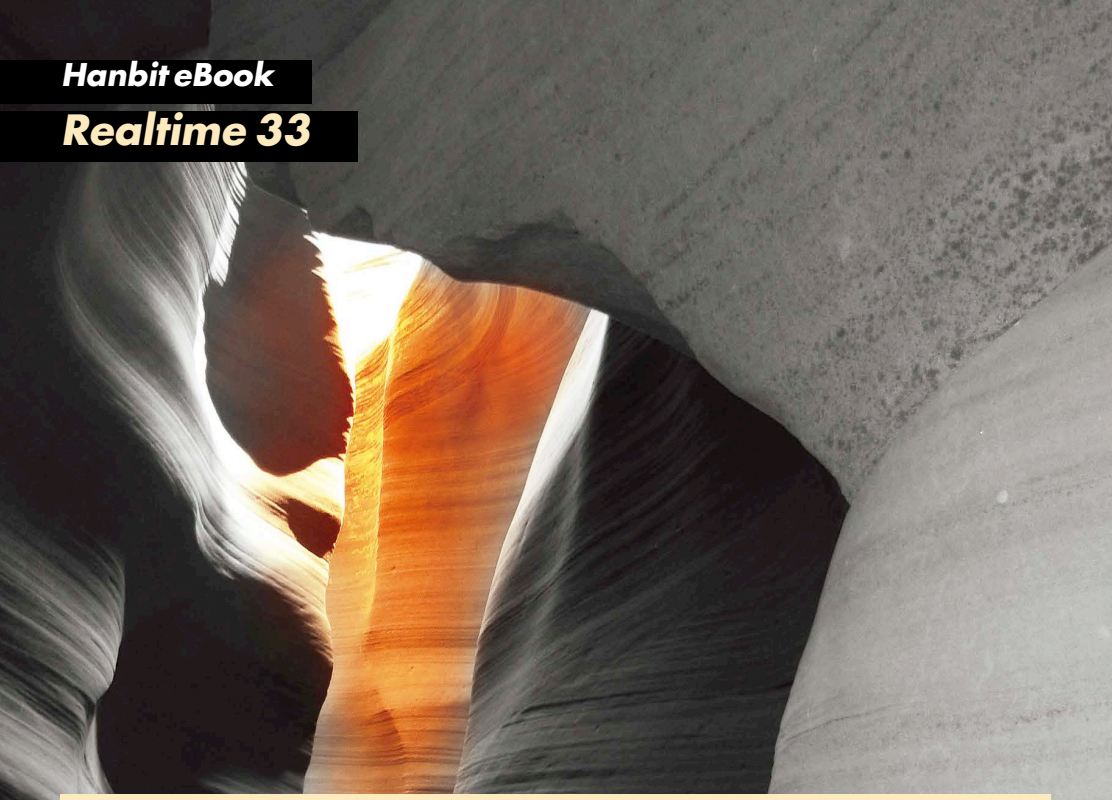


Hanbit eBook

Realtime 33

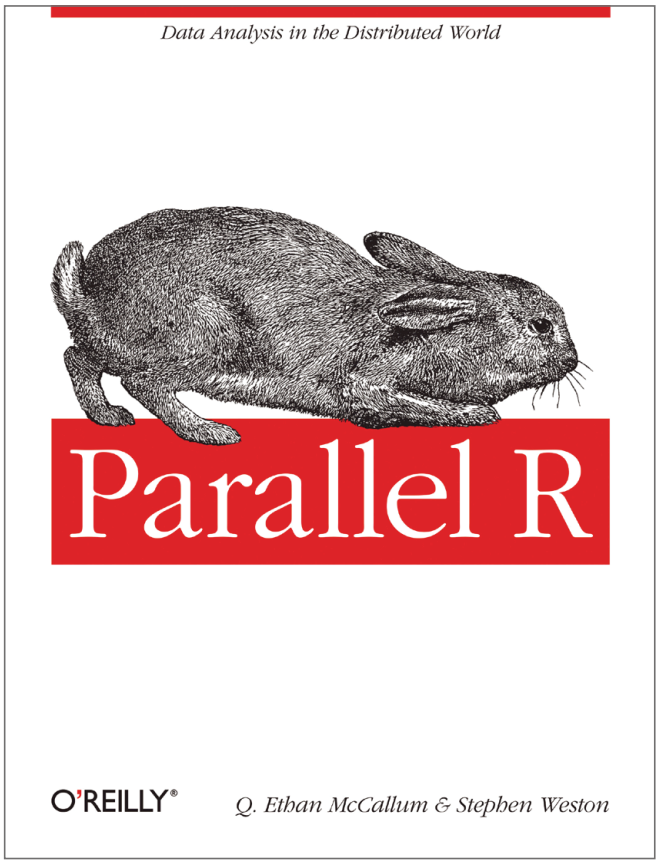
DATA SCIENCE

R 병렬 프로그래밍

빅데이터 분석을 위한 R 멀티코어 병렬 처리

Q. 에덴 맥컬럼, 스테판 웨스턴 지음 / 임재현 옮김

O'REILLY® 한빛미디어 Hanbit Media, Inc.



이 도서는 O'REILLY의
Parallel R의
번역서입니다.

DATA SCIENCE

R 병렬 프로그래밍

빅데이터 분석을 위한 R 멀티코어 병렬 처리

DATA SCIENCE **R 병렬 프로그래밍** 빅데이터 분석을 위한 R 멀티코어 병렬 처리

초판발행 2013년 6월 28일

지은이 Q. 에덴 맥컬럼, 스테판 웨스턴 / **옮긴이** 임재현 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-636-4 15000 / **정가** 9,900원

책임편집 배용석 / **기획·편집** 김병희
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 김현미
마케팅 박상용, 박주훈, 정민하

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanb.co.kr / **이메일** ask@hanb.co.kr

Published by HANBIT Media, Inc. Printed in Korea

저자 소개

지은이_ Q. 에덴 맥컬럼Q. Ethan McCallum

Q. 에덴 맥컬럼은 컨설턴트이자 작가이며, 기술 마니아로 오라일리 네트워크O'Reilly Network와 Java.net의 여러 작업에 참여하였으며, C/C++ Users Journal, Dobb's Journal, Linux Magazine 등에 글을 기고하기도 했습니다. 그는 회사들이 데이터와 기술을 통해 좀 더 나은 선택을 할 수 있도록 돕고 있습니다.

지은이_ 스테판 웨스턴Stephen Weston

스테판 웨스턴은 25년 이상 고성능 병렬 컴퓨팅 분야에서 일해왔습니다. 90년대에 Scientific Computing Associates에 입사하여 데이비드 겔렌터David Gelernter가 개발한 린다Linda 프로그래밍 시스템과 관련한 일을 맡아왔습니다. Revolution Computing, Inc.의 창립자이기도 하며, 이 회사에서는 nws, foreach, doSNOW, doMC 등 R을 이용한 병렬 컴퓨팅 패키지를 개발하고 있습니다. 현재는 예일 대학교에서 고성능 컴퓨팅 전문가로 활동하고 있습니다.

