GridRPC 의 DAG 기반 Co-scheduling 을 위한 프로그래밍 인터페이스

최지현, 이동우, 김미옥, R.S.Ramakrishna
광주과학기술원 정보통신공학과
e-mail : {jhchoi80, leepro, mokim, rsr}@kjist.ac.kr

Programming Interface for DAG-based Co-scheduling of GirdRPC

Ji-Hyun Choi, Dong-Woo Lee, Mi-Ok Kim, R.S.Ramakrishna
Dept. of Information and Communication, K-JIST

요 약

이 논문에서는 그리드환경에서 Remote Procedure Call(RPC) 프로그래밍 인터페이스를 위한 메커니즘인 GridRPC 의 성능향상을 위해 DAG 기반의 Co-scheduling API 를 제안한다. 네트워크 상의 통신횟수를 줄임으로써 GridRPC call 의 최적화를 도모하기 위한 프로그래밍 인터페이스와 이를 가능하게 하는 서버구조를 제안한다. DAG 기반의 co-scheduling 은 서버-클라이언트간의 연산에 사용되는 입력값과 출력값들의 흐름을 분석하여 사용자로 하여금 DAG(Directed Acyclic Graph)로 GridRPC call 들을 구성하고 이를 기반으로 GridRPC call 들을 최적화하는 방법이다. 또한, GridRPC 가 Client Interface 이기 때문에 생기는 문제점인 서버간의 지원의 문제점을 SOAP 서버의 Wrapping 을통해 해결한다.

1. 서론

그리드 컴퓨팅 기술이 차세대 컴퓨팅의 기반으로 인정 받고 있음에 따라, 그리드 분야에서 중요한 이슈들이 많이 생겨나고 있다. 자원을 신속하게 제공하는 방법과 최저의 비용으로 계산 경로를 구하는 것은 중요한 쟁점이며 네트워크의 속도보다 훨씬 빠른 비용으로 발전하는 프로세서의 속도로 인해, 데이터 전송은 고성능 컴퓨팅 어플리케이션에서 큰 오버헤드를 가지고 있다. 이 논문에서는 서버-클라이언트간의 통신을 최적화 화여 어플리케이션의 성능을 향상시키기 위해 그리드 미들웨어인 GridRPC[1,5]의 call을 co-scheduling함으로써 얻을 수 있는 성능향상에 대해서 다룬다. GridRPC는 그리드 환경에서 원격 라이브러리 접근과 병렬처리 프로그래밍 모델을 지원하는 미들웨어 인터페이스이며, 대표적인 시스템으로는 Ninf와 NetSolve등이 지원하고 있다. GridRPC는 그리드에서 특정 서버에 명시된 자원을 활용 하거나 여러 개의 서버에 존재하는 다른 파라미터들의 계산이나 다양한 형식의 병렬 처리 프로그램 레벨에서 사용자에게 프로그래밍 인터페이스의 투명성을 제공하면서 네트워크에 산재 되어 있는 컴퓨팅 자원을 사용하도록 한다. 하지만 스케쥴링을 고려하지 않는 RPC의 경우 통신비용에 많은 오버헤드를 가지고 있다. 스케쥴링을 통해서 네트워크 통신에 들어가는 오버헤드를 줄일 수 있는 방법의 하나로 RPC call들을 sequencing하여 통신비용을 줄이고자 하는 것이다. 이것이 바로 GridRPC의 Co-scheduling이며, 이것을 통하여 성능이 향상된 GridRPC를 사용하고자 한다.

2. GridRPC

GridRPC[1,5]는 그리드 환경에서 병렬처리 프로그래밍 모델과 원격의 라이브러리에 기존의 시스템인 Ninf[9]와 NetSolve[10]등에 구애 받지 않고 접근할 수 있도록 클라이언트에게 투명성을 제공하는 프로그래밍 인터페이스이다.

