통합컴퓨팅 환경을 위한 스케쥴러 설계 및 Prototype 구현

성진우, 이상동, 김성준, 이영주, 김중권
한국과학기술정보연구원 슈퍼컴퓨팅센터
e-mail: jwsung@kisti.re.kr

Design and Prototype Implementation of Scheduler for Consolidation Computing Environment

Jin-Woo Sung, Sang-Dong Lee, Sung-Jun KIM,
Young-Joo Lee, Joong-Kwon Kim
Supercomputing center,
Korea Institute of Science and Technology Information(KISTI)

요 약

고성능의 컴퓨터들이 많이 보급되었으며, 또한, 클러스터 시스템 기술의 발전으로 클러스터 시스템의 활용이 크게 늘고 있다. 이러한 고성능의 시스템을 사용하는 사용자들은 다수의 시스템에서 수행하는 작업들을 관리(실행, 모니터링, 삭제, 결과물 관리 등)할 때에 반복적인 작업들이 많이 존재한다.

그러므로, 이러한 반복적인 작업을 손쉽게 수행해 줄 수 있는 기능이 제공된다면 업무를 효율적으로 처리할 수 있을 것이다. 이 논문에서는 분산된 다수의 시스템에서 작업을 수행할 때 스케쥴러(예: PBS, NQS)와 작업을 수행하는 일을 대신해줄 수 있는 통합 작업 스케쥴러(CJS, Consolidation Job Scheduler)를 설계하고, 그 prototype을 구현해 보았다. 스케쥴러의 종류가 많기 때문에 여기서는 PBS와 NQS에 한정하여 구현하였다.

1. 서론

인터넷과 고성능 컴퓨터의 이용성 그리고 초고속 네트워크의 성장은 오늘날 우리가 계산하는 일과 컴퓨터를 사용하는 방법을 변화시키고 있다.[1] 그리고, 리눅스 클러스터형 컴퓨터의 기술 발전으로 고성능의 컴퓨팅 시스템이 많이 보급되었다. 이러한 변화는 HPC(High Performance Computing) 계산을 필요로 하는 연구자들에게는 더 많은 연구를 할 수 있는 도구로 활용되고 있다.

이러한 고성능의 컴퓨팅 시스템들은 자원을 효율적으로 활용하기 위하여 각 시스템에서는 작업 스케쥴러를 사용하고 있다. 그래서 연구자들은 그 시스템의 작업 환경과 필요한 명령어를 숙지하는 것뿐만 아니라 스케쥴러와 관련한 명령어를 알아야 한다. 또한 시스템마다 서로 상이한 스케쥴러를 사용한다면 더욱 복잡해 질 것이다. 분산되어 있는 이기종의 자원을 공유하여 사용할 수 있는 그리드 컴퓨팅 기술이 있지만 아직은 많은 연구가 필요한 실정이다. multi-MPP 시스템이나 클러스터에 기반한 전형적인 그리드 환경에서 자원관리와 작업 스케쥴링은 여전히 도전적인 문제 중의 하나이다.[2]

그러므로, 활용할 수 있는 고성능의 컴퓨터가 많아짐에 따라 각각의 시스템에서 작업을 수행하는 작업과 그 시스템들을 관리하는 일은 많아지고 복잡해 질 것이다. 이를 위하여 사용자의 작업을 대행해 주고, 사용자에게 보다 편리한 사용자 인터페이스 환경을 제공해 줄 수 있는 에이전트 시스템이 필요할 것이다.

본 논문에서는 분산된 컴퓨팅 자원들을 통합하여 사용할 수 있는 계층적 방식[3]과 유사한 스케쥴러(CJS)를 설계하였으며, 그 prototype을 구현하였다.

본 논문의 구성은 2장에서 기본 개념을 살펴보고, 3장에서는 설계 및 구현결과에 대하여, 4장에서는 결론을 기술한다.

