pISSN: 2287-5905



http://dx.doi.org/10.3745/KTSDE.2013.2.6.383

# Dynamic Clustering based Optimization Technique and Quality Assessment Model of Mobile Cloud Computing

Dae Young Kim[†] · Hyun Jung La[††] · Soo Dong Kim[†††]

## ABSTRACT

As a way of augmenting constrained resources of mobile devices such as CPU and memory, many works on mobile cloud computing (MCC), where mobile devices utilize remote resources of cloud services or PCs, have been proposed. Typically, in MCC, many nodes with different operating systems and platform and diverse mobile applications or services are located, and a central manager autonomously performs several management tasks to maintain a consistent level of MCC overall quality. However, as there are a larger number of nodes, mobile applications, and services subscribed by the mobile applications and their interactions are extremely increased, a traditional management method of MCC reveals a fundamental problem of degrading its overall performance due to overloaded management tasks to the central manager, i.e. a bottle neck phenomenon. Therefore, in this paper, we propose a clustering-based optimization method to solve performance-related problems on large-scaled MCC and to stabilize its overall quality. With our proposed method, we can ensure to minimize the management overloads and stabilize the quality of MCC in an active and autonomous way.

**Keywords : Mobile Cloud Computing, Clustering, Logical Nearness, k-means Algorithm, Optimization**

# 동적 클러스터링 기반 모바일 클라우드 컴퓨팅의 최적화 기법 및 품질 평가 모델

김 대 영[†] · 라 현 정[††] · 김 수 동[†††]

## 요 약

CPU, 메모리 등 모바일 디바이스의 제한된 자원문제를 해결하기 위한 방법으로, 모바일 디바이스의 자원이 아닌 클라우드 서비스 또는 PC 등 외부 자원을 사용하는 모바일 클라우드 컴퓨팅(Mobile Cloud Computing, MCC)이 부각되고 있다. 전형적인 MCC 환경(MCC Environment, MCE)은 다른 운영체제 및 플랫폼을 가지는 여러 개의 노드, 모바일 애플리케이션과 서비스들로 구성되어 있고, 중앙관리자는 MCE 전체 품질이 일정 수준 이상을 유지하도록 관리 태스크를 수행한다. 그러나, 노드 수, 모바일 애플리케이션 수, 서비스의 수가 많아지고 서비스 실행빈도가 높아질 경우, 중앙 관리자의 관리 태스크 과중으로 병목현상과 성능저하 문제가 제기될 수 있다. 본 논문에서는 이러한 대규모 *MCE*의 병목과 성능저하 문제를 해결하고, 전체 품질을 안정화시키기 위한 클러스터링(Clustering) 기반의 최적화 기법을 제안한다. 본 기법을 적용하면 *MCE*의 전체 품질을 안정화시키기 위한 부하를 최소화하면서, 능동적이며 자율적인 방식으로 품질을 보장할 수 있다.

**키워드 : 모바일 클라우드 컴퓨팅, 클러스터링, 논리적 근접도, k-means 알고리즘, 최적화**

## 1. 서 론

모바일 디바이스는 크기가 작고 가벼워 휴대가 용이하기 때문에, PC를 대신하여 컴퓨팅 디바이스로 널리 사용되고 있다. 그러나, PC에 비해서 상대적으로 낮은 성능을 가지고 있기 때문에, 자원 소비가 많고 높은 복잡도를 가진 애플리케이션은 모바일 디바이스에서 설치 및 운영되기 어렵다. 이런 문제를 해결하기 위해, 모바일 디바이스의 자원이 아닌 클라우드 서비스 또는 PC 등 외부 자원을 사용하는 모바일 클라우드 컴퓨팅(Mobile Cloud Computing, MCC) 연구가 활발히 진행되고 있다[1][2][3].

일반적으로 MCC 환경 (MCC Environment, MCE)에는 다른 운영체제와 플랫폼을 가지는 여러 개의 노드, 다양한 모바일 애플리케이션과 서비스 등이 배치되어 있다. 즉, 여러 이질적인 (Heterogeneous) 특성을 가지는 요소들이 서로 상

※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(2012R1A1B3004130, 2012R1A6A3A01018389).
† 준 회 원 : 숭실대학교 컴퓨터학과 석사과정
†† 종신회원 : 숭실대학교 모바일 서비스 소프트웨어공학 센터 연구 교수
††† 종신회원 : 숭실대학교 컴퓨터학부 교수
논문접수 : 2013년 1월 28일
수 정 일 : 1차 2013년 3월 13일
심사완료 : 2013년 3월 13일
* Corresponding Author : Hyun Jung La(hjla80@gmail.com)

호작용하여 *MCE* 환경을 이룬다. 이런 환경에서 모바일 디바이스에 자원 부족, 서비스 품질 저하 등의 문제가 발생하면, 주로 기능 컴포넌트를 다른 노드로 오프로딩(Offloading) 시킴으로써 해당 문제를 해결하는 접근법을 주로 사용한다. *MCE*에서는 오프로딩을 포함한 다양한 품질 향상 활동을 수행하기 위해, 여러 노드들을 관리하는 중앙 관리자가 존재한다[1][2][3]. 하나의 중앙 관리자는 이런 이질적인 요소들이 일정 수준 이상의 품질을 유지하도록 품질 측정, 품질 향상 활동 수행 등의 여러 관리 태스크를 수행한다.

*MCE*에 속한 노드들이 적을 경우에는 하나의 중앙 관리자의 품질 관리 관련 오버헤드가 비교적 적지만, 노드 수, 모바일 애플리케이션 수, 모바일 애플리케이션들이 사용하는 서비스의 수가 많아지거나 실행빈도가 높아질 경우, 기존의 중앙 관리자를 통한 서비스 실행 관리는 병목현상으로 인해 성능저하를 유발시킬 수 있는 문제를 가지고 있다. 즉, 중앙 관리자의 관리 오버헤드로 인해, 모바일 애플리케이션의 품질 저하 문제를 해결하지 못하고, 전체 *MCE* 품질이 오히려 저하되는 부작용을 초래하게 된다.

본 논문에서는 이런 중앙 관리자의 관리 오버헤드를 줄이기 위해, 클러스터 (Cluster) 라는 논리적인 집합 단위를 *MCE*에 도입한다. 클러스터는 비슷한 특성을 가지는 객체들끼리 모아놓은 집합을 의미하며[4], *MCE*에서는 의존성 및 결합도가 높은 노드들을 묶은 관리 단위이다.

본 논문의 구성은 다음과 같다. 먼저, *MCE*의 메타모델을 제시하여 클러스터를 지원하기 위한 핵심 요소들을 설명하고, 클러스터링 프로세스를 제시한다. 그리고, *MCE*에서 동적 클러스터링을 하는데 가장 중요한 기준인 논리적 거리를 계산하는 메트릭을 정의하고, 대표적인 클러스터링 알고리즘인 *k-means* 를 이용하여 클러스터링을 구성하는 기법을 제안한다. 그리고, 클러스터링 결과가 최적으로 진행되었는지를 평가하는 *MCE* 최적화 평가 모델을 정의한다. 마지막으로 실험을 통해, 클러스터링을 통한 관리 성능 향상 여부와 *k-means* 알고리즘을 이용하여 전체 클러스터가 최적화 상태로 구성되는지를 확인한다. 본 논문에서 제시된 기법을 이용하여 전체 클러스터를 최적화 상태로 구성함으로써, *MCE*의 품질 관리 문제를 해결함과 동시에 관리 오버헤드를 줄이면서 효율적으로 품질 관리를 할 수 있음을 기대한다.

## 2. 관련 연구

MCE 클러스터링과 직접적인 관련이 있는 연구는 선행되고 있지 않지만, 센서 네트워크 분야에서 적은 비용으로 클러스터링을 유지하는 기법, 로드 밸런싱을 위한 클러스터링 기법이 진행되고 있다.

