# 편재형 컴퓨팅 환경에서의 e-비즈니스 응용을 위한 분할 동기화 이동 트랜잭션 처리 모델

최 미 선[†] · 김 영 국[††]


## 요 약

제한된 무선통신 대역폭 및 불완전한 무선통신 인프라, 이동 단말기의 배터리 용량 등과 같은 이동 컴퓨팅 환경의 고유한 특성으로 인해 이동 단말기에서 실행되는 e-비즈니스 응용은 잦은 접속단절에 직면하게 된다. 또한 고가의 무선통신 비용이나 잦은 무선 통신으로 인해 급격하게 소모되는 이동 단말기 전력을 절약하기 위해 자발적인 접속단절 상태에서 동작하기도 한다. 본 연구에서는 데이터비축을 이용하여 대부분 접속단절 상태에서 이동 단말기에서 자치적으로 이동 트랜잭션을 처리하면서도 데이터 중복과 네트워크 분할로 인해 발생가능한 일관성 문제를 효율적으로 해결할 수 있는 분할 동기화 이동 트랜잭션 모델을 제안한다. 분할 동기화 이동 트랜잭션 모델은 이동 트랜잭션을 컴포넌트 단위로 분할한 후, 서버에서의 사용 가능성과 충돌 가능성을 고려하여 컴포넌트 트랜잭션별로 동기화 우선순위를 할당하고 우선순위가 높은 컴포넌트 트랜잭션들부터 동기화를 우선 실시하여 부분 결과를 공개한다. 결과적으로 이동 클라이언트에서 변경한 데이터에 대한 서버에서의 가용성을 높이고, 중요도가 낮은 부분은 이동 단말기의 제한된 자원 및 무선 대역폭과 고가의 통신 요금 등을 고려하여 서버에 늦게 반영함으로써 무선 대역폭 및 컴퓨팅 자원의 활용도를 극대화시키는 효과를 기대할 수 있다.


# A Split Synchronizable Mobile Transaction Processing Model for e-Business Applications in Ubiquitous Computing Environment

Mi-Seon Choi[†] · Young-Kuk Kim[††]


## ABSTRACT

An e-business client application in ubiquitous mobile computing environment may become disconnected from the enterprise server due to broken communication connections caused by the limitation of mobile computing environments(limited battery life of the mobile device, low bandwidth communication, incomplete wireless communication infrastructure, etc). It is even possible that mobile client application intentionally operates in disconnected mode to reduce communication cost and the power consumption of the mobile device. We use "data hoarding" as a means of providing local autonomy to allow transactions to be processed and committed on the mobile host despite of disconnection. The key problem to this approach is the synchronization problem that serialize potentially conflicting updates from disconnected clients on master objects of the server database. In this paper, we present a new transaction synchronizing method that splits a transaction into a set of independent component transactions and give the synchronization priority on each component taking the possibility of use and conflicts in the server into consideration. Synchronization is performed component by component based on synchronization priority. After the preferred component of a mobile transaction succeeds in synchronization with the server, the mobile transaction can pre-commit at server. A pre-committed transaction's updated value is made visible at server before the final commit of the transaction. The synchronization of the component with low synchronization priority can be delayed in adaption to wireless bandwidth and computing resources. As a result, the availability of important data updated by mobile client is increased and it can maximize the utilization of the limited wireless bandwidth and computing resources.


**키워드 : 접속단절(Disconnection), 이동 트랜잭션(Mobile Transaction), 컴포넌트 트랜잭션(Component Transaction), 분할 동기화(Split Synchronization)**

## 1. 서 론

이동 컴퓨팅이란 사용자가 이동 중일 때도 휴대용 컴퓨터를 소지하고 무선으로 네트워크에 접속하여 컴퓨팅을 수행할 수 있도록 해주는 컴퓨팅 패러다임으로 이제는 유선 컴퓨팅 만큼이나 일반적인 컴퓨팅 환경이 되어가고 있다. 노트북이나 팜탑 등과 같은 휴대용 컴퓨터에 무선 LAN 카드를 장착하여 무선 AP(Access Point) 주변에서 유선 LAN과 동일한 품질의 무선 LAN을 이용하는 것이 보편화되어 가고 있으며, 휴대폰이나 PDA, 스마트폰 등과 같은 이동전화 단

※ 본 연구는 충남대학교 학술진흥장학재단과 정보통신부의 대학 IT연구센터(ITRC) 지원을 받아 수행되었음.
† 정 회 원 : 우송대학교 컴퓨터학과 초빙교수
†† 종신회원 : 충남대학교 컴퓨터과학과 교수
논문접수 : 2004년 3월 10일, 심사완료 : 2004년 6월 10일

말기로도 기본적인 통화서비스 이외에 이동 전화 사업자가 제공하는 다양한 종류의 무선 인터넷 서비스를 제공받을 수 있다. 무선 인터넷 컨텐츠를 액세스하거나 전자메일 송수신 등과 같은 서비스 외에도 모바일 뱅킹, 온라인 주식거래, 쇼핑, 티켓팅 등과 같은 다양한 형태의 이동 전자 상거래 서비스가 제공 중이고, 더 나아가서는 이동 중에 회사의 인트라넷에 접속하거나 구매 주문 입력 및 승인, 영업 정보 이용, 서비스 요청 접수 등과 같은 비즈니스 업무를 처리하는 것까지 가능해졌다.

이동 컴퓨팅 환경에서의 데이터 처리에 영향을 미치는 중요한 제한요소로는 휴대용 노트북과 이동 전화, PDA 등을 포함하는 이동 단말기의 배터리 용량과 무선 통신 대역폭 및 불완전한 무선 통신 인프라(infrastructure) 등을 들 수 있다. 유선망에 연결되어 있는 컴퓨터에 비해 이동 단말기는 지속적인 전력 공급이 불가능하므로 전력 사용에 있어 매우 제한적인데다가 무선 통신은 유선 통신에 비해 이동 단말기의 전력을 많이 소모하게 된다. 또한 제한된 무선 통신 대역폭은 송수신되는 데이터의 양에 제약을 가하며, 이동 단말기가 무선 통신 음영 지역에 진입하거나 전송매체에 대한 외부 간섭들로 인해 접속단절(disconnection)이 자주 발생할 수 있고 이로 인해 전송 데이터의 파손 및 분실이 증가될 수 있다[1, 6]. 송수신되는 데이터 패킷 양에 의해 무선 통신 요금이 부과되는 경우에는 유선 통신에 비하여 고가인 무선 통신 요금도 이동 단말기와 네트워크간에 지속적인 연결을 방해하는 주요 요인으로 작용할 수 있다. 이러한 이유들로 인해 이동 단말기는 의도적 또는 비의도적인 접속단절에 자주 직면하게 된다.

이동 컴퓨팅 환경의 한계로 인해 유발되는 접속단절을 극복하기 위해, 이동 트랜잭션 처리 모델은 접속단절로 인한 트랜잭션 철회를 최소화하고, 접속단절 기간동안 공유 데이터에 대해 발생할 수 있는 충돌을 해결할 수 있어야만 한다. 접속단절로 인해 이동 단말기에서 트랜잭션 실행이 블록킹(blocking)되는 것을 방지하기 위해 접속단절에도 불구하고 트랜잭션이 이동 단말기에서 처리될 수 있는 자치성(autonomy)을 제공해야만 한다. 이동 단말기는 "캐싱(caching)"에 의해 전력과 데이터의 고정 공급원으로부터의 접속단절을 극복할 수 있다. 이동 단말기에 내장되어 있는 배터리가 전력에 있어서의 캐시 역할을 하는 것처럼, 데이터도 이동 단말기에서 실행될 응용 프로그램이 사용할 수 있도록 단말기의 로컬 저장소에 캐시될 수 있다. 접속단절에 대비하여 미리 필요한 데이터를 이동 단말기의 지역 저장소에 의도적으로 다운로드하는 것을 "데이터 비축(hoarding)"이라고 하며, 이동 단말기의 제한된 저장 용량 및 통신 비용, 동시성 제어 문제 등을 고려해볼 때, 이동 단말기에 캐시되는 부분은 최소화하는 것이 바람직하다[2]. 필요한 데이터 객체의 크기가 클 경우, 객체 전체를 캐시하는 것보다는 애플리케이션이나 데이터의 시맨틱(semantic)을 활용하여 객체를 분할하여 캐시함으로써 트랜잭션 처리 성능 향상과 일관성(consistency) 문제를 보다 쉽게 다룰 수 있다[12].

본 논문에서는 편재형(ubiquitous) 컴퓨팅 환경에서 다양한 형태의 이동 정보 단말기에서 수행되는 e-비즈니스 응용을 지원하기 위한 효율적인 이동 트랜잭션 처리 모델을 제안하고 성능을 평가하는 것을 목표로 한다. 예를 들어 e-비즈니스 응용에서 이동 비즈니스 사원이 휴대하고 다니면서 고객의 주문을 접수하는 등의 비즈니스 트랜잭션 처리에 사용하는 이동 단말기와 회사 내의 엔터프라이즈 서버 간의 통신은 주로 무선통신을 사용하지만, 무선통신 비용 및 이동 단말기의 배터리 용량 문제와 무선 통신 음영 지역에 진입할 가능성이 존재하기 때문에 서버와 지속적인 연결 상태를 유지한다는 것은 매우 어렵다. 따라서 기존에 유선환경에서 사용되던 온라인 트랜잭션 처리 방법을 그대로 따를 경우, 통신비용이 증가하고 잦은 네트워크 단절로 인해 트랜잭션 철회가 자주 발생하고 트랜잭션 응답시간이 길어지는 등의 문제가 발생할 수 있다. 이러한 문제를 해결하기 위해 본 연구에서는 접속단절을 지원하는 대부분의 이동 트랜잭션 처리 모델[5, 9, 10]들과 같이 접속단절 시 트랜잭션 처리에 필요한 데이터를 이동 단말기에 비축한 후, 서버와의 접속단절 상태에서 이동 단말기에 비축된 데이터를 이용하여 이동 트랜잭션을 수행하고, 서버와의 일시적인 연결을 설정하여 필요한 데이터를 충전하고 일관성 조정 작업을 수행하는 방식을 사용한다. 그러나 서버와의 일관성 조정 작업을 수행하는 동기화 방식에 있어서는 접속단절 동안 이동 클라이언트에서 변경된 데이터에 대한 서버와의 동기화 과정을 기존의 방법들처럼 단순히 트랜잭션 단위로만 실시하는 것이 아니라, 이동 트랜잭션을 트랜잭션을 구성하는 데이터 객체나 연산의 시맨틱에 따라 컴포넌트 단위로 분할한 후, 컴포넌트 단위로 변경된 데이터의 시급성 및 충돌 가능성을 고려해 우선순위를 부여하여 동기화를 실시한다. 제안하는 분할 동기화 이동 트랜잭션(Split Synchronizable Mobile Transaction : SSMT) 모델에서는 하나의 트랜잭션을 구성하는 컴포넌트 별로 동기화 우선순위를 할당하여 다른 클라이언트들 및 서버와의 충돌 가능성과 사용 가능성이 높은 부분부터 먼저 서버 데이터베이스에 반영함으로써 이동 클라이언트에서 변경된 데이터에 대한 서버에서의 가용성을 높이고, 충돌 가능성과 사용 가능성이 낮은 부분은 이동 단말기의 제한된 자원 및 무선 대역폭과 고가의 통신 요금 등을 고려하여 서버에 늦게 반영함으로써 제한된 무선 대역폭 활용도를 극대화시킬 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 기존에 연구되었던 접속단절을 지원하는 이동 트랜잭션 처리 모델에 대하여 기술하며, 3장에서는 본 연구에서 제안하는 e-비즈니스 트랜잭션 처리 시스템 및 분할 동기화 이동

트랜잭션 모델에 대하여 기술하고 4장에서는 적용 사례 및 적용사례를 이용한 성능평가 결과를 보인다. 마지막으로 5장에서는 결론 및 향후 연구 방향에 대해 기술한다.

## 2. 관련 연구

이동 컴퓨팅 환경에서 이동 컴퓨터가 접속단절됨으로써 발생할 수 있는 네트워크 분할 상황에서 이동 컴퓨터에서의 지속적인 이동 트랜잭션 처리를 보장하기 위한 방법으로서 데이터 비축을 이용하는 경우가 일반적이다[5, 9, 10]. 이 방법들에서는 접속단절에 대비해서 필요한 데이터를 미리 이동 컴퓨터에 비축한 다음, 접속단절 상태에서 비축된 데이터를 사용하여 이동 트랜잭션을 수행한다. 접속단절 상태에서 독자적으로 수행된 이동 트랜잭션은 데이터 중복으로 인한 데이터 일관성 문제를 야기하게 되고, 이 문제를 해결하기 위해 네트워크에 재 연결 시, 이동 컴퓨터에서 수행되었던 트랜잭션의 변경사항을 데이터의 원본(primary copy)을 저장하고 있는 마스터 데이터베이스에 반영하는 재통합(reintegration) 과정을 실행한 후, 그 결과를 이동 컴퓨터에 전송해주는 메커니즘을 사용한다.

