그리드 서비스 환경에서 효율적인 자원 관리 프레임워크

(Efficient Resource Management Framework on Grid Service)

송 은 하 † 정 영 식 ††

(Eun-Ha Song) (Young-Sik Jeong)

요 약 본 논문은 그리드 서비스 환경에서 효율적인 자원 관리를 위한 프레임워크를 개발한다. 자원 관리는 그리드 서비스의 핵심이며, 자원의 가변적 특성에 적응적으로 대처하기 위한 프레임워크인 GridRMF(Grid Resource Management Framework)를 모델링하고 개발한다. GridRMF는 그리드 자원의 참여 의도에 따라 계층적으로 관리한다. 계층적 자원 관리는 가상 조직 관리를 위한 VMS(Virtual organization Management System)와 메타데이타 관리를 위한 RMS(Resource Management System)로 관리 도메인을 구분한다. VMS는 최적 가상 조직 선택 전략에 의해 자원을 중개하며, LRM(Local Resource Manager) 자동 회복 전략에 의해 가상 조직의 결함에 대처한다. RMS는 자원 상태 모니터링 정보를 적응적 성능 기반 작업 할당 알고리즘에 적용하여 부하균등화와 결함에 대처한다.

키워드 : 그리드 서비스, 계층적 자원 관리, 그리드 자원 할당 및 관리, 그리드 자원 모니터링 정보 가시화, 작업할당 알고리즘

Abstract This paper develops a framework for efficient resource management within the grid service environment. Resource management is the core element of the grid service; therefore, GridRMF(Grid Resource Management Framework) is modeled and developed in order to respond to such variable characteristics of resources as accordingly as possible. GridRMF uses the participation level of grid resource as a basis of its hierarchical management. This hierarchical management divides managing domains into two parts: VMS(Virtual Organization Management System) for virtual organization management and RMS(Resource Management System) for metadata management. VMS mediates resources according to optimal virtual organization selection mechanism, and responds to malfunctions of the virtual organization by LRM(Local Resource Manager) automatic recovery mechanism. RMS, on the other hand, responds to load balance and fault by applying resource status monitoring information into adaptive performance-based task allocation algorithm.

Key words : Grid Service, Hierarchical Resource Management, Grid Resource Allocation & Management, Grid Resource Monitoring Information Visualization, Task Allocation Algorithm

· 이 논문 또는 저서는 2006년 정부(교육인적자원부)의 재원으로 한국학술진흥재단의 지원을 받아 수행된 연구임(KRF-2006-521-D00387)

† 정 회 원 : 원광대학교 전기전자 및 정보공학부
ehsong@wku.ac.kr
†† 종신회원 : 원광대학교 전기전자 및 정보공학부 교수
ysjeong@wku.ac.kr

논문접수 : 2007년 8월 21일
심사완료 : 2008년 1월 30일

정보과학회논문지: 시스템 및 이론 제35권 제5호(2008.6)

1. 서 론

그리드 서비스 환경은 가상 조직의 중심에 있으며 지역적으로 분산된 다양한 계산 자원이나 정보 자원을 가상 조직의 멤버가 하나의 가상 컴퓨터로써 이용한다. 이러한 환경에서 사용자가 단일 시스템처럼 그리드를 이용해 거대 문제를 해결하기 위해서는 그리드 자원의 접근을 가능하게 하는 정책이나 메커니즘 결정이 요구된다[1-3]. 그리드 서비스 환경은 기본적으로 수용 가능한 자원의 수가 무한대이고, 지역 및 조직적으로 분산되어 있으며, 이기종 시스템이고, 특히 가상 조직을 포함한 자원의 동적인 생성 및 소멸, 자원의 소유 상태가 항상

유동적이다[4]. 이러한 서비스 환경에서 시스템의 성능을 극대화하기 위해서는 사용자의 요구와 사용 가능한 자원을 파악해야 한다. 즉, 그리드를 구성하는 자원은 상이한 위치, 참여 의도 및 목적, 특성을 가지며 처리되는 어플리케이션은 상이한 연산량, 사용자 요구조건을 가지므로 자원 관리 모델과 작업 관리 모델이 필요하며 이를 검증하기 위한 프레임워크가 개발되어야 한다. 본 논문에서는 그리드 자원의 가변성과 어플리케이션의 다양성에 적응하면서 그리드 서비스를 제공하는 프레임워크인 GridRMF(Grid Resource Management Framework)를 개발한다.

GridRMF는 프레임워크에 참여한 그리드 사용자의 자원 공유 관점에서 3-계층 자원 관리 구조(3-hierarchical resource management structure)를 설계하며, 자원 운영 관점에서 VMS(Virtual organization Management System)와 RMS(Resource Management System)로 관리 도메인을 분류한다. 본 프레임워크의 기반 인프라스트럭쳐는 계층적 자원 관리 구조에 따른 하부 통신의 일관성 유지를 위한 통신 모델, 그리드 정보 저장의 중복성을 고려한 정보 저장 모델, 서버의 오버헤드를 줄이기 위한 원격 객체 참조 모델을 정의한다. VMS는 그리드 자원의 접근을 제어하며 어플리케이션의 요구조건을 만족하는 그리드 자원 중개 시스템이다. 즉, 본 논문은 어플리케이션의 특성을 반영한 최적 가상 조직 선택 전략과 그리드 서비스의 가용성을 고려한 LRM 자동 회복 전략을 제안한다. RMS는 성능에 결정적인 영향을 주는 시스템으로 기존의 PDP[4] 시스템의 작업 할당 알고리즘에 그리드의 자원의 적응성을 반영한 작업할당 알고리즘을 제안한다. 특히 본 프레임워크는 그리드 자원을 수집하고 통합하는데 있어 자원 상태 모니터링과 다양한 가시화 뷰로 자원 동작 상태 및 성능 분석 결과를 작업 스케줄링에 반영한다[5]. 또한 GridRMF는 하부 구조와의 연관성을 줄이며 최소한의 코드 수정으로 특정 어플리케이션과의 독립성을 지원하기 위해 정형화된 API인 어플리케이션 프록시를 설계한다. 구현한 그리드 프레임워크는 2개의 요구 조건과 연산량이 다른 어플리케이션을 적용하여 제안한 그리드 서비스의 타당성을 보인다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구와 관련된 연구를 소개하고, 3장에서는 그리드 서비스 환경에서 효율적인 자원 관리를 위한 프레임워크를 설계하고 제안한 그리드 서비스의 성능평가를 하며, 4장에서는 어플리케이션 적용 예시로 프레임워크의 수행 결과를 보여주고, 5장의 결론으로 끝을 맺는다.

2. 관련연구

본 장에서는 본 프레임워크를 모델링하는 데 적용된 자원 관리와 성능 향상을 위해 모니터링을 적용한 작업 관리로 구분하여 관련연구를 기술한다.

그리드는 다른 개인이나 조직들에 소유되는 다양한 자원들을 논리적으로 결합함으로써 자원 관리 모델 선택으로 효율적인 정보 관리가 실현될 수 있다. 그리드 자원 관리를 위한 구조 모델[6] 중 계층적(hierarchical) 모델(Globus[7], Legion[8], Ninf, NetSolve[9] 등)은 자원 제공자와 자원 요청자간의 상호 관계를 따르며, 추상적 소유자(abstract owner) 모델은 실제 자원들이 어떤 환경에서 사용될 수 있는 가에 대한 허가, 자원 사용을 위한 메커니즘을 포함한다. 본 논문은 계층적 모델의 구성 요소인 수동적인 것(passive)과 능동적인 것(active)을 역할과 운영 관점에서 재구성한다. 수동적인 구성 요소 중에 성능에 영향을 주는 자원(resources)은 가변성 유무에 따라 정적 자원 요소와 동적 자원 요소로 분류하여 스케줄링 정보로 사용된다. 능동적 구성 요소 중의 도메인 제어 에이전트는 직접 큐잉을 통해 정보 서비스를 퍼블리싱하여 상태 정보를 제공하는 Maui Scheduler, Globus GRAM, Legion Host Object와는 달리, 본 프레임워크는 GridRMF의 구성 요소의 역할에 따른 동적 가상 관찰자(virtual observer)를 두어 통합 상태 정보를 모니터한다. 또한 어플리케이션의 연산량과 요구조건을 충족하는 가상 조직 즉 실제 연산을 수행할 그룹의 선택은 기 수행된 로그파일 정보에 의해 LRM에게 자원 공유를 허가하는 메커니즘을 제공한다.

계층적 모델을 따르는 가장 대표적인 그리드 관련 연구인 Globus는 하위 계층에 있는 지역적으로 분산된 이기종 자원들을 그리드 작업에 이용될 수 있도록 연결 관리하여 상위 계층에서 필요한 서비스를 제공하는 역할을 한다. Globus는 분리될 수 없는 단일 시스템이 아니라 그리드에서 필요로 하는 다양한 서비스를 독립적인 요소로서 제안한다. Globus의 요소들이 하드웨어 측면이나 소프트웨어 측면에서 상이한 시스템들 간의 성능 저하를 줄이면서 통합 기능을 제공한다는 관점에서 본 논문에서 구축한 GridRMF와 유사하다. 그러나 Globus는 메타데이타 관리 측면에서 데이타의 특성을 무시한 획일적인 변경 및 전달 주기를 갖고 있으므로 비효율적이다. 또한 자원 동시 할당은 전체 할당에 대해서만 고려하며 만약 하나라도 문제가 발생할 경우에는 모든 할당을 중지하는 방식으로 효율성에 제한된다. 따라서 그리드 성능을 저해하는 요소를 유발시킬 수 있다.

