그리드 상호 운용을 위한 Ganga 플러그인 설계 및 구현

(Design and Implementation of Ganga Plugins for Grid Interoperability)

김 한 기 † 황 순 욱 †† 이 윤 기 ††† 김 은 성 †††† 염 헌 영 †††††
(Hangi Kim) (Soonwook Hwang) (Yoonki Lee) (Eunsung Kim) (Heon Y. Yeom)

요 약 고에너지 물리나 생명공학 분야의 거대 문제를 풀기 위해서는 다수의 계산 자원이 요구되는데 이는 하나의 그리드 환경을 통해서는 확보되기가 어려울 수 있다. 각 그리드 환경에서 각각의 사용자 인터페이스를 통해서 작업을 제출할 수 있지만, 수 백 개 이상의 작업들로 이루어지는 거대 문제를 각기 다른 그리드 환경에서 따로 관리하기에는 많은 비용과 노력이 요구될 수 있다. 본 논문에서는 이와 같은 문제점을 그리드 사용자 인터페이스 시스템인 Ganga의 Gridway 백엔드와 InterGrid 백엔드를 개발하여 이기종의 그리드 환경에서 동일한 사용자 인터페이스를 제공하여 해결하고자 한다. 우리는 Globus 기반의 그리드 자원을 백엔드로 사용할 수 있도록 Ganga의 Gridway 백엔드 모듈을 개발하였다. 또한 gLite 기반의 그리드 자원과 Globus 기반의 그리드 자원을 함께 사용할 수 있도록 지원하기 위해서 InterGrid 백엔드 모듈도 개발하였다. 이와 함께, 개발된 백엔드 모듈들의 실용성을 보여주기 위해서 WISDOM 프로젝트에서 사용되고 있는 AutoDock 프로그램을 지원하는 Autodock application 플러그인을 개발하여, Globus 기반의 PRGMA 자원과 gLite 기반의 EGEE 자원을 동시에 활용하는 연동실험을 수행하였다.

키워드 : 그리드 컴퓨팅, 그리드 상호 운용, 작업 관리 시스템, Ganga, Gridway, 인터그리드, AutoDock

Abstract To solve big problem in high energy physics or bioinformatics, it needs a large number of computing resources. But it hard to be provided by one grid environment. While user can submit each job by using it's own user interface in each grid environment, it may need many cost and efforts to manage several hundred jobs conserved in each grid environment separately. In this paper, to solve this problem we develop Ganga's Gridway backend and InterGrid backend. Therefore as we provide the same grid user interface between different grid environments. We developed a Gridway backend module that provide users with access to globus-based grid resources as well. We have also developed an InterGrid backend that allows users to submit jobs that have access to both glite-based resources and globus-based resources. In order to demonstrate the practicality of the new backend plugin modules, we have integrated the AutoDock application used by WISDOM project into Ganga as a new built-in application plugin for Ganga. And we preformed interoperability experiment between PRAGMA grid based on Globus and EGEE grid based on gLite.

Key words : Grid Computing, Grid Interoperability, Job Management System, Ganga, Gridway, InterGrid, AutoDock

† 정 회 원 : 한국과학기술정보연구원 차세대연구환경개발실 연구원
hgkim@kisti.re.kr
†† 정 회 원 : 한국과학기술정보연구원 차세대연구환경개발실 책임연구원
hwang@kisti.re.kr
††† 비 회 원 : 서울대학교 컴퓨터공학부
yoonki.lee@gmail.com
†††† 학생회원 : 서울대학교 컴퓨터공학부
eskim@dcslab.snu.ac.kr
††††† 종신회원 : 서울대학교 컴퓨터공학부 교수
yeom@snu.ac.kr

논문접수 : 2009년 8월 21일
심사완료 : 2009년 9월 29일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제15권 제12호(2009.12)

1. 서 론

최근 들어, 수년간의 그리드 기술의 발전으로 다양한 목적을 가진 그리드 환경들이 등장하고 있으며 많은 과학기술자들이 이러한 환경을 이용하여 연구 및 실험을 진행하고 있다. 하지만 이러한 그리드 환경들은 자신들의 목적에 맞게 다양한 그리드 미들웨어를 이용해서 구축되고 있다. 예를 들어 EGEE(Enabling Grids for E-sciencE) 사용하고 있는 gLite 미들웨어와 OSG(Open Science Grid), TeraGrid, NGS(National Grid Service), PRAGMA(Pacific Rim Applications and Grid Middleware Assembly) 등에서 사용되고 있는 Globus 미들웨어, 그리고 DESY(Deutsches Elektronen Synchrotron)에서 사용되고 있는 Unicore 등이 대표적인 그리드 미들웨어로 사용되고 있다.

서로 다른 그리드 미들웨어(middleware)에 기반을 둔 그리드 환경은 서로 호환성을 제공하지 않는다. 따라서 사용자가 특정 그리드 환경을 사용하기 위해서는 해당 그리드 환경에 구축된 미들웨어 인터페이스를 학습해야 되며, 다른 종류의 미들웨어로 구축된 그리드 환경을 이용하기 위해서는 또 다시 사용법을 학습하여야 하는 문제가 발생되고 있다.

또한, 고에너지 물리나 생명공학 분야의 거대 문제를 풀기 위해서는 다수의 계산 자원이 요구되는데 이는 하나의 그리드 환경을 통해서는 확보되기가 어려울 수 있다. 이를 위해서 다수의 그리드 환경에 각각의 사용자 인터페이스를 통해 작업을 각각 제출할 수 있지만, 수백 개 이상의 작업들로 이루어지는 거대 문제를 각기 다른 그리드 환경에서 따로 관리하기에는 많은 비용과 노력이 요구될 수 있다.

본 논문에서는 이와 같은 문제점을 해결하기 위하여 그리드 사용자 인터페이스 시스템인 Ganga[1]의 Gridway[2] 백엔드와 InterGrid 백엔드를 개발하여 다양한 그리드 환경에서 동일한 인터페이스를 통해서 작업을 제출할 수 있도록 하며, 다수의 그리드 환경이 이용 가능할 경우 사용자가 이를 편리하게 사용할 수 있도록 해주는 시스템을 개발하고자 한다.

이후 본 논문의 구성은 다음과 같다. 2장에서는 그리드 상호 운용과 관련된 기존 연구들에 대해서 알아보고, 3장에서는 Ganga와 Gridway에 대해서 알아본다. 4장에서는 Gridway 플러그인, InterGrid 플러그인 그리고 Autodock 플러그인의 설계와 구현에 대해서 알아본다. 5장에서는 개발된 플러그인들을 활용하여 Globus 그리드 환경의 PRAGMA와 gLite 그리드 환경의 EGEE를 함께 연동하는 실험에 대해서 알아본다. 마지막으로 6장에서는 결론 및 향후 연구과제에 대해서 알아보도록 하겠다.

