분산처리 작업의 동적 원격실행을 위한 P2P 기반 작업 할당 관리자

A Job Allocation Manager for Dynamic Remote Execution of Distributed Jobs in P2P Network

이 승 하* 김 양 우**
SeungHa Lee Yangwoo Kim

요 약

컴퓨터와 네트워크 기술의 향상은 예전에는 슈퍼컴퓨터에서나 가능한 일을 분산 처리할 수 있는 환경적 기반을 제공한다. 분산 컴퓨팅 환경을 제공하기 위해서는 우선 분산 런타임 시스템이 구축되어야 하는데 기존의 전통적인 분산 런타임 시스템들은 대부분이 정적인 마스터 노드와 작업 노드들로 구성되는 구조를 갖기 때문에 분산처리 작업량의 변동에 따라 시스템을 유연하게 동적으로 재구성할 수 없다는 단점을 갖는다. 이에 본 논문에서는 P2P 환경에서의 분산 런타임 시스템인 작업 할당 관리자의 모델을 제시하고 구현하여 유연하고 동적인 시스템 구축이 가능하도록 하였다. 즉, P2P 표준 프로토콜인 JXTA 플랫폼 상에서 협업 환경을 위해 개발자들 간에 작업 프로그램의 전달과 관리, 그리고 원격 컴파일과 실행 작업들을 수행할 수 있도록 하였다. 이 방식은 유연하고 동적인 시스템 구축이 가능하기 때문에 작업의 분산처리를 위해 필요한 유휴 자원들을 필요한 시점에 즉시 확보하여 활용할 수 있다는 장점을 가진다. 이와 더불어 인터넷 정보검색을 위해 방대한 데이터를 수집하는 크롤러를 본 논문에서 구현한 시스템을 이용하여 분산 처리시킴으로써 본 시스템의 유용성과 분산처리 성능을 보여 줄 수 있도록 하였다.

Abstract

Advances in computer and network technology provide new computing environment that were only possible with supercomputers before. In order to provide the environment, a distributed runtime system has to be provided, but most of the conventional distributed runtime systems lack in providing dynamic and flexible system reconfiguration depending on workload variance, due to a static architecture of fixed master node and slave working nodes. This paper proposes and implements a new model for distributed job allocation and management which is a distributed runtime system in P2P environment for flexible and dynamic system reconfiguration. The implemented system enables job program transfer and management, remote compile and execution among cooperative developers based on P2P standard protocol Jxta platform. Since it makes dynamic and flexible system reconfiguration possible, the proposed method has some advantages in that it can collect and utilize idle computing resources immediately at a needed time for distributed job processing. Moreover, the implemented system's effectiveness and performance increase are shown by applying and processing the crawler jobs, in a distributed way, for collecting a large amount of data needed for Internet search.

☞ Keyword : Peer-to-peer, Remote runtime, Jxta, Distribute Computing, P2P Grid

1. 서 론

엔터프라이즈 환경에서 사용자들의 다양한 요구는 그에 따른 다양한 개발 환경을 필요로 하게 되었다. 네트워크 환경은 손쉽게 단일 사용자 환경에서 다중 사용자 환경으로 변화될 수 있는 기반을 제공하였고, 개발자들의 다양한 요구는 작업 특성에 맞는 환경을 제공하는 시스템을 구성할 필요성을 강조하고 있다. 그러나 이런 시스템을 단일 시스템으로 모두 구성하는

* 정회원 : 동국대학교 정보통신공학과 박사과정
lesh915@dongguk.edu
** 정회원 : 동국대학교 정보통신공학과 부교수(교신(책임)저자)
ywkim@dgu.edu
[2006/07/19 투고 - 2006/08/21 심사 - 2006/11/15 심사완료]

데는 많은 한계가 있다. 따라서 원격에 있는 개발환경을 서로 공유해서 사용할 수 있는 협업 시스템이 필요하게 되었으며, 이러한 협업 시스템은 일반적으로 그룹 형성을 통해서 관리가 이루어진다. 그러나 이러한 그룹 형성은 정적으로 항상 동일한 그룹의 형태로 존재하는 것이 아닌 동적으로도 그룹을 구성할 수 있어야 한다. 이와 더불어 일반 PC뿐만 아니라 노트북과 같은 이동 가능한 기기들도 사용할 수 있어야 하고, 때에 따라서는 필요한 사양을 구성하고 있는 시스템들을 검색해서 작업을 요청할 수도 있도록 해야 한다. 이러한 문제점을 해결하기 위해서는 유연한 그룹을 구성하고 작업을 원격 시스템에 분산해서 처리할 수 있는 분산 시스템이 필요하다.

분산 시스템은 작업 처리를 위한 모델로 요청-응답 모델(request-reply model)인 클라이언트/서버 방식과 클러스터 방식을 통해 서비스를 수행한다. 대표적인 클라이언트/서버 모델로는 웹 브라우저와 인터넷 웹 서버의 형태가 있다. 그러나 이 방식은 처리량 증가에 따라 서버에 많은 부하가 집중된다는 단점이 존재한다. 현재는 서버의 부하집중을 방지하고 안정되며 다양한 서비스를 제공하기 위해서 중간 단계로 웹 응용 서버(Web Application Server), 트랜잭션 서버, DB 서버 등으로 분리하는 n-tier 방식을 사용하고 있다. 하지만 이러한 구조 역시 각각의 서버마다 많은 양의 부하를 효율적으로 분산하는데 어려움이 있다.

클라이언트/서버 방식의 문제점을 해결하기 위해 대안으로 등장한 기술로 P2P(Peer-to-Peer) 모델이 있다. SETI@Home, Napster 등으로 대표되는 P2P모델은 서버에게 집중되는 작업들을 피어들에게 분산시켜서 처리하는 방식이다. P2P 모델을 통한 부하 분산은 사용되지 않는 유휴 자원을 효율적으로 찾아서 이를 피어 그룹으로 처리한 방식으로 일종의 머신 풀(pool)을 형성해서 서비스를 처리할 수 있다는 장점이 있다.

일반적으로, 분산 처리 시스템은 하나의 마스터와 다수의 작업 노드들로 풀을 구성하며, 마스터는 작업의 분배와 결과를 취합하고, 각 노드는 마스터로부터 주어진 작업을 처리하고 이를 반환하는 형태를 가진다. 이는 그리드(GRID) 시스템의 VO(Virtual Organization)를 통해 그리드 서비스를 구성하는 방식과 유사하다. 그리드 시스템은 그리드 미들웨어를 통해 VO라는 형태로 그룹을 구성하고 그룹에 속한 노드들의 자원 공유와 노드들 간의 통신을 수행한다. 그러나 그리드 시스템은 기존의 클라이언트/서버 구조에서 확장된 방식으로 그리드 미들웨어를 통한 자원의 관리와 그룹 관리가 현재까지는 동적으로 처리되지 않아 VO 구성의 유연성이 떨어지는 단점이 존재한다. 반면에 P2P 모델을 기반으로 각 피어의 자율적인 의지로서 그룹에 가입/탈퇴를 수행하게 만들면 동적으로 피어 그룹의 규모를 유연하게 조정할 수 있게 된다.

한편, 분산 시스템을 구성하기 위해서는 기본적인 분산 런타임 시스템이 필요하다. 분산 런타임 시스템은 크게 클러스터 인식 JVM(Java Virtual Machine) 방식, 컴파일러 기반의 Java DSM(Distributed Shared Memory) 방식, 그리고 표준 JVM 사용 방식 등 세가지로 분류할 수 있다[1]. 첫 번째 방식인 클러스터 인지 JVM 방식은 각 참여 노드마다 변경된 JVM을 사용하여 클러스터 형식으로 처리하는 방식이다. 두 번째 컴파일러 기반의 Java DSM 방식은 자바 프로그램을 그 시스템에 맞는 머신 코드로 다시 컴파일하여 작업을 처리하는 방식이다. 세 번째 표준 JVM 방식은 대부분의 자바 기반 분산 처리 시스템에서 사용하는 방식이다. 첫 번째와 두 번째 방식은 모두 시스템의 성능을 향상시키기 위해 각각 머신에 적합하게 변경된 JVM을 사용하고 있고, 이것은 표준 JVM의 장점인 이식성과 호환성을 포기한 방식이다. 반면에 세 번째 방식은 표준 JVM을 사용하기 때문

에 좋은 이식성과 호환성을 갖는다는 장점이 있으나 다른 두 방식에 비해 상대적으로 성능이 떨어진다. 이와는 별도로 위의 분산 시스템들은 모두 미리 정해진 마스터와 작업 노드들을 가지고 시스템을 구성하여 작업해야 하기 때문에 매번 정적인 시스템 구성을 취해야만 하는 문제점이 있다. 즉, 작업량의 증가로 인해 시스템에서 필요한 컴퓨팅 자원들을 추가로 확보하는 경우에는 매번 시스템 구성과 환경을 다시 조정해야 한다. 이러한 문제점을 해결하고 유휴 자원을 동적으로 확보하기 위해서는 유연한 작업 그룹의 형성과 관리가 가능한 P2P 구조를 기반으로 시스템을 구축하는 것이 한 좋은 대안이 될 수 있다.

