分布式统计计算

1. 程序性能和并行计算

李舰

华东师范大学

2018-11-19

目 录

目 录

什么是算法

- 算法（Algorithm）
  - 一种有限、确定、有效的并适合用计算机程序来实现的解决问题的方法。
- 自然语言描述算法
  - 2300 年前的欧几里德算法被认为是世界上最早的算法，用来找到两个数的最大公约数：
  - 计算两个非负整数 p 和 q 的最大公约数：若 q 是 0，则最大公约数为 p。否则将 p 除以 q 得到余数 r，p 和 q 的公约数即为 q 和 r 的最大公约数。
- 伪代码描述算法
  - 以编程语言的书写形式指明算法职能，不用拘泥于具体实现：

```
qs <- function(x) {
  if (length(x) <= 1) return()
  find small group
  find large group
  move small group to left part of x
  ...
}
```

算法和程序

- 算法的特点
  - 输入：有 0 个或者多个外部量作为算法的输入。
  - 输出：算法产生至少一个量作为输出。
  - 确定性：组成算法的每条指令是清晰的、无歧义的。
  - 有限性：算法中每条指令的执行次数有限，执行每条指令的时间也有限。
- 程序（Program）
  - 程序是算法用某种程序设计语言的具体实现。
  - 程序可以不满足算法的有限性，例如操作系统，它是在无限循环中执行的程序，因此不是算法。
- 算法的设计
  - 选择该问题的数据模型。
  - 明确初始状态和结果状态，以及两个状态之间的关系。
  - 探索初始状态到结果状态之间的运算步骤。

目 录

算法复杂度的表示

- 时间复杂度和空间复杂度
  - 算法运行时需要的时间资源称为时间复杂度。
  - 算法运行时需要的空间资源称为空间复杂度。
  - 空间复杂度相对简单，通常用时间复杂度来描述算法复杂度。
- 时间复杂度的大 O 表示法
  - 如果算法 A 在规模为 n（数据量）的问题上需要的时间与 $f(n)$ 成正比，则称算法 A 为 $f(n)$ 阶，记为 $O(f(n))$。函数 $f(n)$ 称为算法 A 的增率函数。
  - 常见的算法复杂度级别：
    - $O(1)$ 称为常数级别。
    - $O(log(n))$ 称为对数级别。
    - $O(n)$ 称为线性级别。
    - $O(nlog(n))$ 称为线性对数级别。
    - $O(n^2)$ 称为平方级别。
    - $O(2^n)$ 称为指数级别。
  - 如果问题的复杂度不大于 $O(n^k)$（k 为常数值），则称为多项式复杂度。

P 和 NP 问题

- P 和 NP
  - 如果一个问题的算法复杂度可以表示为多项式函数，那么这个问题就叫做 P（Polynomial）问题，又叫多项式问题，这类问题被认为是计算机可以有效解决的。
  - 如果算法复杂度不可以用多项式表示，也就是说复杂度高到没有办法用多项式表示或者找不到有效的公式描述复杂度，那么就叫 Non-Polynomial 的问题，即非多项式问题。
  - 有些问题，我们无法确切地知道它的多项式复杂度的算法到底是否存在，所以叫做非确定的多项式（Nondeterministic Polynomial，简称 NP）问题。
- NPC 问题
  - 在 NP 问题中，有一类最难的问题叫做 NPC（NP-Complete）问题。可以证明所有 NP 问题都可以在多项式时间内归约成 NPC 问题，而任意一个 NPC 问题只要找到多项式时间的算法，就可以解决其他问题，称为 $P = NP$。
  - 目前的状况是找不到任何一个 NPC 问题的解，但也无法证明 $P \neq NP$。

目 录

目 录

R 的性能问题

- 固有的缺陷
  - 无法多线程。
- 权衡的牺牲
  - 解释型语言与交互式环境，可以使用 compiler 包来弥补。
  - 免费工具与代数运算库，可以换用商业 BLAS/LAPACK。
  - 统计建模与计算机编程，可以使用 C 或者 Fortran 开发对性能要求高的函数（注意，要多使用内置函数）。
- 并非特有的缺点
  - “基于内存的计算是个缺陷”。除了 SAS 等少数分析工具可以隐式执行内存外分析之外，数据量大都会撑爆内存。只是其他语言的用户不大容易遇到大数据分析的问题。解决办法都是内存外运算或并行，可以参考 R 中的 bigmemory 和 parallel 等包。
  - “难以处理大数据”。数据大到一定程度之后就不是编程语言的问题了，业界通常是借助于数据库或者并行系统来解决，可以参考 R 中的 ORE 或者 Rmpi、RHadoop 等包。

