계층형 구조를 기반으로 한 모니터링 시스템
(A Monitoring System based on Layered Architecture)

권 성 주 † 최 재 영 † 이 지 수 ††
(Sungju Kwon) (Jaeyoung Choi) (Jisoo Lee)

요 약 그리드 컴퓨팅과 같은 분산 시스템은 하드웨어와 소프트웨어 구성 요소들이 다양하고 복잡하며 분산 배치되어 있어 전체적인 관리가 어렵다. 그리드나 유비쿼터스처럼 환경적인 요소가 다양하며 유동적으로 변경될 가능성이 많은 시스템은 실시간으로 자원에 대한 정보를 모니터링하고 그에 따라 반응할 수 있는 메커니즘을 제공해야 한다. 모니터링 어플리케이션의 개발시 시스템 자원의 다양성은 정보를 수집하는 에이전트의 개발에 대한 비용 부담을 증가시킨다. 본 논문에서는 다양한 시스템 환경에서 추가적인 에이전트 개발의 부담을 최소화시키며 환경적인 변화에 능동적으로 대처할 수 있고 통신 프로토콜과 사용자 질의간의 의존성을 없애주는 계층형 구조를 기반으로 구현된 모니터링 시스템을 제시한다. 본 논문의 계층형 구조는 통신, 질의 언어, 모니터링 에이전트의 분리를 통해 분산 환경에서 모니터링 시스템의 구성 및 변경을 용이하게 한다.

키워드 : 모니터링 시스템, 에이전트, 그리드, SOA

Abstract Grid computing is the complex deployments of various hardware and software components. The Grid environment should provide a mechanism for real-time monitoring and notification. It is very important to implement a monitoring mechanism in the Grid environment. Most existing monitoring systems only focus on their own requirements. With the development of Grid computing technology, the extensible monitoring systems become more and more feasible and popular. In this paper, we describe our research and development works on M-Mon, a novel framework for the flexible and adaptive Grid monitoring system. M-Mon system focuses on some critical issues like scalability, reusability, runtime extensibility, protocol transparency and uniform data representation. To provide interoperability with other monitoring systems and to reuse legacy facilities with a minimum effort, our monitoring system has been developed using service-oriented architecture.

Key words : Monitoring System, Agent, Grid, SOA

1. 서 론

그리드 시스템[1,2-4]과 같이 컴퓨팅 자원이 대규모화 및 복잡화되는 환경은 전체 자원의 관리에 대한 비용이 증가한다. 고정된 환경으로 구성되는 기존의 단일 시스템이나 클러스터의 경우에는 미리 정의된 자원에 대한 모니터링만이 필요했다. 하지만, 지리적으로 분산 배치되어 있는 하드웨어와 그 위에서 수행되는 소프트웨어 구성 요소들의 다양성은 전체 시스템의 관리를 복잡하게 만든다. 이러한 환경에서는 하드웨어나 소프트웨어 구성 요소들이 언제든지 바뀔 수 있으며 오류의 발생 가능성도 많다. 따라서, 그리드나 유비쿼터스 환경과 같은 이기종 분산 시스템에서는 다양한 종류의 자원이 유동적으로 변화하는 것에 대해 능동적으로 대처할 수 있는 메커니즘이 필요하다. 모니터링 시스템의 목적은 특정 시점에서의 자원을 측정하고 상태를 나타내는 것이다. 단일 호스트나 클러스터 환경에서 사용하던 모니터링 메커니즘은 다음과 같은 이유 때문에 그리드와 같은 분산 시스템에서 적합하지 않다.

첫째, 단일 호스트 시스템의 경우 모니터링하는 자원의 종류가 한정되어 있다. 하지만, 분산 시스템의 경우 서로 다른 구조를 가진 여러 시스템들이 엮여 있어서 자원의 변화가 많이 발생한다. 그러므로 분산 시스템을 위한 모니터링 어플리케이션은 자원의 변화에 대해 능

· 본 논문은 과학기술부의 국가 e-Science 프로젝트의 지원으로 연구되었습니다.
† 학생회원 : 숭실대학교 컴퓨터학과
lithlife@ssu.ac.kr
†† 종신회원 : 숭실대학교 정보과학대학 컴퓨터학부 교수
choi@ssu.ac.kr
††† 정 회 원 : 한국과학기술정보연구원, 슈퍼컴퓨팅센터장
jysoo@kisti.re.kr
논문접수 : 2006년 1월 15일
심사완료 : 2006년 4월 7일