Gray[5]에 의해 제안된 2단계 중복 모델은 이동 컴퓨팅 환경에서의 잦은 접속단절 문제를 해결하기 위한 모델로 유선에 의해 연결되어 항상 연결 상태를 유지하고 있는 베이스 노드를 하나의 단계로, 무선에 의해 연결되어 대부분의 시간을 접속단절 상태로 남아있는 이동 노드를 다른 하나의 단계로 분리하여 다루고 있다. 베이스 노드는 데이터베이스의 중복 객체들을 저장하고 있으며 대부분 객체의 원본을 저장하고 있는 마스터 노드의 역할을 수행한다. 이동 노드는 특정 데이터베이스 객체들의 복사본을 저장하며 일부 데이터 객체의 마스터 노드일 수도 있다. 이동 노드에 저장되는 각 데이터의 복사본마다 마스터(master) 버전과 임시(tentative) 버전의 2가지 버전이 유지된다. 트랜잭션 타입으로는 임시 트랜잭션과 베이스 트랜잭션의 2가지가 존재한다. 마스터 버전은 이동 노드가 객체의 마스터로부터 받은 가장 최근의 값을 유지하며 임시 버전은 접속단절 동안 이동 노드에서 변경된 내용을 유지한다. 임시 트랜잭션이란 임시 버전 데이터에 대해 실행되는 트랜잭션으로 새로운 임시 버전 데이터를 생성하는 반면, 베이스 트랜잭션은 마스터 데이터 버전만을 사용하여 실행되고 역시 마스터 데이터를 생성한다. 이동 노드가 기지국과 다시 연결되면 접속단절 동안 실행되었던 임시 트랜잭션들은 베이스 노드로 전송되어 베이스 트랜잭션으로서 다시 처리되고 애플리케이션에서 지정한 허용기준을 만족하지 못한다면 그 트랜잭션은 철회되고 메시지가 이동 노드로 반환된다.

Liu[8]는 Gray가 제안한 2단계 중복 모델에서 임시 트랜잭션이 베이스 트랜잭션으로 다시 수행됨으로써 발생하는 오버헤드를 감소시키기 위한 방안으로 트랜잭션의 시맨틱을 활용하여 마스터 데이터에 대해 수행된 베이스 트랜잭션들의 실행 히스토리인 베이스 히스토리와 이동 노드에서 수행된 임시 트랜잭션들의 실행 히스토리인 임시 히스토리를 병합하고, 병합된 히스토리 결과를 이동 노드로 보냄으로써 베이스 노드에 전송되어 재처리되어야할 임시 트랜잭션들의 양을 감소시킬 수 있는 방법을 제안하였다. 이 방법은 베이스 히스토리와 임시 히스토리 간에 병합되어질 수 있는 부분이 많다면, 이동 노드에서 베이스 노드로 전송되는 임시 트랜잭션 정보량 및 베이스 노드에서의 트랜잭션 재처리 오버헤드를 상당히 줄일 수 있는 장점이 있는 반면, 제한된 컴퓨팅 자원을 가지고 있는 이동 노드에서 베이스 노드에 전송할 임시 트랜잭션들의 임시 히스토리 정보를 생성하기 위한 별도의 컴퓨팅 오버헤드를 유발하는 문제점을 가지고 있다.

Chrysanthis[4]는 이동 컴퓨터의 메모리 및 컴퓨팅 능력, 배터리 용량의 한계를 극복하고 이동 분산 환경에서 데이터 일관성을 유지하기 위한 방안으로 기존의 개방 중첩형(open-nested) 트랜잭션 모델에 이동 호스트와 이동 지원 고정 호스트(MSS : Mobile Support Station)가 트랜잭션의 부분 결과를 공유해가면서 트랜잭션을 공동 수행할 수 있도록 지원하는 보고트랜잭션(Reporting-transaction)과 공동트랜잭션(Co-Transaction)이라는 새로운 유형의 트랜잭션이 추가된 확장된 개방 중첩형 트랜잭션 모델을 제안하였다. 개방 중첩형 트랜잭션 모델은 이동 트랜잭션을 부모 트랜잭션이나 다른 트랜잭션의 실행 상태와 무관하게 독자적으로 커미트하거나 철회할 수 있으며, 부모 트랜잭션이 철회될 경우, 연관된 보상(compensation) 트랜잭션이 수행됨으로써 독자적으로 커미트했던 결과에 대한 보상이 가능한 컴포넌트 트랜잭션의 집합으로 정의한다.

Walborn[12]은 무선 통신 비용 및 전송 지연을 줄일 수 있을 뿐만 아니라 접속단절 시에도 이동 호스트가 자치적으로 고정 호스트에 저장되어 있는 공유 데이터에 대한 연산을 계속 수행할 수 있도록 하는 시맨틱 기반 트랜잭션 처리 모델로서 "단편화 가능 객체"의 개념을 제안하였다. 이 모델에서는 데이터베이스에 저장되어 있는 객체의 구조를 이용하여 크고 복잡한 객체를 관련이 있는 데이터끼리 분할하여 작고 다루기 쉬운 단편(fragment)으로 나눈다. 공유 데이터의 원본을 가지고 있는 고정 호스트의 데이터베이스 서버는 이동 호스트가 "분할(split)" 연산을 사용하여 공유 데이터의 단편을 요청할 경우, 객체의 단편을 이동 호스트에 분배해준다. 일단 이동 호스트에 분배된 데이터베이스 내의 단편은 다른 호스트들에게는 보이지 않게 된다. 이동 호스트는 분배받은 객체의 단편을 지역 저장소에 캐시해 놓고, 다른 호스트들과 통신할 필요 없이 캐시 내의 단편 내에서만 일관성을 유지하면서 독립적으로 연산을 수행한다. 필요한 연산이

끝나고 네트워크가 연결된 상태이면, 이동 호스트는 "합병(merge)" 연산을 통해 캐시되었던 단편을 데이터베이스 서버에 반환하게 된다. 이 모델은 객체 구조의 시맨틱을 이용하여, 객체를 단편으로 나누고 이 단위로 일관성을 유지함으로써 호스트들에서 발생할 수 있는 충돌 가능성을 사전에 제거하고, 이동 컴퓨터에서 캐시한 단편에 대해서는 네트워크의 상태와 관계없이 독자적으로 트랜잭션을 수행할 수 있다는 장점이 있는 반면, 이 모델이 적용될 수 있는 데이터의 종류는 서로 연관성이 없는 단편으로 분할할 수 있는 특징을 가지는 경우로 제한된다는 단점을 가진다.

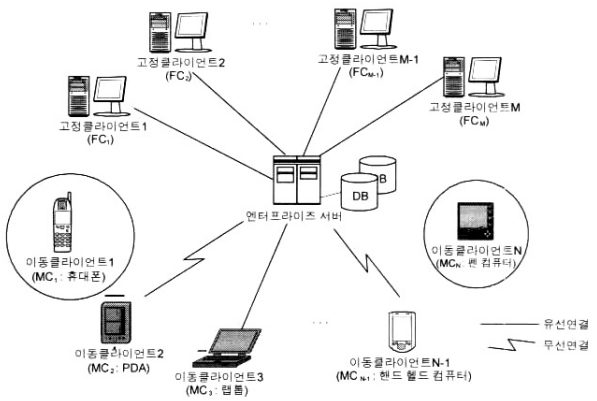
Madria[7]는 이동 호스트 및 고정 호스트에서의 데이터 가용성을 개선하기 위하여 이동 트랜잭션에 prewrite 연산과 pre-commit 메커니즘을 도입한 이동 트랜잭션 모델을 제안하였다. prewrite는 실제 데이터 객체의 값을 변경시키지는 않고, 트랜잭션이 최종 커미트된 이후에 데이터 객체가 가지게 될 미래의 값만을 공개하여 다른 트랜잭션들이 사용할 수 있도록 해주는 연산이다. 이동 트랜잭션은 데이터 객체를 직접 변경하는 대신에, prewrite 값만을 공개함으로써 이동 호스트에서 빠른 시간 내에 pre-commit 되어질 수 있다. 실제 데이터 객체에 대한 변경 작업은 시간과 자원이 많이 소모되는 작업이므로 나중에 MSS에서 실행된다. 결과적으로 트랜잭션 실행 중 비용이 많이 드는 부분에 대한 책임을 MSS에 전가하기 때문에 이동 호스트에서의 계산 비용을 줄여주는 효과를 기대할 수 있다. 또한 pre-commit된 트랜잭션의 prewrite 값은 트랜잭션의 최종 커미트 이전에 이동 호스트와 고정 호스트에서 사용할 수 있으므로 이동 컴퓨팅에서 일반적인 잦은 접속단절 동안에도 데이터의 가용성을 높일 수 있다.

본 연구에서는 이동 단말기에서의 트랜잭션 처리와 서버와의 기본적인 동기화 방법에 있어서는 Gray의 2단계 중복 모델에 기반을 두고 있다. 2단계 중복 모델은 트랜잭션 단위의 동기화만을 실시하므로 트랜잭션 내부에 존재하는 연산들이나 연산에서 사용하는 객체의 시맨틱은 전혀 고려하지 않는다. 그러나 본 연구에서는 Chrysanthis와 Walborn등의 연구를 2단계 중복 모델에 접목시켜 서버와의 동기화 과정에서 트랜잭션이 액세스하는 객체와 연산의 시맨틱을 이용할 수 있도록 이동 트랜잭션을 동일한 객체 및 연산 시맨틱을 가지는 컴포넌트 트랜잭션으로 분할하고 컴포넌트 트랜잭션 단위로 동기화 우선순위를 부여한다. 제한된 무선 통신 대역폭 및 무선 통신 비용 등을 고려하여 우선 순위에 따라 컴포넌트 트랜잭션 별로 서버와의 동기화를 실시한다. 또한 이동 트랜잭션이 변경한 데이터에 대한 서버에서의 가용성을 개선하기 위해 Madria의 pre-commit 메커니즘을 사용하여, 전체 트랜잭션에 대한 동기화가 안된 상태이더라도 동기화에 있어서 중요한 컴포넌트 트랜잭션이 동기화에 성공했다면 트랜잭션을 pre-commit하고, pre-commit된 트랜잭션에 속하는 컴포넌트 트랜잭션들의 부분결과를 공개한다.

## 3. 편재형 e-비즈니스 환경을 위한 분할 동기화 이동 트랜잭션 처리 모델

### 3.1 이동 비즈니스 트랜잭션 처리 환경

편재형 e-비즈니스 환경을 위한 이동 트랜잭션 처리 시스템은 (그림 1)과 같이 이동 비즈니스 사원들의 이동 단말기들로 이루어지는 이동 클라이언트들과, 유선 망에 의해 연결되어 있는 고정 클라이언트들, 회사의 엔터프라이즈 서버로 구성되는 클라이언트/서버 환경으로 이루어진다. 이동 클라이언트 단말기로는 랩탑이나 노트북, 펜 컴퓨터, 핸드 헬드 컴퓨터, 휴대폰, PDA, 스마트폰 등과 같이 무선 통신 기능과 컴퓨팅 능력을 가진 다양한 종류의 단말기들이 사용될 수 있다. 이동 단말기에는 간단한 내장형 이동 데이터베이스가 탑재되어 있고 자치적인 처리 능력 및 서버와의 무선 통신 기능을 가지고 있다고 가정한다.




(그림 1) 이동 비즈니스 트랜잭션 처리 환경

이동 클라이언트들은 이동 단말기가 서버와 무선 통신이 불가능한 음영지역에 진입했거나 이동 단말기의 배터리 절약과 무선 통신 비용 절감을 위해 의도적으로 대부분의 시간 동안 서버와 접속단절 상태에서 트랜잭션을 수행한다. 이동 클라이언트는 접속단절 이전에 필요한 데이터를 미리 자신의 지역 데이터베이스에 비축하여 접속단절 동안 비축된 데이터를 이용하여 트랜잭션을 처리한다. 이와 같은 이동 클라이언트의 연산 모드를 **접속단절연산모드**라고 부르기로 한다. (그림 1)에서 $MC_1$, $MC_N$이 서버와 접속단절연산모드에서 비즈니스 트랜잭션을 수행하고 있는 이동 클라이언트들이다.