저자 서문

"수학 좋아하세요? 그럼 마이크와 얘기해보세요. 제가 소개해드리죠." 그때는 잘 몰랐지만, 이 한 마디가 이 프로젝트의 시작이었습니다. 마이크 루키디스Mike Loukides와 이메일 및 전화 연락을 주고받는 사이, 필자가 알아채기도 전에 우리는 새로운 책을 쓸 준비가 되어 있었습니다. 그래서 필자는 마이크와 필자를 연결해준 베스티와 로렐에게 깊은 감사를 표하고 싶습니다. 이 책의 공동 저자인 스테판 웨스턴Stephen Weston도 마이크와의 대화를 통해 만날 수 있었습니다. 그가 이 모험에 함께 참여하기로 했을 때 필자는 정말로 기뻤습니다. 특히 필자가 작업할 수 있는 별도의 공간을 제공해준 Cafe les Duex 고객 여러분께도 심심한 감사를 표합니다.

Q. 에덴 맥컬럼

이 책은 제 첫 집필이었기에, 초보적인 혼동과 실수들을 너그러이 이해해준 공동 저자와 편집자에게 깊은 감사를 표합니다. 그들은 프로젝트를 진행하는 동안 항상 제게 자애로운 모습을 보여주었습니다. 닉Nick, 롭Rob, 제드Jed 등은 제가 작업한 챕터를 모두 읽고 실수를 바로잡을 수 있도록 도와준 고마운 분들입니다. 또한 아내 다이애나와 딸 에리카는 이 책이 자신의 관심사가 아닌데도 원고를 읽고 교정해 주었습니다.마지막으로 이 책에서 언급한 모든 패키지를 개발한 분들께 깊은 감사를 드립니다. 제가 작성한 세 개의 패키지에 관한 코드를 살펴보며 즐거운 시간을 보낼 수 있었습니다. 특히 snow 관련 소스 코드는 제가 R 프로그래밍을 처음 공부하면서 가장 많이 참고한 코드이기도 합니다.

스테판 웨스턴

역자 소개

옮긴이_ **임재현**

바이오 빅데이터를 공부하고 있는 대학원생이며, 얼마 전에 결혼에 성공하여 신혼을 즐기고 있다. 어린 시절의 꿈이었던 불로장생의 영약을 만들기 위해 열심히 공부하고 있다. 대학원에 처음 들어왔을 때부터 R을 사용하였으며, 종종 부족한 실력이지만 R 강의를 하기도 하였다. 지금도 데이터의 가공 및 통계적 분석, 평가, 시각화에 이르기까지 작업 대부분에 R을 활용하고 있으며, 요즈음에는 리눅스 서버 환경에서 R을 효율적으로 사용하는 방법을 고민하고 있다.

역자 서문

제가 R을 처음 사용했을 때만 하더라도, R은 그리 유명한 툴이 아니었습니다. 대부분의 기업, 대학, 연구소 등에서는 통계 분석에 SAS나 SPSS 등을 활용하였고, R은 매우 한정적인 분야에서 활용하고 있었습니다. 서울의 대형서점에서도 R과 관련된 책을 구하기 어려웠고, 기본적인 패키지들조차 한글화된 메뉴얼이 없었습니다. 그랬기에, 최근에 온·오프라인에서 R이 중요한 이슈로 부각되고 있는 것이 한편으로는 놀랍고, 또 한편으로는 반갑습니다. R은 특히 최근 화두가 되고 있는 빅데이터, 비정형 데이터의 분석에 다른 툴들에 비해 강점이 있습니다. 데이터를 살펴보면서 다양한 통계적 방법을 대화형 인터프리터를 통해 실시간으로 적용해보고, 결과를 바로바로 시각화할 수 있으며, 기본적인 프로그래밍 기능을 제공해주기 때문입니다. R을 빅데이터 분석에 적용하는 데 가장 큰 문제가 되었던 것은 R이 한정적인 메모리를 사용하며, 서버에서 사용하기 어렵다는 점이었습니다. 이 책은 R의 이러한 문제를 해결하기 위해 세계의 R 사용자들이 어떤 노력을 기울이고 있으며, 문제를 어떤 식으로 해결할 수 있는가를 알 수 있는 책입니다. 저조차도 이 책을 번역하며 많은 것을 배울 수 있었습니다. R 교재를 만들거나 강의한 적은 있었지만, 직접 번역에 참여한 것은 처음이었습니다. 저의 초보적인 실수를 짚어주시고 인내심을 가지고 프로젝트를 진행해주신 에디터님에게 깊은 감사를 표합니다. 항상 저에게 친절하고 따스한 모습을 보여주시어 번역을 끝까지 진행할 수 있었습니다. 또, 이 책에서 언급한 모든 패키지의 개발자분들과 이 책의 저자분들께 감사의 마음을 전합니다. 이 책의 번역을 통해 정말 많이 배울 수 있었고, 코드를 살펴보는 것이 매우 큰 즐거움이었습니다. 마지막으로, 항상 저를 지원해주고 사랑해주는 아내와 가족들에게 사랑과 감사의 마음을 전합니다.

2013년 여름

임재현

대상 독자 및 예제 파일

초급 | 초중급 | **중급** | 중고급 | 고급

이 책은 R이 무엇인지, 어떻게 사용하는지 알고 있는 독자를 대상으로 R 병렬 프로그래밍 방법을 소개한다.

- 1장에서는 R 병렬 프로그래밍을 학습하기 위한 워밍업을 한다.
- 2장에서는 snow 패키지를 알아본다.
- 3장에서는 multicore 패키지를 알아본다.
- 4장에서는 parallel 패키지를 알아본다.
- 5장~8장에서는 맵리듀스와 하둡에 관해 간략하게 알아본 후, R에서 이를 어떻게 활용하는지를 나머지 장들에서 알아본다.
- 9장에서는 최근에 개발된 몇 가지 방법들에 대하여 살펴본다.

이 책에서 사용한 예제 파일은 다음 웹 사이트에서 받을 수 있다.

- http://examples.oreilly.com/0636920021421/

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위하여 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

1 | R 병렬 프로그램 시작하기

이 장에서는 이 책의 전반적인 내용을 간략히 훑어볼 것이다. 책 전체를 읽을 시간이 부족한 독자는 필요한 장을 선택하여 읽는 것도 좋다. "1.5 R 병렬 패키지 훑어보기"에, 이 책에서 소개할 패키지의 장단점을 정리해 두었으니 읽을 곳을 선택하는 데 도움이 될 것이다. 필요한 내용을 살펴본 후에는, 다시 이 장으로 돌아와서 용어와 패키지에 대해서 읽기를 권한다.

1.1 R을 사용하는 이유?

R을 써야 하는 이유는 자명하다. 고성능이며 다양한 플랫폼에서 사용할 수 있는데다 오픈 소스로 개발된 통계 소프트웨어를 누가 마다하겠는가? R은 데이터를 잘 살펴보기 위한 대화형 인터페이스를 갖추고 있으며, 원하는 분석과정을 반복하기 위하여 스크립트 언어처럼 사용할 수도 있다. 게다가 대부분의 통계적인 분석 알고리즘이 내장되어있어 직접 코드를 짜지 않아도 된다. 무엇보다도 R은 공짜다.

