XPool: 그리드 컴퓨팅 상에서 마스터-워커 어플리케이션을 위한 풀 웹 서비스

권원석*, 한성원**, 김성수*
*아주대학교 정보통신전문대학원
**아주대학교 정보 및 컴퓨터공학부
e-mail : wonseok@ajou.ac.kr

XPool: Pool Web Service for Master-Worker-style Applications on Grid Computing

Wonseok Kwon*, Sungwon Han**, Sungsoo Kim*
*Graduate School of Information and Communication, Ajou University
**Division of Information and Computer Engineering, Ajou University

요 약

웹 서비스(Web Service)와 그리드 컴퓨팅(Grid Computing)은 서로 다른 목적을 위해 개발되고 있다. 하지만 두 개의 기술은 " 동적이며 여러 기관에서 참여하는 가상 조직을 통한 문제 해결과 자원 공유" 와 같은 동일한 문제를 다루고 있다. Computational 웹 서비스(CWS: Computational Web Service)는 그리드 기반 기술로 웹 서비스를 사용하는 새로운 아이디어이다. 현재의 CWS 는 워커와 CWS 사이의 중개 역할을 하는 풀 서비스(Pool Service)를 각자 고유하게 하나씩 소유하고 있다. 그러나 풀 서비스가 특정 CWS 에 소속되어 있다면 자원 사용 측면에서 비효율적이다. 본 논문에서는 CWS 에 독립적이며 일반적인 풀 웹 서비스(PWS: Pool Web Service)와 이를 위한 스케줄링 방법을 제안한다.

1. 서론

웹 서비스는 원격 프로시져 호출(Remote Procedure Call) 기능을 제공하는 새로운 웹 어플리케이션이다. 웹 서비스가 인터넷 상에 배치되면, 또 다른 웹 서비스를 포함하여 어떠한 어플리케이션도 배치된 서비스를 검색하고 호출할 수 있다[1]. 이미 RMI(Remote Method Invocation), CORBA(Common Object Request Broker Architecture), DCOM(Distributed Component Object Model), JINI 와 같은 분산 컴퓨팅을 위한 미들웨어 기술이 소개되었으나 이들 중 어떠한 것도 유일한 표준이 되지 못하였다. 웹 서비스는 규칙적이며 광범위한 접근 인터페이스를 제공함으로써 앞에서 열거했던 플랫폼들을 보완할 수 있다[2]. 웹 서비스의 핵심 기술로써 XML(eXtensible Markup Language), SOAP(Simple Object Access Protocol), WSDL(Web Services Description Language), UDDI(Universal Discovery Description and Integration) 등이 있다. XML 은 웹에서 구조화된 데이터를 이동시킬 수 있는 기술이며, SOAP 은 XML 데이터를 전송할 수 있는 방법을 정의한 프로토콜 명세이다. 또한 SOAP 은 통신 프로토콜로써 HTTP 를 사용하여 RPC 를 수행할 수 있도록 정의한다. WSDL 과 UDDI 는 서비스의 검색을 가능하게 한다.

그리드 컴퓨팅은 네트워크에 연결된 컴퓨팅 자원들을 공유할 수 있도록 하는 통신 서비스의 일종이라

This work is supported in part by the Ministry of Information & Communication of Korea ("Support Project of University Foundation Research<2001>" supervised by IITA).
This work is supported in part by the Ministry of Education and Human Resources Development of Korea (Brain Korea 21 Project Supervised by Korea Research Foundation).
본 연구는 2002 년도 정보보호진흥원 대학 동아리 정보보호활동 지원사업에 의한 결과임.

