JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 18, No. 7, pp. 1411-1418, Nov. 2017

배치 작업 로그 분석을 통한 스케줄링 최적화 연구

Study of Scheduling Optimization through the Batch Job Logs Analysis

윤 준 원[1] · 송 의 성[2*]
1한국과학기술정보연구원 슈퍼컴퓨팅본부
2부산교육대학교 컴퓨터교육과

JunWeon Yoon[1] · Ui-Sung Song[2*]

[1]Department of Supercomputing Center, KISTI, Daejeon 34141, Korea

[2*]Department of Computer Education, Busan National University of Education, Busan 47503, Korea

[요 약]

배치 작업 스케줄러는 클러스터 환경에서 구성된 계산 자원을 인지하고 순서에 맞게 효율적으로 작업을 배치하는 역할을 수행한다. 클러스터내의 한정된 가용자원을 효율적으로 사용하기 위해서는 사용자 작업의 특성을 분석하여 반영하여야 하는데 이를 위해서는 다양한 스케줄링 알고리즘을 파악하고 해당 시스템 환경에 맞게 적용하는 것이 중요하다. 대부분의 스케줄러 소프트웨어는 전체 관리 대상의 자원 명세와 시스템의 상태뿐만 아니라 작업 제출부터 종료까지 다양한 사용자의 작업 수행 환경을 반영하게 된다. 또한 작업 수행과 관련한 다양한 정보 가령, 작업 스크립트, 환경변수, 라이브러리, 작업의 대기, 시작, 종료 시간 등을 저장하게 된다. 본 연구에서는 배치 스케줄러를 통한 작업 수행과 관련된 정보를 통해 사용자의 작업 성공률, 수행시간, 자원 규모 등의 스케줄러의 수행 로그를 분석하여 문제점을 파악하였다. 향후 이 연구를 바탕으로 자원의 활용률을 높임으로써 시스템을 최적화할 수 있다.

[Abstract]

The batch job scheduler recognizes the computational resources configured in the cluster environment and plays a role of efficiently arranging the jobs in order. In order to efficiently use the limited available resources in the cluster, it is important to analyze and characterize the characteristics of user tasks. To do this, it is important to identify various scheduling algorithms and apply them to the system environment. Most scheduler software reflects the user's work environment, from job submission to termination, as well as the state of the inventory and system status of the entire managed object. It also stores various information related to task execution, such as job scripts, environment variables, libraries, wait for tasks, start and end times. In this paper, we analyze the execution log of the scheduler such as user 's success rate, execution time, and resource size through information related to job execution through batch scheduler. Based on this, it can be used as a basis to optimize the system by increasing the utilization rate of resources.

색인어 : 고성능컴퓨팅, 배치작업, 스케줄링, 로그분석, 최적화

Key word : HPC, Batch Job, Scheduling, Log analysis, Optimization

http://dx.doi.org/10.9728/dcs.2017.18.7.1411

Received 18 October 2017; **Revised** 20 November 2017
Accepted 25 November 2017

***Corresponding Author; Ui-Sung Song**

Tel: +82-051-500-7326
E-mail: ussong@bnue.ac.kr

Ⅰ. 서 론

배치 작업 스케줄러는 자원 관리자(Resource Manager, RM), 분산 자원 관리자(Distributed Resource Manager, DRM), 분산자원 관리 시스템(Distributed Resource Management System, DRMS), 워크로드 관리시스템(Workload Management System, WMS) 또는 작업 스케줄러(Job Scheduler)라 불린다[1].

스케줄러가 제공해야하는 기본적인 특징은 자원의 특성을 정확하게 반영해야 한다는 점이다. 즉, 작업 수행에 요구되는 다양한 계산자원의 리소스 가령 라이선스, CPU 아키텍처나 메모리 나아가 최근 화두가 되는 GPU나 PHI와 같은 매니코어 기반의 가속 시스템까지도 관리 대상에 포함 되어야 한다. 자원에 대한 인지이후 자원 분배가 공평하게 배분되기 위한 Fair-Share 정책, 우선순위가 높은 작업에 대한 선점(preemption) 지원, 자원의 확장성 보장, 다양한 시스템 환경에 대한 지원 등의 요구사항들 또한 충족시켜야 한다[2].

클러스터 환경을 구성하고 배치 형태의 작업을 관리하기 위해서는 위와 같은 여러 환경의 특성을 반영하여 스케줄러를 선택하게 된다. 이에 본 연구에서는 대표적인 스케줄러 소프트웨어를 선별하여 조사하고 특징을 나열하였다. 이후, 자원 분배의 효율성을 재고하기 위해 스케줄링 알고리즘을 연구하고 나아가 KISTI의 슈퍼컴퓨터 4호기 Tachyon2 시스템에서 추출한 사용자 작업 수행 관련 정보들을 분석하였다[3]. 이 데이터를 통해서 사용자의 작업 성공률, 수행시간, 자원 규모 등을 파악할 수 있으며 선행된 스케줄링 알고리즘 적용을 통해 최적화할 수 있음을 인지할 수 있다.

