SOAP기반의 분산처리 스케줄링 프레임웍: pyBubble

이동우°, 최지현, 성기문, R.S.Ramakrishna
광주과학기술원 정보통신공학과
{leepro°, jhchoi80, gimun, rsr}@kjist.ac.kr

SOAP-based Distributed Processing Scheduling Framework: pyBubble

DongWoo Lee° , Jihyun Choi, Gimun Sung, R.S.Ramakrishna
Department of Information and Communication,
Kwangju Institute of Science and Technology

요 약

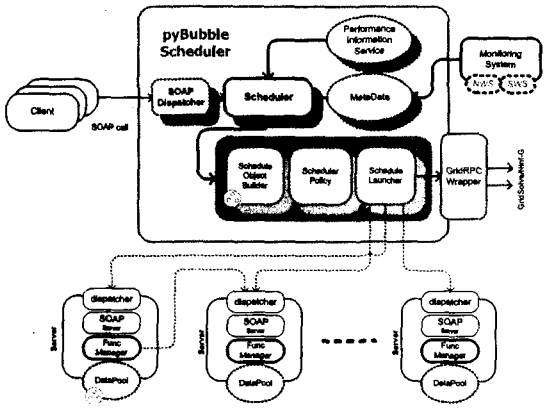
본 논문은 웹 서비스 프로토콜인 SOAP기반의 병렬처리 프레임웍인 pyBubble의 설계와 구현에 관한 것이다. 그리드 어플리케이션 프로그래밍의 어려움을 덜기 위해 그리드 미들웨어로부터의 복잡성에 투명성을 제공하는 것을 본 논문의 목표로 한다. 이는 RPC스타일의 프로그래밍 인터페이스를 지원하면서 파이썬 스크립트 언어의 이식성과 확장성을 통해 기존 병렬처리 어플리케이션의 그리드화와 다양한 자원 스케줄링을 연구 할 수 있도록 하는 스케줄링 프레임웍이 주요 기능적 요소이다. 병렬처리를 위해 비동기 SOAP과 이를 이용한 Task-Farming과 DAG기반의 스케줄링의 지원함으로써 고성능의 그리드 계산환경을 제공하고자 한다.

1. 서론

계산 그리드를 구축하기 위해 다양한 미들웨어의 연구개발로 가상 기관들간의 자원을 공유할 수 있는 토대가 마련되었다. 이는 원격지 자원사용의 보안적 안전성을 가지며 공유된 자원을 활용할 수 있도록 하는 다양한 레벨의 그리드 도구들의 조합으로 가능하게 되었다. 앞으로의 계산 그리드 환경에서 보다 편리하고 빠른 어플리케이션의 개발은 기존의 그리드 미들웨어를 기반의 PSE(Problem Solving Environment)와 같은 통합적 문제해결 환경으로 발전하고 있다. 또한, 그리드 미들웨어도 Globus 3.0의 OGSA[1]와 WS-Resource Framework (WSRF)[2]등의 웹 표준기술을 이용한 서비스 기반의 그리드 아키텍쳐로 발전하고 있다. 이에 비해 실제 사용자가 사용하는 프로그래밍 도구/인터페이스의 발전은 기존의 시스템의 그리드화를 통해 이루어지는 것이 대부분이다. 따라서 사용자로 하여금 기존 그리드 미들웨어의 복잡도에 투명성을 제공함으로써 독립적으로 어플리케이션을 작성 하도록 하는 것이 중요하다 하겠다.

본 논문은 사용자에게 그리드 어플리케이션의 작성에 있어 친숙한 RPC기반의 인터페이스를 제공하면서, 그리드 환경에서 병렬/분산처리를 위한 필수적인 기능적 요소들을 제공함으로써 기본적인 그리드 환경에 실장이 가능한 pyBubble[9,10]의 설계와 구현에 관한 것이다. pyBubble은 실제 자원을 효율적으로 사용하기 위한 스케줄링에 초점을 두어 다양한 스케줄링 알고리즘을 실험 할 수는 프레임웍을 제공한다. 또한, 그리드 어플리케이션의 작성에 있어 표준API인 GridRPC[5]와의 연계로 이질의 그리드 PSE에도 쉽게 적응하는 구조를 가지도록 한다.

본 논문의 구성은 2장에서 pyBubble의 설계와 구현에 대해 소개하고, 3장에서는 pyBubble을 이용한 성능결과에 대해 간단히 언급한다. 4장에서는 관련연구에 대해 소개하고, 끝으로 5장에서 결론과 향후 계획으로 글을 맺는다.



[그림 1] pyBubble 프레임웍

2. pyBubble: 설계 및 구현

그리드 인프라를 기반으로 실제 스케줄링 가능한 어플리케이션을 작성하기 위한 프로그래밍 런타임 시스템으로 pyBubble은 다음과 같은 특징을 가지도록 설계/구현되었다. (그림1은 pyBubble의 구조도 이다.)