본 논문에서는 그리드와 같은 분산처리 환경에서 Java 분산처리 작업의 원격실행을 위한 P2P기반 작업 할당 관리자를 설계 및 구현하였다. 제안된 시스템은 위에서 언급된 세가지 런타임 시스템 방식 중 세 번째 방식인 표준 JVM을 이용하는 방식을 취하기 때문에 작업의 호환성과 이식성이 우수할 뿐만 아니라, P2P 기술을 기반으로 설계되었기 때문에 유연한 피어 그룹의 동적 구성과 관리가 용이하다. 이와 더불어 여분의 유휴 자원 사용을 통하여 가능한 자원의 확보를 극대화 시킴으로써 작업의 증가에 따른 추가 시스템 구축 비용을 절감 할 수 있다.

본 논문에서 제시하는 P2P기반 작업 할당 관리자는 자바에 기반한 JXTA 플랫폼을 사용하여 구현하였고, 이에 따른 장점으로 이식성, 확장성, 그리고 자율성을 갖는 동시에 결함 허용 기능을 구현할 수 있다. 또한 자바 스윙(Java Swing) 기반의 GUI 환경을 통해 편리한 사용자 및 관리자 인터페이스를 제공하도록 하였다. 본 논문에서 제안하는 작업 할당 관리자는 각 피어에 설치된 표준 JVM 방식의 원격 컴파일/실행 런타임 시스템을 통한 소스의 바이트코드 생성 및 실행 서비스를 제공한다. 또한 본 시스템에는 그룹에 속한 피어 자원들의 CPU 유휴량 및 메모리 사용량을 실시간으로 모니터링하는 기능을 추가하여 사용자가 필요한 자원의 개수만 명시하면 그룹에 속한 자원들 중에서 유휴한 자원들만을 자동으로 선택하여 작업 수행을 시킬 수 있도록 하였다. 유휴 자원의 판단 기준이 되는 CPU 및 메모리 사용량과 지정된 사용량 미만으로의 상태 지속시간은 사용자 정책에 따라 결정되고 입력할 수 있으나 본 논문에서는 CPU 유휴량은 90%이상, 사용 가능한 메모리 크기는 50M이상으로 지정하였고, 이 수치를 만족하는 상태로 5분 이상 지속될 때 유휴자원으로 분류되고 사용될 수 있도록 설정하여 실험하였다.

본 논문의 구성은 다음과 같다. 2장에서는 일반적인 자바 분산 처리 런타임 시스템들과 P2P 모델 및 JXTA 플랫폼에 대해서 살펴보고, 3장에서는 본 논문에서 제시하는 작업 할당 관리자의 핵심인 P2P기반 분산 런타임 시스템을 위한 원격 컴파일/실행 모델의 구조와 특징에 대하여 설명하였다. 이어 4장에서는 구현된 모델에 대하여 설명하고, 이를 적용하여 실험한 분산 웹 크롤러의 성능 향상 및 평가를 통하여 본 논문에서 제안한 시스템의 유용성을 보여주도록 하였다. 마지막으로 5장에서 결론 및 향후 연구 과제에 관해서 제시하도록 한다.

2. 관련 연구

2.1 기존 자바 분산 런타임 시스템

가. 클러스터 인식(Cluster-aware) VM

클러스터 인식(Cluster-aware) VM(Virtual Machine)은 작업 노드들이 비표준 JVM 방식으로 분산 처리를 실행하는 시스템이다. 가장 대표적인 시스템의 예로는 Java/DSM(Distributed Shared Memory)[2], 자바를 위한 클러스터 VM(cJVM :

Cluster VM for Java)[3], 그리고 JESSICA2[4] 등이 있다.

Java/DSM의 로컬 작업 노드에 있는 VM은 표준 JVM과 유사하다. 다만 차이점은 모든 객체들이 C기반의 DSM 소프트웨어에 할당되어 처리된다는 점이다. 작업 노드들 간에 쓰레드 이동은 불가능하며, Java/DSM에서 처리되는 단일 시스템 이미지 또한 불완전하기 때문에 쓰레드의 위치는 프로그래머한테 투명하지 않다. 반면에 자바를 위한 클러스터 VM과 JESSICA2는 전통적인 JVM을 기반으로 보다 향상된 단일 시스템 이미지를 제공한다.

표준 JVM을 사용하는 시스템과 비교하여 클러스터 인식 VM을 사용하는 큰 이점은 머신의 자원들(메모리, 네트워크 인터페이스 등)을 JVM에 비해 보다 더 직접적으로 이용할 수 있다는 것이다. 수정된 JVM을 사용하는 방식은 각 머신에 적합하게 변경된 JVM이기 때문에 이식성이 떨어지는 단점이 존재하지만 클러스터 인식 VM은 이식성이 떨어지는 단점을 감수하고도 성능의 향상을 위해 비 표준 JVM을 사용한다.

나. 컴파일러 기반 Java DSMs

컴파일러 기반의 DSM 시스템은 자바 프로그램에 DSM 핸들러라고 불리는 것을 추가하여 네이티브 머신 코드로 재 변환하여 컴파일하는 방식이다. 이 방식에는 대표적으로 Hyperion[5]과 Jackal[6] 방식이 있다. 두 방식 모두 표준 자바 문법을 지원한다.

Hyperion은 자바 바이트코드를 C 소스 코드로 변경하고 나서 C 소스 코드를 네이티브 C 컴파일러를 사용해서 컴파일 한다. 이 과정에서 DSM 핸들러는 C로 변경되는 동안에 추가되고, 자바 바이트코드를 C로 변경하는 변경자는 DSM 성능을 높이기 위해 다양한 최적화 기법을 수행한다. 예를 들어, 만약에 공유 객체가 반복문에서 매번 참조된다면 객체의 로컬 지역에 복사된 캐시에서 얻어진 코드로 반복문을 수행하도록 해서 그만큼의 속도를 향상시키는 것이다.

Jackal은 확장된 자바 컴파일러와 런타임 환경을 제공하고, 이 컴파일러를 이용해 자바 코드를 인텔 x86 코드로 변환한다. Jackal은 Hyperion과 비슷하게 좀 더 효과적인 분산 실행을 위해 다양한 최적화를 수행하지만, 분산된 가비지 컬렉터와 객체 지역 투명성을 제공한다는 차이점이 있다.

컴파일러 기반의 시스템들은 C 또는 C++ 컴파일러를 이용하여 생성된 머신 코드를 사용하기 때문에 성능상의 장점이 있지만 이식성은 클러스터 인식 VM 보다 더 좋지 않다. 물론 클러스터 인식 VM도 변경된 JVM을 사용하기 때문에 표준 JVM을 사용하는 방법만큼의 이식성은 가지고 있지 않다. 따라서 각 머신에만 적합한 작업을 처리하는 경우에는 성능상의 장점을 갖지만 바이트코드의 이식성이 요구되는 작업의 경우에는 적합하지 않다.

다. 표준 JVM 사용 시스템들

대부분의 자바 기반 분산 시스템들은 표준 JVM을 이용한다. JavaParty[7]와 JDSM[8] 같은 시스템들은 순수 자바에 가까운 표준 자바 프로그램을 실행할 수 있다.

JavaParty는 전처리와 런타임을 포함한 확장된 자바를 통해 분산된 병렬 프로그래밍을 지원한다. 주요 수정사항은 분산된 머신에 원격 처리하는 방법을 추가하여 처리한다는 것이다. 이 방법은 소스 코드에 RMI 코드를 추가해서 RMI 컴파일러를 이용하여 전송하는 것이다. 이와 같이 구성된 단일 시스템 이미지는 프로그래머가 원격과 로컬 메소드 호출을 구분해야 할 필요성을 줄여준다.

