韓國IT서비스學會誌
第13卷 第3號
2014年 9月, pp.201-219

Journal of Information Technology Services
http://dx.doi.org/10.9716/KITS.2014.13.3.201

클라우드에서 서비스 지향 응용을 위한 최적 서비스 배치와 우선순위 결정 기법*

김길환** · 금창섭*** · 배현주***

Service Deployment and Priority Optimization for Multiple Service-Oriented Applications in the Cloud*

Kilhwan Kim** · Changsup Keum*** · Hyun Joo Bae***

■ Abstract ■

This paper considers service deployment and priority optimization for multiple service-oriented applications sharing reusable services, which are deployed as multiple instances in the cloud. In order to handle variations in the workloads of the multiple applications, service instances of the individual reusable services are dynamically provisioned in the cloud. Also service priorities for each application in a particular reusable service are dynamically adjusted. In this paper, we propose an analytic performance model, based on a queueing network model, to predict the expected sojourn times of multiple service-oriented applications, given the number of service instances and priority disciplines in individual reusable services. We also propose a simple heuristic algorithm to search an optimal number of service instances in the cloud and service priority disciplines for each application in individual reusable services. A numerical example is also presented to demonstrate the applicability of the proposed performance model and algorithm to the proposed optimal decision problem.

Keyword : Performance Analysis, Service-Oriented Application, Cloud Computing, Service Priority Discipline, Queueing Network

Submitted : January 24, 2014 1st Revision : September 8, 2014 Accepted : September 12, 2014

* 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발 사업의 일환으로 수행하였음 (09-911-05-001, 네트워크 기반 수요자 지향 융합서비스 공통플랫폼 기술개발).

** 상명대학교 경영공학과
*** 한국전자통신연구원 통신인터넷연구소

1. 서 론

서비스 지향 아키텍처(Service-Oriented Architecture; SOA)는 기존 서비스를 재사용하여 새로운 응용을 신속하게 구축할 수 있는 환경을 제공한다(Erl, 2004; Huhns and Singh, 2005; Papazoglou et al., 2007). 이러한 장점으로 최근에 많은 관련 연구들이 수행되어 왔다. SOA 환경에서 서비스 재사용 가능하므로 여러 응용에 의해 동시적으로 이용될 수 있다. 그리고 재사용 가능 서비스의 작업부하는 서비스를 이용하고 있는 기존 응용의 사용량의 변화뿐 아니라 서비스의 설계 당시에는 고려하지 못했던 새로운 응용의 출현 및 기존 응용의 정지, 재시작, 제거에 의해서도 크게 영향을 받게 된다. 따라서 SOA 환경에서 재사용 가능 서비스의 작업부하 변화에 신속하게 대처할 수 있는 적절한 메커니즘이 요구된다.

한편 클라우드 컴퓨팅 기술의 발전은 가용한 컴퓨팅 자원의 규모를 쉽게 늘이거나 줄일 수 있도록 하므로(Armbrust et al., 2010; Buyya et al., 2009), 작업부하가 크게 변동하는 재사용 가능 서비스의 실행 환경으로 클라우드는 매우 매력적인 대안이 된다(Huang et al., 2012; Espadas et al., 2013). 이런 이유로 클라우드의 사용자 수용성에 대한 연구들이 최근에 많이 수행되었다(Park and Kim, 2013; Ryu et al., 2013).

클라우드에서 재사용 가능 서비스는 하나 이상의 인스턴스로 배치될 수 있고, 개별 서비스의 작업부하의 변동에 따라 서비스 인스턴스가 적재된 VM(virtual machine)의 수는 동적으로 조정될 수 있다. 또한 현재 클라우드에 새로운 인스턴스를 프로비저닝하는 것은 수 분만에 완료될 수 있다 있다(Dejun et al., 2010). 따라서 클라우드 환경에서 재사용 가능 서비스의 성능 관리에서 핵심적인 문제는 서비스 인스턴스를 동적으로 조정하고 배치하는 실행 메커니즘보다는 작업부하의 변화에 따라 신속하게 필요한 서비스 인스턴스의 수, 즉 서비스의 처리 용량을 최적으로 결정하는 의사결정 메커니즘이다.

한편 SOA에서 다수의 응용은 하나 이상의 재사용 가능 서비스를 공유한다. 그러므로 한 응용의 성능은 자신이 이용하고 있는 서비스의 처리 용량뿐 아니라, 같은 서비스를 이용하는 다른 응용의 사용량에 의해서도 영향을 받게 된다. 대부분의 경우 응용은 자체적으로 목표하는 성능 목표나 SLA (Service Level Agreement)가 있고, 이러한 성능 목표나 SLA는 응용 별로 상이하다. 응용 간의 이러한 성능의 상호 의존성과 성능 목표의 차별성은, 재사용 가능 서비스의 성능 관리 메커니즘에 응용별 서비스 우선순위를 고려할 필요를 말해 준다. 예를 들어 엄격한 성능 목표 수준을 가진 응용과 그보다 느슨한 성능 목표 수준을 가진 응용이 병목 현상이 발생하고 있는 서비스를 동시에 이용한다고 가정해 보자. 두 응용에 대한 서비스 우선순위가 없다면 병목 현상이 발생한 서비스의 처리 용량을 늘려야만(서비스 인스턴스를 더 많이 프로비저닝해야만) 두 응용 모두 성능 목표 수준으로 만족할 수 있다. 하지만 병목 현상이 발생한 서비스에서 엄격한 성능 목표 수준을 가진 응용에 서비스 우선순위를 부여한다면, 적은 서비스 처리 용량만으로 두 응용의 성능 목표 수준을 만족시킬 수 있다.

그러므로 SOA 환경의 재사용 가능 서비스를 사용하는 응용들의 서비스 사용량의 변화와 다양한 성능 목표치를 고려하여, 클라우드 환경에서 서비스 별 최적 서비스 용량과 우선순위 정책을 결정하는 의사결정 메커니즘이 필요하다. 이를 위해서는 서비스별 인스턴스 수와 우선순위의 결정에 따라 응용의 성능이 어떻게 변화할지 분석할 수 있어야 한다. 그것도 클라우드가 가진 수 분 내의 실행 환경 변화 능력을 고려할 때 매우 짧은 시간 안에 결과를 줄 수 있는 성능 분석 모델이 필요하다.

본 논문에서는 SOA 환경에서 다수의 응용이 재사용 가능 서비스를 공유하고, 재사용 가능 서비스는 클라우드에 배치되는 경우에, 최적의 서비스 별 인스턴스의 수와 서비스 우선순위 정책을 결정

할 수 있는 대기행렬 네트워크(queueing networks)를 이용한 성능 분석 모델과 최적해를 탐색하는 단순한 휴리스틱 알고리즘을 제안한다.

논문의 구성은 다음과 같다. 제 2장에서는 관련 연구를 살펴보고, 제 3장에서는 논문에서 다루고자 하는 시스템을 소개하고 문제를 정식화한다. 제 4장에서는 목표 문제를 해결하기 위한 성능 분석 모델을 수립한다. 제 5장에서는 성능 분석 모델을 이용해 최적해를 찾는 단순한 휴리스틱 알고리즘을 제안하고, 제 6장에서는 제시한 성능 분석 모델과 알고리즘의 응용 가능성을 보이기 위해 수치 예에 대한 분석을 수행한다. 마지막으로 제 7장에서 논문에 대한 의의와 한계를 제시한다.

2. 관련 연구

지금까지 SOA 환경에서 응용의 성능 분석 방안에 대해서 다수의 연구가 수행되어 왔다(Gao et al., 2005; Kim, 2013a, 2013b; Liu, Gorton and Zhu, 2007; Teixeira et al., 2009; Wada et al., 2009; Yuan and Ji, 2007). Brebner et al.(2009)는 SOA 환경에서 성능 분석 모델을 신속히 할 수 있는 방법과 툴을 제안하였고 이를 시뮬레이션 하는 방법을 제시하였다. Brebner(2009)는 EMUL ESB를 이용하여 성능 분석 모델을 제시하였으며 성능 분석의 방법은 시뮬레이션을 기반으로 하고 있다. Chudov et al.(2011)은 Siemens 사례를 통해 SOA 기반 시스템의 시뮬레이션 기반 성능 분석 방법을 제시하였다. Teixeira et al.(2009)은 확률적 Petri nets을 기반으로 서비스 기반 응용의 성능을 분석하였으며, Wada et al.(2009)은 M/D/N 대기행렬 모델을 이용하여 클라우드 환경에서의 서비스 배치를 최적화하는 방안을 제시하였다. Yuan and Ji (2007)는 SOA의 성능 분석을 위해 다중 요인 기반의 민감도 분석을 위한 통계적 접근을 제안하였다. Liu et al.(2007)은 대기행렬 네트워크를 이용하여 ESB(Enterprise Service Bus) 상에서 조합된 서비스 기반 응용의 성능을 분석하고 예측하는 모델을 제시하였다. Gao et al.(2005)은 마아코프 체인을 이용하여 분석적인 성능 분석 모델을 구성하였으나 최적의 서비스 조합 방식을 결정하는 의사결정 문제에 이를 응용하였다. Kim(2013b)은 서비스의 인스턴스가 오직 한 개인 경우에 다수의 응용이 있는 경우의 우선순위 정책이 응용의 성능에 미치는 영향을 분석할 수 있는 모델을 제시하였다. 반면 Kim(2013a)은 클라우드 환경에서 다수의 서비스 인스턴스를 배치할 수 있는 경우의 서비스 지향 구조에서 응용의 성능 분석 모델을 제시하였다.

