그리드 환경에서 RPC프로그램을 위한 신뢰성있는 DAG기반의 Co-Scheduling을 위한 메커니즘

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

DAG-based Co-Scheduling for RPC Program in the Grid with Dependability

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

요 약

본 논문은 그리드 환경에서 RPC 프로그래밍 메커니즘의 성능향상을 위하여 DAG기반의 Co-scheduling시스템의 신뢰성 향상에 관한 것이다. 제안된 Co-scheduling의 목적은 복수개의 관련된 RPC들의 데이터 입출력 관계를 고려하여 불필요한 네트워크상의 데이터전송을 제거함으로써 전체 RPC의 실행시간을 줄이는 것이다. 사용자로 부터 요청된 어플리케이션을 DAG로 구성하여 실행기반 시스템을 통해 수행된다. 이 논문에서는 신뢰성 향상을 위한 재시동 프토토콜을 구현하여 결함발생시 전체 작업을 정상적으로 완료하기 위한 메커니즘을 소개하고 재시동 프로토콜의 효율성을 검증한다.

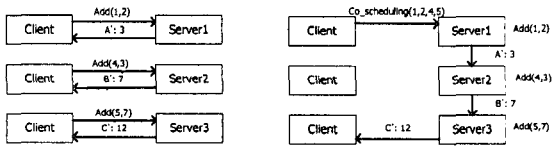
1. 소개

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 인터페이스를와 데이터의 전송을 최소화하여 전체 작업의 실행 시간을 줄이기 위한 DAG기반의 co-scheduling의 신뢰성향상을 위한 재시동 프로토콜의 메커니즘에 대해 설명하고 결함발생시에 재시동 프로토콜에 의한 성능결과를 비교하고 그 장점들에 대해서 알아 본다.

2. DAG기반 Co-scheduling 시스템

2.1 co-scheduling의 목적

DAG기반의 co-scheduling의 목적은 여러 자원을 통한 일련의 RPC들을 실행하는 동안 특정 자원에서 계산된 결과가 클라이언트에게 반환되지 않고, 그 값을 직접 요구하는 다른 자원으로 전달할 수 있도록 하는 것이다. 클라이언트로 부터 작업을 요청받은 브로커가 일련의 RPC들을 co-scheduling하여 각자원에 작업을 할당하고 실행을 진행시킨다 브로커로 부터 스케쥴링된 정보에 따라 모든 작업들이 실행되고 중간 입출력 데이터들이 정해진 자원들에게 전달되어 마지막 결과만이 클라이언트에게 전달된다. 그림1은 co-scheduling에 의해 감소된 네트웍 전송횟수를 보여준다.

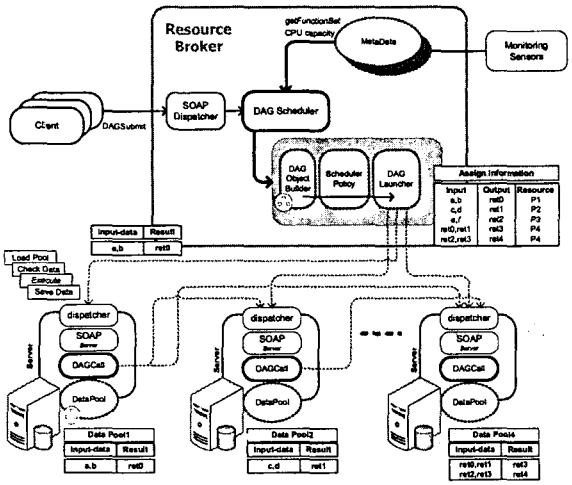


[그림1] Co-scheduling를 통한 데이터 흐름의 비교

2.2 pyBubble 구조

그림2의 실행 시스템인 pyBubble[10]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 Co-scheduling[9]이 구현된 실행 시스템이다. pyBubble은 클라이언트, 자원 브로커, 그리고 서버자원의 세 부분으로 구성된다. 클라이언트는 작업을 요청하는 RPC의 프로그래밍 인터페이스를 이용하여 어플리케이션을 작성하고 API를 통해 자원 브로커의 dispatcher에게 작업을 제출한다. 브로커는 클라이언트로부터 요청 받은 작업들을 서버자원에 적절하게 할당하는 스케줄링 정책을 담당하고 있다.