JDSM에서는 일관된 메모리 작업을 유지 하

기 위한 메소드 호출 형태의 접근 체크가 사용자나 상위레벨의 프로그램 변환자에 의해서 매뉴얼 하게 삽입된다. JDSM은 입력 프로그램으로 SPMD 스타일 멀티스레드 프로그램을 요구한다. 또한 프로그래머는 JDSM에서 제공하는 특별한 클래스를 사용하여 공유 객체를 표시하여야 한다.

표준 JVM을 사용하는 데는 몇 가지 장점이 있다. 첫 번째는 전체 시스템을 이기종 노드들의 집합으로 구성할 수 있다는 것이고, 두 번째는 각각의 노드들이 JIT 컴파일러를 통해 자신들의 성능을 최적화할 수 있다. 세 번째로는 로컬 가비지 컬렉터를 이용하여 사용되지 않는 로컬 객체와 참조되지 않는 객체들을 자동으로 수집한다.

하지만 단점은 시스템의 가용성과 유연성이 떨어진다는 점이다. 전통적인 분산 시스템과 마찬가지로 이 방법 또한 특정 마스터와 작업 노드가 정해지고 각 시스템의 성능 이상으로 컴퓨팅 파워가 필요한 경우에는 추가로 새로운 시스템들을 확보한 후 다시 전체 시스템을 재구성해야 한다.

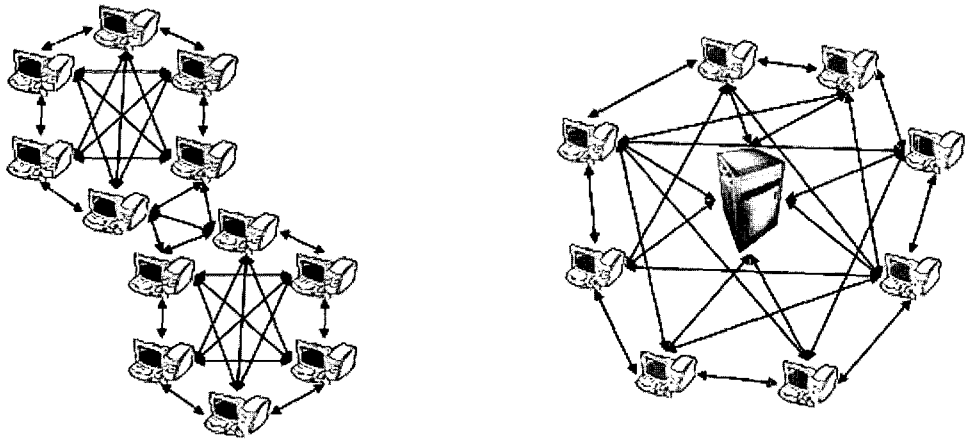
본 논문에서는 위에 언급된 표준 JVM을 사용하여 분산 런타임 시스템을 구성하되, P2P 네트워크를 기반으로 구축함으로써 피어들의 그룹 형태로 작업 노드들을 관리할 수 있고, 이에 따라 동적이고 유연한 작업 노드 그룹을 구성할 수 있도록 하자는 것이다. 작업 노드들은 수시로 필요에 따라 검색을 통해 찾을 수 있으며, 그룹에 가입과 탈퇴는 피어의 자율적인 판단에 따라 이루어 진다.

2.2 Peer-to-Peer (P2P) 모델

P2P는 응용수준의 프로토콜로 사용자 사이의 실시간 통신이나 자원분배 및 교환 등을 지원하는 동기적 상태지원 기술이다[9][13]. 즉, 인터넷 혹은 네트워크 상에서 각 컴퓨터가 동등한 입장에서 저장장치와 주변장치 등의 컴퓨팅 자원을 별도의 서버를 거치지 않고 공유하는 것을 말한다. 이 때문에 P2P는 클라이언트 PC 사이에서 처리가 완결 될 수 있어 서버를 거치지 않는 분산처리가 실현 가능하도록 한다. 또한 실시간 정보를 생성하는 정보원의 역할과 실시간 정보공유 및 디지털 컨텐츠의 교환을 특별한 조작 없이 가능하게 하여 동기적 커뮤니티 서비스에 대한 새로운 클라이언트 컴퓨팅 이용환경을 제공한다.

<그림 1>에서 보듯이 P2P 모델은 다음과 같이 크게 혼합(Hybrid)형과 순수(Pure) 형으로 구분할 수 있다[9]. 혼합형은 서버와 복수의 서번트(servant)로 구성되는 시스템이다. 서버는 정보의 검색기능, 인증기능과 메시지의 일시적 보관기능 등을 수행한다. 검색의 속도나 검색 성공률 증대 및 동시 접속자의 증가에 따라 서버의 증설계획 및 서버의 관리가 필요하다. 그러나 혼합형 방식은 개방형 시스템의 일종이기 때문에 서버 부하가 크지 않아 서버 성능이나 수에 대한 투자상의 문제는 크지 않다. 단지 중앙에 서버가 있음으로 해서 저작권의 문제에 민감한 단점이 있다.

순수형은 순수 P2P 형으로 네트워크에 연결된 각각의 컴퓨터가 서버와 클라이언트의 모든 역할을 한다. Server + Client = Servant 라고 부르기도 한다. 즉, 네트워크에 연결된 피어를 동적으로 찾고 피어가 중앙서버의 의존 없이 자료나 리소스를 공유하며 상호 대칭적 의사소통을 한다. 그러나 검색명령 등의 재전송으로 인한 검색시간이 길어지고, 네트워크에 부하를 주며, 이용자에 따른 사용권한을 설정할 수 없어 보안유지를 기대할 수 없는 단점이 있다. 이에 따른 해결책으로 각 서번트 측에서 개별적으로 인증하는 방법을 사용하기도 한다. 시스템 전체에 대한 관리가 필요 없으며, 시스템 가동을 제한하는 요인도 없으므로 365일 24시간 무정지로 운용된다. 서버부하가 없으며, 시스템의 구



〈그림 1〉 P2P의 유형 분류, 순수형 P2P와 혼합형 P2P의 시스템 구성도

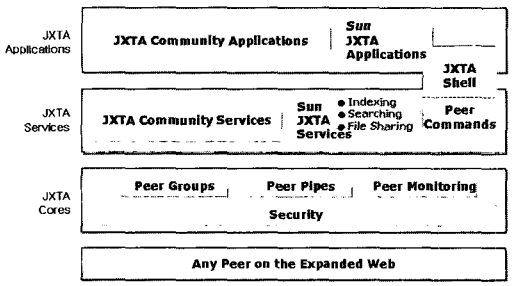
성이 서번트 사이의 통신부하를 유동적으로 분산할 수 있다.

이와 같은 P2P 기반의 시스템들로는 SETI@Home[17]이나 Korea@Home[18] 등이 있다. SETI@Home이나 Korea@Home 같은 시스템은 작업의 제어와 관리의 편의를 위해 중앙 서버를 두고 작업하는 혼합형 P2P 모델을 지원하는데 반해 본 논문에서 제시하고자 하는 모델은 순수 P2P 모델을 지향한다. 이를 통해 자율적으로 피어의 그룹 참여 및 작업 할당과 처리에서 유연성을 가지도록 하고 있다. 또한 기존 시스템의 경우에는 통신 방법으로 대부분 자신만의 프로토콜을 사용하거나 방화벽 문제를 고려하여 HTTP를 기반으로 사용하지만, JXTA 플랫폼을 이용하면 네트워크의 가상화가 이루어져 다양한 네트워크 환경에서도 시스템이 실행될 수 있다. 이와 더불어 일반적인 P2P 시스템들은 개발된 목적 외에는 사용될 수 없지만, 본 논문에서 구현한 작업 할당자의 경우에는 JXTA 플랫폼을 이용하였기 때문에 다른 JXTA 응용 프로그램과 결합이 용이한 것은 물론이고 윈도우나 리눅스 또는 유닉스 등 다양한 이질적인 환경에서도 상호 동작한다.

2.3 JXTA™ Platform

JXTA[10,12,14]는 P2P 네트워크 플랫폼이며, "Juxtapose"의 줄임 말로서 네트워크 프로그래밍과 분산 컴퓨팅 환경을 P2P기반으로 제공하는 기술이다. JXTA 프로젝트는 원래 작은 규모의 연구소나 회사에 소속되어있는 전문가들의 참여를 확대시키기 위해 개발된 것으로 현재 오픈 소스로 개발이 진행되고 있는 프로젝트이다.

