한국경영과학회지
제39권 제4호
2014년 11월
http://dx.doi.org/10.7737/JKORMS.2014.39.4.019

빅데이터 분석을 위한 슈퍼컴퓨터 환경에서 R의 병렬처리*

이상열[1] · 원중호[2†]
[1]고려대학교 산업경영공학과, [2]서울대학교 통계학과

Parallel Computing Environment for R with on Supercomputer Systems

Sang Yeol Lee[1] · Joong Ho Won[2]
[1]Division of Industrial Management Engineering, Korea University
[2]Department of Statistics, Seoul National University

■ Abstract ■

We study parallel processing techniques for the R programming language of high performance computing technology. In this study, we used massively parallel computing system which has 25,408 cpu cores. We conducted a performance evaluation of a distributed memory system using MPI and of a the shared memory system using OpenMP. Our findings are summarized as follows. First, For some particular algorithms, parallel processing is about 150 times faster than serial processing in R. Second, the distributed memory system gets faster as the number of nodes increases while shared memory system is limited in the improvement of performance, due to the limit of the number of cpus in a single system.

Keywords : Supercomputer, Parallel Programming, R, MPI, OpenMP

논문접수일 : 2014년 09월 12일 논문게재확정일 : 2014년 10월 13일
* 이 연구는 서울대학교 신임교수 연구정착금으로 지원되는 연구비와 2013년도 및 2014년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(Nos. 2013R1A1A1057949, 2014R1A4A1007895).
† 교신저자, wonj@stats.snu.ac.kr

1. 서 론

슈퍼컴퓨터는 전산 과학 분야에서 대용량의 데이터를 초고속으로 계산할 수 있는 컴퓨터를 말하며 HPC(High-Performance Computing)로 부르기도 한다. 국가차원에서 슈퍼컴퓨터 자원의 활용과 활성화를 위해서 많은 사용자들을 지원해주고 있는데 그 중에서 한국과학기술정보연구원(KISTI)의 슈퍼컴퓨터 서비스 센터(KSC : KISTI Supercomputing Service Center)는 가장 활발하게 운영되고 있는 슈퍼컴퓨터 인프라 기관이다.

이번 연구에서는 KISTI의 슈퍼컴퓨터인 타키온2(Tachyon2) 시스템에 오픈소스 통계소프트웨어 R을 올려 병렬처리를 실험하였다. R은 현재 빅데이터(Big data)에 대한 사회적인 관심이 늘어남에 따라 주목 받고 있는 오픈소스 통계소프트웨어이다. 빅데이터는 기존 데이터보다 규모가 크고 그 형태가 정해져 있지 않은 비정형 데이터까지 포함하여 일반적인 방법이나 도구로 처리하기 어려운 데이터 세트를 말하는데 데이터의 양이 급증하면서 대용량 데이터를 처리하는 기술들이 요구되고 있다. 대용량 데이터를 처리하기 위해 많은 오픈소스 솔루션들이 개발되고 있는데 이 중 R은 통계 계산에 최적화되어 있으며 다른 소프트웨어와의 시스템 통합이 용이한 프로그래밍 언어이다. 대용량 데이터를 분석하기 위해 분산저장 및 병렬처리에 대한 연구가 중요해지고 있으며 R에서의 병렬처리에 대한 연구 또한 활발하게 진행되고 있다. R 2.14.0버전부터는 parallel 패키지가 기본 패키지로 내장되어 있어 snow, multicore 패키지의 기능을 parallel 패키지에서 유사하게 작동할 수 있다.

병렬 작업을 고수준 레벨에서 최적화하고 관리하기 위해서는 병렬프로그래밍에서 표준 모델로 자리 잡은 MPI(Message passing Interface)나 OpenMP(Open Multi-Processing)가 실행되어야 한다. 타키온2의 슈퍼컴퓨터 자원은 MPI 병렬 라이브러리와 OpenMP 컴파일러를 모듈로 지원해주고 있다. 본 연구는 고성능컴퓨팅의 표준 병렬프로그래밍 모델을 소개하고 슈퍼컴퓨터에서 R의 병렬처리 실행 방법 설명을 목적으로 한다.

본 연구의 구성은 다음과 같다. 제 2장에서는 기존의 사례 연구들을 살펴본다. 제 3장에서는 고성능 컴퓨팅 기술을 설명하고 R에서의 고성능 컴퓨팅 패키지들을 소개한다. 제 4장에서는 슈퍼컴퓨팅의 사용법과 병렬처리 실험을 통해 얻어지는 성능을 살펴보고 비교한다. 제 5장에서는 앞 절에서 논의한 내용을 간략히 정리하고 추후 나아갈 수 있는 연구 방향을 제기함으로써 글을 맺는다.

2. 기존연구

학계에서 R을 통한 병렬처리에 관한 기존연구로는 2가지로 나눠볼 수 있다. 하나는 메모리 구조에 따른 병렬 패키지 개발에 대한 연구와 다른 하나는 분산처리 플랫폼인 하둡(Hadoop)과 연동으로 나누어진다. 본 연구에서는 메모리 구조에 따른 병렬 패키지 개발에 대한 연구를 다루도록 한다.

병렬 패키지 snow와 snowfall의 성능 평가를 하여 snow 패키지에서 사용할 수 있는 통신 방법인 MPI와 SOCKET의 비교를 통해 4개 코어 이하에서는 SOCKET 방식이 MPI 방식보다 배깅 트리 알고리즘에서 있어서 연산 속도가 빠름을 보였다[1]. 대용량 처리 패키지에서는 bigmemory와 병렬 패키지와의 연동 방법을 소개하였고 코어의 개수가 늘어남에 따라 snow 패키지와 mulitcore 패키지의 연산 속도를 비교하였다. 코어의 개수가 늘어날 때 bigmemory 패키지를 사용한 경우가 사용하지 않는 것에 비해서 snow 패키지의 계산 성능이 우수하다는 것을 제시하였다[13]. 병렬 패키지 중에 고장 감내성과 부하 분산의 두 가지 특징 모두를 만족하는 패키지는 snowFT이며 사용성 측면에서는 pnmath 패키지가 우수하다고 제시하였다[20]. 지금까지 기존 논문에서는 R의 병렬 패키지를 소개하고 계산 실험을 하면서 얻어진 연구들의 결과들이다. 본 연구에서는 기존 연구에 더하여 소규모 클러스터가 아닌 슈퍼컴퓨터를 이용한 대규모 클러

스터에서 실험을 하였다. 컴퓨터와 계산 코어의 숫자가 늘어남에 따라 연산 속도가 얼마큼 향상되는지를 확인하고자 한다.

