콘도 분산 시스템의 모바일 인터페이스
(Mobile Interface in Condor Distributed Systems)

이 송 이 †
(Song-Yi Yi)

요 약 콘도(Condor)는 네트워크로 연결된 컴퓨터 간에 작업을 공유할 수 있도록 하는 분산 일괄처리(batch) 시스템이다. 콘도 분산 시스템은 콘도 작업을 수행하기 위해 모든 기계가 언제나 네트워크에 연결되어있다는 가정 하에서 계발되었다. 그런데, 무선 통신과 이동 계산 기술이 급속히 발전함에 따라 이제 기존의 분산 컴퓨터 시스템은 고정된 클라이언트뿐만 아니라 이동하는 클라이언트도 포함할 수 있게 되었다. 또한, 이동컴퓨터의 사용자는 자원을 많이 소모할 가능성이 있는 작업들을 다른 곳에서 수행하도록 함으로써 가능한 한 전력을 적게 소모하길 원한다. 이 논문에서는 이동 클라이언트를 지원하기 위해 콘도 분산 배취 시스템(Condor distributed batch system)의 모바일 인터페이스의 설계와 구현 방안을 제시한다. 이 연구의 주요 목적은 이동 컴퓨터의 사용자가 콘도 시스템과의 연결에 관계없이 언제 어디서나 콘도 작업을 수행하도록 하는데 있다. 또한 기존 콘도 시스템을 많이 변경하지 않고서도 동일한 콘도 서비스를 이동하는 사용자에게 제공할 수 있도록 한다.
키워드 : 콘도 분산 시스템, 이동 클라이언트, 모바일 인터페이스

Abstract Condor is a distributed batch system for sharing the workload among the computers connected by a network. Condor distributed system was developed on the basis that every machine in a Condor pool is always connected by a network to run a Condor Job. Due to advances in wireless communication and mobile computing technology, conventional distributed computer systems can now include "mobile" clients as well as "fixed" clients. Moreover, mobile users want to lower their power consumption by off-loading potentially power and resource consuming jobs. In this paper, we describe the design and implementation of mobile interface for mobile clients in Condor distributed batch system. The main purpose of this work is to enable users on mobile computers to interact with Condor environment any time anywhere regardless of their connection to a Condor pool. The mobile Condor distributed system also aims to provide mobile users the same Condor services without making any significant changes to the existing Condor system.
Key words : Condor distributed system, mobile clients, mobile interface

1. 서 론

노트북 컴퓨터나 랩톱, PDA와 같은 이동 컴퓨터의 사용이 증가하고 점점 더 많은 응용프로그램이 이동 컴퓨터에 탑재될 수 있도록 계발되고 있다. 이에 따라, 여러 분산 화일 시스템이나 클라이언트 서버 시스템, 응용프로그램 등은 이동 사용자도 지원할 수 있도록 확장되고 있다[1-3]. 중앙 집중형 시스템과 분산 시스템 사이의 선택 문제는 곧 고정된 시스템과 이동 시스템사이의 선택 문제로 변경될 것이다. 그런데 대 다수의 현재 분산 시스템과 분산 시스템 프로토콜은 상대적으로 고속, 저 비용의 안정된 유선 데이타 링크를 대상으로 계발된 것으로 이동 컴퓨터에 적용하기에는 부 적절하다. 이동 컴퓨터는 그 특성 상 데스크 탑 컴퓨터에 비해 낮은 네트워크 성능과 고 비용이 문제되기 때문이다.

현재 기존의 분산 시스템이나 분산 알고리즘 및 응용프로그램 등이 이동 컴퓨터도 지원할 수 있도록 하는 여러 연구가 활발히 진행되고 있다. 이들 연구는 기존 시스템의 특성이나 응용 프로그램이 가지는 성격에 따라 다음 두 가지 방향 중 하나를 따르게 된다[2]. 한 가지 방법은 이동 컴퓨터를 위한 응용프로그램을 새로이 계발하는 방법이다. 전자 우편을 읽는 응용 프로그램(e-mail readers)이나 이동 컴퓨터용 웹 브라우저 등 많은 응용프로그램에 대해 이동 컴퓨터용 버전이 새로이 계발되어 사용되고 있다. 이러한 프로그램은 기존 응

† 정 회 원 : 서울대학교 BK21 정보기술사업단 교수
yis@snu.ac.kr
논문접수 : 2003년 9월 24일
심사완료 : 2003년 11월 21일

용 프로그램의 많은 부분을 변형시키고 수정하여 계발되거나 이동 컴퓨터상에서 효과적으로 작동할 수 있도록 새로이 계발된다.

두 번째 방법은 기존의 분산 시스템이나 응용프로그램은 조금만 수정하거나 거의 수정하지 않고, 이동 컴퓨터에서도 적절히 동작할 수 있도록 하는 작은 소프트웨어를 계발하여 덧붙이는 방법이다. 이 방법으로 계발된 이동 사용자를 지원하는 전자 우편 시스템은 [3]에서 찾아볼 수 있다. 이동 사용자를 위한 전자 우편 시스템은 기존의 전자 우편 시스템을 전혀 수정하지 않고 기존 전자 우편의 사용자 인터페이스를 지원하도록 하면서 이동 사용자에게도 동일한 전자우편 기능을 제공할 수 있도록 하고, 변화하는 네트워크 환경에 동적으로 적응할 수 있는 기능을 추가하였다.

이 논문에서 제안하는 이동 사용자를 위한 콘도 시스템도 두 번째 기법을 바탕으로 계발된 것이다. 이동 콘도 시스템은 기존의 콘도 분산 시스템을 기본 시스템으로 하여 이 시스템을 거의 수정하지 않고, 이동 클라이언트를 지원하기 위한 작은 프로그램들을 그 위에 덧붙이는 기법으로 구축하였다. 콘도(Condor)[4-7]는 네트워크로 연결된 워크스테이션들 간의 작업을 공유할 수 있게 하는 분산 일괄 처리 시스템으로 위스콘신 주립대학교에서 계발되어 현재도 성능과 기능이 계속 확장 개선되고 있다. 이 논문에서는 기존의 콘도 시스템이 이동 클라이언트를 지원할 수 있도록 기존의 콘도 시스템에 이동 클라이언트 인터페이스를 구현하는 방안을 제시한다. 만일 이동 컴퓨터의 사용자가 시간이 오래 걸리는 수백 번의 시뮬레이션 작업을 수행할 필요가 있다고 할 때, 자원의 재약과 "이동성" 때문에 자의적 혹은 타의적으로 네트워크와의 연결/해제가 반복되고 전원이 빈번하게 꺼지는 이동컴퓨터에서는 이러한 작업을 수행시킬 엄두도 내지 못하고 고정된 컴퓨터로 가서 콘도 작업을 수행시켜야 한다. 그러나 이동 컴퓨터에서 콘도 분산 시스템에 접근할 수 있는 인터페이스를 구현하면 이동 컴퓨터 사용자도 콘도 분산 배취 작업 시스템에 작업을 전송할 수 있게 되고, 콘도 풀에서 제출된 작업들이 모두 완료된 뒤에 결과 파일만 전송받으면 되므로 이동컴퓨터가 가지는 제약점과 상관없이 오래 걸리거나 많은 자원을 필요로 하는 작업도 수행할 수 있게 될 것이다. 그림 1은 이 논문에서 구현하고자 하는 이동 사용자와 콘도 풀(pool)이 무선 링크로 연결된 이동 콘도 시스템이다. 콘도 풀은 하나의 근거리 통신망으로 연결된 고정된 워크스테이션의 묶음(cluster)이다.

다음의 세 가지 기본 원칙이 이동 클라이언트의 콘도 인터페이스를 설계하는데 고려되었다.

1. 이동 클라이언트는 네트워크와의 연결성에 관계없이 언제나 콘도를 사용할 수 있도록(콘도에 언제나 연결된 것처럼) 인터페이스가 설계되어야 한다.
2. condor_submit, condor_q, condor_rm과 같은 콘도 인터페이스가 이동 사용자에게도 동일하게 제공되어야 한다.
3. 이동 클라이언트를 지원하는 것이 현존하는 콘도시스템의 많은 부분 수정을 필요로 해서는 안 된다.

이 논문의 구성은 다음과 같다. 2장에서는 콘도 분산 시스템의 간략한 개요와 기본 프로토콜을 설명한다. 3장에서는 현재 분산 시스템을 이동 클라이언트를 위해 확장할 때 생기는 설계 문제점과 개요를 제시한다. 4장에서는 이동 클라이언트를 위한 콘도 시스템의 모바일 인터페이스를 제시한다. 이동 클라이언트를 위한 확장된 콘도 시스템은 크게 두 부분으로 나뉘어진다. 이동 컴퓨터에서 콘도 작업을 제출하는 부분과 고정된 콘도 풀 상에서 작업의 수행이 완료된 후에 이동 클라이언트가 결과를 받아보는 부분이다. 이 두 부분이 이동 클라이언트를 지원하기 위해 어떻게 구현되는지 보여준다. 5장은

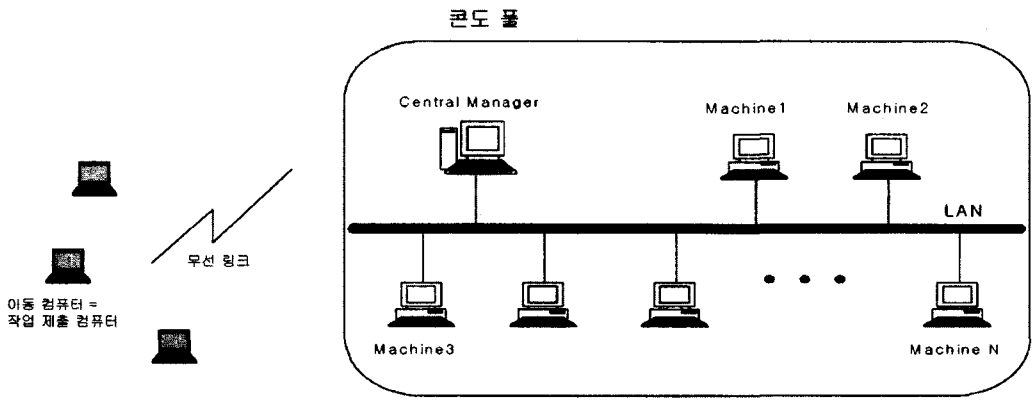


그림 1 이동 컴퓨터와 콘도 풀의 결합

이동 클라이언트의 지원과 기존 콘도의 매치메이커 프로세스와의 관계에 대하여 설명하고 6장은 향후 연구 방향과 결론을 제시한다.