가. JXTA 구조



〈그림 2〉 프로젝트 JXTA 소프트웨어 계층 구조

썬마이크로시스템즈는 초기 많은 P2P 애플리

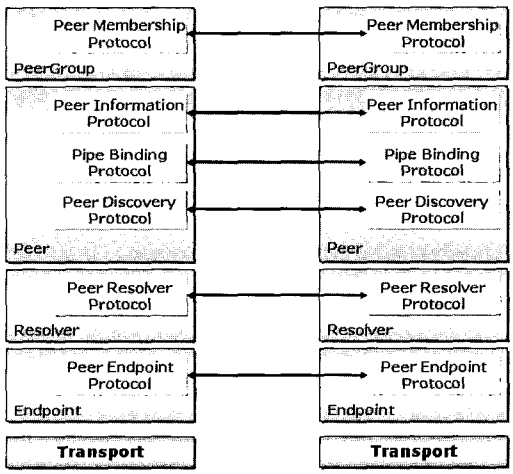
케이션들이 갖는 공통적인 아키텍처를 분석해서 개념적인 일반적 계층 구조를 발견했다. 그리고 이러한 계층 구조를 바탕으로 <그림 2>와 같은 소프트웨어 계층 구조도를 작성했다[10,16].

JXTA 코어(Cores) 계층은 JXTA의 플랫폼 계층으로 피어와 피어그룹의 생성 및 보안 설정 등 P2P 네트워킹을 위한 기본적인 사항들을 포함하고 있다. JXTA 서비스 계층은 P2P 환경에서 요구되는 기본적인 서비스들에 대해 정의한 부분이다. 예를 들어 검색, 인덱싱, 저장 시스템, 파일 공유, 분산 파일 시스템, 인증 등과 같은 서비스들을 포함한다. JXTA 응용 계층은 통합 구현된 애플리케이션을 포함한다. 예를 들어 P2P 인스턴스 메신저, 문서와 자원 공유, P2P E-mail, 그리고 분산 경매 시스템과 같은 애플리케이션들이 여기에 포함된다.

나. JXTA 프로토콜

JXTA는 XML 메시지 형식의 6개의 프로토콜을 가지고 피어간의 통신을 정의하고 있다[15]. 피어들은 이 프로토콜을 이용하여 서로간의 발견 및 광고, 탐색과 자원 전달, 통신, 그리고 메시지 라우팅 등의 기능을 수행할 수 있다. 각 프로토콜은 그 역할에 따라 다른 작업을 수행하지만 JXTA 애플리케이션에서 모두 이용할 필요는 없으며 단지 원하는 작업을 수행하기 위해 알맞은 프로토콜을 사용하면 된다. <그림 3>은 이들 프로토콜들 간의 관계와 계층도를 보여주고 있다.

Peer Discovery Protocol (PDP)은 피어 자신의 자원이나 다른 피어들의 자원을 광고(Advertisement)하는 프로그래밍 중립적인 메타데이타를 찾는 프로토콜이다. 피어 광고(Advertisement)는 피어의 이름 및 자원의 존재를 설명하고 출판하는 XML 구조의 문서이다. PDP는 광고를 이용하여 정의된 피어와 피어그룹 그리고 파이프 및 서비스들을 발견하며, 기본적으로 우선 "World Peer Group"에 포함된 피어그룹과 피어들에 대한 정보를 검색한다. 이 프로토콜은 모든 JXTA 피어들이 기본적으로 사용하여야 하는 기본 프로토콜이다.



〈그림 3〉 JXTA 프로토콜

Peer Information Protocol (PIP)은 피어들의 상태 정보를 획득해서 메시지 형태로 제공하는 프로토콜이다. 피어들의 상태 정보는 예를 들어 업타임, 상태, 최근 트래픽 등과 같은 것들이 있고, ping 메시지와 같이 상대 피어가 작동 중인지를 알려주는 것도 있다. 피어에 질의를 할 때는 이름과 값을 문자열 형태로 만들어 피어의 속성을 지정해서 이용할 수 있다.

Peer Resolver Protocol (PRP)은 피어들이 다른 피어들과 요청한 질의를 주고 받을 수 있는 프로토콜이다. PDP와 PIP는 둘 다 이 프로토콜을 이용하여 질의를 주고 받을 수 있다. 일반적으로 PRP는 데이터 저장소를 접근할 수 있고, 향상된 검색 기능을 가진 다른 피어에 의해 구현된다.

Pipe Binding Protocol (PBP)은 하나 이상의 피어들 사이에 가상의 통신 채널을 설정하여 이용할 수 있게 하는 프로토콜이다. PBP는

HTTP와 같은 전송 계층을 설정할 수 있는 추상 계층을 지원한다. 이렇게 파이프가 설정되면 광고 메시지는 다른 피어들에 의해 이용될 수 있게 된다.

Endpoint Routing Protocol (ERP)은 피어가 원하는 목적지를 찾기 위한 라우팅 정보를 제공하는 프로토콜이다. 이 프로토콜을 이용하면 하나의 피어에서 다른 목적지 피어로 메시지를 보내 라우팅 정보를 얻을 수 있다. 또한 연결된 피어가 원하는 목적지가 아닐 경우는 계속적으로 라우팅 정보를 얻기 위한 질의를 예약한다. 이 예약된 질의를 통해 다음 피어에서 라우팅 정보를 알게 되면 원하는 목적지를 알 수 있다.

Rendezvous Protocol (RVP)는 피어 그룹 안에 있는 피어들에게 메시지를 전송하기 위한 프로토콜이다. 피어 그룹 안에 있는 피어들은 각각 랑데부 피어가 되거나 랑데부 피어들을 리스닝할 수 있다. RVP는 모든 피어들에게 메시지 보내는 것을 허용하며, Peer Resolver Protocol (PRP)과 Pipe Binding Protocol (PBP)을 사용하여 메시지를 전송한다.

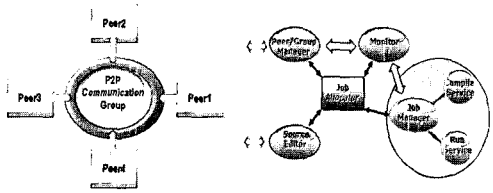
3. P2P기반 분산 런타임 시스템을 위한 원격 컴파일/실행 모델

P2P기반 분산 런타임 시스템을 위한 원격 컴파일/실행 모델은 JXTA 플랫폼 상에서 작동하는 애플리케이션으로 이를 이용하고자 하는 모든 컴퓨터에는 JVM(Java Virtual Machine)과 JXTA 플랫폼이 필요하다.

본 논문에서 제시하고 있는 모델의 분산 작업 할당자는 자바 소스를 작성할 수 있는 자바 스윙(Java Swing) 기반의 GUI 에디터를 제공하여 원하는 소스 프로그램을 작성할 수 있도록 한다. 작성된 소스는 로컬이나 원격 두 가지 방법을 다 이용하여 컴파일을 수행할 수 있도록 하고, 컴파일 된 바이트코드를 로컬이나 원격에 저장할 수 있다. 컴파일 된 바이트코드는 로컬이나 원격에서 실행시켜 그 결과를 취합할 수 있다. 마지막으로 모든 피어 시스템의 상태 정보와 할당 받은 작업의 처리 결과를 모니터링할 수 있는 모니터링 기능도 제공한다.

작업은 사용자가 작성한 작업처리 구성정보 파일의 값들을 바탕으로 처리된다. 이 파일에는 사용자가 사용할 피어 개수, 그리고 사용할 수 있는 피어의 시스템 상태 조건 등이 입력되어 있다.

작업 할당자는 피어들의 시스템 상태 모니터링을 통하여 수집된 피어의 상태 정보와 사용자가 작성한 작업처리 구성정보 파일을 비교하여 필요한 개수의 피어들과 수행할 작업을 자동으로 할당한다.



〈그림 4〉 P2P기반 분산 런타임 시스템을 위한 원격 컴파일/실행 모델 구조도

3.1. 소스 에디터 서비스

소스 에디터 서비스는 자바 소스와 바이트코드 파일 및 기타 필요한 작업 파일들을 관리하는 GUI환경을 제공한다. 파일 관리자 모듈은 현재 로컬 컴퓨터에 있는 파일 탐색 기능을 제공함으로써 로컬 컴퓨터의 자바 소스 파일과 바이트코드 등을 관리하는 기능을 가진다. 소스 에디터 모듈은 자바 소스를 작성하고 이를 파일로 저장하거나 소스 코드를 수정하고 필요 없는 파일의 경우 삭제할 수 있는 기능을 가지고 있다. 로컬이나 원격에서 수행된 컴파일이나 실행의 결과는 바로 결과 창에서 확인할 수 있다.