동적으로 대처할 수 있어야 하며, 서로 다른 형태로 제공되는 자원 정보들을 일관된 형태로 표현할 수 있어야 한다. 둘째, 단일 호스트나 클러스터를 위한 모니터링 어플리케이션의 경우 고정된 통신 프로토콜을 사용하고 미리 정의된 시스템 구조만을 지원한다. 그러나, 분산 시스템에서 각 호스트들은 환경적인 요인에 의해서 사용할 수 있는 통신 프로토콜들이 바뀔 수 있다. 따라서 한가지 이상의 통신 프로토콜 방식을 제공하여 자원 모니터링 요청을 위해 시스템에 접근하는 방법을 사용자가 선택할 수 있도록 통신 프로토콜과 실제 정보 수집 부분간의 분리가 필요하다. 셋째, 대부분의 단일 시스템을 위한 모니터링 어플리케이션들은 정보 수집을 위해 자체적인 정보 수집 에이전트들을 개발하므로서 기존에 나와 있는 기능들을 재사용할 수 있는 방법을 제공하지 않는다. 비록 그러한 방법을 제공하는 경우에도 내부적으로 미리 정한 형태만을 사용할 수 있다. 대규모 자원들로 이루어진 분산 시스템에서는 모든 종류의 자원에 대해 직접 정보 수집을 위한 에이전트를 만드는 것이 불가능하기 때문에 기존 모니터링 기능들을 사용자의 설정에 따라 재사용할 수 있는 방법이 제공되어야 한다. 넷째, 대부분의 모니터링 어플리케이션들은 정보 수집을 위한 질의 언어가 따로 존재하지 않으며, 통신 프로토콜과 직접적으로 묶여 있는 경우가 많다. 질의 언어가 지원되는 경우일지라도 미리 고정된 형태의 질의만이 가능하다. 하지만 다양한 사용자 및 어플리케이션의 요구에 따라 모니터링 정보를 제공해야 하는 분산 시스템에서는 질의 언어와 통신 프로토콜을 분리할 필요가 있으며 다양한 질의 언어를 설치 및 선택할 수 있어야 한다.

본 논문에서는 환경적인 변화가 빈번하며 제약사항이 다양한 분산 시스템에서 효과적으로 구성요소들을 선택할 수 있으며 최소한의 개발비용으로 사용자의 요구사항을 만족시킬 수 있는 모니터링 시스템을 제시한다. 통신 프로토콜, 질의 언어, 자원 수집 에이전트의 동적인 구성이 가능하도록 컴포넌트 기반의 계층형 구조인 MAGE(Modulear and Adaptive Grid Environment)[5,6]를 사용하였다. 제시한 모니터링 시스템의 기반이 되는 계층형 구조를 만들기 위해 개발된 MAGE는 위치 및 구현 언어에 대한 의존성을 최소화시키기 위해서 서비스 지향 구조를 사용한다. 본 논문에서는 모니터링 어플리케이션을 구축하기 위한 기반 환경이 되는 모니터링 시스템을 제공하며, 이 환경을 기반으로 동적으로 모니터링 노드들을 구성할 수 있는 분산 시스템을 위한 모니터링 어플리케이션을 개발하여 제시한다.

본 논문은 다음과 같은 순서로 구성되어 있다. 2장에서는 분산 환경의 모니터링 시스템들과 관련된 연구들을 살펴본다. 3장에서는 본 논문에서 제시하는 모니터링 구조를 살펴보며 4장에서는 기반이 되는 계층형 구조인 MAGE를 설명한다. 5장에서는 기본 서비스위에 구성된 실제 모니터링 어플리케이션을 살펴보며 6장에서는 결론을 맺는다.

2. 관련 연구

모니터링 되는 자원들은 사용자의 요구사항과 적용되는 환경에 따라 다양하다. 이러한 자원에는 소프트웨어, 호스트 하드웨어, 네트워크들이 포함된다. 모니터링 시스템은 확장성, 자원, 상호 운영성, 단일화된 자료 표현, 정보 검색, 실행시 확장성, 자료의 필터링, 공개/표준화된 프로토콜, 보안, 소프트웨어의 가용성 및 의존성 등에 의해 분류할 수 있다[1]. GMA(Grid Monitoring Architecture)[7]는 GGF의 GMA-WG(Grid Monitoring Architecture Working Group)[8]에서 공개 구조로 제시되었다.

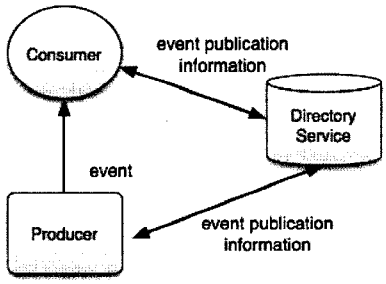

그림 1 GMA의 구조

GMA는 그림 1과 같이 디렉토리 서비스(Directory Service), 생산자(Consumer), 소비자(Producer)의 세 부분으로 구성되어 있다. 디렉토리 서비스는 생산자, 소비자, 이벤트의 발행과 발견을 지원한다. 생산자는 모니터링 자료를 만들어내고 소비자에게 정보를 전달할 수 있다. 소비자는 모니터링 정보를 접근하고 사용할 수 있다. GMA는 모니터링 시스템의 기본 구성 방식을 제시하지만 구현에 관한 세부 사항이 정의되어 있지 않아 명세서를 기반으로 구현된 모니터링 어플리케이션간의 호환성이 결여된다.

