재 유

2007년 홍익대학교 컴퓨터정보통신학과 공학사. 2009년 숭실대학교 컴퓨터학과 공학석사. 2009년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 서비스 지향 아키텍처(SOA), 자율 컴퓨팅(AC)

김 수 동

1984년 Northeast Missouri State University 전산학 학사. 1988년/1991년 The University of Iowa 전산학 석사/박사. 1991년~1993년 한국통신 연구개발단 선임연구원. 1994년~1995년 현대전자 소프트웨어연구소 책임연구원. 1995년 9월~현재 숭실대학교 컴퓨터학부 교수. 관심분야는 서비스 지향 아키텍처(SOA), 클라우드 컴퓨팅(Cloud Computing), 모바일 서비스(Mobile Service), 객체지향 S/W공학, 컴포넌트 기반 개발(CBD), 소프트웨어 아키텍처