Ⅱ. 관련 연구

클러스터 컴퓨팅은 서버를 여러 대 묶어서 하나의 장비처럼 구성하는 것이다. 클러스터 환경을 구성하는 계산노드들에게 작업을 지시하고, 그 중 장애가 있는 시스템을 구분하는 것, 그리고 서버들간에 통신을 관리하는 기능이 스케줄러의 대표적인 특징이라 할 수 있다. 본 연구에서는 우선 대표적으로 사용되고 있는 스케줄러의 솔루션들을 조사하였다. 널리 사용되는 클러스터 배치 시스템들로는 Moab, Oracle Grid Engine, Portable Batch System, LoadLeveler, Condor, OAR, Platform LSF 등이 있다. 대표적인 상용 제품으로는 Platform LSF(IBM), Univa Grid Engine, Moab(Adaptive Computing), PBSPro(Altair) 등이 잘 알려져 있다. 오픈소스 제품으로는 SGE(SoG&OGS), SLURM, TORQUE/Maui 등이 있다[4].

■ Grid Engine(GE)

Sun Grid Engine(SGE)는 가장 널리 사용된 스케줄러 솔루션으로 다양한 기능과 안정성을 제공하였다. SUN은 빈번하게 버그와 추가적인 기능에 대한 패치를 매년마다 주기적으로 업데이트를 해왔다. SUN이 ORACLE에 합병되면서 Oracle Grid Engine으로 변경되었고 약자도 SGE에서 OGE로 변경되었다. 기본적으로 오픈 소스 개발 모델을 택해 소스 코드가 공개되어 있다. 오라클에서 인수된 이후 기존의 오픈 소스를 이용하여 만든 것이 OGS 즉 Open Grid Scheduler이다. 그 후 Grid Engine 부문을 Univa에 매각하였다.

ORACLE은 무료 바이너리 버전 배포를 중단하고 Sun Industry Standards Source License(SISSL) 라이선스로 배포되던 소스코드의 배포도 역시 중단하였다.

SGE는 크게 마스터 서버와 계산수행노드로 나뉜다. 일반적으로 마스터 서버는 실제 작업을 수행하는 대신 전체 클러스터에 작업을 분배하고, 클러스터 상태를 관리하는 역할을 수행한다. 계산수행노드는 실제 작업을 수행하는 서버로 마스터 서버의 지시를 받아 작업을 수행한다. 계산수행노드는 실제 일하는 서버이므로 성능이 중요한 반면 마스터서버는 전체 클러스터의 상태를 볼 수 있는 유일한 통로이므로 안정성이 중요하다.

■ SLURM

SLURM은 LLNL(로렌스 리버모아 국립연구소)에서 시작된 오픈소스 배치 시스템으로 2010년 개발자들이 SchedMD를 설립하여 지속적인 개발과 배포를 진행하는 프로젝트이다. Top 500 슈퍼컴퓨팅 사이트에서 점차 비중을 확대하고 있다. 높은 확장성과 추가적인 플러그인을 통해 다양한 기능을 제공하고 있으며 가속 프로세서(GPU, PHI)의 기능도 일부 지원한다. SLURM은 Quadrics RMS에서 영감을 받아서 개발되었으며 대규모 슈퍼컴퓨팅 센터들의 요구사항을 충족시키는 것을 목표로 삼고 있다. SLURM의 핵심 요소는 slurmctld라는 컨트롤 데몬이며 가용자원의 모니터링과 배차작업 스케줄링을 담당한다. 사용자, 작업 및 수행정보는 slurmdbd를 통해 구현되는데 보통 MySQL과 같은 데이터베이스를 통해 저장한다. SLURM은 추가적인 기능들을 확장하기 위한 많은 선택적인 플러그인을 가지는 모듈 구조로 되어 있고 Maui와 같은 다른 스케줄러와 통합될 수 있도록 개발되었다. SLURM은 매우 높은 확장성으로 인해서 아주 대규모의 페타급 슈퍼컴퓨터에 적용되고 있다[7].

■ TORQUE

TORQUE(Terascale Open-source Resource & QUEue Manager)는 분산 리소스 관리자로 OpenPBS 프로젝트를 기반으로 하는 커뮤니티 프로젝트이다. TORQUE는 기본적인 스케줄링 기능만을 제공하며 오픈소스 Maui 클러스터 스케줄러 또는 상용인 Moab 워크로드 매니저와 통합하여 사용된다. 지속적인 커뮤니티 패치를 수행하고 있으며 확장성 개선, fault tolerance 와 같은 다양한 기능 개발을 위해 전 세계의 많은 HPC 기관들이 참여하고 있다. 중소규모 클러스터들에서 배치 시스템으로 많이 사용하고 있다.