提升性能的方法

- 系统升级
  - 升级硬件。
  - 使用 64 位操作系统。
  - 利用 GPU。
  - 租用云计算。
- 开发层面的优化
  - 算法优化。
  - 调用 C 或者 Fortran。
- 应用层面的优化
  - 充分利用 R 的内部机制。
  - 突破内存的限制。
  - 增强 R 的矩阵运算：加速 BLAS。
  - 并行计算。

性能相关的计算机常识

- 硬件组成
  - CPU：中央处理器，一块超大规模的集成电路，是一台计算机的运算核心和控制核心，通常包含运算器、高速缓存、总线等。
  - 内存：用来存储程序和数据，也称为内部存储器，CPU 可通过数据总线对内存寻址。
  - 硬盘：一种磁介质信息技术存储载体，是计算机的主要外部存储设备。
- 进程和线程
  - 进程（Process）是操作系统进行资源分配和调度的基本单位。每个进程对应一个程序，并拥有一块独立的内存地址空间。对单核 CPU 来说，任何时刻都只有一个进程在占用资源。
  - 线程（Thread）是程序执行流的最小单元，也被称为轻量级进程，如果进程包含多个子任务，让一个线程执行一个子任务，这样一个进程就包含了多个线程，每个线程负责一个单独的子任务。

程序性能的度量

- 时间度量
  - system.time 函数。
  - 将需要执行的代码传入。
  - 返回运行的时间信息。
- 程序运行的时间
  - user 表示用户时间，是 R 自身花费的时间。
  - system 表示系统时间，是操作系统花费的时间（比如读取文件或者与网络资源的交互）。
  - elapsed 表示消耗时间，是指运行程序总共花费的时间。
- 空间度量
  - 用消耗的内存来描述。
  - 有些算法可以实现“时间换空间”或者“空间换时间”。

性能分析

- Rprof 函数
  - 记录详细的性能状况。
  - 使用 Rprof 函数并指定输出文件则开始记录。
  - 执行一段代码之后如果需要结束，则输入 Rprof() 命令
- summaryRprof 函数
  - 对 Rprof 的结果汇总。
  - 参数 chunksize 表示一次读取的行数。
  - 参数 memory 表示如何显示内存信息。
  - 参数 exclude 表示排除在统计结果之外的函数。
- profvis 函数
  - 包含在 profvis 包中。
  - 用动态可视化的方式描述程序各部分的性能。

目 录

安装 R 的开发工具

- 安装 Rtools
  - Windows 中，下载安装包：http://cran.r-project.org/bin/windows/Rtools。
  - 双击安装，使用默认配置。
  - 可以选择自动设置环境变量（不建议勾选）。
- 配置环境变量
  - 在系统环境变量中增加 R_HOME 环境变量，其值为 R 的安装目录。
  - 将 R_HOME 下的 bin 目录以及 RTools 下 gcc 的 bin 目录添加到 PATH 环境变量。
- 测试安装
  - 任务栏中输入 cmd 打开命令行窗口。
  - 输入 Rscript --version 然后回车。
  - 输入 gcc --version 然后回车。

C 语言基础操作

- 编译和运行
  - 在文本文件（后缀名通常为 .c）中编写 C 语言的源码，通过 gcc 命令进行编译。
  - 通常编译得到可执行文件，运行后可以得到结果。

```
gcc -g learn.c
a.exe
```

- 注意事项
  - 每个 C 程序都需要一个 main() 函数，它是程序执行的起点。
  - 每个变量在使用前都需要被声明，即我们要让编译器为它分配空间。
  - C 语言中数组的索引默认是从 0 开始的。
  - 符号 * 开头的变量表示一个指针变量。符号 & 表示对内存地址求值，返回该变量的内存地址。

目 录

compiler 与编译

- R 的新特性，compiler 包
  - compiler 包是 R 2.13.0 及之后版本自带的一个标准包，它可以把一段 R 代码编译成字节码，从而在执行时提升效率。