JAMM[9,10]은 로렌스 버클리 국립 연구소의 Data Intensive Distributed Computing 그룹에서 개발된 광역 GMA형 시스템이다. 이 시스템은 컴퓨터 호스트의 정보를 수집하고 발행하기 위해 센서들을 사용한다. 클라이언트는 원격 센서들의 실행을 제어할 수 있으며 이벤트의 형태로 모니터링 자료를 받을 수 있다. JAMM은 NetLogger 툴킷[11]을 이용한 프로젝트에서 사용되었다. JAMM은 센서(Sensor), 센서 관리자(SM, Sensor Manager), 이벤트 게이트웨이(Event Gateway), 센서

디렉토리(Sensor Directory), 이벤트 소비자(Event Consumer)로 구성되어 있다. JAMM은 결함 허용성이 있는 시스템을 만들기 위해 센서 자료의 확장성과 복제성을 제공할 수 있는 LDAP 서버를 사용한다. 실행 중에 센서가 업데이트 될 수 있으며 새로운 종류의 센서를 배포할 수 있다. JAMM에서는 이기종 자료를 단일화된 자료 형태로 표현하는 능력을 제공하지 못한다. 하지만 어플리케이션의 출력 결과를 뽑아내서 형식화하여 NetLogger 형태을 만들 수 있도록 파싱 모듈을 수행할 수 있는 메커니즘을 제공한다.

MDS4(Monitoring and Discovery Service)[12]는 글로버스 프로젝트의 일부분으로 개발되었다. 이것은 그리드 단계에서 자원 선택을 위해 주로 사용되는 모니터링 시스템을 위해 개발되었다. MDS4는 자료의 발행, 발견, 구독/통지 등 자료에 접근하는 표준 인터페이스를 사용하며 WS-ResourceProperties, WS-BaseNotification, WS-ServiceGroup 등과 같은 표준 웹 서비스를 사용한다. MDS4 컴포넌트에는 인덱스 서비스, 트리거 서비스, 정보 프로바이더, 클라이언트 컴포넌트인 WebMDS 등과 같은 상위 단계의 서비스들이 포함된다.

R-GMA (Relational Grid Monitoring Architecture)[13,14]는 그리드 정보 및 모니터링 시스템으로서 유럽 DataGrid 프로젝트[15]에서 개발되었다. R-GMA는 GMA명세서에 대한 구현물이며 정보 처리를 위해 관계형 모델을 제공한다. 이것은 호스트, 네트워크, 서비스, 어플리케이션 모니터링 자료 등에 대한 정보 전송 계층을 제공한다. R-GMA는 R-GMA 클라이언트, 에이전트, 생산자, 소비자, 중재자, 레지스트리 등으로 구성된 분산 구조이다. R-GMA는 생산자에서 소비자로 정보를 전달하기 위해 두 가지 접근 방법을 제공한다. 풀(Pull) 모델을 사용할 경우 소비자는 생산자에 대해서 일회성 질의를 요청할 수 있다. 푸쉬(Push) 모델의 경우에 소비자는 자료의 변경에 대한 즉각적인 통지를 위해 스트리밍 형태의 질의를 수행할 수 있다. 생산자는 표준화되고 최적화된 명명 스키마를 사용하여 정보를 발행함으로써 자료에 대해 가상 관계형 표현을 제공할 수 있다. 이것은 클라이언트에게 전체 모니터링 자료가 하나의 가상 데이타베이스로 보여지도록 한다.

현재 많은 시스템들이 현장에서 사용되고 있으며 각각의 필요에 따른 기능을 제공한다. 하지만, 이들 시스템들은 고정된 형태의 모니터링 구조 및 자원 수집 에이전트를 제공하기 때문에 자원의 변화가 많으며 구성요소가 다양한 분산 시스템에서 효율적이지 못하다. 유연하고 적응적인 모니터링 시스템을 만드려면 확장성, 재사용성, 실행시의 추가 용이성, 프로토콜 투명성, 단일화된 자료 표현력 등에 대한 부분을 고려해야 한다.

3. 계층형 프레임워크를 기반으로 한 모니터링 시스템 구조

3.1 모니터링 시스템의 구조

본 논문에서 제시하는 모니터링 시스템은 컴포넌트를 기반으로 GMA 명세서를 따르는 구조로서 생산자, 소비자, 저장소로 구성된다. 그림 2는 본 논문에서 제시하는 모니터링 시스템의 구조를 보여준다. 전체 모니터링 구조는 GMA의 Producer에 해당하는 생산자와 Consumer에 해당하는 소비자, Directory Service에 해당하는 저장소로 구성된다. 저장소는 생산자와 소비자 사이의 메시지 발생에 대한 통지 및 서비스 발견을 위한 기본 정보를 제공한다. 생산자는 소비자에게 모니터링 정보를 제공하기 위해 모니터링 정보 수집 에이전트들을 수행시킨다. 생산자와 저장소는 통신 프로토콜, 질의 언어 처리, 모니터링 프로바이더 등의 변경이 쉽도록 계층형 프레임워크인 MAGE를 기반 구조로 사용한다. 본 모니터링 시스템은 기존 시스템 요소들을 최소의 노력으로 재사용할 수 있게 해주는 모니터링 프로바이더들을 제공한다. 모니터링 시스템은 모니터링 프로바이더와 자원 기술서로 구성되어 있다. 최종 사용자는 모니터링 프로바이더를 통해 수집하고자 하는 자원 정보를 명시하기 위해 자원 기술서를 사용한다.