만약 기본적으로 제공된 함수들로 충분하지 않다면, 다른 사용자가 만든 수많은 애드온 패키지와 GUI 환경 등을 추가로 다운받아 사용할 수 있다. 이런 특징이 R을 빅데이터 시대의 화두로 만들어주었다.

그러나 R이 완벽하다면, 이 책은 여기에서 끝내도 괜찮을 것이다. 빅데이터에 R을 적용하기에는 몇 가지 문제점이 있다.

1.2 R의 한계점?

R의 불완전성은 소프트웨어 자체의 문제라기보다는 '시대의 문제'다. 좀 더 간단하게 말하면, R은 빅데이터 혁명이 이루어질 것을 예견하지 못하고 만들어졌다.

R이 처음 개발된 1995년에는 컴퓨터의 저장 용량이 그리 크지 않았으며, RAM은 더욱 그러하였다. 인터넷도 이제 막 걸음마를 떼던 시기였다. 빅데이터라던가 고성능 컴퓨팅과 같은 용어는 매우 드물게 사용되었다. 사실 월가Wall Street나 대학 연구실을 제외하고는 데이터라고 할만한 것이 부족하였다.

다시 현재로 돌아와서 생각해보면, 현재 하드웨어는 이전에 비해 매우 저렴해졌다. 컴퓨터를 이용한 계산도 온라인을 통해 단돈 몇 푼이면 해결할 수 있다. 이런 환경에서 많은 사람이 갑자기 데이터를 모으고 분석하는데 관심이 생기기 시작하였고, 필요한 대부분의 데이터를 쉽게 구할 수 있게 되었다.

이러한 데이터의 폭발은 R의 두 가지 단점을 부각시켰다. R은 스레드를 한 개만 사용하며, 메모리의 제한이 있다. R 언어 체계에는 스레드나 뮤텍스같은 병렬의 개념이 없기 때문에, R에 내장된 함수만을 사용해서는 CPU가 여러 개라도 활용할 수 있는 방법이 없다.

R은 사용자의 데이터셋 전체가 RAM[01]의 크기보다 작아야 사용할 수 있다. 램이 4GB라면 절대로 4GB보다 큰 데이터를 다룰 수 없다.

다행히도 이런 문제점이 극복하기 어려운 것은 아니다.

1.3 해결 방법: 병렬로 실행하기

최근 몇 년간, 많은 사람이 R의 문제점을 극복하기 위해 다양한 대안을 내놓았다. 큰 행렬을 다뤄야 할 때는, R에서 BLAS[02]를 불러들여 사용할 수 있다. 데이터의 크기가 클 경우에는 관계형 데이터베이스를 이용하여 데이터를 작게 나누어 다룰 수 있다.

---

01 이는 굉장히 큰 문제로, R은 특별한 이유 없이 여러 개의 데이터 객체를 생성하기 때문에 사용하고 있는 데이터셋보다도 더 큰 램을 요구하게 된다. 만약 충분한 메모리를 확보하지 못하면 작업 자체뿐 아니라 사용하고 있는 머신도 점점 제대로 동작하지 않게 될 것이다. Swapoff 명령어를 이용하여 가상 메모리를 강제로 종료하면 R이 메모리를 점유하는 것을 강제로 종료시킬 수 있다.

02 역자 주_ 70년대부터 개발되어온 행렬 계산용 프로그램으로, R을 인터페이스로 사용할 수 있다.

이러한 대안들 중 가장 각광받고 있는 것이 병렬 컴퓨팅이다. 작업을 여러 개의 CPU에 분산함으로써 R이 한 개의 스레드만을 가지고 있는 단점을 극복할 수 있다. 작업을 여러 대의 컴퓨터에 분산할 경우 멀티 스레드의 장점뿐 아니라 R의 메모리 문제도 해결할 수 있다. 이 책에서는 병렬 컴퓨팅을 R에 활용할 수 있는 방법을 알아볼 것이다.

1.4 전반적인 책 내용

지금까지 이 책의 목적에 대해 알아보았다. 이제 좀 더 구체적으로 나머지 내용이 어떻게 구성되어있는지 알아보겠다.

1.4.1 책에서 다루는 내용

각 챕터는 R 병렬 컴퓨팅을 위한 전략을 하나씩 소개할 것이다. 다음은 그 내용이다.

- R 병렬 컴퓨팅 전략이 무엇인가
- R 병렬 컴퓨팅을 위한 패키지를 어디에서 구할 수 있는가
- R 병렬 컴퓨팅을 어떻게 사용하는가
- R 병렬 컴퓨팅을 어떤 상황에서 써야 하는가

2장에서는 snow 패키지를 소개할 것이고, 3장에서는 multicore 패키지를 소개할 것이다. 그리고 4장에서는 R 2.14 버전 이후에 추가된 새로운 병렬 패키지인 parallel에 관하여 알아볼 것이다. 5장에서는 요즘 병렬 및 클라우드 컴퓨팅 환경으로 각광받고 있는 하둡과 맵리듀스에 관해 간략하게 알아본 후, R에서 이를 어떻게 활용하는지를 나머지 장들에서 알아볼 것이다.

1.4.2 추가로 알아 볼 것들

9장에서는, 가장 최근에 개발된 몇 가지 방법들에 대하여 살펴볼 것이다.

아마도 필자가 이 책을 저술할 시점에는 알지 못하던 다른 방법들이 있을 수도 있을 것이다.[03] http://parallelrbook.com/을 통하여 알려주면 좋겠다.

1.4.3 이 책을 읽기 전에 미리 알고 있어야 할 것들

이 책은 R에 관한 책이지만, 기본적인 사용법은 미리 알고 있어야 한다. 만약 R을 처음 사용하거나 R과 관련한 참고서가 필요하다면 O'Reilly에서 출판한 『R Cookbook』이나 Manning에서 출판한 『R in Action』 등을 기본서로 활용하는 것이 좋다. 특히, lapply() 함수는 이 책의 전반에 걸쳐 다채롭게 활용될 것이기 때문에 정확히 이해하고 있어야 한다.

몇 가지 주제는 큰 서버를 직접 다루어야 하기 때문에 관리자 계정이 필요할 수 있다. 또는 인터넷을 통하여 컴퓨팅 리소스를 빌리거나 구매해야 할 수도 있다. 아마존[04] 등에서 이러한 서비스를 하고 있다.

1.5 R 병렬 패키지 훑어보기

시간이 부족한 독자를 위하여, 이 책에서 다룰 패키지에 대하여 간략하게 장단점을 소개하였다.

1.5.1 snow

전통적인 클러스터에서 잘 동작하며, 특히 MPI가 사용 가능할 경우 좋은 결과를 낸다. MPI, PVM, nws, socket을 데이터 전송 방식으로 지원하며, 리눅스나 Mac OS, 윈도우 등 다양한 운영체제에서 동작한다.

---

03 R을 병렬로 활용하는 전략이 어떤 식으로 발전해 나가는지를 예측하는 것은 굉장히 어려운 문제다.

04 http://aws.amazon.com

- 목적 : 멀티 스레드, 메모리 제약 해결
- 장점 : 가장 많이 사용되고 있으며 유지보수가 잘되고 있다. MPI의 복잡도와 관계없이 효율적으로 동작한다.
- 단점 : 사용을 위한 설정이 어렵다.