2.1 GridRPC의 특성

그리드 환경에 맞게 GridRPC는 비동기성의 병렬처

리를 위한 하이레벨의 프로그래밍 모델을 제공한다. 프로그래머들로부터 그리드의 불안정성, 불확실성, 동적성 등에 구애 받지 않도록 한다. Coarse-grained call에 대한 지원과 비동기적인 병렬처리 프로그래밍의 다양한 스타일을 지원하여 이미 만들어진 다양한 미들웨어로 구현된 서버 함수들을 수정없이 그리드의 자원으로 이용할 수 있도록 한다. 이와 같이 GridRPC는 개개의 어플리케이션이 분산되는 것뿐만 아니라 그리드에서 분산되고 정확한 계산을 요구하는 컴포넌트들과 같은 높은 레벨의 소프트웨어 등의 기반을 제공할 수 있다.

2.2 GridRPC vs. RPC

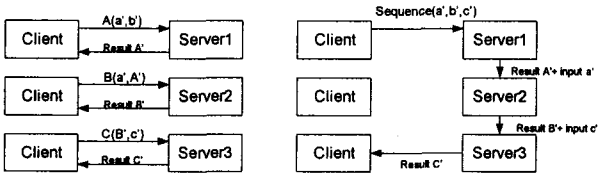
GridRPC는 Grid를 위한 하이레벨의 병렬처리 프로그래밍을 쉽게하는 개념의 인터페이스이며 단일화되지 않은 그리드 컴퓨팅 시스템에 클라이언트가 접근할 수 있는 인터페이스 투명성을 제공한다. 즉, 각각의 계산 미들웨어의 프로그래밍 인터페이스를 몰라도 remote procedure call(RPC) 메커니즘의 인터페이스를 이용하여 접근할 수 있도록 하는것이다. 기존의 RPC와 다르게, loosely-coupled 시스템의 Grid환경에 맞추어진 RPC메커니즘이다. 프로그래밍 인터페이스 측면에서 표준화와 이식성이 부족한 그리드 컴퓨팅에서 널리 쓰일 수 있도록 하기 위해 GGF[11]에서 제안되었다.

3. 문제점

기존의 NetSolve[10]시스템을 통한 request sequencing의 실험결과[2]는 RPC스타일의 시스템에서 최적화된 통신의 성능향상의 가능성을 보여준 예이다. 이 실험은 NetSolve시스템에서 클라이언트와 서버간의 request에 있어 클라이언트로부터 서버에 전달되는 RPC call들을 순차화하여 불필요한 네트워크 전송을 막고 중복된 전송을 제거함으로써 통신 횟수를 줄여 통신 비용을 절감하였다. 이 시스템에서는 위와 같은 과정을 위해서 Direct Acyclic Graph(DAG)를 이용하였다.

DAG[3]은 서로 의존성을 가진 프로세스의 집합으로 구성된 병렬 프로그램을 모델화 하는데 쓰이는 일반적인 방법이다. DAG 에 있는 하나의 노드는 같은 프로세서에서 선점 없이 순서대로 실행되어야 하는 명령들의 집합이 순서대로 있는 하나의 작업을 표현한 것이다. 각 노드는 모든 입력값이 입력되었을때, 실행이 시작 된다. 그 그래프는 작업들 사이의 종속 순서를 표현하는 단방향 edge 들을 가진다. 이 종속 순서는 DAG 의 선행 제약을 의미한다.

본 논문의 목표는 GridRPC 에 네트워크 전송을 줄이고 전체 request 의 response time 을 줄이도록 하기위해 DAG 기반의 새로운 인터페이스를 제안하는 것이다. 이를 위해 불필요한 데이터가 전송되지 않아야 하고 또한 모든 필요한 데이터만이 전송되도록 해야 한다는 것이 목적이다. 일련의 requests 의 입력과 출력 파라미터의 세부적인 분석을 수행하고 각 작업들과 그 작업의 실행 의존성을 표현한 DAG 를 생성하는 과정을 통해서 위와 같은 설계 목표를 이룰 수 있다. 사용자에 의해 생성된 DAG 는 작업의 실행을 위해 request 들이 스케쥴 될 시스템의 서버에 전송되어 실행을 최적화 하기 위한 정보로 활용된다.