2. 콘도 분산 시스템의 개요와 기본 프로토콜

콘도 풀은 여러 대의 고정된 컴퓨터로 구성된다. 이들 컴퓨터는 여러 가지 다양한 구조와 운영체제를 가지고 랜에 의해 연결되어 있다. 콘도 풀은 또한 원거리 통신망에 의해 여러 개의 풀이 연결되어 함께 동작할 수 도 있다[8]. 두 개의 프로세스 스케디(schedd)와 스타트디(startd)는 콘도 풀의 모든 컴퓨터에서 언제나 수행되고 있다[그림 2]. 스케줄러 데몬(scheduler daemon)인 스케디는 해당 컴퓨터에 제출된 모든 작업을 관리하는 일을 한다. 스케줄러 데몬에는 두개의 모듈이 있는데 큐 관리자와 스케줄러이다. 스케줄러는 콘도 풀의 중앙 관리자(central manager)와 통신하여 작업을 수행시키기 위한 여러 가지 절차를 수행하고, 작업 실행 시 작업의 그림자 프로세스(shadow)를 포크(fork) 하는 등, 작업 제출과 관련된 프로세스이다. 큐 관리자는 컴퓨터에 제출된 작업을 작업 큐에 넣거나 제거하는 등 큐 상에서 필요한 작업을 수행한다. 일괄처리(batch) 작업을 시작시키는 작업 시작 데몬(start daemon)인 스타트디는 중앙 관리자에 의해 해당 컴퓨터에서 수행시키도록 할당받은 작업을 시작하고, 모니터하고 종료하는 등의 일을 담당한다. 또한 스타트디는 키보드나 마우스 동작상황, CPU의 작업부하와 같이 콘도 작업을 수행하는데 필요한 정보들을 모니터한다.

중앙관리자는 풀의 모든 자원과 작업에 관한 정보를 유지 관리한다. 중앙관리자로 지정된 워크스테이션은 이러한 목적으로 데몬 프로세스를 수행한다. 콘도 풀 상의 모든 스케디와 스타트디는 자신의 정보를 중앙관리자안의 콜렉터(collector) 데몬에게 보고한다. 콜렉터는 콘도 풀 전체의 정보를 볼 수 있으므로 콘도 풀 상태에 관한 질의를 처리한다. 중앙관리자의 또 다른 데몬인 협상 프로세스(negotiator)는 주기적으로, 콘도 풀에서의 수행을 기다리고 있는 작업과 작업을 수행하기에 적정한 컴퓨터를 찾아 준다. 이러한 작업을 협상 사이클(negotiation cycle)이라고 하며 5분마다 수행된다.

콘도는 다음과 같이 동작한다. 작업이 컴퓨터에 제출되면 작업 큐에 넣어진다. 스케디는 콘도에 제출된 작업을 큐에 넣고 작업이 수행될 자원을 찾는다. 모든 작업은 작업을 수행하기 위해 필요한 작업 요구 사항(job context)을 명시한다. 한편 스타트디는 중앙관리자에게 주기적으로 해당 컴퓨터의 자원 제공 제의(machine context)를 광고한다. 작업 요구 상황은 차례대로 중앙관리자에게 전달되고 중앙관리자는 이러한 요구 조건이 만족되는 컴퓨터를 찾는다. 중앙관리자는 작업 요구 사항과 컴퓨터의 자원 제공 제의를 맞춰 선택된 스케디와 스타트디에 이 사실을 통보한다. 스케디에 제출된 작업은 이제 작업수행 컴퓨터의 스타트디에 의해서 수행하게 된다. 스타트디와 스케디는 동일한 콘도 풀에 속해 있으며 그림 3은 이러한 표준 콘도 프로토콜을 표시한 것이다. 콘도 시스템에 작업을 제출하거나 취소하고 작업 제출상태를 보는 등 콘도 사용자를 위해 제공되는 콘도 사용자 명령어는 표 1에 정리하였다. 프로토콜과 명령어에 관한 보다 자세한 사항은[7,9,10]에서 찾아볼 수 있다.

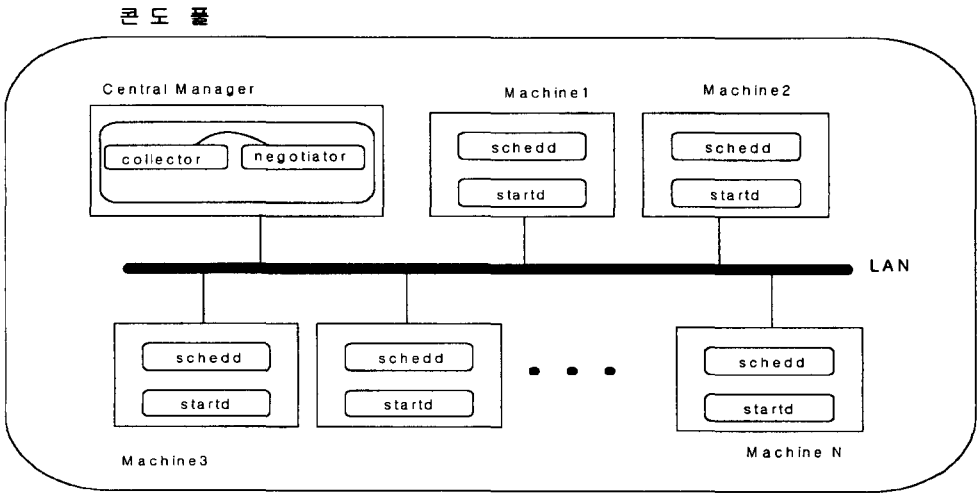
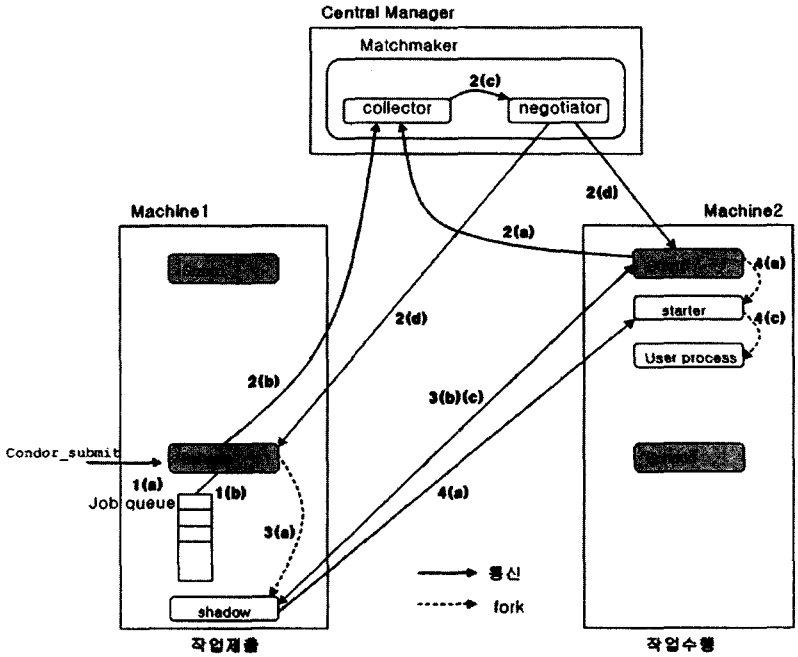


그림 2 콘도 풀의 구조



1. 콘도에 작업 제출
   (a) 사용자가 작업하는 컴퓨터에 작업 J와 수행에 필요한 요구사항 제출
   (b) 스케디(schedd) S가 작업큐에 J를 삽입
2. 매치메이킹(Matchmaking)
   (a) 스타트디(startd) E의 자원상태(machine context)를 중앙관리자에 제출(주기적으로 수행)
   (b) 스케디(schedd) S가 J의 요구사항(job context)을 중앙관리자에 제출 ((a)와 병렬 수행)
   (c) 중앙관리자가 J의 요구사항에 맞는 E의 자원상태 검색
   (d) 중앙관리자가 E와 S에 매치를 통보
3. 스케디와 스타트디의 연결
   (a) 스케디가 새도우를 포크(fork)
   (b) 새도우가 J의 요구사항 전달
   (c) if(E가 J의 요구사항 만족)
       새도우에 OK 메시지 전달
     else
       not-OK메시지 전달
       OK메시지 전달 받을 때까지 2,3 반복
4. 작업 시작
   (a) 스타트디가 스타터(starter)를 포크
   (b) 새도우가 J'의 수행화일을 스타터에게 전송
   (c) 스타터가 J의 수행화일을 시작한다.