2. 관련연구

그리드 상호 운용과 관련해서 OGF의 GIN(Grid Interoperability Now) 커뮤니티 주도로 많은 연구들이 진행되고 있다. GIN 커뮤니티는 데이터 관리와 이동, 정보 서비스와 스키마의 4개 분야에 대해서 미들웨어 간의 상호 운용에 관한 연구 및 논의를 진행 중에 있다. GIN 의 여러 연구 프로젝트와 연계해서 현재 사용되고 있는 프로덕션 레벨 그리드의 미들웨어 레벨의 상호 운용도 연구되고 있다. 사우스햄턴(Southampton) 대학의 OMII-UK(Open Middleware and Infrastructure Institute)팀은 JSDL(Job Submission Description Language)과 OGSA-BES(Open Grid Services Architecture Basic Execution Service), 그리고 HPC-Profile(High Performance Computing Profile)에 기반을 둔 작업 제출 메커니즘의 상호 운용에 대한 연구를 수행하였다[3]. GridSAM이나 NGS Application Repository와 같은 툴들도 JSDL을 표준 작업 기술 언어로 사용해서 다양한 그리드 환경에 작업을 제출하고 있다. 작업 제출 레벨의 상호 운영의 또 다른 예제로는 새로운 Condor 버전을 들 수 있다. 새로운 Condor는 여러 Globus Toolkit (GT) 버전과 Unicore, 그리고 NorduGrid 등에 작업을 제출할 수 있다.

영국에서 가장 큰 NGS(National Grid Service)와 GridPP(UK Computing for Particle Physics) 그리드 센터는 미들웨어 레벨의 그리드 상호 운용에 대해서 단기, 장기별 해결방법에 관한 연구조사결과를 발표하였다. Globus, gLite, Unicore와 같이 널리 사용되고 있는 그리드 미들웨어간의 장기적인 상호 운영을 제공하기 위한 GRIP이나 OMII-Europe 같은 유럽의 프로젝트들도 시작되었다.

이밖에도 그리드 미들웨어 레벨이 아닌 워크플로우(workflow) 레벨에서 P-GRADE 포털을 활용한 그리드 상호 운용 연구[4]도 진행되고 있다.

본 논문에서는 그리드 사용자 인터페이스 시스템인 Ganga의 Gridway 백엔드를 개발하여 사용자가 Globus 미들웨어에도 작업을 제출할 수 있도록 하였으며, InterGrid 백엔드를 개발하여 사용자가 동일한 인터페이스를 사용하여 gLite 미들웨어와 Globus 미들웨어에 선택적으로 작업을 제출할 수 있도록 하였다.

Ganga 사용자가 사용할 수 있는 미들웨어의 종류를 확장한 측면과, 이를 통해서 그리드 사용자에게 이기종의 그리드 환경을 동일한 인터페이스를 사용해서 활용할 수 있도록 해 주었다는 측면을 본 논문의 기여라고 볼 수 있다.

3. Ganga & Gridway

이번 장에서는 작업 정의 및 관리 도구인 Ganga와 오픈 소스 메타 스케줄러(Meta-scheduler)인 Gridway에 대해서 알아본다.

3.1 Ganga

Ganga는 파이썬으로 구현된 작업 정의 및 관리 도구로 사용하기 편한 작업 환경을 제공한다[5]. Ganga는 GAUIDI/ATHENA and Grid Alliance의 약어로, GAUDI와 ATHENA는 CERN의 4대 실험에 속하는 ATLAS, LHCb 실험에서 공통으로 사용되는 프레임워크(framework) 이다. Ganga라는 이름에서 알 수 있듯이 초기에는 ATLAS 실험과 LHCb 실험에 참여하는 물리학자들에게 사용하기 쉬운 그리드 사용자 인터페이스를 제공해 주고자 하는 목적으로 개발되었으나, 현재는 고에너지 물리 분야뿐만 아니라 Geant4 regression 테스트나 웹 검색 이미지 분류 등 범용의 목적으로 사용되고 있다. Ganga는 실험 결과의 분산 해석 시스템(distributed-analysis system)을 주요 기능으로 제공하고 있다.

Ganga에서 하나의 작업은 그림 1과 같이 여러 개의 빌딩 블록으로 이루어져 있다. 모든 작업에는 실행 하고자 하는 소프트웨어(응용 프로그램)와 작업을 수행하고자 하는 시스템(백엔드)이 정확히 명시되어야 한다. 여기에서 백엔드는 사용자가 응용 프로그램에서 지정한 소프트웨어가 실제로 실행되는 시스템을 의미한다. 사용자는 자신의 프로그램을 프로덕션 레벨 그리드에 바로 적용해서 실행 하는 것이 아니라, 먼저 로컬호스트(Localhost)에서 실행시켜 보고, 프로그램이 문제없이 실행된다고 생각되면, 그 다음으로 PBS(Portable Batch System), LSF(Load Sharing Facility), SGE(Sun Grid Engine)등과 같은 배치 시스템(Batch system)에서 실행 시켜보고, 마지막으로 Globus나 gLite 같은 프로덕션 레벨의 그리드에서 작업을 실행하는 과정으로 Ganga를 활용할 수 있다. Ganga에서는 프로그램이 실제로 실행되는 백엔드를 로컬호스트, 배치 시스템, 그리드(Grid) 등으로 간편하게 변경해서 실행할 수 있다. Ganga는 "Configure once, run anywhere", 즉 프로그램 실행에 필요한 설정사항을 한번만 설정해 주면 여러 시스템 환경에서 프로그램을 편리하게 실행할 수 있는 환경을 제공해 주고 있다.

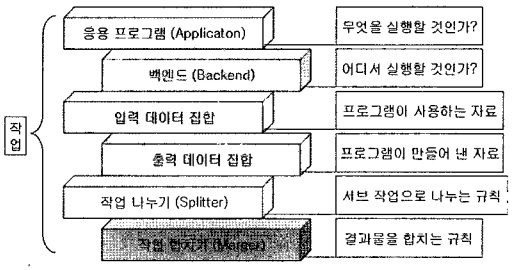

그림 1 Ganga 작업(Job)의 구성 블록

필요에 따라서, 큰 작업을 여러 개의 작은 서브 작업으로 나누어서 병렬적으로 실행한 후, 결과를 합칠 수 있도록 해주는 Spitter and Merger 기능을 제공한다.