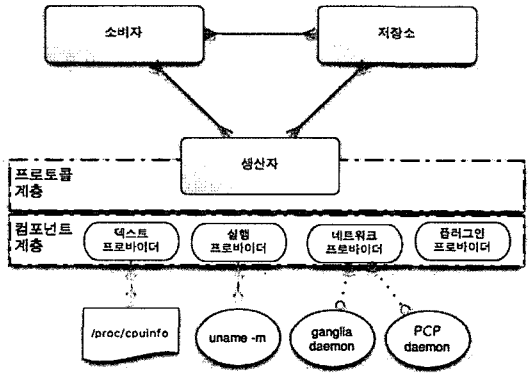

그림 2 모니터링 시스템의 구조

3.2 프로바이더와 서비스의 분리

본 논문의 모니터링 시스템은 그림 2와 같이 텍스트 파일 프로바이더, 실행 프로바이더, 네트워크 프로바이더, 플러그인 프로바이더를 제공한다. 사용자는 필요한 프로바이더를 선택하고 정보를 수집하는 방법을 명시하여 자원을 모니터링 할 수 있다. 그림 3은 자원 기술서 파일의 예를 보여주고 있다.

그림 3은 각 프로바이더를 사용하기 위한 기술서 정보의 예를 보여준다. 프로세스 정보를 제공하는 /proc/

그림 3 자원 기술서 파일 예

그림 4 프로바이더 등록을 위한 XML 문서 예

cpuinfo와 같은 텍스트 파일은 텍스트(text) 프로바이더를 통해 수집된다. 텍스트 프로바이더는 파일 내용을 필터링하여 사용자에게 제공한다. 프로그램의 수행 결과를 수집해야 하는 경우에는 실행(exec) 프로바이더를 사용한다. 실행 프로바이더는 프로그램 수행 후 표준 출력으로 나온 결과를 수집한다. 네트워크 인터페이스를 통해 정보를 제공하는 자원의 경우에는 네트워크(network) 프로바이더를 사용할 수 있다. 네트워크 프로바이더는 소켓 통신을 통해 정보를 제공하는 시스템에 접속하여 자원 정보를 수집한다. 현재 네트워크 프로바이더는 단순한 형태의 프로토콜에 대해서만 사용자가 정보 수집 방법을 명시할 수 있다. 마지막으로 사용자는 플러그인(plugin) 프로바이더를 통해 모니터링 시스템에 자신의 에이전트를 등록할 수 있다. 플러그인 프로바이더가 제공하는 인터페이스를 통해 사용자는 최소한의 코드 작업을 통해 기존 어플리케이션의 기능을 재사용할 수 있다. 플러그인 프로바이더는 등록된 플러그인에 대해서 내부적인 정보 관리의 일을 수행한다.

이 네 가지 형태의 프로바이더 외에도 사용자는 자신의 프로바이더를 작성하여 추가할 수 있다. 그림 4는 프로바이더 등록의 예를 보여준다.

프로바이더 등록을 위한 기술서는 XML 형식으로 작성한다. 소비자 어플리케이션은 프로그램 수행 중에 프로바이더 기술서를 만들어 생산자 어플리케이션에 전달하여 새로운 프로바이더를 자동으로 설치 할 수 있다. 소비자는 호스트들간에 컴포넌트를 옮기거나 제거하도록 지시할 수 있다. 그림 4의 예에서는 컴포넌트 설치를 위한 ComponentInstaller와 각 노드간에 이동하면서 정보를 수집할 수 있는 MovableMonitor 프로바이더의 예를 보여주고 있다. 기술서에는 각 프로바이더에 대한 실행 파일의 정보와 실행시에 프로바이더에게 전달할 인수들 및 제공하는 서비스의 목록을 작성한다.

3.3 통신 프로토콜 컴포넌트의 제어

본 논문의 모니터링 시스템에서 통신은 한개 이상의 통신 프로토콜 프로바이더들을 통해서 처리된다. 통신 프로토콜 프로바이더는 MAGE의 프로토콜 계층에서 처리된다. 소비자는 자료 교환을 위해 자신에게 설치된 프로토콜과 생산자의 통신 프로토콜 중 호환되는 프로토콜을 결정해 메시지를 주고 받는다. 통신 프로토콜은 실제적인 서비스를 제공하는 하위 계층에는 영향을 미치지 않는다. 통신 프로토콜 계층은 각 통신 프로토콜 프로바이더가 사용하는 메시지 형태와 모니터링 시스템 내부적으로 사용하는 메시지 간의 변환을 위한 API를 제공한다.

각 통신 프로토콜 프로바이더는 더욱 추상화된 형태인 프로바이더 체인으로 묶일 수 있다. 체인 메커니즘은 통신 프로토콜 구성에 대한 유연성을 제공한다. 그림 5는 프로토콜 체인을 사용한 예를 보여준다. 통신 프로바이더 체인은 한개 이상의 통신 프로토콜들을 연결하여

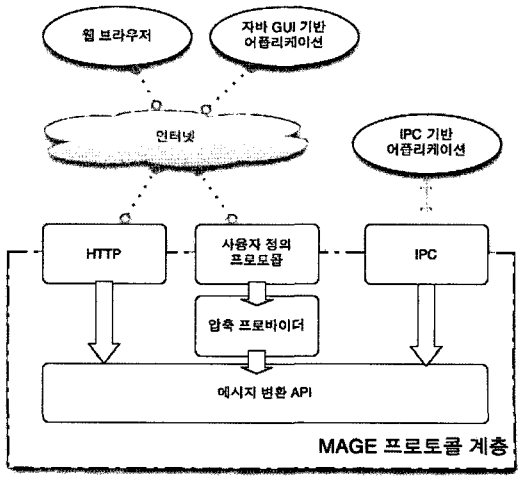

그림 5 통신 프로토콜의 구성 예

기능의 확장성을 제공해 준다. 그림 5에서 HTTP 프로토콜은 다른 프로바이더 없이 작업하지만, 사용자 정의 프로토콜의 경우에는 압축 프로바이더와 체인으로 묶여 있다. 이 흐름에서 송수신되는 자료는 사용자 정의 프로토콜과 상관없이 자동으로 압축되거나 압축 해제된다.