이동 클라이언트는 본격적인 접속단절연산모드로 진입하기위해 서버로부터 접속단절연산모드에서 필요한 데이터베이스 객체들을 다운로드하여 자신의 지역 데이터베이스에 캐시하는 **"데이터비축"**을 실행한다. 데이터비축 및 반환 과정을 처리하기 위해서 **체크아웃(checkout)과 체크인(chec-**

**kin)**의 2가지 연산이 필요하다. 이동 클라이언트는 체크아웃에 의하여 비축하고자 하는 데이터베이스 객체를 서버로부터 캐시하고 접속단절연산모드에 진입하여 비축된 데이터를 이용하여 이동 비즈니스 트랜잭션을 처리한다. 이동 클라이언트는 모든 비즈니스 트랜잭션들의 처리가 끝나고 비즈니스를 마감하고자 할 때, 접속단절 연산 모드를 종료하고 체크인을 사용하여 자신의 지역 데이터베이스에 비축되었던 데이터베이스 객체를 서버에 반환한다. 체크아웃 및 체크인은 전송해야할 데이터양이 비교적 많으므로 고대역 통신이 가능하면서도 안정적이고 비용이 저렴한 통신매체를 사용하는 것이 바람직하다. (그림 1)의 $MC_3$이 체크아웃 또는 체크인을 위하여 서버와 유선 연결 상태에서 작업을 수행하고 있는 이동 단말기에 해당한다. 이동 클라이언트는 접속단절 상태에 있다가 정해진 스케쥴이나 통신망의 사용가능 여부에 따라 일시적으로 서버와 연결을 설정하고 필요한 데이터를 다운로드하거나 동기화 과정 등을 수행할 수 있다. (그림 1)의 $MC_2$와 $MC_{N-1}$은 서버와 일시적인 무선 연결이 설정되어 부분적인 동기화 작업을 진행하고 있는 상태에 있음을 보여준다.

### 3.2 데이터베이스 객체

서버 데이터베이스 객체는 이동 비즈니스 업무를 시작하기 전에 사무실에서 고대역폭의 유무선 네트워크를 이용하여 이동 단말기의 데이터베이스에 비축된다. 이동 단말기의 제한된 저장용량 및 처리 능력 등을 고려할 때, 데이터베이스 객체의 크기가 클 경우 객체 전체를 이동 클라이언트에 캐시하는 것은 비효율적이며, 동일한 객체를 캐시한 이동 클라이언트들간의 일관성 유지 작업도 복잡해질 수 있다. 이동 클라이언트가 서버 데이터베이스 객체를 서로 겹치지 않는 단편으로 분할하여 캐시할 수 있다면, 공유 객체에서 발생할 수 있는 충돌의 가능성을 사전에 제거하고, 이동 단말기 내의 저장장소 및 통신 비용을 절약하는 것이 가능하다. 이러한 특징을 갖는 객체를 **단편화가능객체(Fragmentable Object)**로 정의하기로 한다.

단편화가능객체란 클라이언트 애플리케이션에서 서버에 적절한 객체 분할조건을 제출함으로써 객체의 일부분만을 자신의 지역 데이터베이스에 캐시한 후, 접속단절에도 불구하고 다른 클라이언트와의 충돌 가능성 없이 독자적으로 연산을 수행할 수 있는 객체로 2장의 관련연구에서 언급했던 Walborn의 단편화가능객체와 같은 개념이다. 단편화가능객체의 예로 특정 회사의 고객 테이블을 들 수 있는데 세일즈 사원마다 담당 고객이 정해져 있으므로, 각 세일즈 사원은 고객 테이블로부터 자신이 담당한 고객 관련 레코드들만을 분할하여 이동 컴퓨터의 지역 데이터베이스에 비축함으로써 접속단절 시에도 다른 클라이언트와 충돌 가능성 없이 고객 레코드를 독자적으로 변경할 수 있다. 서버에 저장되어 있는 객체들 중 일부는 객체의 구조나 응용의 성격에 따라 단편화가능객체로 분류될 수 있다. 서버 데이터베이스에는 단편화가능객체 이외에도 단편화가 불가능하여 이동 클라이언트들이 객체 전체를 공유해야만 하는 객체인 **단편화불가능객체**들도 존재할 수 있다. 단편화불가능객체라 하더라도 이동 클라이언트에서 객체를 읽기 전용으로만 사용한다면 일관성유지에 있어서 별 문제가 되지 않으나 캐시된 단편화불가능객체에 대해 여러 클라이언트들에서 접속단절 동안에 독자적으로 변경 연산들을 수행한다면 일관성 유지 작업이 매우 복잡해질 것이다.

이동 단말기에서 수행되는 e-비즈니스 응용은 단편화가능객체 및 단편화불가능객체에 대한 액세스를 포함할 수 있다. 이동 비즈니스 사원이 주문 접수 시, 제품 가격을 조회할 때 사용하는 가격 테이블이 단편화가 불가능하여 테이블 전체를 이동 클라이언트에 캐시하여 사용하는 경우를 생각해보자. 고객을 상대로 비즈니스를 수행하는 이동 클라이언트는 가격 테이블을 조회용으로만 사용하고 가격을 변경시키는 작업은 서버 또는 다른 전담 클라이언트에서 이루어질 것이다. 만일 여러 이동 클라이언트가 접속단절 상태에서 단편화불가능객체의 내용을 독자적으로 변경시키는 것을 허용한다면 일관성 조정 작업이 매우 복잡해지고, 단편화가능객체에 대한 연산도 이 객체와 종속성을 가짐으로써 충돌로 인한 트랜잭션 철회가 빈번해질 것이다. e-비즈니스 응용의 시맨틱 상 이동 클라이언트에서 단편화불가능 객체를 사용하는 용도는 상품의 가격이나 관련 정보를 얻기 위한 경우가 대부분이므로 본 연구에서는 단편화불가능 객체에 대한 액세스를 읽기 전용으로 제한하고 변경 연산이 필요한 경우에는 서버와의 접속을 설정하여 서버에게 변경을 요청하는 방식을 사용한다. 반면에 이동 클라이언트에 단편화불가능 객체가 읽기 전용으로 캐시되었다 하더라도 서버에서 단편화불가능 객체의 내용을 변경시키는 것은 허용하도록 한다.

### 3.3 분할 동기화 이동 트랜잭션 모델

이동 클라이언트가 접속단절상태에서 로컬 데이터베이스에 비축되어있는 데이터를 이용하여 e-비즈니스 이동 트랜잭션을 수행했을 때, 데이터 중복과 네트워크 분할로 인해 중복 데이터의 일관성 유지문제가 발생하게 된다. 본 연구에서는 특정 e-비즈니스 응용의 시맨틱에 의해 이동 클라이언트가 액세스하는 데이터를 단편화가능객체와 단편화불가능객체로 구분하고 단편화불가능객체에 대한 액세스는 읽기 전용으로 제한한다. 이러한 가정 하에 단편화가능객체 및 단편화불가능객체에 대한 액세스를 포함하고 있는 e-비즈니스 이동 트랜잭션이 서버와의 접속단절 상태에서 독자적으로 수행됨으로써 발생할 수 있는 서버와의 동기화 문제를 효율적으로 처리하기 위한 분할 동기화 이동 트랜잭션 (Split Synchronizable Mobile Transaction : SSMT) 모델을

제안한다.

분할 동기화 이동 트랜잭션 모델은 접속단절 동안 이동 클라이언트에서 변경된 데이터에 대한 서버와의 동기화 과정을 기존의 방법들처럼 트랜잭션 단위로 실시하지 않고, 이동 트랜잭션을 컴포넌트 단위로 분할한 다음, 우선순위를 부여하여 컴포넌트 단위로 우선순위에 따라 동기화를 실시하는 이동 트랜잭션 모델이다. 동일한 트랜잭션 내에 포함되어 있는 내용 중에서 다른 클라이언트들이나 서버와의 충돌 가능성이 높거나 사용 가능성이 높은 부분부터 먼저 서버에 반영함으로써 이동 클라이언트에서 변경된 데이터에 대한 서버에서의 가용성을 높이고, 그렇지 않은 부분은 이동 단말기의 제한된 배터리 용량 및 무선 대역폭과 고가의 통신 요금 등을 고려하여 서버에 나중에 반영함으로써 제한된 무선 대역폭 활용도를 극대화시키고자 하는 것이다.

SSMT를 적용할 수 있는 대표적인 예로서 이동 세일즈와 같은 e-비즈니스 응용을 들 수 있다. 이동 세일즈 사원이 고객의 주문을 처리하기 위해 이동 클라이언트에 NewOrder 트랜잭션을 제출한다고 가정하자. NewOrder 트랜잭션은 고객이 주문한 물품의 가격이나 재고 등의 물품 정보를 읽어서 주문 레코드를 생성하여 주문 테이블에 저장하는 부분과 고객의 주문 내역을 고객 관련 테이블에 저장하는 2개의 부분으로 구성된다. 주문 레코드와 고객 레코드의 변경은 동일한 주문에 관련된 것이므로 하나의 트랜잭션에서 함께 처리되어야만 하지만 서로 다른 성격을 가진다. 주문 레코드는 고객 레코드에 비해 상대적으로 빠른 시간 내에 서버에 전달되어 처리되어야 하며, 접속단절 동안 서버에서의 물품정보(가격이나 재고여부 등) 변경 여부에 영향을 받는 레코드들이다. 반면에 고객 이름, 주소 등과 같은 고객 기본 정보나 주문 히스토리 등을 저장하고 있는 고객 관련 레코드는 서버에서 사용될 가능성도 높지 않고 세일즈사원마다 담당 고객이 정해져 있다면, 변경된 고객 레코드에 대해서 서버와의 충돌 가능성도 존재하지 않는다. 이와 같이 서로 상이한 주문 레코드와 고객 레코드의 시맨틱을 이용한다면, 시맨틱에 대한 고려가 전혀 없이 단순히 NewOrder 트랜잭션의 내용을 한꺼번에 서버에 전달하여 동기화를 시키는 것보다는 트랜잭션을 성격에 따라 적절히 분할하여 선별 동기화를 시킴으로써 제한된 이동 컴퓨팅 환경의 자원을 효율적으로 사용하는 것이 가능하다. 즉, 서버에서의 사용 가능성이 높고 충돌 가능성도 높은 주문 레코드 관련 부분을 먼저 전송하여 동기화를 시키고, 그렇지 않은 고객 레코드 부분에 대한 동기화는 이동 단말기가 유선 네트워크와 전력의 공급원에 안정적으로 연결된 상태로 이동 클라이언트에 비축되었던 모든 데이터를 반환하는 체크인과정에서 실행함으로써 이동 단말기의 배터리 소모와 무선 통신 요금을 절약하는 효과를 기대할 수 있을 것이다. 더불어서, 먼저 서버로 전송된 주문 레코드 변경 부분이 동기화에 성공한다면, 주문 레코드를 서버에서 다른 트랜잭션들이 사용할 수 있도록 공개하여 서버에서 이동 클라이언트가 변경한 데이터에 대한 가용성이 높아지는 효과도 함께 기대할 수 있다.

이동 세일즈 외에도 고객의 주문을 전화나 인터넷 등으로 접수받아 빠른 시간 내에 비디오를 가정에 배달해주는 이동 비디오 버스 응용이나, 택배회사에서 사용되는 화물 배달 추적 시스템 등과 같이 하나의 트랜잭션에 실시간 처리를 요하는 부분(각 비디오의 대여 및 반환 상태를 변경시키는 부분)과 시간적으로 여유가 있는 부분(고객의 히스토리 등을 변경시키는 부분)이 혼재되어 있는 경우에도 효과적으로 적용할 수 있다.

#### 3.3.1 분할 동기화 이동 트랜잭션의 구성

SSMT는 분할 동기화가 가능한 독립적인 동기화 컴포넌트 트랜잭션 $t_1, t_2, \cdots, t_n$들의 집합으로 구성된 이동 트랜잭션으로 정의할 수 있다. SSMT의 각 동기화 컴포넌트 트랜잭션 $t_1, t_2, \cdots, t_n$은 독립적인 실행 및 동기화가 가능하며 실행 순서가 바뀌어도 동일한 최종 결과를 생성하는 특징을 가진다.

**[정의 1]** 다음 조건을 만족하는 어떤 이동 트랜잭션 MT를 분할 동기화가 가능한 독립적인 동기화 컴포넌트 트랜잭션들로 구성된 분할 동기화 이동 트랜잭션으로 정의한다.

1. $MT = \{ t_1, t_2, \cdots, t_n \}$
2. $t_i \subseteq \{ r_i(x), w_i(x) \mid x$는 서버로부터 이동 클라이언트에 비축된 데이터 항목$\}$
3. $1 \le i, j \le n$인 모든 $i \ne j$에 대하여, $WriteSet(t_i) \cap WriteSet(t_j) = \varnothing$
4. $1 \le i, j \le n$인 모든 $i \ne j$에 대하여, $ReadSet(t_i) \cap WriteSet(t_j) = \varnothing$

위의 조건 2는 SSMT가 액세스하는 데이터는 서버에서 이동 클라이언트로 다운로드되어 비축된 데이터임을 의미한다. 조건 3, 조건 4는 SSMT의 각 컴포넌트들이 독립적인 실행 및 동기화가 가능하기 위해 만족되어야만 하는 조건들로 각 컴포넌트 트랜잭션 간에는 실행순서에 관계없이 충돌이 발생하지 않음을 의미한다. 만약 컴포넌트 트랜잭션들 간에 충돌 가능성이 존재한다면, 이들은 하나의 컴포넌트 트랜잭션으로 병합되어야만 한다. 최악의 경우에 이동 트랜잭션이 하나의 컴포넌트 트랜잭션으로만 구성되는 경우도 있을 수 있다. 이런 경우에는 별도의 분할 동기화 과정 없이 일반적인 이동 트랜잭션과 같은 방법으로 처리될 것이다. 앞에서 예로 제시했던 이동 세일즈 사례에서의 NewOrder 트랜잭션을 (그림 2)와 같이 SSMT로 구성할 수 있다.