Ganga에서는 서로 다른 타입의 응용 프로그램, 백엔드, 데이터 집합 등을 그림 2와 같은 플러그인 클래스 형태로 구현할 수 있어 손쉬운 기능 확장성을 제공한다.

Ganga에서는 현재 LHC(Large Hadron Collider) 실험에서 사용 중인 여러 응용 프로그램을 지원하고 있다. 대표적인 프로그램으로 ROOT, Gauss, Boole, Brunel, DaVinci 등이 있다. 또한 로컬 백엔드를 기본으로 PBS (Portable Batch System), LSF(Load Sharing Facility), Condor, LCG(LHC Computing Grid), DIRAC, gLite, PANDA, ARC 등을 백엔드로 사용가능하다. 하지만 Ganga는 CERN에서 개발되어 gLite를 그리드 기본 미들웨어로 사용하고 있어서 미국 등지에서 많이 사

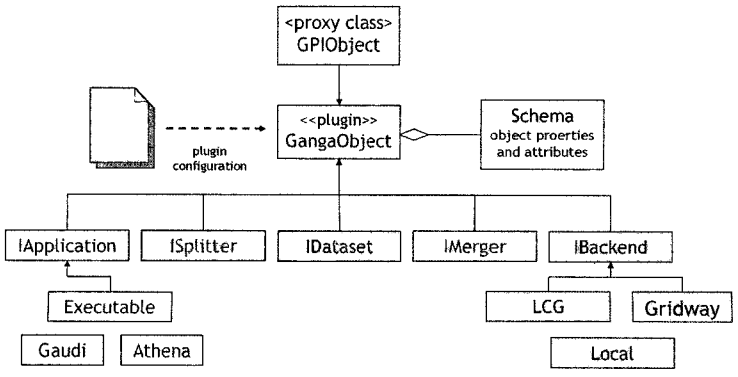

그림 2 Ganga 작업(Job) 클래스를 구성하는 여러 추상 인터페이스(abstract interface)를 구현한 컴포넌트 클래스

용하는 Globus는 백엔드로 지원하고 있지 않았다. 이에 본 논문에서는 Globus 메타 스케줄러인 Gridway를 지원하는 새로운 백엔드 플러그인을 개발하여 Ganga에서도 Globus 기반의 그리드를 사용할 수 있도록 하였다.

3.2 Gridway

Gridway[6,7]는 Globus Toolkit의 메타 스케줄링 컴포넌트로 GRAM(Grid Resource Allocation and Management), MDS(Monitoring and Discovery Service), GridFTP의 Globus 서비스 위에서 스케줄러 가상화 계층(scheduler virtualization layer)을 제공한다.

오픈 소스(Open source) 프로젝트로 최종 사용자(end user)나 응용 프로그램 개발자에게 로컬 DRM(Distributed Resource Management) 시스템에서 제공하는 기능들, 즉 작업 제출, 모니터링, 동기화, 작업 제어하기 등의 작업 스케줄링 기능을 제공한다. Gridway는 작업 스케줄링을 위해 CLI(Command Line Interface)와 OGF 표준 방식인 DRMAA(Distributed Resource Management Application API)를 모두 제공한다.

Gridway는 서로 다른 그리드 서비스를 연동하기 위해서 미들웨어 접속 드라이버(Middleware Access Drivers)를 사용한다. 기본적으로 다음의 MAD가 제공된다.

- pre-WS GRAM과 WS GRAM 서비스와 연동하는 실행 매니저(Execution Manager)
- MDS2(MDS and GLUE schema)와 MDS4 서비스와 연동하는 정보 매니저(Information Manager)
- GridFTP 서비스와 연동하는 전송 매니저

본 논문에서 개발된 Gridway 백엔드는 Ganga에서 Gridway 백엔드를 사용한 작업이 제출되면 해당 작업의 그리드웨이 작업 템플릿(Gridway Job Template)을 자동으로 만들고 이를 Gridway CLI(Command Line Interface)를 사용해서 Globus 미들웨어로 제출하고, 작업을 모니터링하고, 종료된 작업 결과를 가져오는 방식으로 동작한다.

4. 새로운 Ganga 플러그인

4.1 Gridway 백엔드 플러그인

Gridway 는 오픈 소스 기반의 Globus 미들웨어의 메타 스케줄러로 여러 컴퓨팅 자원을 효율적으로 사용할 수 있게 해 준다. Globus 미들웨어는 gLite 미들웨어와는 달리 RB(Resource Broker) 컴포넌트를 따로 제공하지 않기 때문에, Gridway를 사용함으로써 Gridway 백엔드도 LCG 백엔드와 비슷한 계층 구조를 가지도록 하였다.

Ganga에서 백엔드는 사용자가 지정한 응용 프로그램이 실제로 돌아가는 시스템으로 로컬호스트, 배치 시스템, 그리드 등이 될 수 있다. 백엔드는 사용자를 대신해서 그리드에 작업을 제출하고, 상태를 모니터링 하고, 실행 결과를 가져오는 일을 수행한다. gLite 미들웨어 그리드에 대한 LCG 백엔드의 예를 들어보면, gLite 미들웨어 그리드에 작업을 제출하기 위해서는 먼저 해당 작업에 대한 JDL(Job Description Language) 파일을 작성하고, glite-wms-job-submit 명령을 사용해서 해당 작업을 그리드에 제출한다. 그리고 난 후 glite-wms-job-status 명령을 사용해서 해당 작업의 상태를 확인한다. 작업이 완료 되었으면, glite-wms-job-output 명령을 사용해서 작업의 실행결과를 사용자의 로컬 시스템으로 가져온다. LCG 백엔드는 Ganga의 응용 프로그램 블록에 지정된 정보를 사용해서 JDL파일을 자동으로 만들고, 이와 같은 일련의 작업들을 자동적으로 수행해 주는 역할을 한다.

Ganga에서 새로운 백엔드를 만들기 위해서는 Ganga에서 제공하는 IBackend라는 클래스를 상속해서 구현해야 한다. 이 클래스를 상속받아 필수적으로 구현해야 하는 함수로는 submit(), resubmit(), kill(), updateMonitoringInformation() 등이 있다.