Maui 스케줄러는 비록 새로운 개발은 중단된 상태이기는 하지만, Adaptive computing 사에 의해서 유지 및 지원된다. 차세대 상용 스케줄러는 Moab Cluster suite의 일부이며 Maui 스케줄러에서 많은 기본 개념을 차용하여 개발되었다. Maui의 개발자들은 오프소스 라이선스 정책을 따르고 있으며, 상업적 사용은 허용하지 않고 있다. TORQUE/Maui의 메인 관리 노드에는

pbs_server 데몬과 maui 스케줄러 데몬이 설치되어 동작하며, 각 계산 노드에는 pbs_mon 데몬이 구동된다[8].

■ LSF

LSF(Platform Load Sharing Facility는 HPC 분야의 분산 워크로드 관리 플랫폼인 작업 스케줄러이며 다양한 아키텍처를 지원하고 Unix 또는 Windows 시스템에서 배치 작업을 실행하는데 사용할 수 있다. LSF는 Toronto 대학의 Utopia 연구 프로젝트를 기반으로 시작되었다. 2007년에 Platform은 Platform Lava를 출시했으며 Platform Lava 는 LSF 릴리스의 구 버전을 기반으로 하는 LSF의 단순화 된 버전으로 GNU General Public License v2에서 라이선스를 받았고 2011 년에 중단되었다가 2012년 Platform Computing은 IBM에 인수되었다[9].

클러스터 환경을 구성하고 다양한 사용자의 작업을 배치형태로 수행하기 위해서는 스케줄러의 특성을 파악하고 시스템 환경에 맞게 적용하여야 한다. 가령, 작업 작업의 수행시간, 개수, 사용자의 제한, 작업 스크립트의 오류, 라이브러리의 사용여부 등을 필터링 하여 가용 리소스를 적절하게 사용할 수 있게 하는 기능들은 시스템 자원을 보다 효율적으로 활용할 수 있게 한다. <표 1>은 위에 언급한 대표적인 HPC 기반의 스케줄러의 특성과 기능적 요소들을 나열하고 비교하였다[4].

표 1. 스케줄러 기능적 특징
Table 1. Functional features of the scheduler

	Grid Engine	Slurm	Torque	LSF
License	Commercial, Open Source	Open Source	Commercial, Open Source	Commercial
OS Support	Linux	Linux	Linux	Linux, Windows
parallel and array jobs	V	V	V	V
Backfilling	V	V	V	V
Gang Scheduling	N/A	V	N/A	N/A
Job dependencies	V	V	V	V
Advanced reservations	V	V	V	V
Scalability	10K+	100K+	20K+	10K+
Prolog/Epilog	V	V	V	V
Checkpoint	V	V	V	V
Job migration &restarting &preemption	V	V	V	V
Job preemption	V	V	V	V

Ⅲ. 스케줄링 정책

사용자는 작업을 수행하기 위해 작업 스크립트 파일을 작성하여 스케줄러에 배치 형태로 제출하게 된다. 이후 큐(queue)에 등록되고 대기 과정을 거쳐 계산자원이 확보되면 작업이 수행되게 된다. 큐에 등록된 작업들은 먼저 제출된 작업이 먼저 수행되는 FCFS(First Come First Served) 방식을 기본 정책으로 할당된다. FCFS는 작업 수행 순서의 공정성(Fairness)을 보장하는 가장 좋은 방식이기는 하나 계산자원의 규모가 커질수록 자원을 효율적으로 사용하는데 제한이 있다. 왜냐하면 각 작업마다 요구하는 자원의 규모가 달라 실제 가용자원이 사용되지 못하고 단편화가 발생할 수 있기 때문이다. 이런 문제를 해결하기 위해 단편화(Fragmentation)된 자원에 맞는 작은 작업을 우선적으로 배치하는 방식인 SJF(Shortest Job First)은 자원의 활용성을 높여 전체적인 성능을 향상을 가져올 수 있다. 하지만 이 방식은 공정성을 보장하지 못하는 단점을 지닌다. 따라서 스케줄링 정책은 자원 규모, 작업 특성 등을 고려하여 FCFS, SJF 두 가지를 혼용하는 방식으로 사용된다[10].

백필 스케줄링(Backfill Scheduling)은 자원 사용량이 큰 선행 작업으로 인해 작은 작업이 수행되지 못할 때 우선 수행할 수 있도록 순서를 재조정하는 스케줄링 방식으로 작업의 공정성과 성능 향상을 가져올 수 있다. 백필 스케줄링에서는 각 작업의 요구 수행시간이 반드시 명시되어야 한다. 기존에 사용되는 백필 스케줄링에는 다음과 같이 크게 두 가지 방식이 있다[11].