본 논문에서 모델링한 GridRMF는 VMS와 RMS로 구분하여 자원의 연관성을 고려한 그리드 서비스를 제공한다. VMS는 노드 구조 정보를 계층적으로 그룹화하여 사용자 요청에 적합한 가상 조직을 제공한다. RMS

는 메타데이타 상태 정보를 가상 관찰자에 의해 주기적으로 받아들여 로컬 자원 스케줄러를 구동하는 작업할당 기법으로 수행 시간을 고려한다. 뿐만 아니라, GridRMF는 다양한 가시화 뷰에 따른 실시간 그리드 자원 정보를 관측하는 모니터링 기술을 추가하여 그리드 자원 정보의 변화에 대응하고 관리의 용이성을 지원한다. 표 1은 Globus와 제안하는 시스템이 지원하는 사항의 비교이다.

그리드 시스템의 성능 모니터링에 대한 툴, 메소드, 어플리케이션에 대한 연구가 진행되고 있으며, 각 모니터는 독립적이지 않고 고유의 그리드 시스템에서 실행될 수 있다[11]. 본 프레임워크의 모니터링의 적용은 모니터링 정보를 추출하여 원하는 형태로 생성, 조합이나 연관성 등을 필터링하여 원하는 형태로 처리, 모니터링 정보의 로그파일을 생성하여 시스템 관리자에게 유포; 수집되어 처리된 정보를 표현한다[12]. 이를 지원하고자 하는 모니터링 시스템들은 다음과 같다.

DRMonitor[13]는 모니터링된 자원 정보를 기록하지는 않지만 장애를 가진 컴퓨터를 검출하고 부하 균형화 정책을 도와주기 위한 성능 평가 결과 값을 일정시간마다 갱신하고 그래프를 통해 모니터링 정보를 제공한다. TOPSYS는 이질적인 기계에 상관없이 어플리케이션을 확장하기 위해 사용법을 단순화하는 통합 도구 환경을 제공한다. TOPSYS 모니터링은 프로그램의 동적인 행동을 전반적으로 이해하여 부적절한 동기성과 통신 지연에 의한 성능 병목현상과 예기치 못한 행동을 검출한다. TMP[14]는 실시간 컴퓨팅 시스템에서 호스트 성능에 최소한의 영향을 주기 위해 하드웨어 모니터와 혼합된 형태이다. 호스트 시스템이 모니터 되는 결과에 동적으로 반응하도록 시스템에 모니터 정보를 제공하여 좋은 성능 튜닝, 스케줄링, 에러 핸들링이 가능하다. 하지만 H/W형태의 저수준의 모니터를 제공하므로 호스트의 한정된 정보만을 얻을 수 있다. Supermon은 노드의 레벨에 따라 작은 서버 프로그램(mon, supermon)을 제공한다. 기존의 방법보다 빠르게 노드의 행동 모니터가 가능하나 특정한 시스템에 대한 모니터링으로 제한된다. NWS는 SNMP로 관리되는 네트워크에서 사용하기 위한 서비스이다. SNMP agent를 사용하기 때문에 광범위한 자원 요소 집합을 가지고 있지만 SNMP를 지원하지 않는 환경에서는 모니터링 할 수 없다. WatchTower[15]는 사용자가 알지 못하는 컴퓨터의 남아있는 자원의 사용률을 모니터하는 서비스이다. WatchTower는 몇몇의 자원들을 사용하고 자동적으로 시작과 종료를 구성하므로 사용자의 간섭이 필요 없고 윈도우 기반의 디스플레이를 하지 않는다. 본 논문에서는 각각의 모니터링 시스템이 제안하는 서비스를 그리드 서비스 환경으로 적용하였으며, 모니터링 요소를 성능 향상에 초점을 둔다. 표 2는 본 프레임워크와 모니터링 시스템의 비교이다.

3. 그리드 자원 관리 프레임워크

3.1 GridRMF의 구조

GridRMF는 그림 1과 같이 자원의 참여 의도나 목적에 따라 구성요소를 4가지로 분류하고, 자원 정보 관리에 따라 2개의 관리 시스템으로 계층화한다. 구성요소는 그리드 어플리케이션을 수행하기 위해 자원을 요구하는 자원 요청자(Resource Requester, 이하 RR), 자신의 유휴 자원을 제공하고자 하는 자원 제공자(Resource Provider, 이하 RP), 자원 제공자에게 실제 작업 수행을 지시하고 이들을 관리하는 지역 자원 관리자(Local Resource Manager, 이하 LRM) 그리고 전반적인 작업 요청 제어 및 연결 기능을 수행하는 전역 정보 관리자(Global Information Manager, 이하 GIM)가 있다. 자원 정보 관리 시스템은 물리적으로 나눠진 개인이 소유한 메타데이타 관리를 위한 RMS와 가상 조직을 포함하는 VMS로 계층화한다. 그림 1은 제안하는 프레임워크의 전체 구조 및 제어 흐름이다.

구성 요소별 핵심 모듈의 기능은 표 3과 같으며, 논리

표 1 Globus와 제안하는 시스템의 지원 사항 비교

지원 사항 \ 시스템	Globus	제안하는 시스템	비고
그리드 확장성 고려	●	●	3-계층 자원 관리 구조
동적 가상조직 구성	●	●	
서비스에 대한 가용성 고려	●	●	최적 가상 조직 선택 및 Auto-Recovery 기법
결함에 따른 중복 연산 고려	×	●	
자원 상태 모니터링	●	●	PDH Library[10], Virtual Observer
자원 상태 가시화	×	●	다양한 가시화 뷰
부하 균등화 정책	▲	●	Performance-based Task Allocation 기법
결함 허용 정책	▲	●	
어플리케이션과의 독립성	●	●	Application Proxy
사용자 접근 인터페이스	●	●	Java Applet

표 2 기존 모니터링 시스템과의 비교

시스템 / 항목	DRMonitor	TOPSYS	TMP	SuperMon	NWS	Watch Tower	제안하는 시스템
결함허용	×	●	●	●	×	●	●
정보 기록	×	×	×	×	×	●	●
작업 스케쥴링	×	×	●	×	×	●	●
정보 시각화	●	●	▲	×	●	×	●
자원 성능평가	●	●	×	×	×	●	●
기반 기술	-	-	C	RPC	SNMP	PDH	JNI&PDH
실행환경	네트워크상의 개인 컴퓨터	병렬시스템	실시간 시스템	클러스터	네트워크 컴퓨팅	클러스터	그리드 컴퓨팅