- 示例

```
library(compiler)
la1c <- cmpfun(la1)
y <- 1:1000

system.time(for (i in 1:1000) la1(y, is.null))

##    user  system elapsed
##    1.44    0.01    1.45

system.time(for (i in 1:1000) la1c(y, is.null))

##    user  system elapsed
##    0.77    0.03    0.79
```

向量运算

- 循环与向量运算
  - R 是一种方便的矩阵运算语言，向量是其最基础的数据结构。
  - 将向量作为一个整体进行计算比对每个值进行循环操作的性能要高很多。
- 向量运算的优势
  - 循环消耗的时间会线性增长，而向量会一起操作。
  - R 的向量运算会调用外部库。
  - 向量运算和矩阵运算是 R 中提高性能的有效手段。
- 示例
  - 编写函数计算一个向量每个元素的平方，然后比较性能差异。
  - 使用循环程序，对每一个数计算平方。
  - 使用向量操作，对整个向量计算平方。

目 录

什么是并行？

- 并行计算（Parallel Computing）
  - 一般是指许多指令得以同时进行的计算模式。
  - 在同时进行的前提下，可以将计算的过程分解成小部份，之后以并行方式来加以解决。
  - 通常指单机多核或多台机器的并行。
- 显式并行（Explicit Parallelism）
  - 由用户控制的并行，需要在算法上做专门的处理。
  - 在某些并行框架的基础上，可以降低并行算法的难度。
- 隐式并行（Implicit Parallelism）
  - 系统自动进行的并行处理。
  - 通常只能处理一小部分的运算，需要用户将问题界定清楚。

并行的性能问题

- 速度的障碍
  - 通信开销：通常，数据会在进程间来回传输，传输所花费的时间会带来性能上的代价。如果运算的粒度比较粗那么开销显得不是大问题，如果粒度很细，要尽量避免开销。
  - 负载均衡：如果不注意把任务分配给进程的方式，可能会给一些节点分配较多的任务，这样就影响了性能。
- 延迟和带宽
  - 通信信道的速度，不管是处理器内核与共享内存平台的内存之间，还是一堆机器组成的集群的各节点之间，是用延迟和带宽来度量的。
  - 延迟指一个信息位从端到端的传输时间，带宽是指每秒钟能够往信道里发送的信息位数目。

R 的并行方式方式

- 并行的硬件平台
  - 多核机器。
  - 集群。
- 消息传递：Rmpi
  - 最常见的显式并行。
- snow 和 parallel
  - 可以使用 MPI、NWS、PVM、Sockets 这四种传递方式来实现并行计算。
  - 在多核或者计算机集群上实现并行计算。
  - 从 R 2.14.0 开始内置 `parallel` 包，它基于 `multicore` 和 `snow` 包构建，提供了两个包的大部分功能。
- doParallel 和 foreach
  - `doMC` 注册多核。
  - `foreach` 取代循环，分配到多核运算。
  - 对于循环来说，`foreach` 的方式相当于隐式并行。

多线程的 R

- R 的并行方式
  - 多进程并行，使用 Socket 或 MPI 等软件库进行进程通信，或者使用 Hadoop/Spark 等平台。
  - C/FORTRAN 等底层代码的并行，可以达到线程级别，但涉及 R 对象时并非线程安全。
  - 基于 GPU 的并行。
- R 中多线程的探索
  - Duncan Temple Lang 和 Luke Tierney 一直在致力解决这个问题，提出了 OpenMP 的尝试解决方案。
  - 普渡大学刘传海教授重写了 R 内核，开发了 SupR。
- SupR 简介
  - SupR 是一款同时支持多线程和分布式计算的修改版 R。
  - 在可能引起线程冲突的部分加入互斥锁，保证程序正常运行。
  - 目前处于实验开发阶段，正式版将以开源形式发布，详情请参考第九届中国 R 语言会议（北京）中邱怡轩的报告“SupR：让 R 语言走向多线程并行计算”。