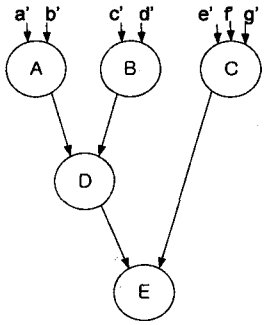
[그림1] Request sequencing을 이용한 클라이언트와 서버간의 데이터 흐름 비교

[그림 1]은 논문[2]에서 예로 제시한 것으로 서버-클라이언트간의 call들을 sequencing하여 Co-scheduling한 것으로 데이터의 흐름이 감소된 것을 볼 수 있다. 왼쪽의 그림은 각 서버에서 계산된 결과값들이 클라이언트에게 전송되어 그 다음 계산을 위해 서버로 보내지지만 오른쪽 그림은 Co-scheduling된 call들의 불필요한 전송을 줄이고 서버간의 필요한 데이터를 전달 함으로써 데이터 흐름이 6 번에서 4 번으로 감소된 것을 볼 수 있다.

4. GridRPC 에서 DAG 기반의 Co-scheduling 을 위한 API 와 Wrapper 서버구조 제안

이 논문에서는 기존의 실험결과[2]에서 본 DAG를 이용한 성능향상과 같이 GridRPC의 DAG 기반 Co-scheduling을 이용하여 통신 비용 절감을 위한 프로그래밍 인터페이스를 제안 한다.

4.1 일련의 call 에 대한 DAG 생성



[그림2] 일련의 GridRPC Call에 의해 생성된 DAG

위의 [그림2] 는 임의의 작업을 위한 일련의 GridRPC call들에 대한 서버간의 데이터 흐름을 표현한 DAG이다.여기서 DAG의 역할은 클라이언트-서버간의 연산의 선행구조를 명시하고 클라이언트가 요청한 일련의 작업을 위해서 서버간의 Call에 대한 입력 데이터와 출력데이터를 분석하여 데이터의 흐름이 각각 다른 서버에서 일어나는 계산의 결과값들을 클라이언트에게 반환하지 않고 중간 결과값이 되는 출력 값들

을 입력으로 필요로 하는 다른 서버들에게 바로 전송할 수 있도록 한다. A와 B노드에서 입력된 a' ,b' ,c' ,d' 값들은 A와 B노드를 통해 A' ,B' 라는 중간 결과를 생성한다. 여기서 A' 와 B' 는 노드 D의 입력값이 되는데 이 A' 와B' 가 노드A와 노드B에서 생성된 결과값이 다시 클라이언트에게 돌아가는 것이 아니라 노드 D의 입력값으로 다른 서버에게로 전달되어 중간 값이 클라이언트로 돌아가는 전송비용을 줄일 수 있는 것이다.

4.2. 사용자 DAG로부터 생성된 GridRPC call

하나의 작업을 위해서 수행되어야 하는 일련의 call들의 시작과 끝을 선언하여 모든 call에 대한 입력 값이 들어온 이후에 실행하도록 한다 일련의 call들이 끝날 때 까지 입력 변수와 출력 변수들은 클라이언트에게 반환되지 않고 서버에 저장되어 있어 있다가 다른 서버의 작업에 사용된다.

여기서 제안될 API는 DAG 의 구성, 수행, 결과확인으로 구성된다.

DAG의 구성

GridRPC_DAG_Begin("DAG콜명"): DAG 콜의 시작을 표시하는 것으로 "DAG콜명" 으로 관리된다

GridRPC_DAG_Call("함수명" ," 결과라벨' , 입력파라미터 1,…,입력파라미터n); 하나의 GridPRC Call을 의미하는 것으로 "함수명" 은 서버의 구현되어 있는 "함수명" 을 의미한다. "결과라벨" 은 입력파라미터들과 지정한 서버함수에 의해 수행된 결과를 의미 하는 심볼이다. 이 "결과라벨" 을 다음에 나오는 DAG Call의 입력 파라미터러 사용하여 DAG를 구성할 수 있도록 한다.

GridRPC_DAG_End("DAG콜명"): "DAG콜명" 의 DAG Call의 끝을 의미한다. 전송이 가능하도록 marshalling한다.

DAG콜의 실행

GridPRC_DAG_Submit("DAG콜명"): 구성된 DAG콜을 전송하고 DAG콜 ID를 반환한다.

결과확인

GridRPC_DAG_Poll("DAG콜명"): DAG콜의 수행의 완료를 알리는 것으로 false와 true값을 반환한다.

GridRPC_DAG_GetResult("DAG콜명" , "결과라벨"): 지정된 "DAG콜명" 과 그에 해당하는 마지막 "결과라벨" 로 DAG콜의 결과를 반환한다.