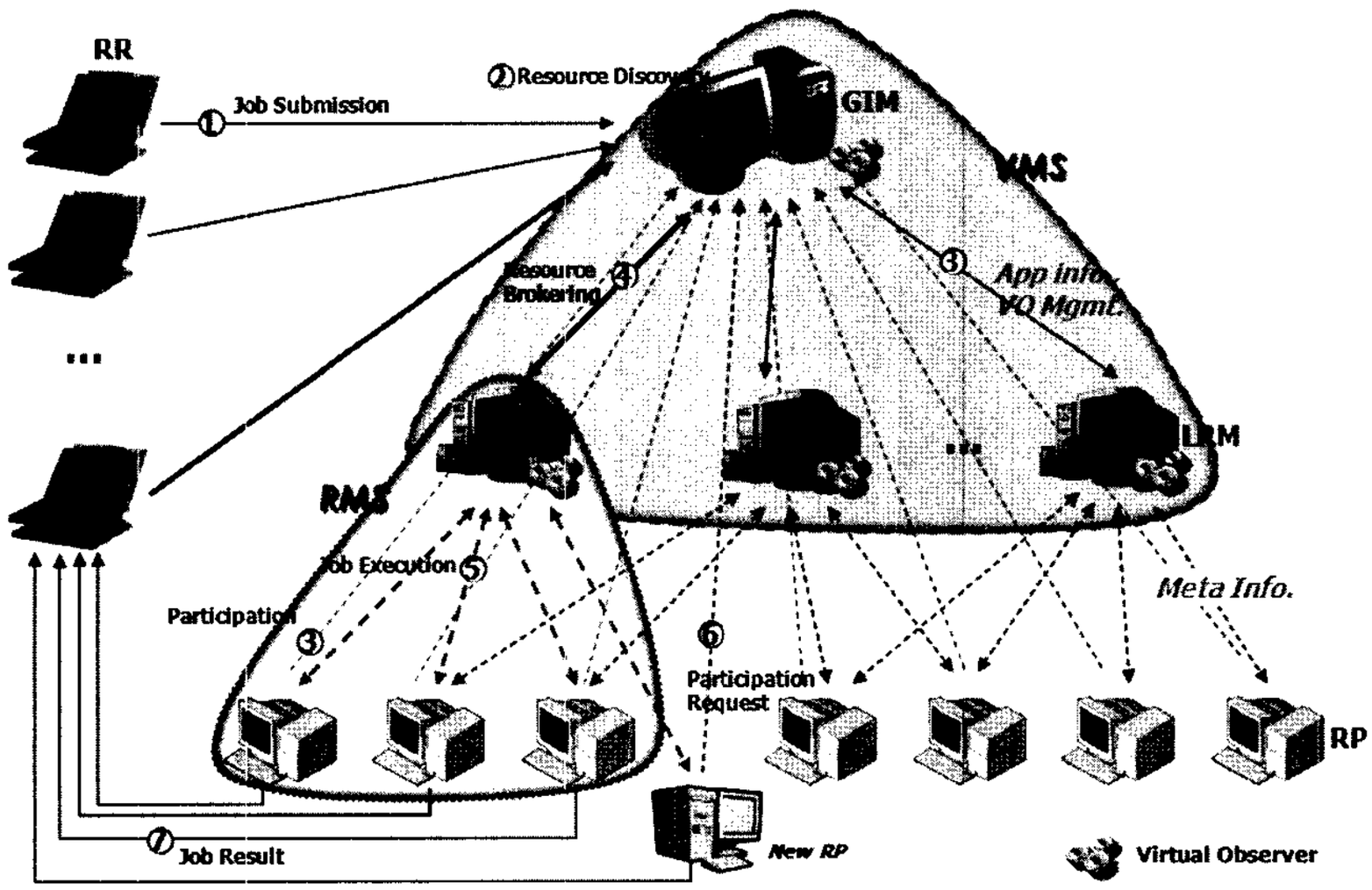


그림 1 GridRMF의 전체 구조 및 흐름

표 3 GridRMF 구성 요소의 핵심 모듈

구성요소	모듈	기능
RR	RR UI	GIM과의 상호 작용을 위해 애플릿을 다운로드, RMI를 통해 프레임워크에 등록
	App UI	어플리케이션의 작업 명세 및 수행 결과를 나타내는 인터페이스
	App Info Provider	어플리케이션 명세인 작업 단위 부여
	Task Info	전체 작업량과 작업 결과를 알 수 있는 부울 타입의 작업 테이블
	Task Results Gathering/Verify	RP들의 작업 결과를 통합하여 App UI 전달, 진행률과 수행시간 가시화
GIM	GIM UI	사용자 이벤트 및 화면 가시화 뷰
	Connection Brokering	구성요소들의 초기 접속 중개, 벡터 테이블 형태의 Entry를 생성하여 리스트 형태로 관리, LRM 목록을 RP에게 전달
	RR/LRM Join Brokering	RR과 수행 가능한 최적 LRM과 연결 중개
	LRM Virtual Observer	LRM의 상태 정보 및 갱신 정보의 전송을 대행
LRM	LRM UI	접속 및 바인딩에 관한 상태, 이벤트와 프로토콜의 관리 프레임, 모니터링 가시화 뷰
	Process Connection	RR과 RP의 연결
	Resource Factor	RP의 메타데이타 정보 기록하여 RP의 성능 평가
	Scheduler Manager	스케줄러 구동을 요구
	Scheduler	태스크 작업할당/재할당 제어
	RP Virtual Observer	RP의 갱신 정보를 관리와 LRM의 성능 및 상태 감지
RP	Resource Info DLL	RP의 메타데이타 추출, PDH 라이브러리
	Resource Broker	Resource Info DLL의 사용을 위한 JIN[16]에서 네이티브 메소드 선언
	Resource Register	자원 정보 추출 및 정적 자원 정보와 동적 자원 정보 분리
	Processor Manager	스케줄링 방법의 중개와 Processor(Dynamic/Static) 생성
	Process Proxy	태스크 수행을 위한 프로세서 중개

적으로 계층화된 자원 정보 관리 시스템은 세부 절에서 기술한다.

3.2 GridRMF의 인프라스트럭쳐

GridRMF는 본 프레임워크에 포함되는 그리드 자원의 원활한 데이타 흐름을 위해 공통의 하부 구조를 설계한다. 첫째로, 그리드로 연결된 시스템이나 자원들 즉, 구성요소간의 데이타, 명령 및 쿼리 등을 주고받는데 있어서 일관된 형식으로 자원들의 능동적인 결합에 대처할 수 있는 통신 모델이 요구된다. GridRMF의 통신 모델은 계층 구조를 따르는 자원들간의 연결을 세션으로 정의하는 연결 객체(connectivity object)와 명령이나 쿼리 등의 요구사항 생성 및 해석을 위한 프로토콜 객체(protocol object)를 둔다. 특히 프로토콜 객체는 명령이나 쿼리를 해당 구성요소가 인식할 수 있는 형태로 번역/해석하는 구성 요소별 Protocol Data Parser를 두며, 이를 통해 전달되는 메시지는 공통의 헤더 정의를 하여 모니터링 요소로 사용된다. 표 4는 프로토콜 객체의 메시지 헤더 정의의 일부이다.

둘째로, 본 프레임워크의 자원은 지역 자원과 전역 자원으로 구분할 수 있는데, 자원의 정보에 대해 최대한의 중복 저장과 사용상의 일관성이 필요하다. 이를 위해 자원 정보를 기능별, 용도별로 그림 2와 같이 엔트리(entry)를 생성하여 관리한다.

SuperEntry는 모든 엔트리를 제어하며 식별자와 이름을 부여한다. 구체화된 엔트리는 보유한 식별 정보를 통해 구성요소간의 연결 제어, 스케줄링, 모니터링에 적용되며 여러 개의 엔트리와 연계한다. EntryTable은 개

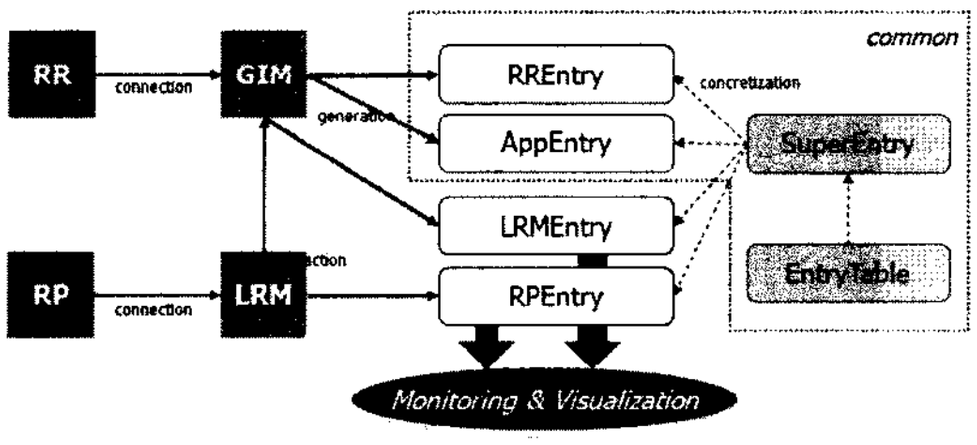


그림 2 GridRMF의 자원 정보 저장

넘이 같은 구성요소들을 해시테이블 형태로 목록화하여 저장한다. AppEntry는 각각의 어플리케이션에 대해 독립적인 정보를 저장한다. 어플리케이션은 RR, LRM과 RP에서 직접 적용되므로 해당 구성 요소가 필요한 정보를 기록한다. 즉, RR로부터는 어플리케이션 사용자 인터페이스 정의를 위한 UI 이름, RP은 공통 어플리케이션 프록시를 가져와서 처리하는 프록시 인터페이스 이름, LRM은 스케줄링을 위한 참조 데이타를 가져오기 위한 스케줄러 이름 등이 기록된다. RREntry는 RR의 정보를 저장하며 스케줄링에 사용된다. RREntry는 현재 해당 RP가 수행중인 어플리케이션 이름, 원격으로 결과를 얻기 위한 클래스 이름을 기록한다. LRMEntry는 GIM에 의해 관리되는 엔트리이며, 해당 LRM이 가지고 있는 RP들의 총 벤치마킹 정보(TotCpuSpeed, TotCpuNum, NumofRP, 등)를 기록한다. RPEntry는 LRM의 모니터링 요소로 사용되며, RP의 메타데이타 정보, 상태 정보, 작업량 등이 기록된다.

셋째로, 원격 객체 참조(Remote Object Reference)에 의해 RP가 연산 결과를 RR에게 직접 전송함으로써 프레임워크를 구성하는 특정 자원들의 오버헤드를 줄인다. 원격 객체 참조 메커니즘은 다음과 같다. LRM은 RREntry 리스트를 가진 RefBinder를 생성한 후 원격 인터페이스인 RRRefBroker로 변환하여 바인딩 시켜 RR과 RP를 검색한다. RR은 결과를 전송받는 Task-Receiver 클래스를 가지고 원격 객체 참조값을 Remote-TaskResult 인터페이스에 의해 LRM에게 전달되고 LRM이 생성한 RefBinder 클래스로부터 원격 결과 처리 클래스를 등록하게 된다. 따라서 RP는 해당 클래스를 RRReferBroker로부터 받아와 RR에게 작업 결과를 전송한다. 이 메커니즘은 자원 제공자가 관리자를 통하지 않고 직접 자원 요청자에게 결과를 전송함으로써 통신 시간이 감소된다.