SSMT로 구성한 NewOrder 트랜잭션은 2개의 동기화 컴포넌트를 가진다. 첫 번째 동기화 컴포넌트는 서버와의 충돌 가능성이 존재하여 전체 트랜잭션의 동기화 성공 여부를 결정짓는 중요한 동기화 컴포넌트로 PCOMPONENT는 3.3.2절

에서 설명되어질 우선(preferred) 동기화 컴포넌트를 의미한다. 우선 동기화 컴포넌트는 하나의 SSMT를 구성하는 컴포넌트 트랜잭션들 중에서 가장 높은 우선순위인 PRIORITY 0값을 할당받는다. 두 번째 컴포넌트는 서버와의 충돌 가능성이 존재하지 않는 컴포넌트로 동기화 성공 여부가 첫 번째 컴포넌트의 동기화 성공 여부에 의해 결정된다. 두 번째 컴포넌트는 2개의 단편화가능객체에 대한 변경 내용을 포함하고 있는데, 필요에 따라 이를 2개의 컴포넌트로 다시 분할하고 서로 다른 우선순위를 부여하는 것이 가능하다.

```
BEGIN TR
        BEGIN PCOMPONENT
            read item record ;
            insert order record ;
        END PCOMPONENT
        BEGIN COMPONENT WITH PRIORITY 1
            update customer record ;
            update history record ;
        END COMPONENT
END TR
```

(그림 2) SSMT로 표현한 NewOrder 트랜잭션

이와 같이 이동 트랜잭션을 컴포넌트 트랜잭션들로 분해하여 각 컴포넌트 단위로 트랜잭션을 분할 처리하는 형태는 기존의 개방 중첩형 이동 트랜잭션 모델들[4]에서도 찾아볼 수 있다. Chrysanthis는 이동 지원 호스트와 이동 클라이언트 간에 이동 트랜잭션을 분할 처리하고 이동 트랜잭션의 부분 결과를 공유하거나 이동 트랜잭션의 일부를 공동 수행하기 위해 이동 트랜잭션을 컴포넌트 트랜잭션으로 분할하였다. 하지만, SSMT에서는 서버와의 동기화 과정을 분할 처리하기 위하여 이동 트랜잭션을 컴포넌트 트랜잭션들로 분할한다. e-비즈니스용 이동 클라이언트는 제한된 컴퓨팅 자원을 가지며, 대부분 단일 사용자 모드로 사용된다. 이와 같은 특성 및 사용 패턴을 반영하여 SSMT는 이동 클라이언트에서의 연산부분을 최소화시키고자 이동 클라이언트에서 컴포넌트 트랜잭션들 간의 병렬 처리나 컴포넌트 트랜잭션 단위의 일방적인 커미트나 철회 등을 허용하지 않는다. SSMT는 이동 클라이언트에서 트랜잭션 단위로 처리되어 일반적인 이동 트랜잭션들과 마찬가지로 순차적으로 실행되고, 트랜잭션이 실패했을 때 철회되며, 성공적으로 수행되었을 때 커미트되는 원자성(atomicity)을 가진다. 이때의 커미트는 [정의 1]의 조건 2에 의하여 접속단절된 동안 서버에서 이동 트랜잭션에서 읽은 데이터를 변경했을 가능성이 존재하므로 이동 클라이언트에서만 지역적으로 커미트되는 로컬 커미트(local-commit) 상태로서 서버와 연결된 후, 동기화 과정을 거쳐야만 최종적으로 트랜잭션이 완료되는 전역 커미트(global-commit) 상태로 전이된다. 로컬 커미트된 이동 트랜잭션은 동기화 컴포넌트 트랜잭션 단위로 분리되어 서버와의 동기화 과정을 분할 수행한다.

3.3.2 동기화 우선순위 및 우선 동기화 컴포넌트

동기화 컴포넌트 트랜잭션마다 서버와의 동기화 우선순위를 부여할 수 있다. 우선순위는 0, 1, 2, …와 같은 방법으로 부여하며 숫자가 작을수록 우선순위가 높음을 의미한다. 특수한 의미의 우선순위를 부여하기 위한 방법으로서 우선순위 값을 가지지 않는 동기화 컴포넌트가 있을 수 있다. 우선순위가 부여되지 않는 동기화 컴포넌트는 서버와의 일시적인 연결 상태에서는 동기화를 실시하지 않고, 이동 클라이언트의 체크인과정에서 동기화를 실시함을 의미한다. 이동 클라이언트에서 이동 트랜잭션이 로컬 커미트된 후, 이동 클라이언트가 서버와 일시적으로 연결되었을 때, 동기화 우선순위에 따라 동기화 컴포넌트 트랜잭션 단위로 동기화가 수행된다.

SSMT의 각 동기화 컴포넌트 트랜잭션 $\{t_1, t_2, \cdots, t_n\}$들은 컴포넌트 트랜잭션 별로 서버와 동기화가 완료되는 순서에 따라 동기화 히스토리(SH : Synchronization History)를 생성한다.

**[정의 2]** SSMT 트랜잭션 $T_k = \{t_1, t_2, \cdots, t_n\}$에 대한 동기화 히스토리 $SH_{Tk}$ 및 이동 클라이언트 $MC_i$에서 수행되는 SSMT 트랜잭션들의 집합 $T = \{T_1, T_2, \cdots, T_m\}$에 대한 전체 동기화 히스토리 $SH_i$는 다음과 같이 정의될 수 있다.

1. $SH_{Tk} = \cup_{i=1,n}\ t_i$ ; 이고
2. 동기화우선순위($t_i$) > 동기화우선순위($t_j$)인 모든 i, j에 대하여 $t_i < SH_{Tk}\ t_j$
3. $SH_i = \cup_{k=1,m}\ SH_{Tk}$ ; 이고
4. $< SH_i \supseteq \cup_{k=1,m} < SH_{Tk}$

[정의 2]의 조건 1과 조건 2는 동일한 SSMT 트랜잭션을 구성하는 컴포넌트들의 서버와의 동기화 순서는 동기화 우선순위에 의해 결정됨을 의미한다. 따라서 첫번째로 동기화가 이루어지는 동기화 컴포넌트의 우선순위가 가장 높다. SSMT 트랜잭션을 구성하는 컴포넌트 트랜잭션들의 동기화가 모두 성공적으로 이루어지면 이동 트랜잭션은 전역 커미트 상태가 된다. [정의 2]의 조건 3과 조건 4는 임의의 이동 클라이언트에서 수행되는 SSMT 트랜잭션들은 서버와의 동기화 순서가 컴포넌트별로 서로 혼합되어 수행될 수 있지만, 동일한 SSMT 트랜잭션에서 우선순위에 따라 결정된 동기화 순서는 변경되지 않음을 의미한다.

하나의 이동 트랜잭션에 대한 동기화를 분할 실시하고, 서버에서 SSMT의 모든 컴포넌트 트랜잭션의 동기화가 완료된 후에야 이동 트랜잭션에서 변경한 데이터를 서버에서 수행되는 다른 트랜잭션에게 공개한다면, 변경된 데이터를 액세스하고자 하는 트랜잭션은 오랜 기다림에 직면해야만 한다. SSMT에서는 이러한 문제점을 해결하기 위해 부분 동기화된 트랜잭션의 중간 결과를 서버에서 실행되는 다른 트랜잭션에게 공개할 수 있는 pre-commit 매커니즘[7]을 제공한다.

SSMT의 동기화 컴포넌트 트랜잭션을 액세스하는 데이터 객체의 성격에 따라 우선(preferred) 동기화 컴포넌트와 지연(deferred) 동기화 컴포넌트 트랜잭션으로 구분한다. 우선 동기화 컴포넌트는 이동 트랜잭션의 동기화 성공 여부를 결정짓는 중요한 컴포넌트로서 우선 동기화 컴포넌트가 서버와의 동기화 작업에서 성공하면, 이동 트랜잭션의 전역 커미트가 보장된다. 따라서 하나의 이동 트랜잭션을 구성하는 동기화 컴포넌트 트랜잭션들 중에서 가장 높은 우선순위를 부여하여 가장 먼저 서버와의 동기화 작업을 실행하게 함으로써 이동 트랜잭션이 생성한 부분 결과에 대한 가용성을 높일 수 있다. 반면에 지연 동기화 컴포넌트 트랜잭션은 동기화 작업의 성공 여부가 우선 동기화 컴포넌트에 의해서 결정되는 컴포넌트이므로 동기화 우선순위가 우선 컴포넌트보다 낮고, 우선 동기화 컴포넌트 트랜잭션이 동기화에 실패한다면, 동기화과정을 실행할 필요가 없다. 결과적으로 동기화 과정에 필요한 나머지 정보를 서버로 전송할 필요가 없으므로 전송 패킷양이 감소하여 통신 비용을 줄이는 효과를 기대할 수 있다.

**[정의 3]** SSMT의 우선 동기화 컴포넌트와 지연 동기화 컴포넌트 트랜잭션을 다음과 같이 정의한다.

1. x가 이동 클라이언트에 비축된 서버가 마스터 권한을 가지고 있는 데이터 객체이고, $r_i(x) \in t_i$ 라면, $t_i$는 우선 동기화 컴포넌트이다.
2. x가 이동 클라이언트에 비축된 서버가 마스터 권한을 가지고 있는 데이터 객체이고, $r_i(x) \notin t_i \wedge w_i(x) \notin t_i$ 라면, $t_i$ 는 지연 동기화 컴포넌트이다.
3. 만일 $t_i$가 우선 동기화 컴포넌트라면, 동일한 트랜잭션 내에 존재하는 모든 지연 동기화 컴포넌트 $t_j$에 대하여, 동기화우선순위($t_i$) > 동기화우선순위($t_j$)이다.
4. 한개의 SSMT 트랜잭션에는 오직 한개 이하의 우선 동기화 컴포넌트 트랜잭션만이 존재할 수 있다.

3.2절에서 언급했던 것처럼 e-비즈니스 응용의 시맨틱에 따라 본 연구에서는 이동 클라이언트에서 서버가 마스터 권한을 가지고 있는 단편화불가능 객체에 대한 액세스는 읽기 전용으로 제한하고 있다. [정의 3]의 조건 1은 우선 동기화 컴포넌트가 단편화불가능객체에 대한 읽기 연산을 포함하고 있음을 의미한다. 접속단절동안 서버에서 단편화불가능객체에 대한 변경이 발생할 수 있고, 단편화불가능객체의 마스터는 서버이므로 이동 클라이언트에 비축되어 있는 데이터를 읽음으로써 서버와의 충돌이 발생할 수 있다. 반면에 지연 동기화 컴포넌트는 단편화가능객체에 대한 액세스만을 포함하는 부분으로 단편화가능객체는 이동 클라이언트가 마스터이고 서버와의 충돌 가능성도 없으므로 그 자체만으로는 동기화 실패 가능성이 없고 오직 우선 동기화 컴포넌트의 동기화 성공 여부에만 영향을 받는다. [정의 3]의 조건 3은 우선 동기화 컴포넌트에 대한 동기화 우선순위가 모든 지연 동기화 컴포넌트들에 대한 동기화 우선순위보다 높으므로 가장 먼저 서버와의 동기화를 실시하게 됨을 의미한다. 이동 클라이언트에서 로컬 커미트된 이동 트랜잭션의 우선 동기화 컴포넌트가 서버와의 동기화 작업에 성공하면, 이동 트랜잭션은 pre-commit 상태가 된다. SSMT 및 우선 동기화 컴포넌트의 정의에 의하여 pre-commit은 전역 커미트를 보장한다. pre-commit된 SSMT가 변경한 데이터는 서버에서 실행되는 다른 트랜잭션들에게 공개된다.

**[정의 4]** 이동 클라이언트에서 로컬 커미트되고 우선 동기화 컴포넌트에 대한 서버와의 동기화가 성공적으로 완료된 SSMT 이동 트랜잭션을 pre-commit 상태로 정의한다. pre-commit된 SSMT 트랜잭션은 최종 커미트가 보장되며, 동기화된 부분 결과를 공개할 수 있다.

아직 전역 완료되지 않고 pre-commit만 된 트랜잭션의 부분 결과를 공개하여 다른 트랜잭션이 사용함으로써 트랜잭션의 연쇄 철회 등이 발생할 수 있으나 pre-commit은 최종 커미트를 보장하므로 문제가 되지 않는다. 서버에서는 pre-commit된 SSMT 이동 트랜잭션에 대한 정보를 유지하고 있다가 pre-commit된 이동 트랜잭션의 마지막 컴포넌트에 대한 동기화 작업이 완료되면 이동 트랜잭션을 최종 커미트한다.