그림 3 표준 콘도 프로토콜

3. 이동 분산 시스템 설계 개요와 확장시의 문제점

3.1 이동 컴퓨터의 특징

이동 컴퓨터는 분산 시스템상의 고정된 컴퓨터와는 다른 특징을 가지므로 이러한 특징들은 기존 분산 시스템을 이동 클라이언트를 위해 확장할 때 설계와 구현에 있어 여러 가지 문제점을 제기한다[11,12]. 다음은 시스템 확장 시 고려해야 할 이동 컴퓨터의 특징 몇 가지이다.

1. 이동 컴퓨터는 데스크톱 컴퓨터에 비해 전력소모에 있어 비교적 제약을 받는다. 이동 컴퓨터는 배터리와 같은 충전식 전원을 사용하여 주로 자체적으로

표 3 주요 콘도 명령어

주요 명령어	수행 내용
condor_compile	콘도 작업을 위해 재 링크된 수행화일 생성
condor_history	콘도 작업의 모든 로그 보여줌
condor_hold	큐에 있는 작업을 정지 상태로 보류
condor_off	로컬 컴퓨터에서 수행중인 콘도 데몬 정지
condor_on	로컬 컴퓨터의 정지중인 콘도 데몬 (재) 수행
condor_q	로컬 컴퓨터의 작업 큐 정보를 보여줌
condor_rm	로컬 컴퓨터의 작업 큐의 작업 제거
condor_submit	수행시키려는 작업을 콘도 풀에 제출
condor_advertise	콜렉터 데몬에 클래스애드를 전송

전원을 공급받아야 하기 때문이다.

2. 이동 컴퓨터와 고정된 네트워크를 연결하는 무선 링크의 대역폭은 유선 링크에 비해 상대적으로 낮고 신뢰성이 떨어진다.
3. 이동 컴퓨터와 네트워크의 연결 상황은 고정적이지 않고 계속 변한다. 이동 컴퓨터는 종종 긴 시간 동안 네트워크로부터 자발적으로 연결을 끊기도 한다.
4. 이동 컴퓨터는 화일 시스템을 공유하지 않을 수 있다. 즉, 이동 컴퓨터는 앤드류 파일 시스템(AFS)과 같은 분산 화일 시스템에서 작업하지 않을 수 있다.

이러한 특징들로 인해 이동 컴퓨터를 지원하기 위해 기존 분산 시스템을 확장할 때 이동 컴퓨터를 분산 작업 수행의 자원으로 보지 않고, 작업을 제출하고 수행 결과를 받아보는 클라이언트로 간주한다. 즉, 이동 컴퓨터는 이동 사용자에게 고정된 분산 배취 시스템에 대한 인터페이스만을 제공한다. 이동 컴퓨터가 가지는 제약점을 보완하려는 노력은 또한 본 연구에 대한 동기를 부여하였다. 이동 콘도 시스템을 구축함으로써 이동 컴퓨터의 사용자도 자원이나 전력 소모가 많은 작업을 수행할 수 있게 된다. 즉, 이동 컴퓨터를 콘도 분산 환경과 결합시켜 이동 컴퓨터의 작업 수행 능력을 확장할 수 있다.

3.2 설계 개요

그림 4는 이동 컴퓨터와 이동 컴퓨터를 지원하기 위해 확장된 콘도 풀의 구조이다. 이동 클라이언트가 콘도 분산 시스템 상에서 작업하기 위해서는 몇 가지 프로세스가 필요하다. 첫 번째는 이동 컴퓨터가 분산 시스템과의 네트워크 연결이 끊어진 상태에서도 인터페이스를 제공할 수 있는 프로세스가 필요하다. "이동 사용자 인터페이스" 또는 "큐 관리자"라고 불리는 이 프로세스는 네트워크와의 연결 여부와 관계없이 이동 클라이언트가 동일한 인터페이스로 언제나 분산 시스템에 접근할 수 있도록 해준다. 또한 이동 컴퓨터에서 발생한 콘도 작업을 작업 큐에 넣는 등 이동 컴퓨터에 있는 콘도 작업 큐를 관리한다.

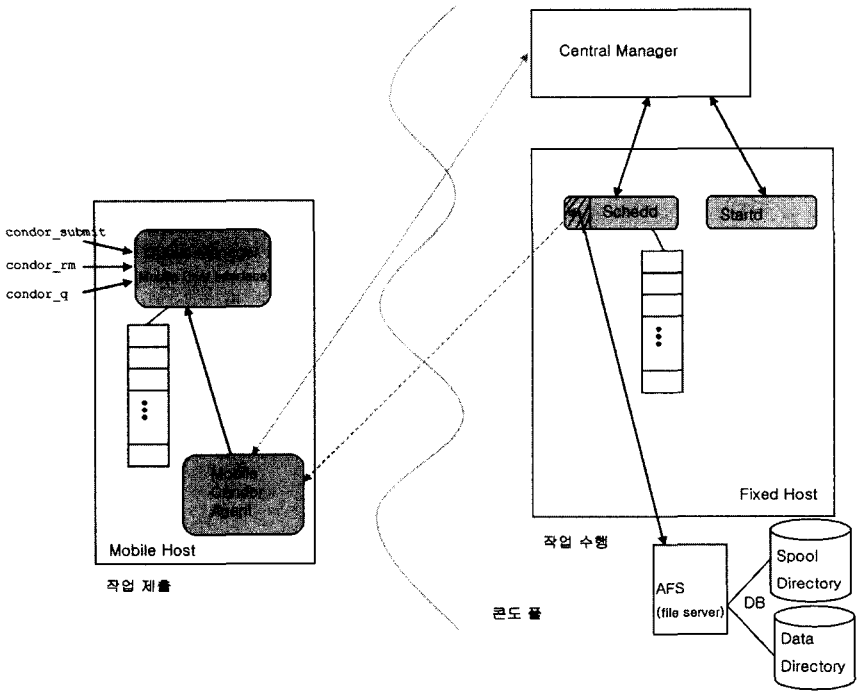


그림 4 이동 컴퓨터와 콘도 풀

두 번째는 MCA라고 하는 이동 콘도 에이전트(Mobile Condor Agent) 프로세스인데, 기존 분산 시스템과 이동 컴퓨터가 서로 대화하기 위해 필요한 통신을 담당하는 프로세스이다. MCA는 네트워크와의 접속 단절 이후부터 이동 사용자가 제출하여 이동 컴퓨터의 작업 큐에 저장되어 있는 작업 요청을 분산 시스템에 전송하고 이전에 보내진 작업 중 수행이 완료된 작업을 찾아 이동 컴퓨터로 가져오는 등 필요한 데이타 전송을 수행한다.

마지막으로 이동 에이전트와 대화할 분산 시스템 상의 고정된 컴퓨터에 "통신 에이전트"가 필요하다. 기존의 콘도 스케디는 이미 프로세스를 포크하여 실행 화일을 전송하는 기능을 가지고 있다. 따라서 이동 컴퓨터와의 통신을 위해 전용 프로세스를 고정된 컴퓨터에 새로이 설치하는 대신 기존의 스케디에 이동 컴퓨터와의 데이타 전송 기능을 추가(그림 4의 스케디에 빗줄 친 부분)하여 고정된 에이전트 역할을 담당하게 하였다.

3.3 설계 시 고려사항

다음은 콘도 분산 배취 시스템을 이동 사용자를 위해 확장할 때 설계 시 고려해야 할 사항이다.

- 작업제출자로 간주되는 이동 컴퓨터 - 이동 컴퓨터가 가지는 제약점과 빈번하게 연결이 해제되는 통신 링크로 인해 이동 컴퓨터는 오직 클라이언트로 간주된다. 이미 앞에서 언급했듯이 본 연구에서도 이동컴퓨터는 단지 작업 제출과 결과 검색을 한다고 가정하였다. 이동 컴퓨터는 단지 콘도 시스템에 작업을 제출하고 콘도 풀에서 작업이 완료된 후에 결과를 전달 받는다.
- 데이타 전송의 최소화 - 일반적으로 이동 컴퓨터와 고정된 컴퓨터간의 통신은 두 고정된 컴퓨터간의 통신보다 높은 비용이 든다. 또한 이들 컴퓨터간의 통신은 상대적으로 속도가 느리며 신뢰성이 낮다. 이동 컴퓨터는 주로 작업 제출과 결과 전송을 요청하므로 이동 컴퓨터와 고정된 분산 배취 시스템 간에는 많은 화일 전송이 필요하게 된다. 하나의 작업 요청에 대해 수행 화일과 입력 화일, 출력 화일 등이 전송되어야 한다. 이동 사용자를 지원하는 분산 배취 시스템은 이러한 화일 전송을 최소화할 수 있도록 설계되어야 한다. 콘도 시스템에서는 여러 작업이 동일한 수행 화일을 공유할 수 있다. 이동 콘도 에이전트는 수행 화일을 공유하는 이러한 작업들에 대해서 오직 한번만 수행 화일을 전송한다. 또한, 기존의 콘도처럼 사용자가 원할 때마다 현재까지 수행된 결과를 사용자에게 제공하는 것과는 반대로, 제출된 작업이 전부 수행을 완료한 후에만 최종적으로 출력 화일을 전송받고 중간 결과는 이동 사용자에게 제공하지 않는다.
- 밀기(Push) 대 끌기(Pull) - 제출된 작업이 고정된 시스템 상에서 수행이 끝나면 이동하는 사용자가 결과를 받아볼 수 있도록 수행된 결과가 이동 컴퓨터로 전달되어야 한다. 두 컴퓨터 간에 데이타를 전달하는 방법은 송신자가 데이타를 보내는 "밀기(Push)"와 수신자가 데이타를 가져오는 "끌기(Pull)"로 나눌 수 있다. "밀기" 모델은 결과를 보내는 쪽이, 보내고자 하는 데이타가 준비되면 결과를 받아보는 쪽과 연결하여 네트워크 상에 데이타를 보낸다. 연결이 불가능한 경우에 송신자는 적정한 시간 후에 다시 시도한다. 이 방법에 의한 데이타 전송은 실제 데이타 외에 다른 부가적인 메시지의 교환이 없고 네트워크의 연결 상태와 데이타 수신자의 상태가 안정된 경우 만족스럽게 동작한다. "끌기" 모델은 데이타 수신자가 전송 작업을 시작한다. 데이타는 송신자가 요청하기 전까지 수신자 쪽에 있다. 수신자가 데이타를 원할 때 송신자에게 데이타를 요청 한다. 수신자가 성공적으로 데이타를 받지 못하면 수신자는 다시 송신자에게 데이타를 요청한다. 수신자가 네트워크로부터 자주 연결을 해제하면 이 끌기 모델이 데이타 송수신에 더 적합하다.