Gridway 백엔드는 id, status, actualCE라는 스키마로 구성되어 있다. id는 Ganga에서 관리하는 job id와는 다른 것이다. 즉, gwsubmit 명령어로 Gridway로 작업을 제출할 때 Gridway 자체에서 관리되는 작업 id이다. Gridway는 작업 id를 0부터 시작하는 정수로 관리한다. 그러므로 Gridway 백엔드의 id에는 0, 1, 2, ...와 같은 값이 들어가게 된다. status에는 현재 작업의 상태 정보가 저장된다. Gridway는 Pending, Hold, Prolog, Pre-wrapper, Wrapper, Epilog, Migrate, Stopped, Failed, Done 등의 상태 정보를 제공하는데, 각 상태가 바뀔 때마다 status 정보에 상태 정보가 저장된다.

submit() 함수는 크게 두 가지 문맥으로 구성된다. 첫째, 작업에 대한 준비를 하고 job template 파일의 경로를 리턴하는 것이다. 작업을 준비하는 것은 우선 사용자가 명시한 input 파일들을 압축파일로 만들고, 프로그램의 이름과 인자의 이름들도 얻어서 job template 파일을 만드는 것이다. job template 파일은 Gridway에게 작업을 던지기 위한 작업 명세 파일이다. 여기에 실행 파일 이름, 인자, 인풋 파일, 아웃풋 파일, 환경변수 등을 이용하여 job template 형식에 맞게 바꿔 준 다음, 리턴하게 되는 것이다. 여기에서 실행 프로그램은 사용자가 명시한 실제 프로그램이 아니라 Ganga wrapper가 된다. Ganga wrapper 안에서 사용자가 명시한 프로그램을 실행시키고, wrapper는 그 프로그램 실행을 관리하는 또 다른 일을 하게 되는 것이다. 두 번째 문맥은 앞서서 리턴받은 job template 파일을 가지고 Gridway

에게 실제로 작업을 요청하는 과정이다. 실제로 명령어를 실행하기 위해서는 "gwsubmit -t <template file>" 형태의 커맨드 라인 명령어를 사용해야 한다. submit_jt()라는 함수가 이 과정을 수행하게 된다. 그리고 gwsubmit 명령어로 작업을 제출하고 나서 gwps 명령어로 작업의 상태를 파악한 후, 현재 작업의 상태를 리턴하게 되는 것이다. 이 두 가지 문맥이 실행됨으로써 submit() 함수의 역할을 다 할 수 있는 것이다.

resubmit() 함수는 작업을 제출하는 과정에서 문제가 있어서 Ganga 작업 상태가 new로 되어 있을 때, 다시 작업 제출을 요청하면 실행되는 함수이다. 그러므로 이미 작업에 대한 정보를 가지고 있으므로 submit() 함수가 하는 일과 비슷한 일을 하게 구현하였다.

kill() 함수는 Ganga에서 작업을 제거하는 명령을 실행했을 때 호출되는 함수이다. Ganga에서 작업을 제거하기 위해서는 remove() 명령어를 사용한다. remove() 명령어를 통해 kill()함수가 호출되면 Gridway의 작업을 제거하는 gwkill 커맨드를 호출해야 한다. 작업 자체에 Gridway의 작업 id가 들어 있기 때문에 gwkill 커맨드의 인자로 이 id를 넘겨주면 된다. 그리고 gwkill 명령이 성공적으로 끝났는지, 아니면 오류가 나서 제대로 끝나지 못했는지에 대한 상태 정보를 리턴하게 된다.

마지막으로 updateMonitoringInformation() 함수는 작업의 상태가 어떻게 바뀌어 가는 지 체크하면서 상태가 바뀔 때마다 상태 정보를 업데이트 하는 일을 하는 함수이다. 이 함수는 Ganga에서 주기적으로 실행되면서 상태 정보를 업데이트 하도록 되어 있다. Gridway에서는 작업들의 상태 정보를 알아보기 위해 "gwps"이라는 커맨드 라인 명령어를 제공한다. 이 명령어를 실행하면 Gridway로 제출된 모든 작업들의 상태 정보를 확인할 수 있다. 이 결과를 파싱하여 Ganga의 job id와 맞는 작업의 실제 실행 호스트와 상태 정보를 업데이트 하는 것이다. 그리고 작업이 완료 되었다면 사용자가 지정한 아웃풋 파일들을 Ganga의 워킹 디렉토리의 output 디렉토리로 옮겨오는 일도 해야 한다. updateMonitorInformation은 인스턴스를 만들지 않아도 실행 가능한 static 함수로 선언해야 하므로 Gridway 백엔드 클래스에 updateMonitoringInformation = staticmethod(updateMonitoringInformation)이라는 문구를 추가해 주어야 한다. 그림 3에서는 Gridway 백엔드에서 상태 정보를 업데이트 하기 위해 사용된 gwps 명령의 출력 예제를 보여주고 있으며, 그림 4에서는 Gridway 백엔드에 제출된 작업이 처리되는 과정을 보여준다.

4.2 InterGrid 백엔드 플러그인

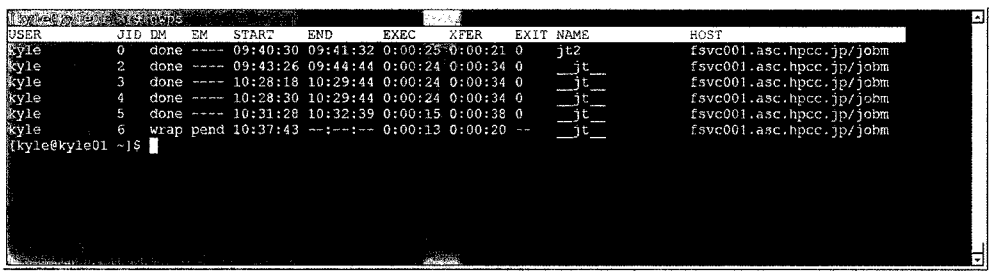

그림 3 Gridway에 제출된 작업들의 상태를 확인하는 gwps 명령

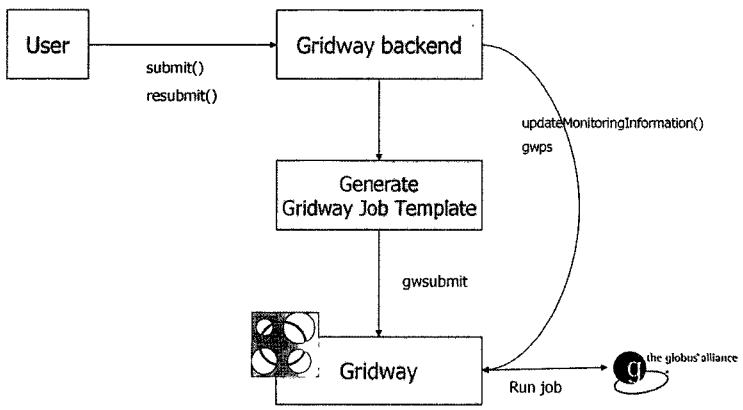

그림 4 Gridway 백엔드에 제출된 작업이 처리되는 과정

Gridway 백엔드와 마찬가지로 gLite 그리드 자원과 Globus 그리드 자원을 함께 사용하기 위해서 만들어진 InterGrid 백엔드도 역시 IBackend 인터페이스를 상속받아서 구현하였다. Gridway 백엔드와 마찬가지로 여기에서도 필수적으로 submit(), resubmit(), kill(), updateMonitoringInformation() 등의 함수를 구현해야 한다. 우선 InterGrid 백엔드의 스키마에 대해서 살펴본 후, 각각의 함수에 대한 동작을 살펴보도록 하겠다.