Gerla와 Parekh의 연구에서는 노드간 근접 정도를 기반으로 클러스터링을 하는 기법을 제시하였다[5]. 한 노드는 전송 가능한 거리에 있는 노드들과 클러스터링되며, 가장 많은 노드의 정보를 가진 노드를 관리자로 지정한다. Baker와 Ephremides의 연구에서는 식별 기반 클러스터링 기법을 제시한다[6]. 각 노드마다 ID를 가지고 있고, 노드들은 클러스터 관리자 역할을 하는 노드보다 항상 작은 ID를 가지도록 한다. 필요시 클러스터 관리자 노드는 자신의 역할을 자신보다 작은 ID를 가진 노드에 위임할 수 있고, 따라서 각 클러스터들은 서로 일부 노드를 공유하게 된다. 노드 공유 시 발생 할 수 있는 문제들은 게이트웨이 역할을 노드에 부여하여 해결하고 있다. Aim과 Prakash의 연구에서는 하나의 클러스터에 최적 개수의 노드를 위치하도록 하는 기법을 다루고 있다[7]. 주기적으로 최적의 개수를 클러스터가 보유하고 있는지 검사하여, 조건을 만족하고 있지 않으면 관리자 역할의 노드를 일반 노드로 역할을 변경하고 새로운 관리자 노드를 임명하도록 한다. 이 외에 노드의 에너지 효율성을 고려하거나 노드의 이동성을 고려한 클러스터링 등의 다양한 기법들이 제시되고 있다.

현재까지 진행된 연구는 네트워크 대역폭을 기반으로 노드 간 근접도를 측정하여 클러스터링하는데 초점을 맞추고 있다. 본 논문에서는 네트워크 대역폭뿐만 아니라 물리적 거리, 상호작용 횟수 등의 다양한 요소를 바탕으로 논리적 근접도를 구하고, 상호작용이 많은 노드들을 클러스터링하여 MCE가 관리 오버헤드를 줄이면서 효율적으로 품질 관리를 할 수 있도록 한다. MCE 환경에서는 노드의 상태에 따라서 동적으로 클러스터링이 이루어지기 때문에 노드가 최적의 클러스터에 위치하도록 지속적인 모니터링 및 관리가 이루어져야 한다. 다른 연구에서는 로드 밸런싱 등을 통해 동적으로 클러스터링 되기는 하지만 모니터링 및 관리에 대한 평가 요소가 제한적이라 클러스터링 효과 또한 크지 않다. 그리고, 다른 연구에서는 로드 밸런싱 등을 통해 동적으로 클러스터링 되기는 하지만, 동적 클러스터링의 수행 시기를 판단하기 위한 필수 활동인 모니터링을 거의 고려하고 있지 않다. 또한, 기존 연구의 클러스터링은 MCE 클러스터링의 목적과 차이가 있으므로 이들을 그대로 MCE 클러스터링에 적용하기는 어렵다.

## 3. 클러스터링 기반 모바일 클라우드 컴퓨팅 메타 모델

현재까지 진행된 MCC에 대한 연구는 하나의 관리자가 전체 노드들을 관리하는 접근법을 사용하므로, 다음과 같은 기술적 이슈를 가지고 있다[1][2][3][5][6][7]. 동적으로 변화하는 모바일 클라우드 환경을 하나의 관리자가 모두 모니터링 하기 어렵고, 모바일 클라우드 환경에 다양한 요소들이 포함되어 있기 때문에, 요소들에 대한 최신 정보로 상태를 유지하는데 오버헤드가 증가한다. 이와 같은 기술적 이슈들을 해결하기 위해서 Fig. 1과 같은 메타 모델을 제안한다.

MCE는 크게 노드(Node), 클러스터(Cluster), MCE 관리자 3가지 요소로 구성된다. 노드는 어플리케이션이나 서비스를 설치하여 운영하는 물리적 컴퓨팅 디바이스로서, 모바일 노드, 클라우드 노드, 데스크톱 노드, 서버 노드로 구분할 수 있다.

MCE 관리자는 할당 노드의 관리 태스크를 담당하는 노

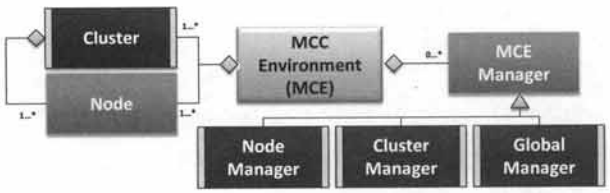


Fig. 1. MCE Meta Model

드 관리자, 클러스터 단위에서 필요로 하는 관리 태스크를 수행하는 클러스터 관리자, 전체 MCE의 관리 태스크를 담당하는 중앙 관리자로 구분된다. 클러스터 관리자는 해당 클러스터의 품질을 안정화하며, 중앙 관리자는 클러스터 간의 전체 MCE의 품질을 안정화한다.

클러스터는 노드 간 상호작용 빈도, 공유하는 데이터 집합 등의 관점에서 의존성 및 결합도가 높은 노드들을 그룹핑하여 관리하는 개념적인 구성단위이다. Fig. 2는 클러스터링으로 구성된 MCE의 구성도를 보여준다.

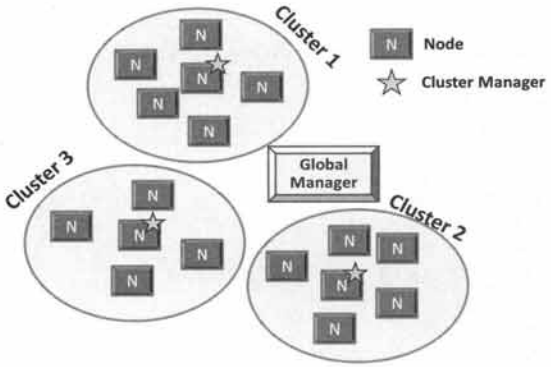


Fig. 2. Example of MCE Clustering

만약 위의 그림에서 클러스터 개념이 없다면, 중앙에 위치한 글로벌 관리자는 모든 노드들에 설치된 애플리케이션이나 서비스의 품질을 관리하지만, 이를 3개의 클러스터 매니저가 나누어 관리하므로, 중앙 관리자의 부하가 1/3으로 줄어들게 된다.

## 4. MCE에서의 동적 클러스터링 프로세스

MCE는 품질 관리하는 프로세스와 동적 클러스터링 프로세스가 서로 독립적으로 수행된다. Fig. 3은 MCE에서 수행되는 2개의 프로세스 간의 관계를 보여준다.

품질 관리 프로세스는 여러 노드 및 애플리케이션, 서비스에서 측정된 품질 데이터를 기반으로 MCE에서 발생하는 품질 관련 결함을 식별하며, 이를 해결하기 위한 여러 품질 향상 활동을 수행한다. 이런 품질 관리 프로세스를 수행하면서, 새로운 노드가 추가 및 삭제되는 등 MCE 형상에 다양한 변화가 발생하게 된다. 이런 변경된 형상 정보를 최적화로 유지하기 위해, 재클러스터링 (re-clustering)이 수행되어야 하며, 이를 위해 동적 클러스터링 프로세스가 수행된다. 이 두 프로세스는 MCE 전체 품질을 최적화된 상태로 자율적이며 효율적인 방법으로 유지할 수 있도록 한다.

최적화된 클러스터링을 구성하기 위한 기본 원칙은 다음과 같다. 클러스터 간의 의존도를 줄이고 독립성을 유지하여 관리 오버헤드를 줄이기 위해, 1) 각 클러스터 내의 노드들은 응집도가 높고, 2) 클러스터 간의 결합도는 최소화가 되어야 한다. 그리고, 3) 이용가능성을 포함한 클러스터 전반적인 품질이 사전 정의된 임계치(Threshold Value)보다 높은 상태를 유지해야 한다. 이 원칙을 만족시키기 위해 클러스터링 프로세스를 Fig. 4와 같이 정의한다.

활동 ①과 활동 ②는 품질 관리 프로세스를 수행하면서 수집된 정보를 형상 관리 저장소로부터 획득하는 단계이다. 형상 관리 저장소로부터 클러스터 간 결합도 및 응집도를 계산하기 위한 정보와 클러스터 내의 시간/자원 효율성 정보를 수집한다. 만약 이 정보들이 미리 정의한 임계치보다 낮으면, 현재 클러스터링을 다시 구성해야 할 필요성이 있다고 판단한다.