하지만 이러한 연구의 대부분은 개별 서비스에서 응용에 대한 우선순위를 부여할 수 있는 경우를 고려하지는 않았으며, 대부분 시뮬레이션 기반의 성능 분석 모델을 이용하였다(Brebner, 2009; Brebner et al., 2009; Chudov et al., 2011; Teixeira et al., 2009) 그런데 제 4장에서 볼 수 있듯이 최적 서비스 별 인스턴스와 우선순위 조합의 해를 탐색하기 위해서는 수많은 반복적인 성능 분석 모델의 실행이 필요하다. 따라서 짧은 시간 내에 의사결정이 필요한 클라우드 환경에서는 시뮬레이션을 기반으로 한 모델은 한계를 가진다.

성능 분석을 신속하게 수행할 수 있는 수리적 모델을 이용한 연구에서도, 다수 응용이 아닌 단일 응용만을 고려하여 다수 응용 간의 성능 의존성을 고려하지 않거나(Wada et al., 2009), 서비스 인스턴스 수가 하나만 고려된 성능 분석 모델이어서 클라우드에서 동적인 프로비저닝 결정에 활용하기에는 한계를 갖거나(Gao et al., 2005; Kim, 2013b; Liu et al., 2007), 다수 서비스 인스턴스 프로비저닝을 고려하였으나 우선순위를 고려하지 않거나 최적해를 찾는 알고리즘에 대한 논의가 없었다(Kim, 2013a; Liu et al., 2007).

Yuan and Ji(2007)의 연구는 수리적인 통계적 기법을 이용하여 성능 예측 모델을 만들었는데 환경이 변화하면 새로운 성능 데이터를 다시 수집하여 통계 모델을 재구축해야 하므로 환경 변화에 따른 성능을 선제적으로 예측하고 의사결정할 수 없는 한계를 가진다.

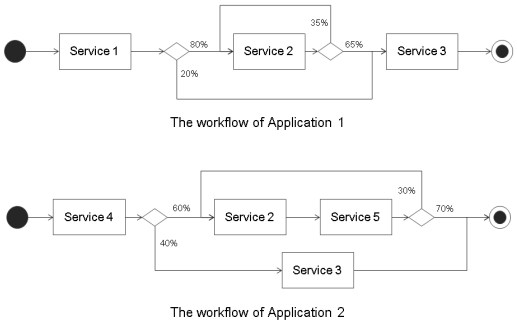
본 연구는 동적으로 다수의 서비스 인스턴스를 프로비저닝 할 수 있고 서비스 우선순위를 조정할 수 있는 시스템의 수리적 성능 분석 모델을 제시하고 최적해를 찾는 휴리스틱 알고리즘을 제시하였다는 데 기존 연구와 차별성이 있다.

3. 클라우드 환경에서 재사용 가능 서비스의 배치 및 서비스 우선순위 부여 문제

본 장에서는 본 논문에서 고려하고 있는 시스템과 해결하고자 하는 성능 분석 및 최적화 문제를 기술한다.

3.1 클라우드 환경에서의 서비스 기반 시스템

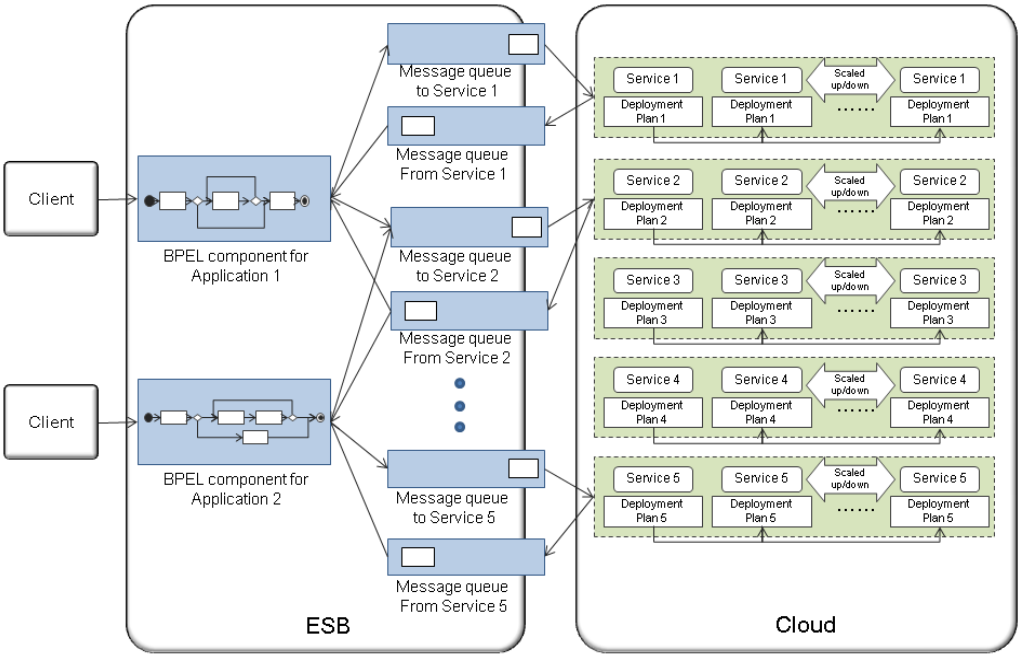
SOA 환경에서 다수의 응용으로 구성된 전체 시스템을 고려해 보자. 한 응용은 다수의 기능으로 분할되며, 분할된 개별 기능은 여러 응용에 의해 공통적으로 사용된다. 예를 들어 화상회의 응용, 동영상 스트리밍 응용, 원격진료 응용으로 구성된 시스템을 생각해 보자. 이 경우 과금과 지불 기능은 모든 응용에 의해 공통적으로 사용될 수 있다. 각각의 기능은 SOA 패러다임에 따라 독립적인 재사용 가능 서비스로 구현되며, SOAP이나 REST 방식의 웹 서비스(Pautasso et al., 2008)로 구현될 수 있다. 그리고 각 응용은 <Figure 1>과 같이 다수의 재사용 가능 서비스로 구성된 작업흐름(workflow)로 표현되며, 작업흐름은 BPEL(OASIS, 2007) 등으로 구현될 수 있다. 한 응용의 작업흐름의 실행 될 때 동일한 서비스가 여러 번 호출될 수 있다. 예를 들어 <Figure 1>에서는 응용 1과 응용 2는 서비스 2를 한번 이상 반복적으로 호출할 수 있다. 또한 응용 1과 응용 2는 서비스 2와 서비스 3을 공유하고 있음을 볼 수 있다. 따라서 어느 한 재사용 가능 서비스에 도착하는 작업량은 그 서비스를 이용하는 응용에 대한 최종 사용자의 사용량과 응용의 작업흐름에 의해 결정된다.



〈Figure 1〉 Typical Examples of the Workflows of Service-Oriented Applications(Adapted from Kim(2013b))

이제 클라우드에 재사용 가능 서비스를 배치하는 구조를 살펴보자. 재사용 가능 서비스는 클라우드 안에서 하나 이상의 서비스 인스턴스로 프로비저닝될 수 있다. 본 논문에서는 서비스 인스턴스 간의 처리용량의 독립성을 위해 하나의 VM(virtual machine)엔 한 개의 서비스 인스턴스 만이 배치되고 동일한 재사용 가능 서비스를 위한 모든 서비스 인스턴스는 동일한 배치 정책(deployment plan)을 가진 VM에 배치된다고 가정한다(<Figure 2> 참조). 그리고 클라우드는 서비스에 대한 작업부하량의 변동에 따라 각 서비스의 인스턴스를 동적으로 프로비저닝하고 제거할 수 있다.

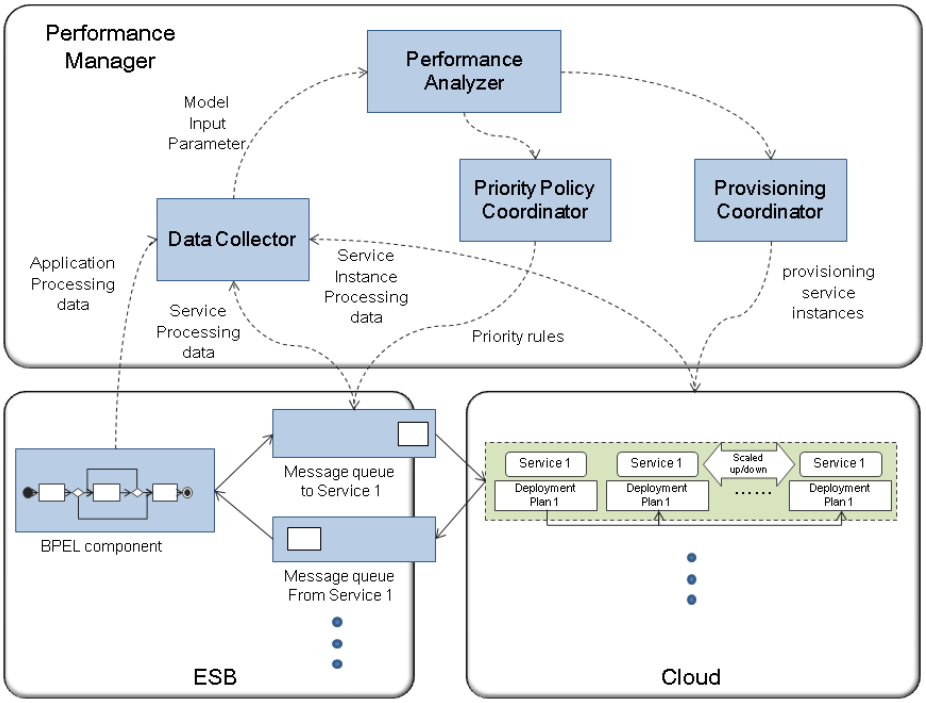
동일한 서비스에 대해 다수의 인스턴스가 존재하므로, 서비스 인스턴스 간의 부하를 분배하기 위해 서비스에 대한 작업요청을 적절히 라우팅하는 방법이 필요하다. 본 연구에서는 작업요청의 라우팅을 구현하는 방법으로 ESB(enterprise service bus)를 이용하여 서비스 작업요청에 대한 메시지 큐를 이용하는 모델을 상정한다(<Figure 2> 참조). ESB에서 BPEL 등을 이용하여 개별 응용별로 작업흐름을 구현 한다. BPEL에서 개별 서비스를 부를 때는 ESB의 메시지 큐를 이용한다. 메시지 큐는 JMS(Java messaging service)(Richards et al., 2009)등의 기술을 이용할 수 있다. 서비스에 대한 작업요청은 먼저 ESB에 있는 해당 서비스를 위한 메시지 큐에 라우팅되어 대기한다. 클라우드에 배



