大数据编程

10

R并行计算

中央财经大学 商学院
姚凯
2016

主要内容

- R中实现并行处理
- 通过并行进行数据分析
- R中并行的优缺点

R中实现并行计算

- 一种假的多线程方式
- 其实是多进程方式

- 线程和进程的区别:
  - 一个进程可以拥有多个线程
  - 线程可以共享进程中的资源
  - 进程之间无法共享资源

Parallel in R

- Vectorized computation (apply)
- Create the matrix of the data first
- Allocate the data (row or column) to different process
- Return the results to the main process

Parallel in R

- Packages
  - Parallel
  - Snow
  - Snowfall
- Key functions
  - detectCores (parallel)
  - sfInit (snowfall)
  - sfCpus (must be used after sfInit)
  - sfApply (snowfall)
  - sfStop (snowfall)

Example 1

```
mysort <- function(x){
replicate(5, sort(x))
 return(sort(x)[1:10])
}
M = matrix(rnorm(10000000), 100, 100000)
print('sequence run:')
print(system.time(x<-apply(M, 2, mysort)))
```

Example 2

```
library("snow")
library("snowfall")
library("parallel")
mysort <- function(x)
{
 replicate(5, sort(x))
  return(sort(x)[1:10])
}
M = matrix(rnorm(10000000), 100, 100000)
cpus= detectCores ();
sfInit(parallel=TRUE, cpus=cpus)
print(sprintf('%s cpus to be used', sfCpus()))
print('parallel time cost:')
print(system.time(x<-sfApply(M, 2, mysort)))
sfStop()
```

sfInit VS sfStop

- sfInit: 同时开启多个进程
- sfStop: 结束多个进程

Example 3

```
mysort <- function(x){
replicate(5, sort(x))
message(msg);
 return(sort(x)[1:10])
}
M = matrix(rnorm(10000000), 100, 100000)
msg ="Hello parallel";
print('sequence run:')
print(system.time(x<-apply(M, 2, mysort)))
```

Example 4

```
mysort <- function(x)
{
replicate(5, sort(x))
message(msg);
 return(sort(x)[1:10])
}
M = matrix(rnorm(10000000), 100, 100000)
msg ="Hello parallel";
cpus= detectCores ();
sfInit(parallel=TRUE, cpus=cpus)
print(sprintf('%s cpus to be used', sfCpus()))
print('parallel time cost:')
print(system.time(x<-sfApply(M, 2, mysort)))
sfStop()
```

Share data with the slaver

- sfExport()
- sfExportAll()

Example 5

```
mysort <- function(x)
{
replicate(5, sort(x))
message(msg);
 return(sort(x)[1:10])
}
M = matrix(rnorm(10000000), 100, 100000)
msg ="Hello parallel";
cpus= detectCores ();
sfInit(parallel=TRUE, cpus=cpus)
sfExport("msg");
print(sprintf('%s cpus to be used', sfCpus()))
print('parallel time cost:')
print(system.time(x<-sfApply(M, 2, mysort)))
sfStop()
```

Example 6

```
library("taRifx")
mysort <- function(x){
replicate(5, sort(x))
x=c(1,2,3); y=c(2,3,1); fm = data.frame(x,y);
sort.data.frame(fm,formula=~y);
 return(sort(x)[1:5])
}
M = matrix(rnorm(100), 10, 10)
print('sequence run:')
print(system.time(x<-apply(M, 2, mysort)))
```

Example 7

```
library("taRifx")
mysort <- function(x){
replicate(5, sort(x))
x=c(1,2,3); y=c(2,3,1); fm = data.frame(x,y);
sort.data.frame(fm,formula=~y);
 return(sort(x)[1:10])
}
M = matrix(rnorm(10000000), 100, 100000)
cpus= detectCores ();
sfInit(parallel=TRUE, cpus=cpus)
print(sprintf('%s cpus to be used', sfCpus()))
print('parallel time cost:')
print(system.time(x<-sfApply(M, 2, mysort)))
sfStop()
```

Load the library on all clusters

- sfLibrary()

Example 8

```
library("taRifx")
mysort <- function(x){
replicate(5, sort(x))
x=c(1,2,3); y=c(2,3,1); fm = data.frame(x,y);
sort.data.frame(fm,formula=~y);
 return(sort(x)[1:10])
}
M = matrix(rnorm(10000000), 100, 100000)
cpus= detectCores ();
sfInit(parallel=TRUE, cpus=cpus)
sfLibrary(taRifx);
print(sprintf('%s cpus to be used', sfCpus()))
print('parallel time cost:')
print(system.time(x<-sfApply(M, 2, mysort)))
sfStop()
```

Summary

- R中并行的粒度相对比较大
- 通过多进程的方式实现并行
- 如何实现
  - Snowfall
  - 共享变量
  - 共享包

课堂练习1

- 下载数据
- 将数据拆分为多个小数据
- 利用并行编程方法统计每个城市中不同菜系的个数和平均价格

课堂练习2

- 重写爬虫程序
- 利用并行计算，同时爬取多个URL的商品
- 将结果保存到csv文件中