2.1 프로그래밍 인터페이스

사용자에게 친숙한 RPC (Remote Procedure Call)의 프로그램 인터페이스를 제공해 개발을 용이하게 한다. 내부적으로 계산 서버와의 인터페이스를 위하여 플랫폼.중립적인 표준 웹 기반 인터페이스인 SOAP을 이용한다. 이는 런타임 시스템을 단순화시켜 이질적 그리드 미들웨어와의 연계를 자연스럽게 유도한다.

2.2 확장성/이식성

런타임 시스템은 스크립트 언어인 Python으로 기존 코드의 재활용과 빠른 원형 생성에 초점을 두었다. 또한, 기존의 NetSolve나 Ninf와 같이 IDL(Interface Definition Language)가 필요 없이 네트웍에서 사용 가능한 Wrapper 를 SWIG[3]와 같은 도구를 사용하여 자동 생성하도록 한다.

2.3 성능정보 서비스

기존의 그리드 미들웨어(Globus등) 또는 어플리케이션 레벨의 스케줄러(AppLeS[8]등)와 같이 CPU, Network의 성능정보 서비스로 NWS(Network Weather Service)를 사용하도록 한다. 추가로 디스크 I/O성능보장을 위한 스케줄링을 위해 SWS(Storage Weather Service)를 지원한다. 이는 보다 다양한 스케줄링 정책을 위한 것이다.

2.4 병렬처리 서비스

기존의 SOAP기반의 시스템과 달리 비동기의 SOAP을 지원해 보다 효율적인 RPC를 제공한다. 비동기 SOAP을 통해 분산작업에 필수적인 Task-Farming과 같은 병렬성 제공이 가능해 진다.

2.5 스케줄링 서비스

성능정보 서비스를 통해 가용한 자원의 현재 상태를 파악하여 유동적인 스케줄링을 제공한다. 이는 스케줄 객체 빌더(Schedule Object Builder)를 통해 요청한 작업과 자원의 일치성(Affinity)을 고려한 스케줄 객체를 만든다. 스케줄러 정책(Scheduler Policy)을 통해 필터링된 후보자원들로 압축되어 실제 계산서버에 작업을 보내는 스케줄 실행기(Schedule Launcher)에 의해 각 서버에 보내진다. 특히, pyBubble은 어플리케이션에 대한 DAG(Direct Acyclic Graph)[10]기반 코딩을 통해 어플리케이션의 작업흐름에 따른 자원의 효율적인 스케줄링 실행환경을 제공한다.

2.6 SOAP 성능 최적화

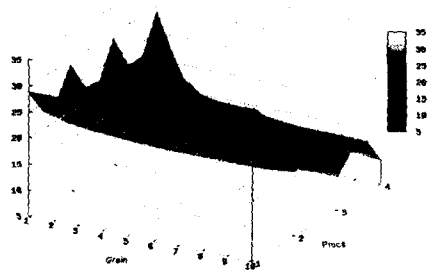
SOAP이 기본적으로 웹 서버를 기반으로 한 Stateless RPC이기 때문에 이때 생기는 고장을 감내하기 위해 각 계산 서버에 데이터 영속성을 제공함으로써 어플리케이션의 실행 중 계산 서버의 고장으로 인한 어플리케이션의 재시동 시에 대규모의 자료 재전송 및 반복 계산을 피하고 이미 계산된 결과를 이용하여 반응시간을 줄이도록 설계한 재시동 프로토콜 (Restart Protocol)[10]을 구현했다. 또한, SOAP의 XML처리로 인한 통신상의 오버헤드를 줄이기 위해 XML문서에 대한 압축을 통해 전송될 자료의 양을 현격히 줄여 효율을 높였다.

2.7 이질의 그리드 서비스와 연계:

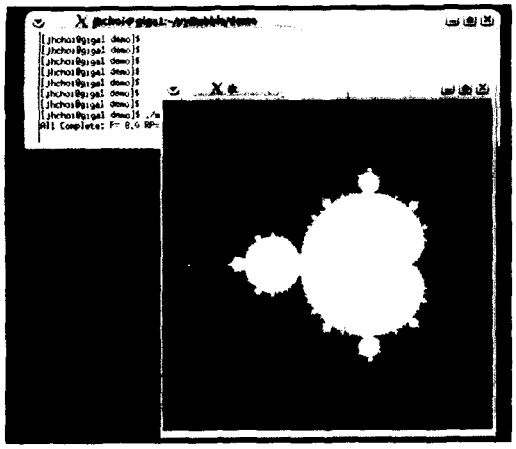
이질적인 그리드 미들웨어와의 연계를 위해 RPC인터페이스 기반의 NetSolve/GridSolve[4]나 Ninf-G[6]와 같은 시스템과의 연계를 위해 GridRPC[5]의 API로 매핑을 통하여 호환성을 제공한다. 이는 pyBubble의 SOAP Call을 GridRPC의 호출 파라미터 스택으로 재조합하여 가능해진다. 또한, WSDL의 제공으로 특정 기능의 SOAP서비스 연계가 가능하다.