〈Figure 2〉 Deployment and Priority-Based Routing of Service Instances in the Cloud

치된 해당 서비스의 서비스 인스턴스들은 유휴 상태가 되면 주기적으로 메시지 큐에 연결하여 서비스에 대한 작업요청을 가져온다. 이 때 메시지 큐는 선입선출 방식으로 작업요청을 처리하는 것이 아니라, 서비스별로 설정된 응용별 작업 우선순위 정책에 따라 작업요청을 서비스 인스턴스에 분배한다. 예를 들어 응용 2가 응용 1보다 서비스 1에서의 작업 우선순위가 높으면 응용 2의 작업이 응용 1보다 먼저 서비스 인스턴스에 배분된다. 서비스 인스턴스는 해당 작업요청을 처리하고 나면 해당 서비스에서 BPEL로 향하는 메시지 큐에 처리 결과를 전달한다.

현실에서 응용의 수요 변동에 따라 서비스별 최적의 인스턴스 수를 결정하고 응용별 작업 우선순위를 동적으로 부여하기 위해서는, 실행환경에서 이러한 최적 운영 정책을 결정하고 적용시킬 수 있는 메커니즘이 필요하다. <Figure 3>은 이러한 의사결정 및 적용 메커니즘의 구현 가능 예로 성능 관리자(Performance Manager)라는 구조를 예시하고 있다. 성능 관리자는 <Figure 3>과 같이 데이터 수집자, 성능 분석자, 우선순위 조정자, 프로비저닝 조정자 등의 형태로 구현될 수 있다. 데이터 수집자는 BPEL에서 응용에 대한 사용자 요청과 처리 시간에 대한 데이터를, 메시지 큐에서 서비스에 대한 요청과 처리 시간에 대한 데이터를, 그리고 클라우드에서 서비스 인스턴스의 상태에 대한 데이터를 수집한다. 그리고 수집한 데이터를 기반으로 성능 분석 모델의 매개변수를 생성하여 성능 분석자에 전달한다. 성능 분석자는 이 논문의 제 3장의 성능 분석 모델을 이용하여 서비스 인스턴스 수와 우선순위 정책에 따른 응용별 성능을 예측한다. 그리고 예측된 성능 정보로 제 4장에 제시된 알고리즘으로 서비스별로 최적 인스턴스 수와 응용별 작업 우선순위를 결정하여 우선순위 조정자와 프로비저닝 조정자에게 결과를 전달한다. 그러면 우선순위 조정자는 ESB의 대기열에서 우선순위를 동적으로 조절하고, 프로비저닝 조정자는 클라우드의 서비스 인스턴스 수를 동적으로 조정한다.



〈Figure 3〉 An Architecture of Performance Manager

3.2 재사용 서비스의 최적 배치 및 서비스 우선순위 부여 문제

최적의 서비스 인스턴스 수와 우선순위 정책을 결정하기 위해서 먼저 시스템의 비용 함수가 결정되어 있어야 한다. 본 논문에서는 비용이 응용 작업 처리시간이 길어져서 발생하는 사용자 불만족 비용과, 서비스 인스턴스를 유지하는 데 발생하는 시스템 유지 비용만을 고려한다. 두 비용은 서로 절충관계에 놓여 있다. 응용 작업 처리 시간을 줄이려면 많은 서비스 인스턴스를 유지해야 하고, 서비스 인스턴스 수를 줄이면 응용 작업 처리 시간이 늘어나게 된다.

앞서 설명하였듯이 응용은 사용자 요청에 대하여 목표한 성능 수준에서 사용자 요청을 처리해야 하고, 각 응용이 만족시켜야 하는 목표 수준은 서로 다르다. 예로서 동영상 스트리밍 응용에 비해 화상 회의 응용은 더 엄격한 처리시간에 대한 목표 수준을 요구할 수 있다. 이러한 환경에서 여러 응용에 대한 사용자 요구가 폭증하여 처리시간이 길어지고 있다면 병목이 되고 있는 서비스들은 동적으로 인스턴스 수를 조정하여 응용들의 처리시간을 전체적으로 감소시킬 수 있다. 그런데 만약 어떤 응용은 목표한 처리시간이 짧아서 달성이 위협받고 있고, 다른 응용은 상대적으로 목표 수준 만족에 여유가 있다면 병목이 되는 서비스의 인스턴스 수를 전체적으로 증가시키기 보다는 병목 서비스에서 성능 목표 수준 만족이 위협받고 있는 응용에게 작업 우선순위를 부여하는 것이 더 효과적일 것이다. 반대로, 응용에 대한 사용자 요청이 전반적으로 줄어들고 있다면 처리 용량이 여유가 있는 서비스가 생기므로, 그런 서비스의 인스턴스 수를 줄여서 운영 비용을 절감할 수 있다. 그런데 사용자 요청이 응용 별로 비대칭적으로 감소하였다면 여유 서비스에 대한 인스턴스 수의 조정과 함께 적절한 응용별 작업 우선순위의 재조정 한다

면 더 많은 수의 서비스 인스턴스 수의 감소가 가능할 것이며 이는 직접적인 운영비용 절감 효과를 발생시킬 것이다

본 예제에서는 비용 함수 T가 평균 사용자 요청 처리 시간과 서비스 인스턴스 수에 대한 다음과 같은 선형 함수로 정의된다고 가정한다.

$$T(W^1, \cdots, W^m; c_1, \cdots, c_J) = \sum_{m=1}^{M} u_m \lambda^m W^m + \sum_{j=1}^{J} h_j c_j \quad (1)$$

단, u_m, $m=1, \cdots, M$은 응용 m의 한 작업의 처리 시간이 한 단위 시간이 늘어날 때 추가로 발생하는 사용자 불만 비용이고, h_j, $j=1, \cdots, J$는 서비스 j의 서비스 인스턴스가 하나 늘어날 때 추가로 발생하는 단위 시간당 발생하는 시스템 유지 비용이다. 사용자 불만 비용에 단위 시간당 도착율 λ^m이 곱해져 있으므로 비용 함수 T는 단위 시간당 발생하는 평균 비용이 된다. 결국 최적 시스템 운영 정책을 결정하는 문제는 비용함수 T를 최소화하는 개별 서비스의 서비스 인스턴스 수 c_j와 우선순위 결정 함수 $\delta_j(m)$를 결정하는 다음과 같은 최적화 문제가 된다.

$$\min_{c_1, \cdots, , c_J; \delta_1, \cdots, \delta_J} \sum_{m=1}^{M} u_m \lambda^m W^m + \sum_{j=1}^{J} h_j c_j \quad (2)$$

식 (2)의 최적화 문제를 해결하기 위해서는 먼저 개별 서비스의 서비스 인스턴스 수 c_j와 우선순위 결정 함수 $\delta_j(m)$가 정해져 있을 때의 각 응용별 평균 처리 시간 $W^1, \cdots, W^m$을 계산할 수 있어야 한다. 그러기 위해서는 응용 간에 서로 공유하는 서비스에 의한 발생하는 처리 성능의 의존관계를 적절히 고려하여 분석할 수 있는 성능 분석 모델의 수립이 필요하다. 제 3장에서는 이러한 성능 분석을 수행할 수 있는 모델을 제시한다.

4. 성능 분석 모델

본 장에서는 제 3장에서 설명한 클라우드 환경에서 우선순위가 있는 다수의 서비스 기반 응용의 평균 처리 시간을 계산해 낼 수 있는 성능 분석 모델을 수립한다. 본 성능 모델에서는 응용별 사용자 요청의 도착 과정(arrival process)과 사용자 요청이 응용의 작업흐름에 따라 각 서비스에서 처리되는 시간의 분포는 일정 기간 동안 유지되며, <Figure 3>과 같은 시스템의 측정 메커니즘에 의해 성능 분석 모델에 적시에 입력된다고 가정한다.