3. 고성능 컴퓨팅 기술

3.1 병렬 컴퓨팅 기술

전통적으로 컴퓨터 계산을 사용하는 기상예보, 유체역학, 화학, 천문학 등과 같은 고차원 행렬연산이 필요한 분야에서 고성능 컴퓨팅 기술이 필요했지만 현재에 들어서는 다양한 분야에서 컴퓨터 계산 문제에 직면하고 있다. 과거보다 빠른 네트워크와 다중 프로세서 시스템의 등장으로 병렬 컴퓨팅 환경을 구축하기 용이해지면서 상대적으로 값싼 컴퓨터 여러 대를 묶어 하나로 사용함으로써 컴퓨터의 성능을 향상시킬 수 있다. 병렬 컴퓨팅 기술의 장점은 계산 속도를 줄임으로써 시간을 절약할 수 있고 시간이 오래 걸려 처리하지 못했던 순차적 계산의 한계를 극복할 수 있는 점이다. 병렬 컴퓨팅 구조는 메모리 접근 방식에 따라 공유 메모리 시스템, 분산 메모리 시스템의 두 가지로 분류할 수 있다.

3.1.1 공유 메모리 시스템

공유 메모리 시스템은 여러 개의 프로세서가 하나의 공동 메모리를 사용하는데 프로그램이 동시에 접근할 수 있는 메모리로 코어들 간의 데이터가 공유된다. OpenMP가 대표적인 공유 메모리 구조를 지닌 멀티 쓰레드 라이브러리이다. OpenMP는 C, C++, 포트란 언어 등으로 사용할 수 있으며 유닉스 및 윈도우 시스템에서 지원하고 있다. 그 밖에 POSIX 쓰레드(Pthreads)가 있는데 POSIX에서 표준으로 제안한 쓰레드 함수의 모음이다. C 언어에서만 지원해주고 있다. 유닉스의 API인 fork의 경우에는 프로세스가 자식 프로세스를 생성하는데 부모 프로세스와 자식 프로세스는 메모리를 같이 사용하기 때문에 공유 메모리 영역에 속할 수 있다.

3.1.2 분산 메모리 시스템

분산 메모리는 각 컴퓨터마다 개별 메모리를 가지는 시스템으로 OpenMP가 공유 메모리 시스템에서의 표준이었다면 MPI(Message Passing Interface)는 분산 메모리 시스템에서의 표준 모델이다. MPI는 병렬로 실행되는 프로세스간의 커뮤니케이션이 메시지 전달에 의해서 이루어진다. 메시지 전달은 통신으로 시행되는데 같은 커뮤니케이터 내의 프로세스 사이에서만 통신이 이뤄지고 고유 정수의 할당된 번호로 프로세스를 식별한다[3]. 컴퓨터 프로세서의 확장에는 값비싼 컴퓨터 자원이 필요하고 한계가 있지만 분산 메모리 시스템은 저가로 클러스터 장비를 구축할 수 있는 장점을 지녔다.

<표 1>에는 대표적인 병렬프로그래밍 모델인 OpenMP와 MPI의 공통점과 차이점을 요약하였다.

〈표 1〉 OpenMP와 MPI 비교

특징	OpenMP	MPI
구조	공유 메모리	분산 메모리
공통점	MIMD(다수의 명령어가 다수의 데이터 처리 구분) C, Fortran에서 지원	
차이점	1) 쓰레드 기준 할당 2) 지시어 기반 3) 사용하기 용이하며 디버깅이 간편함.	1) 프로세스 기준 할당 2) 메시지 기반 3) 최적화하기 효율적이고 유연함.

3.1.3 병렬프로그래밍 예제

프로그래밍 언어를 학습할 때 기본 예제로 쓰이는 Hello World 프로그램을 병렬프로그래밍 모델을 이용하여 C언어로 작성하였다. Code 1은 공유 메모리의 병렬프로그래밍 모델인 OpenMP를 이용하여 작성한 코드이다.

OpenMP는 컴파일러 지시어, 런타임 라이브러리, 환경변수 3가지로 구성되어 있다. Code1에서 5번째 줄인 #pragma omp parallel는 컴파일러에게 병렬 작업을 쓰레드로 명령하기 위한 지시어이며 7번째 줄인 omp_get_thread_num()는 참여하고 있는 쓰레드의 숫자를 지정하는 병렬 파라미터로 런타임 라이브러리이다. C 컴파일러인 gcc로 컴파일

을 한 후에 OMP_NUM_THREADS 명령어로 환경변수를 설정해줄 수 있다. 환경변수는 시스템에서 사용할 쓰레드의 숫자를 의미한다.

Code1. OpenMP Example HelloWorld_OMP.c

```
#include<stdio.h>
#include<omp.h>
int main()
{
#pragma omp parallel
{
printf("Hello World %d\n", omp_get_thread_num());
}
return 0;
}
```

```
$ gcc -fopenmp -o HelloWorld_OMP.x
HelloWorld_OMP.c
$ export OMP_NUM_THREADS = 4
$ ./HelloWorld_OMP.x
```

```
$OpenMP 실행 결과
Hello World 2
Hello World 3
Hello World 0
Hello World 1
```

OpenMP 실행 결과를 보면 4개의 쓰레드에서 Hello World 프로그램이 실행되었으며 수행된 결과가 환경변수에 설정한 쓰레드 전체 개수와 동일하다. 쓰레드의 아이디는 0부터 3까지 리턴이 되며 먼저 들어오는 쓰레드 순서에 따라 출력되는 것을 확인할 수 있다. 다음의 Code 2는 분산 메모리의 병렬프로그래밍 모델인 MPI를 이용하여 작성한 코드이다.

먼저 MPI 헤더파일을 불러온 후에 함수들을 불러온다. Code 2에서 8번째 줄인 MPI_Init(&argc, &argv)에서 라이브러리를 초기화 해준다. 초기화와 종료는 반드시 한번 식 호출되어야 한다. 호출이 되면 통신할 수 있는 프로세스들의 집합인 커뮤니케이터에 속하게 되는데 각 프로세스를 식별하기 위해서는 식별 번호와 사이즈를 지정해야 한다.