2. 기본 개념

2.1 클러스터 시스템

클러스터는 로컬 지역 내의 컴퓨터(워크스테이

션, 서버, 슈퍼컴퓨터 등) 여러 대를 이더넷(Ethernet)을 이용해 동일한 시스템(OS) 환경에서 상호 연결하여 규모가 큰 하나의 가상화된 컴퓨터처럼 총체적인 서비스를 제공하는 독립적인 개체(Entity)이다. 시중에 판매되는 장비들을 이용하여 구축할 수 있는 시스템으로, 적은 비용으로 높은 성능을 낼 수 있다.[4]

베오울프(Beowulf)라고도 불리는 DIY(do it yourself)형 클러스터 슈퍼컴퓨터는 이미 세계 500위 슈퍼컴퓨터 리스트 중 과반수 이상을 차지하는 기염을 토하고 있으며 국내외에서 다양한 분야에서 활용되고 있다. NASA와 Los Alamos 연구원들에 의하여 처음 시도된 Beowulf system은 작게는 수개의 PC에서 많게는 몇 천개의 PC를 연결한 테라급 슈퍼컴퓨터 등으로 운영되고 있다. 베오울프 시스템의 획기적인 성능향상에 기여한 공로로 이들은 1997년과 1998년 Golden Bell Prize를 받게 되었다. 이후 많은 리눅스 클러스터 슈퍼컴퓨터가 구축되었다. 현재 2004년 11월의 www.top500.org[5]에 따르면 세계 500위내에 드는 슈퍼컴퓨터 중 58.8%가 Beowulf Cluster형으로 구축된 슈퍼컴퓨터이며, 이중 최고의 성능을 보유한 베오울프 클러스터형 슈퍼컴퓨터가 세계4위에 랭크되어 있다.[6]

2.2 큐잉시스템과 스케쥴러

한정된 자원을 사용하기 위해 사용자가 대기하는 모든 시스템을 큐잉 시스템이라고 한다. 예를 들어 식당이나 은행에서 고객이 주문하기 위하여 기다리는 과정을 예로 들 수 있다. 은행에서 은행원이 자원이라고 할 때 고객이 서비스받기 위하여 차례로 줄을 서서 서비스 받는 것도 큐잉시스템의 예라고 할 수 있다.

일반적으로 큐잉 시스템을 구성하는 기본 요소는 다음과 같이 분류해 볼 수 있다.

- 고객(customers):시스템에서 자원을 기다리는 객체
- 자원(resources):고객에게 서비스를 제공하는 것
- 큐(queue):고객이 서비스를 기다리는 공간 혹은 구조

큐잉시스템을 사용하는 이유는, 제한된 자원을 효과적으로 사용하여 대기 시간을 줄이기 위해서이다.[7] 큐잉 시스템에 사용하는 소프트웨어를 스케쥴러라 부르며 대표적인 소프트웨어로는 PBS, LSF, NQS, Loadlevler, Condor 등이 있다.

2.3 관련 연구

다양한 가능성과 특징을 가진 그리드 레벨의 스케쥴러들이 최근 몇 년에 많이 제시되었다. Silver-maui, NimrodG, Condor-G, Sun Grid Engine, Legion, Pegasus, SPHINX, MARS[8] 등이 있다. 이들이 핵심적인 서비스를 제공하지만 어플리케이션에서 필요한 다양한 요구를 충족시키기에는 아직 많은 연구가 필요하다.

3. 설계 및 구현 결과

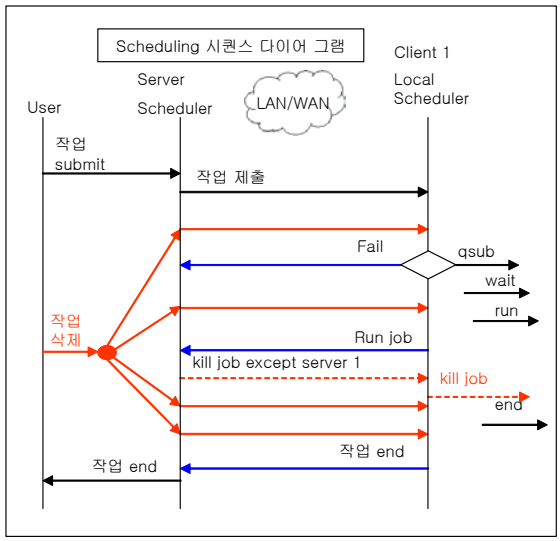
3.1 스케쥴러의 요구기능

개발한 스케쥴러의 기본적인 기능은 아래와 같다.

