MINICURSO

Uma Introdução à Técnicas de Paralelismo Computacional em R

Robson Ortz O. Cunha
robson.ortz@gmail.com

Dr. Clécio da Silva Ferreira
clecio.ferreira@ufjf.edu.br

Departamento de Estatística

Universidade Federal de Juiz de Fora

Introdução

O aumento da geração de informação, vivido nos últimos anos, gerou um acumulo gigantesco de dados, o que criou a necessidade de ferramentas para tratar e extrair informações úteis de tais conjuntos. Assim, métodos computacionais, cada vez mais sofisticados, vem sendo desenvolvidos a fim de tratar o problema do processamento e tratamento dessa sobrecarga de informação.

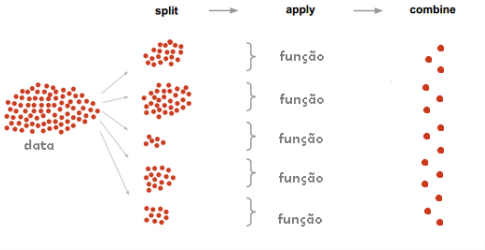
Introdução

Formas de acelerar seu programa, segundo Matloff (2014):

- vetorização - evita o uso de '*loop(s)*' em processos iterativos;
- código compilado (C) - permite um controle maior do código;
- **código paralelizado**.

Introdução

- **Objetivo:** quebrar uma base de dados em partes gerenciáveis, operar em cada peça e, enfim, combinar os resultados obtidos do processamento em uma única saída.
- uso da estratégia 'split-apply-combine'.
- ambiente paralelo: *multicore*, *cluster* e *threads*.
- no R: ***plyr***, *doSNOW*, *doParallel*, *doMC*, entre outros.



Pacote '*plyr*'

Principais Funções

O pacote *plyr* constrói suas funções sobre um mecanismo *built-in*:

INPUT + OUTPUT + ply()

Esse tipo de mecanismo permite controlar os formatos de entrada e saída mantendo uma sintaxe consistente em todas as variações. O quadro a seguir apresenta o resumo das principais funções do pacote.

Entrada	Saída			
	array	*data.frame*	*list*	descarte
array	aaply	adply	alply	a_ply
data.frame	daply	ddply	dlply	d_ply
list	laply	ldply	llply	l_ply

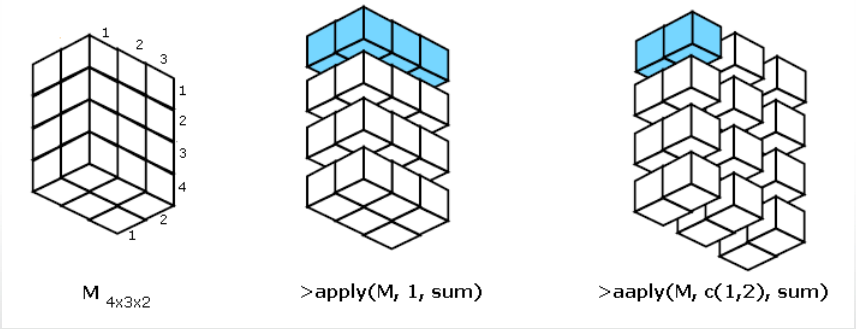
Pacote '*plyr*'
Aplicações

Acesso e Preparo dos Dados:

- **Objetivo:** acessar uma série de bancos de dados presente em um diretório na máquina.
- **Método:** listar os arquivos presentes no diretório, criar uma função para lê-los de forma automática e guardar seu conteúdo em forma de lista ou objetos criados a partir de um vetor de '*strings*' aleatório.

Pacote '*plyr*'
Aplicações

- **Objetivo:** calcular estatísticas ou funções por parte de matrizes 2D (linha ou coluna) e de matrizes 3D (linhas, colunas e categorias).
- Para matrizes 3D, temos a seguinte ilustração para a função '*aaply()*' utilizada:



A linguagem *R* apresenta diversos pacotes que possibilitam e facilitam o trabalho em ambiente paralelo. A abordagem de tais técnicas no processamento foi realizada:

- em ambiente *multicore*;
- via *threads*[1];
- processo de clusterização, por aprendizado não supervisionado.

Principais pacotes utilizados em todo processo: *doSNOW, doParallel, doMC*.

[1] termo em inglês para linha de encadeamento de execução, em termos computacionais é o modo de um processo se auto dividir em duas ou mais partes para executar concorrencialmente uma série de tarefas.

Método geral para construção de códgo em paralelo

Os pacotes que permitem o processamento em paralelo no R seguem basicamente uma estrutura única de inicialização do ambiente, bem como sua finalização. Assim, podemos resumir tal procedimento na seguinte rotina:

- chamada da biblioteca;
- criação de *clusters* via função `makeCluster(n,type)`;
- ativação dos *clusters* alocados via `registerDoSNOW(cl)`, por exemplo, onde cl é o objeto referente ao número de *clusters* solicitados;
- ao fim do processamento, cessamos seu uso utilizando `stopCluster(cl)`.

R em Ambiente Paralelo
doSNOW

O pacote *doSNOW* consiste na junção dos pacotes *snow, foreach* e *iterators* presentes nas antigas versões do *R*.

Principais funções do pacote **doSNOW**:

- ***clusterExport()*** - exporta uma variável passada na função para o ambiente global;
- ***clusterApply()*** - a função faz com que cada *cluster* gerencie uma parte do banco de dados reparticionado via *apply*, é importante ressaltar que o número de partes gerenciáveis da base é igual ao número de *clusters* enviado;
- ***clusterApplyLB()*** - a função reparticiona a base de dados a fim de que cada *cluster* gerencie o processamento de uma parte, o grande diferencial dessa função encontra-se na distribuição de cargas de trabalho para cada *cluster*.

R em Ambiente Paralelo

doSNOW - *clusterApply()* vs. *clusterApplyLB()*

- Saída do código (exemplo 2 - “3.Pacotes em Paralelo"):

```
> system.time({
+ Mlines<-clusterApply(cl, ssplt, function(x) apply(x, 1, mean))
+ })
  usuