그리드 환경에서 효율적 RPC프로그램을 위한 DAG기반의 Co-Scheduling의 구현

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

Implementation of DAG-based Co-Scheduling for Efficient RPC Program in the Grid Environment

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

요 약

본 논문은 그리드 환경에서 RPC 프로그래밍 메커니즘의 성능향상을 위하여 DAG기반의 Co-scheduling시스템의 구현에 관한 것이다. 제안된 Co-scheduling의 목적은 복수개의 관련된 RPC들의 데이터 입출력 관계를 고려하여 불필요하거나 중복되는 네트워크상의 데이터전송을 제거함으로써 실행시간을 줄이는 것이다. 사용자에 의해 만들어진 작업흐름을 DAG로 구성하여 각 작업에 대한 자원을 할당 받아 실행기반 시스템을 통해 수행된다. 이 논문에서는 기존 RPC시스템에서의 오버헤드를 지적하고 그것을 극복하기 위한 DAG기반 Co-scheduling을 설명한다. 실험을 통해 구현된 시스템의 성능향상을 확인한다.

1. 소개

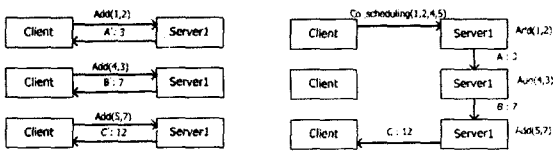
많은 과학계산 어플리케이션들이 고성능의 계산자원과 대용량 데이터 저장장치를 요구함에 따라 그리드 컴퓨팅으로 관심이 집중되고 있다. 그 이유는 이러한 요구들이 더 이상 단일 사이트에서는 충족되지 못하며, 분산되어 있는 자원들을 효과적으로 사용함으로써 어플리케이션들은 더 많은 성능을 가질 수 있기 때문이다. 무엇보다 그리드 자원을 효과적으로 사용하기 위해서는 가용한 자원을 효율적으로 사용하기 위한 스케줄링이 필요하다. 단일한 지역이나 단일 클러스터에서도 물론 스케줄링이 요구 되지만, 그리드와 같은 광범위한 분산환경에서는 모든 자원을 직접 소유하고 관리할 수 없기 때문에 이러한 환경에서 자원의 스케줄링은 더욱 중요 하다. 본 논문에서는 사용자에게 친숙한 RPC 프로그래밍 인터페이스를 그리드에서 사용할 경우 생기는 문제를 지적하고 스케줄링을 통해 데이터의 전송을 최소화하여 전체 작업의 실행 시간을 줄이기 위한 DAG기반의 co-scheduling 의 구현을 제시한다.

2. DAG기반 Co-scheduling 시스템

2.1 동기

일련의 RPC의 실행 시 입력 데이터와 출력 데이터들이 서로 연관성이 있을 때, 즉 하나의 서버에서 생성된 결과가 다른 서버의 입력값이 될때 클라이언트와 서버간의 데이터 전송에 불필요한 전송을 제거하여 RPC의 실행시간을 줄이는 것으로 성능향상을 기대한다.

클라이언트와 서버들 간의 1대1의 RPC만 가능다면 중간 결과값이 클라이언트에게 돌아온 후 클라이언트는 다시 그 중간결과값을 요구 하는 다른 서버에게 재전송해야 하는 오버헤드가 발생한다. DAG기반 co-scheduling을 이용 그러한 전송을 줄여 서로 다른 서버간의 직접적인 데이터 전송이 가능하도록 하는 것이다.



[그림1] Co-scheduling를 통한 데이터 흐름의 비교

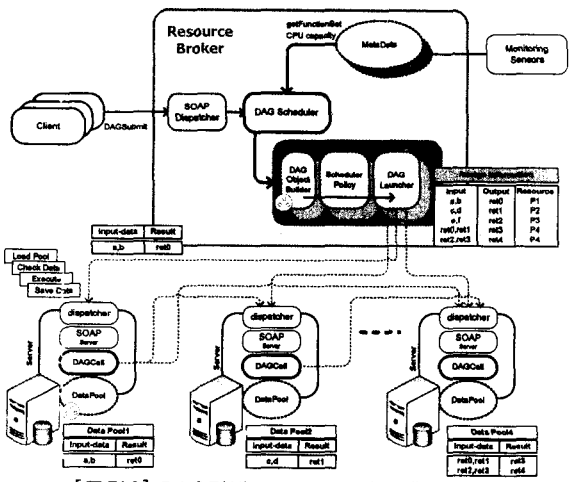
그림1은 co-scheduling을 개념적으로 설명하기 위하여 덧셈연산으로 구성된 일련의 작업들을 두 가지의 경우들로 비교 하였다. 왼쪽 그림은 일반적인 RPC의 경우로 데이터의 흐름이 6회로 클라이언트와 서버와의 단일 인터렉션을 보여준다. 오른쪽 그림의 경우 클라이언트가 서버에게 일련의 RPC를 실행하기 위해 필요한 입력과 정보들을 모두 한번에 전송해주고 클라이언트의 관여 없이 서버들간의 입출력 데이터 전송이 이루어 지며 최종 결과만이 클라이언트로 전송된다. 그리드 환경에서는 로컬네트웍보다 느린 네트웍 성능을 보인다. 이를 감안해 RPC어플리케이션의 최적화는 그림1의 우측과 같은 스케줄링을 통해 RPC실행성능의 향상을 도모 할 수 있다.

2.2 DAG(Direct Acyclic Graph) 모델

일반적인 어플리케이션의 데이터의 흐름은 서로 관련성을 가지고 있으며 그 정보는 Direct Acyclic Graph(DAG)를 통해서 표현된다. DAG는 서로 의존성을 가진 작업들로 이루어진 병렬 프로그램을 위한 일반화된 모형이다[1]. DAG내의 각각의 노드들은 수행해야 할 작업들을 의미하며, 그것으로 구성된 그래프는 각 노드들을 연결하는 방향성을 가진 화살표로 작업들간의 부분적인 실행 순서의 정보를 가지고 있으며 노드간의 작업들은 순환하지 않는다.

3. 구현된 Co-Scheduling 시스템 구조

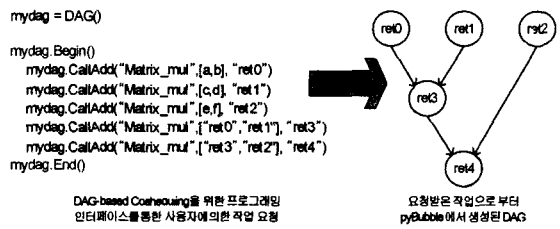
실행 시스템인 pyBubble(그림2)은 RPC 프로그래밍 인터페이스를 제공하는 SOAP기반의 시스템이다. pyBubble 은 대용량의 계산 자원을 이용하는 작업들을 분산된 자원에 할당하여 실행 할 수 있도록 Task-farming[8]과 이 논문에서 제안한 Co-scheduling이 구현된 실행 시스템이다. pyBubble은 클라이언트, 자원 브로커, 그리고 서버자원의 세 부분으로 구성된다. 클라이언트는 작업을 요청하는 사용자 프로그램으로 RPC패러다임으로 어플리케이션을 작성한다. 브로커는 클라이언트로부터 요청 받은 작업들을 서버자원에 적절하게 할당하는 스케줄링 정책을 담당하고 있다. 다음절에서 설명할 API를 통해 클라이언트는 작업을 자원 브로커의 dispatcher에 제출한다. 이를 받아 DAG객체를 생성한다. 가용한 자원의 상태는 자원 모니터링 시스템을 통해 수집되어 스케줄링에 사용한다. DAG객체는 클라이언트에 의해 지정된 DAG를 추상화한 객체로써 계산서버에 데이터와 함께 전송된다. pyBubble에서 비동기의 SOAP을 지원해 보다 효율적인 실행을 제공한다. 생성된 DAG객체를 이용하여 처음 실행 가능한 작업들을 선별하여 DAG Launcher에 의해 병렬실행 된다. 분산된 자원을 직접관리하고 소유하기 힘들기 때문에 요청된 작업에 따라 가용한 자원은 변경 될 수 있다.



[그림2] DAG기반 pyBubble시스템의 구조

3.1 프로그래밍 인터페이스

DAG기반 스케줄링을 위해 클라이언트는 DAG생성을 위한 API를 사용하여야 한다. 이는 RPC와 유사한 형식이다. 그림3은 DAG API를 사용한 예이다.



[그림3] 프로그래밍 인터페이스와 DAG생성

Begin() 과 End()는 클라이언트가 하고자 하는 DAG기반 작업의 처음과 끝을 명시한다. 그 사이에 있는 각 명령들은 DAG의 각 노드로 구성된다.

Begin(): DAG의 생성을 선언하는 것으로 시스템에 DAG의 생성을 위한 초기화를 위한 것이다.

CallAdd(" *Function_Name*" , a, b, " ret1"): 클라이언트가 요구하는 작업들을 DAG의 노드로 구성한다. 클라이언트가 요구하는 작업들은 DAG를 구성하기 위해 관련성을 가진 작업들을 명시적 표현으로 지정한다. 첫번째 파라미터인 Function_Name은 수행될 펑션의 이름으로 서버에 존재하는 이름이어야 한다. 이 펑션의 입력데이터로 a와 b를 명시한다. 이는 로컬 자료형이다. 마지막으로 파라미터 " ret1" 는 그 연산에 해당하는 결과가 저장될 ' 결과심볼' 로 다른 RPC Call의 입력으로 쓰일 수 있도록 명시한다.

End(): DAG단위의 마지막을 선언하는 것으로 DAG객체를 완성한다. 또한, 사용자가 요청한 작업들을 체크하여 오류를 판별한다.