할 수 있다. 최근에 학계와 산업계에서 그리드 컴퓨팅이 활발히 연구되고 있다. 학계에서 연구된 Globus[3]와 Legion[4]과 같은 툴킷은 빠르게 그리드 시스템을 구성할 수 있도록 해주는 뛰어난 미들웨어이다. 또한 Grid Physics Network(GripPhyN)[5], High Energy Physics and Grid Network(HEPGrid)[6], Particle Physics Data Grid(PPDG)[7]와 같이 특정 어플리케이션을 위한 그리드들도 연구되고 있다. 이러한 그리드 미들웨어와 시스템들은 어떻게 분산된 컴퓨팅 자원들을 하나의 가상 컴퓨팅 시스템으로 구성할 것인가에 대해 초점을 맞추고 있다. 이와 동시에 SETI@HOME[8], Bayanihan[9], CX[10], Javelin++[11], Charlotte[12]과 같이 그리드 컴퓨팅에 대한 접근 방식이 다른 시스템들도 있다. 이들 시스템은 인터넷 상의 발런티어 사용자의 유휴 컴퓨팅 시간을 사용하기 때문에 글로벌 컴퓨팅(Global Computing), 발런티어 컴퓨팅(Volunteer Computing)이라고도 부른다.

웹 서비스와 그리드 컴퓨팅은 서로 다른 목적을 위해 개발되고 있다. 하지만 두 개의 기술은 " 동적이며 여러 기관에서 참여하는 가상 조직을 통한 문제해결과 자원 공유" 와 같은 동일한 문제를 다루고 있다[13]. Computational 웹 서비스는 그리드 기반 기술로 웹 서비스를 사용하는 새로운 아이디어이다.

Bayanihan .NET 은 일반 어플리케이션 지원을 위해 웹 서비스를 사용하여 그리드 컴퓨팅을 수행하는 최초의 시스템이다. 이 시스템은 웹 서비스 형태로 어플리케이션 서비스를 제공한다. 어플리케이션 서비스는 Computational 클라이언트에게 웹 메소드를 제공하여 그들이 마스터-워커 형태의 어플리케이션을 실행할 수 있도록 한다. 그러므로, Computational 클라이언트는 CWS 에서 제공하는 웹 메소드들을 사용하여 손쉽게 어플리케이션을 만들 수 있다. Bayanihan .NET 은 발런티어의 컴퓨팅 자원들에서 수행되는 수많은 작업을 관리할 수 있는 풀 서비스를 제공하며 작업들은 이거 스케줄링(Eager Scheduling) 기법[12]으로 스케줄링된다. 하지만 Bayanihan .NET 은 그들의 논문[9]에서 기술적 문제를 자세하게 기술하지 않았다. 이것은 아직도 CWS 를 구축하는데 필요한 연구들이 많다는 것을 의미한다. 또한 그들의 풀 서비스는 CWS 에 속하기 때문에 자원 사용에 있어서 효과적이지 않다. 풀 서비스에 서로 다른 어플리케이션에서 제출된 수천의 발런티어와 작업들이 존재한다면, 풀 서비스는 리소스 사용을 측면에서 효과적으로 서비스하지 못하게 된다. 왜냐하면 이거 스케줄링은 배치 단위로 작업을 스케줄링하기 때문이다[14]. 그러므로 본 논문에서는 XPool 로 명명한 CWS 에 독립적인 좀 더 일반화된 풀 웹 서비스(PWS)를 제안한다.

2. 마스터-워커 어플리케이션

마스터-워커 컴퓨팅 모델은 몬테 카를로 시뮬레이션(Monte Carlo Simulation), Brute-Force 검색, 이미지 렌더링, 이미지 프로세싱과 같은 문제를 푸는데 적당하다. 이 모델에서 마스터는 그림 1 과 같이 반복적으로 배치를 실행한다. 배치 속의 서로 독립적인 작업 유닛은 사용 가능한 발런티어에게 분배되어 실행되고 결과 값들이 다시 마스터에게 전송된다. 마스터는 전송받은 결과 값들로 적절한 작업을 수행한 후, 다음에 실행될 배치를 생성하게 된다.