Code 2의 9번째 줄과 10번째 줄에서 MPI_COMM_WORLD의 커뮤니케이터 내에 식별 번호와 속한 프로세스의 총 사이즈 개수가 결정된다. Code 2의 11번 째 줄에서 프로세스의 이름을 가져온 뒤 출력

Code2. MPI Example HelloWorld_MPI.c

```
#include<stdio.h>
#include<mpi.h>
int main(int argc, char *argv[])
{
int nRank, nProcs;
char procName[MPI_MAX_PROCESSOR_NAME];
int nNameLen;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &nRank);
MPI_Comm_size(MPI_COMM_WORLD, &nProcs);
MPI_Get_processor_name(procName,&nNameLen);
printf("Hello World. (Process Name=%s, nRank=%d,
nProcs=%d)\n", procName, nRank, nProcs);
MPI_Finalize();
return 0;
}
```

```
$ mpicc -o HelloWorld_MPI.x HelloWorld_MPI.c
$ mpirun -np 4 -hostfile hosts ./HelloWorld_MPI.x
```

하고 14번 째 줄에서 MPI를 종료함으로써 자원들을 반환한다. MPI를 종료하고 나서 MPI_Init 함수를 제외하고 어떠한 MPI 함수도 사용할 수 없다. 컴파일은 mpicc 명령어로 할 수 있고 컴파일 된 파일을 mpirun 명령어를 이용하여 작업을 실행한다. 이 코드에서는 4개의 병렬 프로세스가 실행된다.

```
$MPI 실행 결과
Hello World (Process name = c3, nRank = 2, nProces = 4)
Hello World (Process name = c2, nRank = 1, nProces = 4)
Hello World (Process name = c4, nRank = 3, nProces = 4)
Hello World (Process name = c1, nRank = 0, nProces = 4)
```

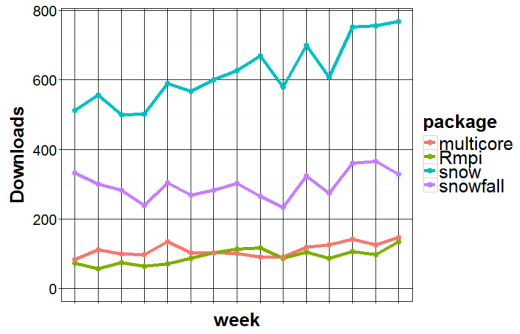
MPI 실행 결과를 보면 4개의 프로세스에서 Hello World 프로그램이 실행되었으며 프로세스의 이름, 식별번호, 사이즈가 출력되는 것을 확인할 수 있다. 식별번호는 0부터 3까지 랭크 번호가 할당이 되고 프로세스의 총 사이즈는 4개로 확인할 수 있다. 먼저 들어오는 프로세스 순서에 따라 출력되는 것을 확인할 수 있다.

3.2 고성능 컴퓨팅을 위한 R 패키지

3.2.1 분산 메모리 시스템용 R 패키지

R의 네트워크를 관리하는 CRAN(Comprehensive

R Archive Networks)에서는 R의 패키지를 33개의 주제로 분류 관리하고 있다. 그 중 하나의 주제인 HPC (High Performance Computing)는 고성능 컴퓨팅과 관련된 패키지들을 설명하고 있다[6]. CRAN 서버인 0-Cloud에서는 패키지 다운로드 숫자를 집계하고 있는데 2014년 01월부터 2014년 04월까지 4개월 동안의 R의 HPC 관련 패키지의 다운로드 순위를 ggplot2[27] 패키지를 통해 시각화하였다. [그림 1]에서 보면 snow 패키지가 가장 인기가 많았으며 snowfall, multicore, Rmpi 패키지 순인 것을 알 수 있다.



[그림 1] CRAN Download Log R HPC Packages

HPC 분야 안에서 분산 메모리 시스템을 위한 패키지는 외재적 병렬(Explicit parallelism) 카테고리에 들어있다. 대표적으로 snow, Rmpi, nws, snowfall, snowFT 패키지 등이다. 가장 널리 알려진 패키지는 snow[24]이다. snow은 R 명령어를 병렬로 처리하기 위한 함수들을 제공해주고 있다. snow는 병렬처리를 위해 마스터/슬레이브 구조를 사용하는데 마스터는 슬레이브에 있는 작업을 분산시키는 역할을 하고 슬레이브는 실제 작업을 수행한 후 결과를 마스터에 보내는 역할을 한다. snow 패키지의 특징은 다양한 통신 방법들을 제공해준다. SOCKET 통신, MPI 통신, PVM 통신, NWS 통신을 이용해서 클러스터를 설정할 수 있다. socket 방식에서는 공개키(RSA)를 클러스터 내에서 공유하는 방식을 사용한다. MPI 통신을 이용할 때에는 Rmpi 패키지가 설치되어 있어야 하며 설치 전에 MPI 라이브러리가 불려온 상태에서 클러스터를 설정할 수 있다. Rmpi[28]는 MPI의 함수들을 가져다 그대로 사용할 수 있는 패키지이다. snow 패키지에 비해 데이터를 전달하는 통신에서 MPI 함수를 사용하는 세부적인 기능을 수행할 수 있다. nws[19]는 Revolution Analytics 회사의 전신인 REvolution Computing에서 개발한 netWorkSpaces 서버를 이용하는 패키지이다. netWorkSpaces는 R 뿐만 아니라 다른 스크립트 언어인 Python, Matlab에서도 가능한 프레임워크이다. snowfall[14]는 snow와 동일한 기반 환경에서 작동되며 snow 패키지보다 이용성 측면에서 좀 더 편리하게 클러스터를 관리할 수 있도록 설계되었다. snowFT[22]는 snow 패키지에 없는 고장 감내성의 특징을 사용자들에게 제공하여 재현성 있는 실행이 가능한 패키지이다.

3.2.2 공유 메모리 시스템용 R 패키지

공유 메모리 시스템을 위한 패키지는 내재적 병렬(Implicit parallelism) 카테고리에 들어있다. 대표적으로 pnmath, fork, multicore, Rdsm 패키지 등이다.

pnmath[23]는 CRAN에 정식으로 등록되어 있는 패키지는 아니며 현재 0.04버전까지 소스 파일이 공개되어 있다. pnmath는 R에서 사용하는 표준 수학 함수들(예 : sqrt(), dnorm(), ptukey() 등 89가지 함수)을 OpenMP 버전의 함수들로 대체하였다. fork[26]는 유닉스 계열에서 새로운 프로세스를 만들고 관리하는 fork의 API 함수들을 제공한다. multicore[25]는 내재적 병렬 패키지 중에서 가장 쉽게 실행할 수 있고 외재적 병렬 패키지인 snow와 비교하여 많은 예제가 소개되어 있다. multicore 패키지는 fork를 기반으로 실행되기 때문에 POSIX 기반 OS에서만 사용할 수 있으며 현재 윈도우 버전에서는 지원해주지 않고 있다. Rdsm[17]는 snow 패키지의 클러스터 함수를 이용하여 쓰레드 환경에서 병렬처리가 가능하도록 구현되었으며 쓰레드 내에 공유할 수 있는 변수에 bigmemory 패키지에 있는 big.matrix 객체를 사용할 수 있는 특징을 지니고 있다.