우선 InterGrid 백엔드는 id, status, actualCE, actualBackend, middleware, targetBackends 등을 속성으로 하는 스키마로 구성되어 있다. id는 InterGrid 백엔드가 관리하는 각 백엔드에서 관리하는 작업의 id이다. 예를 들어, LCG(LHC Computing Grid) 백엔드에서 관리하는 id는 URL을 포함한 문자열이 될 것이고, Gridway 백엔드에서 관리하는 id는 0부터 시작하는 정수가 될 것이다. 각 백엔드에서 리턴한 작업의 id값이 InterGrid 백엔드의 id 필드에 저장되는 것이다. status 에도 id와 마찬가지로 InterGrid가 관리하는 각 백엔드의 작업에 대한 상태 정보가 저장된다. 각 백엔드가 상태 정보를 나타내는 방식이 약간씩 다를 수 있지만, InterGrid는 actualBackend라는 필드에 실제 어떤 백엔드에 작업을 던졌는지에 대한 정보도 저장하므로 이것을 이용하면 된다. actualCE는 실제로 작업이 수행되는 워커 노드가 어디인지를 알려주고, 작업이 실행되는 작업 매니저(Fork, PBS, Condor 등)가 무엇인지도 알려주게 된다. 이것도 역시 각각의 백엔드에서 리턴한 정보를 가지고 알 수 있다. actualBackend는 InterGrid로 작업을 던졌을 때 실제로 여러 백엔드 중에서 어느 곳으로 작업이 제출되었는지를 저장하는 필드이다. middleware는 LCG 백엔드를 위해 존재하는 속성인데, LCG 백엔드에는 EDG 미들웨어와 gLite 미들웨어 2개가 존재한다. 그러므로 LCG 백엔드라는 정보만 가지고서는 작업을 관리하기가 힘들다. 그러므로 이 필드에 EDG 또는 gLite라는 정보를 넣어서 LCG 백엔드로 제출된 작업을 관리하는데 이용하게 된다. targetBackends는 InterGrid 백엔드에서 관리할 백엔드들을 사용자로 하여금 정할 수 있게 하는 것이다. 기본 값으로 LCG와 Grdiway가 설정되어 있다.

InterGrid 백엔드는 기본적으로 targetBackends 들에 구현되어 있는 함수들을 그대로 이용하는 wrapper역할을 하는 백엔드라고 생각할 수 있다. 우선 submit() 함수의 구현에 대해 살펴보도록 하자. 우선 InterGrid는 여러 개의 백엔드 중에서 한군데로만 작업을 제출하게 되는데, 이 한군데의 백엔드를 고르는 데 기준이 있어야 한다. 이번 연구에서는 targetBackends에 지정되어 있는 각 그리드의 작업 로드를 보고, 로드가 적은 쪽으로 작업을 보내도록 하였다. 구체적인 구현은 다음과 같다.

Gridway에서는 Gridway가 관리하는 각 노드의 정보를 볼 수 있는 명령어로 "gwhost"라는 명령어를 제공하고 있다. 그러므로 InterGrid 백엔드는 이 명령어를 사용하여 Gridway 백엔드의 작업 로드를 파악할 수가 있다. gwhost 명령은 각 노드의 운영체제, 아키텍처, CPU 클럭, 메모리, 디스크, LRMS(Local Resource Management System)등에 대한 정보를 알려 준다. 다음 그림 5는 gwhost 명령을 실행시킨 출력 예를 보여주는 그림이다.

gwhost 명령을 실행해서 얻을 수 있는 정보 중에서 그리드의 작업 로드를 알 수 있는 것은 CPU에 대한 정보이다. 즉, 전체 CPU 슬롯 수와 사용 중인 CPU 슬롯 수를 알려주므로 그 정보를 사용해서 Gridway 백엔드의 작업 로드를 구해낼 수 있다. InterGrid 백엔드에서는 이렇게 해서 Gridway 백엔드의 작업 로드를 구해낼 수 있다.

LCG 백엔드의 작업 로드를 구해내는 것도 Gridway 백엔드에서 이용한 방법과 비슷하다. LCG 백엔드에서는 해당 VO(Virtual Organization)에서 사용할 수 있는 노드들에 대한 정보를 출력하는 명령어로 lcg-infosites라는 명령어를 제공한다. 이 명령어를 이용해서 CE (Computing Element) 리스트를 출력하면 총 CPU 슬롯 수, 총 작업 개수, 실행 중인 작업 개수, 대기 중인 작업 개수, CE 주소 등을 알 수 있다. 다음 그림 6은 lcg-infosites 명령어를 통해서 FKPPL VO(France Korea Particle Physics Laboratory Virtual Organization)에 있는 CE 들의 정보를 출력한 결과를 보여준다.

이 명령어를 사용하면 Gridway 백엔드에서와 마찬가

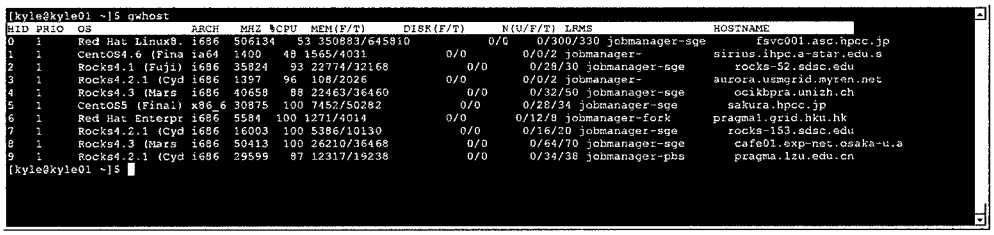

그림 5 현재 Gridway에 접속된 컴퓨팅 노드들의 정보

```
kyle@kyle01 ~]$ lcg-infosites --vo fkppl.kisti.re.kr ce
CPU    Free    Total Jobs      Running Waiting ComputingElement
----------------------------------------------------------
228     228       0              0       0     cclcgceli03.in2p3.fr:2119/jobmanager-bqs-short
228     228       0              0       0     cclcgceli04.in2p3.fr:2119/jobmanager-bqs-short
1920    1878      42             30      12    cclcgceli03.in2p3.fr:2119/jobmanager-bqs-medium
3403    3340      63             41      22    cclcgceli04.in2p3.fr:2119/jobmanager-bqs-long
1920    1873      47             38      9     cclcgceli04.in2p3.fr:2119/jobmanager-bqs-medium
3403    3345      58             43      15    cclcgceli03.in2p3.fr:2119/jobmanager-bqs-long
kyle@kyle01 ~]$
```