1.5.2 multicore

성능이 좋은 CPU가 필요하지만, 하둡 설치가 어려울 때 사용할 수 있으며 R 화면에서 바로 R 코드를 병렬로 실행할 수 있다.

- 목적 : 멀티 스레드
- 장점 : 간단하지만 효율적이다. 설치가 간단하며, 별도의 설정을 하지 않아도 된다.
- 단점 : 컴퓨터 한 대에서만 사용할 수 있으며, 윈도우를 지원하지 않는다. 난수를 생성하는 문제에 적용할 수 없다.

1.5.3 parallel

snow와 multicore를 합친 패키지로, R 2.14.0부터 내장되어있다.

- 목적 : 멀티 스레드, 메모리 문제 해결
- 장점 : 별도의 설치 작업을 하지 않아도 되며 난수를 생성하는 문제에 매우 적합하다.
- 단점 : 윈도우에서는 한 대의 컴퓨터에서만 사용할 수 있다. 만일 여러 대의 컴퓨터를 연계하여 리눅스 환경에서 사용할 경우 설정이 어려워진다.

1.5.4 R+Hadoop

하둡 클러스터에서 R 코드를 실행시킬 수 있다.

- 목적 : 멀티 스레드, 메모리 문제 해결
- 장점 : 하둡 대부분의 기능을 활용할 수 있다.

- 단점 : 하둡 클러스터가 필요하며 코드를 한 개의 논리적 과정으로 분해하는 작업이 필요하다.

1.5.5 RHIPE

R 화면에서 바로 하둡을 사용할 수 있다.

- 목적 : 멀티 스레드, 메모리 문제 해결
- 장점 : R+Hadoop에 비해 R 환경과 비슷하게 작업할 수 있으며, 맵리듀스를 원래의 R 코드를 그대로 이용하여 쓸 수 있다.
- 단점 : 하둡 클러스터가 필요하며 추가적인 설정이 필요하다.

1.5.6 Segue

R 연산을 하둡 클러스터에서 원격으로 수행할 수 있다.

- 목적 : 멀티 스레드, 메모리 문제 해결
- 장점 : EMR[Elastic MapReduce]를 활용할 수 있다.
- 단점 : EMR만을 사용하여야 하며, 자체적으로 설치되어있는 하둡을 활용할 수 없다.

1.6 정리

지금부터 R 병렬 프로그래밍을 알아볼 것이다. 먼저, 현존하는 패키지 중 가장 유명한 snow 패키지에 관하여 알아보자.

2 | snow

snowSimple Network of Workstation는 현재까지 병렬처리를 위해 개발된 R 패키지 중 가장 잘 알려져있는 패키지다. 루크 티러니Luke Tierney, A.J. 로시니Rossini, 나 니Na Li, H. 세비시코바Sevcikova가 개발했으며, 이들 중 루크 티러니가 패키지를 관리하고 있다. 이 패키지는 2003년 CRAN에 등록되었으며, 이후 현재까지 꾸준히 발전해왔다.

2.1 snow 살펴보기

R 스크립트script를 좀 더 빠르게 실행시키려면 리눅스 클러스터를 사용하여야 한다. 예를 들어, 몬테 카를로 시뮬레이션Monte Carlo Simulation처럼 많은 시간이 요구되는 알고리즘을 개인용 컴퓨터에서 돌릴 경우, 수 시간에서 길게는 수일 동안 작업이 끝나기를 기다려야만 한다.

snow 패키지를 이용하여 R 코드를 회사나 대학의 리눅스 클러스터에서 실행시킬 수 있다. snow 패키지는 기존에 많이 사용하던 클러스터 환경에 최적화되어 있으며, MPI[01]를 이용하여 InfiniBand[02]를 비롯한 고속 통신 네트워크에 잘 활용할 수 있다.

2.2 snow는 어떻게 동작하는가?

snow는 R 명령어를 병렬로 처리하기 위한 함수를 제공한다. 이 함수들은 대부분 R의 내장함수인 lapply() 함수의 변형이기 때문에 사용자가 쉽게 활용할 수 있다. snow는 병렬 처리를 위해 마스터Matser/워커Worker 구조를 사용하는데, 여기서

01 역자 주_ MPI(Message Passing Interface)는 '메시지 전달 인터페이스'로, 병렬 프로그래밍을 하기 위해 사용하는 함수의 집합이다.

02 역자 주_ 프로세서와 입출력 장치 간 데이터 흐름에 대한 구조 및 규격이다.

마스터는 워커에 있는 작업을 분산시키는 역할을 하고, 워커는 실제 작업을 수행한 후 결과를 마스터에 보낸다.

snow 패키지의 중요한 특징 중 하나는 마스터와 워커 간의 소통을 위한 다양한 메시지 전달방법을 지원한다는 것이다. 이는 snow 패키지가 다양한 고성능 메시지 전달 방식의 장점을 유연하게 활용할 수 있도록 해준다. snow는 socket, MPI, PVM[Parallel Virtual Machine, 병렬 가상 머신], NetWorkSpace 방식으로 사용될 수 있다. 이들 중 socket 방식은 다른 추가적인 패키지 없이 바로 사용할 수 있어 다양한 상황에 적용할 수 있으며, MPI, PVM, NetWorkSpace 방식은 각각 Rmpi, rpvm, nws 패키지를 통해 사용할 수 있다. MPI 방식은 리눅스 클러스터에서 많이 사용하고 있으며, socket 방식은 윈도우 컴퓨터[03]와 같은 멀티코어 컴퓨터[multicore computer]에서 사용한다.[04]

snow는 주로 전통적인 방식의 클러스터에서 사용하도록 만들어졌지만 MPI 방식이 가능할 경우에도 유용하다. 이는 몬테 카를로 시뮬레이션, 부트스트랩핑, 교차검정, 앙상블 기계 학습 알고리즘, K-means 군집 분석 등에 적합하다.

rsprng이나 rlecuyer 패키지를 이용하여 병렬로 난수를 생성하는 문제에도 적용할 수 있다. 이는 시뮬레이션이나 자동 처리, 기계 학습 등을 위해 난수 생성이 필요한 경우에 사용할 수 있다.

snow는 데이터 파일을 직접 워커에 분할하는 것과 같이 큰 데이터를 다루기 위한 방법은 제공하지 않는다. snow 기능을 사용하려면, 인수는 사용 가능한 메모리 용량을 초과하지 않아야 하며 연산 결과 또한 list의 형태로 반환되기 전에는 마스터에 의해 메모리에 저장된다. snow는 빅데이터를 다루기 위해 다른 고성능 파일 분산

03 역자 주_ Windows 운영체제가 설치된 컴퓨터를 지칭한다.

04 보통 사용하는 멀티코어 컴퓨터에는 multicore 패키지를 사용하는 것이 좋지만, 이 패키지는 윈도우에서 제대로 동작하지 않는다. multicore 패키지에 대해서는 3장에서 자세히 설명하였다.

시스템과 함께 사용할 수 있지만, 이를 연계하는 것은 사용자의 몫이다.