3.2.3 대용량 처리 R 패키지

R은 기본적으로 데이터세트 전체가 R 소프트웨어의 메모리 공간에 적재되어야 사용할 수 있다. R의 버전이 올라감에 따라 적재 가능 데이터 크기가 커지기는 했지만 여전히 데이터를 메모리에 적재하기 때문에 코어 팀에서는 데이터의 크기가 컴퓨터 메인 메모리의 10~20%의 보다 크면 적합하지 않다고 말하고 있다. R Core Team[18]이 한계를 극복하기 위해 데이터를 메모리가 아닌 하드디스크에 저장하는 방법이 HPC 분야 안에서 소개한 대용량 처리 패키지(Large memory and out-of-memory data) 카테고리에 들어있다. 대표적으로 ff, bigmemory, ffbase, biglm, speedglm, bigrags, bigrf, biganalytics 등이 있는데 이 패키지들은 데이터 처리 패키지와 데이터 분석 패키지로 나누어볼 수 있다.

1) 데이터 처리 패키지 : ff, bigmemory,
2) 데이터 분석 패키지 : ffbase, biglm, speedglm, biglars, bigrf, biganalytics

데이터 처리 패키지 ff[4]는 useR! 2007에서 처음으로 소개되었으며 메인 메모리 대신 이진 플랫 파일로 하드 디스크에 저장하여 관리하는 방법을 사용한다. 여러 개의 대용량 데이터가 동시에 사용될 수 있으며 메인 메모리에 제한되지 않는다. ff는 두 가지 층으로 구성되는데 C++으로 쓰여진 low-level 층과 R로 쓰여진 high-level 층이다. ff가 사용 가능한 운영 체제는 윈도우, 리눅스, Mac OS가 가능하다. 현재 ff 패키지는 분석 패키지인 ffbase, biglm, biglars, bigrf 패키지 등과 연동하여 사용할 수 있다. bigmemory[9]는 ff와 유사한 패키지로 대용량 매트릭스 형태로 조작하여 저장할 수 있다. 매트릭스는 공유 메모리에 할당된 메모리에 매핑된 파일로 남아 사용될 수 있다. R에서는 값에 의한 호출 방법을 이용해서 데이터를 처리하는 반면 bigmemory는 참조에 의한 호출 방법을 이용하여 데이터를 파일로 백업하여 재사용이 용이하며 동일 컴퓨터에서 여러 개의 프로세스가 데이터에 접근하여 이용할 수 있다. bigmemory는 유닉스 운영체제에서만 가능하며 double, integer, short, char와 같이 C/C++ 자료 구조로 데이터를 통일해야 한다. 현재 bigmemory 패키지는 분석 패키지인 biganalytics, bigtabulate, bigalgebra, synchronicity 패키지와 연동하여 사용할 수 있다[7]. 그 외 패키지에는 소개하지 않았지만 레볼루션 어낼리틱스(Revolution Analytics Inc.) 회사에서 상용 제품으로 출시한 Revolution R Enterprise가 있다. 병렬 처리가 가능한 상업용 R 배포판을 학생들에게는 아카데미 버전으로 무료로 제공해주고 있는데 현재 7.2.0버전까지 출시되었다. 이 제품을 사용하면 ff, bigmemory 패키지보다 더 빠르게 데이터를 적재할 수 있고 회귀, 분류, 군집, 의사결정나무 등과 같은 통계알고리즘을 사용자에게 제공해주고 있다.

데이터 분석 패키지 ffbase[12]는 ff 패키지에 사용하는 데이터 타입인 ffdf을 가지고 기본적인 통계분석할 수 있는 함수들을 제공한다. 기초 통계량을 구하거나 히스토그램을 그릴 수 있다. biglm[16]은 계산 양이 너무 커서 메모리에 적재할 수 없는 데이터를 위한 선형회귀분석 패키지이다. 처음에는 데이터베이스 패키지와 연동해서 사용했지만 이후 ffbase, biganalytics 패키지가 개발되면서 biglm 함수를 그대로 사용하며 데이터 처리 패키지와 연동하였다. speedglm[10]는 대용량 데이터의 일반화 선형 모형 함수들을 제공한다. biglm 패키지와 다르게 QR분해를 사용하지 않고 정규식을 직접 풀어서 계산하여 많은 계산 시간이 소요되기 때문에 연산 속도를 빠르게 하기 위해 BLAS의 수치 계산 라이브러리를 사용하여 계산한다. biglars[21]은 대용량 데이터의 LAR(Least Angle Regression)와 Lasso 회귀를 계산할 수 있는 함수들을 제공한다. 또한 ff 패키지에 사용하는 데이터 타입인 ffdf을 사용할 수 있다. bigrf[15]은 메모리에 적재할 수 없는 대용량 데이터의 랜덤포레스트 알고리즘을 수행하는 함수들을 제공한다. biganalytics[8]은 bigmemory 패키지에 사용하는 데이터 big.matrix 타입을 가지고 통계 분

석할 수 있는 함수들을 제공한다. big.matirx 객체에 적용할 수 있는 apply 함수를 제공하고 군집분석과 기초통계량을 구할 수 있는 함수들을 제공한다.

4. 슈퍼컴퓨팅 실험

4.1 사용 방법

연구개발 수행에 슈퍼컴퓨팅 자원이 필요한 연구자는 KISTI의 심사를 받아 계정을 신청할 수 있다. 본 연구에서는 원격으로 로그인 노드에 접속한 후에 컴퓨팅 노드에 배치 스케줄러로 작업을 실행한다. 타키온 2는 클러스터 배치 스케줄러인 SGE(Sun Grid Engine)를 사용하고 있는데 사용 가능한 작업은 대기 시간 옵션과 코어의 숫자 옵션에 따라 우선순위가 결정되어 해당 큐의 작업이 진행된다. 작업 스크립트에는 MPI 라이브러리를 사용할 전체 작업의 개수를 설정할 수 있으며 컴퓨팅 노드의 코어 개수 또한 설정 가능하다. Tachyon 2는 <표 2>에서 보인 바와 같은 사양의 시스템이다. 노드는 하나의 컴퓨터를 의미한다고 보면 된다. 시스템의 한 노드 당 메모리는 24기가바이트이다.