3.4 사용자 요청 메시지 처리를 위한 질의 인터페이스

모니터링 어플리케이션에서 사용자가 요청한 메시지를 처리할때 유연성을 제공하기 위해 메시지를 처리하는 부분을 통신 프로토콜과 질의 계층으로 나눈다. 통신 프로토콜 프로바이더는 사용자로부터의 연결 요청을 접수하며 HTTP 등과 같이 자신이 사용하는 프로토콜로 메시지를 해석한다. 해석된 메시지는 메시지 내용 형태에 따라 적절한 질의 인터페이스에게 전달된다. 모니터링 정보의 기본적인 질의 처리를 위해서 MonQuery 프로바이더를 개발하였다. 이 프로바이더는 SQL 형태의 문법을 사용하여 요청 메시지를 해석한다. 이것은 모니터링 요청 처리에 적합하도록 SQL의 SELECT 문법을 약간 수정한 것이다. 쿼리 문법은 다음 예와 같은 방식을 사용한다.

```
SELECT MemTotal, MemFree FROM MemInfo
    DOMAIN apple* WHERE MemFree > 256

SELECT 1, 2 FROM DoExec USING cat /proc/
cpuinfo
```

이 예제에서 사용자는 apple이라는 이름으로 시작하는 노드들에 있는 MemInfo 프로바이더를 통해 모니터링 정보를 수집한다. 모니터링되는 정보를 최소화하기 위해 WHERE 문법을 사용해서 미사용 중인 메모리가 256MB 이상인 노드만 검색하도록 하고 있다. 두번째 문장에서는 DoExec 프로바이더를 사용하여 /proc/cpuinfo 파일을 읽어 처음 두 줄만 가져오도록 하고 있다.

3.5 모니터링 정보의 구독과 발행

본 논문의 모니터링 시스템은 GMA 명세서를 기반으로 하여 생산자와 소비자간의 통신을 위해 발행/구독 모델을 사용한다. 한쪽에서는 이벤트 메시지를 레지스트리에 발행하며, 다른 쪽에서는 이 메시지를 구독할 수 있다. 그림 6은 모니터링 시스템에서 사용하는 이벤트 모델의 흐름을 보여준다. 서비스 프로바이더로서 생산자는 소비자에게 전달하려는 이벤트를 레지스트리에게 통지한다. 소비자도 생산자에게 이벤트를 통지할 수 있다. 자료 발생이 빈번하지 않은 경우에 'Pull' 방식의 자원 수집은 비효율적이다. 이러한 경우에는 불필요한 자료 요청 때문에 생산자의 시간이 낭비된다. 자료가 많지 않은 경우에는 생산자가 자료 발생시마다 'Push' 하는 방식이 더욱 효율적이다. 이것은, 소비자가 레지스트리에

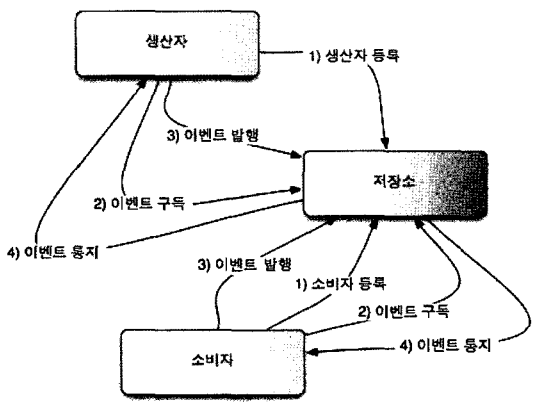

그림 6 모니터링 시스템의 이벤트 처리 흐름

자신의 요구사항을 등록하면 생산자가 새로운 이벤트를 발행했을 때 레지스트리에 의해 소비자에게 이벤트 통지가 이루어진다.

4. 모니터링 시스템을 위한 컴포넌트 기반의 계층형 구조

본 논문의 모니터링 시스템은 모니터링 에이전트 및 통신 프로토콜, 질의 프로바이더의 동적 구성을 위해 MAGE(Modular and Adaptive Grid Environment)를 사용한다. MAGE는 기본 인터페이스를 통해 서비스 지향 구조로 어플리케이션을 개발할 수 있도록 한다. MAGE는 다음과 같은 특징들을 제공한다. 첫째, MAGE는 동적으로 제어가 가능한 컴포넌트 기반 구조를 사용하여 분산 환경에서 효율적으로 구성 요소를 변경할 수 있다. 둘째, MAGE는 서비스 지향 구조를 사용하여 기능에 대한 추상화를 제공한다. 마지막으로 다양한 통신 프로토콜들을 컴포넌트 형태로 추가 및 제거할수 있어 서비스와 통신의 분리를 가능하게 한다. 그림 7은 MAGE의 기본 구조를 보여준다.