1) Conservative Backfilling

Conservative Backfilling 알고리즘은 초기 버전으로 스케줄링의 기본 원칙인 FCFS 방식을 고수하면서 후순위에 있는 작업 이 단편화 된 자원에 충족될 경우 우선 수행하게 된다. 이 알고리즘 수행을 위해서는 두 개의 데이터 구조를 가지게 되는데 첫 번째는 큐 목록의 작업리스트와 실행 시작 시간을 저장하는 리스트 구조와 두 번째로는 사용될 프로세서 사용량 프로파일이다. 이 알고리즘은 후순위 작업으로 인해 선순위 작업들에 지연이 전혀 없어야 하나 선행 작업이 예측 시간보다 일찍 종료되었을 경우 계획된 작업 순서를 보장할 수 없다. <표 2>는 Conservative Backfilling 알고리즘을 나타내고 있다.

표 2. Conservative Backfilling 알고리즘
Table 2. Conservative Backfilling Algorithm

1) Search for the start time
   ① Find the required resources required and find the first point of available processors that have enough work to do.
   ② Start from this point and continue to find out whether the processor is available as expected until the end of the operation.
   ③ If not, go back to ① and continue scanning to scan the next possible anchor point

2) Updating the information of the waiting queue resource details reflecting the allocated details of the processors from the start point to the end point of the task

3) If the point of operation is the current point, perform the task immediately

2) Easy Backfilling

Easy Backfilling 알고리즘은 좀 더 공격적인 알고리즘으로 큐 안에 앞에 있는 작업을 지연시키지 않으면서 작은 작업을 우선 수행하게 된다. 다른 작업과의 상호 작용은 체크하지 않으며 다른 작업은 지연될 수도 있다. 목적은 큐 목록의 순서를 고려하여 현재 사용률을 최대한 향상시키는 것으로 Conservative 알고리즘과 달리 작업이 큐에 들어갈 때 마다 전체 목록을 확인해서 여유 자원이 확보되면 백필을 수행하게 된다. <표 3>은 Easy Backfilling 알고리즘을 나타내고 있다.

표 3. EASY Backfilling 알고리즘
Table 3. EASY Backfilling Algorithm

1) Finding available time and available resources ① Sort the list of running jobs according to the expected end time ② Repeat the list until the number of available nodes is enough for the first job in the queue and collect the nodes ③ The time it takes to run during the waiting time is not running ④ At this point, in addition to the resources required by the first waiting task 2) Find backfill work ① Repeat queue waiting job list in order of arrival of job ② Make sure that one of the following conditions is met for each – do not need more than the currently available node – If you do not need more than the minimum of the currently available and available nodes ③ The first operation can be used for backfill * Whenever a new job arrives at the queue or one execution job is terminated, the algorithm continues to iterate

Ⅳ. 배치작업 분석

4-1 시스템 환경

KISTI 슈퍼컴퓨터 4호기 Tachyon2는 SUN 블레이드 6275로 구성된 클러스터 컴퓨팅 시스템으로 300TFlops 이론성능(Rpeak)를 가지고 있으며, 3,200대의 계산 노드와 인프라 노드들(스케줄러 서버, 스토리지, 인터커넥터, 아카이빙 시스템, 냉각장치 등)로 구분된다. (그림 1)은 Tachyon2 시스템의 개요도를 나타내고 있다. Tachyon2의 OS는 RHEL 5.3으로 운영되고 있으며, 인터커넥터로는 인피니밴드(Infiniband, 이하 IB) 네트워크의 2 Layer, Fat Tree 토폴로지로 구성되어 있다. IB는 QDR(quad data rate) 타입으로 32Gbit/s의 데이터 전송률 가지며 계산노드와 병렬파일시스템의 I/O로 사용하고 있다. 두가지 타입의 스토리지를 병렬 파일시스템 사용하고 있으며 여기에는 오픈소스 기반의 Lustre v1.8과 v2.4가 설치되어 있다.Tachyon2에서는 클러스터 내의 배치 작업 수행을 위해서 초기 SUN에서 개발된 Grid Engine을 사용하고 있다. 스케줄러에 제출된 배치 작업은 정책에 따라 큐(queue)에 대기한 후 계산자원이 확보되면 작업 수행을 시작하게 된다[3]. SGE 스케줄러는 <표 1>에서와 같이 다양한 기능적 특징을 가지고 있다. 최근 Tachyon2에서는 사용자 작업이 제출되어 수행되기까지 평균 10시간 이상의 대기시간을 갖는데 작업스크립트나 자원 명세 등을 명확히 표기하지 않으면 작업이 정상적으로 수행되지 않고 다시 제출하여야 한다. 이를 방지하기 위해 스케줄러의 전처리 기능을 사용하여 사전에 작업 수행의 오류를 필터링하게 된다. 이로써 자원 활용을 보다 효율적으로 수행할 수 있게 된다.

