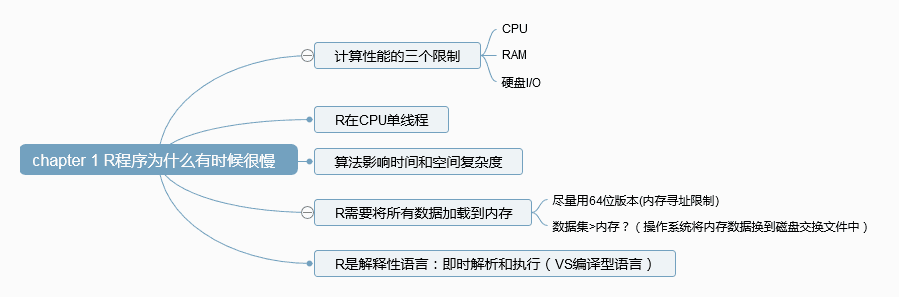
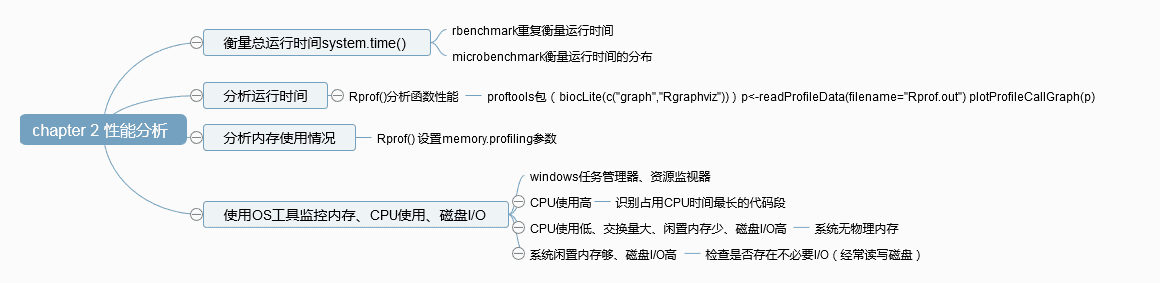
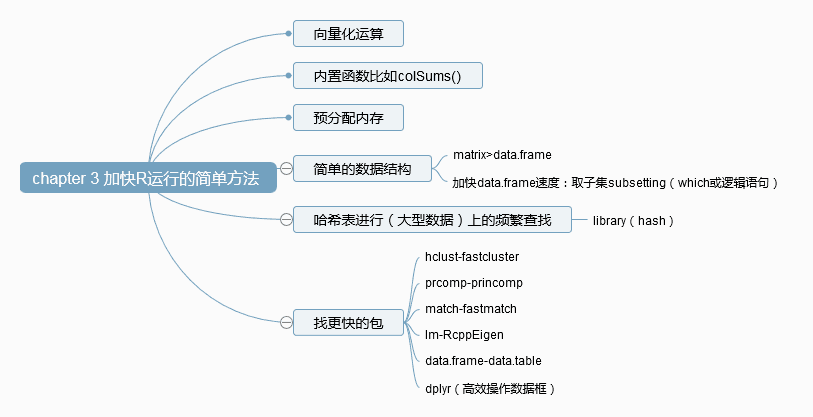
**R high performance programming**