3.2. 피어/그룹 관리 서비스

피어/그룹 관리 서비스는 런타임 시스템에서 동일 서비스로 동작할 수 있는 그룹과 피어를 검색하고 그룹에 피어의 참여/탈퇴를 관장하는 서비스이다. 피어/그룹 관리 서비스는 피어가 JXTA 네트워크에서 동작 중일 경우 그룹에 동참해 있는 피어들의 리스트를 검색할 수 있다. 검색 된 피어의 정보는 피어 광고(Advertisement) 형태로 확인할 수 있고, 원격에서 실행할 수 있는지에 대한 상태 정보를 볼 수 있다. 각 피어는 이 상태에서 작업을 할 수 있도록 대기한다. 또한 각 피어는 다른 피어들에게 처리하고자 하는 작업을 송수신하기 위한 파이프 서비스를 제공한다. 파이프 서비스는 JXTA 네트워크 상에서 데이터를 송수신하기 위한 가상채널이다. 검색된 피어는 실제로 파이프를 통해서 자바 소스와 바이트코드를 전송해야 하기 때문에 각 피어로부터 사용할 수 있는 파이프 정보를 가져와야 한다.

3.3. 컴파일/실행 서비스

컴파일/실행 서비스는 로컬과 원격의 컴파일러를 동작시켜 작업을 실행시키는 서비스로 로컬의 JVM을 사용하여 작업을 수행할 수 없는 경우에도 원격으로 작업을 처리할 수 있다. 원격으로 작업을 처리하기 위해서는 우선, 서비스가 가능한 피어들에 대한 상태 정보를 이미 모니터링하여 가져온 상태여야 하고, 현재 상태가 사용 가능한 경우여야 한다. 또한 처리된 작업의 실행 결과를 볼 수 있도록 한다.

3.4. 정보 제공 모니터 서비스

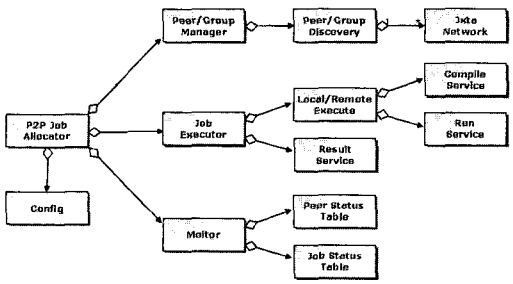
정보 제공 모니터 서비스는 현재 사용 가능한 피어의 상태 정보를 주기적으로 관리하는 서비스이다. 현재 로컬에서 동작하는 피어의 정보와 원격의 피어 정보 등을 저장하고 관련 피어의 광고와 파이프 광고를 가지고 현재 서비스가 가능한지 파악한다. 또한 해당 시스템의 상태 정보를 파악하고 필요한 경우 이를 관리자가 자동으로 작업을 할당 할 수 있도록 제공한다.

모니터 서비스에서 파악하는 정보들은 크게 두 가지이다. 첫 번째는 피어 그룹에 참여한 피어 상태 정보이고, 이들은 피어 상태정보 테이블에 저장된다. 피어 상태정보 테이블은 각 피어들의 피어 ID, 피어 이름과 피어의 시스템 상태 정보(CPU 사용량, CPU 유휴량, 총 메모리 크기, 메모리 사용량)를 저장한다. 두 번째는 각 피어에서 할당 받아서 처리하는 작업에 대한 상태 정보이고, 이들은 작업상태 테이블에 저장된다. 작업상태 테이블은 작업을 할당 받은 피어 ID와 작업 시작, 처리, 종료 등과 같은 작업의 상태 정보를 저장한다.

4. 원격 컴파일/실행 모델의 구현 및 실행 처리 응용

4.1 원격 컴파일/실행 모델의 구현

가. 시스템 핵심 컴포넌트



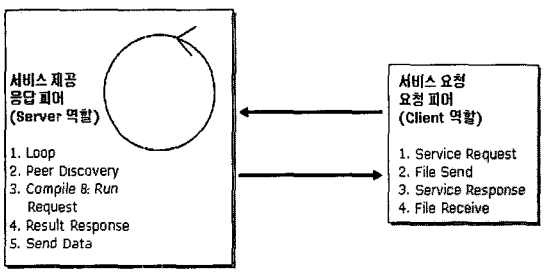
〈그림 5〉 핵심 시스템 컴포넌트 구성도

<그림 5>에서 보듯이 분산 P2P 작업 할당자의 핵심적인 부분은 피어/그룹 매니저 컴포넌트와 작업 처리 컴포넌트와 모니터 컴포넌트로

구성된다. 피어/그룹 매니저 컴포넌트는 피어/그룹을 JXTA 네트워크에서 검색해서 찾아주고 그 정보를 관리하는 컴포넌트이다. 처음 시작부터 JXTA 네트워크에 연결하도록 설정하고 그룹을 먼저 검색한 다음 그 그룹에 속한 피어들을 찾는 역할을 수행한다. 작업 처리 컴포넌트는 로컬/원격에서 작업을 처리하도록 한다. 각각 로컬과 원격에서 컴파일과 실행 처리를 수행하고 이때 처리된 결과는 반환되어 결과 서비스에서 관리되도록 한다. 모니터 컴포넌트는 피어의 시스템 상태정보 테이블과 작업상태 테이블을 주기적으로 관리하는 컴포넌트이다. 모니터링을 통한 상태정보 테이블들은 작업 할당을 위한 피어 지정 및 동적인 작업 관리를 위해 사용된다.

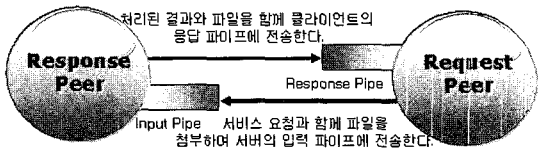
나. 서비스 제공과 요청 클래스

응답 피어는 요청 피어의 컴파일 및 실행 서비스 요구를 감지하기 위한 데몬을 띄워 대기한다. 요청 피어에서는 어떤 작업을 처리할지를 요청하며, 서비스가 가능한 경우 자바나 클래스 파일을 함께 첨부하여 응답 피어에게 보낸다.



〈그림 6〉 요청 피어와 응답 피어의 서비스 구성도

요청 피어와 응답 피어 간의 요청과 응답 처리는 <그림 7>과 같이 입력 및 응답파이프를 이용하여 처리한다. 파이프란 JXTA에서 사용하는 가상채널로 실제 데이터 송/수신을 담당하는 부분이며, 서비스 요청과 처리 시에 필요한 자바 소스와 바이트 코드를 전송할 때 사용된다.