그리드 컴퓨팅 상에서 마스터-워커 어플리케이션에서 가장 많이 사용되는 스케줄링 기법은 이거 스케줄링이다. 이 스케줄링 기법은 단지 하나의 어플리케이션만 스케줄 된다면 무리없이 동작한다. 그러나 많은 수의 어플리케이션이 하나의 시스템(예를 들어, PWS)에서 스케줄링된다면 성능 문제가 발생한다. 이거 스케줄링의 특성 상 풀 서비스에 배치가 제출되면 현재 스케줄링되고 있는 배치가 끝나야만 스케줄링이 될 수 있다. 그리고 배치 속의 작업들이 서서히 완료될 때마다 더 많은 워커들이 아직 완료되지 못한 작업에 할당된다. 만약 현재 스케줄링되고 있는 배치 속 작업들의 평균 수행 시간이 길다면, 다음에 실행될 배치는 오랫동안 기다려야 할 것이다. 이것은 자원을 낭비하는 결과를 초래한다. 현재 풀 서비스에서 실행되고 있는 배치를 위한 일부 워커들이 다음 순서의 배치에 할당되어 동시에 스케줄링된다면 좀 더 나은 성능 향상이 기대될 것이다.

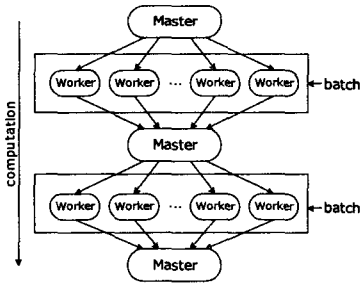

그림 1. 마스터-워커 컴퓨팅 모델

3. XPool: 풀 웹 서비스

CWS 가 자신의 고유한 풀을 가지는 것이 좀 더 관리하기 쉽지만, 여러 개의 CWS 가 하나의 풀서비스를 공유하여 대규모의 발런티어 워커들을 이용한다면 자원 이용율 측면에서 좀 더 효과적일 것이다. 만약 풀을 통합하여 관리한다면 몇가지의 해결해야 할 문제가 발생한다. 첫째, 통합 스케줄러는 동시에 몇 개의 배치를 스케줄링할 것인가에 대한 최적화의 문제이다. 둘째는 스케줄러가 어떠한 방식으로 스케줄링을 할 것인가에 대한 문제이다. 기존의 이거 스케줄링 기법은 풀이 통합된 환경에서는 적합하지 않기 때문이다.

Bayanihan .NET 에서는 그림 2 와 같이 CWS 가 자신의 풀을 소유하고 있으며 *createPool*(), *addWork*(), *getResult*(), *deletePool*()과 같은 웹 메소드를 제공한다.

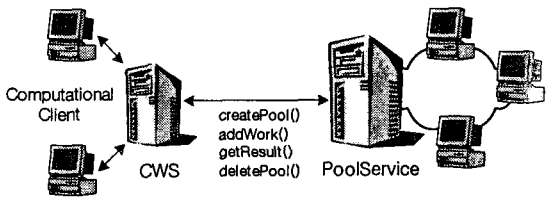

그림 2. Bayanihan .NET 의 구조

본 논문에서 제안하는 XPool 은 CWS 와 워커들 사이의 중개 역할을 하는 것이다. XPool 에서는 모든 것이 XML 로 관리된다. CWS 는 SOAP-Messaging 을 이용하여 XML 로 기술된 작업들로 이루어진 배치를 제출한다. 또한 워커들도 작업을 가져오기 위해 SOAP-Messaging 을 이용한다. 워커들은 XML 문서(작업)를 웹 메소드를 통해 얻고 이것을 파싱하여 적절한 클래스(예를 들어, JAVA 또는 C++ 클래스)들을 파싱된 인자와 함께 실행한다. 워커가 가져야 할 기본적인 프로그램 기능으로써 PWS 로부터 작업을 가져오기 위한 기능, XML 을 파싱하는 기능, 필요한 클래스를 다운로드하는 기능, 결과를 XPool 에게 전송하는 기능들이 있다. XPool 는 다음과 같은 기본 웹 메소드를 CWS 와 워커들에게 제공해야 할 것이다.