3.2 자원 할당 알고리즘

그림4는 DAG 스케줄링을 위한 pyBubble의 자원 할당 알고리즘으로 계산서버의 현재 성능과 제공 가능한 펑션 리스트를 바탕으로 task노드와 서버와의 매핑이 이루어진다.

4. 실험

RPC의 Co-scheduling으로 인한 성능향상을 보이기 위해 그림3과 같은 데이터의 흐름을 갖는 행렬의 곱셈 연산을 DAG로 구성하여 실험하였다. 행렬의 크기를 2부터 70까지 늘려가면서 각각의 행렬크기에서 결과가 클라이언트로 돌아 올 때까지의 반응시간을 체크 하였다. 그림5는 1대1의 RPC일때의 실행시간과 Co-scheduling통한 RPC의 실행시간을 비교하였다. 실선은 1대1 RPC실행 결과이며, 점선은 co-scheduling한 실행 결과이다.

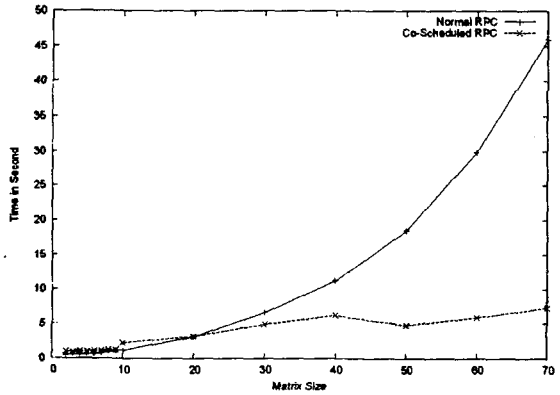
```
DAG Scheduling by Functional Availability:

Dag ← collection of the requests described in DAG
Rmax: the number of resources
ResourcePool ← { R1, R2, ...Ri...Rmax },
Ri ← { F1,F2,...Fn }

Begin
  For each T in Dag
  Do
     CandidateResourceSet ← FindResHavingFunc(T,ResourcePool)
     AssignedRes ← RandomSelect(CandidateResourceSet)
     MappingTaskResource(AssignedRes, T)
  Done
  RootTask ← FindRootTask(Dag)
  Excute(RootTask) // Root Tasks execute in parallel
End

FindResHavingFunc(Task, ResourcePool)
  CandidateSet = { }
  For each R in ResourcePool
  Do
     If Task.Func ⊂ R.func then
     CandidateSet ← R
  Done
Return CandidateSet
```

[그림4] 자원 할당 알고리즘



[그림5] Co-scheduling에 의한 RPC 실행시간

그래프에서 보이는 것처럼 행렬크기의 2x2에서 20x20까지는 스케줄링된 RPC가 근소한 성능 저하를 보인다. 이는 co-scheduling을 위한 서버간의 통신 오버헤드에서 기인한 것이다. 20x20이후의 행렬크기에서 co-scheduling실행시간은 점진적인 상승을 한다. 하지만 1:1 RPC의 경우 데이터의 크기가 커지면서 그에 비례해 통신 오버헤드가 커지는 것을 볼 수 있다. 스케줄된 RPC의 경우 70x70에서 47/7=6.7배의 성능향상을 보인다. 이처럼 데이터의 크기가 커질수록 단위 RPC의 성능은 통신 오버헤드에 의해 급상승 하는 것을 알 수 있다. 연관 계산 작업의 DAG생성으로 병렬처리가 가능하도록 co-scheduling함으로써 반복될 수 있는 통신횟수를 줄여 전체적인 작업 수행 시간을 단축할 수 있다는 것을 이 결과로써 확인할 수 있다.

5. 결론

본 논문에서는 DAG기반의 Co-scheduling을 이용한 RPC시스템을 제안하고 구현에 대해 설명하였다. SOAP 기반의 pyBubble에 구현되어 RPC인터페이스의 그리드 어플리케이션에서 DAG로 구성된 작업이 Co-scheduling 통해 데이터 전송에 이득을 보는 것을 알 수 있었다. 향후 보다 나은 성능 향상을 위해 그리드 환경에서 RPC의 Co-scheduling을 위한 다양한 성능 메트릭의 활용이 요구된다.

감사의 글

본 연구는 광주과학기술원 BK21 정보기술 사업단의 지원에 의한 것입니다.

참고문헌

[1] Y. Kwok and I.Ahmad. *Benchmarking and Comparison of the Task Graph Scheduling Algorithms.* Journal of Parallel and Distributed Computing, 59(3): 381-422, December 1999.

[2] Dorian C. Arnold, D.B., and Jack Dongarra, *Request Sequencing: Optimizing Communication for the Grid.* Proceedings from the 6th International Euro-Par Conference on Parallel Processing, 2000: p. p.1213-1222.

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

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

[5] 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.

[6] 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

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

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