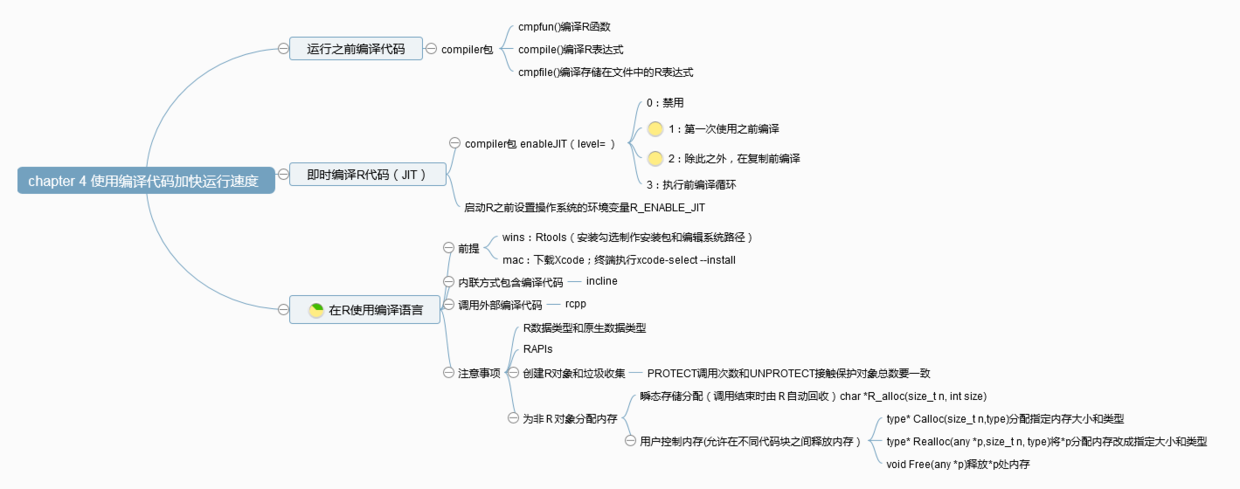
Aloysius Lim， William Tjhi （著）

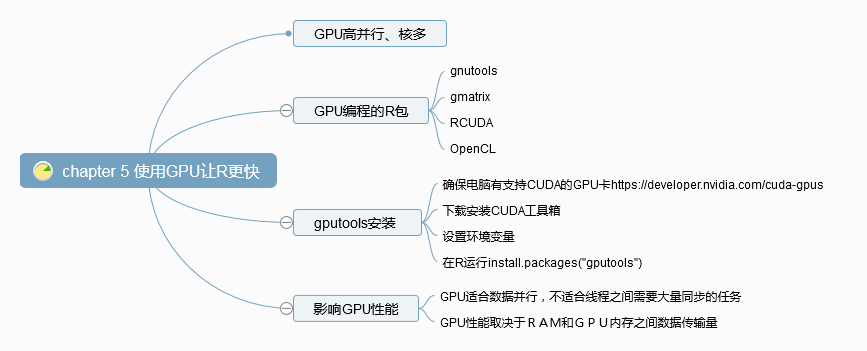
唐李洋（译）

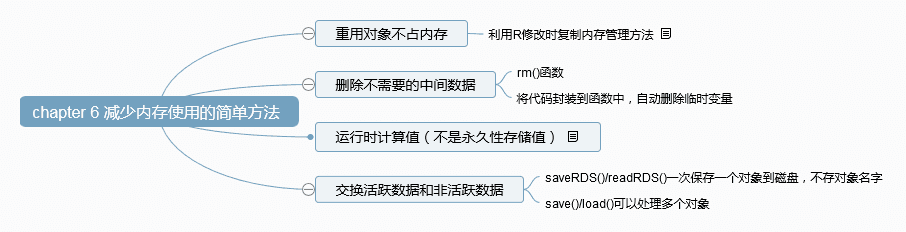










****

注解：

修改时复制copy-on-modification：

即从已有对象创建新的对象有时并不需要占用额外内存

*//*检查对象大小

object.size()

pryr包的object\_size()

*//*查看对象所指内存块

pryr包的address()

*//*追踪对象复制方法

tracemem()

运行时计算值部分 实例 -- 层次聚类

//方法1：计算每对观察值的距离矩阵然后决定哪一对最近

A<-matrix(rnorm(1E5),1E4,10)

dist\_mat<-as.matrix(dist(A))

diag(dist\_mat)<-NA

res1<-which(dist\_mat==min(dist\_mat,na.rm=T),arr.ind=T)[1,]

object\_size(A)

*##800k*

object\_size(dist\_mat)

*##801MB*

//距离矩阵需要成倍的内存空间存储所有观察对的距离

//方法二：可以逐对计算，需要内存少,但时间长

library(pdist)

temp\_res<-lapply(1:nrow(A),function(x){

temp<-as.matrix(pdist(X=A,Y=A[x,]));

temp[x]<-NA;

output\_val<-min(temp,na.rm=T);

output\_ind<-c(x,which(temp==output\_val));

output<-list(val=output\_val,ind=output\_ind);