컴포넌트 계층은 컴포넌트의 관리 기능을 제공한다. 모든 기능들은 MAGE에서 제공하는 컴포넌트 인터페이스에 따라 구현된다. 컴포넌트는 수행 중 동적으로 적재 및 제거될 수 있으며, 사용자는 컴포넌트 관리 API를 사용하거나 MAGE에 미리 적재되어 있는 원격 제어 컴포넌트를 통해서 컴포넌트들을 제어한다. 컴포넌트에 대한 모든 요청은 컴포넌트 계층의 한 부분인 메시지 전달자에 의해 이루어진다.

서비스 계층은 서비스 관리, 이벤트 관리, 서비스 자료 저장소를 위한 기능을 제공한다. 이러한 기능들은 서비스 처리를 위한 기본적인 역할을 수행한다. MAGE는 UDDI에서 제공하는 서비스 발견과 같은 기능을 제공하지는 않지만, 서비스 발견 컴포넌트를 이용해 유사한 기

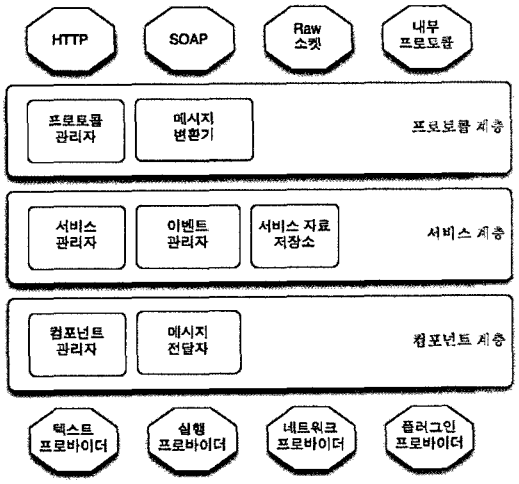

그림 7 MAGE의 구조

능을 플러그인해서 사용할 수 있다. 서비스 관리자는 컴포넌트들에 의해서 제공되는 서비스들의 정보를 관리한다. 이벤트 관리자는 GMA 명세서를 따라서 컴포넌트들 간의 이벤트 흐름을 제어한다. 다른 컴포넌트와 통신하고자 하는 컴포넌트는 수행역할에 따라서 구독자 또는 발행자가 될 수 있다. 각 서비스는 작업을 진행하는 중에 개인적인 자료를 가질 필요가 있다. MAGE는 물리적인 개념인 컴포넌트와 논리적인 개념인 서비스 사이의 의존성을 제거하고 개별 자료를 독립적으로 다룰 수 있도록 하기 위해서 각 서비스에 대해 유일 식별자를 사용해 서비스 자료를 관리한다.

프로토콜 계층은 현재 설치되어 있는 통신 프로토콜 프로바이더들을 관리한다. 프로토콜 프로바이더도 동적으로 적재 가능한 컴포넌트로 구성되지만 서비스 컴포넌트와는 다른 인터페이스를 사용한다. 프로토콜 계층은 통신 프로토콜 프로바이더의 효율적인 활용을 위해 프로토콜 프로바이더들을 묶어서 향상된 하나의 기능을 제공할 수 있다. 각 프로토콜 프로바이더는 고유의 통신 방식을 사용하지만, 수신된 정보는 MAGE 내부 포맷으로 변환되어 사용된다. 변환된 메시지는 서비스 계층과 컴포넌트 계층을 통해 적절한 컴포넌트에게 전달된다.

5. 모니터링 시스템을 기반으로 구현된 그리드 모니터링 어플리케이션

본 논문에서 제시한 모니터링 시스템을 기반으로 그리드 모니터링 어플리케이션을 개발했다. 모니터링 시스템에서 제공하는 기능들은 모니터링을 위한 노드간의 배치구조에 관여하지 않기 때문에 사용자는 필요한 환경에 따라 시스템의 배치 구조를 선택하여 추가 기능을 넣을 수 있다. 본 논문에서 개발한 모니터링 어플리케이션은 트리형태의 구성을 택했다. 기본 모니터링 시스템의 질의 언어 외에 동적으로 노드 정보를 수집하고 처리하기에 적합하도록 Heart-beat 질의 언어를 추가했다. 사용자는 하위 서비스들에 영향을 미치지 않는 상태에서 질의 언어를 바꿀 수 있다. 시스템 관리자는 새롭게 추가되는 노드들에 대해 일일이 등록할 필요가 없도록 멀티캐스트 프로토콜을 사용하여 수행중인 노드에 대한 최신 정보를 얻는다. 모니터링을 위한 기본 서비스로서 몇 개의 새로운 프로바이더가 추가되었으며, 모니터링 구조를 만들기 위해 서버 브로커, 서버 프록시, 서버 에이전트가 적절한 노드에 설치된다. 사용자는 추가로 모니터링 수집을 위한 프로바이더를 설치할 수 있다. 그림 8은 모니터링 어플리케이션의 구성 예이다.