본 논문에서는 성능 분석 모델을 위해 다수 서버와 "우선순위를 가진" 개방형 잭슨 대기행렬 네트워크를 이용한다(잭슨 대기행렬 모델은(Jackson, 1963; Jackson, 1957) 참조). 이러한 대기행렬 모델을 이용하는 이유는 다음 두 가지이다. 첫째, 우선순위를 가진 개방형 대기행렬 네트워크는 다수의 서비스 기반 응용의 작업흐름을 어느 정도 적절히 모델링하기 때문이다. 둘째, 개방형 잭슨 대기행렬 네트워크의 분석은 독립적인 "우선순위를 가진" M/M/c 대기행렬 모델로 환원될 수 있기 때문에 수리적 분석이 용이하기 때문이다.

다수의 서비스 기반 응용의 작업흐름을 다음과 같이 모델링한다. 클라우드 환경에 J개의 재사용 가능 서비스와 재사용 가능 서비스를 사용하는 M개의 응용이 배치되어 있다고 가정한다. 각각의 응용은 J개의 재사용 가능 서비스 중 일부를 사용하여 작업흐름을 표현된다(<Figure 1> 참조). 한 작업흐름 안에는 순차흐름, 조건에 따른 분기흐름, 반복흐름, 병렬흐름의 제어 흐름들이 있다. 수리적 분석의 용이성을 위해 본 연구에서는 병렬흐름은 모델에서 제외하기로 한다. 왜냐하면 병렬적 흐름 처리는 기본적인 대기행렬 네트워크 모형으로 표현하기 어렵기 때문이다. 병렬 흐름처리가 있을 시에는 가장 긴 처리 시간을 가지는 서비스로만 입력되는 것으로 근사하기로 한다. 병렬 흐름에 대한 좀 더 정확한 모델링 방안은 추후 연구에 의해 보강되어져야 할 사항이다.

시스템 외부에서 응용 m, $m=1, \cdots, M$으로 인입되는 작업은 매개변수 λ^m을 따르는 포아송 과정을 따른다고 가정한다. 응용 m의 작업흐름에 따라 외

부에서 인입된 응용 m 작업은 확률 q_{0j}^m, $j=1, \cdots, J$, $m=1, \cdots, M$로 서비스 j로 라우팅이 이루어진다. 외부에서 인입된 작업은 시스템을 떠나기 전에 적어도 한 개의 서비스로 라우팅된다고 가정하면 다음이 성립한다.

$$\sum_{j=1}^{J} q_{0j}^m = 1,\ m=1, \cdots, M.$$

응용 m의 작업은 서비스 j에서 작업이 완료되면 작업흐름에 따라 다른 서비스로 라우팅 되거나 서비스가 모두 완료되어 시스템 밖으로 나가데 된다. 서비스 j를 떠난 작업이 작업 흐름에 의해 서비스 k로 라우팅될 확률을 q_{jk}^m, j, $k=1, \cdots, J$, $m=1, \cdots, M$, 시스템 밖으로 라우팅될 확률을 q_{j0}^m, $j=1, \cdots, J$, $m=1, \cdots, M$라고 정의한다. 라우팅 확률은 과거 데이터에 의해 추정되고 작업의 이전 작업흐름에 무관하게 현재 작업 완료되어 떠나는 서비스 종류와 응용 종류에 의해서만 결정된다고 가정한다. 한 서비스에서 처리가 완료되어 떠난 작업은 다른 서비스 또는 시스템 밖으로 라우팅될 것이므로 다음이 성립한다.

$$\sum_{k=1}^{J} q_{jk}^m = 1,\ m=1, \cdots, M.$$

그러면 <Figure 1>과 같이 순차, 분기, 반복을 가지는 작업흐름의 라우팅 확률을 행렬의 형태로 표현된다(자세한 내용은 제 6장의 수치 예제 참조).

다음으로 재사용 가능 서비스들의 서비스 특성을 모델링한다. 재사용 가능 서비스 j가 클라우드에 배치된 서비스 인스턴스 수를 c_j라 하고, 각 서비스 인스턴스에서 하나의 작업의 처리 시간은, 응용 종류에 무관하게 처리율 μ_j를 따르는 지수분포를 따른다고 가정한다. 서비스 별로 서로 다른 응용별 작업 우선순위를 부여할 수 있고, 동일한 서비스의 대기열에 있는 응용별 작업은 응용의 우선순위가 높은 것이 먼저 서비스 되며 비축출(non-preemptive) 우선순위 정책에 따라 서비스 된다. 우선순위 정책을 수식적으로 표현하기 위해, 응용 m의 작업이 서비스 j에서 가지는 작업 우선순위를 $\delta_j(m)$로 표기하고, $\delta_j(m)$의 값이 낮을수록 작업 우선순위가 더 높다고 하자. 즉 $\delta_3(1)=2, \delta_3(2)=1$라고 하면 서비스 3에서 응용 2의 작업 우선순위가 응용 1보다 높다는 것을 의미한다. 같은 응용에 속한 작업은 FCFS(First Come First Served) 순서로 서비스 받는다.

위와 같은 가정을 만족하면 전체 시스템은 우선순위와 다수 서버를 가지는 개방형 잭슨 대기행렬로 모델링할 수 있다. 클라우드에 배치되는 각 서비스는 잭슨 대기행렬의 각 노드로 표현되며, 서비스 인스턴스는 잭슨 대기행렬 노드에 있는 다수의 서버 형태로 표현된다. 응용별 작업은 다중 클래스를 가지는 고객의 도착과정으로 표현된다 .

Λ_j^m을 응용 m의 작업이 서비스 j에 도착하는 도착율이라고 하자. 응용 m의 작업은 시스템 바깥에서 서비스 j로 바로 인입될 수도 있고, 다른 서비스에서 완료되니 작업이 작업흐름에 따라 인입된 것일 수도 있다. 따라서 $j=1, \cdots, J$, $m=1, \cdots, M$에 대하여 다음과 같은 식이 성립한다.

$$\Lambda_j^m = \lambda^m q_{0j}^m + \sum_{k=1}^{J} \Lambda_k^m q_{kj}^m \tag{3}$$

또한 Λ_j를 응용에 무관하게 서비스 j에 도착하는 모든 작업의 도착율이라고 하면 다음이 성립한다.

$$\Lambda_j = \sum_{m=1}^{M} \Lambda_j^M,\ j=1, \cdots, J.$$

다중 클래스 개방형 잭슨 네트워크의 각 노드에서의 고객수 분포는 대응되는 M/M/c 대기행렬의 고객수 분포와 같다(Gross and Harris, 1998). 따라서 P_n^j를 서비스 j에 머물고 있는 작업, 즉 서비스 j의 인스턴스에서 작업 중이거나 서비스 j의 대기열에 있는 모든 작업이 n, $n=0, 1, \cdots$, 개일 확률이

라고 하면 다음이 성립한다.

$$\Lambda_j P_n^j = (n+1)\mu_j P_{n+1}^j, \ 0 \le n < c_j \tag{4}$$
$$\Lambda_j P_n^j = c_j \mu_j P_{n+1}^j, \ n \ge c_j$$

식 (4)를 축차적으로 정리하면 다음의 결과를 얻는다.

$$P_n^j = \begin{cases} \dfrac{\Lambda_j^n}{n!\mu_j^n} P_0^j, & 0 \le n < c_j \\ \dfrac{\Lambda_j^n}{c_j^{n-c_j} c_j! \mu_j^n} P_0^j, & n \ge c_j. \end{cases} \tag{5}$$

단,

$$P_0^j = \left(\sum_{n=0}^{c_j - 1} \frac{\Lambda_j^n}{n!\mu_j^n} + \frac{\Lambda_j^{c_j}}{c_j! \mu_j^{c_j}} \cdot \frac{c_j \mu_j}{c_j \mu_j - \Lambda_j} \right)^{-1}. \tag{6}$$

$L_{q,j}^m$와 $W_{q,j}^m$를 각각 정상 상태에서의 서비스 j의 대기열에서 대기하고 있는 응용 m의 평균 작업수와 응용 m의 작업이 서비스 j의 대기열에서 기다리는 평균 대기 시간이라고 하자. 응용 m의 작업은 서비스 j에 여러 번 인입될 수도 있고 한번도 인입이 되지 않을 수 있다. 따라서 응용 m의 서비스 j에 인입되는 도착율은 Λ^m이므로 리틀의 법칙에 의해 다음이 성립한다.

$$L_{q,j}^m = \Lambda_j^m \cdot W_{q,j}^m. \tag{7}$$

또한 우선순위가 있는M/M/c 대기행렬의 대기시간에 대한 분석 결과(Hillier and Lieberman, 2010)를 이용하면 응용 m 요청이 서비스 j에 한번 인입되었을 때 대기열에서 기다리는 평균 시간 $W_{q,j}^m$에 대하여 다음 관계식을 얻는다.

$$W_{q,j}^m = \frac{\displaystyle\sum_{k;\delta_j(k) \le \delta_j(m)} \frac{L_{q,j}^k}{c_j \mu_j} + \sum_{n=c_j}^{\infty} \frac{p_n^j}{c_j \mu_j}}{1 - \displaystyle\sum_{k;\delta_j(k) < \delta_j(m)} \frac{\Lambda_j^k}{c_j \mu_j}}. \tag{8}$$