4. 이동 콘도 인터페이스의 설계와 구현

4.1 이동 콘도 인터페이스의 설계

네트워크 연결이 이루어졌을 때 이동 컴퓨터는 먼저 콘도 풀의 중심 관리자와 연결하여 콘도 스케디의 포트 주소를 전달받는다. 그 다음, 이동 컴퓨터의 작업 큐에서 대기하고 있는 작업이 순서대로 콘도 풀의 고정된 컴퓨터로 제출된다. 제출된 작업은 고정된 컴퓨터상에 있는 스케디의 작업 큐에 다시 들어간다. 그런데, 현재 고정된 컴퓨터로 연결된 콘도 풀은 앤드류 화일 시스템(Andrew File System)상에서 작동을 하는 반면 이동 컴퓨터는 AFS상에서 작업하지 않고 독립된 화일 시스템을 가지고 작업한다. 즉, 이동 컴퓨터의 화일은 콘도 풀 내의 고정된 컴퓨터에서 인식할 수 없다. 더욱이 네트워크 연결이 끊어 졌을 때는 콘도 풀의 고정된 컴퓨터에서 접근할 수 없다. 따라서 이동 컴퓨터에서 제출된 작업이 수행되기 위해서는 그 작업에 필요한 모든 입력 화일과 수행 화일이 작업이 수행되는 콘도 풀로 이동되어야 한다. 필요한 화일이 모두 콘도 풀 상의 고정된 컴퓨터에 이동이 되면 다른 콘도 작업과 마찬가지로 수행될 수 있다. 일단 모든 화일이 이동하면, 이동 컴퓨터에서 고정된 컴퓨터의 스케디로 제출된 작업은 고정 컴퓨터에서 발생된 작업과 동일하게 취급되어 순서대로 콘도 풀에서 수행된다.

다음은 이동 콘도 시스템 설계 원칙이다.

- 이동 컴퓨터에서 제출된 콘도 작업은 콘도 풀 상의

고정된 컴퓨터로 이동되어 표준 콘도 프로토콜에 의해 수행된다.

- 이동 컴퓨터에서 제출된 콘도 작업은 이동 컴퓨터와 고정된 네트워크가 연결되었을 때 콘도 풀로 전송된다.
- 이동 컴퓨터에서 제출된 콘도 작업의 결과는 콘도 풀 상에서 작업이 완료된 뒤에 출력 화일로 저장된 결과는 이동 컴퓨터로 다시 전송되어야 한다.
- 스케디의 선택 - 이동 클라이언트는 이동 컴퓨터에 이동 콘도 에이전트가 설치될 때 콘도 작업을 전송할 스케디를 지정할 수 있다. 스케디가 지정되지 않은 경우에는 중앙관리자의 매치메이커를 통해 적절한 스케디를 지정받는다.

첫 번째 원칙은 이동 컴퓨터가 가지는 제약점에 기인한 것이고, 두 번째와 세 번째는 이동 컴퓨터와 콘도 풀 상에 빈번하게 끊어지는 통신 링크 때문에 생겨난 것이다.

4.2 큐 관리자

큐 관리자는 이동하는 사용자에게 콘도 인터페이스를 제공한다. 또한 기존 콘도 스케디가 가지는 큐 관리 기능도 수행한다. 큐 관리자는 표준 콘도 스케디에서 스케줄링 기능을 제외하여 만든 것이다. 큐 관리자는 기존 콘도의 표준 스케디가 작업 요청을 작업 큐에 저장하듯이 이동 컴퓨터에서 작업 제출 요청을 받아 이동 컴퓨터의 작업 큐에 저장한다. 스케디의 작업 큐의 각 엔트리는 클래스애드(classad)라 불리는 작업 설명 항목의 모음으로 구성된다. 클래스애드는 작업의 고유번호(ID), 작업 수행 화일 이름, 입력 화일 이름, 작업의 현재 수행 상황, 작업 경로 명, 작업 수행에 필요한 자원 요구사항 등을 포함한다. 이동 컴퓨터의 작업 큐도 이러한 정보들을 빠짐없이 작업 큐에 저장하고 있어야한다. 이러한 정보들은 나중에 작업이 수행될 때 고정된 컴퓨터의 작업 큐로 전송되어 작업을 수행하기 위한 매치메이킹(matchmaking)과 실제 작업 수행에 사용된다.

기존 클래스애드의 항목 외에 이동 컴퓨터의 작업 큐 엔트리는 세 개의 항목을 더 추가로 가지게 되는데, "RealClusterID"와 "RealProcessID", "ScheddID"이다. 이 들 항목을 추가하는 이유는, 이동 컴퓨터에 제출된 작업이 나중에 다시 콘도 풀에 전송될 때 부여받는 정보를 이동 컴퓨터에서도 저장하기 위해서이다. "RealClusterID"와 "RealProcessID"는 콘도 작업이 고정된 컴퓨터로 전송 되어 실제 콘도 풀 상의 고정된 컴퓨터의 작업 큐에 들어갔을 때 할당되는 클러스터 번호와 프로세스 번호이다. 이 번호는 이동 컴퓨터의 작업 큐에도 저장되어 이동 컴퓨터가 고정된 콘도 풀 상에서 제출 작업을 식별하는데 사용한다. "ScheddID"는 작업이 제출된 콘도 풀의 스케줄러 데몬 번호이다. 큐 관리자를 기존의 표준 콘도 스케디를 변형하여 동일한 사용자 인터페이스를 제공하도록 만들었으므로, condor_submit, condor_rm, condor_q 같은 표준 콘도 사용자 명령어는 이동 컴퓨터에서도 그대로 작동 할 수 있다.

4.3 이동 콘도 에이전트(Mobile Condor Agent)

이동 컴퓨터와 고정된 콘도 풀 상의 컴퓨터와의 연결은 계속적으로 빈번하게 변한다. 이동 콘도 에이전트(MCA)는 이동 컴퓨터와 고정 네트워크와의 연결을 주기적으로 확인 한다. 연결이 설정된 경우 이동 콘도 에이전트는 큐 관리자를 통해 이동 컴퓨터의 작업 큐에 콘도 작업이 새로 제출되어 전송을 대기하고 있는지 확인한다. 만일 대기 중인 작업이 있는 경우 콘도 풀의 고정된 컴퓨터에 작업 큐의 내용과 수행 화일과 입력 파일 등 필요한 모든 화일을 전송한다. 작업에 관한 전송이 끝난 후에는 해당 작업 큐의 항목에 전송되었음을 알리는 값을 저장한다. 고정된 컴퓨터로 더 이상 전송할 작업이 없는 경우 MCA는 고정된 컴퓨터의 스케디를 통해 고정된 컴퓨터의 작업 큐를 확인한다. 이전에 이동 컴퓨터에서 보낸 작업의 결과를 확인하여 만일 제출된 작업의 수행이 완료 상태이면 그 결과를 이동 컴퓨터로 가져온다.

4.4 콘도 작업의 제출

4.4.1 콘도 작업 ID(Condor Job Ids)

콘도작업은 클러스터 형태로 제출된다. 여기서 클러스터란 동일한 이진 수행 코드를 가지지만 서로 다른 입출력 화일을 갖는 작업들의 집합을 말한다. 모든 작업은 제출될 때 고유의 클러스터 번호와 프로세스 번호를 부여 받는다[그림 5]. 이 때 부여된 번호는 작업이 제출된 작업 큐 상에서만 고유하고 콘도 시스템 전체에서 고유한 번호는 아니다. 이동 컴퓨터 사용자가 콘도 작업을 제출할 때 작업 큐에 작업이 들어가면서 클러스터 번호와 프로세스 번호를 부여 받는다. 이때 부여된 번호들은 이동 컴퓨터 내에서만 고유하므로 이 작업이 다시 고정

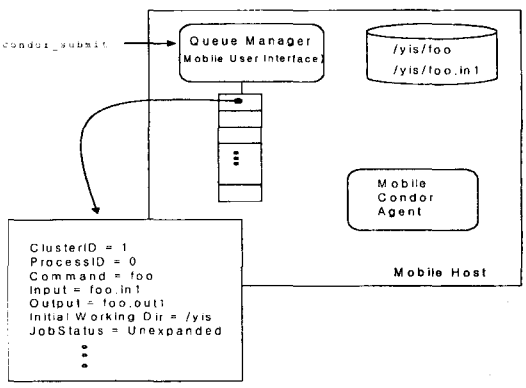


그림 5 이동클라이언트에서 콘도작업의 제출

된 콘도 풀의 컴퓨터로 전송될 때, 스케디에 의해 새로운 클러스터 번호와 프로세스 번호를 받게 된다. 즉, 이동 컴퓨터에서 제출되는 작업은 두 개의 고유 번호를 가지는데 이동 사용자가 작업 제출 시 이동 컴퓨터의 큐 관리자가 작업에 부여하는 클러스터 고유번호와 프로세스 고유번호, 다시 이 작업이 이동 컴퓨터에서 고정된 컴퓨터로 전송될 때 고정된 컴퓨터의 스케디에서 부여받는 실제 클러스터 고유번호 *"RealClusterID"*와 실제 프로세스 고유번호 *"RealProcessID"*가 그것이다.