3.3.3 분할 동기화 이동 트랜잭션의 처리

SSMT를 지원하는 이동 클라이언트에서의 SSMT 처리방법은 다음과 같이 요약될 수 있다.

① 동기화 컴포넌트마다 우선 동기화 컴포넌트 여부, 동기화 우선순위 및 충돌해결방법을 지정하거나 이동 트랜잭션 전체에 대한 보상 트랜잭션을 정의
② SSMT를 실행
③ SSMT가 로컬 커미트되면, 컴포넌트 트랜잭션에서 변경한 내용을 동기화 우선 순위별로 할당된 동기화 메시지 큐에 저장. SSMT 실패 시 트랜잭션을 철회
④ 서버와 연결되었을 때, 우선 순위가 높은 동기화 메시지 큐에 저장된 컴포넌트 트랜잭션의 동기화 메시지들부터 서버에 전달

자세한 이동 클라이언트에서의 SSMT 처리 알고리즘이 (그림 3)에 기술되어 있다. 이동 클라이언트에서는 서버와의 일시적인 연결기간 동안 동기화 우선순위에 따라 동기화 메시지를 전송하기 위하여 동기화 우선순위별로 별도의 동기화 메시지 큐인 SyncMsgQueue를 유지하고 동기화 우선순위에 따라 컴포넌트의 동기화 메시지를 큐에 삽입해야만 한다.

(그림 4)는 서버와 일시적인 접속 설정 시 수행되는 동기화 메시지 전송 알고리즘으로 동기화 우선순위 큐인 SyncMsgQueue에서 순서대로 동기화 메시지를 서버에 전송한다.

메시지 전송 중에 서버와의 접속이 끊어질 수 있으므로 전송이 완료된 동기화 메시지만을 큐에서 삭제한다.

```
// T : a SSMT transaction issued by Mobile Client
Algorithm processSSMT(T)
{
    lcommitflag = true ;
    for (each component transaction ti in T) {
        execute ti ;
        if (ti execution fail) {
                lcommitflag = false ;
                break ;
        }
        generate synchronization message SYNC_MSG for ti ;
    }
    if (lcommitflag) {
        for (each component transaction ti in T) {
                priority = ti's synchronization priority ;
                enqueue(SYNC_MSG, SyncMsgQueue priority) ;
        }
        local commit T ;
    }
    else abort T ;
}
```

(그림 3) 이동 클라이언트에서의 SSMT 처리 알고리즘

```
Algorithm sendSSMT()
{
    while (connect_server) {
       for (i = highest_priority ; i <= lowest_priority ;
            i = next_highest_proirity) {
            while (!empty(SyncMsgQueuei)) {
                SYNC_MSG = dequeue(SyncMsgQueuei) ;
                send SYNC_MSG to server ;
            }
       }
    }
}
```

(그림 4) 이동 클라이언트에서의 동기화 메시지 전송 알고리즘

(그림 5)은 우선순위에 따라 SyncMsgQueue에 삽입되었다가 서버로 전달되는 SSMT 동기화 메시지 형식이다. 전체 동기화 메시지 중에서 (total_ctr_cnt, ctr_id, priority)부분이 SSMT에서 분할동기화에 의해 발생되는 오버헤드 메시지에 해당한다.

| MC_id | tr_id | total_ctr_cnt | ctr_id | priority | sync_contents |
|---|---|---|---|---|---|

MC_id : 이동 클라이언트 ID
tr_id : 이동 트랜잭션 ID
total_ctr_cnt : 컴포넌트 트랜잭션 총 개수
ctr_id : 컴포넌트 트랜잭션 ID(priority 순)
priority : 컴포넌트 우선순위
sync_contents : 동기화 내용

(그림 5) 동기화 메시지 형식

이동 클라이언트와 일시적으로 연결되었을 때, 서버는 다음과 같은 동기화 과정을 수행한다.

① 서버는 수신된 동기화 컴포넌트 트랜잭션의 내용을 서버 데이터베이스에 반영한다. 컴포넌트 트랜잭션에 대한 동기화가 성공하고 이 컴포넌트가 우선 동기화 컴포넌트라면, 이동 트랜잭션을 pre-commit하고 이 트랜잭션에 의해 변경된 부분 결과를 다른 트랜잭션들에게 공개한다. 우선 동기화 컴포넌트의 동기화 작업에 실패했다면, 트랜잭션을 철회하고 이동 클라이언트에게 트랜잭션의 철회내용을 전송한다.

② 서버가 pre-commit된 SSMT의 모든 컴포넌트 트랜잭션의 내용을 서버 데이터베이스에 성공적으로 적용했다면, 최종적으로 이동 트랜잭션을 전역 커미트한다.

```
// msgPacket format : (MC_id, tr_id, total_ctr_cnt, ctr_id, priority,
sync_contents)
Algorithm receiveSSMT(msgPacket)
{
    if ((MC_id, tr_id) transaction is a new transaction) {
       start (MC_id, tr_id) transaction ;
    }
    else if ((MC_id, tr_id) transaction is active transaction) {
       synchronize (MC_id, tr_id, ctr_id) component transaction ;
       if (ctr_id component transaction is PCOMPONENT) { //
       우선 동기화 컴포넌트
         if (ctr_id component transaction's synchronization fail) {
             abort (MC_id, tr_id) transaction ;
             send (MC_id, tr_id) transaction synchronization fail
             message to mobile client ;
         }
         else pre-commit (MC_id, tr_id) transaction ;
       }
       else {
         if (ctr_id component transaction is last component of
         (MC_id, tr_id) transaction) {
              global commit (MC_id, tr_id) transaction ;
         }
       }
    }
    else if ((MC_id, tr_id) transaction is aborted)
         ignore SSMT synchronization message packet ;
}
```

(그림 6) 서버에서의 동기화 메시지 처리 알고리즘

서버가 이동 클라이언트로부터 (그림 5)와 같은 형태의 동기화 메시지 패킷을 수신하였을 때, 수행되는 동기화 알고리즘이 (그림 6)에 기술되어 있다. 우선 동기화 컴포넌트가 서버와의 동기화 작업에 실패한다면, 서버에서는 이동 트랜잭션을 철회하고 이후에 들어오는 동일한 이동 트랜잭션에 대한 동기화 요청을 무시한다. 이동 클라이언트에 동기화 실패 메시지를 전송하여 우선 동기화 컴포넌트의 동기화에 실패한 이동 트랜잭션에 대한 동기화 정보를 더 이상 전송하지 않도록 한다. 이동 클라이언트에서는 동기화 실패 메시지를 서버로부터 받을 경우에 동기화 메시지 큐에 저장되어 있는 동기화 컴포넌트의 동기화 정보들을 삭제하고 각 트랜잭션에서 동기화 컴포넌트 별로 정의되어 있는 충돌해결함수 또는 보상 트랜잭션을 실행하여 로컬 커미트된 SSMT

트랜잭션을 취소(undo)한다.

3.3.4 Correctness 증명

이동 클라이언트에서 서버로 전송된 SSMT의 동기화 컴포넌트들은 분할 동기화가 진행되는 과정에서 서버 데이터베이스에 대한 연산을 수행하면서, 서버에서 제출되어 수행되는 서버 트랜잭션들과 함께 전역 히스토리(Global History)를 생성한다.

직렬화 가능한(serial) 히스토리는 모든 트랜잭션 쌍 $T_i$와 $T_j$에 대하여 $T_i$에서 수행된 모든 연산들이 $T_j$에서 수행된 모든 연산들보다 먼저 수행되거나 나중에 수행되어야만 한다. 동일한 트랜잭션들의 집합으로 구성된 히스토리 H와 H′는 서로 충돌하는 연산들의 순서가 같은 경우에 서로 동치(equivalent)가 되고 직렬화 가능한 히스토리와 동치인 임의의 히스토리를 conflict serializable이라고 한다[3]. 주어진 히스토리가 conflict serializable인지를 결정하기 위한 방법으로 각 노드가 트랜잭션을, $T_i$가 $T_j$의 연산과 충돌이 발생하는 연산을 포함하고 있으면서, $T_j$보다 먼저 수행됨을 나타낼 경우에 $T_i \rightarrow T_j$를 링크로 포함시키는 직렬화 그래프 SG(H)를 분석하면 된다. 임의의 히스토리 H는 SG(H)가 사이클을 형성하지 않는다면, conflict serializable이다[3]. 실행 히스토리는 각 데이터 항목에 대한 연산과 함께 잠금(lock)과 잠금해제(unlock) 연산을 포함한다. 각 연산 $p_i(x)$에 대해 잠금 연산 $pl_i(x)$가 먼저 발생하고 잠금해제 연산 $pul_i(x)$가 $p_i(x)$ 이후에 발생한다.

SSMT 분할 동기화 이동 트랜잭션 모델의 정확성을 증명하기 위해 트랜잭션 처리 알고리즘의 정확성 증명에 일반적으로 사용하는 직렬성 정리(serializability theorem)를 사용하도록 한다[3]. 서버에서는 공유 데이터에 대한 잠금 알고리즘으로서 2단계 잠금(2 phase locking) 알고리즘을 사용한다고 가정한다. 분할 동기화 이동 트랜잭션 모델 및 2단계 잠금 알고리즘에 기초하여 다음과 같은 property들을 기술한다.

**[Property 1]** 트랜잭션을 구성하는 임의의 연산 $o$에 대하여, $ol(x) < o(x) < oul(x)$이다.

**[Property 2]** SSMT 트랜잭션 $T_i$에 대하여, $pc_i$와 $c_i$가 각각 pre-commit과 commit이라 할 때, $pc_i <_i c_i$이고, $pc_i < pc_j$라면 $c_i < c_i$이다.

**[Property 3]** $pc_i \in T_i$라면, $a_i \notin T_i$이다.

2단계 잠금 알고리즘으로부터 다음과 같은 property를 얻을 수 있다.

**[Property 4]** 서버 트랜잭션 $T_i$에 대하여, $p_i(x)$와 $q_i(y)$가 $T_i$를 구성하는 연산들이라면, $pl_i(x) < qul_i(y)$이다. 즉, 모든 잠금 연산 $l_i \in T_i$와 잠금해제 연산 $ul_i \in T_i$에 대하여, $l_i < ul_i$이다.

pre-commit 및 SSMT 우선 동기화 컴포넌트 트랜잭션의 정의에 의해 다음의 property를 얻을 수 있다.

**[Property 5]** SSMT 트랜잭션 $T_i$에 대하여, $p_i(x)$와 $q_i(y)$가 $T_i$의 우선 동기화 컴포넌트를 구성하는 연산들이라면, $pl_i(x) < qul_i(y) < pc_i$이다. 즉, SSMT의 우선 동기화 컴포넌트가 동기화에 성공하여 pre-commit할 경우에, 우선 동기화 컴포넌트 내의 연산에서 취득하였던 데이터 항목에 대한 모든 잠금을 해제하여, 다른 서버 트랜잭션에서 사용할 수 있도록 한다.

**[Property 6]** $p_i(x)$와 $q_j(x)$가 서로 충돌이 발생하는 연산들이라면 다음을 만족한다.

(1) $pul_i(x) <_H ql_j(x)$. *or*

(2) $qul_j(x) <_H pl_i(x)$.

위의 성질들을 만족하는 히스토리는 사이클을 가지지 않는 직렬화 그래프를 가진다는 것을 보이고자 한다. 히스토리 H에 대하여, SG(H)는 각 트랜잭션에 해당하는 노드들을 가지고, $T_i$가 연산 $p_i$를 가지고, $T_j$의 연산 $q_j$와 충돌이 발생한다면, $T_i \rightarrow T_j$를 연결선으로 가진다. SG(H)의 각 노드는 서버에서 제출되어 수행되는 서버 트랜잭션 또는 서버에서 동기화가 진행 중인 SSMT 트랜잭션일 수 있다.

**[Lemma 1]** $T_1 \rightarrow T_2$가 SG(H)에 있다면, 모든 잠금연산 $ql_2 \in T_2$에 대하여 $pul_1(x) < ql_2(x)$인 잠금해제 연산 $pul_1 \in T_1$이 존재한다.

**[증명]** $T_1 \rightarrow T_2$이므로 $p_1(x) < q_2(x)$인 서로 충돌하는 연산 $p_1(x)$와 $q_2(x)$가 존재해야만 한다. Property 1에서 다음을 얻을 수 있다.

(a) 연산 $p_1(x)$에 대하여, $pl_1(x) < p_1(x) < pul_1(x)$이다.

(b) 연산 $q_2(x)$에 대하여, $ql_2(x) < q_2(x) < qul_2(x)$이다.

Property6에 의하여,

(1) $pul_1(x) < ql_2(x)$ *or*

(2) $qul_2(x) < pl_1(x)$

(2)번의 경우에 (a), (b)를 적용하면, $q_2(x) < qul_2(x) < pl_1(x) < p_1(x)$를 얻을 수 있고, 이는 $p_1(x) < q_2(x)$와 모순된다. 따라서, $pul_1(x) < ql_2(x)$ 이어야만 한다.