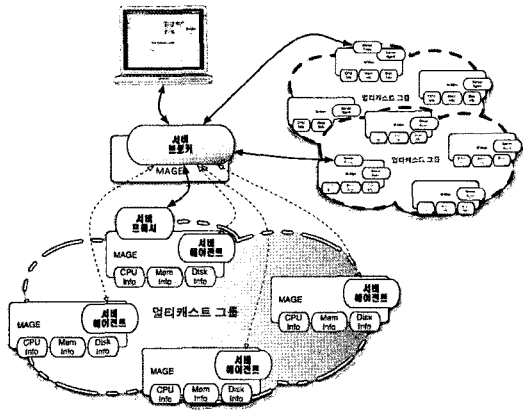

그림 8 트리 형태의 모니터링 어플리케이션 구조

5.1 기본 프로바이더

서버 브로커: 이 프로바이더는 서버 프록시를 통해 활동 중인 노드들의 리스트를 수집한다. 서버 브로커는 서버 프록시에게 현재 활동 가능한 노드들의 정보를 수집하기 위해 heart-beat 신호를 보내며 활동중인 서버 프록시의 리스트를 관리한다. 또한 사용자의 모니터링 요청 메시지를 활동 중인 노드 중 적절한 서비스를 가지고 있는 노드들에게 전달하는 일을 수행한다. 성능을 개선하기 위해 서버 브로커는 사용자 요청 메시지를 필터링하여 선택된 노드들에게만 메시지를 전달한다.

서버 프록시: 서버 브로커로부터 heart-beat 신호를 받은 후 멀티 캐스트 프로토콜을 통해 멀티 캐스트 그룹내의 모든 노드들에게 heart-beat 신호를 전달한다. 각 멀티캐스트 그룹 내에는 한 개 이상의 서버 프록시가 존재 한다.

서버 에이전트: 서버 프록시를 통해 서버 브로커의 heart-beat 신호를 접수한 후 서버 브로커에게 노드에 대한 정보를 제공한다. 노드 정보에는 네트워크 주소, 현재 사용 가능한 프로토콜 목록, 설치되어 있는 질의

인터페이스들, 이 노드가 제공하는 서비스 목록 등이 포함된다.

5.2 모니터링 클라이언트

추가된 모니터링 기능들을 이용하여 CPU, 메모리, 디스크의 사용량과 같은 노드에 대한 기본적인 정보를 그림 9와 같은 클라이언트 그래픽 인터페이스를 통해서 보여준다. 클라이언트는 서버 브로커 프로바이더가 설치되어 있는 노드의 주소를 명시하여 그림과 같은 형태의 수행 결과를 얻을 수 있다. 사용자는 사용자 인터페이스의 왼쪽 상단에 있는 리스트 박스를 사용하여 모니터링하고자 하는 그룹을 선택할 수 있다. 그림 9에서는 네트워크 상에 두 개의 서버 프록시로 apple1.ssu.ac.kr과 sslab이 수행 중이다. 일부 노드 그룹에 대한 정보를 보고자 한다면 서버 프록시 이름을 선택하면 된다. 개발된 어플리케이션에서는 사용자가 오직 서버 브로커와 서버 프록시의 주소만 명시하면 된다는 장점을 제공한다. 수행이 시작된 후 서버 브로커는 수행중인 모니터링 노드와 서버 프록시 노드에 대한 정보를 실시간으로 수집한다.

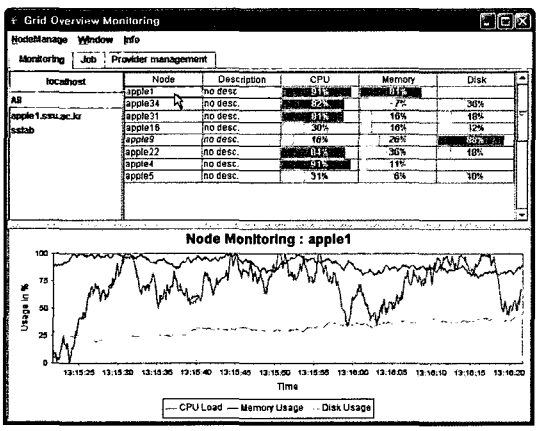

그림 9 모니터링 정보 수집 화면

6. 결 론

다루어야 할 자원이 많고 다양한 분산 시스템에서는 전체 시스템의 관리를 위한 모니터링 어플리케이션의 개발에 드는 비용 부담이 크다. 따라서, 기존 모니터링 어플리케이션들을 최대한 재사용할 수 있으며 환경적인 변화에 능동적으로 대처할 수 있는 시스템이 필요하다. 관련 연구에서 살펴본 바와 같이 기존 모니터링 시스템들은 그리드를 위해서 만들어진 경우에도 그리드 환경의 동적인 변화에 능동적으로 대응할 수 있는 구조를 제공하지 못하고 있다. 본 논문에서 제시한 모니터링 시스템은 각 서비스의 모듈화와 기능의 계층화를 통해 구성 요소의 변경을 용이하게 하며, 기존 시스템과의 통합성을 강화했다.