- 접속
  - 작업을 실행하기 위하여 각 시스템에 접속하는 것을 서버의 스케쥴러에서 담당한다.
- 작업 관리(작업 제출, 작업 모니터링, 작업 삭제)
  - 작업 제출: 사용자의 작업 스크립터에 따라 적합한 클라이언트에 작업을 제출된다.
  - 제출한 작업은 서버에서 모니터링이 되어야 한다.
  - 제출한 작업을 삭제할 수 있어야 한다.
- 클라이언트 노드 모니터링
  - 사용자의 작업이 수행되는 각 클라이언트 시스템의 상태도 주기적으로 모니터링한다.
- 파일 관리(send, receive, delete)
  - 작업에 필요한 파일을 작업이 실행될 클라이언트 시스템에 전송한다.
  - 작업이 종료한 후에 생성된 결과 파일을 각 클라이언트에서 서버로 전송한다.

3.2 스케쥴러 설계

아래 그림1은 서버와 클라이언트의 스케쥴러 모듈간의 기본적인 work flow이다. 서버와 클라이언트간의 통신은 1:1방식이 아니라 1:N방식으로 이루어지며, 그림 1은 여러 클라이언트 중에서 1개 클라이언트와의 통신만 나타내었다.



[그림 1] 스케쥴링 시퀀스 다이어그램

설계한 스케쥴러의 주요 특징은 "Job Forwarding"이라 할 수 있다. 이 기법에서는 사용자 작업을 처리하는 것은 클라이언트의 스케쥴러에게 전담시키며, 서버의 스케쥴러는 사용자와 클라이언트간의 인터페이스 역할과 클라이언트 작업들을 통제하는 기능을 한다.

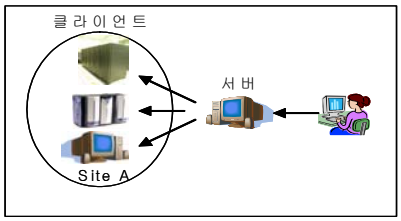
작업의 처리 단계는 다음과 같다.

1. 사용자가 작업을 submit한다.
2. 서버는 작업을 하위의 모든 클라이언트에게 전달한다.
3. 각 클라이언트의 스케쥴러는 그 작업을 submit한다.
4. 작업이 실행(run) 단계이면 클라이언트는 서버에게 통보한다.
5. 서버는 가장 먼저 통보 받은 클라이언트를 제외한 모든 클라이언트에게 작업을 삭제하도록 지시한다.
6. 작업이 종료하면 서버에게 알리고 결과 파일을 돌려준다.
7. 서버는 작업을 종료한다.

본 논문에서 설계 및 구현한 스케쥴러에서는 PBS와 NQS 스케쥴러를 인식하는 prototype을 개발하였다.

3.3 시험 환경

시험 환경은 그림 2와 같이 1대의 서버와 3대의 클라이언트(컴퓨팅 시스템)로 구성되어 있다.



[그림 2] 시험 환경

시험에 사용된 서버의 사양은 P3 500MHz, 126MB와 40GB 하드디스크, 100Mbps Ethernet Card가 장착되어 있다. O.S는 Linux 2.4.20이다. 서버와 클라이언트 정보는 표 2와 표 3에 나타나 있다.

서버명	MARS
O.S	LInux
Processor	P3, 500MHz
CPU수	1
Scheduler	-

[표 2] 서버 환경

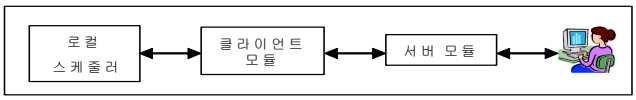
서버명	hamel	necsx5	dgtal
O.S	Redhat-Linux7.3 w kernel 2.4.20-28.7smp	SUPER-UX	Linux
구조	cluster	Vector	
Processor	Xeon DP 2.8 GHz	312.5 MHz	P4 1.7 MHz
CPU수	512	8	1
Scheduler	PBSpro	NQS	PBS

[표 3] 클라이언트 환경

클라이언트들은 O.S와 로컬 스케쥴러가 상이한 환경이 되게끔 시험 환경을 구성하였다.