4.4.2 이동 클라이언트의 작업 재출 프로토콜

이동 사용자가 콘도 작업을 제출할 때는, 표준 콘도의 condor_submit 명령을 그대로 사용한다. 표준 condor_submit 명령어의 이진 코드가 이동 컴퓨터에 그대로 설치되기 때문이다. 작업 제출 시에 큐 관리자는 작업 고유 번호, 즉 클러스터 번호와 프로세스 번호를 부여한다. 또한 필요한 모든 클래스애드 항목이 작업 큐에 기록된다. 이 때 저장된 모든 항목의 값은 나중에 작업이 고정된 컴퓨터로 전송될 때 함께 전송되어 고정된 작업 큐에 저장된다.

콘도 풀과 연결 상태 일 때, MCA는 이동 컴퓨터에 있는 작업 큐에서 작업을 순서대로 검사한다. 즉 클래스애드의 작업 상황 필드(JobStatus)를 검사하여, 그 값이 "Unexpanded"인 경우 작업은 고정된 컴퓨터로 아직 전송 되지 않았음을 나타내므로 로컬 클러스터와 프로세스 고유번호를 제외한 클래스애드의 모든 필드를 고정된 컴퓨터로 전송한다. 고정된 컴퓨터의 스케디는 자신의 작업 큐에 작업을 넣으면서 실제 클러스터 고유 번호와 실제 프로세스 고유번호를 부여하는데, MCA는 이 값들을 얻어 이동 컴퓨터 내 해당 작업의 클래스애드 항목 *"RealClusterID"*와 *"RealProcessID"*로 저장한다. 이 고유번호는 이동 컴퓨터에서 결과를 조회할 때 사용한다. 그런데, 콘도 풀의 컴퓨터에서 작업을 수행시키려면 작업과 관한 클래스애드 항목과 더불어 실제 수행 화일과 입력 화일에 접근할 수 있어야 한다. 그런데, 이동 컴퓨터와의 네트워크 연결은 불안전하므로 콘도 작업의 클래스애드항목과 더불어 수행 화일과 입력 화일도 고정된 컴퓨터로 함께 전송되어야 한다. 이들 화일은 콘도 작업을 위한 두 개의 전용 디렉토리에 저장된다. 스풀(Spool) 디렉토리와 데이타(Data) 디렉토리가 콘도 작업을 위한 전용 디렉토리인데, 모든 수행 화일은 스풀 디렉토리에, 모든 입력 화일, 출력 화일, 에러 화일은 데이타 디렉토리를 사용한다.

이동 컴퓨터에서 전송되는 모든 데이타 화일은 데이타 디렉토리라고 불리는 하나의 디렉토리에 저장되므로

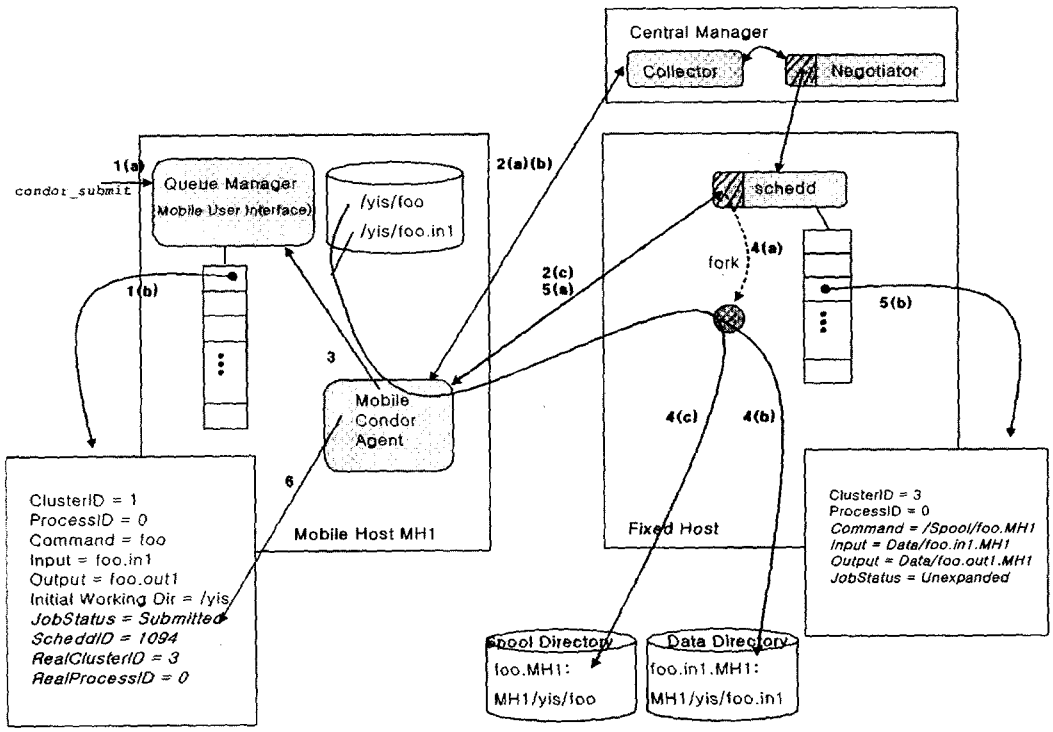


그림 6 이동 클라이언트의 작업 제출 과정

1. 이동컴퓨터의 사용자가 콘도작업 제출(콘도 풀과의 연결여부에 관계없이 수행)
(a) 이동 컴퓨터 사용자가 "condor_submit"을 실행하여 작업을 제출한다.
(b) 제출된 작업이 이동 컴퓨터의 작업 큐에 삽입된다.
(콘도 풀과 연결이 되는 경우 다음 단계부터 실행)
2. 이동컴퓨터와 스케디의 연결 설정
(a) 이동콘도에이전트(MCA)는 콜렉터에게 콘도작업을 제출할 콘도 풀 상의 스케디의 이름을 전송한다. 만일 MCA가 스케디 이름을 지정하지 않으면 콜렉터는 매치메이커를 통해 스케디를 선택한다.
(b) 콜렉터는 스케디의 컴퓨터 주소와 포트번호를 MCA에 반환한다.
(c) MCA는 스케디와 연결을 설정한다. 만일 연결 설정 실패 시 2(a)부터 다시 시작하여 다른 스케디의 주소를 얻는다.
3. MCA는 큐 관리자를 통해 전송할 작업, 즉 Job Status = "Unexpanded" 인 작업을 선택한다.
4. 이동컴퓨터에서 콘도 풀로 화일 전송:
(a) 콘도 풀의 스케디는 이동 컴퓨터에서 화일을 받기 위해 프로세스 포크(fork)
(b) 이동컴퓨터는 입력 화일을 전송한다.
(c) 스풀 디렉토리에 출력 화일이 없으면 출력 화일을 전송한다.
5. 이동 컴퓨터의 작업 큐 엔트리 전송:
(a) 제출 작업의 큐 엔트리(classad)를 전송한다. 이때, 큐 엔트리 항목 중 클러스터 아이디, 프로세스 아이디, 입력, 출력, 오류, 수행 화일은 콘도 풀에 제출될 때 새 값이 지정되므로 전송하지 않는다.
(b) 전송된 작업의 큐 엔트리(클래스애드)가 콘도 풀의 고정 컴퓨터의 작업 큐에 삽입된다. 이 때 새로운 클러스터 아이디와 프로세스 아이디가 부여된다.
6. 이동 컴퓨터의 작업 큐 엔트리 수정:
(a) 작업이 제출된 "*ScheddID*"와 새로 부여된 클러스터 아이디와 프로세스 아이디가 "*RealClusterID*"와 " *RealProcessID*" 항목으로 저장된다.
(b) 클래스애드 항목 중 전송된 작업의 상태(Job Status)를 "submitted"로 수정
7. 더 이상 전송할 작업이 없을 때까지 3번부터 다시 실행 한다.

그림 7 이동 클라이언트의 작업 제출 프로토콜

화일 이름 간에 충돌이 존재할 수 있다. 즉, 서로 다른 이동 컴퓨터에서 같은 이름을 갖는 화일을 사용하는 경우이다. 동일 이름 사용에 따른 충돌을 방지하기 위해서, 이동 컴퓨터에서 화일을 전송할 때는 화일의 원래 위치를 나타내는 전체의 디렉토리 경로와 기계이름을 함께 첨부하여 저장하고 식별한다[그림 6]. 또한 이동 컴퓨터에서 고정된 컴퓨터로 전송된 콘도 작업의 클래스애드는 "수행 화일 이름"항목과 "입력 화일"항목에 이동 컴퓨터내의 원래 화일 경로를 포함한 이름 대신에 스풀 디렉토리와 데이타 디렉토리에 전송되어 있는 화일들을 가리키게 된다. 콘도 작업의 클래스애드와 더불어 수행 화일과 입력 화일이 성공적으로 전송되면 이동 컴퓨터의 클래스애드 항목 중 작업 상태 필드는 "Unexpanded"에서 "Submitted" 상태로 표시된다.