식 (8)의 의미는 다음과 같다. 먼저 분자의 첫 번째 항은 응용 m 요청이 서비스 j에 인입될 때 대기열에 있는 자신보다 우선순위가 같거나 높은 모든 작업 요청들에 의해 발생하는 평균 대기시간이다. 분자의 두 번째 항은 서비스 j에 인입될 때 c_j개의 모든 서비스 인스턴스가 서비스 중이서 평균적으로 대기해야 할 시간이다. 분모는 분자의 시간만큼 기다리는 동안 도착하는 자신보다 우선순위가 높은 작업 요청에 의해 늘어나는 대기 시간의 증가 요소이다. 식 (8)에 식 (5)~식 (7)의 결과를 대입하면 다음 결과를 얻는다.

$$\left(1 - \sum_{k;\delta_j(k) \le \delta_j(m)} \frac{\Lambda_j^k}{c_j \mu_j}\right) W_{q,j}^m = \sum_{k;\delta_j(k) < \delta_j(m)} W_{q,j}^k \cdot \frac{\Lambda_j^k}{c_j \mu_j} + E_j \tag{9}$$

단,

$$E_j = \left[\frac{(c_j \mu_j - \Lambda_j) c_j! \mu_j^{c_j}}{\Lambda_j^{c_j}} \sum_{n=0}^{c_j - 1} \frac{\Lambda_j^n}{n!\mu_j^n} + c_j \mu_j \right]^{-1}$$

식 (9)를 우선순위가 가장 높은 응용부터 축차적으로 정리하면 다음 식을 얻는다.

$$W_{q,j}^m = \frac{E_j}{\left(1 - \displaystyle\sum_{k;\delta_j(k) < \delta_j(m)} \frac{\Lambda_j^k}{c_j \mu_j}\right)\left(1 - \displaystyle\sum_{k;\delta_j(k) \le \delta_j(m)} \frac{\Lambda_j^k}{c_j \mu_j}\right)} \tag{10}$$

L_j^m와 W_j^m를 각각 서비스 j에 있는, 즉 서비스 j의 대기열과 서비스 인스턴스에서 서비스 받고 있는 응용 m의 평균 작업수와, 응용 m의 작업이 서비스 j에 체류하는 평균 시간-즉 서비스 j에서 대기열에서의 평균 대기시간과 서비스 평균 처리 시간의 합-이라고 하자. 모든 작업은 서비스 j에서 μ_j의 매개변수를 가지는 지수분포로 서비스를 받으므로 다음이 성립한다.

$$W_j^m = W_{q,j}^m + \frac{1}{\mu_j} \tag{11}$$

또한 리틀의 법칙에 의해 식 (7)과 마찬가지로 다음을 얻는다.

$$L_j^m = \Lambda_j^m \cdot W_j^m \tag{12}$$

L^m과 W^m를 각각 시스템 전체에 있는 응용 m의 평균 작업수와 응용 m의 작업의 시스템에 머무는 평균 체류시간-즉 시스템에 들어온 후 시스템을 나갈 때까지의 평균 시간-이라고 하면 시스템에 있는 전체 작업 수는 각 서비스에 있는 작업의 합이 되므로 다음이 성립한다.

$$L^m = \sum_{j=1}^{J} L_j^m.$$

그러나 시스템에 머무는 총 체류시간 W^m은 각 서비스에 인입될 때 머무는 평균 체류시간의 합으로 구하면 안 된다. 왜냐하면 한 서비스에 여러 번 인입될 수 있기 때문이다. 따라서 시스템에 모든 작업은 한번만 인입되고, 응용 m의 작업이 λ^m이므로 리틀의 법칙을 이용하면 응용 m의 작업의 시스템 평균 체류시간은 다음과 같다.

$$W^m = L^m/\lambda^m = \sum_{j=1}^{J} L_j^m/\lambda^m \tag{13}$$

식 (13)에 식 (10)~식 (12)의 결과를 대입하면 응용 m의 작업의 시스템 평균 체류시간은 다음과 같다.

$$W^m = \frac{1}{\lambda^m} \times \sum_{j=1}^{J} \left\{ \frac{\Lambda_j^m E_j}{\left(1 - \sum_{k;\delta_j(k) < \delta_j(m)} \frac{\Lambda_j^k}{c_j \mu_j}\right)\left(1 - \sum_{k;\delta_j(k) \le \delta_j(m)} \frac{\Lambda_j^k}{c_j \mu_j}\right)} + \frac{\Lambda_j^m}{\mu_j} \right\} \tag{14}$$

따라서, 식 (14)를 이용하면 식 (2)의 최적화 문제의 각 대안별 비용을 구할 수 있다.

5. 최적 서비스 인스턴스 수와 우선순위 정책 결정을 위한 단순한 휴리스틱 알고리즘

본 장에서는 제 4장에서 제시된 성능 분석 모델을 이용하여 서비스별 최적 서비스 인스턴스 수와 우선순위 정책을 결정하는 단순한 알고리즘을 제안한다.

최적의 서비스 인스턴스 수와 우선순위 정책을 결정은 식 (2)의 비용 최소화 문제를 해결하는 것이다. 그런데 식 (14)에서 볼 수 있듯이 비용함수가 의사결정 변수 c_j와 $\delta_{j(m)}$에 대해 복잡한 비선형 형태를 가지므로 최적의 해를 구하는 일반적 방정식을 찾기 어렵다. 따라서 먼저 모든 가능한 경우를 고려하여 최적해를 구하는 알고리즘을 고려해 볼 수 있다. 그러나 서비스 인스턴스 수를 고정하고 서비스별로 서로 다른 우선순위 정책을 헤아리는 경우의 수조차도 매우 많은 경우가 발생한다. 예로서 모든 응용에 대해 동등한 우선순위를 부여하거나 또는 우선순위를 부여할 때는 동률인 응용이 없도록 우선순위를 부여하는 경우의 수는 $(1+M)^J$개가 있다. 서비스 인스턴스 수가 결정되어 있지 않다면 고려해야 할 경우의 수는 크게 증가한다. 전체 VM의 수가 $k, k \ge J$라고 하자. 개별 서비스가 J개이므로 서비스 당 하나의 서비스 인스턴스는 운영되어야 하므로 J개의 VM은 각 서비스 별로 하나씩 배정되어야 한다. 나머지 $k-J$개의 서비스 인스턴스를 J개의 서비스로 할당하는 경우의 수는 ${}_{k-1}C_{J-1}$이다. 만약 전체 서비스 인스턴스 수가 K, $K \ge J$개가 넘지 않는 모든 경우의 수는 다음과 같다.

$$\sum_{k=j}^{J} {}_{k-1}C_{J-1} = {}_{K}C_{J} = \frac{K!}{J!} \ge (J+1)^{K-J}.$$

따라서 전체 서비스 인스턴스 수가 K개가 넘지

않으면서 모든 우선순위의 정책을 고려하는 경우의 수를 $O_{all}(K)$라고 하면, 다음이 성립한다.

$$O_{all}(K) \geq (1+m!)^{J} \cdot (J+1)^{K-J}.$$

<Figure 1>처럼 응용이 2개, 서비스가 5개일 때 전체 서비스 인스턴스 수가 10개 이하에 대해서만 가능한 모든 대안을 고려해도 1,889,568번 이상의 계산 반복을 수행하여야 한다.

따라서 최적해를 빠른 시간 내에 찾기 위해서는 적절한 휴리스틱 방법이 필요하다. 본 논문에서는 제시된 성능 분석 모델을 이용하여 최적해를 찾는 다음과 같은 중첩된 그리디 알고리즘을 이용하는 휴리스틱 방법을 제안한다. 내부 알고리즘은 각 서비스 별 서비스 인스턴스의 수가 고정되어 있을 때 서비스 별로 응용의 최적 우선순위 규칙을 찾는 그리디 알고리즘이다. 외부 알고리즘은 총 서비스 인스턴스 수를 하나씩 증가시켜 나가면서 어느 서비스에 인스턴스를 증가시켜야 하는지를 찾는 그리디 알고리즘이다. 외부 알고리즘에서 어느 서비스의 인스턴스를 증가시킬지에 대한 대안을 비교할 때 내부 알고리즘을 이용해 각 서비스 인스턴스 배치별로 최적 우선순위 규칙을 찾아서 그 때의 목적식의 값을 비교한다.

다음은 내부 알고리즘을 담당하는 서비스 별로 서비스 인스턴스 수가 고정되어 있을 때 최적 우선순위 규칙을 찾는 그리디 알고리즘이다.