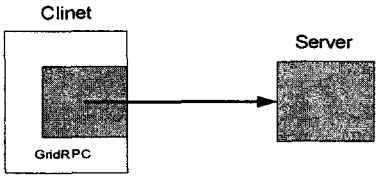
[그림 3]은 [그림 2]를 제안한 DAG콜로 표현한 것이다.

```
GridRPC_DAG_Begin();
GirdRPC_DAG_Call("A'", "A", a', b');
GirdRPC_DAG_Call("B'", "B" , c', d');
GirdRPC_DAG_Call("C'","C", e', f', g');
GirdRPC_DAG_Call("D'","D", A', B');
GirdRPC_DAG_Call("E'","E", D', C');
GridRPC_DAG_End();
```

[그림3] DAG를 이용한 GridRPC의 Co-scheduling을 위한 API

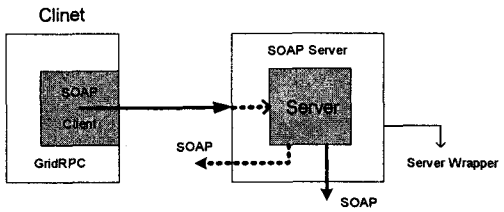
4.3 서버 지원의 문제점

GridRPC를 이용한 Co-scheduling에는 GridRPC가 클라이언트인터페이스라는 특성으로 인한 서버지원의 문제점이 존재한다.



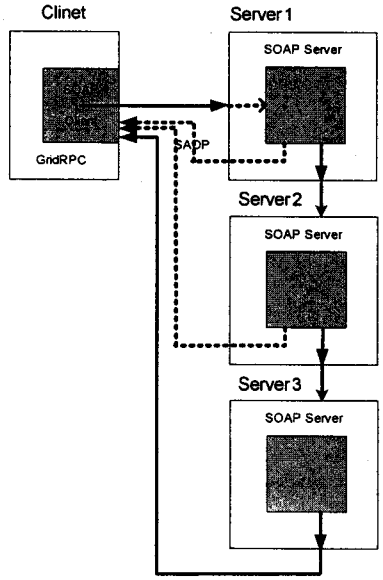
[그림4] GridRPC를 이용한 Server call

GridRPC는 클라이언트 인터페이스이므로 클라이언트와 서버간의 연산이 아닌 서버간의 연산에 있어 클라이언트가 서버간의 수행을 제어할 수 있는 핸들이 부재하다. 그러므로 그러므로 여기서는 서버간의 수행을 위해 서버 Wrapper 를 제안한다.



[그림5] 클라이언트와 서버간의 GridRPC call의 Co-scheudling을 위한 Server Wrapper인 SOAP Server와 SOAP Client

[그림 4]는 기존의GridRPC를 이용해서 클라이언트가 서버에 request call을 할 때 서버-클라이언트 간의 call을 묘사하였다. 클라이언트는 특정 서버에 대해 request call을 할 수 있지만 다른 서버간의 Co-scheduling을 위해서 Server wrapper를 이용해서 접속 하고자 하는 서버간의call을 실행할 수 있다. 여기서 우리는 Server Wrapper로 SOAP 서버를 사용한다. 서버를 둘러싸고 있는 SOAP서버로 인해 클라이언트가 보낸 일련의 call들을 수행할 서버간의 정보를 얻고 그 서버들간의 데이터를 전송하고 데이터의 흐름을 관리 할 수 있도록 한다.



[그림6] SOAP서버를 이용한 서버들간의 연산을 이용하는 GridRPC call

[그림 6]은 SOAP 서버를 이용하여 클라이언트측에서 서버간의 GridRPC Call 을 실행하는 모습을 묘사하고 있다. 클라이언트가 서버로 보낸 DAG 콜은 서버 1 에서 수행된 후 다음으로 전송될 서버로 GridRPC Call 을 이용하여 선정된 다음 서버로 실선의 흐름을 따라 전송된다. 각 서버에서 계산된 결과가 DAG 의 마지막 콜일 경우 점선을 따라 바로 클라이언트에게 리턴된다.