그림 6과 그림 7은 이동 클라이언트를 위한 작업 제출 프로토콜을 나타낸 것이다. 그림 6의 번호는 그림 7의 프로토콜 각 단계를 표시한다. 그림에서 작업 제출 프로토콜 단계를 강조하기 위해 화일 서버 부분은 생략하였다. 만일 화일을 전송하는 도중 불안정한 네트워크 연결이 끊어지게 되면 과정 5와 6은 수행되지 못한다. 이 경우 작업의 상태가"Unexpanded"로 남아있게 되므로 나중에 콘도 풀과 연결이 될 때 MCA가 작업 큐를 검사하는 과정에서 미 제출 작업으로 인식하므로 다시 화일들을 전송하게 된다.

4.5 이동 컴퓨터와 콘도 풀 간의 화일 전송 최소화

수행 화일의 중복된 전송을 피하기 위하여 동일한 수행 화일을 가지는 콘도 작업들은 클러스터형태로 묶여서 제출된다. MCA는 작업마다 필요한 수행 화일은 전송하는 대신 각 클러스터 당 하나의 수행 화일만 전송한다. 이렇게 함으로써 수행 화일을 공유하는 작업이 있을 때 동일한 화일을 여러 번 전송하는 것을 피할 수 있다. 입력 화일의 경우에는 이동 컴퓨터에서 화일을 전송하기 전에 고정된 콘도 풀 상의 데이타 디렉토리에 동일 이름의 화일이 있는지 검사하고 동일 이름

의 화일이 존재하는 경우 화일의 크기와 생성 시각을 검사한다.

4.6 고정된 컴퓨터로부터 결과를 가져 오기

결과를 전송하는 고정된 컴퓨터는 결과를 받는 이동 컴퓨터가 언제 네트워크에 연결될지 예측하지 못한다. 더욱이 고정된 컴퓨터의 스케디는 이동 컴퓨터에서 제출된 작업이 어느 것인지에 관한 정보를 유지하지 않는다. 스케디 큐의 모든 작업은 어디서 제출되었는지 관계없이 동일하게 처리되기 때문이다. 콘도 풀의 데이타 디렉토리에는 이동 컴퓨터에서 전송된 작업의 수행이 끝난 뒤에 출력 화일과 에러 화일 등의 결과물이 저장된다. 이 결과물은 이동 컴퓨터와 연결이 되었을 때 이동 컴퓨터가 요청함으로써 이동 컴퓨터로 전송된다. 이 "끌기" 모델에서는 이동 컴퓨터가 결과물의 전송을 요청하기 전에 어떻게 제출된 작업의 종료를 확인하여야 하는지가 주요 문제이다.

4.6.1 이동 컴퓨터에서 작업의 종료를 검사하는 방법

MCA는 이동 컴퓨터에서 전송된 작업의 결과를 요청하기 전에 먼저 제출된 작업의 상태를 확인하여야 한다. 전송된 작업의 수행 생태를 확인할 수 있는 한 가지 방법은 새도우의 사용자 로그를 확인하는 것이다. 새도우는 작업이 수행될 때 포크되어 사용자 로그 화일에 작업 종료 이벤트를 기록한다[10]. 모든 로그는 새도우가 수행되는 고정된 컴퓨터상의 디스크에 기록된다.

또 다른 방법은 MCA가 단순히 고정된 컴퓨터상의 작업 큐를 검사하여 해당 작업의 클래스애드 항목이 큐 엔트리로 존재하는지를 검사하는 방법이다. 현재 콘도 버전 6은 작업이 종료되면 해당 작업 큐 엔트리가 큐에서 제거된다. 작업 큐에서 해당 작업의 엔트리를 발견하지 못하면 MCA는 작업이 종료된 것으로 간주할 수 있다.

이동컴퓨터에서 고정된 컴퓨터의 디스크를 접근할 필요가 있는 첫 번째 방법보다는 두 번째 방법이 더 간단하고 빠르므로 여기서는 두 번째 방법을 사용하여 작업의 종료를 검사한다.

4.6.2 결과를 가져오는 프로토콜

이동 컴퓨터가 콘도 풀에 연결되면 이동 컴퓨터내의 MCA가 고정된 컴퓨터의 스케디와 접촉하여 재출된 작업이 종료되었는지를 문의한다. MCA는 이 때 작업을 재출할 때 스케디로부터 받았던 "RealClusterID"와 "RealProcessID"를 사용한다. 만일 MCA가 고정 컴퓨

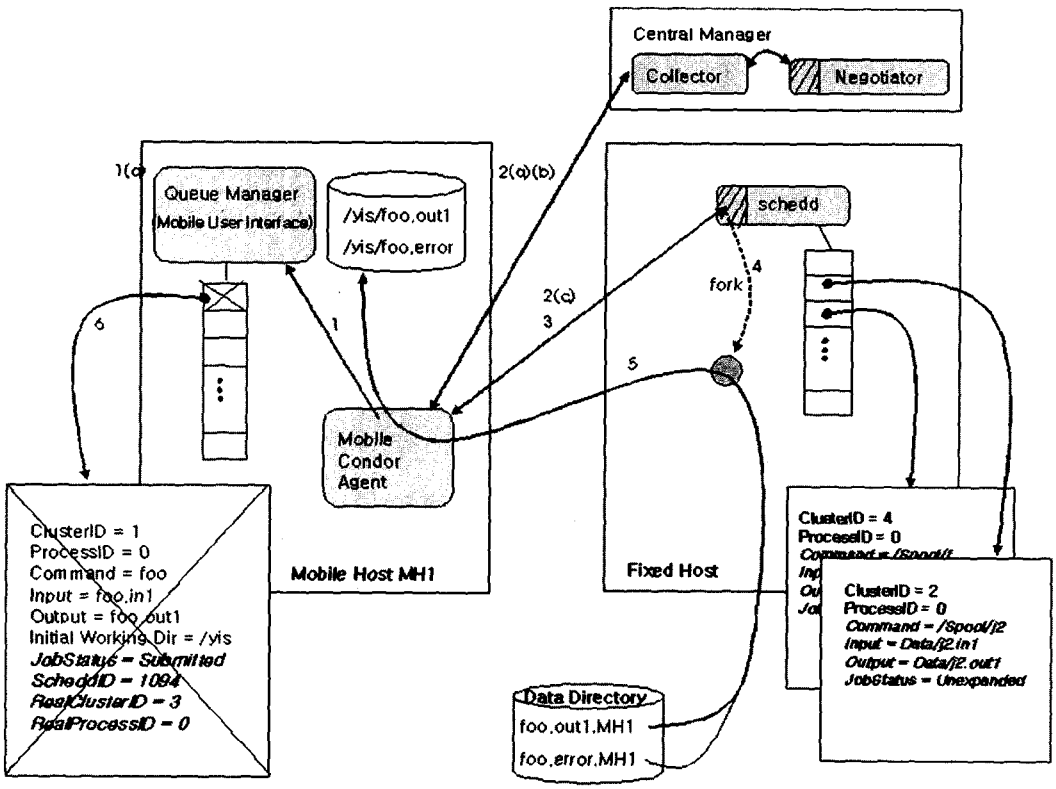


그림 8 출력 화일을 이동 컴퓨터로 가져오는 과정

1. 이동컴퓨터와 네트워크 연결이 되었을 때, MCA는 작업 큐를 탐색하여 작업의 상태(Job Status)가 "submitted"인 작업 J 가 있는지 검사한다. 만일 그런 작업이 있으면 다음 단계를 수행한다.
2. 이동 컴퓨터와 스케디의 연결:
(a) MCA는 작업 큐 엔트리에 저장된 스케디 아이디를 콜렉터에 전송한다.
(b) 콜렉터는 해당 스케디의 컴퓨터 주소와 포트번호를 반환한다.
(c) MCA는 반환된 주소와 포트 번호에 따라 스케디와 연결한다.
3. J의 완료 여부를 검사:
(a) 만일 J의 *"RealClusterID"*와 *"RealProcessID"*가 스케디의 작업 큐에 있다면 아직 J가 완료되지 않은 것이다. 이 때는 다른 제출 작업에 관한 검사를 하기 위해 다음 작업에 대해 1단계부터 다시 수행한다.
(b) 만일 J의 *"RealClusterID"*와 *"RealProcessID"*가 스케디의 작업 큐에 없다면 J가 이미 완료된 것을 의미한다. 다음 단계를 수행한다.
4. 스케디가 출력 화일들을 이동 컴퓨터로 전송하기 위해 프로세스를 포크한다.
5. 데이터 디렉토리에 있는 출력화일과 오류 화일을 이동 컴퓨터로 전송한다.
6. 이동 컴퓨터의 작업 큐 엔트리에 있는 J가 삭제된다.

그림 9 이동 클라이언트의 출력 화일 송신 프로토콜

터의 작업 큐에서 이들 아이디를 찾지 못하면, 제출된 작업이 종료된 것으로 간주한다. MCA는 데이타 디렉토리에서 출력 화일과 에러 화일을 끌어(pull) 온다. 성공적으로 화일 송신이 끝나면 이동 컴퓨터의 작업 큐에서 해당 큐 엔트리가 제거된다. 그림 8과 그림 9의 프로토콜이 이동 클라이언트가 콘도 풀에 접속되었을 때 결과물들이 어떻게 이동 컴퓨터로 전송되는 지의 과정을 보여준다.