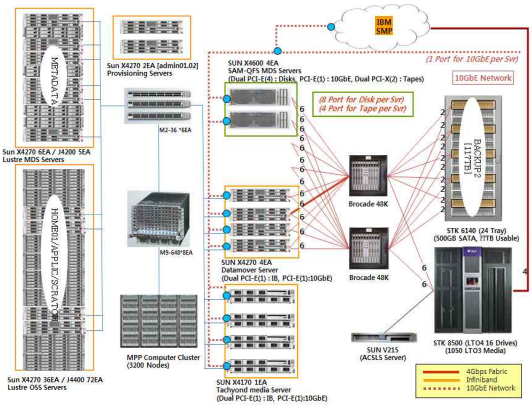

그림 1. Tachyon2 시스템 개요도
Fig. 1. Tachyon2 System Overview

4-2 배치작업 관리

배치작업 관리를 위한 스케줄러는 자원을 공유하기 위해 순서나 일정을 조정해주는 프로그램으로 SGE, Condor, Torque, PBS, LoadLeveler 등 다양한 솔루션들이 있다.

앞서 언급하였듯이 Tachyon2에서는 배치작업을 다루기 위해 SGE를 사용하고 있다. 스케줄러를 통해서 사용자는 작업이 어떤 계산자원에서 수행되는지 확인할 필요 없이 제출할 수 있다. SGE에서 작업을 제출하기 위해서는 <표 2>와 같이 작업 스크립트 파일을 작성한 후 SGE의 작업제출 명령어인 qsub 명령을 통해 큐에 적재한다. 해당 큐에 대기 후 계산자원이 확보되면 작업을 수행하게 된다. Tachyon2에서는 한 노드에 한 사용자의 작업만이 실행되는 배타적인 노드 할당 정책이 설정되어 있다. 이는 복수 작업이 단일 노드에 할당되어 발생할 수 있는 간섭현상을 제거할 수 있다.

표 4. 사용자 작업 스크립트
Table 4. User Job Script

```
#! / bin / bash
# $ -V
// submit job node shell environment variable to compute node(default)
# $ -cwd // use current directory as working directory
# $ -N hybrid_job // job name
# $ -pe mpi_4cpu 8 // 8 total MPI tasks, 4 MPI tasks per node
# $ -q normal // Name of job to execute
# $ -R yes // Resource Reservation
# $ -wd / scratch / <user01> / hybrid # Set working directory
# $ -l h_rt = 10: 00: 00 // Work elapsed time (hh: mm: ss)
// (wall clock time)
# $ -l OMP_NUM_THREADS = 2 5
// Number of OpenMP thread5s per MPI process
# $ -M myEmailAddress // Work related user's mail address
# $ -m e // send mail at end of job
export OMP_NUM_THREADS = 2

mpirun_rsh -hostfile $ TMPDIR / machines -np $ NSLOTS ./hybrid.exe
// Program files to be run by the user
```

Tachyon2에서는 작업의 시작과 종료시점에 각각 작업 정보를 저장하게 된다. 작업을 제출하고 큐에 대기한 후 자원을 할당받아 시작이 되면 먼저 사용자의 작업 스크립트를 백업하고 라이브러리, 컴파일러 등의 수행환경과 할당된 노드정보를 저장한다. 작업이 종료되면 SGE의 로그를 추출, 재처리하여 <표 5>와 같은 형식으로 수행내역을 저장하게 된다[12].

작업수행일자, 작업 ID, 작업명, 작업제출과 시작 그리고 종료된 시간, 수행된 시간, 사용 CPU 개수, 종료코드, 실패코드, 쓰레드 개수 등의 작업 정보 로그를 남기게 된다. 수행 완료된 작업은 스케줄러 상에서 정상 또는 비정상 종료를 구분하여 표시하는데 그 결과를 16번째의 STATUS 필드에 저장한다. D(Done)이면 정상 E(Error) 비정상으로 종료됐음을 확인할 수 있다. 또한 작업 수행이 정상적으로 종료되었는지를 확인하기 위해 19번과 20번 필드의 Exit-code(status), Failed code의 정보를 확인하여 종료 상태를 파악할 수 있다. Exit-code(status)와 Failed code의 결과 값이 모두 0인 경우에는 작업이 이상 없이 수행됐음을 나타내며 그 외의 값인 경우에는 작업 또는 시스템 이상으로 인하여 불완전하게 작업이 종료된 경우로 그 원인에 따라 서로 다른 값을 남기게 된다[13].