활동 ③에서는 클러스터링 알고리즘을 수행하는데 필요한 요소들에 대한 Lock을 획득한다. MCC에서는 품질 관리 프로세스와 클러스터링 프로세스가 동시에 수행되기 때문에, 클러스터링 프로세스를 수행하는 동안에 품질 관리 프로세

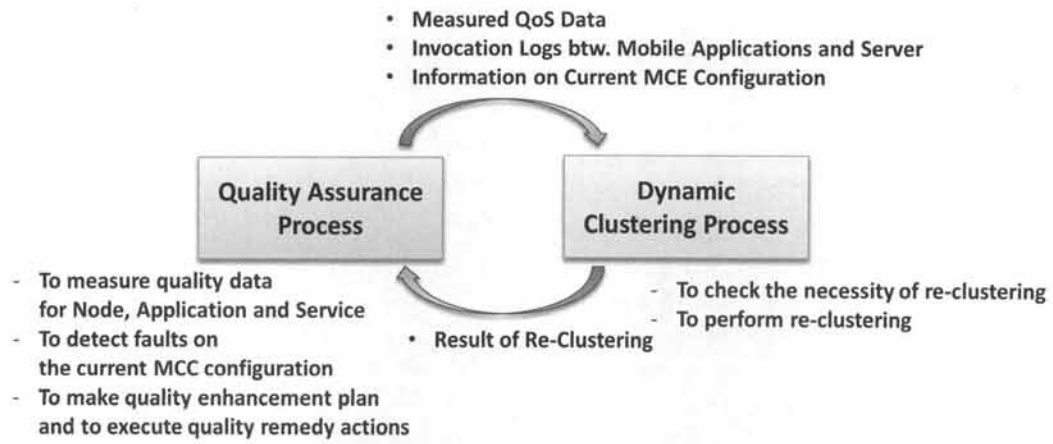


Fig. 3. Relationships between Two Processes in MCE

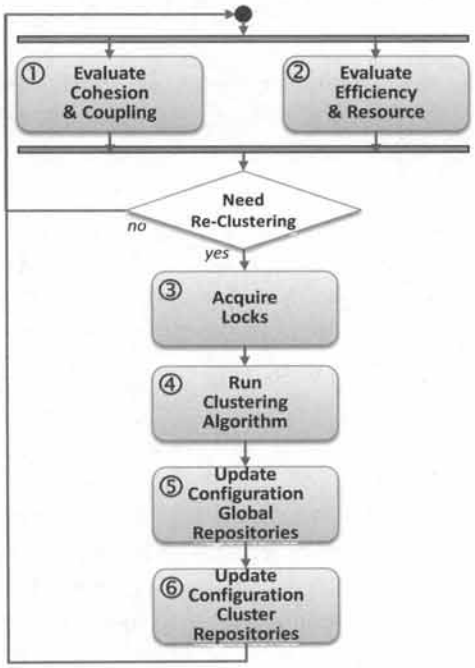


Fig. 4. Dynamic Clustering Process

스가 해당 자원을 접근하지 못하게 하기 위해서이다. Lock을 획득하면, 활동 ④에서는 5장에서 정의된 클러스터링 알고리즘을 수행하여, 현재 상황에 최적화된 클러스터링을 구성한다. 마지막으로 활동 ⑤와 활동 ⑥에서 변경된 형상 정보를 업데이트한다. 활동 ⑤의 경우, 클러스터링 알고리즘 수행 이후에 MCC 전체 클러스터에 대한 정보들이 업데이트 된다. 이후에 활동 ⑥을 통해서 클러스터 단위에서 변경이 발생할 경우에 클러스터의 정보들을 업데이트 한다.

## 5. MCE 노드간 논리적 근접도 (Logical Nearness) 측정 기법

### 5.1 논리적 근접도 측정의 요소

노드 간 논리적 근접도는 노드 사이의 물리적인 거리 외에 여러 기준을 이용하여 계산한 상대적인 가까움을 논리적으로 나타내는 척도이다. 노드간 논리적인 근접도를 구하기 위한 함수로서 Distance Function (DF)를 정의하기 위해 Table 1과 같은 요소 목록 (Factor List, FLIST)을 적용한다.

많은 연구에서 네트워크 대역폭이나 응답시간에 따라서 노드간 거리를 측정한다[1][2]. 본 논문에서는 FLIST1~FLIST5를 사용하여 두 노드간 상호작용 및 물리적 거리를 측정한다. 논리적 요소뿐만 아니라 물리적 요소를 바탕으로 논리적 근접도를 판단하여 하나의 클러스터에 할당하게 되면, 두 노드간 상호작용 성능을 향상시킬 수 있다. 따라서 Table 1과 같이 논리적 근접도 측정 요소를 선정하였다. 본 논문에서는 위에 기술된 일반적인 기준 5가지만 고려하였다. 만일 논리적인 거리에 영향을 주는 요소가 새롭게 발견되면 위 기준을 확장할 수 있다.

Table 1. Factor List for Distance Function

| DF Factor | Description | Unit |
|---|---|---|
| FLIST[1] | Geographical direct distance | Km |
| FLIST[2] | Declared network bandwidth between two nodes | Mbit/sec |
| FLIST[3] | Measured network bandwidth between two nodes | Mbit/sec |
| FLIST[4] | Frequency of invocations between two nodes | |
| FLIST[5] | Average response time between two nodes | Millisecond |

FLIST는 각각 다른 단위와 값 범위를 가지고 있기 때문에, 동일한 기준에서 비교하기가 어렵다. 그러므로, 0과 1사이의 값을 가지도록 다음과 같은 메트릭을 이용하여 정규화하는 과정이 필요하다. 정규화를 거친 모든 FLIST들은 1에 가까울수록 두 노드 사이가 논리적으로 멀리 떨어져 있음을 의미한다.

FLIST[1]은 노드간 물리적인 거리(Geographical Direct Distance, GDD)를 의미하며, 해당 노드가 MCE 환경에 등록될 때 형상 관리 저장소에 저장된 값으로, 맵 서비스를 이용해 측정된다. 그러므로, *GDD*를 측정하기 위한 별도의 수식은 없으며, 정규화된 값인 $GDD_{norm}$은 다음의 식을 이용하여 측정할 수 있다.

$$GDD_{norm} = FLIST[1]/MAX_{GDD}$$

여기서, $MAX_{GDD}$는 MCE의 형상 관리 저장소에 저장된 가장 큰 GDD 값을 의미한다.

FLIST[2]는 노드간 선언된 네트워크 대역폭(Declared Network Bandwidth, DNB)을 의미하며, 해당 노드가 MCE 환경에 등록될 때 형상 관리 저장소에 저장된다. *DNB*를 측정하기 위한 별도의 수식은 없으며, 정규화된 값인 $DNB_{norm}$은 다음의 식을 이용하여 측정할 수 있다.

$$DNB_{norm} = 1 - (FLIST[2]/MAX_{DNB})$$

여기서, $MAX_{DNB}$는 MCE의 형상 관리 저장소에 등록된 가장 큰 DNB을 의미한다.

FLIST[3]는 노드간 측정된 네트워크 대역폭(Measured Network Bandwidth, MNB)으로, MCE 형상 관리 저장소의 로그를 이용하여 다음의 식을 통해 측정할 수 있다.

$$MNB = \frac{\sum_{i=1}^{NumTX} \frac{Size(TX_i)}{TransTime(TX_i)}}{NumDataTXs}$$

이 값은 일정 시간 동안 발생한 네트워크 상의 데이터 전송에 대한 기록을 이용하여 측정된다. 여기서 $Size(TX_i)$는

해당 기간 동안 i번째로 전송된 데이터의 크기이며, $TransTime(TX_i)$는 $TX_i$가 전송되는 동안 소요되는 시간을 의미한다. 그리고, $NumDataTXs$는 관찰한 기간 동안 발생한 전체 데이터 전송횟수이다. 즉, $MNB$는 특정 기간 동안 발생한 전체 데이터 전송 속도의 평균 값을 의미한다.

이를 이용하여 정규화된 값인 $MNB_{norm}$은 다음의 식을 이용하여 측정한다.

$$MNB_{norm} = 1-(FLIST[3]/MAX_{MNB})$$

여기서, $MAX_{MNB}$는 MCE 형상 관리 저장소에서 관리하고 있는 가장 큰 MNB 값을 의미한다.