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 효율적인 실행을 제공한다. 생성된 DAG객체를 이용하여 처음 실행 가능한 작업들을 선별하여 DAG Launcher에 의해 병렬실행 한다. 분산된 자원을 직접관리하고 소유하기 힘들기 때문에 요청된 작업에 따라 가용한 자원은 변경 될 수 있다.



[그림2] DAG기반 pyBubble시스템의 구조

3. 신뢰성 있는GridRPC를 위한 메커니즘

그리드 환경에서 빈번히 생길수 있는 고장의 경우를 조사하고 그 상황에 대처하여 신뢰성 있는 GridRPC를 위한 메커니즘과 구현에 대해서 알아 본다.

3.1 고장을 일으킬수 있는 경우들

① 자원(메모리, 디스크, CPU)고갈: 디스크 공간, 메모리 부족,CPU사용의 과부하등의 이류로 작업의 실행시간이 계속 지연될 수 있다.
② 프로세스 실패: 예기치 못한 상황에 의하여 런타임 시스템이 아닌 프로세스가 죽을 경우에 작업에 대한 응답을 무한히 기다리게 된다.
③시스템 고장: 시스템자체가 외부적인 문제나 관리를 위하여 소유자에의하여 꺼지거나 재부팅 될 수 있다. 이경우 요청된 작업에 의한 결과는 반환되지 않을 것이다.

3.2 재시동 프로토콜

본 연구에서는 위와 같은 경우에 실행중인 시스템을 이러한 결함으로 부터 보호하고 시스템을 정상적으로 작동하기 위한 재시동 프로토콜을 구현하였다. 그림3은 타임아웃 폴링 기반의 검출 알고리즘의 슈도코드이다.

```
ExecuteTask()
Begin
   If CheckallInput(T)
       execution(T)
   Else
       PollCheckThread(T)
End

PollCheckThread(T)
Begin
   While flag==0:
   Do
     If CheckCanRun(T)
       Flag= 1
     Else
       If Tout<Telapsed
         StatusParent = CheckParentSatus(T)
         FaultDetecttion(T, Status)
       Else
         Thread.sleep(10)
   Done
End
```

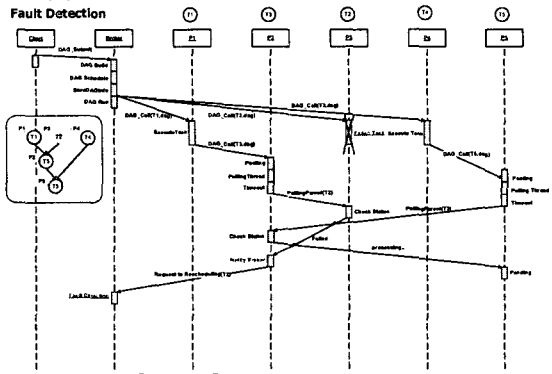
[그림 3] 타임아웃 폴링기반의 검출 알고리즘

DAG객체 내의 모든 작업 T는 ExecuteTask()를 통해서 실행된다. 우선 실행조건들을 검사하고 입력데이터가 충분치 않으면 PollCheckThread(T)로 해당작업의 상태를 점검하며, 일정시간이 지난후에 CheckParentStatus(T)로 부모 노드의 상태를 검사하여 작업 진행상황을 점검한다.

3.3 재시동 프로토콜의 메커니즘

시스템 실행중에 일어나는 결함에 대비하여 재시동 프로토콜을 수행하기 위해서 우선 시스템 실행시 일어나는 결함을 발견해야 한다. 그리고 결함에 따라 작업을 재시동한다.