4.7 이동 클라이언트를 위한 condor_q와 condor_rm 명령어

이 작업의 큰 목표 중 하나는 현재 콘도 시스템의 변화를 최소화 시키는데 있다. 이동 컴퓨터의 큐 관리자가 표준 콘도 스케디를 변형하여 작성되었으므로 기존의 사용자 콘도 인터페이스를 가지도록 지원하기 때문에 condor_submit, condor_q, condor_rm과 같은 명령어는 그대로 이동 컴퓨터에 설치될 수 있다. 그런데, 작업 큐의 현재 내용을 표시해주는 condor_q의 경우 현재의 작업 상황을 표시하기보다는 가장 최근에 콘도 풀과 연결이 되었을 당시의 상황을 표시하게 된다. 왜냐하면 이동 컴퓨터와 고정된 콘도 풀이 연결되었을 때 이동 컴퓨터 상의 작업 상태 정보가 갱신되고 그 후에 계속 연결이 해제된 상태라면 이동 컴퓨터 사용자에게 제공되는 작업 큐의 정보가 실제 콘도 풀 상의 작업 상황을 반영하지 못하기 때문이다. 또한 이동 컴퓨터와 고정 콘도 풀 간의 통신을 줄이기 위해, 기존 콘도 condor_q에서 보여주던 CPU 사용 시간과 같이 계속적으로 변하는 중간 작업 정보는 이동 컴퓨터로 전송하지 않는다. 작업 중 생기는 화일들도 작업이 완료되기 전에는 가져오지 않고 작업이 완료되어 결과물이 완성된 후에만 이동 컴퓨터로 가져온다.

제출된 작업을 취소하는 condor_rm 명령어는 작업 큐에서 해당 고유번호의 작업을 제거한다. 이동 컴퓨터의 큐 관리자는 condor_rm 명령어를 받고나서 작업 큐의 작업 엔트리를 제거한 후에 그 작업의 "RealClusterID"와 "RealProcessID"를 저장한다. 후에 콘도 풀과의 연결이 설정되면 고정된 컴퓨터의 작업 큐를 검사한다. 만일 작업 큐에서 해당 고유번호의 작업 엔트리를 발견하면 그 엔트리의 삭제 명령을 스케디에 보낸다. 스케디는 고정된 컴퓨터의 작업 큐에서 그 엔트리를 삭제한다. 만일 이미 작업이 종료된 상태여서 작업 큐에 해당 작업의 엔트리가 존재하지 않는 경우는 이동 컴퓨터에 저장되었던 고유번호들을 삭제한다. 이동 컴퓨터의 작업 큐에서는 condor_rm 명령어를 받았을 때 네트워크와의 연결 여부와 관계없이 즉시 이들 번호의 작업 엔트리가 삭제되었으므로 출력 화일과 에러 화일 등의 결과물은 이동 컴퓨터로 전송되지 않는다.

4.8 수행 예제

이 절에서는 간단한 예제를 통해 이동 컴퓨터에서의 콘도 작업의 제출과 작업 큐의 상태를 보이도록 한다. 먼저, 그림 10은 콘도 풀과 단절된 이동 컴퓨터에서 condor_submit 명령어를 통해 작업을 제출한 후 condor_q 명령어를 수행 시킨 결과이다. "my_job"은 c로 쓰여진 간단한 테스트 프로그램으로 gcc로 컴파일하

그림 10 콘도 풀과 단절된 이동 컴퓨터에서 작업 제출 후

그림 11 콘도와 연결된 이동 컴퓨터에서 작업이 제출된 후

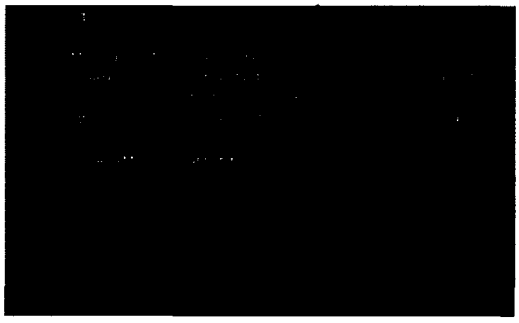


그림 12 첫 번째 제출한 작업이 완료된 후 큐의 상태

그림 13 condor_history의 이동컴퓨터 제출 작업 정보

고 condor_compile 명령어를 통해 "my_job.submit"을 생성하였다. 수행 파일 my_job은 현재 이동 컴퓨터의 작업 큐에 하나의 클러스터로 제출된 상태이므로 "Unexpanded"로 표시된다.

그림 11은 이동컴퓨터와 콘도 풀이 연결 된 후, 다시 두개의 작업(하나의 클러스터 내 두 개의 프로세스)을 더 제출한 후 큐의 상태를 보기위해 condor_q 명령어를 실행 시킨 것이다. 이동 컴퓨터의 작업은 모두 콘도 풀로 전송되어 작업을 전송받은 고정 컴퓨터가 표시되며 작업 상태는 "Submitted"로 바뀌었다. 이 때 작업 큐에 저장된 제출 시각은 이동 컴퓨터에 사용자가 작업을 제출한 시간을 의미하며, 이동 컴퓨터에서 고정된 콘도 풀로 작업을 전송한 시각을 의미하는 것이 아니다.

그림 12는 첫 번째 작업의 수행이 끝난 후에 작업 큐의 상태이다. 첫 번째 작업이 수행된 후 작업 큐에서 제거되므로 이동 사용자는 이것을 통해 작업이 완료된 것을 알게 된다. 이 때, condor_q 명령어가 보여주는 작업 큐의 내용은 콘도 풀 상에서의 현재 수행 상황을 반영하지 못한다. 통신비용을 줄이기 위해 수행 중인 작업의 중간 실행 상황을 전송하지 않기 때문이다.

마지막 그림 13은 콘도 풀의 고정된 컴퓨터에서 condor_history 명령어를 수행 시켜 이동 컴퓨터에서 제출된 작업에 관한 부분만을 보인 것이다. 전체 로그에서 이동 컴퓨터가 제출한 콘도 작업은, 이동 컴퓨터가 작업을 전송한 고정된 컴퓨터의 작업으로 표시된다.

5. 이동 클라이언트를 위한 매치메이커의 개선

이동 클라이언트는 콘도 풀과의 연결 설정과 해제가 빈번하다. 따라서 이동 클라이언트가 연결 해제 이후에 처음으로 콘도 풀에 접속할 때, 콘도 풀의 중앙 관리자와 통신을 통하여 작업이 제출될 스케디가 있는 컴퓨터의 주소와 포트 번호와 같은 필요한 정보를 얻어야 한다. 콘도의 중앙 관리자[그림 14]는 콘도 풀 상의 모든 스케디와 스타트디의 정보를 콜렉터라는 프로세스를 통해 가지고 있다. 또한 스케디와 스타트디의 자원 연결과정을 통해 스케디와 스타트디 사이의 연결 작업 로그도 유지하고 있다. 중앙관리자는 이동 클라이언트로부터 스케디를 요구 받았을 때, 매치메이커가 유지하고 있는 로그를 분석하여 가장 자원 요청이 적은(Least Frequently Used) 스케디의 이름을 전달한다. 이를 위해 매치메이커의 협상 프로세스(negotiator)를 수정하여 가장 적은 횟수로 자원을 요청한 스케디를 찾아 이름을 반환하는 모듈을 첨가하였다.

현재 구현된 이동 콘도 시스템은 이동 컴퓨터의 MCA가 스케디의 이름을 지정하여 작업을 제출할 수도 있다. 만일 스케디 이름을 지정하지 않으면 협상 프로세스로부터 가장 자원 요청 횟수가 적은 스케디의 이름이 차례대로 반환된다. 자원요청을 많이 한 프로세스일수록 작업 큐의 평균 길이가 길다고 추측할 수 있기 때문이다.

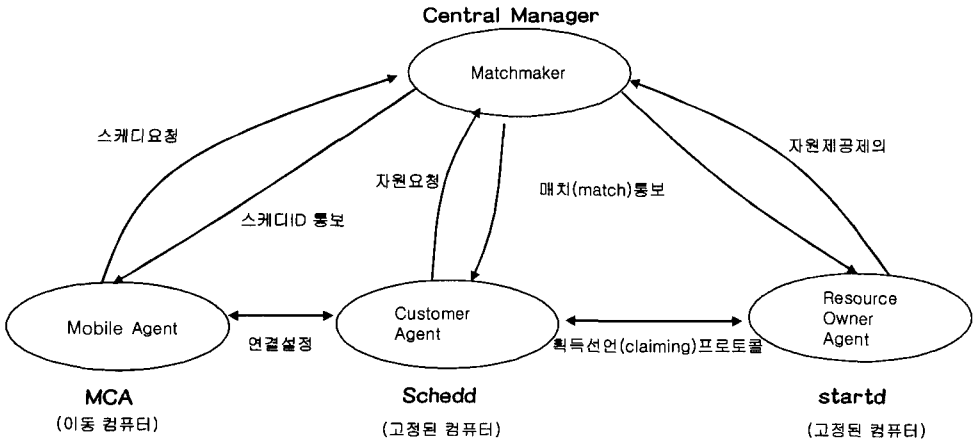


그림 14 이동 콘도 시스템의 매치메이커

이동 컴퓨터의 MCA가 스케디 이름을 지정하는 경우 동시에 여러 대의 클라이언트가 하나의 스케디에 작업 제출을 원하면 이동 컴퓨터와 스케디의 연결이 병목 현상을 초래할 수 있다. 4장의 작업 제출 프로토콜은 스케디와의 연결이 실패하는 경우 다시 중앙 관리자와 통신하여 다른 스케디의 이름을 얻도록 설계되었다. 이동 컴퓨터와 스케디의 연결과정은 기존 콘도 시스템의 스케디와 스타트디의 연결과정[13,14]과 유사하므로 중앙 관리자의 콜렉터 프로세스와 협상프로세스를 동일하게 이용한다. 단, 협상 프로세스의 스타트디 선택 기준은 스케디의 클래스애드가 요구하는 작업 수행 조건을 제공하는 스타트디를 찾는 것이지만, 이동 컴퓨터와 연결 될 스케디의 선택 기준은 LFU에 의하게 된다. 그림 6과 그림 8의 협상 프로세스(negotiator) 중 빗줄 친 부분이 이동 컴퓨터가 작업을 제출 할 스케디를 찾는, 즉 작업 제출 횟수가 가장 적은 스케디를 선택하기 위해 추가된 부분이다.