})

val\_vec<-sapply(temp\_res,FUN=function(x) x$val)

ind\_vec<-sapply(temp\_res,FUN=function(x) x$ind)

res2<-ind\_vec[,which.min(val\_vec)]

object\_size(temp\_res)

*##2.72MB*

object\_size(val\_vec)

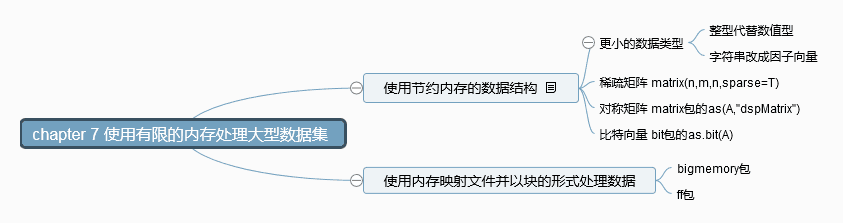
*##80kB*

object\_size(ind\_vec)

*##80.2kB*

（1）方法2在实际应用可以采用FNN包的knn()；

（2）代码并行化：用parallel包的parLApply()代替lapply()



rep.int(x,times):整型

object.size(rep.int("0123456789",1e6))

*##8000096 bytes*

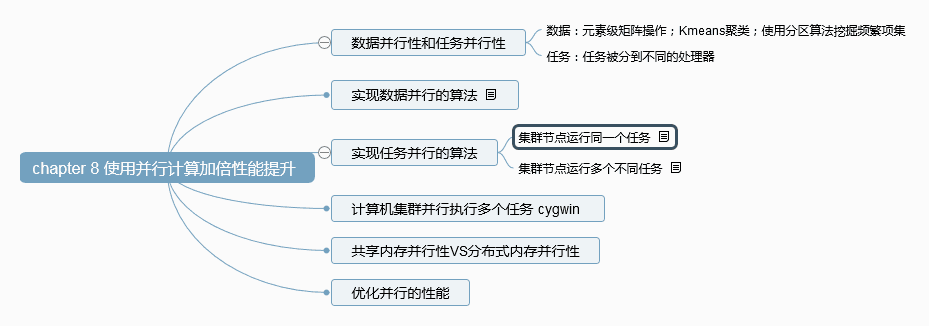
object.size(rep.int(formatC(seq\_len(1e3),width=10),1e3))

*##8056040 bytes*

object.size(formatC(seq\_len(1e6),width=10))

*##64000040 bytes*

字符型向量存储指向包含实际数据的其他向量的指针；需要的存储量取决于向量中唯一字符串的个数。



集群节点运行同一任务

*#*衡量串行算法的运行时间

*#L'Ecuyer*组合多递归生成器

RNGkind("L'Ecuyer-CMRG'")

nsamples<-5e8

lambda<-10

system.time(random1<-rpois(nsamples,lambda)

*#*在集群上生成随机数

*#*将这个任务平均分配到*worker*上

cores<-detectCores()

cl<-makeCluster(ncores)

samples.per.process<-diff(round(seq(0,nsamples,length.out=ncores+1)))

*#*在基于*socket*的集群上生成随机数之前，每个*worker*需要不同的种子来生成随机数流

clusterSetRNGStream(cl)

system.time(random2<-unlist(

parLapply(cl,samples.per.process,rpois,lambda)))

stopCluster(cl)

集群节点运行不同任务

RNGkind("L'Ecuyer-CMRG'")

nsamples<-5e7

pois.lambda<-10

system.time(random1<-list(pois=rpois(nsamples,pois.lambda),unif=runif(nsamples),norm=rnorm(nsamples),exp=rexp(nsamples)))

cores<-detectCores()

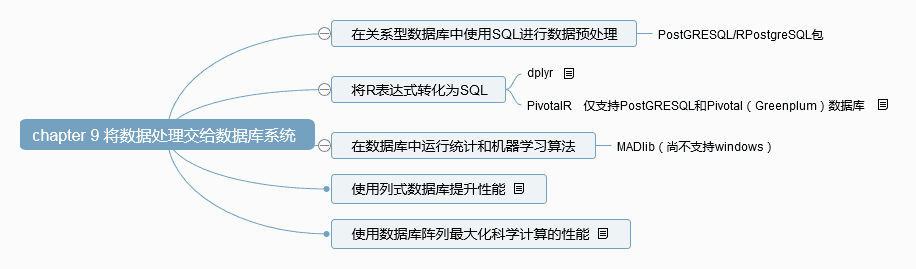
cl<-makeCluster(cores)