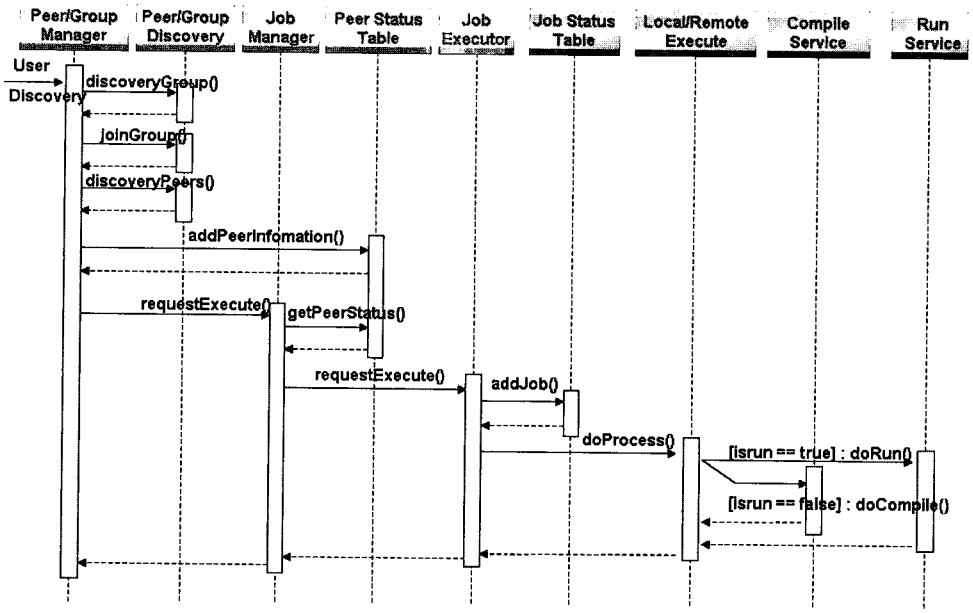
〈그림 7〉 파이프를 이용한 요청 피어와 응답 피어 간 데이터 전송

다. 컴파일과 실행 클래스

응답 피어에는 컴파일과 실행의 처리를 위해서 사전에 표준 JVM이 설치되어 있어야 한다. 처리 과정은 <그림 8>과 같고, 작업 관리자는 실행할 작업과 작업 실행을 위한 피어들의 상태 기준을 정하기 위해 작업처리 구성정보 파일을 작성한다. 이 구성 정보 파일에는 필요한 피어의 개수와 함께 실행 피어의 시스템 상태 기준을 저장한다. 작업을 실행할 피어의 시스템 선택 기준은 CPU 유휴량과 사용 가능한 메모리량이며, 이를 바탕으로 그 기준을 만족하는 피어들이 선택되고 작업의 할당이 자동으로 실행된다. <그림 8>에서 보듯이 작업처리 관리 클래스는 피어 상태정보 테이블을 획득한 후, 그 중에서 작업할 피어들을 선정하고 그들의 정보를 가지고 있는 상태에서 작업 처리기에 실행 요청을 한다. 작업 처리기는 런타임 처리를 위한 명령어를 작성하고, 소스 코드를 받아 컴파일과 실행을 할 것인지 또는 바이트 코드를 받아 실행만 할 것인지를 결정한다. 또한 작업할 피어를 로컬 피어로 할 것인지 아니면 원격 피어로 할 것인지 여부를 결정하고, 작성된 명령어와 파일(자바 소스, 바이트코드)을 전송하여 처리한다. 수신된 작업은 응답 피어에서 실행되고 실행된 결과 또는 에러 정보는 요청 피어에게 반송된다.

라. 피어/그룹 관리 클래스

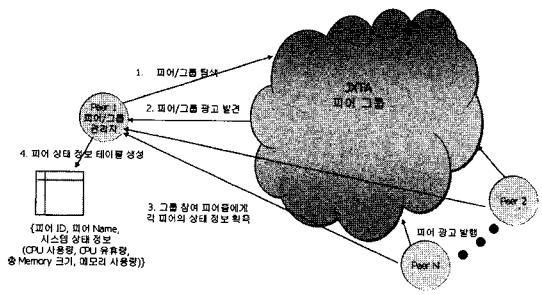
피어/그룹 관리 클래스는 본 논문에서 제시한 플랫폼을 기반으로 동작하는 피어들을 관리



〈그림 8〉 작업 실행 순서도

하기 위한 클래스이다. 피어/그룹 관리 클래스는 첫 작업으로 피어 그룹을 생성하거나 검색한다. 피어 그룹은 처음부터 정해진 특정 이름으로 생성해야 한다. 그렇지 않을 경우에는 동일한 그룹으로 판단하지 않고 계속해서 각 피어마다 그룹을 생성할 수 있기 때문이다. 피어 그룹을 검색해서 찾은 경우 피어는 바로 그 그룹에 참여할 수 있고, 그룹에 참여하면 바로 자신의 피어 정보를 피어 그룹에 있는 다른 피어들이 알 수 있도록 광고한다.

피어/그룹 관리 클래스는 <그림 9>에서 보듯이 그룹에 참여한 피어들의 상태 정보를 획득하고 이를 피어 상태정보 테이블에 저장한다. 피어 상태정보 테이블은 모니터링 서비스를 위해서 생성되고 유지된다. 처음 피어의 상태 정보를 획득한 후, 각 피어들의 ID와 이름과 같은 기본정보는 변경되지 않지만 피어의 시스템 상태 정보는 주기적으로 변경된다. 따라서 일정한 시간 간격으로 피어들의 시스템 상태 정보를 획득할 필요가 있다. 이 간격은 적당한 시간으로 정해야 한다. 혹시라도 간격이 너무 크면 시스템의 변경된 상태를 바로 알 수 없기 때문에 무리하게 작업을 할당하는 경우가 발생할 수 있고, 반면에 간격이 너무 작으면 자주 상태 정보를 전송해야 하기 때문에 네트워크 부하량을 증가시켜서 네트워크 성능을 저하시킬 수도 있다.



〈그림 9〉 피어/그룹 관리자와 피어/그룹 정보 관리 구성도

마. 작업처리 관리 클래스

작업처리 관리 클래스는 컴파일과 실행 작업을 처리하고 작업의 상태 정보를 관리한다. 작

업 처리 관리 클래스는 작업의 상태 정보를 관리하기 위해서 작업상태 테이블을 생성하고, 이 정보는 모니터 클래스에서 사용된다. 작업 상태 정보는 작업 시작, 처리, 종료 형태로 저장된다.

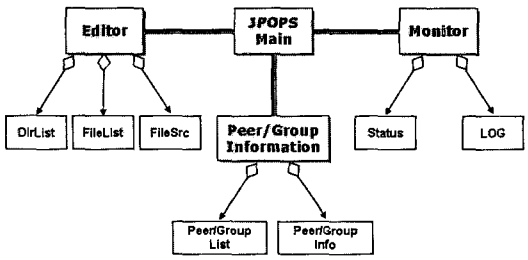
모니터 클래스는 피어 상태정보 테이블과 작업상태 테이블을 이용한다. 모니터 클래스는 피어들의 시스템 상태 정보를 주기적으로 획득하여 피어 상태정보 테이블을 업데이트한다. 작업처리 관리 클래스는 모니터 클래스를 이용하여 피어의 변경된 시스템 상태 정보를 파악하고, 이 정보는 해당 피어의 사용 가능 여부를 판단하는 기준이 된다. 작업처리 관리 클래스는 이 정보를 바탕으로 작업할 피어의 선정과 작업 할당을 자동으로 수행할 수 있다.

작업의 할당과 실행은 작업처리 구성정보 파일에 기술된 기준을 만족하는 피어들을 선택하여 처리된다. 작업처리 구성정보 파일에는 필요한 피어의 개수와 실행 피어의 시스템 상태 기준이 저장되어 있다. 예를 들어 작업의 종류는 "Java", 작업 파일은 "Crawler.class", 필요한 피어의 개수는 8개, 실행 피어의 시스템 상태 기준으로 CPU 유휴량은 90%이상 그리고 사용 가능한 메모리 크기는 50M이상으로 지정할 수 있고 이 조건을 만족하는 상태가 5분 이상 지속될 경우 사용 가능한 유휴 자원으로 판단한다. 이렇게 작업 처리의 기준이 지정되면 작업처리 관리 클래스는 피어 상태정보 테이블에서 그룹에 참여한 피어들의 상태 정보를 획득하고, 획득한 정보에서 기준을 만족하는 피어들을 지정한 후, 자동으로 작업을 할당하여 실행시킨다.

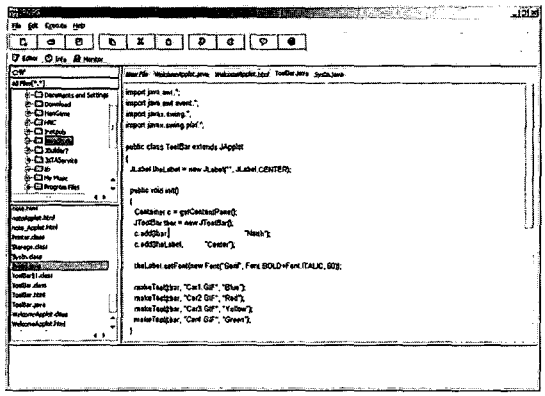
작업이 할당되면 작업 상태 정보는 작업상태 테이블에 저장된다. 작업상태 테이블에는 작업을 수행하는 피어 ID, 작업 종류와 이름, 그리고 작업 상태를 저장한다. 작업상태 테이블은 할당 받은 작업을 실행 중인 피어 시스템의 상태가 변경되었을 경우 동적으로 작업을 중지, 종료시키거나 작업 할당이 되어 있지 않은 다른 피어에게 작업을 재할당할 수 있도록 하기 위해 사용된다.

4.2 구현된 원격 컴파일/실행 모델의 실행 및 응용

JPOPS(Java P2P remote cOmpile Portable System) 는 3개의 중요 기능들 각각을 별도의 클래스로 분리하여 구현하였다. Java Swing으로 작성된 Main 클래스는 이 클래스들 각각을 탭으로 분리하여 선택할 수 있도록 구성되어 있다. 클래스의 구성은 <그림 10>과 같고, JPOPS 메인 화면은 <그림 11>과 같이 왼쪽에는 디렉토리와 파일 리스트 창, 오른쪽에는 파일의 내용을 볼 수 있는 창, 그리고 아래에는 처리 결과를 볼 수 있도록 구성되어 있다.