- STEP 1 : 초기 상태 우선순위 규칙 $\delta_j^0(m)$으로 모든 서비스에서 모든 응용이 동등한 최상위 우선순위를 갖는다. 즉, $\delta_j^0(m)=1,\ j=1, \cdots, J,\ m=1, \cdots, M$. 그리고 성능 분석 모델과 비용 함수를 이용하여 초기 상태 우선순위 규칙일 때의 총 비용 T^0를 구한다. 현재의 최적 우선순위 규칙과 총 비용을 $\{\delta_j^{current}(m);\ j=1, \cdots, J,\ m=1, \cdots, M\}$와 $T^{current}$라고 하면, $\delta_j^{current}(m)=\delta_j(m)$와 $T^{current}=T^0$로 하여 알고리즘의 반복을 시작한다.
- STEP 2 : 현재의 우선순위 규칙에서 어떤 하나의 서비스에서 어떤 하나의 응용의 우선순위를 한 단계 내려서 만들어지는 새로운 우선순위를 "한 단계 변형 우선순위 규칙"이라 하자. 즉, 현재의 우선순위 규칙을 $\{\delta_j^{current}(m);\ j=1, \cdots, J,\ m=1, \cdots, M\}$이라 하고 서비스 j_0에서 응용 m_0의 우선순위를 낮춘 한 단계 변형 우선순위 규칙을 $\{\delta_j^{alt}(m; j_0, m_0);\ j=1, \cdots, J,\ m=1, \cdots, M\}$라고 하면 다음이 성립한다.

$$\delta_{j_0}^{alt}(m_0; j_0, m_0)=\delta_{j_0}^{current}(m_0)+1$$
$$\delta_j^{alt}(m; j_0, m_0)=\delta_j^{current}(m), \quad j \neq j_0 \text{ or } m \neq m_0$$

  성능 분석 모델과 비용 함수를 이용하여 모든 한 단계 변형 우선순위 규칙 $\{\delta_j^{alt}(m; j_0, m_0);\ j=1, \cdots, J,\ m=1, \cdots, M\}$에서의 총 비용 $T^{alt}(j_0, m_0)$를 계산한다.
- STEP 3 : 모든 가능한 한 단계 변형 우선순위 규칙의 총 비용 최소값이 현재의 우선순위 규칙의 총 비용 보다 낮으면, 즉 $\min T^{alt}(j_0, m_0) < T^{current}$이면, 최소 총 비용을 만든 한 단계 변형 우선순위 규칙과 총 비용을 현재의 최적 우선순위 규칙과 총 비용으로 변경한 후 STEP 2로 돌아간다. 만약 $\min T^{alt}(j_0, m_0) \geq T^{current}$이지 않으면 현재의 최적 우선순위 규칙과 총 비용을 최적해로 간주하고 알고리즘을 종료한다.

다음은 외부 알고리즘을 담당하는 최적의 서비스 인스턴스 수를 구하는 그리디 알고리즘이다.

- STEP 1 : 초기의 서비스 인스턴스 배치로 각 서비스 별로 하나의 서비스 인스턴스를 배치한다. 즉, $c_j=1,\ j=1, \cdots, J$ 위에서 제시한 최적 우선순위 규칙을 찾는 알고리즘을 이용하여 현재의 서비스 인스턴스 수에서 가장 최적의 우선순위 규칙 $\delta_j^{current}(m)$을 결정하고 이 때의 최적 비용 함수 $T^{current}$를 구한다.

- STEP 2 : 현재의 서비스 인스턴스 배치에서 임의의 하나의 서비스에 한 개의 서비스 인스턴스 수를 증가시키는 배치를 한 단계 증가 서비스 인스턴스 배치라고 하자. 즉, 현재의 서비스 인스턴스 배치를 $c_j^{current}$에서 서비스 j_0의 인스턴스를 하나 증가시킨 한 단계 증가 서비스 인스턴스 배치를 $c_j^{alt}(j_0)$라고 하면 다음이 성립한다.

$$c_{j_0}^{alt}(j_0) = c_{j_0}^{current} + 1$$
$$c_j^{alt}(j_0) = c_j^{current},\ j \neq j_0$$

  모든 한 단계 증가 서비스 인스턴스 배치 $c_j^{alt}(j_0)$에 대해 최적 우선순위 규칙을 찾는 알고리즘을 이용하여 최적 우선순위 규칙 $\delta_j^{alt}(m; j_0)$과 최적 비용 함수 $T^{alt}(j_0)$를 구한다.
- STEP 3 : 모든 가능한 한 단계 증가 서비스 인스턴스 배치의 총 비용의 최소값이 현재의 서비스 인스턴스 배치의 총 비용보다 낮으면, 즉 $\min T^{alt}(j_0) < T^{current}$라고 하면, 최소 총 비용을 갖는 한 단계 증가 서비스 인스턴스 배치와 그 때의 최적 우선순위 규칙과 총 비용을 현재의 서비스 인스턴스 배치, 현재의 최적 우선순위 규칙, 현재의 최적 총 비용으로 변경한 후 STEP 2로 돌아간다. 만약 $\min T^{alt}(j_0) \geq T^{current}$이면 현재의 서비스 인스턴스 배치, 서비스 우선순위 규칙, 총 비용을 최적해로 간주하고 알고리즘을 종료한다.

본 논문에서는 제시한 휴리스틱 알고리즘의 정확성을 검증하기 위한 수학적 분석은 수행하지 않았다. 대신 제 6장의 수치 예제를 통해 제시한 휴리스틱 알고리즘이 최적해를 잘 찾아냄을 예시한다.

6. 수치 예제

본 장에서는 제 4장에서 제시된 성능 분석 모델과 제 5장에서 제시한 최적해를 찾는 알고리즘이, 제 3장에서 제시된 최적 서비스 인스턴스 수와 우선순위 정책 결정 문제를 해결하는 데 적용될 수 있음을 보이기 위해 간단한 수치 예제를 다룬다. 수치 예제로는 <Figure 1>에서 기술된 두 개의 응용과 5개의 재사용 가능 서비스로 구성된 시스템을 고려할 것이다. Kim(2013a)에서는 우선순위가 없는 경우에 대해 동일한 수치 예제가 분석되었으니 우선순위가 없는 경우와 결과를 비교하기 위해 참조하기 바란다.

<Figure 1>의 예에서 응용 1의 작업은 먼저 서비스 1의 인스턴스 중 하나로 라우팅된다. 그리고 응용 1의 작업 흐름은 분기를 가진다. 작업 흐름의 분기별로 라우팅 확률은 과거 데이터로 추정한다. <Figure 1>의 예에서는 응용 1의 사용자 요청은 확률 0.8로 첫 분기에서 서비스 2로 분기되고 확률 0.2로 서비스 3으로 분기된다. 서비스 2에서 서비스가 완료된 후에는 응용 1의 작업은 반복 작업 흐름을 갖고 확률 0.35로 서비스 2로 다시 피드백되거나 확률 0.65로 서비스 3으로 라우팅된다. 서비스 3에서 서비스가 완료된 후에는 응용 1의 작업은 모든 서비스가 완료되어 시스템을 떠나게 된다. 응용 2의 작업도 작업 흐름만 다르지 동일한 방법으로 라우팅된다.

응용 1과 응용 2의 작업의 도착 과정은 포아송 분포를 따르며 단위 시간 당 한 개의 요청이 도착한다($\lambda^1 = \lambda^2 = 1$). 모든 재사용 가능 서비스는 매개변수 3의 동일한 지수 분포를 따른다($\mu_j = 3,\ j = 1, \cdots, 5$). 마지막으로 <Figure 1>의 응용 1과 응용 2의 작업흐름으로부터 각 응용별로 서비스 사이의 라우팅 확률 행렬 $\boldsymbol{Q^m} = (q_{ij}^m)_{0 \leq i,j \leq 5},\ m = 1, 2$을 다음과 같이 구할 수 있다(Kim(2013a)과 같은 결과이지만 논문의 완결성을 위해 기술한다).

$$\boldsymbol{Q^1} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.8 & 0.2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.35 & 0.65 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},\ \boldsymbol{Q^2} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.6 & 0.4 & 0 & 0 & 0 \\ 0.7 & 0 & 0.3 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

응용별로 작업 처리 시간에 대한 단위 비용은 $u_1 = 10$

〈Table 1〉 The Total Cost under Various Priority Rules Generated by the Optimum Priority Searching Algorithm

Iteration		S1	S2	S3	S4	S5	W^1	W^2	Total Cost
0	App 1 Priority	1	1	1	1	1	2.474	2.090	81.99
	App 2 Priority	1	1	1	1	1			
1	App 1 Priority	1	2	1	1	1	2.850	1.714	76.35
	App 2 Priority	1	1	1	1	1			
2	App 1 Priority	1	2	2	1	1	2.895	1.669	75.68
	App 2 Priority	1	1	1	1	1			

과 $u_2 = 25$라고 하자. 응용 2의 처리 시간 지연이 응용 1의 지연보다 2.5배 정도 더 비용을 발생시키므로 응용 2가 처리 시간 지연에 더 민감한 작업임을 알 수 있다. 서비스가 사용하는 VM은 모두 동일한 형태로 한 개의 VM을 유지하는 데 드는 비용은 모두 동일하고 단위 시간당 $h_j = 1,\ j = 1, \cdots, J$로 발생한다고 가정한다.

개별 서비스의 서비스 인스턴스 수와 우선순위 정책이 결정되어 있으면 식 (3)~식 (14)를 이용하여 응용 별로 작업 처리 시간의 기대치를 계산할 수 있고, 식 (2)로 총 비용을 계산할 수 있다. <Table 1>은 모든 서비스의 인스턴스가 1개일 때 최적 우선순위 규칙을 제 4장에서 제시한 휴리스틱 알고리즘으로 구한 결과이다. 초기 우선순위 규칙 이후

〈Table 2〉 The Total Cost under Various Numbers of Service Instances and Priority Rules Generated by the Optimum Service Instances Deployment and Priority Searching Algorithm

Iteration		S1	S2	S3	S4	S5	W^1	W^2	Total Cost
0	The number of Instances	1	1	1	1	1	2.894926	1.669231	75.68003
	App 1 Priority	1	2	2	1	1			
	App 2 Priority	1	1	1	1	1			
1	The number of Instances	1	2	1	1	1	1.646073	1.420787	57.9804
	App 1 Priority	1	2	2	1	1			
	App 2 Priority	1	1	1	1	1			
2	The number of Instances	1	2	2	1	1	1.330099	1.355298	54.18345
	App 1 Priority	1	2	2	1	1			
	App 2 Priority	1	1	1	1	1			
3	The number of Instances	1	2	2	2	1	1.330099	1.198155	51.25487
	App 1 Priority	1	2	2	1	1			
	App 2 Priority	1	1	1	1	1			
4	The number of Instances	1	2	2	2	2	1.330099	1.089822	49.54654
	App 1 Priority	1	2	2	1	1			
	App 2 Priority	1	1	1	1	1			
5	The number of Instances	2	2	2	2	2	1.172956	1.089822	48.97511
	App 1 Priority	1	2	2	1	1			
	App 2 Priority	1	1	1	1	1			
6	The number of Instances	2	3	2	2	2	1.114169	1.063706	48.73435
	App 1 Priority	1	2	2	1	1			
	App 2 Priority	1	1	1	1	1			

에 총 2번의 “한 단계 변형 우선순위 규칙” 생성을 통해 최적으로 추정되는 우선순위 규칙을 제시하였다. 결과에서 보듯이 응용 2의 처리 시간 지연에 대한 단위 비용이 높으므로 응용 1과 응용 2과 동시에 공유하는 서비스 2와 서비스 3에서 응용 1의 우선순위를 낮추는 방향으로 최적 우선순위 규칙을 탐색해 나갔다.