目 录

目 录

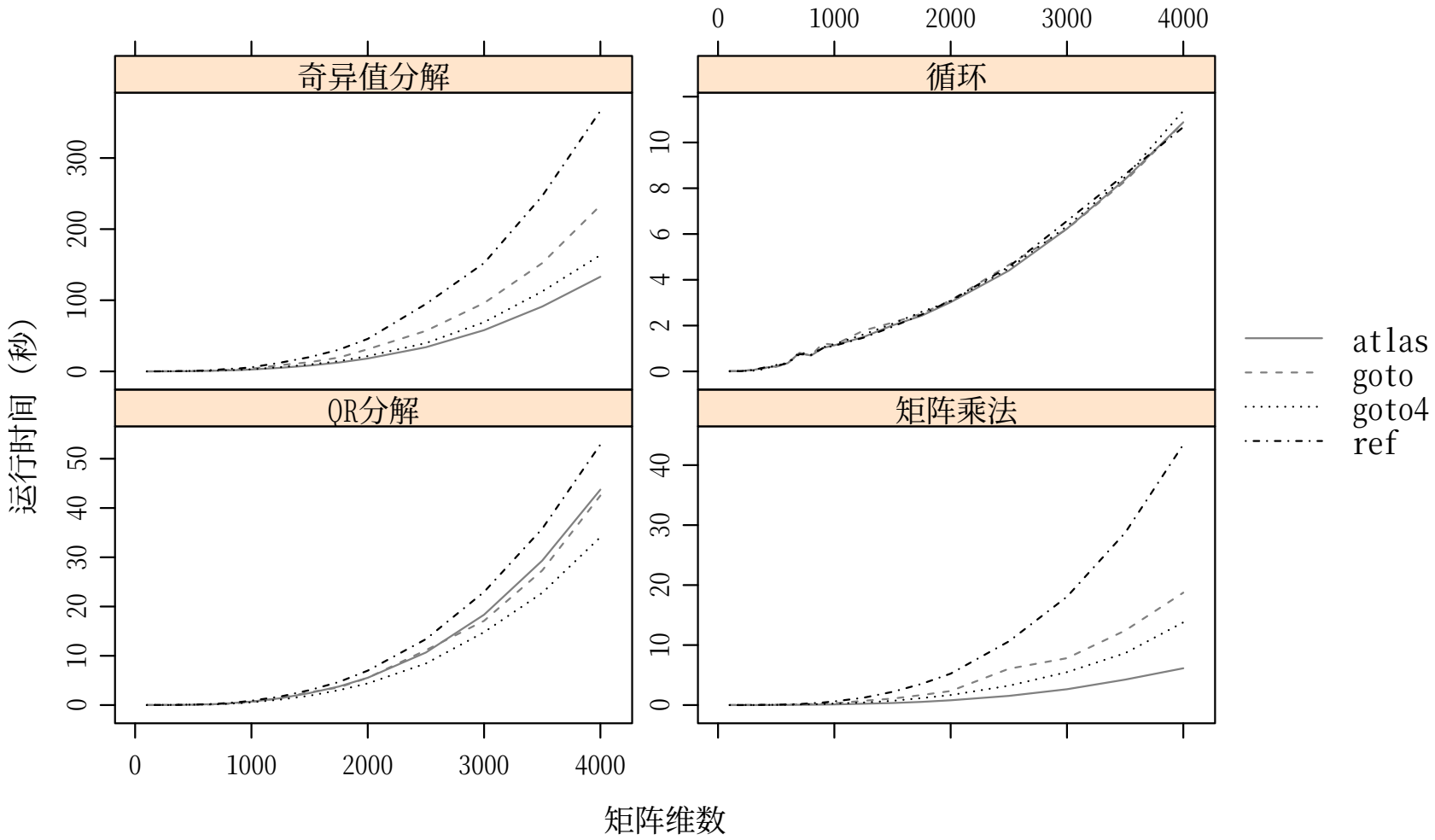
BLAS 和 LAPACK

- BLAS
  - Basic Linear Algebra Subprograms，基础线性代数程序集。
  - http://www.netlib.org/blas/。
  - 基础的线性代数操作，向量和矩阵乘法等。
  - C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh, Basic Linear Algebra Subprograms for FORTRAN usage, ACM Trans. Math. Soft., 5 (1979)。
- LAPACK
  - Linear Algebra PACKage。
  - 依赖 BLAS，解多元线性方程式、线性系统方程组的最小平方解、计算特征向量、用于计算矩阵 QR 分解的 Householder 转换、以及奇异值分解等。