**[Lemma 2]** $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n (n>1)$을 SG(H)에 존재하는 경로라고 가정할 때, H안에 존재하는 데이터 객체 x와 y, 임의의 연산들 $p_1(x)$와 $q_n(y)$에 대하여 $pul_1(x) < ql_n(y)$이 만족한다.

**[증명]** 수학적 귀납법에 의하여, $n = 2$인 경우에는 Lemma1을 따른다. lemma가 $k \geq 2$에 대하여 만족한다고 가정하고, $n = k+1$일 때도 만족한다는 것을 보이고자 한다. 귀납적 가정

(induction hypothesis)에 의해 $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_k$의 경로는 H내에 $pul_1(x) < ol_k(z)$를 만족하는 데이터 항목 x와 z, 연산 $p_1(x)$와 $o_k(z)$가 있다는 것을 의미한다. $T_k \rightarrow T_{k+1}$과 Lemma1에 의해, 데이터 항목 y와 $o'ul_k(y) < ql_{k+1}(y)$를 만족하는 서로 충돌하는 연산 $o'_k(y)$와 $q_{k+1}(y)$이 존재한다. $T_k$와 $T_{k+1}$가 각각 동기화가 진행 중인 SSMT 트랜잭션과 서버에서 제출되어 수행되는 트랜잭션일 경우, $q_{k+1}(y)$과 충돌하는 연산 $o'_k(y)$는 SSMT의 우선 동기화 컴포넌트의 정의에 의해 SSMT 내의 우선 동기화 컴포넌트 내에 포함되어야만 한다. $T_{k+1}$가 SSMT 트랜잭션인 경우에도 마찬가지로 $q_{k+1}(y)$가 SSMT 내의 우선 동기화 컴포넌트 내에 포함되어야만 한다. 따라서 Property 4와 Property 5에 의하여, $ol_k(z) < o'ul_k(y)$를 얻을 수 있다. $pul_1(x) < ol_k(z)$와 $ol_k(z) < o'ul_k(y)$, $o'ul_k(y) < ql_{k+1}(y)$를 결합함으로써 $pul_1(x) < ql_{k+1}(y)$를 얻을 수 있다.

**[Theorem 1]** SSMT 트랜잭션 모델과 2단계 잠금 프로토콜에 의해 얻어지는 모든 히스토리 H는 직렬화 가능하다.

**[증명]** 모순(contradiction)에 의해, SG(H)가 사이클 $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n \rightarrow T_1 (n > 1)$를 가진다고 가정하자. Lemma 2에 의해, H안에 존재하는 연산 중에, $pul_1(x) < ql_1(y)$인 연산 $p_1(x)$와 $q_1(y)$가 존재한다. 이것은 Property 4와 Property 5에 위배된다. 따라서, SG(H)는 사이클을 가지지 않고, 직렬성 정리에 의하여 H는 직렬화 가능하다.

## 4. 성능 평가

분할 동기화 이동 트랜잭션 모델의 성능평가를 위해 편재형 컴퓨팅 환경에서의 e-비즈니스 응용의 대표적인 사례라고 할 수 있는 이동 세일즈 응용을 적용 사례로서 제시하고 이를 이용하여 성능평가를 실시하였다. 성능실험에서는 시뮬레이션을 통해 이동 세일즈 응용에 분할 동기화 이동 트랜잭션 모델을 적용하였을 때, 트랜잭션 단위로 동기화를 실시하는 기존의 방법들보다 이동 클라이언트에서 이동 트랜잭션에 의해 변경된 데이터에 대한 서버에서의 가용성이 증가되고, 무선으로 전송되는 패킷양을 감소시킴으로써 무선 통신 요금 및 이동 단말기의 배터리 소모량을 절약하는 효과를 가져올 수 있음을 보인다.

### 4.1 성능평가 모델

통상적인 이동 세일즈 환경을 모델링하기 위한 방법으로서 상용 온라인 비즈니스 트랜잭션 처리 시스템의 성능을 시험하기 위한 목적으로 1992년에 발표되어 현재까지도 널리 사용되고 있는 TPC-C 벤치마크[13]에서 정의되어 있는 데이터모델 및 트랜잭션 모델을 이동 컴퓨팅 환경 및 본 연구의 목적에 맞도록 간략하게 재구성하였다.

이동 세일즈 응용을 위한 시나리오는 다음과 같다. 이동 세일즈 사원은 이동 단말기를 휴대하고 고객을 방문하여 주문을 접수하거나 결제를 하는 등의 다양한 비즈니스 트랜잭션을 수행한다. 회사는 고객정보와, 상품정보, 세일즈사원 정보 등을 통합 관리하고 세일즈 사원이 접수한 주문을 처리하는 엔터프라이즈 데이터베이스 서버를 운영한다. 세일즈 사원은 세일즈를 나가기 전에 회사의 엔터프라이즈 서버와 유선으로 접속하여 그날의 비즈니스에 필요한 담당 고객 정보 및 물품정보 등의 데이터를 이동 단말기에 비축하는 체크아웃 과정을 실행한다. 체크아웃이 종료된 후 사무실 밖에서 이동 비즈니스를 수행하는 이동 세일즈 사원은 무선 통신 요금 및 이동 단말기 배터리 절약 등의 이유로 접속단절 상태에서 비즈니스 트랜잭션을 처리한다. 이동 단말기는 미리 정해진 데이터충전 스케쥴 또는 필요시에 서버와 무선 연결을 설정하고 시급하게 필요한 데이터의 다운로드와 함께 접속단절동안 처리된 이동 트랜잭션에 대한 동기화작업을 수행한다. 이동 단말기는 다시 접속단절 상태에서 이동 비즈니스 트랜잭션을 처리하게 되고 계속 이러한 과정을 반복하다가 그날의 비즈니스를 종료하고 회사로 돌아와서 체크인을 통하여 이동 단말기에 비축된 데이터에 대한 최종적인 동기화과정을 실행한다.

#### 4.1.1 데이터베이스 스키마

이동 세일즈 사례에서 사용되는 데이터베이스 스키마는 관계형 모델을 사용하였을 때, (그림 7)과 같은 8개의 테이블로 표현될 수 있다. 밑줄 그어진 필드들은 각 테이블의 기본키를, *표시가 된 필드들은 다른 테이블의 필드를 참조하는 외래키임을 나타낸다. 각 테이블의 의미와 테이블을 구성하는 필드들에 대한 자세한 설명은 "TPC Benchmark C" 표준 규격을 참조하기 바란다[13].

```
DISTRICT(D_ID, D_NAME, D_ADDRESS, D_TAX, D_YTD,
    D_NEXT_O_ID)
CUSTOMER(C_ID, C_D_ID*, C_NAME, C_ADDRESS, C_PHONE,
    C_DISCOUNT, C_BALANCE)
HISTORY(H_C_ID*, H_C_D_ID*, H_D_ID*, H_DATE, H_AMOUNT)
ORDERS(O_ID, O_D_ID*, O_C_ID*, O_ENTRY_D, O_OL_CNT)
ORDER_LINE(OL_O_ID*, OL_D_ID*, OL_NUMBER, OL_I_ID*,
    OL_DELIVERY_D, OL_QUANTITY, OL_AMOUNT)
NEW_ORDER(NO_O_ID*, NO_D_ID*)
ITEM(I_ID, I_NAME, I_PRICE, I_DATA)
STOCK(S_I_ID*, S_QUANTITY, S_YTD, S_ORDER_CNT)
```

(그림 7) 이동 세일즈 데이터베이스 스키마

이동 세일즈 사원은 체크아웃을 수행하여 서버 데이터베이스에 저장된 단편화가능객체인 DISTRICT, CUSTOMER, HISTORY 테이블들로부터 자신의 D_ID와 일치하는 레코드들만을 다운로드하여 이동 데이터베이스에 비축한다. 반면에 ITEM 테이블은 단편화불가능객체이므로 전체 레코드들을 모두 다운로드하여 이동 데이터베이스에 저장하게 된다.

4.1.2 이동 비즈니스 트랜잭션

이동 세일즈 사원들은 고객들을 방문하여 신규 주문을 내거나 상품 구매 대금을 지불하고 주문 내역을 확인하는 등의 이동 비즈니스 트랜잭션을 제출할 수 있다. TPC-C 벤치마크에 정의되어 있는 세일즈 트랜잭션들과 발생빈도는 신규 주문(NewOrder) 45%, 결제(Payment) 43%, 주문 상태 조회(OrderStatus) 4%, 배달(Delivery) 4%, 재고 조사(StockLevel) 4%로 설정되어 있다[13]. 이들을 이동 비즈니스 환경에 적용해보면, NewOrder, Payment, OrderStatus 트랜잭션은 이동 세일즈 사원의 단말기, 즉 이동 클라이언트에서 제출되어 처리되는 이동 비즈니스 트랜잭션으로, Delivery, StockLevel 트랜잭션은 서버 또는 고정 클라이언트에서 제출되어 처리되는 일반 트랜잭션으로 구분할 수 있다. 본 연구에서 관심이 있는 트랜잭션은 서버에서 제출되거나 단순히 테이블 내용을 조회만 하는 트랜잭션이 아니라 이동 클라이언트에서 제출되고 접속단절

**(1) *MobileTransactionNewOrder( )***

```
{
    // get $customername from customer, $d_id is salesman's id
    select $c_id from CUSTOMER where C_NAME = $customername ;
    select $c_discount from CUSTOMER where C_ID = $c_id ;
    select $d_next_o_id, $d_tax from DISTRICT where D_ID = $d_id ;
    update DISTRICT set D_NEXT_O_ID = $d_next_o_id + 1 where
    D_ID = $d_id ;

    $o_id = $d_next_o_id ;
    // get $o_ol_cnt(order count) from customer
    $total = 0 ;
    for (ol_number=1 ; ol_number <= $o_ol_cnt ; ol_number++) {
        // get $i_id(item id) & $ol_quantity(order quantity) from
          customer
        select $i_price, $i_name, $i_data from ITEM where i_id = $i_id ;
        $ol_amount = $ol_quantity * $i_price * (1+$d_tax) *
        (1-$c_discount) ;
        $total += $ol_amount ;
        insert into ORDER_LINE (ol_o_id, ol_d_id, ol_number, ol_i_id,
          ol_quantity, ol_amount)
              values ($o_id, $d_id, $ol_number, $i_id, $ol_quantity,
              $ol_amount) ;
    }
    insert into ORDERS (o_id, o_d_id, o_c_id, o_entry_d, o_ol_cnt)
              values ($o_id, $d_id, $c_id, $datetime, $o_ol_cnt) ;
    insert into NEW_ORDER (no_o_id, no_d_id) values ($o_id, $d_id) ;
    update CUSTOMER set C_BALANCE = C_BALANCE - $total where
              C_ID = $c_id ;
    local commit ;
}
```

**(2) SSMT NewOrder**

```
BEGIN TR
        BEGIN PCOMPONENT
              insert record into ORDER_LINE( … ) ;
              …
              insert record into ORDER_LINE( … ) ;
              insert record into NEW_ORDER( … ) ;
        END PCOMPONENT
        BEGIN COMPONENT
              insert record into ORDERS( … ) ;
              update DISTRICT record ;
              update CUSTOMER record ;
        END COMPONENT
END TR
```

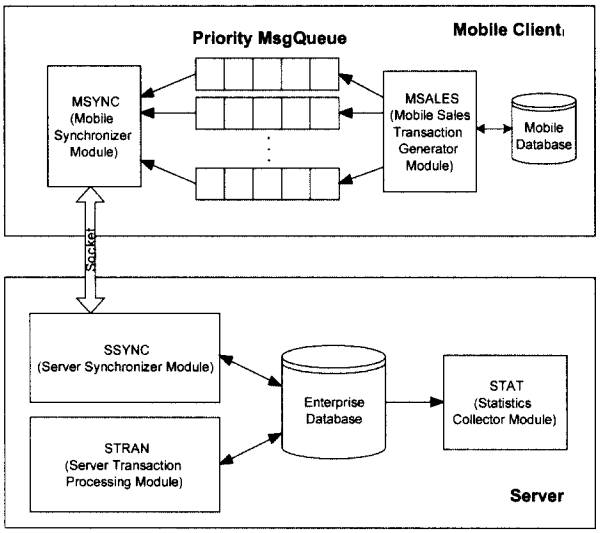
(그림 8) NewOrder 트랜잭션

상태에서 처리되며 비축된 데이터 테이블의 내용을 변경시키는 NewOrder, Payment 트랜잭션이므로 NewOrder, Payment 트랜잭션을 위주로 성능평가 실험을 실시하였다.