calls<-list(pois=list("rpois",list(n=nsamples,lambda=pois.lambda)),unif=list("runif",list(n=nsamples)),norm=list("rnorm",list(n=nsamples)),exp=list("rexp",list(n=nsamples)))

clusterSetRNGStream(cl)

system.time(random2<-parLapply(cl,calls,function(call){do.call(call[[1]],call[[2]])}))

stopCluster(c)



dplyr包

library(dplyr)

db.conn<-src\_postgres(dbname="rdb",host="hostname",port=5432,user="ruser",password="rpassword")

*#*创建两个到数据表*sales*和*trans\_items*的引用

sales.tb<-tbl(db.conn,"sales")

trans\_items.tb<-tbl(db.conn,"trans\_items")

*#inner\_join()*联结*sales*和*trans\_items*表

joined.tb<-inner\_join(sales.tb,trans\_items.tb,by="trans\_id")

*#group\_by()*根据客户*ID*对项目分组

cust.items<-group\_by(joined.tb,cust\_id)

cust.spending<-summarize(cust.items,spending=sum(price))

cust.spending<-arrange(cust.spending,desc(spending))

cust.spending<-select(cust.spending,cust\_id,spending)

*#collect()*用于运行*SQL*语句并获取结果

custs.by.spending<-collect(cust.spending)

top.custs<-head(cust.spending,10L)

*#dplyr*包提供*%>%*将操作联结起来，前面可以写为

top.custs<-sales.tb%>%inner\_join(trans\_items.tb,by="trans\_id")%>%grouped\_by(cust\_id)%>%summarise(spending=sum(price))%>%arrange(desc(spending))%>%select(cust\_id,spending)%>%head(10L)

PivotalR包

library(PibotalR)

db.conn<-db.connect(host="hostname",port=5432,dbname="rdb",user="ruser",password="rpassword")

sales.tb<-db.data.frame("sales",db.conn)

trans\_items.tb<-db.data.frame("trans\_items",db.conn)

*#*执行*SQL*并获取结果

lookat(count(sales.tb$cust\_id))

*#content*方法查看数据库服务器执行的*SQL*查询

content(max(trans\_items.tb$price))

trans<-by(trans\_items.tb['price'],trans\_items.tb$trans\_id,sum)

sales.value<-merge(sales.tb[c("trans\_id","cust\_id","store\_id")],trans,by="trans\_id")

cust.sales<-by(sales.value,sales.value$cust\_id,function(x){

trans\_count<-count(x$trans\_id)

total\_spend<-sum(x$price\_sum)

stores\_visited<-count(x$store\_id)

cbind(trans\_count,total\_spend,stores\_visited)})

names(cust.sales)<-c("cust\_id","trans\_count","total\_spend","stores\_visited")

lookat(cust.sales,5)

使用列式数据提升性能

MonetDB（https://www.monetdb.org/Downloads）

windows选择开始|程序|MonetDB|启动服务器，初始化并启动服务器。

library(MonetDB.R)

db.drv<-MonetDB.R()

db.conn<-dbConnect(db.drv,host="hostname",post=50000,dbname="rdb",user="monetdb",password="monetdb")

dbWriteTable(db.conn,"sales",sales)

dbWriteTable(db.conn,"trans\_items",trans.items)

library(microbenchmark)

microbenchmark({res<-dbGetQuery(db.conn,'SELECT store\_id,SUM(proce) as total\_sales FROM sales INNER JOIN trans\_items USING (trans\_id) GROUP BY strore\_id;')},times=10)

使用数据库阵列最大化科学计算的性能

1、下载安装SCIDB

2、在SCIDB服务器安装shim

3、从CRAN安装scidb包

library(scidb)

scidbconnect(host="hostname",port=8080)

*#*使用*as.scidb()*将数据装载到数据库

A<-as.scidb(matrix(rnorm(1200),40,30),name="A")

B<-as.scidb(matrix(rnorm(1200),30,40),name="B")

*#scidb*提供类似*r*的语法来操纵*SCIDB*矩阵和数组