<Table 2>는 모든 서비스의 인스턴스가 1개일 때부터 하나씩 서비스 인스턴스를 늘려가면서 최적 서비스 인스턴스 배치와 우선순위 규칙을 휴리스틱 알고리즘을 이용하여 구한 결과이다. 초기 단계에서 총 6번의 한 단계 서비스 인스턴스 증가를

〈Table 3〉 The Optimum Service Instances Deployment and Priority under Various Unit Costs

Unit Costs		Optimum	S1	S2	S3	S4	S5	W^1	W^2	Total Cost
u_1	u_2									
1	10	The number of Instances	1	2	1	2	2	1.646073	1.155311	21.19918
		App 1 Priority	1	2	2	1	1			
		App 2 Priority	1	1	1	1	1			
2	9	The number of Instances	1	2	2	2	1	1.330099	1.198155	21.4436
		App 1 Priority	1	2	2	1	1			
		App 2 Priority	1	1	1	1	1			
3	8	The number of Instances	1	2	2	2	1	1.330099	1.198155	21.57554
		App 1 Priority	1	2	2	1	1			
		App 2 Priority	1	1	1	1	1			
4	7	The number of Instances	1	2	2	2	1	1.330099	1.198155	21.70748
		App 1 Priority	1	2	2	1	1			
		App 2 Priority	1	1	1	1	1			
5	6	The number of Instances	1	2	2	1	1	1.330099	1.355298	21.78228
		App 1 Priority	1	2	2	1	1			
		App 2 Priority	1	1	1	1	1			
6	5	The number of Instances	1	2	2	1	1	1.307613	1.377784	21.7346
		App 1 Priority	1	1	1	1	1			
		App 2 Priority	1	2	2	1	1			
7	4	The number of Instances	2	2	2	1	1	1.15047	1.377784	21.56443
		App 1 Priority	1	1	1	1	1			
		App 2 Priority	1	2	2	1	1			
8	3	The number of Instances	2	2	2	1	1	1.15047	1.377784	21.33711
		App 1 Priority	1	1	1	1	1			
		App 2 Priority	1	2	2	1	1			
9	2	The number of Instances	2	2	2	1	1	1.15047	1.377784	21.1098
		App 1 Priority	1	1	1	1	1			
		App 2 Priority	1	2	2	1	1			
10	1	The number of Instances	2	2	2	1	1	1.15047	1.377784	20.88248
		App 1 Priority	1	1	1	1	1			
		App 2 Priority	1	2	2	1	1			

거쳐서 최적으로 추정되는 서비스 인스턴스 배치와 우선순위 규칙을 제시하였다. 결과에서 보듯이 모든 단계에서 응용 2의 처리 시간 지연에 대한 단위 비용이 높으므로 응용 1의 우선순위를 낮추는 방향으로 최적 우선순위 규칙을 찾아갔다. 그리고 응용 1과 응용 2과 동시에 공유하는 서비스 2와 서비스 3를 가장 우선적으로 새로 추가되는 서비스 인스턴스를 할당하였다.

<Table 3>은 비용함수의 특성에 따라 최적해가 어떻게 바뀌는지를 보기 위해 응용별의 작업 처리 시간당 발생 비용을 변경해보았다. 모든 경우에서 공통적으로 이용되는 서비스 2와 서비스 3에 우선적으로 서비스 인스턴스를 배치하는 경향을 보였으며, 응용 1의 작업 처리 비용이 응용 2에 비해 상대적으로 많이 발생하면 응용 1만 사용하는 서비스 1에도 서비스 인스턴스를 더 배치하였고, 반대로 응용 2의 비용이 상대적으로 커지면 응용 2만 사용하는 서비스 4와 서비스 5에 서비스 인스턴스를 배치하는 경향을 보였다. 우선순위 정책은 응용 1의 처리 시간에 대한 비용이 응용 2에 비해 상대적으로 커지면 공통적으로 사용하는 서비스 2와 서비스 3에서 응용 2의 우선순위를 낮추었고, 응용 2의 처리시간에 대한 상대적 비용이 커지면 서비스 2와 서비스 3에서 응용 1의 우선순위를 낮추었다.

마지막으로 제 5장에서 제시한 알고리즘이 전역 최적해를 발견하는 정확도를 검증해 본다. 검증을 위해 열거법에 의해 최적 서비스 인스턴스 수와 우선순위 정책 조합을 찾은 후 제시된 휴리스틱 알고리즘과 비교하였다. <Table 4>는 <Table 3>의 예에 대하여 열거법(Enumeration로 표시)과 제안된 휴리스틱 알고리즘(Greedy로 표시)의 계산 반복수와 시간을 비교하고, 두 알고리즘에 의해 찾은 최적해가 동일한지를 비교한 결과이다. 열거법에서 서비스 인스턴스의 수는 3회 이상 최적해의 개선이 이루어지지 않으면 열거를 중단하도록 하였다. 결과에서 보듯이 동일한 작업 흐름 하에서 10개의 서로 다른 비용 구조에 대하여 제시한 휴리스틱 알고리즘이 열거법과 동일한 전역 최적해를 찾음을 보여준다. 반면 제안된 휴리스틱 방법이 최적해를 찾는 동안 걸린 계산 반복회수와 탐색 시간은 열거법의 0.126%와 0.130%로 크게 감소하였다.

〈Table 4〉 Comparison of the Proposed Algorithm with the Enumeration Algorithm

Unit Costs		Identical Optimum	Greedy		Enumeration	
u1	u2		Iteration	Time	Iteration	Time
1	10	Yes	651	0.16	425,984	94.27
2	9	Yes	651	0.15	448,512	99.15
3	8	Yes	651	0.15	456,704	101.11
4	7	Yes	651	0.15	456,704	100.92
5	6	Yes	496	0.11	440,320	97.11
6	5	Yes	496	0.11	440,320	97.28
7	4	Yes	651	0.14	564,224	127.1
8	3	Yes	651	0.15	564,224	127.35
9	2	Yes	651	0.15	564,224	126.96
10	1	Yes	651	0.16	564,224	126.93
Average		100%	620	0.143	492,544	109.818

다양한 작업 흐름에 대하여 제시한 휴리스틱 알고리즘이 전역 최적해를 발견해내는 정확도를 확인하기 위해, 응용의 수는 2, 재사용 서비스의 수는 5인 경우에 두 응용의 작업흐름 쌍을 임의로 100개 생성하여 열거법에 의한 최적해와 제시한 알고리즘의 최적해를 비교하였다. <Table 5>는 임의로 생성한 100개의 작업 흐름 쌍에서 응용의 서비스 이용 빈도를 보여주고 있다. 예를 들어, 응용 1의 100개의 작업 흐름은 서비스 1을 55번 포함하였고, 응용 2의 100개의 작업 흐름은 서비스 1을 54번 포함하였다. 그리고 100개의 작업 흐름 쌍에서 두 응용의 작업흐름 모두 서비스 1을 공통으로 이용한 경우는 32번 발생하였다. 평균적으로 응용 1의 작업 흐름은 2.66개의 서비스로 구성되었고, 응용 2의 작업흐름은 2.86개의 서비스로 구성되었다. 하나의 작업흐름 쌍이 공통으로 사용한 서비스는 평균적으로 1.55개였다. <Table 6>은 임

의로 생성한 100개의 작업흐름 쌍에 대해 제안된 휴리스틱 알고리즘과 열거법에 의해 구한 최적해를 비교한 결과이다. 100개의 작업흐름에 대하여 100% 최적해가 일치하였다. 따라서 제안된 최적해가 전역 최적해를 상당히 정확하게 찾는다는 것을 제한된 예를 통해서 확인할 수 있다. 계산 반복회수와 탐색 시간은 열거법의 0.13.6%로 모두 크게 감소하였다.