FLIST[4]는 노드간 호출 빈도(Frequency of Invocations, FOI)를 의미하며, MCE 형상 관리 저장소의 로그를 통해서 얻을 수 있다. MCE 형상 관리 저장소의 호출 관련 로그를 바탕으로 FOI 값을 다음의 식을 이용하여 측정할 수 있다.

$$FOI = NumInv(App_i, S_{j^*})$$

여기서 $App_i$는 i번째 애플리케이션이며, $S_{j^*}$는 j번째 서비스 타입의 k번째 인스턴스를 의미한다. $NumInv(App_i, S_{j^*})$는 특정 기간 동안의 $App_i$와 $S_{j^*}$간의 호출 빈도수를 의미한다. 이를 이용하여 정규화된 값인 $FOI_{norm}$은 다음의 식을 이용하여 측정한다.

$$FOI_{norm} = 1-(FLIST[4]/MAX_{FOI})$$

여기서, $MAX_{FOI}$는 MCE 형상 관리 저장소에서 관리하고 있는 가장 큰 FOI 값을 의미한다.

FLIST[5]는 노드간 평균 반응 시간(Average Response Time, ART)로, 일정 시간 동안의 서비스 호출시 측정되는 반응 시간의 평균 값이다. MCE 형상 관리 저장소의 호출 관련 로그로부터 다음의 식을 이용하여 측정할 수 있다.

$$ART = \frac{\sum_{x=1}^{NumInv(App_i, S_{j^*})} RT(INV(App_i, S_{j^*})_x)}{NumInv(App_i, S_{j^*})}$$

여기서, $RT(INV(App_i, S_{j^*})_x)$는 두 노드간 x번째 호출의 반응 시간을 의미한다. 이를 이용하여 정규화된 값인 $ART_{norm}$은 다음의 식을 이용하여 측정한다.

$$ART_{norm} = FLIST[5]/MAX_{ART}$$

여기서, $MAX_{ART}$는 MCE 형상 관리 저장소에서 관리하고 있는 가장 큰 값을 의미한다.

### 5.2 논리적 근접도 측정 함수, DF( )

DF는 노드 간 논리적 근접도를 위한 함수로서, 각 FLIST 값을 이용하여 다음의 식을 통해 계산된다.

$$DF(NODE_i, NODE_j) = \frac{\sum_{i=1}^{numFLIST} FLIST[i] \times W_i}{numFLIST}$$

도메인마다 FLIST의 중요도가 다르기 때문에, FLIST마다 각각 다른 가중치($W_i$)를 지정하여 논리적 근접도 DF를 계산한다. 예를 들어, 군사 부분의 경우에는 네트워크의 지속적인 연결이 어려울 수 있는데, FLIST[3]에 $W_i$를 높게 책정하여 DF를 구한다. 반면, 가정에 있는 네트워크의 경우 일반적으로 항상 연결되어 있기 때문에 FLIST[3]에 $W_i$를 낮게 책정하여 DF를 구하도록 하여 도메인마다 차별화를 둘 수 있다.

## 6. k-means 기반 동적 클러스터링 기법

### 6.1 최적화 알고리즘

기존 *k-means* 는 클러스터링에 사용되는 대표적인 알고리즘이지만[8][9], MCE 환경에 적용하기에는 Table 2와 같은 차이점을 가지고 있다.

Table 2. Differences between original k-means and k-means in MCE

| Comparison Criteria | k-means | k-means in MCE |
|---|---|---|
| Data used in Clustering | Distance between two nodes | Relative Distances between two nodes |
| How to Choose Centroid | The centroid does not need to be a node. It can exist between any two nodes. | A node is chosen as a centroid. |
| Whether to Estimate Clustering Results | None | Performed based on DF |

먼저, 기존의 *k-means*는 두 노드간 물리적 거리를 기반으로 클러스터링을 수행하지만, MCE에서는 두 노드간 상대적인 논리 거리를 기반으로 클러스터링이 이루어진다. 둘째, 기존 *k-means*에서는 Centroid가 노드의 위치에 상관없는 임의의 공간에 위치할 수 있으나, MCE에서는 클러스터되는 노드 중 하나가 Centroid 역할을 해야 한다. 만약 기존의 *k-means*를 그대로 적용한다면, 임의의 공간에 위치한 Centroid에 새로운 노드를 설치 해야 하는 상황이 발생한다. 이는 불필요한 노드를 추가해야 하고, 해당 노드를 초기화하는 과정 때문에 실행시간 및 자원 활용 관점에서 효과적

이지 못하다. 셋째, MCE 환경에서 클러스터링은 클러스터 관리자의 재배치, 클러스터 관리자가 관리하는 형상 관리 저장소 업데이트, 중앙 관리자가 관리하는 형상 관리 저장소 업데이트 등 전반적인 정보 변경을 수반하므로, 최적화 상태로 클러스터링이 되어 있음을 반드시 확인해야 한다.

Table 2에 보인 *k-means*와 MCC에서의 *k-means*의 차이 때문에 본 논문에서는 *k-means*알고리즘을 MCC환경에 적용하도록 다음을 반영하여 동적 클러스터링 기법에 사용한다.

- $k$ 개의 초기 centroids의 위치를 노드 중 하나로 지정하여 클러스터링되도록 한다.
- 각 노드들 간의 논리적 근접도를 기반으로 클러스터링 한다.
- 클러스터 관리자와 다른 노드간 논리적 근접도를 통해 해당 클러스터의 최적화 정도를 측정 한다.

위와 같은 기준을 적용함으로써, Table 3과 같이 기존 *k-means*알고리즘을 MCC환경에 맞게 수정한다. 위의 기준은 각각 라인 6, 라인 7, 라인 12에 반영되었다.

Table 3. Modified k-means Clustering Algorithm

| | |
|---|---|
| 1 | Begin |
| 2 | // Initialize |
| 3 | Enter '$k$'. |
| 4 | **// Set initial centroids by weight** |
| 5 | **Enter a set of node, NODE = {$x_1$, $x_2$, …, $x_n$}.** |
| 6 | **Enter a set of initial k centroids, CEN_NODE = { $x_i$ \| $x_i$ NODE } and n(CEN_NODE) = k** |
| 7 | **Calculate logical closeness between nodes in NODE** |
| 8 | // Main Loop |
| 9 | Repeat |
| 10 | Assign $x_i$ to the cluster with the closest centroid. |
| 11 | **// Update current centroid, after estimate if it is optimum** |
| 12 | **Update the centroid of each cluster.** |
| 13 | Until the centroids do not change. |
| 14 | // Print output |
| 15 | Return the resulting set of clusters, {$C_1$, $C_2$, …, $C_k$}. |
| 16 | End |

*k-means*를 개선하기 위해 추가된 3가지 기준을 만족시키기 위해서 Table 3의 라인 6에 의해서 할당된 $k$ 개의 초기 centroids가 선정된다. 라인 7에서 노드간 근접도를 산출하고, 산출된 근접도를 바탕으로 라인 12에서 최적의 Centroid를 탐색하여 업데이트하게 되면 해당 클러스터의 최적화 정도를 측정한다. *k-means*의 단점 중 하나는 $k$의 위치에 따라서 결과가 달라질 수 있다는 것이다. 본 논문에서 제안하는 클러스터링 기법도 *k-means*를 개선한 기법이기 때문에 이러한 문제점을 가지고 있다. 하지만 본 논문의 기법에서는 어떠한 $k$, 즉 클러스터 관리자를 어떠한 노드로 선정하더라도 MCE 수준의 최적화는 정도의 차이를 보일 수는 있지만, 최초 노드 선정시 1)도메인에 따른 가중치를 바탕으로 선정하거나, 2) 선정된 노드를 반복적으로 평가하여 최적의 노드를 찾는 방법을 사용하여 기법 적용 전에 비해 최적화를 이루도록 함으로써 *k-means*의 단점을 최대한 보완하고 있다.

### 6.2 설계 모델

본 절에서는 Table 3의 알고리즘을 위한 설계 모델을 보여준다. Fig. 5는 클러스터링을 위하여 노드에 대한 정보를 MCE 형상 관리 저장소로부터 할당하는 부분부터 클러스터링 결과를 업데이트 하는 부분까지의 시퀀스 다이어그램이다.