CWS 를 위해,
createPool(): create a pool to submit batches or tasks
deletePool(): delete a pool
addBatch(): add batch into the pool
addTask(): add task into the pool
getResult(): get result from the pool

발런티어 워커를 위해,
getWork(): get work from the pool
putResult(): put result into the pool
getClass(): get class files to execute task

4. 스케줄링 기법

본 논문에서 제안하는 스케줄링 기법은 이거 스케줄링의 특성을 유지하면서 어떻게 서로 다른 CWS 에서 생성한 풀들에 워커들을 효율적으로 할당시켜야 하는지를 다룬 기법들이다. 제안된 기법의 설명을 쉽게 하기 위해 우리는 3 개의 CWS, C1, C2, C3 가 있고 각 CWS 가 제출하는 배치의 길이를 100, 50, 10 이며 워커는 총 100 개가 계산에 참여한다고 가정하고 제안 기법을 설명할 것이다.

4.1 오리지날 할당 기법

이 기법은 아래의 2 가지 기법과 비교하기 위한 것이며 기존의 이거 스케줄링과 똑같이 스케줄링하는 것이다. 이 기법에서는 C1, C2, C3 에 각각 34, 33, 33 개의 워커가 할당되며 한 워커가 작업을 끝내고 새로운 작업은 이전에 작업을 할당받았던 풀로 요청한다.

4.2 라운드 로빈 할당 기법

라운드 로빈 기법에서 PWS 스케줄러는 워커가 작업을 요청할 때 라운드 로빈의 형태로 각 CWS 풀에서 작업을 가져와 워커에게 할당한다. 그리고 각각 풀들은 이거 스케줄링에 의해 스케줄된다. 워커가 새로운 작업을 할당 받을 때, 워커는 이전에 할당받았던 풀과는 상관없이 아무 풀에서나 작업을 할당받을 수 있다. 4.1 의 기법과 비교하여 이 기법은 자원(워커)을 좀 더 효과적으로 관리할 수 있다.

4.3 확률적 할당 기법

확률적 할당 기법은 PWS 안에 있는 서로 다른 배치의 길이가 서로 다를 때 사용하기 적당한 기법이다. 이 기법은 각 CWS 의 풀에 배치의 길이에 따라 확률값을 설정한다. 그리고 난 후 스케줄러는 이 확률값에 따라 워커들을 각 CWS 의 풀에 할당하게 된다. 각 CWS 의 풀에 제출된 배치의 길이가 길면 길수록 확률적으로 더 많은 워커들이 그 풀에 할당되게 된다. 예를 들어, C3 에 하나의 워커가 할당될 때 확률적으로 C2 와 C1 은 각각 5 개와 10 개의 워커가 할당되게 된다.

5. 성능 평가

우리는 시뮬레이션 기법을 사용하여 제안된 기법의 성능을 평가하였으며 평균 응답 시간(Average Response Time)과 중복도(Redundancy)와 같은 성능 메트릭(metric)을 얻었다. 우리는 시뮬레이션에서 5 개의 CWS 가 PWS 에 자신의 풀을 생성하고 각각 크기가 1000, 700, 600, 300, 100 인 배치를 제출한다고 가정하였다. 각 CWS 가 제출하는 배치의 수와 워커의 수는 시뮬레이션 상에서 변경될 수 있도록 하였다. 컴퓨팅에 참여하는 워커들은 총 5 개의 그룹으로 나누었으며 각 그룹은 동일한 수를 갖도록 하고 각 그룹에 속해있는 워커들은 성능이 모두 같으며 인접 그룹 간의 성능 차이는 100%로 가정하였다.

평균 응답 시간은 각 CWS 가 제출한 배치들이 종료되는 평균 시간으로 정의하였으며 중복도는 배치들의 작업의 총 수와 계산에 참여한 총 워커의 수의 비율로 정의하였다. 이것은 어떤 계산에 사용된 워커의 수가 많으면 많을수록 자원 이용율이 떨어진다는 것을 의미한다.

그림 3 은 워커의 수와 배치의 수를 변화시켰을 때 3 가지 기법의 평균 응답 시간을 나타낸다. 그림을 통해 라운드 로빈 할당 기법이 다른 2 개의 기법보다

성능이 좋다는 것을 알 수 있다. 확률적 할당 기법도 오리지날 할당 기법보다 전체적으로 성능이 좋다는 것을 알 수 있지만 각 CWS 에서 제출하는 배치의 수가 크고 워커의 수가 적을 때는 성능이 약간 떨어진다는 것을 그래프를 통해 알 수 있다.