〈표 2〉 Tachyon2 시스템 사양

구 분	내용	비고
프로세서	Intel Xeon X5570 2.93GHz(Nehalem)	Intel QPI Speed (6.4GT/S)
노드 수	컴퓨팅 노드 3,176대	로그인 노드 4개
CPU 코어	25,408개	8개/노드
메모리	DDR3/1333MHz 76.8TB	24GB/노드, 3GB/코어
운영 체제	RedHat Enterprise Linux 5.3	Kernal 2.6.18-194.17.1
소프트웨어	컴파일러 gcc 4.1.2 MVAPICH 1.2.7, MVAPICH2 1.5, OpenMPI 1.4.3, R 3.1.0	

R은 인터랙티브 모드와 배치 모드 두 가지로 실행할 수 있는데 인터렉티브 모드는 명령어를 입력하면 바로 실행 결과를 보여주고 다음 명령어를 입력하면 다른 결과를 보여주는 형태로 작동한다. 배치 모드는 저장된 R 스크립트를 한꺼번에 실행한 후 결과가 저장된다. 슈퍼컴퓨터에서 R을 실행하기 위해선 인터랙티브 모드가 아닌 배치 모드로 실행해야 하는데 그 이유는 인터랙티브 모드는 로그인 노드에서만 실행되기 때문이다. 로그인 노드에서는 MPI 라이브러리를 사용하더라도 컴퓨팅 노드를 사용할 수 없고 SGE 배치 스케줄러를 이용해야 컴퓨팅 노드를 사용할 수 있다. 작업 스크립트에서 R을 배치 모드로 실행할 수 있게 옵션을 설정할 수 있다.

4.2 분산 메모리에서의 성능 평가

성능 평가는 R에서 Rmpi 패키지를 사용하고 Open-MPI 1.4.3버전을 이용하여 수행하였다. 설정 가능한 MPI 타입은 MPICH, MPICH2, LAM-MPI, OPENMPI의 네 가지가 있는데 이번 실험에서는 OPENMPI로 설정하였다.

분산 메모리 성능 평가를 위해 사용하는 데이터는 Bioconductor의 Biobase[11] 패키지에서 제공하는 유전자 마이크로어레이 데이터이다. 애피메트릭스(Affxmetrix Inc.)의 DNA 칩으로 생성된 유전자를 가지고 전처리를 한 후에 나온 익명 처리된 데이터이다. 실제 유전자 데이터 자료의 일부분이며 유전자 데이터는 행렬 형태로 행은 유전자를 의미하며 열은 사람을 의미한다. [그림 2]는 각 500개 유전자 속성을 지닌 26명의 샘플 데이터이다.

26

row.names	A	B	A	…
AFFX-MurIL 2_at	192.742	85.7533	176.757	
AFFX-MurIL 10_at	97.137	126.196	77.9216	
AFFX-MurIL 4_at	45.8192	8.83135	33.0632	
AFFX-MurFAS_at	22.5445	3.60093	14.6883	
AFFX-BioB-5_at	96.7875	30.438	46.1271	
AFFX-BioB-M_at	89.073	25.8461	57.2033	
AFFX-BioB-3_at	265.964	181.08	164.926	
AFFX-BioC-5_at	110.136	57.2889	67.398	
…				

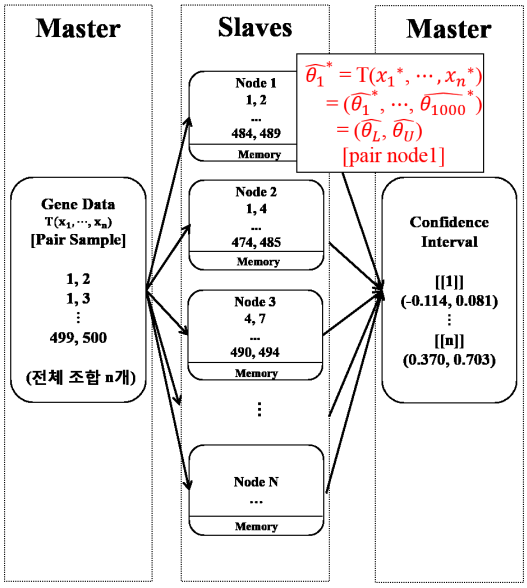
500

[그림 2] 애피메트릭스 유전자 데이터

모든 유전자 속성 간의 상관분석을 실시하기 위해 combn 함수를 이용해서 500개 유전자의 조합을 살펴보면 조합의 개수가 총 124,750개인 걸 확인할 수 있다.

성능 측정을 위하여 데이터 수가 작기 때문에 4배로 늘려 26명이 아닌 104명의 가짜 데이터를 생성하여 유전자 속성 조합 간의 상관분석을 실시한다. 분석에는 붓스트랩 패키지 boot[5]을 이용해서 1000번을 리샘플링하여 각 조합마다 1000개의 상관분석 결과를 얻어 BCa(bias-corrected and accelerated) 방법의 95% 신뢰구간을 구한다. 총 조합 124,750개 중에서 일부분만 뽑아내 조합 5,000개, 7,000개, 10,000개에 대해 Rmpi 함수를 이용하여 시뮬레이션을 진행하였다. Rmpi를 사용하기 위해서는 먼저 슬레이브 노드를 호출하고 슬레이브 노드에게 데이터와 함수를 전송한 후 apply 계열의 함수인 mpi.lapply 함수로 계산하는 과정을 거치게 된다. [그림 3]은 마스터와 슬레이브 노드 간의 병렬처리를 설명한 예제이다.

Rmpi 패키지 코드가 실행되는 순서는 7가지이다.



[그림 3] 노드 간의 병렬처리 작동 예제

1. Rmpi 패키지 로딩
2. 슬레이브 노드 생성
3. 함수와 데이터를 슬레이브 노드에게 전달
4. 슬레이브가 함수를 실행
5. 슬레이브와 커뮤니케이션을 하면서 계산 결과를 확인
6. 각 결과를 리스트 형태로 처리
7. 슬레이브 노드를 해제하고 mpi 종료

Rmpi가 작동되는 자세한 코드는 다음 Code 3에서 확인할 수 있다.