표 5. 작업 수행 로그 (작업완료 후)
Table 5. Job Execution Logs(after completion)

1	2	3	4	5
DATE	JOBID	GID	UID	JOBNAME
20170821	271481	gid032	uid000	mpi_solver
6	7	8	9	10
QNAME	SUBMIT (DATE)	START (DATE)	END (DATE)	WAIT(s)
normal	20170821	20170821	20170821230	201536
11	12	13	14	15
RUN(s)	CPUS	CPU USAGE	MEM USAGE	MAXV MEM
31852	80	22281.74	28.41	1876.06
16	17	18	19	20
STATUS	E-CPU	E-RUN(s)	EXIT-CODE	FAILED
D	80	2868160	0(32)	0(32)

4-3 배치작업 수행 로그 분석

본 논문에서 스케줄링 최적화를 위해 우선 Tachyon2 작업로그를 분석하였다. Tachyon2 시스템은 계산노드 당 8코어(쿼드코어 2소켓 장착) 3,200대로 총 코어개수가 25,600개로 구성되어 있다. 또한 한 노드에 하나의 작업만이 실행될 수 있도록 보장하는 배타적 노드 할당 정책을 기본적으로 적용하고 있다. 이는 단일 노드에서 두 작업 이상이 수행될 때 서로 다른 자원 사용특징에 따라 발생할 수 있는 간섭에 따른 성능 저하를 막고자 하였다. (그림 2)는 Tachyon2 시스템에서 수행된 자원규모에 따른 작업개수 분포를 나태내고 있다. Tachyon2에서는 129코어 이상의 작업부터 512코어까지의 작업의 규모가 가장 큰 부분을 차지하며 1,025코어 이상의 작업은 약 15%를 차지하고 있다.

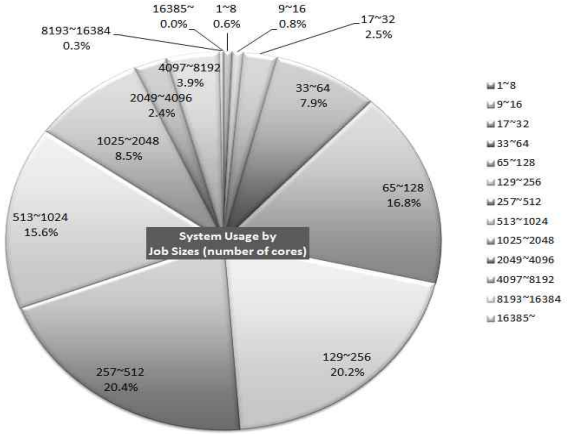

그림 2. 자원규모에 따른 작업개수 분포
Fig. 2. Distribution number of jobs according to resource size

이중 2,000 코어 이상의 대규모 자원 사용에 대해 2015년 1월부터 2017년 7월까지 월별 작업 수행개수를 구분하였다. (그림 3)는 대규모 작업수행 통계를 보여주고 있다. 그림에서와 같이 2015년 대비 2017년 대규모의 작업 수행 개수가 점차 늘어

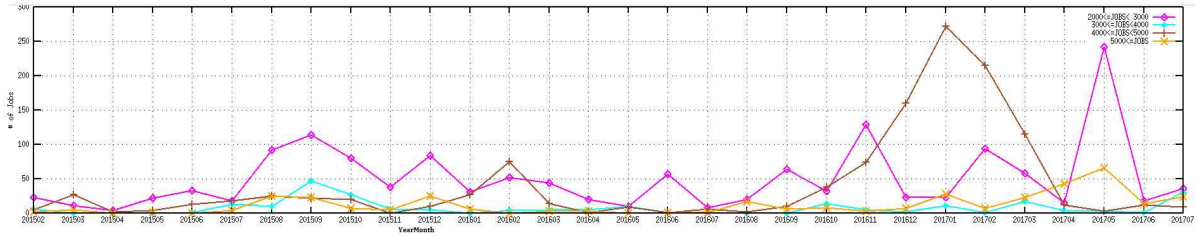

그림 3. 월별 대규모 작업 수행 개수 통계 (2015.01~2017.07)
Fig. 3. Statistics of monthly large-scale job execution count (Jan, 2015~July, 2017)