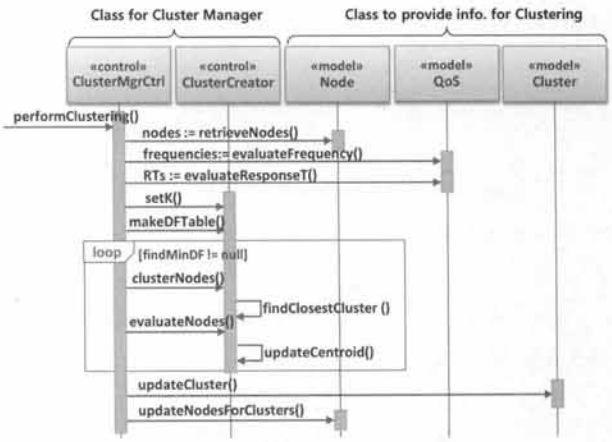


Fig. 5. Sequence Diagram for Clustering

클러스터링에는 총 5개의 클래스가 참여한다. *ClusterMgrCtrl*은 클러스터 관리자가 수행하는 전체 태스크를 관리하고, *ClusterCreator*는 클러스터링에 관련된 일을 수행한다. 오른쪽에 위치한 *QoS, Node, Cluster*클래스는 논리적 거리를 계산하는데 필요한 정보를 제공한다.

먼저, *ClusterMgrCtrl*는 클러스터링을 위해서 *Node*를 MCE 형상 관리 저장소로부터 데이터를 수집하여, *QoS*클래스를 통해서 노드간 상호작용 빈도 및 반응 시간 등을 평가한다. 그리고, *ClusterCreator*는 노드 간 논리적 근접도를 나타내는 DF 테이블을 생성하고, 새로운 클러스터를 구성한다. 마지막으로, 변경된 정보가 업데이트된다.

### 6.3 구현 결과

본 논문에서 제안하는 기법의 핵심인 클러스터링과 각 클러스터의 노드를 평가하는 메소드를 구현한다. Table 4은 지정된 노드와의 논리적 근접도를 바탕으로 그룹화하는 코드 일부를 보여준다.

Table 5는 클러스터링 이후, 지정된 클러스터 관리자 노드가 최적인지 다른 노드들의 평균 DF와 비교하는 과정을 보여준다.

Table 4. Code Snippet for Clustering Nodes

```
1  private static void clusterNodes(int nodeID, Node
       nodes[], DF DFTable[][], HashMap<Integer,
       ArrayList<Node>> clusters) throws
       CloneNotSupportedException {
2    ArrayList<Node> list = null;
3    DF df = null;
4    Node currentNode = null;
5    currentNode = (Node) nodes[nodeID].clone();
6    Object[] key = clusters.keySet().toArray();
7    DF dfList[] = new DF[clusters.size()];
8    for (int i = 0; i < clusters.size(); i++) {
9      df = DFTable[nodeID][(Integer)key[i]];
10     dfList[i] = DFTable[nodeID][(Integer)key[i]];
11   }
12   // Search an optimal centroid
13   DF minDF = findClosestCluster(dfList);
14   // Set a cluster and add a node
15   list = clusters.get(minDF.getClusterManager
         Node().getId());
16   if(df.getDf() != null) {
17     currentNode.setDf(minDF.getDf());
18     list.add(currentNode);
19   }
```

Table 5. Code Snippet for Measuring DF

```
1  public DF evaluateNodes(DF DFTable[][], Node
       nodes[]) {
2   DF minDF = null;
3   DF dfList[] = new DF[DFTable.length];
4   // Calculate average of DF
5   for (int i = 0; i < nodes.length; i++) {
6     DF df = new DF();
7     df.setDf(calculateAVGDF(i, DFTable));
8     df.setClusterManagerNode(nodes[i]);
9     dfList[i] = df;
10  }
11  // Update a centroid to optimal one
12  minDF = updateCentroid(dfList);
13  return minDF;
14 }
```

5~10라인을 통해 현재 클러스터의 각 노드마다 평균 DF를 구하고, 12라인을 통해 평균 DF가 최소인 노드를 탐색한다. 탐색된 노드가 해당 클러스터의 최적의 클러스터 관리자로 선택된다.

## 7. MCC 최적화 평가 모델

동적 클러스터링은 MCC 환경의 관리 품질에 중대한 영향을 미치므로, 클러스터링이 제대로 이루어졌는지를 확인해야 한다. 본 장에서는 제안된 클러스터링 기법을 적용한 MCC의 최적화 정도를 평가하기 위해서 정량적인 측정이 가능한 품질 메트릭을 정의한다.

품질 메트릭에 사용되는 용어(Term)와 함수를 다음과 같이 정의한다.

- *MCE*: MCC를 적용하여 구성된 도메인의 전체 환경을 나타낸다.
- $CLUSTER_i$: *MCE*에 속한 $i$ 번째 클러스터를 나타낸다.
- *NumClusters*: *MCE*를 구성하는 모든 클러스터의 개수로서, $k$와 동일하다.
- $NODE_i$: *MCE*에 참여하는 $i$ 번째 노드를 나타낸다.
- $CENT\_NODE_i$: $i$번째 클러스터인 $CLUSTER_i$의 논리적 중앙 노드를 나타낸다.
- *NumNodes*: *MCE*에 속한 모든 노드들의 개수이다. 즉, $MCE = \{NODE_1, NODE_2, \cdots, NODE_{NumNodes}\}$ 이다.
- $SizeCluster(CLUSTER_i)$: 특정 클러스터 $CLUSTER_i$에 속한 노드들의 개수를 반환하는 함수이다.
- $SetNodes(CLUSTER_i)$: 특정 클러스터 $CLUSTER_i$에 속한 모든 노드들의 집합을 반환하는 함수이다.
- $DF(NODE_i, NODE_j)$: 두 개의 노드인 $NODE_i$와 $NODE_j$ 사이의 논리적 근접도를 계산하는 함수이다.

MCC의 최적화 정도는 1) 각 클러스터가 최적으로 구성되었는지 (클러스터 수준의 최적화 정도)와 2) 전체 MCC 환경이 최적으로 구성되었는지(*MCE* 수준의 최적화 정도)를 평가함으로써 측정된다.

먼저, 클러스터 수준의 최적화 정도는 해당 클러스터가 높은 응집도를 가지도록 구성되었는지를 평가한다. 이를 위해, 다음과 같이 클러스터 수준의 최적화 정도($QoCluster(CLUSTER_i)$)를 정의한다.

$$QoCluster(CLUSTER_i) = \frac{\sum_{NODE_x \in SetNodes(CLUSTER_i)} DF(CENT\_NODE_i, NODE_x)}{SizeCluster(CLUSTER_i)}$$

위와 같이, $QoCluster(CLUSTER_i)$는 $CENT_NODE_i$와 $CLUSTER_i$를 구성하는 노드들 간의 DF 값들의 평균이며, 0과 1사이의 값을 반환한다. 논리적으로 가까운 노드들끼리 클러스터를 구성할수록 *QoCluster*는 0에 가까운 값을 반환할 것이다.

*MCE* 수준의 최적화 정도는 현재 구성된 클러스터들이 서로 결합도가 낮은지를 평가함으로써 측정된다. 클러스터 간에 상호작용이 적다는 것은 논리적으로 멀리 떨어져 있는 노드들끼리 상호작용을 적게 한다는 것을 의미하며, 이는 *MCE*의 전체 품질을 효과적으로 관리하고 있음을 나타낸다. *MCE* 수준의 최적화 정도를 평가하기 위해, *QoMCE* 함수를 다음과 같이 정의한다.

$$QoMCE = \frac{\sum_{i=1}^{NumClusters} QoCluster(CLUSTER_i)}{NumClusters}$$

위와 같이 *MCE* 수준의 최적화 정도는 현재 *MCE*를 구성하는 클러스터들의 *QoCluster* 값의 평균이며, 0과 1사이의 값을 반환한다. 클러스터 내의 응집도가 높고 클러스터 간 결합도가 낮을수록 *QoMCE*는 0에 가까운 값을 반환할 것이다.

## 8. 실험 및 평가

본 장에서는 5장에서 제시된 논리적 근접도 측정 기법과 6장에서 제시된 클러스터링 기법의 효율성과 적용성을 MCC 환경에 대한 시뮬레이션 실험을 통하여 검증한다. 실험 결과는 7장에서 제시된 최적화 평가모델을 이용하여 평가하고, FLIST 마다 동일한 가중치를 선정하여 실험을 진행하였다.