3.3 GridRMF의 가상 조직 관리 시스템

그리드는 분산된 도메인에서 자원과 사용자들을 모아서 하나의 가상 조직을 구성한다. VMS는 그림 1의 GridRMF 구성 요소 중 GIM과 LRM들로 구성된 시스

표 4 메시지 헤더 정의 및 의미

통신 메시지 헤더 선언	통신 메시지 헤더 정의	의미
FRRTLRM_JOB_START	FROM_RR_JOB_START	RR이 RP에게 작업 시작 알림
FRRTGIM_HELLO	FROM_RR_HELLO	RR이 GIM에게 초기 접속 요청
FLRMTRR_JOB_START	FROM_RP_JOB_START	LRM이 RR에게 작업 시작 알림
FLRMTRP_JOB_START	FROM_LRM_JOB_START	LRM이 RP에게 작업 시작을 지시
FLRMTRP_NEW_JOB	FROM_LRM_PROCESS_NEW_JOB	LRM이 RP에게 새로운 작업할당 알림
FLRMTRP_WAIT	FROM_LRM_WAIT	LRM이 RP에게 작업 수행 중단 메시지
FLRMTRP_STOP	FROM_LRM_STOP	LRM이 RP에게 작업 종료 메시지
...		
FGIMTRR_LRM_ERROR	FROM_GIM_LRM_ERROR	GIM이 RR에게 LRM이 결함 발생 알림

템으로 사용자가 원격지 자원을 사용할 수 있도록 하는 중개 시스템이다. 따라서 VMS은 효율적인 자원 중개로 프레임워크의 성능 향상을 지원한다.

3.3.1 자원 중개 규칙

자원 중개는 그리드 사용자와 자원의 연결과 제어를 위해 자원 접속 규칙과 작업 중개 규칙을 따른다.

자원 접속 규칙은 사용자의 접근 제어, 사용자 인증, 작업 제출 및 모니터링을 한다. RR은 접속 요청으로 RREntry가 자동 생성된다. RR에 이식된 어플리케이션 명세에 따라 추가 정보(이미지 파일, 데이타 파일)를 포함한다. GIM은 어플리케이션 이름과 RREntry 정보에 의해 LRM을 검색한다. GIM은 작업 중개 규칙에 의해 해당 LRM에게 작업 중개를 이행하며, 검색된 LRM이 없을 경우 RR을 대기자 명단에 등록한다. RP는 GIM으로 부터 LRM 목록과 시스템 하드웨어 속성 정보를 얻기 위한 DLL 파일을 전송한 후 접속을 해제한다. LRM은 접속 요청으로 LRMEntry가 자동 생성되며, 상태 모니터링을 위한 LRM 가상 관찰자가 동적으로 생성된다. GRM은 LRM 갱신 정보 및 상태 정보를 저장하여 예기치 않은 시스템 오류나 의도적인 오류가 발생할 경우, 로그 파일을 추적하여 결함에 대처한다.

작업 중개 규칙은 요청된 어플리케이션에 적합한 자원을 제공하기 위해 4가지 요소에 의해 최적의 가상 조직을 선택한다. 그림 3은 정의한 4가지 요소에 따른 작업 중개 기법이다.

- 어플리케이션 실행 가능 여부: 프레임워크를 통해 기수행된 어플리케이션 목록의 제공으로 LRM은 수행 가능한 어플리케이션을 선정한다.
- Idle State: 선정한 LRM 목록에 대해 LRM 가상 관찰자는 Login/Wait 상태를 감지한다.
- Application Power: 어플리케이션의 연산 결과를 로그 파일로 기록한다. 로그 파일의 속성은 어플리케이션 이름, GIM과 LRM 정보, LRM과 RP의 노드 수, 초기 정적 CPU 속도의 총량, 전체 어플리케이션의 실행 시간 등이다.
- LRM Performance Index: 어플리케이션 로그 파일의 분석으로 성능 지수를 구한다. 성능 지수는 해당 어플리케이션에 대한 평균 CPU 속도와 노드 수에 비례하여 기대치에 근접한 값을 선택한다.

3.3.2 LRM의 상태 제어 및 자동 회복 기법

LRM은 프레임워크의 자원 집합체이며 중간 통신 중개자로써 제어 메시지 전송이 빈번하다. VMS은 LRM

```
public static int LRMRRJoin(RREntry nRR, EntryTable nLRMList, EntryTable nAppList){
    String nAppName=nRR.GetActiveApp();
    Vector nFilterApp=new Vector();
    Vector nFilterIdle=new Vector();
    /* Step 1 : Possibility of application execution */
    for (int i=0; i<nLRMList.size(); i++){
      String AppList[]=((LRMEntry)nLRMList.get(new Integer(i))).GetAppList();
      if (AppList==null) return -1;
      for (int y=0; y<AppList.length; y++){
        if (AppList[y].equals(nAppName)){
          nFilterApp.add(nFilterApp.size(),(LRMEntry)nLRMList.get(new Integer(y)));
          break;
      }
    }
      if (nFilterApp.size()==0) return -1;
      /* Step 2 : Idle state */
      for (int i=0; i<nFilterApp.size(); i++){
        if(((LRMEntry)nFilterApp.get(i)).GetStateOfLRM().equals(LRMStateDefine.LRMSTATE_LOGIN)||
        ((LRMEntry)nFilterApp.get(i)).GetStateOfLRM().equals(LRMStateDefine.LRMSTATE_WAIT)){
          nFilterIdle.add(nFilterIdle.size(),(LRMEntry)nFilterApp.get(i));
        }
      }
      if(nFilterIdle.size()==0) return -1;
      if(nFilterIdle.size()==1){
        LRMEntry SelectLRM=(LRMEntry)nFilterIdle.get(0);
        return SelectLRM.GetLRMID();
      }
      /* Step 3 : Application power */
      double appPower = this.GetPowerFromHistoryManager(nAppName);
      /* Step 4 : LRM Performance index */
      iSelectLRMID = this.GetMaxPowerLRM(nFilterIdle);
      return iSelectLRMID
}
```

그림 3 LRM/RR 작업 중개 기법

에 따른 가상 관찰자를 생성하여 연산 기능과 제어 기능을 분리한다. LRM 가상 관찰자는 GIM이 위임한 어플리케이션을 개별 정책에 따라 스케줄링을 하거나 가상 조직 내 RP들을 관리하는 LRM의 상태만을 감시하는 모니터이다.

LRM은 해당 어플리케이션 수행 중에 어떠한 결함이 발생할 지라도 실시간 탐지를 통해 다른 LRM으로부터 대응 자원을 얻어 서비스에 대한 가용성을 보장하기 위해 자동 회복 전략을 제시한다.

RR은 의도적으로 GridRMF과의 연결이 해제 상태가 되기까지는 자원들의 사용은 지속되어야 한다. 또한 RR은 수행되는 어플리케이션의 명세를 변경하면서 RR과 연결된 LRM에 속해 있는 자원들의 이용이 가능해야 한다. GIM은 LRM 가상 관찰자에 의해 결함을 감지한다. 이때, 결함이 발생된 LRM에 연결된 RP들은 오류나 결함이 발생하지 않았다고 가정한다. LRM의 결함이 검출되면 유휴 상태에 있는 새로운 LRM을 찾는다. RR은 새로운 LRM을 참조할 수 있는 핵심 정보인 주소가 전달되고 원격 참조 객체와 작업 테이블을 전송하여 새로운 LRM에 의해 연산이 자동으로 진행된다. 새로운 LRM 검색이 실패하면, RR은 대기 상태가 되며 대기자 목록에 추가된다. RR의 작업 테이블 관리와 원격 객체 참조 전략은 해당 LRM이 결함이 발생할지라도 중복 연산이 발생하지 않는다.

3.3.3 VMS의 모니터링 가시화

VMS는 관리자와 시스템 설계자에게 모니터링 정보를 다양한 형태의 가시화 뷰로 제공한다. VMS의 모니터링 가시화 대상은 GIM 측에서 LRM과 RP이다. 가시화 요소는 LRM 가상 관찰자에 의해 추적된 상태를 기록하는 LRMEntry와 RREntry의 갱신 정보가 뷰에 전달된다. VMS의 모니터링 가시화는 그림 4와 같이 2개의 뷰를 제공한다. LRM/RR Join View는 RR과 LRM 간의 연관관계를 테이블 컴포넌트로 제공하고 선택된 행의 세부 정보는 텍스트 컴포넌트로 출력된다. LRM/RP Hierarchical View는 프레임워크의 노드 구조를 계층적으로 제공한다. 그림 4의 (b)는 2개의 LRM과 7개의 RP의 구성이며, 특히 점선으로 표시된 영역은 하나의 RP로 동시에 2가지 연산(LRM#1, LRM#2)을 수행하고 있다.