그림 6 FKPPL VO의 컴퓨팅 노드(Computing Element)들의 정보

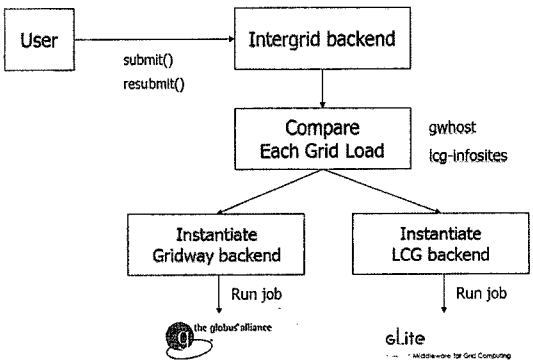

그림 7 InterGrid 백엔드에 제출된 작업이 처리되는 과정

지로 전체 CPU 슬롯 대비 사용 중인 CPU 슬롯의 비율로서 그리드 전체의 작업 로드를 구해낼 수 있다. InterGrid 백엔드에서는 이런 식으로 targetBackends에 있는 백엔드들의 작업 로드를 구하여 작업 로드가 적은 그리드로 작업을 제출한다.

resubmit(), kill(), updateMonitoring() 함수 내부에서는 targetBackends에 있는 클래스들의 각 인스턴스를 만들어서 해당 백엔드의 함수를 호출하고, 그 결과를 그대로 사용하게 된다. 그러므로 InterGrid 백엔드의 핵심은 submit() 함수의 구현에서 각 그리드의 작업 로드를 보고 작업을 제출할 그리드를 판단하는 과정이라고 할 수 있다.

4.3 Autodock 어플리케이션 플러그인

AutoDock[8]은 자동화된 도킹(docking) 도구로 기질이나 약품 후보군과 같은 작은 분자들이 어떻게 3차원 구조의 수용체와 바인딩 되는지를 예측해 보기 위해 디자인되었다. AutoDock은 리간드(ligand)들을 타겟 단백질(target protein)과 도킹(docking)하는 작업을 수행한다. AutoDock은 매우 빠르고, 높은 품질의 리간드 결합도 예측을 제공한다. 그리고 예측된 값과 실제 실험값이 좋은 상관관계를 나타낸다. 또한, AutoDock은 무료 소프트웨어이기 때문에 사용이 용이하다.

본 논문에서는 WISDOM 프로젝트[9]에서 사용되고 있는 AutoDock 프로그램의 스크립트 파일[10]을 사용하였다. 일단 이 스크립트의 동작을 살펴보면 다음과 같다.

먼저 AutoDock 작업에 필요한 파일들을 globus-url-copy 명령어를 사용해서 현재 작업 노드로 가지고 온다. AutoDock 실행 프로그램을 담고 있는 압축 파일과 AutoDock의 도킹(docking) 알고리즘에 대한 파라미터 파일인 dpf3gen.awk 파일, 그리고 단백질에 대한 정보를 담고 있는 압축 파일과 ligand 파일을 스토리지 노드로부터 가지고 오는 것이다.

그 다음에 가져온 압축 파일들의 압축을 푼다. 그리고 나서 스택 사이즈의 제한을 풀어주는 "ulimit -s unlimited"라는 명령어를 실행한다. 이것은 AutoDock 프로그램을 실행하는데 필요한 과정인데, 이는 AutoDock 프로그램 자체가 스택을 정적으로 사용하기 때문이다. 또한 AUTODOCK_UTI라는 환경 변수를 설정해 주어야 한다. 스크립트 안에서는 이 환경 변수를 현재 디렉토리로 설정하고 있다.

그 다음에는 단백질 파일과 리간드 파일로 dpf 파일을 만들어 내야 한다. 이것은 mkdpf3라는 프로그램을 이용해서 수행된다. 마지막으로 AutoDock 프로그램이 수행된다. 앞서 만든 dpf 파일을 인풋으로 넣어 주면 dlg 파일이 아웃풋으로 생성된다. 이것으로서 AutoDock 작업을 마치게 되는 것이다.

본 논문에서는 이 스크립트를 그대로 활용하면서 사용자가 원하는 파라미터를 Ganga에서 임의로 바꾸기 용이하도록 구현하였다. 스크립트에서 사용자가 설정할 수 있는 부분은 스크립트 이름, 단백질, 리간드, 파라미터 파일, 바이너리 파일 이름이다. Ganga의 Autodock 어플리케이션 클래스에서는 이것들을 스키마로 정의한다. 즉, script, protein, ligand, parameter, binary라는 스키마를 정의하여 이 부분을 사용자가 정의하면 그대로 AutoDock 작업이 수행되는 것이다.

LCG 환경과 Globus 환경을 통합하는 과정에서 스크립트 안에 있는 globus-url-copy명령을 Gridway 이용한 Globus 그리드 환경에서는 사용할 수 없다는 것을 이번 연구를 진행하면서 발견하게 되었다. 그래서 Ganga

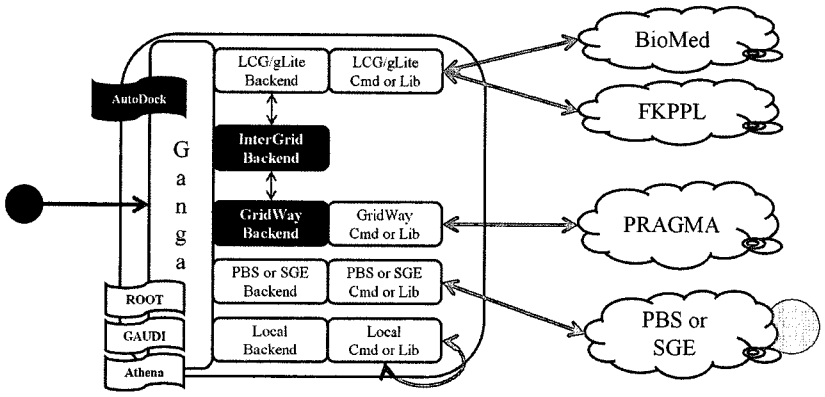

그림 8 Ganga 백엔드 플러그인과 전체 그리드 시스템 아키텍처

를 사용하는 클라이언트 노드에 사용자가 원하는 모든 파일을 두고, Autodock 어플리케이션을 정의하면 클라이언트 노드에 있는 파일들을 실행 노드로 스테이징(staging) 하는 구조로 구현하였다.