### 8.1 실험 환경 설정

시뮬레이션 실험은 Windows 7에서 Java와 Eclipse로 구현하였고, Table 6와 같이 2개의 MCC 환경 즉, MCE에 대하여 5가지의 실험을 수행하였다. 각 실험은 다시 20회의 실행을 통하여 평균치를 구하였다.

실험 1은 클러스터링을 적용하지 않은 경우를 나타내며, 실험 2는 실험 1의 결과와 비교하여 클러스터링의 효율성을 확인하기 위해서 수행한다. 실험 3은 동일한 조건이나 초기 Centroid 노드를 달리 설정하여 실험 2와 클러스터링 결과에 어떤 차이가 있는지를 확인한다. 실험 4와 실험 5는 노드 수와 클러스터 수에 따른 클러스터링 효율성을 확인한다.

### 8.2 실험 진행 과정

본 절에서는 실험 2의 진행과정의 중간 결과값들을 제시된 클러스터링 알고리즘과 관련하여 설명하고, 최적화 평가모델을 적용한다. 실험 데이터는 총 20개에 대해서 노드간의 논리적 근접도를 난수(Random Number)를 이용하여 구하고 Fig. 6과 같이 DF 테이블을 생성하였다.

예를 들면, 노드 7과 노드 10간의 DF는 0.67을 나타낸다. 실제 *MCE* 운영시에는 5.1절에서 정의한 FLIST의 요소들에 기반으로 노드간의 호출 등의 활동을 모니터링하여 계산된다.

이 DF 테이블을 이용하여 6장에서 제시된 클러스터링 알고리즘을 적용하였고, 3번의 Iteration에 걸쳐서 클러스터링이 수행되었다. 첫 Iteration의 결과는 Centroid 노드인 1, 3,

Table 6. Initial Values for Experiments

| Factors \ Experiments | MCE #1 | | | MCE #2 | |
|---|---|---|---|---|---|
| | Case 1 | Case 2 | Case 3 | Case 1 | Case 2 |
| NumNodes | 20 | 20 | 20 | 100 | 100 |
| NumClusters | 1 | 3 | 3 | 3 | 10 |
| Initial Centroids | 7 | 1, 3, 15 | 10, 15, 19 | 1, 5, 10 | 2, 16, 23, 35, 39, 40, 47, 55, 89, 92 |

| | [0] | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [9] | [10] | [11] | [12] | [13] | [14] | [15] | [16] | [17] | [18] | [19] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [0] | null | 0.81 | 0.11 | 0.53 | 0.68 | 0.06 | 0.99 | 0.1 | 0.93 | 0.41 | 0.58 | 0.82 | 0.91 | 0.37 | 0.06 | 0.18 | 0.37 | 0.73 | 0.36 | 0.49 |
| [1] | | null | 0.95 | 0.69 | 0.85 | 0.78 | 0.91 | 0.66 | 0.36 | 0.34 | 0.08 | 0.74 | 0.49 | 0.12 | 0.88 | 0.96 | 0.36 | 0.88 | 0.47 | 0.78 |
| [2] | | | null | 0.72 | 0.24 | 0.64 | 0.25 | 0.52 | 0.53 | 0.81 | 0.98 | 0.28 | 0.74 | 0.17 | 0.64 | 0.48 | 0.46 | 0.86 | 0.82 | 0.42 |
| [3] | | | | null | 0.79 | 0.33 | 0.72 | 0.26 | 0.87 | 0.82 | 0.82 | 0.54 | 0.54 | 0.4 | 0.84 | 0.49 | 0.7 | 0.98 | 0.39 | 0.57 |
| [4] | | | | | null | 0.22 | 0.9 | 0.21 | 0.65 | 0.84 | 0.52 | 0.72 | 0.83 | 0.42 | 0.38 | 0.42 | 0.79 | 0.79 | 0.71 | 0.06 |
| [5] | | | | | | null | 0.26 | 0.02 | 0.86 | 0.82 | 0.66 | 0.65 | 0.79 | 0.31 | 0.51 | 0.61 | 0.11 | 0.9 | 0.03 | 0.53 |
| [6] | | | | | | | null | 0.48 | 0.92 | 0.65 | 0.27 | 0.39 | 0.68 | 0.86 | 0.29 | 0.04 | 0.84 | 0.94 | 0.3 | 0.25 |
| [7] | | | | | | | | null | 0.78 | 0.54 | 0.67 | 0.03 | 0.29 | 0.21 | 0.64 | 1 | 0.16 | 0.8 | 0.68 | 0.56 |
| [8] | | | | | | | | | null | 0.46 | 0.41 | 0.51 | 0.36 | 0.33 | 0.8 | 0.62 | 0.21 | 0.28 | 0.64 | 0.3 |
| [9] | | | | | | | | | | null | 0.96 | 0.29 | 0.12 | 0.67 | 0.17 | 0.53 | 0.03 | 0.71 | 0.92 | 0.92 |
| [10] | | | | | | | | | | | null | 0.61 | 0.75 | 0.06 | 0.42 | 0.57 | 0.39 | 0.76 | 0.95 | 0.82 |
| [11] | | | | | | | | | | | | null | 0.24 | 0.46 | 0.4 | 0.84 | 0.55 | 0.28 | 0.14 | 0.82 |
| [12] | | | | | | | | | | | | | null | 0.58 | 0.22 | 0.57 | 0.07 | 0.05 | 0.2 | 0.54 |
| [13] | | | | | | | | | | | | | | null | 0.75 | 0.81 | 0.21 | 0.08 | 0.17 | 0.66 |
| [14] | | | | | | | | | | | | | | | null | 0.49 | 0.13 | 0.47 | 0.54 | 0.57 |
| [15] | | | | | | | | | | | | | | | | null | 0.11 | 0.87 | 0.79 | 0.94 |
| [16] | | | | | | | | | | | | | | | | | null | 0.87 | 0.19 | 0.79 |
| [17] | | | | | | | | | | | | | | | | | | null | 0.65 | 0.58 |
| [18] | | | | | | | | | | | | | | | | | | | null | 0.17 |
| [19] | | | | | | | | | | | | | | | | | | | | null |

Fig. 6. DF Table

| Cluster ID | Centroid | *SetNodes* ($CLUSTER_i$) | *SizeCluster* ($CLUSTER_i$) | *QoCluster* ($CLUSTER_i$) |
|---|---|---|---|---|
| $CLUSTER_1$ | 1 | {1,8,9,10,12,13} | 6 | 0.23 |
| $CLUSTER_2$ | 3 | {3,5,7,11,18,19} | 6 | 0.23 |
| $CLUSTER_3$ | 15 | {0,2,4,6,14,15,16,17} | 8 | 0.31 |
| | | | QoMCE | 0.26 |

| Cluster ID | Centroid | *SetNodes* ($CLUSTER_i$) | *SizeCluster* ($CLUSTER_i$) | *QoCluster* ($CLUSTER_i$) |
|---|---|---|---|---|
| $CLUSTER_1$ | 1 | {1,8,10,13} | 4 | 0.13 |
| $CLUSTER_2$ | 18 | {3,5,11,12,18,19} | 6 | 0.16 |
| $CLUSTER_3$ | 14 | {0,2,4,6,7,9,14,15,16,17} | 10 | 0.33 |
| | | | QoMCE | 0.20 |

| Cluster ID | Centroid | *SetNodes* ($CLUSTER_i$) | *SizeCluster* ($CLUSTER_i$) | *QoCluster* ($CLUSTER_i$) |
|---|---|---|---|---|
| $CLUSTER_1$ | 13 | {1,2,7,8,10,13,17} | 7 | 0.16 |
| $CLUSTER_2$ | 18 | {3,5,11,12,18,19} | 6 | 0.14 |
| $CLUSTER_3$ | 14 | {0,4,6,9,14,15,16} | 7 | 0.12 |
| | | | QoMCE | 0.17 |

15번을 기준으로 다음과 같은 클러스터링 결과가 도출되었다. 전체 *MCE*의 최적화를 나타내는 *QoMCE*는 0.26으로 계산되었다.

첫 번째 클러스터링 결과를 기반으로 새로운 3개의 Centroid 노드로 {1, 18, 14}가 결정되었고, 두 번째 Iteration이 실행되었으며, 그 결과는 다음과 같다. *QoMCE*는 0.20으로서 첫 Iteration때의 0.26보다 줄어들었고, 이는 클러스터링이 더 최적화 되었음을 나타낸다.