3.4 GridRMF의 자원 관리 시스템

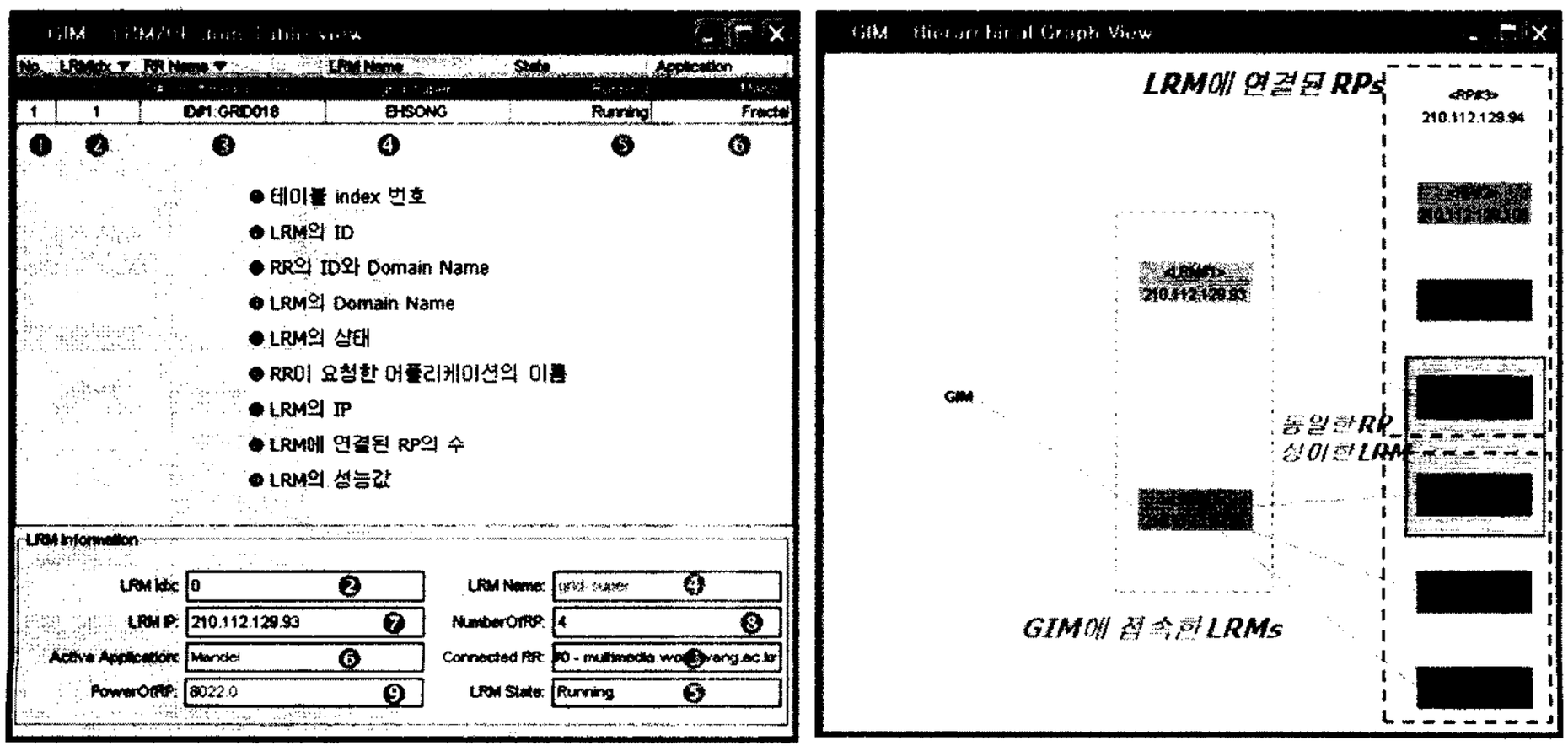
RMS는 프레임워크의 성능을 향상시키기 위한 모니터링 정보에 의한 작업 할당 기법을 제시한다.

3.4.1 작업할당 알고리즘

RMS의 작업할당 알고리즘은 연산에 참여하는 RP의 관리 여부에 따라 표 5와 같이 분류한다.

UTA는 RP의 성능에 상관없이 균등하게 작업을 할당하는 알고리즘이다. 작업 도중 RP의 상태를 고려하지 않으며 재할당이 일어나지 않는다. CPTA는 RP의 벤치마킹(LINPACK)을 통한 성능을 고려하여 작업을 할당한다. 이것은 UTA보다 빠른 수행 시간을 보이지만 벤치마킹 때의 RP의 성능이 작업을 수행하는 동안에 고정적이다. SPTA는 RP의 성능 기반 할당으로 이미 작업을 완료한 RP는 성능이 느린 RP의 작업을 재할당함으로써 CPTA 보다 빠른 수행 시간을 보인다. 하지만, SPTA는 재할당시 RP의 성능을 초기 벤치마킹 값으로 판단하므로, 가변적인 RP의 성능과 상태 변화에 대처하지 못한다.

DPTA는 RP의 성능 변화, 작업 도중 RP의 참여와



(a) (b)

그림 4 VMS 모니터링 정보 가시화

표 5 RMS 작업할당 알고리즘

분류	작업할당 알고리즘	성능과 재할당 요소	RP수
Static	UTA(Uniform Task Allocation)		Fixed
	CPTA(CPU Performance Task Allocation)	MFLOPS	
	SPTA(Static Performance Task Allocation)	MFLOPS	
Dynamic	DPTA(Dynamic Performance Task Allocation)	CPU Speed, Job History	Variable
Adaptive	APTA(Adaptive Performance Task Allocation)	CPU Speed, CPU Usage, Proportional Factor, Realtime State Monitoring	

이탈을 고려한다. 이것은 CPU 속도와 기 수행한 작업 정보인 작업 히스토리를 재할당 요소로 사용한다. APTA는 RP의 성능을 단순히 물리적인 값이 아닌 실시간 모니터링 정보를 기준으로 한다. RP의 성능은 사용자 또는 내부의 시스템에 의해 사용된 CPU 사용률의 범위에 따라 전체 수행 시간에 미치는 정도가 다르므로 그 기준이 되는 값은 비례 상수(Proportional Factor: PF)를 정의하여 적용한다. PF는 동일한 RP에 동일한 연산을 수행시켰을 때 CPU 사용률의 범위에 따른 RP의 작업 수행 시간을 구하여 얻는다. RP의 CPU 속도는 $\{sRP_0, sRP_1, \ldots, sRP_{sum(i)-1}\}$이고 재할당시 전체 CPU 사용률의 기대치가 **cpuUA**이며 작업 프로세서 사용률의 기대치가 **JobUA**일 때, 임의의 **RP**의 성능값은 다음과 같으며 그림 5는 구간별 PF이다.

$$RP_i\,Performance = (100 - (cpuUA - jobUA)) \times PF \times sRP_i \times 0.01$$

그림 6은 성능이 다른 4개의 RP들로 하나의 RMS를 구성하여 제안한 작업할당 알고리즘을 비교한 결과이다. (a)에서 DPTA와 APTA는 고정된 태스크를 할당받은 SPTA에 비해 수행한 태스크 크기에 변화가 일어났으며, (b)와 같이 성능 향상이 있음을 알 수 있다. 특히

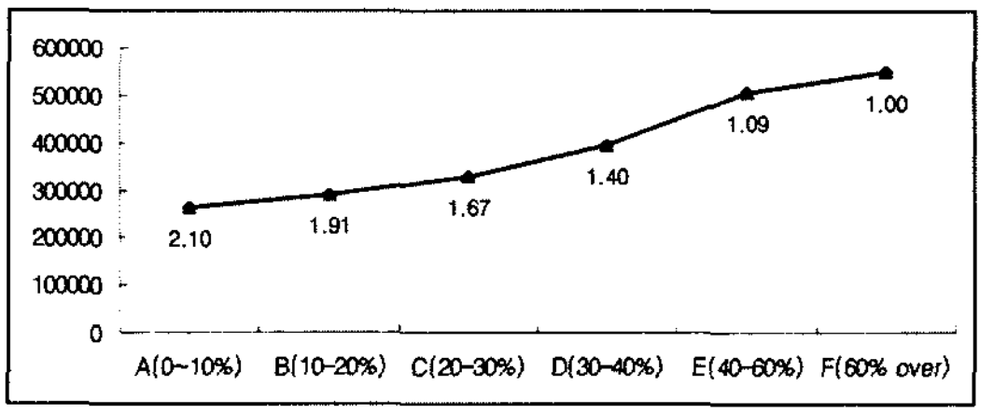


그림 5 구간별 비례 상수

APTA는 RP의 성능 변화에 따른 효율적인 재할당으로 전체 수행 시간이 감소하였다. (c)는 전체 CPU 사용률을 기준으로 작업을 처리하는데 소요된 작업 CPU 사용률과 사용자에 의해 독립적으로 쓰여진 사용자 CPU 사용률을 일정한 간격으로 측정하여 기대치를 구한 결과이다. 사용자 CPU 사용률이 높을수록 전체 CPU 사용률 대비 작업 CPU 사용률의 감소로 전체 수행 시간에 영향을 미쳤다. (d)는 사용자 CPU 사용률을 임의로 부과하여 PF를 적용한 결과로 APTA가 가장 빠른 수행 시간을 보였다.