5. EGEE와 PRAGMA 자원 연동실험

이번 장에서는 앞 장에서 설명한 Gridway, InterGrid, Autodock 플러그인 모듈을 사용하여 Globus 그리드 환경의 PRAGMA[11]와 gLite 그리드 환경의 EGEE[12]를 함께 연동하는 방법과 실험 시 발생했던 문제점 및 해결방법에 대해서 알아본다.

5.1 scmsweb를 이용한 PRAGMA 자원 연동

PRAGMA 그리드는 클러스터 모니터링 툴로 scms [13]와 scmsweb[14]을 사용하고 있어서, Gridway와 PRAGMA 자원을 연동하기 위해서 Thai National Grid Center에서 개발한 scmsweb-gridway 패키지가 제공하는 MAD(Middleware Access Drivers)를 사용하였다. 이 MAD 프로그램은 scmsweb 패키지에 포함된 파이썬 스크립트를 사용해서 PRAMGA 모니터링 사이트 [15]에서 Gridway에 필요한 정보(운영체제, 시스템 아키텍처, CPU 사용량, 메모리, 디스크 사용량, 현재 사용중인 CPU개수, 사용 가능한 CPU개수, 전체 CPU개수, LRMS(Local Resource Management System) 종류, 호스트 이름)를 가져온다.

5.2 이기종 미들웨어 연동시 문제점 및 해결방법

5.2.1 인증(Authentication)

그리드 환경에 작업을 제출하기 위해서는 우선 인증서를 사용한 인증과정을 거쳐 프록시(proxy)를 생성해야 한다. Globus경우, "grid-proxy-init" 명령을 사용하여 이 과정을 진행한다. gLite 미들웨어에서는 VOMS (Virtual Organization Membership Service)를 통해서 사용자 인증을 하며 "voms-proxy-init" 명령을 사용한다. Globus가 사용하는 기본 인증에 VO(Virtual Organization)에 대한 추가적인 정보를 담고 있는 것이 VOMS 인증이므로, VOMS 인증을 Globus 인증의 슈퍼셋(super set)으로 볼 수 있다. 실험 결과 "voms-proxy-init" 명령으로 프록시를 생성한 후 PRAGMA 환경과 EGEE 환경 모두로 작업 제출이 가능함을 확인할 수 있었다.

5.2.2 파일 스테이징(File Staging) 문제점

4.3절에서 언급했던 대로 기존 AutoDock 스크립트는 필요한 파일들을 저장 노드(Storage Element)로부터 globus-url-copy 커맨드 툴로 받아오는 형태로 되어 있었다. 하지만 Gridway 백엔드에서는 스크립트의 globus-url-copy 명령이 동작하지 않는다는 것을 발견하게 되었다. 이유를 분석해 본 결과 Gridway가 작업을 실제 CE(Computing Element)로 제출할 때, 인증서(credential)를 전달하지 않는다는 것을 알게 되었다. 그래서 불가피하게 스크립트를 수정할 수밖에 없었다.

그래서 사용자가 프로그램 실행에 필요한 모든 파일들을 클라이언트 노드에 가지고 있고, 작업을 제출할 때 파일을 스테이징(Staging) 하는 구조로 전환하게 되었다. 하지만 이렇게 구조를 바꾸는 데도 문제가 있었다. LCG 백엔드의 gLite 미들웨어에는 인풋 샌드박스(input sandbox) 크기 제한이 있기 때문이었다. AutoDock 작업에 필요한 파일이 인풋 샌드박스의 용량을 초과하는 경우에는 gLite의 인풋 샌드박스를 그대로 사용할 수 없었다. 반면에, Gridway는 인풋 샌드박스에 용량 제한이 없었다.

하지만, Ganga가 원래 LCG 백엔드에 특화되어 있는 유저 인터페이스이기 때문에 이 문제에 대한 해결책을 가지고 있었다. Ganga를 통해서 LCG 그리드에 작업을

제출할 때, 사용자가 입력한 파일이 인풋 샌드박스 용량을 초과하게 되면, Ganga는 미리 지정된 저장 노드(SE)에 파일을 올려놓고, CE로 가서 다시 그 파일을 다운로드 받아서 사용하게끔 되어 있었다. 결국 Ganga의 이러한 기능으로 인해 파일을 전송하는 문제를 해결할 수 있었다. 즉, 사용자가 프로그램 실행에 필요한 파일들을 명시해 주면, LCG 백엔드, Gridway 백엔드, InterGrid 백엔드 모두에서 실행에 필요한 파일을 CE로 보내서 실행하게 되는 것이다.

5.2.3 Ganga에서 작업 실행하기

Ganga는 응용프로그램 계층과 백엔드가 분리된 구조로 되어 있기 때문에 서로 다른 그리드로 작업을 제출하기 위해서는 Job 클래스의 backend 필드만 해당 그리드에 맞게 설정해 주면 된다.

다음은 'Hello World'를 출력하는 간단한 작업을 수행하는 Ganga 코드이다.

```
j = Job(application=Executable(exe='/bin/echo',args=['Hello World']))
```

백엔드를 지정하지 않고, 작업을 제출(submit)하면 기본 설정인 로컬(Local)로 작업이 제출되어서 실행된다.

다음과 같이 LCG 백엔드를 지정해 주면 작업은 gLite의 미들웨어를 사용하는 EGEE 환경으로 제출되어 실행된다.

```
j.backend=LCG(middleware="GLITE")
```

마찬가지로 Gridway 백엔드를 통해서 Globus 그리드로 작업을 제출하고 싶으면 다음과 같이 백엔드를 지정해 주면 된다.

```
j.backend=Gridway()
```

백엔드를 InterGrid로 지정하고, 선택 가능한 타켓 백엔드를 LCG와 Gridway로 다음과 같이 지정해 주고, 작업을 제출하면 InterGrid 백엔드는 각 백엔드의 작업 로드를 구하고, 작업 로드가 적은 그리드로 작업을 제출한다.

```
j.backend=InterGrid()
j.backend.targetBackends=["LCG","Gridway"]
```

그림 9는 Autodock 응용프로그램에 대한 작업을 정의하고, 백엔드를 InterGrid로 설정한 Ganga 스크립트 파일과 이 스크립트를 Ganga의 execfile 명령을 사용해서 실행하는 화면을 보여준다. 여기서는 Globus 그리드의 작업 로드가 EGEE 그리드의 로드보다 낮아서 실제 작업은 Gridway 백엔드를 통해서 Globus 그리드로 보내졌음을 알 수 있다.