2.3 snow 설치

snow는 CRAN에서 구할 수 있으며, 다른 CRAN 패키지와 동일한 방식으로 설치할 수 있다. snow는 다른 언어를 사용하지 않고 R 언어만으로 만들어져 호환이 잘 되며 설치하는 데 문제가 거의 없다. 또한 윈도우와 Mac OS 환경에서 구동되는 바이너리 패키지binary package도 제공한다.

CRAN 패키지를 설치하는 방법은 여러 가지인데, 일반적으로는 install.packages() 함수를 이용한다.

```
Install.packages( " snow " )
```

사용할 CRAN의 미러 사이트[05]를 선택하면 패키지가 다운로드되어 설치된다. 만약 R의 구버전을 사용한다면, snow를 사용할 수 없다는 메시지를 받을 수 있다. snow 0.3-5 버전부터는 R 2.12.1 버전 이상에서만 구동되기 때문에, R의 구버전에서 snow를 사용하려면 CRAN 패키지 아카이브에서 0.3-3 버전의 snow를 다운받아야 한다. 웹 브라우저에서 'CRAN snow'를 검색하면 CRAN의 snow 다운로드 페이지를 찾을 수 있다. 여기서 'snow archive'를 클릭하고 snow_0.3-3.tar.gz를 다운받으면 된다. 또는 직접 다운받을 수 있는 다음 링크를 입력하여 다운로드 받을 수 있다.

- http://cran.r-project.org/src/contrib/Archive/snow/snow_0.3-10.tar.gz

패키지를 다운받으면 커맨드 창에서 다음 명령어를 입력하여 패키지를 설치한다.

05 역자 주_미러 사이트는 네트워크 트래픽을 줄이기 위하여 다른 컴퓨터 서버를 복사해 놓은 웹 사이트 또는 컴퓨터 파일서버를 말한다.

```
% R CMD INSTALL snow_0.3-10.tar.gz
```

만약 R의 기본 설치 경로에 설치할 수 있는 권한이 없는 경우 '-l' 옵션을 이용하여 다른 경로에 패키지를 설치할 수 있다. 이에 대한 도움말은 '--help' 옵션을 통해 확인할 수 있고, 'R 프로젝트 웹 사이트'[06]에서 제공하는 'R installation and Administration' 매뉴얼의 'Installing packages' 섹션을 보면 R 패키지의 설치에 관해 좀 더 자세한 설명을 볼 수 있다.

NOTE_ 개발자들은 보통 최신 버전의 R을 사용하게 된다. 패키지 대부분은 최신 버전의 R을 이용하여 만든 후 CRAN에 등록하기 때문에, 최신 버전의 R을 사용하면 CRAN의 패키지들을 좀 더 쉽게 설치할 수 있다. CRAN은 이전 버전의 R을 이용하여 만든 패키지도 보관하고 있지만, 해당 패키지에 최근에 업데이트된 내용이 반영되어 있지 않다. 그래서 구 버전의 R에서는 snow이 구동되지 않을 수도 있다. R을 실전에서 사용할 때에는 버전 관리에 유의하여야 한다.

snow를 MPI와 함께 사용하려면, Rmpi 패키지를 설치하여야 한다. 아쉽게도 Rmpi는 설치하려는 운영체제에 MPI가 제대로 설치되어 있어야 사용할 수 있기 때문에 자주 문제를 일으킨다. 이에 관해서는 "2.4.11 Rmpi 설치하기"에서 좀 더 자세히 설명하였다.

다행히도 socket transport는 다른 패키지를 설치하지 않아도 바로 사용할 수 있다. 이러한 이유에서, snow를 처음 사용할 경우에는 socket 전송을 사용하는 것이 좋다. snow를 설치한 후에는 다음 명령어를 이용하여 설치가 제대로 되었는지 확인해야 한다.

```
library(snow)
```

06 http://www.r-project.org/

별다른 에러가 발생하지 않으면 snow를 사용할 준비가 된 것이다.

2.4 snow 사용하기

2.4.1 makeCluster를 이용한 클러스터 만들기

snow를 이용하여 R 함수를 병렬처리할 때, 가장 먼저 해야 할 일은 클러스터Cluster 객체를 만드는 것이다. 클러스터 객체는 클러스터 워커cluster worker와 상호작용하는 데 필요하며, 이는 대부분의 snow 함수에서 첫 번째 인수다. 클러스터 객체는 사용하고자 하는 메시지 전달 방식에 따라 다양하게 생성될 수 있다.

기본적인 클러스터 생성 함수 중 하나는 makeCluster()로, 대부분의 클러스터 객체를 만들 수 있다. 예를 들어, socket 전송 방식을 사용하는 워커가 4개인 클러스터 객체를 만들어보자.

```
cl <- makeCluster(4, type = 'SOCK')
```

첫 번째 인수는 클러스터의 특성을 나타내며, 두 번째 인수는 클러스터의 종류를 의미한다. 첫 번째 인수의 의미는 클러스터가 어떤 종류에 속하는지에 따라 달라지지만, 주로 워커의 수를 의미한다.

socket 클러스터 방식은 문자열을 이용하여 워커 머신worker machine을 좀 더 정확히 기술할 수 있다. 다음 명령어를 통해 컴퓨터 4대를 워커로 사용할 수 있다.

```
Spec <- c('n1', 'n2', 'n3', 'n4')
Cl <- makeCluster(spec, type = 'SOCK')
```

워커는 ssh 명령으로 각각 실행해야 하며, 워커가 'localhost'일 경우에는

makeCluster()가 자체적으로 실행하게 된다. R로 워커를 실행하려면, 각 워커에 암호 없이 접속할 수 있도록 SSH 공개키 인증 접속 기능이 있어야 한다. 자세한 방법은 『SSH, The Secure Shell : The Definitive Guide』(O'Relly, 2005년 5월) 이나 웹 사이트 등을 살펴보기 바란다.

makeCluster()의 좀 더 자세한 설정들은 '2.4.10 snow 설정'에서 다룰 것이다. 타입 인수에는 'SOCK', 'MPI', 'PVM', 'NWS' 등의 값` 등이 들어갈 수 있다. 워커가 4개인 MPI 클러스터를 만들려면 다음 명령어를 실행해보자.

```
cl <- makeCluster(4, type="MPI")
```

별도의 설정이 없다면, 이 명령어로 4개의 MPI 워커를 실행시킬 수 있다.

클러스터는 makeSOCKcluster(), makeMPIcluster(), makePVMcluster(), makeNWScluster()의 함수를 사용해서도 만들 수 있다. 사실, makeCluster() 함수는 이들의 래퍼wrapper 함수다. 만들어진 클러스터를 종료하려면 stopCluster() 함수를 사용하면 된다. 예를 들어, 위에서 만든 cl 클러스터를 종료하려면 'stopCluster(cl)' 명령어를 입력하면 된다.

어떤 클러스터는 R을 종료하면 함께 종료되지만, 그렇지 않은 경우도 있으며 클러스터가 종료되지 않으면 워커도 계속 동작하여 컴퓨팅 파워를 낭비하게 된다. 따라서 snow를 사용할때에는 언제나 stopCluster() 문을 이용하는 것이 좋다.