3.4.2 RMS 모니터링 가시화

RMS 모니터링 가시화는 RP의 상태 변화 값을 측정

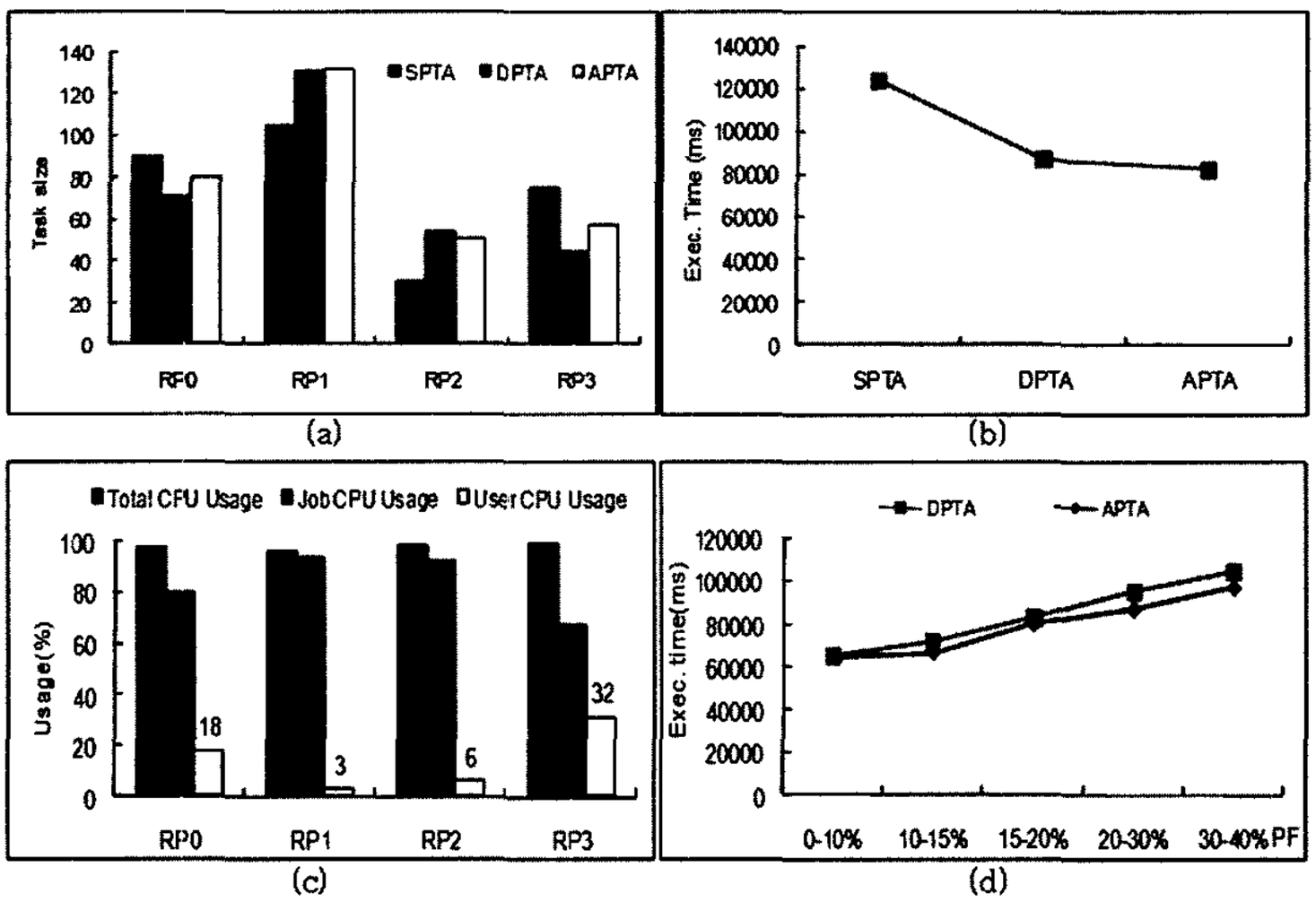


그림 6 작업할당 알고리즘의 성능 평가

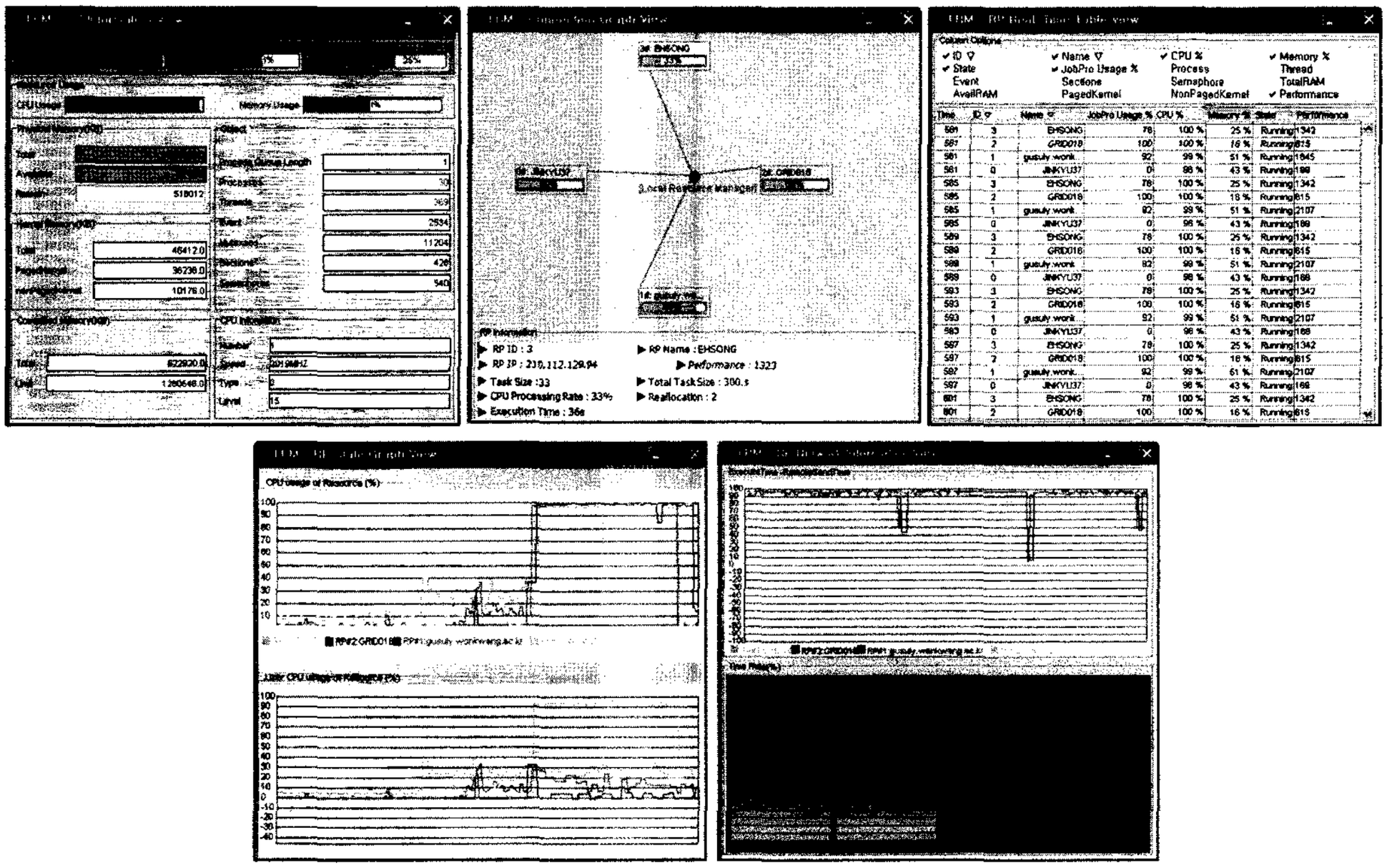


그림 7 RMS 모니터링 정보 가시화

하여 연산의 효율성을 향상하고 고장에 대처한다. RP의 메타데이타 정보가 갱신되면, 동적으로 생성된 RP 가상 관찰자를 통해 상태 정보를 받아들인 후 RPEntry에 전달되어 모니터링 가시화된다. RP의 상태 정보와 분석 결과는 그림 7과 같이 5개의 뷰로 제공된다. RP Information View는 선택된 RP의 메타데이타 정보를 진행바와 텍스트 컴포넌트로 나타내며, 추출된 하드웨어 정보를 분석하여 RP의 성능 평가에 적용된다. RP Real-time Table View는 모든 RP들의 정보를 일정 시간 마다 갱신하여 테이블로 표현한다. RP State Graph View는 모든 RP들의 CPU 사용률과 사용자 CPU 사용률을 실시간 그래프 형태로 표현한다. Connection Graph View는 LRM에 연결된 RP의 작업 처리 상황을 관측한다. RP Network Information View는 RP들의 작업을 처리하는데 소모된 시간과 네트워크 전송 시간 간의 비율을 막대 그래프 형태로 표현한다.

4. 그리드 자원 관리 프레임워크의 적용 및 수행 결과

4.1 어플리케이션 계층 구조

그리드 서비스 환경은 그리드 어플리케이션 명세의 다양성으로 인해 최소한의 연관성을 가지도록 구조적인 분할이 요구된다. 본 논문은 그림 8과 같이 프레임워크와 어플리케이션들 간에 어플리케이션 프록시(application proxy)를 두어 특정 어플리케이션에게 그리드 서비스를 지원하는 정형화된 공통 API로 제공한다. 어플리케이션 프록시는 특정 어플리케이션에 접근하기 위한 플러그 인을 어플리케이션 이름으로 정의하며, 어플리케이션 템플릿에 의해 동적으로 객체를 생성한다.