두 번째 클러스터링 결과를 기반으로 새로운 3개의 Centroid는 {13, 18, 14}로 업데이트 되었으며, 세 번째 Iteration을 적용하였고, 세 번째 클러스터링의 결과는 다음과 같다.

이 결과를 기반으로 새로운 Centroid를 구하였으나, 기존 13, 18, 14 보다 평균 DF가 낮은 노드가 없으므로 클러스터링은 종료된다. 세 번에 Iteration을 걸친 클러스터링 결과를 종합하면 Fig. 7과 같다.

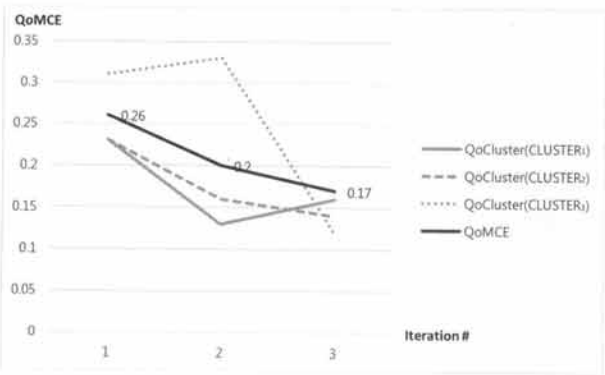


Fig. 7. Results of Clustering with 3 Iterations

수평 축의 1, 2, 3은 Iteration 번호를 나타내며, 수직 축은 최적화 정도를 나타낸다. 클러스터링 Iteration 1, 2, 3은 0.26, 0.2, 0.17의 순서대로 줄어들어, 전체 효율성이 향상되었음을 확인할 수 있다.

각각의 *QoCluster*가 Iteration 마다 증가하거나 감소 할 수는 있지만, 최종적으로 *QoMCE*는 Iteration 이 진행됨에 따라서 점차 0에 가까운 값을 보임으로써 효율성이 향상되고 있음을 알 수 있다.

### 8.3 실험 결과 및 평가

Table 6에서 5종류의 실험을 각각 20회 수행한 결과를 정리하면 Table 7와 같다.

본 절에서는 Table 6에 대한 총 5가지 실험을 통해 6가지 관점에서 평가한다.

• 클러스터링 적용 여부에 따른 *QoMCE* 차이

실험 1에서 클러스터링을 적용하지 않고 클러스터가 하나일 경우를 측정하고 실험 2에서는 클러스터링을 적용한 경우를 각각 측정하여 다음 Fig. 8과 같은 결과를 얻었다. 가로축은 실험 횟수, 세로축은 *QoMCE*를 의미한다.

클러스터링을 하지 않은 실험 1의 경우 *QoMCE*가 0.27~0.41까지 측정되었으나, 클러스터링을 적용한 실험 2의 경우 *QoMCE*는 0.11~0.24까지 측정되었다. 이 결과 클러스터링을 적용 여부에 따라서 *QoMCE* 차이가 크게는 약 4배

Table 7. Experiment Results

| Factors \ Experiments | MCE #1 | | | MCE #2 | |
|---|---|---|---|---|---|
| | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 |
| NumNodes | 20 | 20 | 20 | 100 | 100 |
| NumClusters | 1 | 3 | 3 | 3 | 10 |
| QoMCE for average of executed 20 times | 0.35 | 0.17 | 0.18 | 0.23 | 0.08 |
| Standard Deviation | 0.033 | 0.039 | 0.038 | 0.018 | 0.005 |

까지 나는 것을 확인 할 수 있다. 즉, 클러스터링을 적용하면 *MCE*에 존재하는 노드들이 근접한 노드들끼리 클러스터링 되어 효율성이 증가 하게 될 것을 예상 할 수 있다.

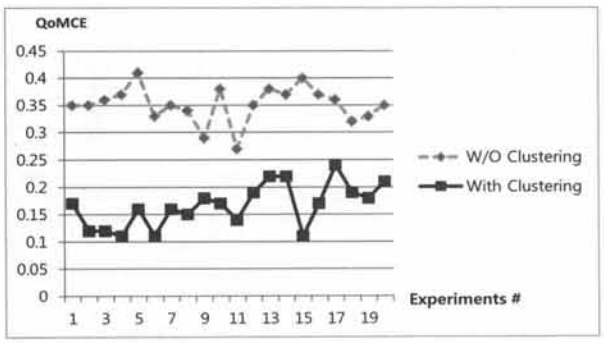


Fig. 8. Comparison between Without Clustering and With Clustering

• 최초 클러스터 관리자 노드(Centroid Set) 결정에 대한 차이

실험 2와 실험 3에서는 동일한 노드 20개를 3개의 클러스터로 클러스터링 할 때 최초 Centroid set에 따라서 어떤 차이가 있는지 평가한다. 각 실험 결과 측정된 *QoMCE*는 실험 2가 0.17, 실험 3이 0.18이다. 이 수치는 각각 동일한 Centroid Set을 가지고 20회씩 측정한 결과의 평균값이다. 본래 *k-means* 알고리즘은 최초 *k*를 무엇으로 선정하는지에 따라서 결과가 달라지지만, 본 논문에서는 어떠한 노드가 Centroid로 선정되더라도 *QoMCE*를 감소시켜 *MCE*를 최적화하도록 하여 *k-means*의 단점을 일부 보완하고 있다.

• *MCE*의 *NumNodes*에 따른 *QoMCE* 차이

실험 3과 실험 4를 통해 MCE에 클러스터링을 적용할 때 *NumNodes*가 어떤 영향을 미치는지 차이를 확인한다. 두 실험에 대해서 측정된 *QoMCE*를 보면 실험 3은 0.18, 실험 4는 0.23으로서 0.05정도의 차이를 보인다. 특히, 실험 4의 표준 편차는 0.018로서 실험 3의 0.038보다 0.020만큼 작게 측정되었는데, 이는 실험 4가 실험 3에 비해서 결과 값이 비교적 고르게 산출 되었다는 의미이다. 실험 3에 비해 실험 4에서의 *QoMCE*가 증가한 이유는 *NumNodes* 증가로 인해서 각 클러스터에 대한 *SizeCluster*가 증가하여, 동시에 *QoCluster*도 증가한 것이다. 즉, *NumNodes*의 증가는 *QoCluster*의 값을 증가시켜 *MCE*가 관리해야 할 노드가 늘어나서 *QoMCE*를 증가시키는 결과를 가져온다.

• *MCE*의 *NumClusters*에 따른 *QoMCE* 차이

실험 4와 실험 5를 통해서 *MCE*에 클러스터링을 적용할 때 클러스터링 개수가 어떤 영향을 미치는지 차이를 확인한다. 실험 4와 실험 5는 동일한 노드를 각각 3개와 10개로 클러스터링 할 때 차이를 보이고 있다. *QoMCE*가 실험 4는 0.23이지만 실험 5는 0.08로서 약 3배 차이가 난다. *NumClusters*의 분명한 상관 관계를 확인하기 위해 *NumClusters*를 1부터 10까지 증가 시켰을 때 *QoMCE*를 측정하는 실험을 추가적으로 수행하였고, Fig. 9는 이에 대한 결과를 보여준다. Fig. 9에서 *MCE*에 클러스터링 적용 시 *NumClusters*가 늘어나면 반비례하여 *QoMCE*가 감소되며 각 클러스터에 존재하는 노드들이 서로 밀접한 관계임을 알 수 있다. 즉, *NumClusters*가 증가함에 따라서 각 클러스터가 관리해야 할 노드가 줄어들게 되어 *QoMCE*가 0에 가까워 지는 것을 알 수 있다.