优化 BLAS

- R 中操作方式
  - R 中默认的 BLAS 基于开源库，性能有很大的优化空间。
  - Windows 下直接替换安装目录中的“blas.dll”文件即可。Linux 系统下可能需要编译安装。
- 常见的 BLAS 版本
  - Atlas，netlib 上优化 BLAS 的工程，如今已经移到 http://math-atlas.sourceforge.net/，R 官网提供了一个编译的版本。
  - MKL，Intel 提供的 BLAS 与 LAPACK 接口，需要进行购买使用许可，Linux 下有非商用的免费版。
  - Goto BLAS，Kazushige Goto（后藤和茂）开发的 BLAS。http://prs.ism.ac.jp/~nakama/ 有 R 版本的 BLAS 提供下载。

Benchmarking



目 录

内存限制

- 32 位系统
  - 操作系统能支持的最大内存为 $\frac{2^{32}}{1024^3} = 4G$。
  - R 在 32 位 Windows 系统中最大使用内存为 $3G$。
  - 通常 R 在内存使用超过 $2G$（Windows）或 $3G$（Unix）时就会报错。
- 64 位系统
  - 操作系统能支持的最大内存为 $\frac{2^{64}}{1024^3} = 172 \times 10^8 G$。
  - 32 位的 R 在一些 64 位的 Windows 系统下最大内存也只有 $4G$。
  - R 能使用的内存受限于硬件和操作系统。
  - 在 R 3.0 之前，由于没有 64 位的整型数据结构，无法定义过大的矩阵，3.0 版本之后 R 中引入了长整型数据结构，理论上不再有大小的限制。

cannot allocate vector of size

- memory.size
  - `memory.size(NA)`，查看系统分配给 R 的最大内存。
  - `memory.size(FALSE)`，查看当前已经使用的内存。
  - `memory.size(TRUE)`，查看已分配的内存。
- memory.limit
  - `memory.limit()`，查看系统分配给 R 的最大内存。
  - `memory.limit(2000)`，分配最大内存为 2G。
- 在 Rgui 进行设置
  - 启动快捷图标的“目标”项中添加选项：`--max-mem-size 2Gb`。

了解 R 的内存

- 内存划分
  - 堆内存（Heap），基本单元是“Vcells”，每个大小为 8 字节。
  - 地址对（cons cells），最小单元一般在 32 位系统中是 28 字节、64 位系统中是 56 字节。
- R 的存储模式
  - `storage.mode`。
  - 对于整型矩阵，可以改变默认的存储模式，从而减小存储空间：`storage.mode(x) <- "integer"`。
- R 对象内存
  - `ls`，查看当前对象。
  - `object.size`，查看对象所占用的内存。

R 的垃圾清理

- rm() 和 gc()
  - rm 清除对象的引用。
  - gc 清扫内存空间，进行垃圾清理。
- R 的垃圾清理机制
  - 当对象清除引用之后，R 会定期对内存中的垃圾进行清理，无需人工干预。
  - 如果计算中内存不够，R 也会自动进行垃圾清理。
  - 使用 gc 函数可以手动开始一次垃圾清理。
- 程序中的垃圾清理
  - 对于长度增加的矩阵，尽量先定义一个大矩阵，然后在矩阵中操作子集，不要不断定义新的矩阵。
  - 注意清除中间对象。

内存外运算：bigmemory 家族

- bigmemory
  - 建立基于文件的大矩阵，实现 nwitch、order 等方法。
- biganalytics
  - 对于 bigmemory 对象，实现 apply、kmeans、lm、colmean 等方法。
- bigtabulate
  - 实现大矩阵的 table、tapply 等方法。
- bigalgebra
  - 实现大矩阵的 BLAS 和 LAPACK。
  - 目前 Windows/Mac/Linux 下都能使用。
- synchronicity
  - 实现 Boost mutex 的功能。

二百万条记录 60 个变量的稠密 Double 型数据的示例

- 系统环境
  - i7 CPU 2.8GHz, 3.7G RAM
  - Ubuntu 10.04, 64bit
- 示例代码

```
require(bigmemory)
require(biganalytics)
require(biglm)
a1 <- attach.resource(dget("sim.desc"))
r1 <- biglm.big.matrix(VAR01 ~ ., data = a1))
```

记录数	变量数	读取数据	回归分析	Kmeans
100 万	60	36.47 秒	16.63 秒	5.56 秒
200 万	60	78.29 秒	42.84 秒	15.48 秒
500 万	60	3 分 36 秒	1 分 59 秒	27.99 秒

Thank you!

李舰 Email: jian.li@188.com