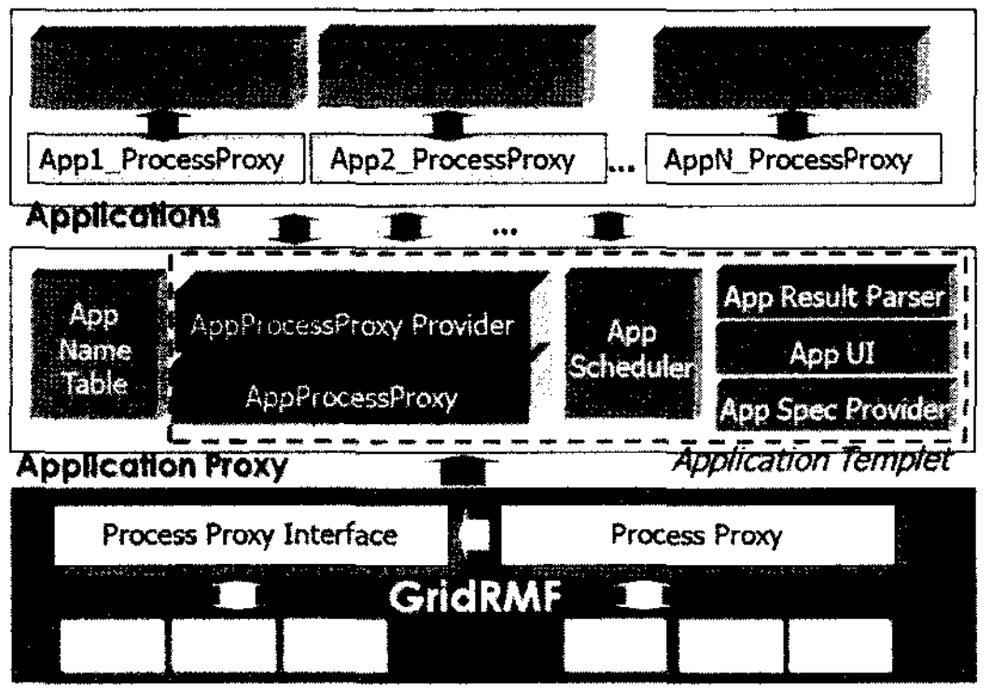


그림 8 GridRMF와 어플리케이션과의 연관성

4.2 어플리케이션 적용 및 수행 결과

GridRMF의 수행결과는 태스크들간에는 연관성이 없으며 적은 데이타에 비해 많은 양의 연산을 요구하는 어플리케이션을 사용하였다. 적용된 어플리케이션은 LSM 채색 기법에 따른 만델브로트 프랙탈 이미지 처리와 랜더링 이미지를 생성하였다.

그림 9는 LRM 자동 회복 전략의 실례이며, 원표시는 수행 과정을 보인다. ❶두 개의 가상조직과 6개의 RP

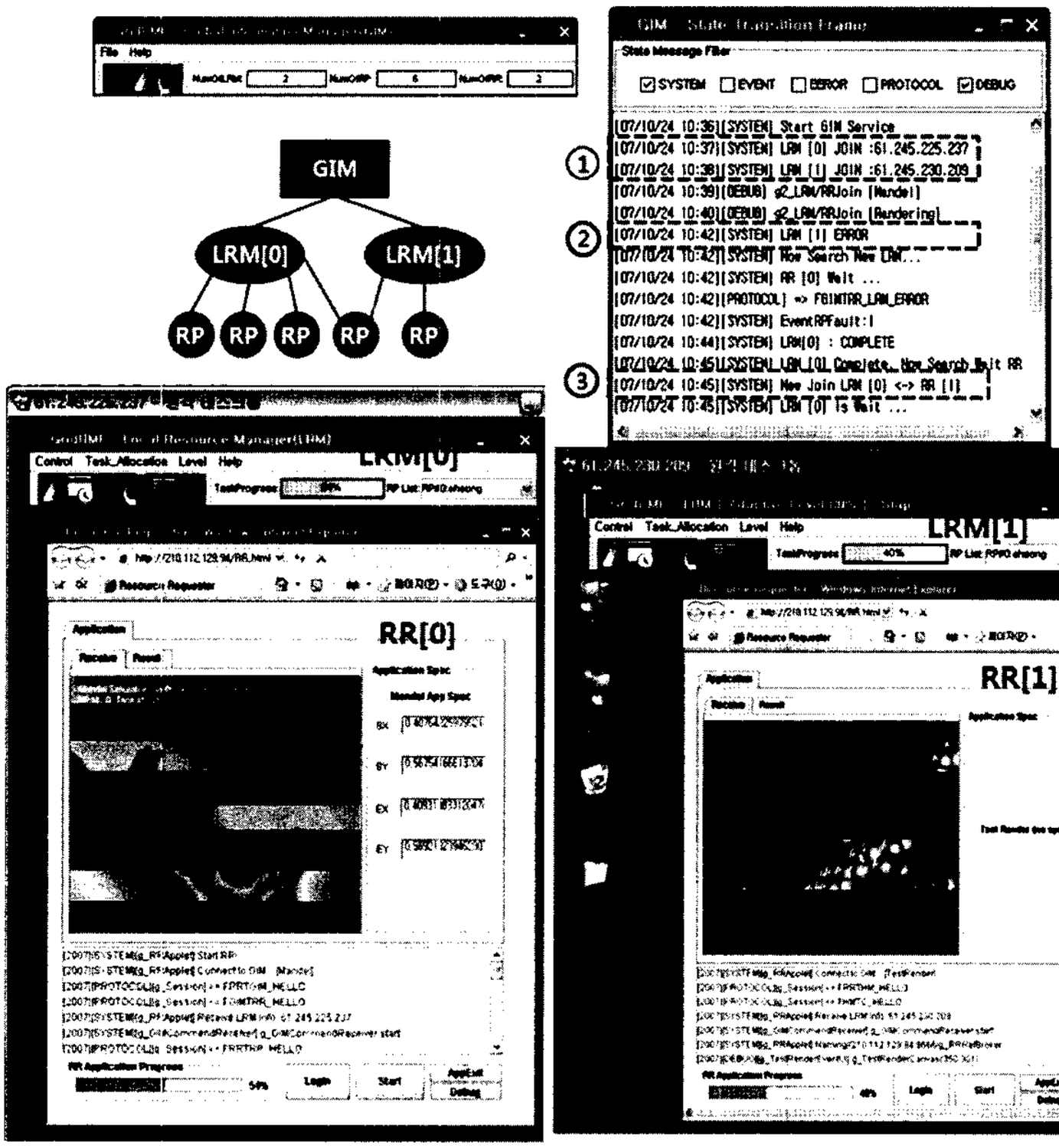


그림 9 LRM 자동 회복 전략 적용 실례

(LRM[0]-4RP, LRM[1]-2RP)을 구성하여 각각 독립된 어플리케이션 수행 도중 ❷LRM[1]의 결함이 발생하였으나 ❸LRM[0]과 RR[1]을 중개하여 연산을 마친 LRM[0]이 LRM[1]을 대신하여 작업을 지속한다. GIM은 LRM 가상 관찰자에 의해 LRM의 작업 정보를 기록함으로써 연산의 중복이 발생하지 않는다.

그림 10은 2개의 RP에 의해 작업을 수행 도중에 2개의 RP가 추가되었으며 APTA 기법에 의해 재할당이 이루어져 작업을 수행한 예이다. (c)의 채색변화는 성능 변화 및 RP 추가에 따른 재할당의 가시화를 보인 것이