〈Table 5〉 The Distribution of the Services Used by 100 Randomly Generated Workflows

Optimum	S1	S2	S3	S4	S5	Mean number of used services
App 1	55	51	53	49	58	2.66
App 2	54	63	50	61	58	2.86
Common	32	35	26	29	33	1.55

〈Table 6〉 Comparison of the Proposed Algorithm with the Enumeration Algorithm for 100 Randomly Generated Workflows

Identical Optimum	Greedy		Enumeration	
	Iteration	Time	Iteration	Time
100%	721.6	0.16	528,660	117.2

7. 결 론

이 논문에서는 SOA의 철학에 따라 다수 응용이 사용하는 재사용 가능 서비스들이 클라우드 환경에 배치되는 경우에, 서비스별 최적 인스턴스 수와 우선순위 정책을 결정하는 방법을 제시하였다. 이를 위해 서비스 인스턴스 수와 우선순위 정책에 따라 응용별 처리 성능을 분석할 수 있는 수리적 성능 분석 모델을 제시하였고, 성능 분석 모델을 이용하여 최적해를 탐색하는 단순한 휴리스틱 알고리즘도 제시하였다. 그리고 간단한 수치 예를 통해 제시된 기법들이 논문에서 정의된 문제에 적절히 이용될 수 있음을 예시하였다.

SOA와 클라우드 환경은 시스템의 빠른 개발, 유지보수, 운영을 위한 핵심 사항으로 자리잡고 있다. 따라서 본 연구의 결과는 클라우드 환경에서 SOA 기반 시스템을 효율적으로 운영할 수 있는 메커니즘을 고려하는 사람들에게 이론적인 도움을 줄 것으로 기대한다.

마지막으로 본 연구의 한계와 이를 극복하기 위한 추후 연구 방향을 제시한다. 첫째, 제시된 성능 분석 모델은 병렬 작업 흐름을 적절히 모델링하는데 한계를 가진다. 향후 대기행렬 네트워크 모델에서 병렬 작업 흐름을 좀 더 적절히 반영할 수 있는 방안에 대한 연구가 필요하다. 둘째, 현재의 최적해를 찾는 알고리즘은 그리디 알고리즘으로 단순하다. 또한 제시한 알고리즘의 타당성에 대하여 제한된 예를 통해서만 검증하였다. 향후 제시한 알고리즘에 대한 수학적인 검증과 다양한 예를 이용한 타당성 검증이 수행되어야 한다. 또한 유전자 알고리즘 등 좀 더 개선된 알고리즘을 여러 가지 예를 통해 비교함으로써 좀 더 최적의 알고리즘을 찾는 연구가 필요하다. 셋째, 본 연구는 성능 분석 모델과 알고리즘만을 제시하였는데, 제 3장의 <Figure 3>과 같은 실제 제어 메커니즘을 구현하여 제시된 방법을 현실 응용에 적용하는 연구가 향후 필요하다. 또한 제 4장의 성능 분석 모형은 실제 시스템의 구현이 아직 되지 않은 상태에서 해당 시스템에 적용될 성능 분석 기법을 제시한 연구이므로, 실제 시스템과 성능 결과를 비교는 할 수 없었다. 따라서 추후 연구에서 성능 분석 모형의 가정에 대한 타당성 검증이 추후 연구로 필요하다.

References

Armbrust, M., A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, “A view of cloud computing”, *Communications of the ACM*, Vol.53, No.4, 2010, 50-58.

Brebner, P., L. O’Brien, and J. Gray, “Performance modeling evolving Enterprise Service Orien-

ted Architectures", in *Software Architecture, 2009 and European Conference on Software Architecture. WICSA/ECSA, Joint Working IEEE/IFIP Conference on*, 2009, 71-80.

Brebner, P., "Service-oriented performance modeling the MULE enterprise service bus(ESB) loan broker application", in *Software Engineering and Advanced Applications, SEAA 35th Euromicro Conference on*, 2009, 404-411.

Chudov, A., Y. Rebrova, and A. Pastsyak, "Performance Modeling of Worldwide Distributed IT Support System at Siemens IT Solutions and Services", in *Computer Modeling and Simulation(EMS), Fifth UKSim European Symposium on*, 2011, 415-417.

Buyya, R., C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms : Vision, hype, and reality for delivering computing as the 5th utility", *Future Generation computer systems*, Vol.25, No.6, 2009, 599-616.

Dejun, J., G. Pierre, and C.-H. Chi, "EC2 performance analysis for resource provisioning of service-oriented applications", in *Service-Oriented Computing, ICSOC/Service Wave Workshops,* 2010, 197-207.

Erl, T., *Service-oriented architecture. Prentice Hall*, Englewood Cliffs, 2004.

Espadas, J., A. Molina, G. Jiménez, M. Molina, R. Ramírez, and D. Concha, "A tenant-based resource allocation model for scaling Software-as-a-Service applications over cloud computing infrastructures", *Future Generation Computer Systems*, Vol.29, No.1, 2013, 273-286.

Gao, T., H. Ma, I.-L. Yen, F. Bastani, and W.-T. Tsai, "Toward QoS analysis of adaptive service-oriented architecture", in *Service-Oriented System Engineering, SOSE IEEE International Workshop*, 2005, 219-226.

Gross, D. and C.M. Harris, *Fundamentals of queuing theory. Wiley*, New York, 1998.

Hillier, F.S. and G.J. Lieberman, *Introduction to Operations Research, 9ed ed., McGraw-Hill*, New York, 2010.

Huang, K.-C., B.-J. Shen, T.-J. Lee, H.-Y. Chang, Y.-H. Tung, and P.-Z. Shih, "Resource allocation and dynamic provisioning for Service-Oriented applications in cloud environment", in *Cloud Computing Technology and Science (CloudCom), IEEE 4th International Conference on*, 2012, 839-844.

Huhns, M.N. and M.P. Singh, "Service-oriented computing : Key concepts and principles", *Internet Computing, IEEE*, Vol.9, No.1, 2005, 75-81.

Jackson, J.R., "Jobshop-like queueing systems", *Management science,* Vol.10, No.1, 1963, 131-142.

Jackson, J.R., "Networks of waiting lines", *Operations Research*, Vol.5, No.4, 1957, 518-521.

Kim, K., "Performance Analysis for Multiple Service-Oriented Applications Sharing Reusable Services in the Cloud", *International Journal of Advancements in Computing Technology*, Vol.5, No.12, 2013a, 387-395.

Kim, K., "Priority Disciplines for Service-Oriented Applications Sharing Common Reusable Services", in *The 3rd International Conference on Convergence Technology*, 2013b, 1227-1228.

Liu, Y., I. Gorton, and L. Zhu, "Performance prediction of service-oriented applications based on an enterprise service bus", in *Computer Software and Applications Con-*

ference, COMPSAC 31st Annual International, Vol.1, 2007, 327-334.

OASIS, "Web services business process execution language version 2.0", *OASIS Standard*, Vol.11, 2007.

Papazoglou, M.P. and W.J. Van Den Heuvel, "Service oriented architectures : approaches, technologies and research issues", *The VLDB journal*, Vol.16, No.3, 2007, 389-415.

Park, Y. and Y. Kim, "A Study on the Factors Affecting the Adoption of Cloud Computing Service : Focused on the Technology Acceptance Model(TAM) and Resistance", *Korea Society of IT Services Journal*, Vol.12, No.4, 2013, 1-23.

Pautasso, C., O. Zimmermann, and F. Leymann, "Restful web services vs. big'web services : making the right architectural decision", in *Proceedings of the 17th international conference on World Wide Web*, 2008, 805-814.

Richards, M., R. Monson-Haefel, and D.A. Chappell, *Java message service. O'Reilly Media, Inc.*, 2009.

Ryu, J.H., H.Y. Moon and J. Choi, "Analysis of Influence Factors on the Intention to Use Personal Cloud Computing", *Korea Society of IT Services Journal*, Vol.12, No.4, 2013, 319-335.

Teixeira, M., R. Lima, C. Oliveira, and P. Maciel, "Performance evaluation of service-oriented architecture through stochastic Petri nets", in *Systems, Man and Cybernetics, SMC, IEEE International Conference on*, 2009, 2831-2836.

Wada, H., J. Suzuki, and K. Oba, "Queuing theoretic and evolutionary deployment optimization with probabilistic SLAs for service oriented clouds", in *Services-I, World Conference on*, 2009, 661-669.

Yuan, X. and C. Ji, "Performance analysis of service-oriented architectures with multi-factor sensitivity analysis", in *Electro/Information Technology, IEEE International Conference on*, 2007, 198-203.

About the Authors

Kilhwan Kim (khkim@smu.ac.kr)

Kilhwan Kim received his master's degree and Ph.D. from KAIST, Daejeon, Korea. Currently, he is an assistant professor in Department of Management Engineering at Sangmyung University. His main research interests include stochastic modeling, queueing theory, and the performance analysis of cloud computing and converged services platforms.

Changsup Keum (cskeum@etri.re.kr)

Changsup Keum is a research director at ETRI, South Korea. He received his Ph. D. degree from Korea Advanced Institute of Science and Technology (KAIST) in 2013. He received his B.S. degree in Computer Science from University of Seoul, Korea in 1992 and received his M.E. degree in Software Engineering from Carnegie Mellon University (CMU) in 2005. He joined ETRI as a member of research staff in 1994. Since 2000 to 2003 he was engaged in research and development of softswitch systems. Since 2004, he has been engaged in research and development of various mobile service platforms. His interests include 5G mobile service platform, mobile cloud architecture, and software engineering. He is a member of KIPS (Korea Information Processing Society) and KIISE (Korean Institute of Information Scientists and Engineers).

Hyunjoo Bae (hjbae@etri.re.kr)

Hyunjoo Bae received her master's degree in Computer Science. from PNU, Pusan, Korea. Currently, she is working as a principal member of engineering staff in Smart Network Research Lab., ETRI (Electronics and Telecommunications Research Institute), Daejeon, Korea. Her main research interests include network service, service computing, and cloud computing.