(그림 8)은 이동 클라이언트에서 수행되는 NewOrder 트랜잭션을 위한 알고리즘이다. NewOrder 트랜잭션이 성공적으로 수행되어 이동 클라이언트에서 로컬 커미트되면 서버와의 동기화를 위해 (그림 8)(2)와 같은 SSMT 트랜잭션이 NewOrder 트랜잭션으로부터 생성된다. (그림 8)(2)는 SSMT 트랜잭션의 구조만을 파악할 수 있도록 간략하게 기술된 것으로 컴포넌트 트랜잭션 내의 DML 문장은 (그림 8)(1)에 기술되어 있는 각 DML 문장들과 같이 구체적인 변경 내용을 포함하고 있으며, 서버에 전송되어 서버와 동기화되는 시점에서 서버에 저장되어 있는 원본 테이블에 대해서 실행되는 부분이다. SSMT NewOrder에서는 서버에서의 사용 가능성과 충돌 가능성에 따라 ORDER_LINE과 NEW_ORDER 테이블을 변경하는 부분을 동기화 우선순위가 가장 높은 PCOMPONENT에 포함시킨 반면, 단편화가능객체이면서 서버에서 사용될 가능성이 낮은 DISTRICT, CUSTOMER, ORDERS 테이블에 대한 변경내용은 우선순위 값을 부여하지 않음으로써 체크인 시에 동기화가 진행되도록 설정하고 있다. 트랜잭션 Payment를 위한 이동 클라이언트에서의 처리 알고리즘 및 생성되는 SSMT 트랜잭션의 형태도 이와 유사한 구조를 가진다.

4.2 성능평가 시스템 모델

(그림 9)는 SSMT 모델의 성능평가를 위한 시뮬레이션 시스템 구조이다. 시뮬레이션 시스템은 C언어와 ORACLE PRO*C를 이용하여 Solaris 2.5.6 운영체제를 사용하는 SUN Ultra Sparc 2.0 상에서 구현되었다. 엔터프라이즈 서버에 탑재되어 운영되는 엔터프라이즈 데이터베이스는 ORACLE 9i를 이용하여 구축하였다[14]. 서버에서 이동 클라이언트로부터 수신된 SSMT 컴포넌트 트랜잭션 패킷에 대한 동기화 과정은 별도의 동기화 전용 모듈인 SSYNC에서 처리된다. SSYNC와는 별도로 수행되는 프로세스인 STRAN은 서버 트랜잭션을 처리하는 모듈로서, SSYNC에 의해 성공적으로 서버 데이터베이스에 반영된 SSMT 트랜잭션의 변경 사항을 포함한 엔터프라이즈 데이터베이스에 대해 Delivery등과 같은 서버 트랜잭션을 처리한다. STAT은 성능평가를 위한 통계를 수집하는 모듈이다. 각 이동 클라이언트 MobileClient$_i$는 체크아웃에 의해 서버로부터 비축된 이동 데이터베이스로부터 SSMT들을 생성하여 로컬 커미트된 트랜잭션의 컴포넌트 트랜잭션들의 동기화 패킷을 부여된 우선순위와 일치하는 MsgQueue에 추가하는 이동 세일즈 트랜잭션 생성기 MSALES를 가진다. MSYNC는 MsgQueue로부터 패킷을 꺼내어 우선순위 별로 정해진 스케쥴에 따라 서버와의 무선 접속을 설정하고 패킷을 서버에 전송한다. <표 1>은 시뮬레이션에서 사용된 파라미터들이다.




(그림 9) 성능평가 시뮬레이션 시스템 구조

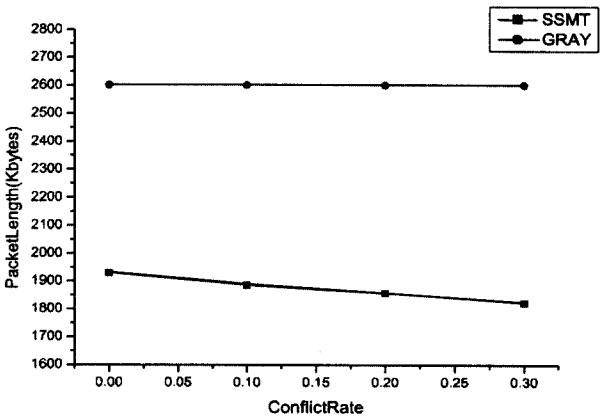
〈표 1〉 시뮬레이션 파라미터

| 파라미터 | 의 미 | 값 |
|---|---|---|
| NumOfMC | 이동 클라이언트 수 | 30 |
| CustPerDist | 이동 세일즈 사원 당 담당 고객 수 | 3000 |
| NumOfItems | 주문 상품 품목 개수 | 10000 |
| OlNumber | 각 주문 당 세부 주문 항목 수 | 5~15 |
| Mobile-SyncInterval | 이동 클라이언트와 서버와의 동기화 수행 시간 간격 | 30분 |
| PcompRate | 전체 트랜잭션 중 우선 동기화 컴포넌트의 구성 비 | 75% |
| Mobile-ThinkTime | 이동 클라이언트에서의 평균 트랜잭션 도착 시간 간격(지수분포) | 시뮬레이션 파라미터 (2~10분) |
| Disconnect-Rate | 서버와 무선 연결 상태에서 동기화 중, 접속단절 발생 확률(베르누이분포) | 시뮬레이션 파라미터 (0.0~0.4) |
| ConflictRate | 우선 동기화 컴포넌트 동기화 시 서버와 충돌 발생 확률(베르누이분포) | 시뮬레이션 파라미터 (0.0~0.4) |

CustPerDist, NumOfItems, OlNumber는 TPC-C 벤치마크에서 정의되어 있는 파라미터들로 NewOrder 및 Payment 트랜잭션을 랜덤하게 생성하는데 사용된다. 이렇게 생성된 NewOrder 트랜잭션은 OlNumber + 3, Payment 트랜잭션은 3개의 연산들로 구성된다. MobileSyncInterval은 이동 클라이언트와 서버와의 동기화 스케쥴을 나타내는 파라미터로서 본 실험에서는 각 이동 클라이언트에서 MobileSyncInterval마다 서버와의 동기화 과정을 규칙적으로 수행하도록 하였다. PcompRate은 하나의 트랜잭션에서 우선 동기화 컴포넌트가 차지하는 비율을 의미한다. PcompRate가 클수록 하나의 트랜잭션에서 동기화 패킷을 무선 통신망을 이용하여 전송하는 비율이 높아지게 된다. MobileThinkTime은 각 이동 클라이언트에서의 이동 트랜잭션 발생률을 조절하기 위한 파라미터로 지수분포를 따르는 평균 트랜잭션 도착시간 간격을 나타낸다. DisconnectRate는 불완전한 무선 통신 인프라와 무선 통신의 불안정성을 나타내는 시뮬레이션 파라미터로 서버와 이동 클라이언트 간, 무선 통신망을 이용하여 동기화를 진행하는 중에 이동 세일즈 사원이 무선 통신 음영지역에 진입하거나 전파 간섭 등으로 인해 예기치 않은 접속단절이 발생하였을 때, 동기화 알고리즘의 성능을 측정하기 위하여 사용된다. ConflictRate는 접속단절 상태에서 이동 클라이언트에 비축된 데이터를 사용하여 수행된 이동 트랜잭션이 서버와의 동기화에 실패할 확률로, 서버에서 이동 클라이언트가 캐시해간 단편화불가능객체의 값을 변경한 경우를 나타내며, 분할 동기화 알고리즘의 빠른 충돌탐지에 따른 전송 패킷 감소율을 측정하는데 사용된다.

### 4.3 성능평가 결과 및 분석

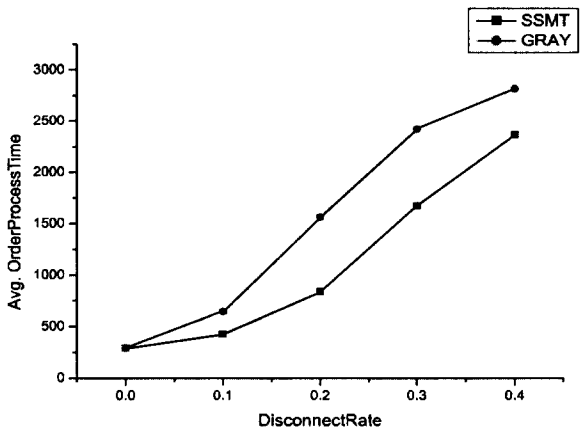
분할 동기화 트랜잭션 처리 모델의 성능을 평가하기 위해 접속단절동안 이동 클라이언트에 비축된 데이터에 대해 임시 트랜잭션을 수행하고, 서버와 연결된 후 임시 트랜잭션의 내용을 서버로 전달하여 트랜잭션 단위로 동기화를 시행하는 Gray의 2단계 중복 모델[5]과의 성능비교를 실시하였다. 동기화 알고리즘의 성능평가 척도로서 실험의 종류에 따라 이동 단말기의 배터리 소모량과 무선 통신 비용을 의미하는 무선 전송 패킷양과 이동 클라이언트가 변경한 데이터에 대한 서버에서의 가용성을 나타내는 평균주문처리시간(average order processing time)을 사용한다. 평균주문처리시간은 이동 클라이언트에서 NewOrder 트랜잭션이 발생한 시점에서부터 NewOrder 트랜잭션이 변경한 데이터를 서버 트랜잭션인 Delivery 트랜잭션이 사용하게 되는 시점 사이의 시간 간격을 나타낸다.



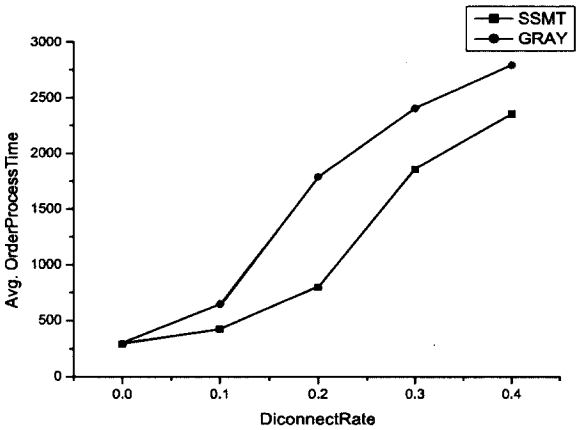

(그림 10) ConflictRate 변화에 따른 무선 전송 패킷양

(그림 10)은 접속단절 상태에서 수행된 이동 트랜잭션이 서버와의 동기화에 실패할 확률인 ConflictRate를 변화시키면서 전체 시뮬레이션 시간동안 무선으로 전송되는 총 패킷양을 측정한 결과이다. GRAY는 ConflictRate의 변화에 관계없이 무선 전송 패킷양이 일정한데 비하여, SSMT는 동

기화 실패확률의 변화에 비례하여 무선 전송 패킷량이 선형(linear)으로 감소한다. 이것은 SSMT 트랜잭션 처리 알고리즘이 서버와의 충돌 가능성이 있는 우선 동기화 컴포넌트를 먼저 서버에 전송하여 동기화를 시키고, 동기화에 실패할 경우 이동 클라이언트에 동기화 실패 메시지를 전송하여 더 이상 동기화에 실패한 트랜잭션의 메시지를 서버로 전송하지 않기 때문이다. 따라서 동기화실패 확률이 높아질수록, 서버에 무선으로 전송되는 패킷양이 감소하기 때문에 결과적으로 무선 통신 비용 및 이동 단말기의 배터리 소모를 절약하는 효과를 기대할 수 있다. (그림 10)에서 ConflictRate = 0.0인 경우에도 SSMT가 GRAY에 비해 무선 전송 패킷양이 상대적으로 작은 것을 확인할 수 있는데, SSMT와 GRAY 간의 무선 전송 패킷양 차이의 절대 수치는 동일한 트랜잭션을 구성하는 우선 동기화 컴포넌트와 지연 동기화 컴포넌트의 구성 비율에 따라 결정된다. (그림 10)는 전체 트랜잭션에서 우선 동기화 컴포넌트가 차지하는 비율인 PcompRate이 75%에 달하고, 동기화 실패 시 전송되지 않는 부분인 지연 컴포넌트가 차지하는 부분은 25% 밖에 되지 않는 트랜잭션 내의 컴포넌트 구성비하에서 이루어진 결과이다. 따라서 동기화 실패 시 전송되지 않는 패킷의 양은 25%내에서만 감소되기 때문에 ConflictRate이 증가함에 따른 전송 패킷 감소율은 상대적으로 완만한 하강곡선을 그리지만, PcompRate이 감소할수록 전송 패킷 감소율은 크게 증가할 것이다.