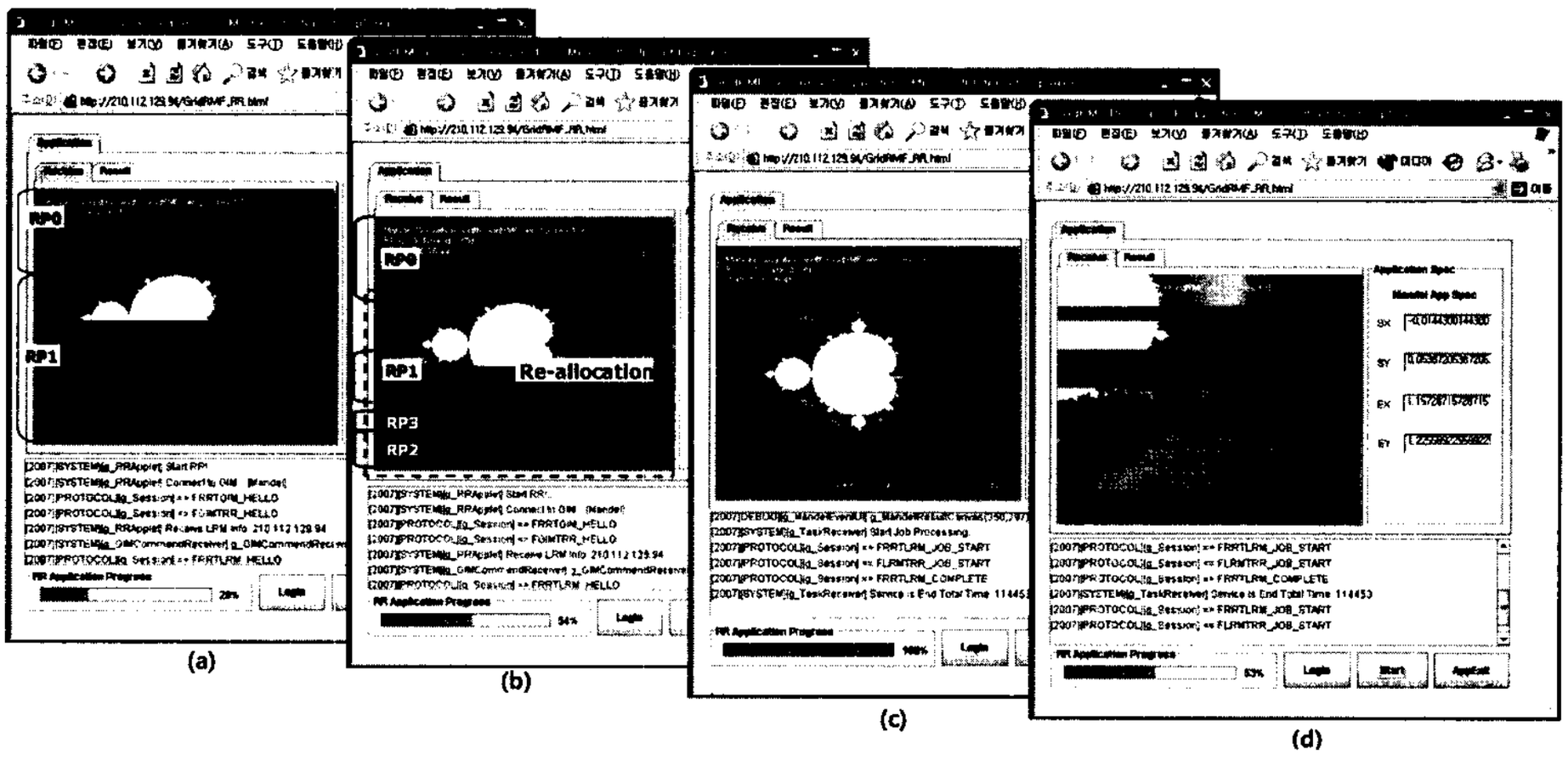


그림 10 APTA 기법 및 작업 명세 변경 실례

며, (d)는 프레임워크를 종료하지 않는 한 작업 명세의 변경으로 지속적인 연산을 수행한다.

5. 결 론

본 논문은 그리드 서비스 환경에서 효율적인 자원 관리를 위한 GridRMF를 설계하였다. GridRMF는 그리드 자원을 작업에 참여하는 목적과 특성에 따라 GIM, LRM, RP, 그리고 RR로 구성요소를 구분하고, 자원 운영 관리 도메인을 VMS와 RMS로 분류하는 계층적 자원 관리 구조를 제시하였다. 계층적 자원 관리 구조를 따르는 구성 요소간의 데이타, 명령, 쿼리 등의 전송에 대한 능동적 결합이 이루어지도록 통신 모델을 정의하였으며, 자원 정보의 중복 저장을 금지 및 자원 사용의 일관성을 위해 엔트리 생성을 통해 관리하는 정보 저장 모델을 제시하였고, 서버의 오버헤드를 줄이기 위해 RR에게 작업 결과를 직접 전송하는 원격 객체 참조 모델을 정의하였다. VMS는 LRM 성능 지수에 의한 최적 가상 조직 선택 기법과 서버(VO)의 결함이 발생할지라도 중복 연산을 고려한 LRM 자동 회복 전략을 제안하였다. RMS는 그리드 자원의 상태 변화, 참여와 탈퇴 및 고장에 대한 대처 방안으로 적응적 성능 기반 작업할당 알고리즘으로 부하 균등화와 결함 허용을 지원하였다. 적응적 성능 기반 작업할당은 실시간 자원 상태 모니터링을 통해 추출된 RPPerformance값의 적용으로 자원의 가변성에 따른 성능 향상을 가져왔다. GridRMF와 그리드 어플리케이션간의 상호 의존성을 최소화하기 위한 메커니즘으로 어플리케이션 프록시를 두어 어플리케이션과의 독립성을 지원한다. 마지막으로 본 논문은 두 개의 어플리케이션을 GridRMF가 제시하는 서비스에 적용하여 그리드 자원의 적응성과 수행성을 분석하였다.

향후 연구로 본 논문에서 제시한 프레임워크의 접근 관점에서 사용자 중심으로 작업 수행을 위한 세분화된 작업 절차를 따르는 사용자 접근 인터페이스의 기능 추가와 태스크들간의 상호 의존성 있거나 기타 특수한 어플리케이션 적용이다. 본 연구에서 개발된 프레임워크를 웹 서비스 컴포넌트화하기 위한 WSRF 기반 인터페이스 API 규격 및 기술 개발이 요구된다.

참 고 문 헌

[1] I. Foster, C. Kesselman, S. Tueche, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International Journal Supercomputer Applications, Vol.15, No.3, pp. 200-222, 2001.

[2] I. Foster, C. Kesselman, "The Grid: Blueprint for a Future Computing Infrastructure," Morgan Kaufmann Publishers, USA, 1999.

[3] R. Buyya, S. Chapin, D. DiNucci, "Architectural Models for Resource Management in the Grid," Lecture Notes in Computer Science, no. 1971, Grid Computing-Grid 2000, pp. 18-34, 2000.

[4] Young-Sik Jeong, Eun-Ha Song, Cheng-Zhong Xu, "Dynamic and Adaptive Parallel Task Processing on GRID Service Architecture," Proceedings of the 23rd IASTED International Multi-Conference, PDCN, pp. 270-275, 2005.

[5] Young-Sik Jeong, Bock-Ja Park, Eun-Ha Song, "Realtime Monitoring for Parallel Distributed Processing System based on Internet," VECPAR04, High Performance Computing for Computational Science, Spain, pp. 857-862, 2004.

[6] Rajkumar Buyya, Steve Chapin, and David DiNucci, "Architectural Models for Resource Management in the Grid," Grid Computing GIRD 2000. First IEEE/ACM International Workshop Bangalore, India, pp. 20-33, 2000.

[7] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," International Journal supercomputer Applications, Vol.11, No.2, pp. 115-128, 1997.

[8] Steve J. Chapin, Dimitrios Katramatos, John Karpovich, and Andrew S. Grimshaw. "The Legion Resource Management System," IPDPS Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, No. 1659, pp. 162-178, 1999.

[9] Henri Casanova and Jack Dongarra, "NetSolve: A Network Server for Solving Computational Science Problems," Intl. Journal of Supercomputing Applications and High Performance Computing, Vol.11, No.3, pp. 21-223, 1997.

[10] Microsoft Corporation. Performance Data Helper Library, "http://msdn.microsoft.com/library/en-us/perfmon/base/performance_data_helper.asp"

[11] A. Waheed, W. Smith, J. George, J. Yan, "An Infrastructure for Monitoring and Management in Computational Grids," Lecture Notes In Computer Science, No.1915, pp. 235-245, 2000.

[12] Haban, D. Shin, K. G, "Application of real-time monitoring to scheduling tasks with random execution times," IEEE Transactions on Software Engineering, Vol.16, pp. 1374-1389, 2000.

[13] Domingues, P. Silva, L. Silva, J.G., "DRMonitor - A Distributed Resource Monitoring System," Parallel, Distributed and Network-Based Processing, Proceedings. Eleventh Euromicro Conference on pp. 127-133, 2003.

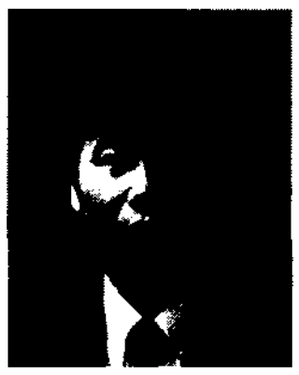
[14] Haban, D. Shin, K. G, "Application of real-time monitoring to scheduling tasks with random execution times," IEEE Transactions on Software Engineering, Vol.16, pp. 1374-1389, 1990.

[15] M. W. Knop, P. K. Paritosh, P. A. Dinda, and J. M. Schopf, "Windows performance monitoring and

data reduction using watchtower and argus," Technical Report NWU-CS-01-6, Department of Computer Science, Northwestern University, 2001.
[16] Sun Microsystems, Inc, Overview of the JNI(Java Native Interface).
[17] Roger T. Stevens, Advanced Fractal Programming in C, M&T Books, pp. 117-123, 1990.

송 은 하

1997년 원광대학교 통계학과 졸업(학사) 2000년 원광대학교 대학원 컴퓨터공학과 졸업(석사). 2006년 원광대학교 대학원 컴퓨터공학과 졸업(박사). 2007년~현재 원광대학교 전기전자 및 정보공학부 전임강사. 관심분야는 그리드컴퓨팅, 시맨틱그리드, 분산병렬시스템



정 영 식

1987년 고려대학교 수학과 졸업(학사) 1989년 고려대학교 대학원 전산학과 졸업(석사). 1993년 고려대학교 대학원 전산학과 졸업(박사). 2004년 웨인 주립대학교 객원교수. 1993~현재 원광대학교 전기전자 및 정보공학부 교수. 관심분야는 그리드컴퓨팅, 시맨틱그리드, 분산병렬시스템