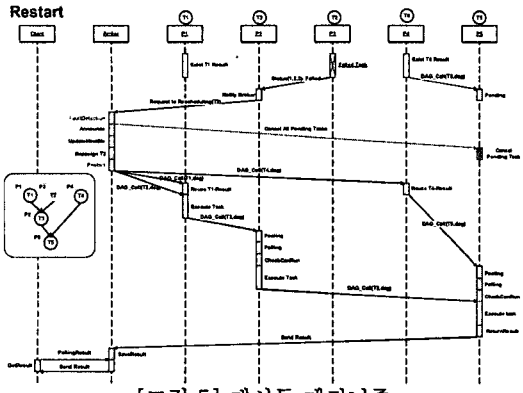
(1) 결함 감지 메커니즘(Fault Detection)



[그림 4] 결함 발견 메커니즘

그림4,5 는 재시동 프로토콜을 위한 두가지 절차를 시퀀스다이어그램으로 나타낸것이다. 각 프로세서들 간에 관련된 작업들의 흐름을 묘사한다.

(2) 재시동 메커니즘 (Restart)



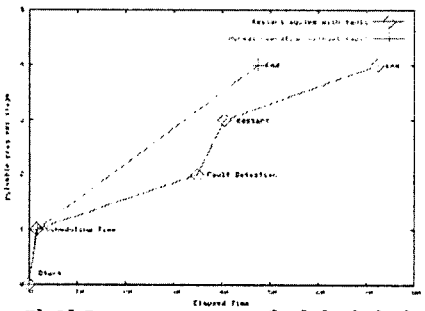
[그림 5] 재시동 메커니즘

3.4 재시동 프로토콜의 잇점

본 논문에서 구현한 재시동 프로토콜은 두가지 잇점을 가지고 있다. 먼저 시스템이 결함에 대비하여 정상적으로 작동할 수 있도록 전체 작업을 재시동시켜 작업을 정상적으로 완수하는것과 다른 하나는 각 계산 서버에 데이터 영속성을 제공함으로써 어플리케이 션의 실행 중 계산 서버의 고장으로 인한 어플리케이션의 재시동 시에 대규모의 자료 재전송 및 반복 계산을 피하고 이미 계산된 결과를 이용하여 반응시간을 줄이도록 설계하여 중복되는 작업을 수행하는 큰 오버헤드를 피할 수 있다.

4. 실험

본연구에서는 재시동 프로토콜에 의한 효율성을 확인하기 위하여 결함발생시의 전체작업의 실행시간을 측정하였다. 그림6은 시스템이 정상적으로 작동했때의 실행시간과 결함발생시에 재시동 프로토콜에 의한 실행시간을 각 단계별로 나타내고 있다. 정상적으로 작동한 경우 Start-SchedulingTime-End의 단계를 거치고, 결함발생시 Start-SchedulingTime-FaultDetection-Restart-End의 단계를 거쳐 작업이 재시동된 실행 시간을 보여준다. 이경우(FaultDetection-End)에, 정상 작동한 Start-End의 시간에 비해 적은 시간으로 작업을 완료하였다.



[그림 6] Restart Protocol에 의한 실행 시간

5. 결론

본 논문은 DAG기반의 Co-scheduling을 이용한 RPC시스템인 SOAP 기반의 pyBubble의 신뢰성있는 시스템을 위한 재시동 프로토콜의 메커니즘에 대해서 설명하였다. 시스템 동작중에 일어나는 결함 발생시 전체 작업을 정상적으로 수행하기 위하여 결함을 감지하고 전체 작업을 효율적으로 재시동시킴으로서 시스템의 신뢰성 향상과 그 장점들을 알아보았다.

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, pp.1213-1222, 2000

[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.

[9] 최지현, 이동우, R.S.Ramakrishna, " 그리드 환경에서 효율적 RPC프로그램을 위한 DAG기반의 Co-scheduling의 구현" , 2004정보과학회 춘계학술대회 학술발표논문집 Vol.31 (A) pp. 472-474, 2004

[10] 이동우, 최지현, R.S.Ramakrishna, " SOAP기반의 분산처리 스케줄링 프레임웍: pyBubble" , 2004정보과학회 춘계학술대회 학술발표논문집 Vol.31 (A) pp. 742-744, 2004