3.4 구현 결과

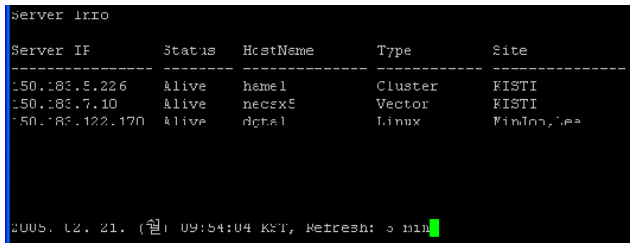
그림 2에서의 서버에는 개발한 스케쥴러의 서버 모듈이 설치되며, 클라이언트에는 클라이언트 모듈이 설치된다. 서버는 사용자가 접속하는 시스템이며, 클라이언트는 사용자 작업이 수행되는 계산 노드이며, 스케쥴러의 클라이언트 모듈이 local 스케쥴러와 작업을 한다. 개략적인 작업의 흐름은 그림 3과 같다.



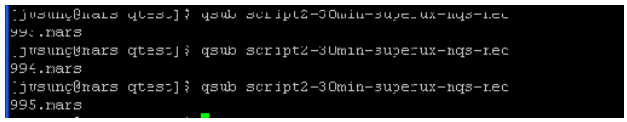
[그림 3] 작업 흐름도

프로그래밍 언어는 shell script를 기본으로 하였다. shell script언어는 인터프리터 언어이므로 컴파일 할 필요가 없고, 가독성이 높은 언어이다.

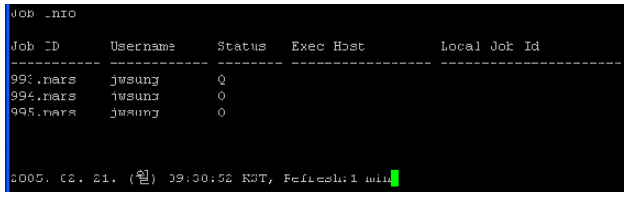
아래의 그림들은 구현결과 화면들이다. 그림 4는 서버에 등록된 클라이언트 시스템들의 상태를 확인하는 화면이다. 그림 5는 서버에서 작업을 제출하는 것을 보여주며, 명령어는 PBS와 NQS와 동일한 qsub이다. 그림 6은 제출한 작업이 클라이언트에서 실행(run)상태가 되기 전인 wait상태인 것을 보여준다. 작업 상태를 확인하는 명령어도 qstat로 동일하다.



[그림 4] 클라이언트 모니터링

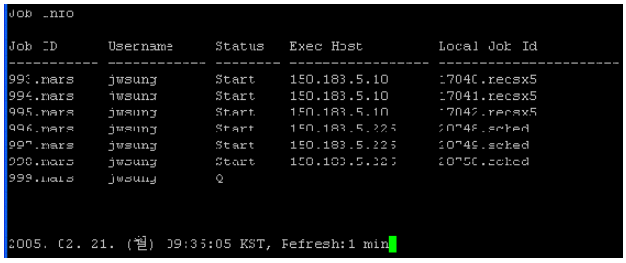


[그림 5] 작업 제출



[그림 6] 작업 대기 상태

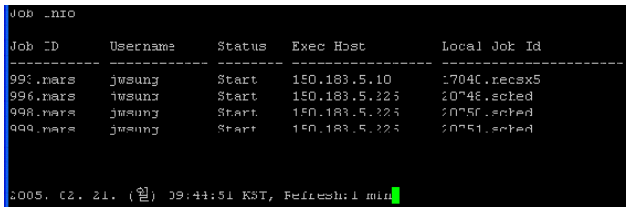
그림 7은 대기 중인 작업들이 사용자 작업이 실행되기에 적합한 클라이언트를 찾은 후 실행 된 상태의 화면이며, 그림 8은 사용자가 작업을 삭제하는 화면이다. 작업을 삭제하는 명령은 qdel이며, PBS와 NQS에서의 삭제명령과 동일하다.