> **NOTE_** 클러스터를 만들 때 생길 수 있는 몇 가지 문제에 대해서는 '2.4.14 snow 프로그램 문제해결'에 자세히 기술하였다.

2.4.2 병렬 K-Means

이제 snow를 이용하여 실제로 병렬 컴퓨팅을 해 볼 것이다(여기서는 병렬로 K-Means 분류를 수행해 볼 것이다). K-Means는 데이터셋을 K개의 군집[07]으로 분류하는 알고리즘으로, 각 군집의 중심좌표를 추정한 후에 '반복적 알고리즘'을 사용하여 중심좌표와 군집의 구성을 최적화한다.

R의 내장 패키지 중 하나인 'stat 패키지'에는 K-Means 클러스터링clustering을 위한 함수인 kmeans()가 포함되어 있다. kmeans() 함수를 사용하려면 군집의 갯수를 설정해주어야 한다. kmeans()는 분류하고자 하는 데이터셋에서 임의로 선택된 값을 군집의 중심좌표로 사용하며, 군집 내부의 제곱합을 계산한 후 군집 내부의 제곱합을 최소화시키는 지점을 찾는다.

kmeans( ) 함수를 이용하여 R의 MASS 라이브러리library에 포함되어있는 'Boston' 데이터를 100개의 임의의 중심좌표를 이용하여 4개의 군집으로 나누어 보겠다.

```
library(MASS)
result <- kmeans(Boston, 4, nstart = 100)
```

지금부터 kmeans() 함수 코드를 수정하지 않고 바로 병렬화하여 사용할 것이다. kmeans() 함수를 각 워커에 할당한 후 원래보다 적은 nstart 값을 사용하여, K-Means 클러스터링을 수행한다. 그리고 전체 결과 중에서 가장 적은 군집내 제곱합 값을 선택한다.

그전에 lapply() 함수를 이용하여 이 방법이 제대로 동작하는지 확인한다. 제대로 동작하는지 확인한 후에, snow를 이용하여 '병렬 K-means 알고리즘'을 구현하

07 여기에서 군집(cluster)은 클러스터 객체나 클러스터 워커와 다른 개념이다.

면 훨씬 쉽게 구현할 수 있다. 다음은 snow를 이용하여 '병렬 K-means 알고리즘'을 구현한 것이다.

```
library(MASS)
results <- lapply(rep(25, 4), function(nstart) kmeans(Boston, 4,
  nstart=nstart))
i <- sapply(results, function(result) result$tot.withinss)
result <- results[[which.min(i)]]
```

위 명령어에서는 25가 4번 반복된 벡터를 nstart 값으로 이용하여 kmeans() 함수를 4번 실행시켰으며, 이는 nstart를 100으로 하여 kmeans() 함수를 한 번 실행시킨 것과 같다. 이 벡터는 워커의 수와 같아야 나중에 코드를 병렬화할 수 있다.

이제 위 코드와 비슷한 방법으로 알고리즘을 병렬화해보자. snow에 포함되어 있는 clusterApply(), clusterApplyLB(), parLapply() 등의 함수를 사용하면 되는데, 여기에서는 clusterApply() 함수를 사용할 것이다. 이 함수는 lapply()와 거의 동일하게 사용할 수 있으며, 첫 인수로 snow cluster 객체를 사용한다는 점만 다르다. 또한, 각 워커에서 MASS 라이브러리를 불러들여 'Boston' 데이터셋을 사용해야 한다.

snow를 불러들였고 'cl'이라는 이름으로 클러스터 객체를 만들었다고 가정해보자. 그러면 다음 코드를 이용하여 kmeans()를 병렬화할 수 있다.

```
ignore <- clusterEvalQ(cl, {library(MASS); NULL})
results <- clusterApply(cl, rep(25, 4), function(nstart) kmeans(Boston,
  4, nstart=nstart))
i <- sapply(results, function(result) result$tot.withinss)
result <- results[[which.min(i)]]
```

clusterEvalQ() 함수는 클러스터 객체와 실행할 명령어를 인수로 받아 각 워커에서 명령어를 실행시킨다. 실행 결과를 리스트 형태로 반환하는데, 지금은 그저 MASS 라이브러리를 불러들이는 데 사용하였기 때문에 무시하였다. 여기에서는 반환값을 사용하지 않기 때문에 문제가 없지만, 만에 하나 문제가 생길 수 있기 때문에 안전을 위해 NULL 값을 반환하였다.

위 예제에서 알 수 있듯, snow를 이용한 코드는 lapply()를 이용한 코드와 크게 다르지 않다. 작업 대부분은 lapply()를 이용하기 위한 코드를 짤 때 이루어졌다. lapply()를 이용하여 짠 코드를 snow를 이용한 코드로 변환할 때 가장 중요한 것은, 데이터를 효율적이고 적절하게 다루어야 한다는 것이다. 여기에서는 데이터셋이 MASS 패키지에 포함되어 있었기 때문에 패키지를 각 워커에 불러오는 것으로 충분했다.

> **NOTE_** kmeans() 함수는 sample.int() 함수를 이용하여 군집의 중심좌표를 선택하는데, 관련 작업은 난수 생성함수random number generator를 사용한다. 좀 더 좋은 결과를 얻기 위하여 각 워커에 다른 난수 생성함수를 사용하게 할 수도 있다. 위의 예제에서는 워커가 동일한 난수 생성함수를 사용했어도 시드Seed가 달랐기에 제대로 동작하였지만[08], 병렬 난수 생성함수를 사용해 보는 것도 좋은 연습이 될 것이다. "2.4.9 난수 생성"에서 좀 더 자세히 설명할 것이다.

2.4.3 워커 초기화하기

이전 장에서 clusterEvalQ() 함수를 이용하여 워커에 패키지를 불러와 클러스터를 초기화하였다. clusterEvalQ() 함수는 매우 편리하지만 사용에 제한이 있다. 예를 들어, 이 함수는 클러스터 워커cluster worker에 단순한 명령어를 전달할 수는 있지만 명령어에 매개변수가 필요할 경우 사용할 수 없다. 또한 함수를 실행시킬 수는 있지만 함수 자체를 각 워커에 전달하지는 않는다.[08]

08 snow가 정확히 어떻게 워커들에게 함수를 전달하는지는 다소 복잡하며, 실행 환경과 상황의 영향을 받는다. "2.4.8 함수와 작업환경"을 참고하라.