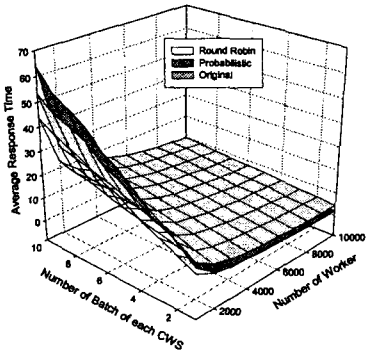

그림 3. 평균 응답 시간

그림 4 는 워커의 수와 배치의 수에 따라 제안된 기법들의 중복도 변화를 나타낸다. 확률적 할당 기법이 가장 좋은 성능을 나타내며 라운드 로빈 할당 기법 또한 오리지날 할당 기법보다 성능이 좋다. 이것은 제안된 2 가지의 기법이 오리지날 기법보다 효과적으로 자원을 관리한다는 것을 의미한다.

라운드 로빈 할당 기법과 확률적 할당 기법 사이에는 Trade-off 가 존재한다. 라운드 로빈 할당 기법은 좋은 성능 향상을 보여주는 반면 확률적 할당 기법은 자원 관리 측면에서 개선된 성능을 제공한다.

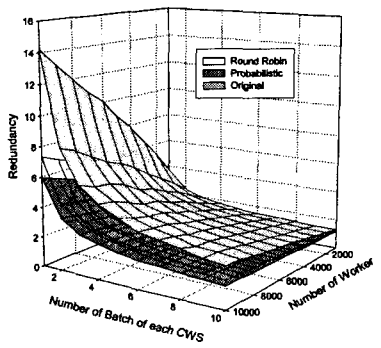

그림 4. 중복도

6. 결론

본 논문에서는 CWS 의 풀서비스를 하나의 풀 웹 서비스로 통합하는 방법과 풀 웹 서비스를 위한 2 가지의 스케줄링 기법을 제안하였다. 시뮬레이션의 결과는 제안된 기법들이 기존 이거 스케줄링 기법보다 성능이 좋다는 것을 보이고 있다. 우리는 그리드 컴퓨팅을 위한 CWS 와 워커들 사이의 중개에 관한 많은 연구가 필요하다고 생각되며 향후 우리는 제안된 풀 웹 서비스를 구현할 것이다.

참고문헌

[1] D. Tidwell, Web Services – The Web's Next Revolution, http://www6.software.ibm.com/developerworks/education/wsbasics/index.html

[2] V. Vasudevan, A Web Services Primer, http://www.xml.com/pub/a/2001/04/04/webservices/index.html?

[3] Globus Project, http://www.globus.org

[4] Legion: A Worldwide Virtual Computer, http://legion.virginia.edu

[5] Grid Physics Network, http://www.griphyn.org

[6] High Energy Physics and Grid Networks, http://www.hepgrid.org

[7] Particle Physics Data Grid, http://www.hepgrid.org/

[8] SETI@HOME, http://www.ssl.berkeley.edu

[9] Bayanihan Computing Group, http://www.bayanihancomputing.net/

[10] CX project, http://www.cs.ucsb.edu/projects/cx/index.html

[11] M. Neary, S. Brydon, P. Kmiec, S. Rollins, and P. Cappello, "Javelin++: Scalability Issues in Global Computing," Concurrency: Practice and Experience Vol. 12, pp. 727 - 753, 2001.

[12] A. baratloo, M. Karaul, Z. Kedem, and P. Wyckoff, "Charlotte: Metacomputing on the Web," In Proceedings of the 9th Conference on Parallel and Distributed Computing Systems, 1996.

[13] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," International J. Supercomputer Applications, Vol. 15, No. 3, 2001.

[14] L. Sarmenta, "Volunteer Computing," Ph.D. Thesis, MIT, 2001.