Code 3. Rmpi 코드

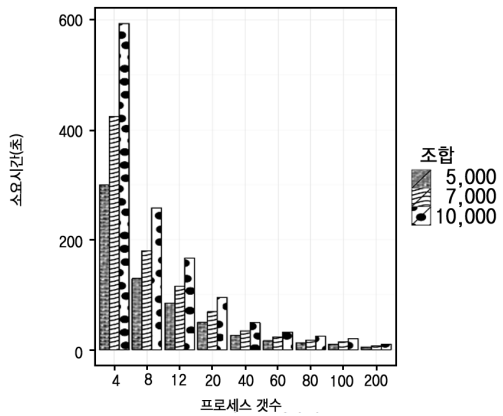
```
library(Rmpi)
library(boot)
mpi.spawn.Rslaves() #슬레이브 노드 생성
mpi.remote.exec(paste(Sys.info()[c("nodename")],"checking in as",mpi.comm.rank(),"of",mpi.comm.size())) #생성된 노드 이름 확인
set.seed(2310) #마스터 노드에서 동일한 값을 얻기 위한 난수값 적용
mpi.setup.rngstream(iseed = 2310) #슬레이브 노드에 적용될 난수값 적용
data(geneData, package = 'Biobase') #유전자 데이터 로딩

pair <- combn(1:nrow(geneData), 2, simplify = F) #조합의 개수 저장
fakeData <- cbind(geneData, geneData, geneData, geneData) #104명의 가짜 데이터 생성
pair2 <- sample(pair, 5000) #조합 중 5000개만 추출
geneCor2 <- function(x, gene = fakeData){
mydata <- cbind(gene[x[1], ], gene[x[2], ])
mycor <- function(x, i) cor(x[i,1], x[i,2])
boot.out <- boot(mydata, mycor, 1000)
boot.ci(boot.out, type="bca")$bca[4:5]
} #1000번 리샘플링한 후 상관분석에 대한 신뢰구간 계산 함수

mpi.bcast.Robj2slave(geneCor2, all=TRUE) #상관분석 함수를 슬레이브 노드에게 전달
mpi.bcast.Robj2slave(fakeData, all=TRUE) #데이터를 슬레이브 노드에게 전달
mpi.remote.exec(set.seed(2310)) #슬레이브 노드에서 난수생성
system.time(out <- mpi.parLapply(pair2, geneCor2)) #슬레이브 노드에서 계산을 실행

head(out)
length(out)
tail(out)
mpi.close.Rslaves() #슬레이브 노드를 해제하고 mpi 종료
q(save="no") #R 종료
```

실험은 하나의 컴퓨팅 노드에 4개 코어 식 사용하고 컴퓨팅 노드의 숫자를 늘려가면서 총 200개의 프로세스를 사용할 때까지에 소요시간의 변화가 있는지 체크하였다. 실험은 똑같은 환경에서 10번 반복하여 수행되었다.



[그림 4] 프로세스 숫자에 따른 평균 소요시간

[그림 4]에서는 Rmpi 프로세스 숫자에 따른 상관분석 소요시간을 ggplot2 패키지를 통해 시각화했는데 프로세스 숫자가 늘어날수록 전체적인 pair 조합의 상관분석의 연산 속도가 점점 줄어드는 것을 알 수 있다. 조합의 숫자가 많은 경우, 곧 계산량이 많아질수록 연산 속도의 감소폭이 커 효과가 증가하는 걸 알 수 있다. <표 3>에서 보면 5,000개 조합을 1개 프로세스에서 실행할 경우에는 연산 속도가 약 790초이었는데 200개 프로세스를 사용했을 경우 약 5초 정도로 연산 속도가 크게 줄어드는 것을 알 수 있다. 1개 프로세스를 사용할 때보다 200개의 프로세스를 사용할 때 모든 조합에 있어 평균적으로 연산 속도가 150배 이상 빨라진다는 것을 실험을 통해 확인할 수 있다.

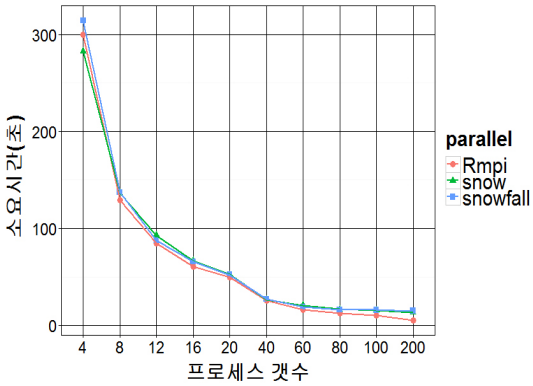
〈표 3〉 순차처리와 병렬처리 소요 시간 비교

pair	1 Process	200 Process	speedup
5,000	약 790초	약 5초	약 158배
7,000	약 1120초	약 7.5초	약 149배
10,000	약 1640초	약 10초	약 164배

Rmpi 패키지 이외에 분산 메모리 시스템에서 사용하는 병렬 패키지인 snow와 snowfall 패키지를 비교하였다. snow, snowfall 패키지는 SOCKET과 MPI 통신을 사용해서 데이터를 전송할 수 있는데 이번 실험에서는 MPI 통신을 사용해서 패키지를 실험하였다. Code 3과 동일한 유전자 데이터의 조합 5,000개에 대한 상관분석을 실시하여 똑같은 환경에서 10번 반복하여 수행되었다. 다음의 Code 4에서 자세한 코드를 확인할 수 있다.

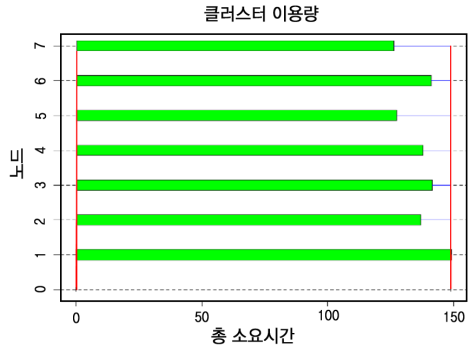
Code 4. snow, snowfall 패키지 코드

```
#Code 3과 중복된 코드는 제외
#snow package
library(Rmpi)
library(snow)
cl <- makeCluster(mpi.universe.size()-1, type="MPI")
#클러스터 생성
clusterCall(cl, function() Sys.info()[c("nodename",
"machine")])
#생성된 노드 이름 확인
clusterSetupRNG(cl, type="RNGstream",
seed=c(1,2,3,4,5,6))
#클러스터에 사용하는 균일한 난수 생성
clusterExport(cl, c("fakeData", "pair2", "geneCor2")
#모든 슬레이브 노드에게 함수와 데이터를 전달
clusterEvalQ(cl, library(boot))
#모든 슬레이브 노드에서 boot 패키지 로딩
system.time(out <- parLapply(cl, pair2, geneCor2))
#슬레이브 노드에서 계산을 실행
stopCluster(cl)
#snowfall package
library(Rmpi)
library(snowfall)
sfSetMaxCPUs(cpus)
#cpu가 32개 이상일 경우 함수로 노드 갯수 설정
sfInint(parallel = TRUE, cpus = cpus, type="MPI")
#클러스터 생성
cl <- sfGetCluster()
clusterCall(cl, function() Sys.info()[c("nodename",
"machine")])
sfClusterSetupRNG(seed=c(1,2,3,4,5,6))
#클러스터에 사용하는 균일한 난수 생성
sfExport(list="fakeData", "pair2", "geneCor2"))
#모든 슬레이브 노드에게 함수와 데이터를 전달
sfLibrary(boot)
#모든 슬레이브 노드에서 boot 패키지 로딩
system.time(out <- sfLapply(pair2, geneCor2))
#슬레이브 노드에서 계산을 실행
sfStop()
```



[그림 5] 병렬 패키지 연산속도 비교

[그림 5]에서 분산 메모리 시스템의 패키지인 Rmpi, snow, snowfall의 연산 속도를 비교했는데 같은 통신방법인 MPI를 사용하고 있기 때문에 연산 속도에는 큰 차이가 없었지만 프로세스 개수가 늘어남에 따라 Rmpi 패키지의 성능이 그 중에서 가장 우수한 것을 알 수 있었다. snow 패키지는 통신을 하면서 발생하는 데이터 전송 시간과 각 슬레이브 노드에서의 소요 시간을 [그림 6]에서 확인할 수 있다.