클러스터 워커를 초기화하는 데 가장 즐겨 사용하는 함수는 clusterCall()이다. 인수는 snow 클러스터 객체와 워커 함수worker function, 그리고 함수에 필요한 값들이다. 이 함수는 함수를 각 클러스터 워커에서 불러와 실행시키며, 결과값을 list로 반환한다. clusterApply() 함수와 거의 동일하지만 apply() 함수에서 사용되는 x 인수가 없다는 점만 다르다.

clusterCall()은 clusterEvalQ() 함수가 할 수 있는 일은 물론 할 수 없는 일도 수행할 수 있다. 예를 들어, 위의 예제에서 MASS 패키지를 각 워커에 불러오는 일을 clusterCall()을 이용하여 수행해보자.

```
clusterCall(cl, function() { library(MASS); NULL })
```

위 명령어는 MASS 패키지를 불러온 후 NULL을 반환하는 간단한 함수를 작성하여 실행시킨다. NULL 값을 반환함으로써 마스터에 쓸데없는 데이터를 전송하는 것을 방지할 수 있다. 다음 명령어를 사용하면 문자열 벡터에 포함된 여러 개의 패키지들을 불러올 수 있다.

```
worker.init <- function(packages) {
  for (p in packages) {
    library(p, character.only=TRUE)
  }
NULL
}
clusterCall(cl, worker.init, c('MASS', 'boot'))
```

character.only 값을 TRUE로 설정하여 문자열만 입력받을 수 있도록 하였다. 만약 이 설정을 해주지 않으면 p라는 이름의 패키지를 불러오려고 할 것이다.

clusterCall()만큼 자주 쓰이지는 않지만, clusterApply() 함수도 워커를 초기화하는 데 사용할 수 있다. 예를 들어, 다음 명령어를 통해 worker ID를 가지는 광역변수를 각 클러스터 워커에 전달할 수 있다.

```
clusterApply(cl, seq(along=cl), function(id) WORKER.ID <<- id)
```

2.4.4 clusterApplyLB 함수 이용하기

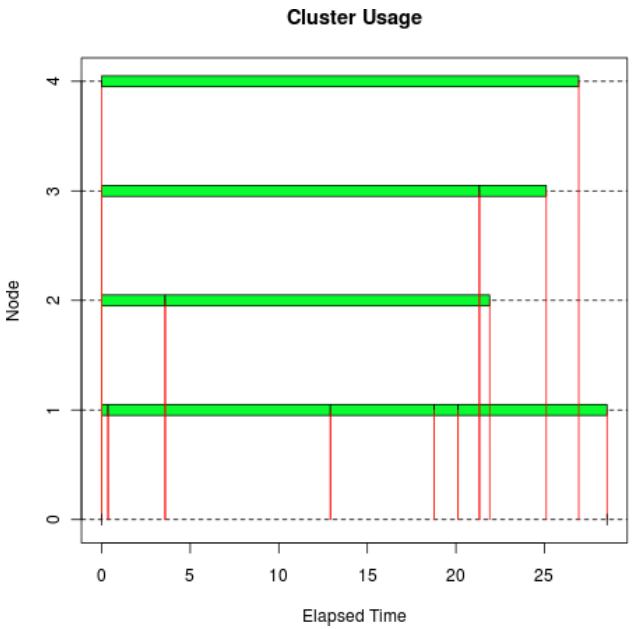
K-Means의 예제를 통해 clusterApply() 함수 사용법을 알아보았다. 다음으로 clusterApplyLB() 함수에 관하여 알아보자. 이 함수는 clusterApply() 함수와 거의 동일하지만, 워커에 작업을 배분하는 방식이 '라운드 로빈round-robin 방식'이 아닌 워커가 이전의 작업을 끝냈때만 새로운 작업을 부여한다.

clusterApply() 함수에서 사용되는 라운드 로빈 방식은, 카드 게임에서 카드를 나누듯이 처음 일을 시작할 때 각 워커에 한꺼번에 작업을 배분한다. 말하자면 clusterApply() 함수는 할당량을 각 워커에 배분하는 방식이라면, clusterApplyLB() 함수는 각 워커가 일할 수 있는 만큼만 작업을 분해한다고 할 수 있다. 하나의 작업에 소요되는 시간이 제각기 다르거나 워커의 작업 속도가 각각 다르다면 clusterApplyLB() 방식이 좀 더 효율적일 수 있다.

clusterApplyLB() 함수를 이용하여 Sys.sleep() 함수를 워커에서 실행시켜보겠다. clusterApplyLB() 함수가 실제로 효율적인지 확인해보기 위해서 snow.time() 함수를 이용하여 전체 실행시간을 확인해보겠다. snow.time() 함수를 이용하면 워크에서 실행된 작업을 그래프로 그릴 수도 있다.

```
set.seed(7777442)
sleeptime <- abs(rnorm(10, 10, 10))
```

```
tm <- snow.time(clusterApplyLB(cl, sleeptime, Sys.sleep))
plot(tm)
```

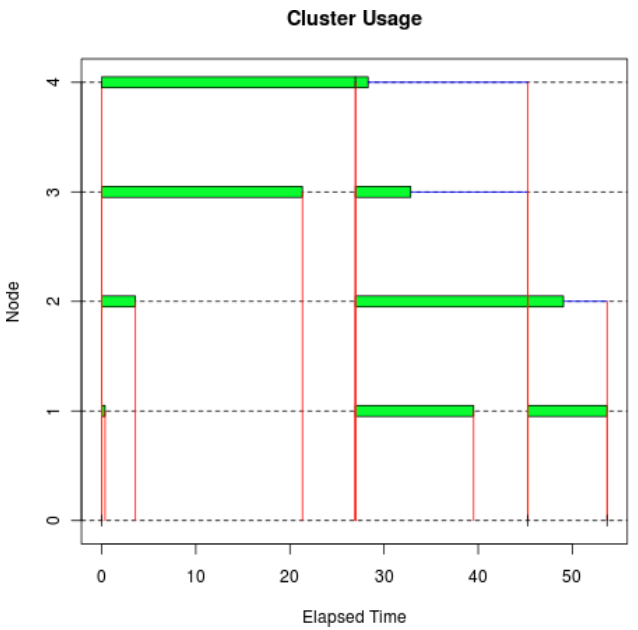


이상적으로는 모든 워커가 일을 계속하여, 그래프에서 보여지는 4개의 노드에 대해 균일한 막대그래프가 그려져야 할 것이다. clusterApplyLB() 함수는 워커 대부분이 마지막 순간까지 일을 하고 있지만, 마지막 순간에 약간의 낭비가 있음을 알 수 있다.

이번에는 같은 작업을 clusterApply() 함수를 사용하여 수행해보자.

```
set.seed(7777442)
sleeptime <- abs(rnorm(10, 10, 10))
```

```
tm <- snow.time(clusterApply(cl, sleeptime, Sys.sleep))
plot(tm)
```



위 그래프를 통해 clusterApply() 함수가 clusterApplyLB() 함수에 비해 비효율적임을 알 수 있다. clusterApplyLB() 함수는 전체 작업을 28.5초만에 끝냈지만, clusterApply() 함수는 53.7초만에 작업을 마쳤다. 그래프는 라운드 로빈 스케쥴링 방식이 확실히 비효율적임을 보여준다.

그러나 clusterApply() 함수도 나름의 장점이 있다. K-Means 병렬화 예제에서 보았듯 각 워커에 비슷한 시간이 소모되는 작업이 배분될 경우에는 충분히 효율적으로 동작한다. 또한 바로 다음에 논의될 parLapply() 함수와 함께 사용할 수 있는 장점도 있다.