(a) ConflictRate = 0.0, MobileThinkTime = 5분



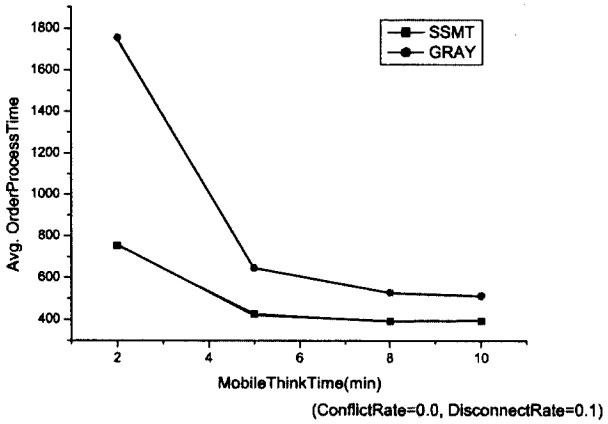

(b) ConflictRate = 0.1, MobileThinkTime = 5분

(그림 11) DisconnectRate 변화에 따른 평균주문처리시간

(그림 11)은 동기화 중 접속단절확률의 변화에 따른 평균 주문처리시간(단위시간 : 5초)의 변화를 보여준다. (a), (b)는 ConflictRate를 제외한 다른 파라미터들은 모두 동일한 상태에서 실험한 결과이다. (a), (b) 모두 동기화 중, 접속단절이 발생하지 않음을 의미하는 DisconnectRate = 0.0인 상태에서는 SSMT와 GRAY가 거의 비슷한 평균주문처리시간을 가지다가, DisconnectRate이 증가함에 주문처리시간 간격이 점점 벌어지다가, 다시 간격이 좁아지는 형태를 나타내고 있다. 이것은 SSMT에서는 이동 트랜잭션에서 서버에서의 가용성 및 충돌 가능성이 높은 컴포넌트들에 높은 우선순위를 부여하고 우선순위가 높은 동기화 패킷을 먼저 서버로 전송하므로 동기화 중에 접속단절이 발생하더라도 우선순위가 높은 컴포넌트들은 서버에 비교적 빨리 전송되어 서버에서 바로 사용할 수 있는 반면, GRAY의 방법에서는 전체 트랜잭션 단위로 동기화 패킷을 전송하므로 접속단절이 자주 발생할 경우에 서버와의 동기화작업이 SSMT보다 많이 지연되어 이동 클라이언트에서 변경된 데이터에 대한 서버에서의 가용성이 저하되기 때문이다. 주문처리시간 간격이 점점 벌어지다가 다시 간격이 좁아지는 것은 실험에서의 PcompRate이 높기 때문에 접속단절이 많이 발생할 경우 SSMT에서 우선순위가 높은 컴포넌트들의 동기화패킷들도 전송이 지연되어 DisconnectRate이 증가할 수록 SSMT와 GRAY의 주문처리시간이 점점 수렴하는 현상을 보인다. 그러나 실제 이동 컴퓨팅 환경에서는 비의도적인 접속단절 비율이 매우 높아지는 경우는 별로 없으므로 전반적으로 SSMT가 GRAY보다 주문처리율에 있어서 좋은 성능을 보인다고 할 수 있다. (그림 11)에서는 ConflictRate 값이 더 큰 (b)에서 SSMT와 GRAY 알고리즘의 성능 차이가 (a)보다 더 뚜렷이 나타나는 것을 볼 수 있는데, 이는 ConflictRate가 클수록 SSMT에서는 동기화에 실패한 트랜잭션의 나머지 동기화 내용은 서버로 전송하지 않으므로 결과적으로 서버로 전송되는 패킷양이 감소함에 따라 전체적인 전송대기시간이 감소하기 때문이다.

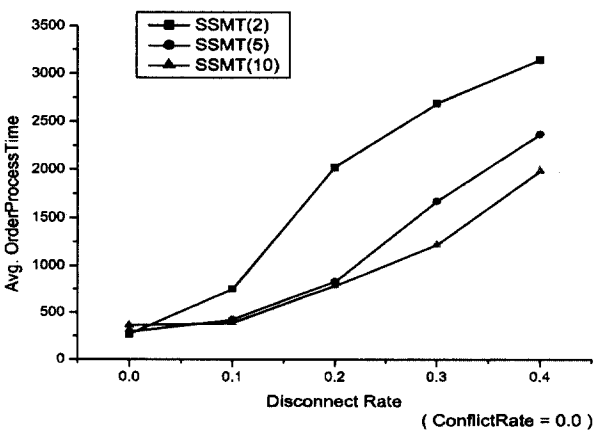
(그림 12)는 접속단절 가능성(0.1)의 존재 하에서 이동 클라이언트에서의 트랜잭션 도착시간 간격 변화에 따른 주문처리시간 변화를 나타내는 그래프로서 MobileThinkTime이 증가할수록 주문처리시간이 감소하는 경향을 보인다. 이는 MobileThinkTime이 작을 때, 이동 트랜잭션 발생률이 높아지므로 서버에 전송될 동기화 패킷량이 증가하여 접속단절

이 발생했을 때, 이동 클라이언트에서의 전송대기 시간이 길어짐에 따라 나타나는 현상이다. 이 때에도 SSMT가 GRAY보다 우수한 성능을 보이는데, SSMT는 접속단절이 발생한 후, 다음번 동기화 시간에 서버에서의 가용성을 높일 수 있는 우선순위가 높은 우선 동기화 컴포넌트 패킷들을 우선적으로 전송하므로, 서버에서 우선 동기화 컴포넌트들에 대한 동기화작업을 진행하여 트랜잭션을 pre-commit 시킴으로써 서버 트랜잭션이 이동 트랜잭션이 변경한 내용을 사용할 수 있기 때문이다.




(그림 12) MobileThinkTime 변화에 따른 평균주문처리시간

(그림 13)은 DisconnectRate를 변화시키면서 이동 트랜잭션 발생 간격을 각각 2분, 5분, 10분으로 변화시켰을 때의 SSMT의 평균주문처리시간을 측정한 그래프이다. 이 그래프를 보면, MobileThinkTime이 작을수록, DisconnectRate이 커질수록 평균주문처리시간이 증가함을 알 수 있다. 이는 MobileThinkTime이 작을수록 이동 트랜잭션 발생률이 높아지므로 서버에 전송될 동기화 패킷량이 증가하고, 접속단절이 자주 발생할수록, 서버로 전송되어야하는 동기화 패킷들이 밀려서 이동 클라이언트에서의 전송대기 시간이 길어지기 때문이다.




(그림 13) DisconnectRate 변화에 따른 SSMT간 평균주문처리시간

## 5. 결 론

본 연구에서는 제한된 무선통신 대역폭 및 불완전한 무선통신 인프라, 고가의 무선통신 요금, 이동 단말기의 배터리 용량등과 같은 이동 컴퓨팅 환경의 한계로 인해 유발되는 접속단절을 극복하기 위해, 접속단절 시 트랜잭션 처리에 필요한 데이터를 미리 이동 단말기에 비축한 후, 서버와의 접속단절 상태에서 이동 단말기에 비축된 데이터를 이용하여 이동 단말기에서 자치적으로 이동 트랜잭션을 수행하면서도 데이터 중복과 네트워크 분할로 인해 발생가능한 일관성 및 동기화 문제를 효율적으로 해결할 수 있는 분할 동기화 이동 트랜잭션 모델을 제안하고 성능평가를 실시하였다. 분할 동기화 이동 트랜잭션 모델에서는 하나의 트랜잭션을 트랜잭션에서 액세스하는 데이터 객체와 연산의 시맨틱에 따라 동일한 시맨틱을 갖는 컴포넌트 트랜잭션으로 분할한 후, 컴포넌트 별로 동기화 우선순위를 할당하여 우선순위에 따라 서버와의 동기화를 실시한다. 서버에서 충돌 가능성 및 사용 가능성이 높은 컴포넌트는 높은 우선순위를 부여하여 무선 연결 상태에서 서버와의 동기화를 실시함으로써 이동 클라이언트에서 변경된 데이터에 대한 서버에서의 가용성을 높이고, 그렇지 않은 컴포넌트 부분은 이동 단말기의 제한된 배터리 자원 및 무선 대역폭과 고가의 무선 통신 요금 등을 고려하여, 저렴한 통신망을 사용하여 서버에 나중에 반영함으로써 제한된 무선 대역폭 활용도를 극대화시키는 것이 가능하다. 성능평가 실험에서 이동 단말기의 배터리 소모량 및 무선 통신 비용과 비례하는 무선 전송 패킷양과 이동 클라이언트가 변경한 데이터에 대한 서버에서의 가용성을 측정한 결과, 제안한 알고리즘은 대체적으로 트랜잭션 단위의 동기화 알고리즘인 GRAY의 알고리즘보다 더 우수한 성능을 나타내었다. 또한 전체 트랜잭션 중에서 서버와 충돌 가능성을 가지고 있는 우선 동기화 컴포넌트가 차지하는 비율이 낮을수록, 트랜잭션 발생 빈도가 높을수록, 서버와의 동기화 실패 확률이 높을수록 기존의 트랜잭션 단위의 동기화 알고리즘에 비해 우수한 성능을 보인다.

본 연구에서 제안한 분할 동기화 이동 트랜잭션 모델은 지금 현재로서는 동기화 컴포넌트 및 컴포넌트의 동기화 우선순위를 결정하는 주체가 이동 클라이언트 응용 개발자로서 클라이언트 응용 개발 시, 사용자의 요구사항에 맞도록 컴포넌트 및 동기화 우선순위를 정하는 제한적인 방법을 사용하고 있다. 향후 과제로서 이 부분을 보완하여 실제 응용에 쉽게 적용할 수 있도록 컴포넌트의 동기화 우선순위를 결정하는 부분을 응용 개발자뿐만이 아니라, 사용자가 필요에 따라 동적으로 동기화 컴포넌트 및 동기화 우선순위를 설정할 수 있도록 하는 연구를 진행할 예정이다.

## 참 고 문 헌


[1] R. Alonso and H. Korth, "Database Issues in Nomadic Computing," In Proc. Of the ACM SIGMOD Conference on Management of Data, pp.388-392, 1993.

[2] B. R. Badrinath and S. Phatak, "Database Server Organization for Handling Mobile Clients," Technical Report DCS-342, Department of Computer Science, Rutgers University, 1997.

[3] P. Bernstein, V. Hadzilacos and N. Goodman, "Concurrency Control and Recovery in Database Systems," Addison-Wesley Publishing Co., 1987.

[4] P. K. Chrysanthis, "Transaction Processing in Mobile Computing Environment," In Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems, Princeton, New Jersey, pp.77-83, October, 1993.

[5] J. Gray, P. Helland, P. O'Neil and D. Shasha, "The Dangers of Replication and a Solution," In Proceedings of the ACM SIGMOD Conference, Montreal, Canada, pp.173-182, 1996.

[6] T. Imielinski and B. R. Badrinath, "Mobile Wireless Computing : Solutions and challenges in data management," Technical Report DCS-TR-296, Dept. of Computer Science, Rutgers Univ., New Brunswick, NJ08903, 1993.

[7] S. K. Madria, Bharat Bhargava, "A Transaction Model to Improve Data Availability in Mobile Computing," Distributed and Parallel Database, Vol.10, No.2, pp.127-160, 2001.

[8] P. Liu, P. Ammann, S. Jajodia, "Incorporating Transaction Semantics to Reduce Reprocessing Overhead in Replicated Mobile Data Applications," 19th IEEE International Conference on Distributed Computing Systems, May, 1999.

[9] S. H. Phatak, B. R. Badrinath, "Conflict Resolution and Reconciliation in Disconnected Databases," DEXA Workshop, pp.76-81, 1999.

[10] E. Pitoura, B. Bhargava and O. Wolfson, "Data Consistency in Intermittently Connected Distributed Systems," Technical Report DCS-96-10, Department of Computer Science, University of Ioannina, 1997.

[11] D. Terry, A. Demers, K. Petersen, M. Spreitzer, M. Theimer, and B. Welch, "Session Guarantees for Weakly Consistent Replicated Data," In Proceedings of the International Conference on Parallel and Distributed Information Systems, pp.140-149, Sep., 1994.

[12] G. Walborn and P. K. Chrysanthis, "Supporting Semantics-Based Transaction Processing in Mobile Database Applications," In Proceedings of the 14th Symposium on Reliable Distributed Systems, pp.31-40, Sep., 1995.

[13] "TPC Benchmark C," Standard specification. www.tpc.org.

[14] Oracle 9i Documentation, www.oracle.com.



**최 미 선**

e-mail : mschoi@cs.cnu.ac.kr

1989년 서울대학교 계산통계학과 학사

1991년 한국과학기술원 전산학과 석사

1991년~1995년 KBS 기술연구소 연구원

1995년~1998년 SK Telecom 중앙연구원 연구원

2002년 충남대학교 컴퓨터과학과 박사과정 수료

2002년~현재 우송대학교 컴퓨터학과 초빙교수

관심분야 : 이동 데이터베이스 시스템, 실시간 데이터베이스 시스템



**김 영 국**

e-mail : ykim@cs.cnu.ac.kr

1985년 서울대학교 계산통계학과 학사

1987년 서울대학교 계산통계학과 석사

1995년 Virginia 주립대학 Computer Science 박사

1996년~현재 충남대학교 정보통신공학부 교수

관심분야 : 실시간 데이터베이스 시스템, 전자상거래 시스템, 이동 데이터베이스 시스템