6. 결론 및 향후 연구 방향

이 논문은 이동 클라이언트를 위한 확장된 콘도 시스템의 설계와 구현을 제시하였다. 이 연구의 결과로 현재 C++로 작성된 콘도 버전6이 확장되어 이동 클라이언트에서 콘도 작업을 성공적으로 제출하고 결과를 받는 것이 가능해졌다. 즉, 네트워크 연결에 관계없이 어느 때나 이동 컴퓨터의 사용자도 콘도 분산 시스템 상에서 작업을 수행할 수 있게 되었다. 이 연구는 또한 이동 사용자를 지원하기 위해서 버전6의 기존 콘도 시스템의 수정을 최소화하는 것을 목적으로 하였다. 이동 클라이언트를 지원하기 위한 프로세스들은 콘도 분산 시스템의 기존 프로세스로부터 많은 코드 부분이 재사용되어 비교적 쉽게 구현되었다. 이동 컴퓨터의 큐 관리자는 스케디의 큐 관리자 모듈을 수정하여 구축되었다. 스케줄링 부분을 제외하고는 기존 콘도시스템의 스케디와 동일한 역할을 하므로 기존 큐 관리자의 코드를 간단히 수정하는 것으로 이동 컴퓨터의 큐 관리자를 구현할 수 있었다. 작업 제출 스케디의 선택 부분도 기존 매치메이커의 스타트디의 선택과정을 수정하여 구축되었다.

이동 컴퓨터 사용자의 관점에서, 이동 사용자는 자원이나 전력 소모가 많은 작업도 얼마든지 콘도분산 환경에서 수행할 수 있다. 이동 컴퓨터가 가지는 자원과 전력의 제약점으로 인해 수행이 불가능했던 작업을 콘도 배취 시스템에 제출함으로써 어떤 작업도 수행할 수 있게 되었다.

현재까지 구현된 이동 콘도 시스템은 앞으로 계속해서 보완되어야 한다. 먼저, 이동 컴퓨터를 지원하는 새로운 분산 화일 시스템을 콘도 분산 시스템에 도입하는 문제를 고려해보아야 한다. 현재 콘도 분산시스템은 고정된 컴퓨터만을 대상으로 한 기존의 분산 화일 시스템인 앤드류 화일 시스템 상에서 계발되어 화일 시스템을 사용하는 모든 기계 사이의 네트워크 연결이 언제나 안정되어있음을 가정으로 한다. 분산 화일 시스템 상에 포함되지 않는 이동 컴퓨터상의 화일은 콘도 풀 상의 컴퓨터가 접근할 수 없기 때문에, MCA의 주요 임무가 이동 컴퓨터와 콘도 풀 상의 컴퓨터 간에 화일들을 이동하는 것이다. 그런데 앞에서 설명하였듯이 이동 컴퓨터와 고정된 컴퓨터간의 화일 전송은 비용, 신뢰성, 시간 면에서 상대적으로 불리하다. 이 문제에 가능한 해결 방법으로 이동 클라이언트를 지원하는 이동 화일 시스템[15,16]을 사용하는 것이다. 이동 클라이언트의 단절된 화일 연산을 처리할 수 있는 분산 화일 시스템을 사용하게 되면 MCA에 의한 대부분의 화일 전송과 수신은 불필요하게 된다. 이동 클라이언트는 고정된 컴퓨터와 동일한 화일 시스템

상에서 작업하게 되고 이때 화일의 복제는 더 이상 MCA의 관할이 아니다. 화일의 이동이나 화일간의 충돌, 갱신등도 MCA가 관여할 필요가 없다.

또한, 이동 컴퓨터의 자유로운 이동성 지원이 더욱 보강될 것이다. 현재의 연구는 이동 사용자가 언제나 동일한 콘도 풀에 접속하여, 작업을 제출하는 콘도 풀과 작업을 수행하는 콘도 풀이 같다는 가정 하에서 출발하였다. 서로 다른 여러 개의 콘도 풀 상에서 사용자가 이동하는 경우에 이동 컴퓨터는 작업 중이던 풀과는 다른 풀에 있음을 알아야 한다. 제출된 작업의 클래스애드 항목 중 *"RealClusterID"*, *"RealProcessID"*, *"ScheddID"* 등이 다른 콘도 풀에서는 더 이상 유효하지 않기 때문이다. 이에 대한 해결 방법으로 5장에서 제안한 이동 컴퓨터와 고정된 스케디룰 연결하는 매치메이커를 확장할 계획이다. 이동 사용자가 처음으로 콘도 풀에 접속할 때, 접속한 풀의 중앙관리자에서 현재 작업 중인 콘도 풀에 대한 정보를 쉽게 얻을 수 있다. 매치메이커는 모든 콘도 풀 마다 상주하므로 매치메이커 간의 통신으로 작업하던 풀과의 통신이 가능해진다.

이 연구는 기존 분산 배취 시스템에 대한 최소한의 수정으로 이동 클라이언트를 지원 가능하게 한 성공적인 사례이다. 논문에서 제시한 콘도 분산 시스템을 확장하기 위한 기본적인 개념과 아이디어는 콘도 시스템에만 국한된 것이 아니라 어떤 유형의 분산 배취 시스템의 이동 클라이언트를 위한 확장에도 적용될 수 있다.

참 고 문 헌

[1] T. Imielinski and B. R. Badrinath, Mobile Wireless Computing: Solutions and Challenges in Data Management, Technical Report DCS-TR-296/WINLAB-TR-49, Department of Computer Science, Rutgers University, 1995.

[2] B. R. Badrinath, A. Acharya, T. Imielinski, "Designing distributed algorithms for mobile computing networks," Computer Communications, Vol. 19, No. 4, pp.309~320, April 1996.

[3] S. Hild, P. Robinson, "Mobilizing applications," IEEE Personal Communications, Vol.4, No.5 pp. 26~34, October 1997.

[4] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor - A Distributed Job Scheduler", in Thomas Sterling, editor, Beowulf Cluster Computing with Linux, The MIT Press, 2002.

[5] T. Tannenbaum, M. Litzkow, "The Condor distributed processing system," Dr. Dobb's journal, Vol.20 No.2, pp.40~49, February 1995.

[6] A. Bricker, M.J. Litzkow and M. Livny, Condor Technical Summary, Version 6.0, Technical Report 1069, Computer Sciences Department, University of Wisconsin-Madison, 1992.

[7] The Condor project homepage, http://www.cs.wisc.edu/condor/

[8] D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers, J. Pruyne, "A *worldwide flock of Condors*: Load sharing among workstation clusters," Future Generation Computer Systems, Vol.12, pp. 53~65, 1996.

[9] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanism for high throughput computing," SPEEDUP Journal, Vol.11, No.1, June 1997.

[10] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny, "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System", University of Wisconsin-Madison Computer Sciences Technical Report #1346, April 1997.

[11] B.R. Badrinath, A. Acharya, T. Imielinski, "Impact of mobility on distributed computations," Operating Systems Review, Vol. 27, No. 2, April 1993.

[12] A. Schill, B. Bellman et al., "System support for mobile distributed applications," Proceedings of 2nd Intl. Workshop on Services in Distributed Network Environments, pp.124~131, June 1995.

[13] N. Coleman, "An Implementation of Matchmaking Analysis in Condor," Masters' Project report, University of Wisconsin, Madison, May 2001.

[14] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing," Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28~31, 1998.

[15] J.J. Kistler, M. Satyanarayanan, "Disconnected operation in the Coda file system," ACM Transactions on Computer Systems, Vol. 10, No. 1, pp.3~25, February 1992.

[16] L.B. Mummert, M.R. Ebling, M. Satyanarayanan, "Exploiting weak connectivity for mobile file access," Proceedings of the 15th ACM Symposium on Operating Systems Principles, December 1995.

이 송 이

1988년 이화여자대학교 전자계산학과, 이학사. 1990년 미시건 주립대학교(Michigan State University) 전산 과학과(Department of Computer Science) 공학 석사. 1997년 서울대학교 컴퓨터 공학과, 공학박사. 1997년~1998년 미국 University of Wisconsin-Madison, Post-Doc. 1999년~2000년 서울대학교 중앙교육연구전산소 특별연구원. 2000년~2003년 성신여자대학교 컴퓨터 정보학부 계약교원. 현재 서울대학교 BK21 정보기술사업단 연구교수, 계약교수