3. 성능 테스트

pyBubble의 테스트 어플리케이션으로 결과의 정확성과 스케줄링의 효율성을 확인하기 위해 Mandelbrot Set 계산(임계치:2^15)을 이용하였다. 이는 Task-Farming 어플리케이션으로 구현되었다. 테스트로 사용한 스케줄링 방법은 1대1 매핑, 1개 쓰레드 1-Greedy 스케줄링과, n개의 멀티쓰레드를 이용한 n-Greedy 스케줄링을 이용하였다. 클라이언트의 병목현상을 고려한 n-Greedy 스케줄링이 좋은 성능을 보였다. 지면 관계상 자세한 설명은 생략한다.(참고[10])



[그림 2] 600x600 Mandelbrot Set 계산 성능



[그림 3] Mandelbrot Set 시각데모

그림 2는 한국(KJIST), 미국(UCSB), 일본(Doshisha대학)에 흩어져 있는 pyBubble 그리드 테스트베드에서 10개의 계산 자원을 이용한 600x600의 Mandelbrot Set계산의 Task-Farming 결과이다. 결과의 3차원 그래프는 grain크기와 사용된 자원의 개수를 달리 하면서 실험한 것이다(10x4=40가지의 실험결과). X축으로 사용된 grain 크기(예를 들어 5일 경우 5x5의 격자로 600x600의 영역을 나눈 것)와, Y축으로 n-Greedy 스케줄링에서 사용된 자원의 개수이다. Z축은 시간(초)이다. 최대 4개의 자원이 동시에 이용되었다. 이는 10개의 서버 중 성능을 고려해 4개의 서버가 스케줄링 되어 각 grain크기의 작업을 Task-farming을 이용해 동시에 계산하는 형태이다. 그래프로부터 grain크기가 작고(10x10) 동시 실행 프로세서의 수가 클 수록 실행시간이 단축되는 것을 볼 수 있다. 그림3은 각 계산 자원에서 결과로 돌아온 결과값을 시각화를 통해 보여준 것이다. 이는 시작적으로 결과의 정확성을 검증하기 위함이다.

4. 관련연구

RPC프로그래밍을 분산/병렬처리를 위한 통신 인터페이스로 한 다양한 연구가 진행 되고 있다. 특히, RPC기반의 그리드 계산시스템으로 NetSolve/GridSolve와 Ninf-G가 대표적이다. 이들 시스템의 경우 바이너리 기반의 런타임 시스템이기 때문에 확장성과 이식성에 한계가 있고, 인터페이스를 정의해주어야 하는 번거로움이 있다. pyBubble에서 API기반으로 제공하는 DAG스케줄링[10]과 달리 TrelliDAG[7]의 경우 RPC프로그래밍 인터페이스를 지원 하지 않고 개별 프로그램의 실행과 연관성을 스크립트로 지정해야 하는 번거로움이 있다.

5. 결론

본 논문은 웹 인터페이스인 SOAP을 이용하여 스케줄링이 가능한 그리드 어플리케이션 런타임 시스템인 pyBubble의 설계와 구현을 소개 하였다. 그리드 어플리케이션의 개발이 용이한 RPC기반으로 표준 웹 서비스인 SOAP을 자원간의 인터페이스로 시스템의 통합과 분리가 용이해진다. 이는 기존의 그리드 미들웨어의 복잡성에 투명성을 제공하여 어플리케이션의 개발과 그 스케줄링에 초점을 맞추도록 한다. 현재 pyBubble을 기반으로 다양한 성능정보 서비스를 통해 스케줄링의 효율성을 높이는 연구가 진행중이다.

감사의 글

본 연구는 광주과학기술원 BK21 정보기술 사업단의 지원에 의한 것입니다.

참조논문

[1] OGSA, http://www.gloubs.org/ogsa

[2] WSRF, http://www.globus.org/wsrf

[3] SWIG, http://www.swig.org

[4] Agrawal, S., Dongarra, J., Seymour, K., Vadhiyar, S, "*NetSolve: Past, Present, and Future*," Making the Global Infrastructure a Reality, Berman, F., Fox, G , Hey, A. eds. Wiley, 2003.

[5] Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee and Henri Casanova, *Overview of GridRPC: A Remote Procedure Call API for Grid Computing*, LNCS 2536, pp.274-278, Nov, 2002

[6] Satoshi Shirasuna, Hidemoto Nakada, Satoshi Matsuoka and Satoshi Sekiguchi, *Evaluating Web Services Based Implementations of GridRPC*, HPDC11, July, 2002.

[7] Mark Goldenberg, Paul Lu, and Jonathan Schaeffer, *TrellisDAG: A System for Structured DAG Scheduling*, 9th Workshop on JSSPP, HPDC12, 2003.

[8] Henri Casanova, Graziano Obertelli, Francine Berman, Richard Wolski, *The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid*, SC00, Nov. 2000.

[9] pyBubble, http://pybubble.sourceforge.net

[10] 최지현, 이동우, R.S.Ramakrishna, *Design and Implementation of DAG-based Co-scheduling of SOAP-based RPC*, Technical Report TR-2004-01.