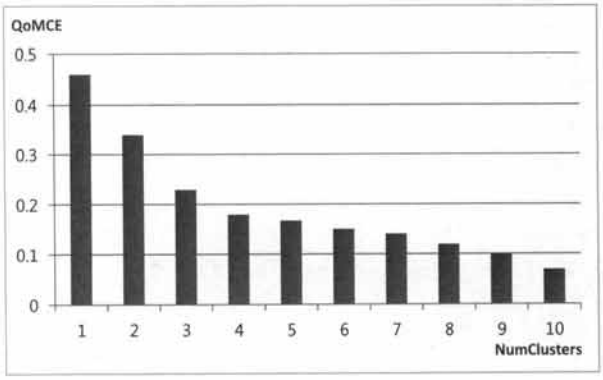


Fig. 9. QoMCE for Experiments #3 and #4

• 클러스터링 알고리즘 평가

본 논문에서 제안하는 기법을 사용하여 8.2절의 실험 2의 진행과정을 통해 *QoMCE*가 클러스터링이 진행 될수록 감소하는 것을 확인하였고, Table 6의 5가지 실험을 통해서 노드 및 클러스터 수가 달라지는 것과 같이 여러 상황에 적용 가능한 기법임을 확인 하였다. 그리고 Fig. 8을 통해서 클러스터링을 적용 하였을 경우 QoMCE가 더 안정적이 되는 것을 증명하였다.

• 근접도 측정 요소에 대한 평가

기존 연구들에서는 노드간 근접도 측정을 위해 네트워크 대역폭을 주요 요소로 설정하고 있다[1][2]. 하지만 본 논문에서는 FLIST로서 다양한 요소를 설정하여 클러스터링을 적용하였다. 따라서 네트워크 대역폭 만을 이용하여 근접도 측정을 한 경우와, FLIST를 이용하여 근접도를 측정한 경우와 비교를 위하여 추가 실험을 진행하였다. Fig. 10은 클러스터링을 수행하지 않은 경우, 네트워크 대역폭 기반으로 클러스터링을 수행한 경우, 본 논문에서 제시한 기법을 이용하여 클러스터링을 수행한 경우의 QoMCE 값을 비교하여 보여준다.

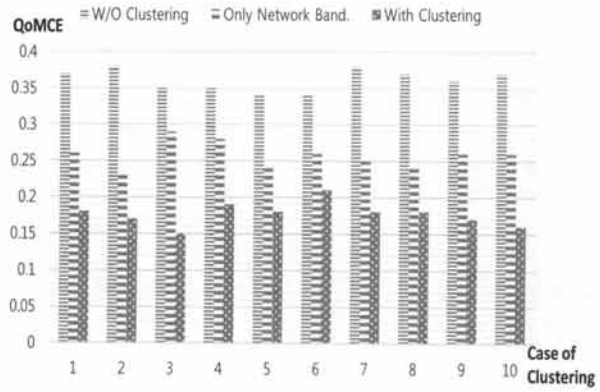


Fig. 10. Comparison Results of 3 Clustering Cases

클러스터링을 수행하지 않은 경우 전반적으로 QoMCE 값이 가장 낮으며, 네트워크 대역폭 외에 여러 기준을 기반으로 클러스터링을 수행한 경우 더욱 효과적으로 클러스터링이 이루어짐을 확인할 수 있다.

• MCC 최적화 모델 평가

기존의 *k-means* 알고리즘을 7장의 최적화 평가모델을 이용하여 클러스터링 결과의 최적화 정도를 평가하도록 수정하였고, 실험 환경을 구축하여 *QoMCE*를 측정하였다. 7장에서 제안하는 평가 모델이 각 실험을 통해서 나타난 결과에 의해서 MCE 최적화 정도를 효과적으로 나타낼 수 있음을 확인하였다.

## 9. 결 론

제한된 모바일 디바이스의 자원을 해결하기 위해, 클라우드에 있는 서비스 또는 자원을 활용하는 MCC 연구가 활발히 진행되고 있다. MCC 노드 수, 모바일 애플리케이션 수, 모바일 애플리케이션들이 사용하는 서비스의 수가 많아지거나 실행빈도가 잦아질 경우, 기존의 중앙 관리자를 통한 서비스 실행 관리는 병목현상으로 인해 성능저하를 유발시킬 수 있는 근본적인 문제를 가지고 있다.

본 논문에서는 중앙 관리자의 관리 오버헤드를 줄이기 위해, MCC에서는 의존성 및 응집도가 높은 노드들을 묶은 관리 단위인 클러스터라는 개념을 소개하였다. 먼저, MCC의 메타모델을 제시하여 클러스터를 지원하기 위한 핵심 요소들을 설명하고, MCC에서 클러스터링이 실행되는 6개의 활동으로 구성된 프로세스를 제시하였다. 마지막으로 실험을 통해, 클러스터링을 적용하면 관리 성능이 향상되고, 제시된 동적 클러스터링 알고리즘을 구성하는 기준인 논리적 거리를 계산하는 메트릭을 정의하였고, 각 메트릭을 계산하는 방법도 정의하였다. 그리고 논리적 거리를 이용하도록 *k-means* 를 수정하여 MCC 클러스터링을 구성하는 기법을 제안하였으며, 클러스터링 결과가 최적으로 진행되었는지를 평가하는 2개의 MCC 최적화 평가 메트릭을 알고리즘을 통해 최적화된 클러스터링이 구성됨을 확인하였다. 본 논문에서 제시된 기법을 이용하면 클러스터링은 최적화 상태로 구성되고, MCC 의 품질 관리 문제를 해결함과 동시에 관리 오버헤드를 줄이면서 효율적으로 품질 관리를 할 수 있다.

## 참 고 문 헌

[1] M. Fernando, S.W. Loke, and W. Rahayu, "Mobile Cloud Computing: A Survey," *Future Generation Computer Systems*, Vol.29, pp.84-106, 2012, doi: 10.1016/j.future.2012.05.023.

[2] L. Guan, X. Ke, M. Song, and J. Song, "A Survey of Research on Mobile Cloud Computing," *Proc. 2011 10th IEEE/ACIS International Conf. on Computer and Information Science (ICIS 2011)*, pp.387-392, Dec. 2011, doi: 10.1109/ICIS.2011.67.

[3] Y. Natchetoi, V. Kaufman, A. Shapiro, "Service-Oriented Architecture for Mobile Applications," *Proc. 1st Int'l Workshop on Software architectures and mobility (SAM 2008)*, pp.27-32, 2008, doi: 10.1145/1370888.1370896.

[4] Wikipedia, Dec., 2012, <http://en.wikipedia.org/wiki/Cluster_(computing)>

[5] M. Gerla and J.T.C. Tsai, "Multicluster, mobile, multimedia radio network, Wireless Networks", *Wireless Networks*, Vol.1, pp.255-265. 1995, doi: 10.1007/BF01200845.

[6] D. Baker, and A. Ephremides, "The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm," *IEEE Transactions on Communications*, Vol.29, pp.1694-1701, 1981, doi: 10.1109/TCOM.1981.1094909.

[7] A. Amis, R. Prakash, T. Vuong, and D. Huynh, "Max-Min D-Cluster Formation in Wireless Ad Hoc Networks," *Proc. 9th Annual Joint Conf. of the IEEE Computer and Communications Societies(INFOCOM 2000)*, Vol.1, pp.32-41, 2000, doi: 10.1109/INFCOM.2000.832171.

[8] P. Harrington, *Machine Learning in Action*, Manning Publication, 2012.

[9] I.H. Witten, E. Frank, and M.A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, Third Edition, Morgan Kaufmann, 2011.

김 대 영

e-mail : dykim723@gmail.com
2011년 성공회대학교 소프트웨어공학과 (공학사)
2012년~현 재 숭실대학교 컴퓨터학과 석사과정
관심분야 : 클라우드 컴퓨팅 (Cloud Computing), 모바일 서비스(Mobile Service)

라 현 정

e-mail : hjla80@gmail.com
2003년 경희대학교 우주과학과(이학사)
2006년 숭실대학교 컴퓨터학과(공학석사)
2011년 숭실대학교 컴퓨터학과(공학박사)
2011년~현 재 숭실대학교 모바일 서비스 소프트웨어공학 센터 연구 교수
관심분야 : 서비스 지향 컴퓨팅 (Service Oriented Computing), 클라우드 컴퓨팅(Cloud Computing), 모바일 서비스 (Mobile Service)

김 수 동

e-mail : sdkim777@gmail.com
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~현 재 숭실대학교 컴퓨터학부 교수
관심분야 : 서비스 지향 아키텍처(SOA), 클라우드 컴퓨팅(Cloud Computing), 모바일 서비스(Mobile Service), 객체지향 S/W공학, 컴포넌트 기반 개발 (CBD), 소프트웨어 아키텍처(Software Architecture)