〈그림 10〉 시스템 화면 구성 GUI 클래스 구성도



〈그림 11〉 시스템 메인 화면

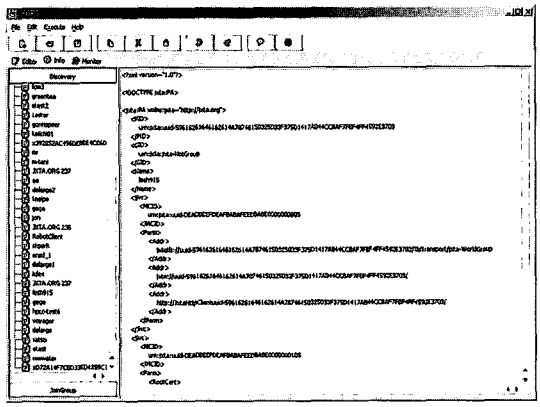
Editor 클래스는 시스템 메인 화면을 처리하는 클래스이다. 현재 시스템의 디렉토리 구조와

각 디렉토리에 들어있는 파일들의 리스트를 보여준다. 그리고 선택 파일의 종류는 모든 종류, 자바 파일, 또는 클래스 파일 등의 기준을 설정하여 필요에 따라 유형별로 지정할 수 있다. 파일리스트에서 내용을 볼 수 있는 파일은 파일의 확장자가 java, html, 또는 txt이어야 한다. 메인 화면에서 보는 것처럼 메뉴 구성은 메모장과 유사하고, 실행(Execute) 메뉴에는 컴파일과 실행 버튼을 분리하여 추가하였다.

PeerInfo 클래스는 현재 알 수 있는 주변 피어에 대한 정보를 검색해서 그 정보를 표시하는 역할을 하고, <그림 12>와 같이 피어 리스트 화면이 구성되어 있다. 피어 리스트 화면에서는 먼저 오른쪽에 Discovery 버튼을 누르는 순간 검색이 시작되며 검색된 결과에는 일단 피어들의 이름만 보여진다. 각 피어의 상세한 정보는 <그림 12>과 같이 특정 피어의 이름을 선택하면 그 피어의 실제적인 정보를 담고 있는 피어 광고의 내용이 XML 형태로 표시된다.

Monitor 클래스는 컴파일 및 실행의 진행 단계를 요청, 파일전송, 처리 중, 결과전송, 그리고 처리완료 순으로 화면 리스트에 표시한다. 또한 현재 시스템의 다른 작업처리 상황들도 각각 텍스트로 표시되어 로그 창에 표시된다.

컴파일과 실행 명령은 메뉴에 구성되어 있다. 이 메뉴를 선택하면 처음에 로컬처리와 원격처리를 선택할 수 있는 다이얼로그 창이 나온다. 이 화면에서 로컬을 선택하면 바로 처리되어 그 결과를 확인할 수 있지만, 원격처리를 선택하면 다음 단계로 진행된다. 원격처리를 선택했을 경우에는 연결 가능한 서버의 파이프 정보를 가지고 있는 다이얼로그 창이 표시되고, 여기서 선택버튼을 누르면 처리하고자 하는 파일이 원격으로 전송되고 그 처리가 완료되면 <그 결과가 표시된다. 마지막으로 처리된 결과는 그림 13>과 같이 화면에 표시된다. 컴파일의 경우에는 단순히 그 완료와 에러 여부, 실행의 경우에도 마찬가지로 그 완료와 처리 결과 그리고 에러 여부가 화면에 표시된다.

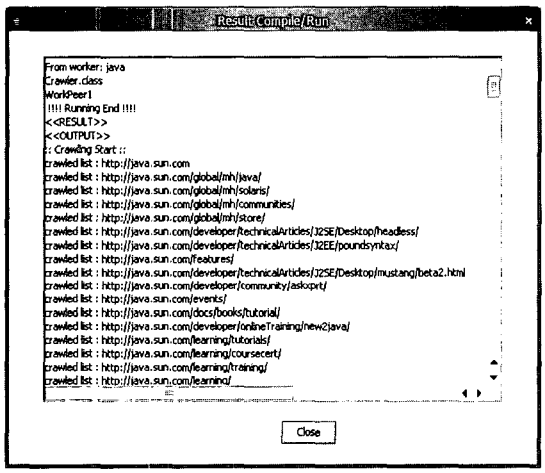


〈그림 12〉 피어 리스트와 각 피어의 정보 화면

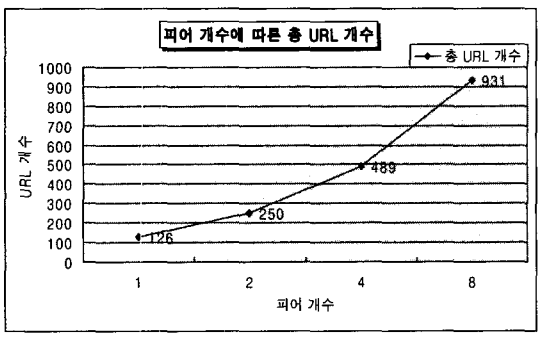
본 논문에서는 구현된 P2P 기반 작업 관리 할당자의 유용성을 보여 주기 위해서 검색엔진이 인터넷 상의 방대한 데이터를 수집하기 위해 사용하는 크롤러를 P2P 분산 처리 환경에 적용하여 보았다. 일반적으로 크롤러는 구글이나 네이버 등의 대규모 검색엔진 시스템에서 전체 컴퓨팅 작업의 절반 정도를 차지할 정도로 많은 컴퓨팅 자원을 요구하는 것으로 알려져 있다. 하지만 크롤러의 기능은 시간적인 제약사항이 약하고 "Divide and Conquer" 형태의 분산처리에 적합하기 때문에 P2P 환경에서 유휴 컴퓨팅 자원을 활용한다면 별도의 전용 컴퓨팅 자원 없이 방대한 데이터를 수집할 수 있다.

본 실험은 컴퓨터 실습실에 있는 30대의 컴퓨터에 환경을 구축하고 그 중 19명의 학생들이 컴퓨터를 사용하고 있는 환경에서 자동으로 유휴 자원들을 찾아서 크롤링하는 자바 프로그램을 전달하고 컴파일 시켜 정해진 시간 동안 실행 처리한 결과를 수집하는 방식으로 실시했다. 즉, 피어의 개수를 증가시키면서 크롤링 처리한 총 URL 개수를 계산함으로써 작업 피어의 증가에 따라 처리된 작업량이 비례해서 증가하는 것을 보여 주었다. 피어의 선택은 사용하고자 하는 피어의 개수를 사전에 작업구성정

보에 입력했으며 그에 따라 작업이 자동으로 진행되도록 하였다. <그림 13>은 작업을 요청한 피어에서 원격 피어 작업의 처리 결과로 크롤링된 URL 리스트를 보여주는 화면이다. 화면에서 보여주는 정보는 작업을 처리한 피어의 이름과 처리한 코드의 이름이 먼저 표시되고, 그 다음으로 컴파일 또는 실행 결과를 표시하며, 마지막으로 작업 처리 시 발생한 출력 또는 에러 정보가 표시된다. <그림 14>에서 보는 것과 같이 작업 피어의 개수가 증가하면서 크롤링된 URL의 총 개수가 선형적으로 비례해서 증가하는 것을 볼 수 있다.



〈그림 13〉 크롤링 프로그래밍 작업의 원격실행 결과 창



〈그림 14〉 피어 개수 증가에 따른 크롤링된 총 URL의 개수

5. 결론 및 향후 연구

네트워크와 컴퓨터 기술의 발전과 이들의 연동은 그리드와 같은 고성능 분산 처리 환경을 구축할 수 있는 기반을 제공한다. 또한, 그 만큼 복잡한 데이터 처리와 많은 양의 자원들이 필요한 응용분야들도 등장하고 있다. 현재까지는 이러한 응용들을 처리하기 위해서 고가의 서버를 이용하는 방법이 주류를 이루고 있으나, 앞으로는 보다 저렴하고 가용성이 높은 P2P 기반의 분산처리 방식을 이용하는 것도 또 하나의 대안이 될 수 있다.