2.4.5 parLapply 함수를 이용한 작업량 분배

지금까지 논의한 clsuterApply(), clusterApplyLB() 함수와 마찬가지로 parLapply() 함수도 lapply() 계열의 함수로 비슷한 인수를 사용한다. 그러나 다른 함수보다 일반적으로 많이 쓰이는 중요한 특징이 한 가지 있다. parApply() 함수는 snow의 가장 상위 함수로, clusterApply() 함수의 기능을 포함하면서도 훨씬 간단해 보이는 래퍼 함수다.

```
> parLapply
function (cl, x, fun, ...)
docall(c, clusterApply(cl, splitList(x, length(cl)), lapply, fun, ...))
<environment: namespace:snow>
```

기본적으로 parLapply() 함수는 입력받은 값 x를 부분벡터로 나누고, 각 부분벡터를 lapply() 함수를 이용하여 클러스터 워커로 처리한다. 이는 마치 클러스터가 워커로 구성되어 작업을 할당하듯이, 미리 전체 작업량을 작은 크기로 나누는 효과가 있다.

기능적으로는 clusterApply() 함수와 같지만, 마스터와 워커 간에 데이터 전송이 줄어들어 실제로는 훨씬 효과적이다. 만약 작업량 x의 길이가 워커의 수와 같다면 이러한 장점은 사라진다. 하지만 보통 lapply() 함수를 사용하는 경우에 x의 크기는 매우 커지기 때문에 parLapply() 문은 clusterApply()에 비해 성능은 좋다.

한 가지 더 생각해보아야 할 것은 parLapply() 함수와 clusterApply() 함수는 인수 처리 방식이 다르다는 것이다. clusterApply() 함수는 하위 레벨low-level 함수기 때문에 인수를 실제로 클러스터 워커에서 실행시킨다. 반면, parLapply() 함수는 인수를 lapply() 함수를 거쳐 클러스터 워커에서 처리하게 한다.

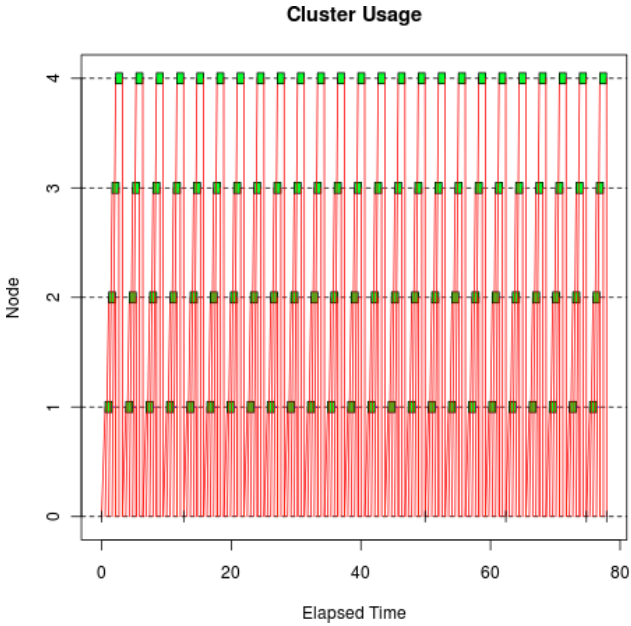
clusterApply() 함수는 실제 연산을 좀 더 직접적으로 다룰 수 있지만, parLapply() 함수를 사용하면 클러스터 워커에 작업을 좀더 편리하게 나눌 수 있다.

다시 한번 강조하자면, parLapply() 함수는 작업의 갯수가 워커의 수보다 많을 때 clusterApply() 함수에 비해 훨씬 효율적이다. 추가적인 인수가 있을 경우에 parLapply() 함수는 워커에 최초 한 번만 전달한다. 다음 명령어를 이용하여 행렬을 인수로 하는 sleep 함수를 만들어보자.

```
bigsleep <- function(sleeptime, mat)
Sys.sleep(sleeptime)
bigmatrix <- matrix(0, 2000, 2000)
sleeptime <- rep(1, 100)
```

위 명령어에서는 sleep 시간을 여러 번으로 작게 나누었으며, 각 시간은 동일하다. 이는 clusterApply()와 parLapply()의 차이를 좀 더 잘 보여줄 것이다.

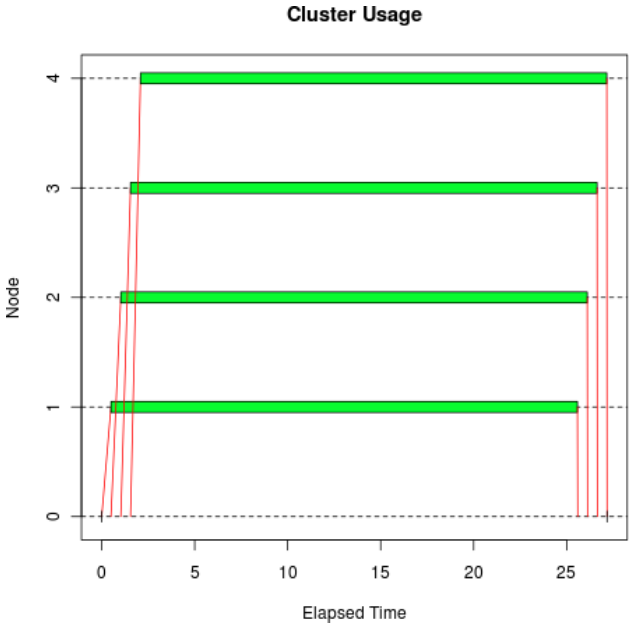
```
tm <- snow.time(clusterApply(cl, sleeptime, bigsleep, bigmatrix))
plot(tm)
```



clusterApply() 문을 이용하여 위에서 정의한 bigsleep 함수를 실행시킨 결과는 그다지 효율적으로 보이지 않는다. 그래프에서 알 수 있듯, 마스터와 워커 간에 메시지를 전송하는 시간이 매우 많아 실제 계산에 소모되는 시간보다 훨씬 더 긴 시간이 소요되었다. 이상적으로는 25초에 모든 연산이 끝나야겠지만, I/O에 소모된 시간 때문에 실제로는 77.9초나 걸렸다.

이제 parLapply() 함수를 사용하여 동일한 작업을 수행해보자.

```
tm <- snow.time(parLapply(cl, sleeptime, bigsleep, bigmatrix))
plot(tm)
```



그래프와 소요시간 모두 극적으로 차이가 났다. 전체 계산을 마치는데 27.2초가 걸렸으며, 이는 clusterApply() 함수를 사용했을 때보다 50.7초나 빠르다.

이는 clusterApply() 함수가 나쁘다는 의미는 아니다. clusterApply() 함수는 매 작업을 수행할 때마다 쓸데없이 행렬 전체를 전송한다. 이는 여러가지 방법을 통해 보완될 수 있지만, 이런 경우에는 parLapply()가 좀 더 결과가 좋다는 뜻이다. 한편으로는, 하나의 큰 작업을 처리해야 할 경우에는 parLapply() 함수가 제대로 작업을 나눌 수 없게 된다. 어쨌거나 결론적으로는 snow의 병렬 함수들 중에서는 parLapply() 함수가 일반적으로 가장 효율적이다. 만약 어떤 함수를 사용해야 할지 모르겠을 때에는 parLapply() 함수를 사용하는 것이 좋다.