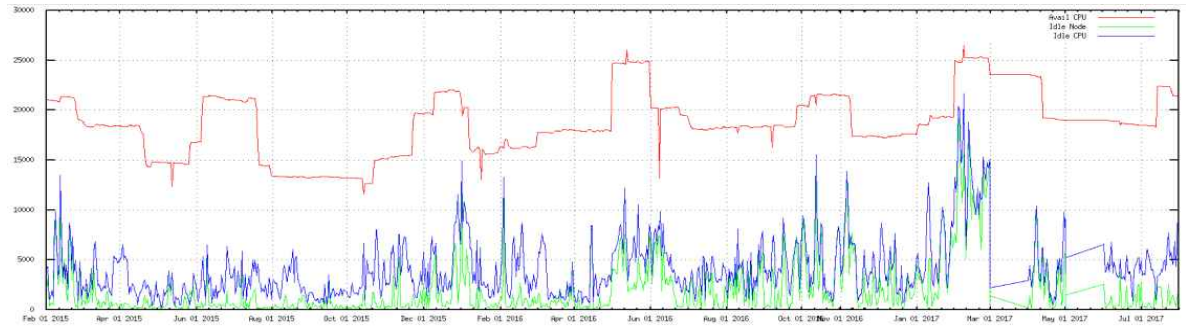

그림 4. 공용 큐 가용노드 통계 (2015.01~2017.07)
Fig. 4. Statistics of public queue available node (Jan, 2015~July, 2017)

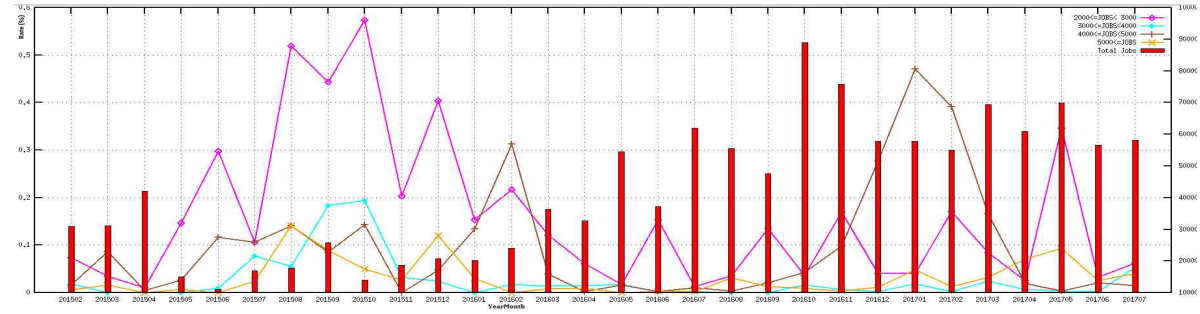

그림 5. 월별 전체 작업 수행 개수 및 대규모 작업 수행 비율 (2015.01~2017.07)
Fig. 5. Total number of monthly performed jobs and rate of large performed jobs (Jan, 2015~July, 2017)

나고 있다. Tachyon2 시스템의 큐 구성은 크게 공용 큐와 전용 큐로 구분이 되는데 전용 큐의 경우는 특정 연구 목적 지원을 위해 별도의 자원을 할당하게 된다. (그림 4)에서 적색그래프는 2015년부터 공용 큐의 가용자원 노드수를 나타내고 있다. 파란 선은 총 유휴 노드의 수를 나타내며, 적색 선은 총 유휴 코어의 수를 나타내고 있다. 이 기간 동안 사용자가 작업을 제출하고 수행되기까지의 걸리는 대기시간의 평균은 약 12.6시간 정도가 소요되었다. (그림 5)는 전체 수행 작업 개수와 2,000 코어 이상의 대규모 작업에 대한 구간별 수행 비율을 보여주고 있다.

Tachyon2는 주 알고리즘으로 FCFS가 적용되어 사용되고 있으며 백필링과 같은 자원의 단편화를 효율적으로 사용할 수 있는 알고리즘이 적용되지 않고 있다. 즉, 평균 대기시간이 12.6시간이지만 공용 큐의 사용률은 70%~80% 정도만을 사용하고 있으며 (그림 5)에서 보여주듯이 대규모 작업의 분포가 커질수록 단편화된 자원이 커져 효율성 낮아지고 대기시간이 증가하게 된다.

Ⅴ. 결 론

본 논문에서는 클러스터 환경에서 배치작업을 수행하기 위한 스케줄러 최적화 연구를 수행하였다. 작업이 수행되는 환경을 스케줄러는 반영하고 작업 요구사항에 맞춰 공평하게 분배해야 한다. 이를 위해 스케줄러는 복잡한 요구사항을 수렴할 수 있는 다양한 기능과 알고리즘을 내포하고 있어야 한다. 이에 대표적인 스케줄러 소프트웨어를 조사하고 각각의 특징을 비교하는 선행 작업이 필수 요소이다.