6. 결론 및 향후 연구과제

이상과 같이 다양한 그리드 환경을 동일한 사용자 인터페이스로 사용할 수 있도록 지원해 주기 위해서, Globus 기반의 그리드 자원을 사용하기 위한 Gridway 플러그인과 Globus 와 gLite 그리드 자원을 함께 사용하기 위한 InterGrid 플러그인, 그리고 이를 활용한 Autodock application 플러그인의 설계 및 구현에 대해서 알아보았다. 현재 PRAGMA 그리드 자원과 EGEE 그리드 자원의 연동 실험을 마친 상태이며, 개발된 플러그인들은 Ganga를 개발한 CERN과의 국제공동연구를 통하여 Ganga 5.3.2 배포판에 포함되었다[16].

이번 연구에서는 InterGrid 백엔드에서 실제로 작업을 제출할 그리드를 선택하는 기준으로 그리드의 작업 로

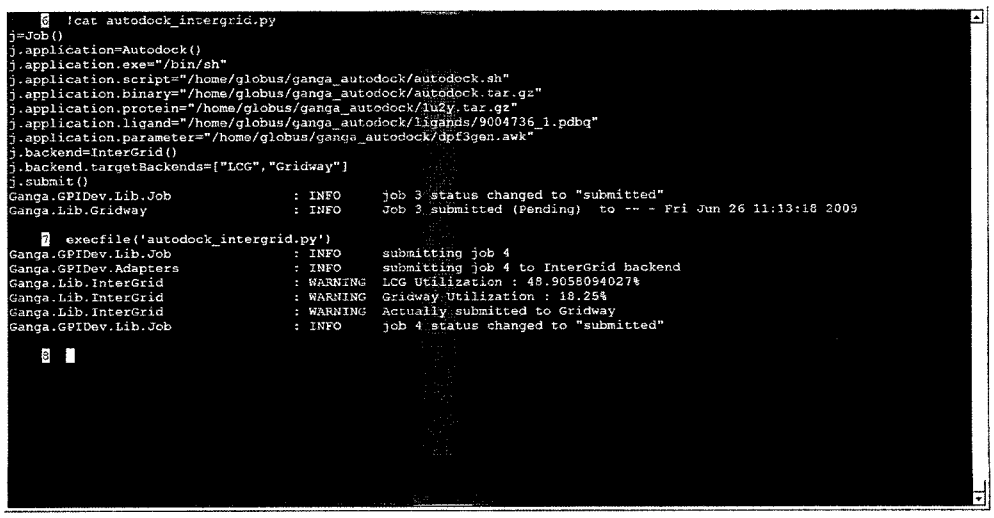

그림 9 InterGrid 백엔드를 사용한 Autodock 어플리케이션의 작업 제출 화면

드만을 사용하였으나, 작업의 특성(계산량 위주/데이터 처리위주)이나 사용자의 설정에 따라 그리드 선택 기준을 여러 가지 기준으로 선택 할 수 있도록 해주는 것을 향후 연구과제로 고려하고 있다.

참 고 문 헌

[1] Ganga, http://ganga.web.cern.ch/ganga/

[2] Gridway Metascheduler, http://www.gridway.org/doku.php

[3] R. Boardman, S. Crouch, H. Mills, S. Newhouse, J. Papay, "Towards Grid Interoperability," *Proceedings of the UK e-Science All Hands Meeting 2007*, Nottingham, UK, 10th - 13th September 2007, pp.127-134

[4] Peter Kacsuka, Tamas Kissb, Gergely Sipos, "Solving the grid interoperability problem by P-GRADE portal at workflow level," *Future Generation Computer Systems*, vol.24, Issue 7, pp.744-751, July 2008.

[5] J.T.Mościcki, F.Brochu, J.Ebke, U.Egede, J.Elmsheuser, K.Harrison, R.W.L.Jones, H.C.Lee, D.Liko, A.Maier, A.Muraru, G.N.Patrick, K.Pajchel, W.Reece, B.H.Samset, M.W.Slater, A.Soroko, C.L.Tan, D.C. Vanderster, M.Williams, "Ganga: a tool for computational-task management and easy access to Grid resources," Computer Physics Communications (submitted)

[6] Eduardo Huedo, Rubén S. Montero and Ignacio M. Llorente, "The GridWay Framework for Adaptive Scheduling and Execution on Grids," Scalable Computing - Practice and Experience 6 (3): 1-8, 2005.

[7] E. Huedo, R. S. Montero, and I. M. Llorente, "A modular meta-scheduling architecture for interfacing with pre-ws and ws grid resource management services," Future Generation Computer Systems, vol.23, no.2, pp.252-261, 2007.

[8] AutoDock, http://autodock.scripps.edu/

[9] Wide In Silico Docking On Malaria, http://wisdom.eu-egee.fr/

[10] WISDOM Production Environment, http://sourceforge.net/projects/wisdom-pe/

[11] PRAGMA, http://www.pragma-grid.net/

[12] EGEE, http://www.eu-egee.org/

[13] SCMS, http://goc.pragma-grid.net/wiki/index.php/SCMS

[14] SCMSWeb, http://research.thaigrid.or.th/en/SCMSWeb

[15] SCMSWeb Monitoring Tool, http://pragma-goc.rocksclusters.org/scmsweb/

[16] http://ganga.web.cern.ch/ganga/release/5.3.2/release/ReleaseNotes-5.3.2

김 한 기

2001년 서강대학교 전산학과(학사). 2003년 KAIST 전산학과(공학석사). 2003년~2004년 LG전자. 2004년~현재 한국과학기술정보연구원. 관심분야 그리드 컴퓨팅, 데이터 마이닝, 정보 검색, 분산 알고리즘

황 순 욱

1990년 서울대학교 수학과 졸업(학사) 1995년 서울대학교 계산통계학과 졸업(석사). 2003년 미국 남가주대학교 전산과학과 졸업(박사). 2003년~2006년 일본 국립정보학연구소 연구원. 2006년~현재 한국과학기술정보연구원 책임연구원. 관심분야는 그리드 컴퓨팅, 워크플로우 시스템 등

이 윤 기

2007년 한양대학교 컴퓨터교육과 학사 2009년 서울대학교 컴퓨터공학부 석사 2009년~현재 파수닷컴(병역특례). 관심분야는 그리드 컴퓨팅, 분산시스템

김 은 성

정보과학회논문지 : 컴퓨팅의 실제 및 레터 제 15 권 제 8 호 참조

염 헌 영

정보과학회논문지 : 컴퓨팅의 실제 및 레터 제 15 권 제 8 호 참조