[그림 6] snow 함수를 사용한 소요시간 계산

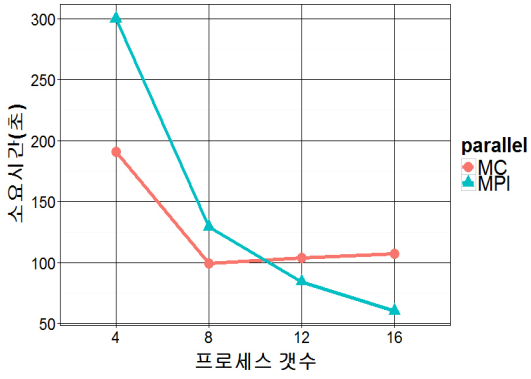
4.3 공유 메모리에서의 성능 평가

공유 메모리에서 성능 평가는 유닉스의 fork API 기반에 자식 프로세스를 생성하는 multicore 패키지를 이용하는 방법과 OpenMP 컴파일러를 이용하는 두 가지 방법을 실험하였다. 먼저 multicore 패키지를 실험했는데 Code 3, 4의 실험과 동일한 유전자 데이터의 조합 5,000개에 대한 상관분석을 실시하였다. multicore 패키지에서는 apply 계열 함수인 mclapply 함수로 계산하는 과정을 거치게 된다. MPI와 다르게 multicore는 공유 메모리 시스템을 사용하기 때문에 데이터는 전송하지 않는다. 실험은 똑같은 환경에서 10번 반복하여 수행되었다. 다음의 Code5에서 자세한 코드를 확인할 수 있다.

Code 5. multicore 코드

```
#Code 3과 중복된 코드는 제외

library(multicore)
library(parallel)
cores <- detectCores() # 시스템의 코어 갯수 확인
system.time(out <- mclapply(pair2, geneCor2,
mc.set.seed = TRUE, mc.cores=cores)) #lapply 함수의 병렬 버전
```



[그림 7] 공유 메모리(MC)와 분산 메모리(MPI) 비교

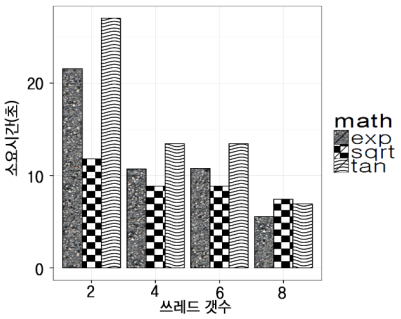
[그림 7]에서 보면 전체 프로세스 숫자가 같을 때 계산 방법에 따라 연산 속도의 차이가 발생하는데 MPI같은 경우에는 프로세스 개수가 4개 일 때에는 연산 속도가 multicore 방법보다 크지만 프로세스 개수가 늘어나면서 multicore 방법에 비해 연산 속도가 줄어드는 것을 확인할 수 있다. 이를 통해 대용량 연산에서 MPI의 분산 메모리 시스템이 fork을 이용하는 공유 메모리 시스템보다

효과적인 걸 알 수 있다. 공유 메모리를 사용하는 multicore의 경우에는 프로세스의 숫자가 8개 이상일 경우에는 연산 속도가 오히려 증가하고 있는데 이 실험에서 사용하는 계산 노드 한 대당 코어의 개수가 8개이기 때문에 8개 이상일 경우에는 연산 속도의 변화가 없음을 알 수 있었다. 일반적으로 multicore 패키지에서 프로세스의 개수를 설정할 때에는 코어 개수만큼 프로세스를 설정하여 시작하는 것을 추천하고 있는데 parallel 패키지에 detectCores() 함수를 사용하면 시스템에 사용할 수 있는 프로세스 개수를 확인할 수 있다[2].

R에서 OpenMP 컴파일러를 이용하는 방법은 2가지로 나눌 수 있다. 첫 번째는 C언어로 계산이 오래 걸리는 문제가 발생하는 부분을 작성한 다음 코드를 컴파일러한 후 C코드를 R 함수로 불러와서 계산하는 방식이 있고 두 번째는 OpenMP를 이용하는 패키지들을 사용하는 방식이 있다. 이번 실험에서는 후자의 방법인 OpenMP를 사용하는 pnmath 패키지를 실험하였다(pnmath는 제 3절 참조). R에서 OpenMP 실행하기 위해 작업 스크립트 옵션을 설정해준 뒤에 배치 스케줄러인 SGE을 이용하여 작업을 제출한다. 실험에 사용하는 코드는 랜덤하게 균등분포(uniform distribution)로부터 10억 개를 생성한 후 sqrt() 함수, exp() 함수, tan() 함수를 쓰레드를 늘려가면서 성능의 차이가 있는지 살펴보도록 하였다. 실험은 똑같은 환경에서 10번 반복하여 수행되었다. 다음의 Code 6에서 자세한 코드를 확인할 수 있다.

Code 6. pnmath 코드

```
library(pnmath) #패키지 로딩
Sys.setenv(OMP_NUM_THREADS=) #환경변수 설정
setNumPnmathThreads() #초기 쓰레드 설정
v1 <- runif(1000000000)
system.time(exp(v1))
system.time(sqrt(v1))
system.time(tan(v1))
q(save="no")
```



[그림 8] 쓰레드 숫자에 따른 평균 소요시간

[그림 8]에서는 OpenMP 쓰레드 개수를 2개부터 8개까지 늘리면서 수학 함수의 연산 속도를 비교했는데 3가지 수학 함수 모두 쓰레드 2개일 때 가장 연산 속도가 컸으며 쓰레드가 4개일 경우와 6개일 경우에는 큰 차이가 없었고 8개일 때 연산 속도가 줄어드는 것을 확인할 수 있었다.