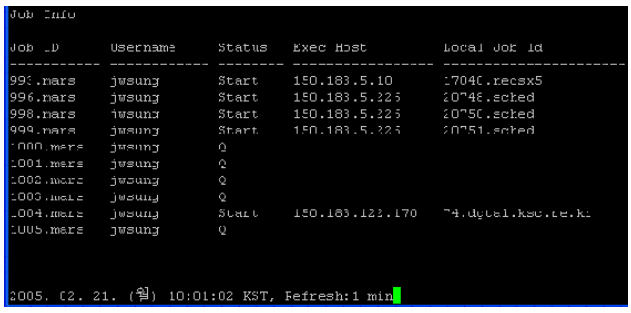
[그림 7] 작업 실행 상태

[그림 8] 작업 삭제

그림 9는 작업이 삭제된 것을 보여준다. 그림 10은 작업 스크립터에 사용자가 특정 큐를 지정하였을 때, 다른 시스템에서 수행되지 않고 지정된 큐를 가지고 있는 시스템의 큐에 대기 하고 있는 상태를 보여준다.

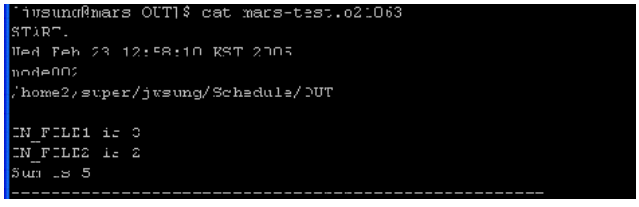


[그림 9] 대기 작업 실행 상태



[그림 10] 큐를 지정한 작업

그림 11은 클라이언트에서 생성된 결과 파일을 서버 시스템에서 확인하는 화면이다. 파일들의 송. 수신이 잘 이루어 졌음을 알 수 있다.



[그림 11] 출력 파일 확인

위의 그림 4에서 그림 11까지의 시험 과정을 통하여 연구자들이 각 클라이언트 시스템에서 수행하던 작업들이 잘 진행되는 것을 확인하였다.

4. 결론

본 논문에서는 분산된 컴퓨팅 시스템을 효율적으로 사용하고 관리하기 위한 목적으로 통합 작업 관리 스케쥴러(CJS)를 설계하고 prototype를 구현하였다. CJS는 사용자를 대신하여 클라이언트 시스템의 local 스케쥴러에게 작업을 제출하고, 삭제하고, 모니터링을 한다. CJS를 이용하여 기존의 각각의 시스템에서 개별적으로 수행하던 반복적인 작업이 CJS를 이용하여 수행이 가능한 것을 확인하였다. CJS의 주요 특징은 shell 스크립터로 작성되어 있어서 코드 수정이 쉽게 이루어진다는 점이다. 그리고 기존에 사용하던 작업 script 파일을 그대로 사용한다는 점이다.

향후 과제로는 보안과 성능을 강화시킬 예정이다. 성능 향상을 위하여 다른 프로그래밍 언어로 재작성하는 것도 검토해 볼 것이며, 원거리에 있는 시스템과의 시험도 필요하다.

참고문헌

[1] Rajkumar Buyya, Steve Chapin, David DiNucci, "Architecture Models for Resource Management in the Grid", The 1st IEEE/ACM International Workshop on Grid Computing

[2] Gerd Quecke, Wolfgang Ziegler, "MeSch - An Approach to Resource Management in a Distributed Environment", The 1st IEEE/ACM International Workshop on Grid Computing

[3] Vijay Subramani, Rajkumar Kettimuthu, Srividya Srinivasan, P.Sadayappan, "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests", 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), July 2002

[4] http://www.ngrid.co.kr/

[5] http://www.top500.org/

[6] 오정수, 이규환, "리눅스 클러스터 슈퍼컴퓨터와 계산과학", p14

[7] 리눅스 클러스터로 만드는 슈퍼컴퓨터, 이정훈, p299

[8] Abhijit Bose, Brain Wickman, Cameron Wood, "MARS: A Metascheduler for Distributed Resources in Campus Grids", 5th ACM/IEEE International Workshop on Grid Computing, November 2004.