본 논문에서 이용한 JXTA 플랫폼은 위에서 언급한 것처럼 P2P 네트워크 환경에서 서비스 개발을 용이하도록 하기 위해 개발된 기술이다. 물론 아직까지는 표준 프로토콜을 기반으로 P2P 시스템들간의 기본적인 개발 플랫폼을 구축하는 단계에 있지만, 전세계적으로 많은 개발자들이 참여하고 있는 JXTA는 오픈 소스 프로젝트로서의 장점을 최대한 살려 급속도로 발전하고 있다.

기존의 전통적인 분산 런타임 시스템들의 경우 대부분이 정적인 마스터 노드와 작업 노드들로 구성되는 구조를 갖기 때문에 분산처리 작업량의 변동에 따라 시스템을 유연하게 동적으로 재구성할 수 없다는 단점을 갖는다. 이에 반해 본 논문에서 구현한 작업 할당 관리자는 기본적으로 P2P 기반 분산 런타임 시스템을 기반으로 한다. 즉, JXTA 플랫폼 상에서 협업 환경을 위해 개발자들 간에 작업 프로그램의 전달과 관리, 그리고 원격 컴파일과 실행 작업들을 수행할 수 있도록 한다. 이 방식은 작업들을 분산처리하기 위해서 필요한 유휴 자원들을 필요한 시점에 확보하여 활용할 수 있는 장점을 가진다. 또한 피어들 각각이 자율 의지로 시스템에 참여/탈퇴가 가능하기 때문에 유연한 동적 협업 그룹을 형성할 수 있다. 이와 더불어 본 논문에서 제시한 모델은 순수 P2P 방식으로 구

성되기 때문에 어떤 피어든지 자신이 마스터가 되어 다른 피어들에게 작업 수행을 요청할 수 있다. 즉, 어느 피어든지 자신에게 할당된 작업을 수행하는 동안 동시에 다른 피어에게 작업을 요청할 수도 있어 응용 작업에 따라 시스템의 구성을 다양하게 할 수 있으며, 그에 따른 협업작업 활용도도 높아지는 장점을 갖는다.

본 논문에서 제안한 시스템은 위에서 언급한 여러 가지 장점을 갖지만 몇 가지 고려사항들이 향후 연구 과제로 존재한다. 첫째, 본 논문의 참조 모델은 자바 언어와 JXTA 플랫폼을 기반으로 동작한다. 따라서 윈도우, 리눅스, 유닉스 등 다양한 운영체제와 이질적인 플랫폼 환경에서도 호환성이 유지되며 동작한다. 그러나 다양한 프로그래밍 환경에는 자바 외에도 C, C++, C#, Perl, Python 등 다양한 언어가 존재하고, 비록 이들 언어들 중 일부는 자바에 비해 호환성이 떨어지는 단점이 존재하지만 이러한 언어들을 추가적으로 지원 가능하게 구성할 필요가 있을 수 있다. 둘째, 이와 더불어 본 논문에서는 제시된 모델의 유용성과 활용 가능성을 보여주기 위해 SPMD(Single Program Multiple Data) 형태의 크롤러 프로그램 작업을 P2P 분산처리 환경에서 모니터링을 통해 선정된 유휴 자원인 피어들을 대상으로 실행시켰다. 하지만 작업 피어의 선정과 작업 할당 등을 위한 간단한 스케줄링이 아닌 보다 복합적인 워크플로우 형태의 스케줄링을 추가하는 방안도 고려해 볼 필요가 있다. 마지막으로, 일반적인 애플리케이션의 컴파일 및 실행을 위해서는 P2P 네트워크를 통해 데이터의 전송이 이루어지기 때문에 전송시 누군가 보안에 위반이 되는 파일을 추가로 전송할 수도 있고, 따라서 이에 따른 보안 정책이 요구된다.

본 논문에서 구현한 시스템은 Korea@Home과 같이 수많은 익명(anonymity)의 자원들로 구성되는 대규모 시스템이 아닌 자원의 소재가 분명한 기업 또는 연구소 내부에서 중소 규모로 사용한다면 특별한 신규 자원의 투자 없이 기존의 컴퓨팅 자원들만으로도 용이하게 분산처리 환경을 구축할 수 있다. 또한 본 시스템은 다양한 유형의 분산 시뮬레이션, 미디어 엔코딩 및 변환, 렌더링, 금융기관의 신상품 개발 및 개인별 투자에 따른 실시간 수익률 산정 등 다양한 분야에서 활용될 수 있을 것으로 기대된다.

참 고 문 헌

[1] Factor, M. Schuster, A. Shagin, K. "A distributed runtime for Java: yesterday and today", *Parallel and Distributed Processing Symposium*, April 2004.

[2] W. Yu and A. L. Cox. "Java/DSM: A platform for heterogeneous computing", *Concurrency-Practice and Experience*, 9(11):1213 - 1224, 1997.

[3] Y. Aridor, M. Factor, and A. Teperman. "cJVM: A single system image of a JVM on a cluster", *International Conference on Parallel Processing*, pages 4 - 11, 1999.

[4] W. Zhu, C.-L. Wang, and F. C. M. Lau. "JESSICA2: A distributed Java Virtual Machine with transparent thread migration support", *In IEEE Fourth International Conference on Cluster Computing*, Chicago, USA, September 2002.

[5] G. Antoniu, L. Boug´e, P. Hatcher, M. MacBeth, K. McGuigan, and R. Namyst, "The Hyperion system: Compiling multithreaded Java bytecode for distributed execution", *Parallel Computing*, 27(10):1279- 1297, 2001.

[6] R. Veldema, R. A. F. Bhoedjang, and H. E. Bal. "Distributed shared memory management for Java", *In Proc. sixth annual conference of the Advanced School for Computing and Imaging* (ASCI 2000), pages 256 - 264, 2000.

[7] M. Philippsen and M. Zenger. "JavaParty - transparent remote objects in Java". *Concurrency: Practice and Experience*, 9(11): 12251242, 1997.

[8] Y. Sohda, H. Nakada, and S. Matsuoka. "Implementation of a portable software DSM in Java", *In Java Grande*, 2001.

[9] G. Flammia, "Peer-to-peer is not for everyone", *IEEE Intelligent Systems [see also IEEE Expert]* , Volume: 16 Issue: 3 , May-June 2001 Page(s): 78-79

[10] Li Gong, "JXTA: a network programming environment", *IEEE Internet Computing*, Volume: 5 Issue: 3 , May-June 2001 Page (s): 88-95

[11] G. Fox, "Peer-to-peer networks", *Computing in Science & Engineering*, Volume: 3 Issue: 3 , May-June 2001 Page(s): 75-77

[12] S. Waterhouse, D.M. Doolin, G. Kan, A. Faybishenko, "Distributed search in P2P networks", *IEEE Internet Computing*, Volume: 6 Issue: 1, Jan.-Feb. 2002 Page(s): 68-72

[13] D. Barkai, "Technologies for sharing and collaborating on the Net", *First International Conference on Peer-to-Peer Computing*, 2001. Proceedings., 2002 Page(s): 13-28

[14] Project JXTA web site. http://www.jxta.org. *Sun Microsystems*. 2001.

[15] Project JXTA, "JXTA v2.0 Protocols Specification", *Sun Microsystems* 2003.

[16] Project JXTA, "Project JXTA: Java Programmer's Guide", *Sun Microsystems* 2005.

[17] Project Seti@Home web site, *http://setiathome.berkeley.edu/*, 2005.

[18] 박찬열외 4인, "Korea@Home; P2P형 인터넷 기반 분산 컴퓨팅", *한국정보과학회지*, 22(3) : 26-34, 2004.

◐ 저 자 소 개 ◑

이 승 하 (SeungHa Lee)
2001년 동국대학교 정보통신공학과(학사)
2003년 동국대학교 정보통신공학과(석사)
2003년~현재 동국대학교 정보통신공학과 박사 과정
관심분야 : P2P, GRID, 검색엔진, 원격실행
e-mail : lesh915@dongguk.edu

김 양 우 (Yangwoo Kim)
1984년 연세대학교 전자공학과(공학사)
1986년 Syracuse Univ. 컴퓨터공학전공(공학석사)
1992년 Syracuse Univ. 컴퓨터공학전공(공학박사)
1992년~1996년 한국전자통신연구원 선임연구원
1996년~현재 동국대학교 정보통신공학과 교수
관심분야 : 분산 그리드 컴퓨팅 시스템, 컴퓨터구조
e-mail : ywkim@dongguk.edu