5. 결 론

데이터를 획득하기가 과거보다 용이해지고 정보화 시대를 통해 데이터 생산이 가속화되면서 생활과 밀접하게 연관되어 있는 모든 부분이 데이터베이스에 저장되는 빅데이터 시대를 맞이하게 되었다. 하나의 코어에서는 도저히 처리할 수 없는 대용량 데이터의 계산연산 문제에 직면하게 되어 이 문제를 해결하기 위해 공유 메모리 시스템과 분산 메모리 시스템의 확장이 시도되었다. R 커뮤니티에서는 병렬 컴퓨팅 또는 분산 컴퓨팅 기술을 도입하기 위해 많은 패키지를 제공해주는데 본 연구에서는 슈퍼컴퓨터의 MPI 라이브러리와 OpenMP 컴파일러를 이용하는 R 패키지들을 실험하였다. 공유 메모리 시스템과 분산 메모리 시스템의 비교 실험을 통해 컴퓨팅이 많이 필요 하는 연산에서 MPI의 분산 메모리 시스템이 fork을 이용하는 공유 메모리 시스템보다 더 효과적인 것으로 나타났으며 같은 코어일 경우에는 공유 메모리의 성능이 약 1.3배 우세하다는 것을 알 수 있었다. 추후의 연구에서는 본 연구에서 제시한 실험에 그치지 않고 데이터마

이닝 알고리즘의 연산 속도를 비교하여 우수한 성능의 알고리즘을 찾는데 전념해야 할 것이다.

참 고 문 헌

[1] 박용민, 고영준, 김진석, "병렬 컴퓨팅을 위한 R 패키지 소개 및 성능평가", 「한국자료분석학회」, 제14권, 제4호(2012), pp.1951-1961.

[2] 서민구, [R을 이용한 데이터 분석 실무], 1판, http://r4pda.co.kr/, [Online; accessed Dec 2013].

[3] 이홍석, 김정한, 이승우, 이식, [멀티코어 시대에 꼭 알아야할 MPI 병렬프로그래밍], 1판, 어드북스, 2010.

[4] Adler, D., C. Glaser, O. Nenadic, J. Oehlschlagel, and W. Zucchini, "ff : Memory-Efficient Storage of Large Data on Disk and Fast Access Functions. R package version 2.2-13, URL http://cran.r-project.org/web/packages/ff," 2007.

[5] Canty, A. and B. Ripley, "boot : Bootstrap Functions(originally by Angelo Canty for S). R package version 1.3.11 URL http://cran.r-project.org/web/packages/boot/index.html," 1999.

[6] Eddelbuettel, D., "CRAN Task View : High-Performance and Parallel Computing with R, URL http://cran.r-project.org/web/views/HighPerformanceComputing.html," 2014.

[7] Emerson, J.W. and M.J. Kane, "The bigmemory Project Website, URL http://www.bigmemory.org/," 2010.

[8] Emerson, J.W. and M.J. Kane, "biganalytics : A Library of Utilities for big.matrix Objects of Package bigmemory. R package version 1.1.1, URL http://cran.r-project.org/web/packages/biganalytics/," 2010.

[9] Emerson, J.W., M.J. Kane, and P. Haverty, "bigmemory : Manage Massive Matrices with Shared Memrmoy and Memory-Mapped Files. R package version 4.4.6 URL http://cran. r-project.org/web/packages/bigmemory," 2008.

[10] Enea, M., "speedglm : Fitting Linear and Generalized Linear Models to large data sets, R package version 0.2 URL http://cran.r-project.org/web/packages/speedglm/index.html," 2012.

[11] Gentleman, R., V. Carey, M. Morgon, and S. Falcon, "Biobase : Base functions for Bioconductor. R package version 2.24.0 URL http://www.bioconductor.org/packages/release/bioc/html/Biobase.html," 2004.

[12] Jonge, E., J. Wijffels, and J.V. Laan, "ffbase : Basic statistical functions for package ff. R package version 0.11.3 URL http://cran.r-project.org/web/packages/ffbase/index.html," 2010.

[13] Kane, M.J., J.W. Emerson, and S. Weston, "Scalable Strategies for Computing with Massive Data," *Journal of Statistical Software*, Vol.55, No.14(2013), pp.323-341.

[14] Knaus, J., "snowfall : Easier cluster computing(based on snow). R package version 1.84-6, URL http://cran.r-project.org/web/packages/snowfall/index.html," 2010.

[15] Lim, A., L. Breiman, and A. Cutler, "bigrf : Big Random Forests : Classification and Regression Forests for Large Data Sets. R package version 0.1.11 URL http://cran.r-project.org/web/packages/bigrf/index.html," 2013.

[16] Lumley, T., "biglm : Bounded Memory Linear and Generalized Linear Models, R package version 0.9-1, URL http://cran.r-project.org/web/packages/biglm," 2006.

[17] Matloff, N., "Rdsm : Threads Environment for R. R package version 2.0.2 URL http://

cran.r-project.org/web/packages/Rdsm/index.html," 2010.

[18] R Core Team, "R Installation and Administration manual, URL http://cran.r-project.org/doc/manuals/R-admin.html," 2013.

[19] REvolution Computing, "nws : functions for NetWorkSpaces and Sleigh. R package version 1.7.0.1, URL http://cran.r-project.org/web/packages/nws/index.html," 2010.

[20] Schmidberger, M., M. Morgan, D. Eddelbuettel, H. Yu, L. Tierney, and U. Mansmann, "State of the Art in Parallel Computing with R," *Journal of Statistical Software*, Vol.31, No. 1(2009), pp.1-27.

[21] Seligman, M., C. Fraley, and T. Hesterberg, "biglars : Scalable Least-Angle Regression and Lasso. R package version 1.0.2 URL http://cran.r-project.org/web/packages/biglars/index.html," 2010.

[22] Sevcikova, H. and A.J. Rossini, "snowFT : Fault Tolerant Simple Network of Workstations. R package version 1.3-0, URL http://cran.r-project.org/web/packages/snowFT/index.html," 2012.

[23] Tierney, L., "pnmath : OpenMP parallel processing directives of recent compilers for implicit parallelism by replacing of internal R functions, R package version 0.0.4, URL http://homepage.stat.uiowa.edu/~luke/R/experimental/," 2010.

[24] Tierney, L., A.J. Rossini, N. Li, and H. Sevcikoca., "snow : Simple Network of Workstations, R package version 0.3-13, URL http://cran.r-project.org/web/packages/snow/index.html," 2003.

[25] Urbanek, S., "multicore : A stub pacakge to ease transition to 'parallel'. R package version 0.2 URL http://cran.r-project.org/web/packages/multicore/index.html," 2009.

[26] Warnes, G.R., "fork : R functions for handling multiple processes. R package version 1.2.4 URL http://cran.r-project.org/web/packages/fork/index.html," 2003.

[27] Wickham, H. and W. Chang, "ggplot2 : An Implementation of the Grammar of Graphics. R package version 1.0.0 URL http://cran.r-project.org/web/packages/ggplot2/index.html," 2007.

[28] Yu, H., "Rmpi : Interface (Wrapper) to MPI (Message-Passing Interface), R package version 0.6-5, URL http://cran.r-project.org/web/packages/Rmpi/index.html," 2002.