본 논문에서 제시한 컴포넌트 구조를 기반으로 한 그리드 모니터링 시스템은 다음과 같은 특징들을 가지고 있다. 첫째, 각 기능의 구성을 위해 컴포넌트 기반 구조를 사용한다. 이 기능은 그리드 모니터링 어플리케이션 환경에서 시스템의 구성을 용이하게 한다. 둘째로, 모니터링 정보 수집을 위해 기존 어플리케이션을 쉽게 재사용할 수 있다. 자원 기술서를 통해 사용자는 /proc이나 모니터링 어플리케이션과 같은 기존 자료를 사용하는 방법을 기술한다. 셋째로, 통신 방식에 대해 체인 형태를 제공한다. 이것은 사용자가 기존 프로토콜 프로바이더들을 조합하여 기능을 변경 할 수 있도록 해준다. 네 번째로, 통신 프로토콜과 메시지 해석을 위한 질의 언어를 분리하여 물리적인 통신 방식과 모니터링 정보 표현 방식간의 의존성을 제거하였다. 마지막으로, 본 모니터링 시스템은 GMA를 따르는 이벤트 전달 메커니즘을 제공한다. 서비스 지향 인터페이스와 이벤트 기반 전달 메커니즘을 통해 효율적인 어플리케이션 개발이 가능하다.

현재는 서비스 지향 인터페이스를 통한 동기식 처리와 이벤트를 기반으로 한 비동기식 처리만을 지원하지만, 향후에는 다양한 사용자의 요구에 대응할 수 있도록 통신 방식을 추가로 개발할 것이다. 또한, 다양한 사용자의 조건을 반영할 수 있도록 앞으로는 더욱 다양화된 프로바이더 인터페이스를 개발할 것이다.

참 고 문 헌

[1] I. Foster and C. Kesselman, ed., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1998.

[2] Globus, http://www.globus.org/

[3] Foster, I., Kesselman, C., Nick, J. and Tuecke, S. (June 2002). The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration, http://www.globus.org/research/papers/ogsa.pdf

[4] OGSA, http://www.globus.org/ogsa/

[5] Sungju Kwon, Jaeyoung Choi, MAGE: Modular and Adaptive Grid Environment, GCA 2005, pp. 83-89.

[6] Sungju Kwon, Jaeyoung Choi, Jysoo Lee, Protocol transparent application framework for Grid, Proceedings of HPC Asia 2005, pp. 378-384.

[7] GMA White Paper, http://www-didc.lbl.gov/GGF-PERF/GMA-WG/

[8] Global Grid Forum(GGF), Grid Monitoring Architecture (GMA) Working Group, http://www-didc.lbl.gov/GGF-PERF/GMA-WG/

[9] Tierney, B., Crowley, B., Gunter, D., Holding, M., Lee, J. and Thompson, M. A Monitoring Sensor Management System for Grid Environments, High

Performance Distributed Computing(HPDC-9), Pittsburgh, Pennsylvania, august 2000, pp. 97-104.
[10] Java Agents for Monitoring and Management (JAMM), July 2000, http://www-didc.lbl.gov/JAMM
[11] The NetLogger Toolkit, April 2004, http://www-didc.lbl.gov/NetLogger/
[12] MDS4, http://www.globus.org/toolkit/mds/
[13] Cooke, A., Gray, A.J.G., Ma, L., Nutt, W., Magowan, J., Oevers, M., Taylor, P., Byrom, R., Field, L., Hicks, S., Podhorszki, N., Coghlan, B.A., Kenny, S. and O'Callaghan, D., R-GMA: An Information Integration System for Grid Monitoring. Robert Meersman, Zahir Tair and Douglass C., Schmidt (eds), COOPIS 2003, Lecture Notes in Computer Science, Springer, 2003, 2888, pp. 462-481.
[14] R-GMA: Relational Grid Monitoring Architecture, December 2003, http://rgma.org/
[15] The Datagrid project. February 2004, http://eu-datagrid.web.cern.ch/eudatagrid/

권 성 주

1997년 숭실대학교 소프트웨어공학과(학사). 2000년 숭실대학교 컴퓨터학과(공학석사). 2001년~현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 분산/병렬 컴퓨팅, 그리드, 웹 서비스, 에이전트

최 재 영

1984년 서울대학교 제어계측공학과(학사). 1986년 미국 남가주대학교 컴퓨터공학(석사). 1991년 미국 코넬대학교 컴퓨터공학(박사). 1992년~1994년 미국 국립오크리지연구소 연구원. 1994년~1995년 미국 테네시 주립대학교 연구교수. 2001년~2002년 미국 국립 수퍼컴퓨팅 응용센터(NCSA) 초빙연구원. 1995년~현재 숭실대학교 정보과학대학 컴퓨터학부 부교수. 관심분야는 고성능컴퓨팅(HPC), 병렬/분산처리, 유비쿼터스컴퓨팅

이 지 수

1985년 서울대학교 물리학과(학사). 1986년 University of Pittsburgh 물리학과(석사). 1992년 Boston University, 물리학과(박사). 1992년~1993년 HLRZ-KFA (German National Supercomputing Center), 연구원. 1993년~1997년 City College of New York, 연구원. 1997년~1998년 서울대학교, 초빙조교수. 1998년~2001년 고려대학교, 연구조교수. 2001년~현재 한국과학기술정보연구원, 슈퍼컴퓨팅센터장. 관심분야는 고성능컴퓨팅(HPC), 전산물리