5. 관련 연구

Remote Procedure Call(RPC)의 개념은 오랫동안 분산 컴퓨팅과 분산 시스템에서 널리 사용되어 왔다. RPC는 명확하게 정의 되어 분산 컴포넌트와 통신하는 것이 가능한 명쾌하고 단순한 개념이었다. RPC의 구현에 있어서 유동성과 안정성과 성능에 관한 논의가 계속 되어 오고 있으며 대부분 이전의 연구에서는 고성능 개발에 집중되었다. 반면, GridRPC의 연구는 광대역 네트워크를 통한 loosely-coupled 시스템과 이기종 시스템과의 통신을 목표로 하고 있다. 많은 실험 시스템들이 NetSolve[10]와 Ninf[9]시스템과 관련이 있으며 이 시스템들은 각각의 데스크탑으로 원격 어플리케이션 서버에 request를 쉽게 보내기 위한 그리드 사용자들을 위한 방법을 제공한다. 이 논문은 GridRPC를 이용하는 사용자에게 더 나은 성능을 제공하기 위한 하나의 방법을 제안하고 있다. 그 이외에 GridRPC의 성능 향상을 위해 웹 서비스기반의 GridRPC 구현의 성능을 측정한 연구결과[4]로 SOAP을 이용한 시스템으로 오버헤드를 줄이기 위하여 HTTP Content-length 제거나 Base64 엔코딩 방식을 사용하여 오버헤드를 줄이고 성능을 최적화하는 방법 등에 대한 연구가 있다.

6. 결론

GridRPC는 loosely-coupled 시스템에서 이기종 시스템간의 통신을 요구하므로 서버들과 클라이언트간에 고성능의GridRPC call이 중요하며 서버들간의 통신을 위한 Co-scheduling이 성능향상을 위해 필요하다. 서버들간의 스케쥴링은 DAG를 기반으로 하여 네트워크에서 생기는 불필요한 전송과 중복되는 데이터를 제거하여 최적의 네트워크 전송비용을 가능하게 한다. 클라이언트측의 인터페이스를 가지고 있는 GridRPC에 있어 서버간의co-scheduling을 요구하는 API제공을 위한 문제가 되는 서버간의 지원문제를 SOAP 서버를 이용하여 Server wrapper로 이용함으로써 클라이언트가 서버간의 통신을 자유롭게 할 수 있도록 하고 제안된 DAG 기반의 GridRPC의 Co-scheduling을 위한 서버간의 통신에 있어 사용자에게 투명함을 제공하여 최적의 네트워크 전송비용으로 GridRPC call을 가능하게 한다.

참고문헌

[1] GridRPC Tutorial. http://ninf.etl.go.jp/papers/gridrpc tutorial/

[2] D. Arnold, D. Bachmann, and J. Dongarra. Request Sequencing: Optimizing Communication for the Grid. In A. Bode, T. Ludwig, W. Karl, and R. Wismuller, editors, Euro-Par 2000 – Parallel Processing, pages 1213–1222. Springer-Verlag, September 2000.

[3] Y. Kwok and I.Ahmad. Benchmarking and Comparison of the Tesk Graph Scheduling Algorithm. Journal of Parallel and Distributed Computing, 59(3):381-422, Decemver 1999.

[4] S. Shirasuna, H. Nakada, S. Matsuoka, and S. Sekiguchi. Evaluating Web Services Based Implementations of GridRPC. In *Proc. of HPDC11*, pages 237–245, 2002.

[5] H.Nakada, S.Matsuoka, K.Seymour, J.Dongarra, C.Lee, H.Casanova, "GridRPC: A Remote Procedure Call API for Grid Computing", July 2002

[6] Simple Object Access Protocol(SOAP) 1.1. http://www.w3.org/TR/SOAP, May 2000. W3C Note

[7] XML-RPC. http://www.xml-rpc.com

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/ogsa, January 2002.

[9] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato,and S. Sekiguchi. Utilizing the Metaserver Architecture in the Ninf Global Computing System. In *High-Performance. Computing and Networking '98, LNCS 1401*, pages 607–616, 1998.

[10] H. Casanova and J. Dongarra. Applying NetSolve'snetwork-enabled server. *IEEE Computational Science & Engineering*, 5(3):57–67, July/Sept. 1998.

[11] Grid Global Forum, http://www.ggf.org