KISTI의 슈퍼컴퓨터 4호기 Tachyon2는 3,200대로 구성된 슈퍼컴퓨터로 2015년부터 현재까지 약 140만개의 작업이 수행되었다. 주목할 것은 Tachyon2 시스템의 평균 사용률이 약 80%, 평균 대기시간이 12시간 정도가 된다는 것이다. 이는 스

케줄링 시 발생하는 자원의 단편화로 볼 수 있다. 물론 사용량과 수행시간을 반영하여 모든 자원을 완벽하게 사용하는 것은 불가능하지만, 해당 계산 자원에서 수행됐던 작업을 분석하고 그에 요구되는 스케줄링 알고리즘을 연구하여 적용한다면 작업 스케줄링 시 발생하는 자원의 단편화를 최소화 자원 활용을 보다 효율적으로 할 수 있다.

본 연구에서는 특히 Tachyon2 시스템에서 점차 증가되고 있는 대규모 작업의 통계를 분석하여 사용자의 작업 성공률, 수행시간, 자원 규모 등을 파악하였다. 통계 결과를 기반으로 Tachyon2에서 발생하는 자원의 단편화를 줄이기 위해 백필링 스케줄링 알고리즘을 연구하였다.

향후 Tachyon2 시스템에 스케줄링 알고리즘을 적용하고 작업 수행 내역을 분석하는 연구를 지속, 반복적으로 수행하고자 한다. 이를 통해 자원 활용을 최적화할 수 있으며 전체적인 사용자 작업 대기시간 또한 줄일 수 있다.

참고문헌

[1] He, Libo, et al., "A Review of Resource Scheduling in Large-Scale Server Cluster", International Conference on Knowledge Management in Organizations. Springer, Cham, pp. 494-505, 2017.

[2] J.H. Abawajy, “An efficient adaptive scheduling policy for high-performance computing”, Original Research Article Future Generation Computer Systems, Vol 25, Issue 3, pp.364-370, Mar 2009.

[3] National Institute of Supercomputing and Networking, KISTI, Available: http://www.nisn.re.kr/.

[4] Reuther, Albert, et al. "Scalable system scheduling for HPC and big data", Journal of Parallel and Distributed Computing 111, pp.76-92, 2017.

[5] Templeton, D., “A Beginner’s Guide to Sun Grid Engine 6.2”, Whitepaper of Sun Microsystems, July 2009.

[6] C. Chaubal, "Scheduler Policies for Job Prioritization in the Sun N1 Grid Engine 6 System", Technical report, Sun BluePrints Online, Sun Microsystems, Inc., Santa Clara, CA, USA.

[7] Zhou, Xiaobing, et al., "Exploring distributed resource allocation techniques in the slurm job management system", Illinois Institute of Technology, Department of Computer Science, Technical Report, 2013.

[8] KLUSÁČEK, Dalibor; CHLUMSKÝ, Václav; RUDOVÁ, Hana, “Planning and optimization in TORQUE resource manager”, In: Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing. ACM, pp. 203-206, 2015.

[9] Quintero, Dino, et al., “IBM Platform Computing Solutions Reference Architectures and Best Practices”, IBM Redbooks, 2014.

[10] Yuan, Yulai, et al., "Guarantee strict fairness and utilize prediction better in parallel job scheduling", IEEE Transactions on Parallel and Distributed Systems Vol. 25, No. 4, pp. 971-981, 2014.

[11] Feitelson, D. G., & Weil, A. M. A. (1998, April). Utilization and predictability in scheduling the IBM SP2 with backfilling. In Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International and Symposium on Parallel and Distributed Processing , IEEE. pp.542-546, 1998.

[12] J. W. Yoon, T. Y. Hong, C. Y. Park, H.C. Yu, “Analysis of Batch Job log to improve the success rate in HPC Environment”, International Conference on Convergence Technology, vol.2 No.1, pp.209-210, July,2013.

[13] El-Sayed, N., & Schroeder, B., "Reading between the lines of failure logs: Understanding how HPC systems fail". In: Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on, pp.1-12, June, 2013.

윤준원(JunWeon Yoon)

2002년~2004년 : 고려대학교 대학원 컴퓨터학과(이학석사)
2010년~2011년 : 고려대학교 대학원 컴퓨터학과 박사 수료
2005년~현 재 : KISTI 국가슈퍼컴퓨팅연구소 선임연구원

관심분야: 분산 컴퓨팅, 결함포용시스템, 슈퍼컴퓨팅, 병렬파일시스템, 배치스케줄링

송의성(Ui-Sung Song)

1991년~1997년 : 고려대학교 컴퓨터학과(학사)
1998년~1999년: 고려대학교 대학원 컴퓨터학과(이학석사)
2000년~2005년: 고려대학교 대학원 컴퓨터학과(이학박사)
2006년~현 재: 부산교육대학교 컴퓨터교육과 교수

관심분야: 컴퓨터교육, 교육용로봇교육, 컴퓨터네트워크, 분산컴퓨팅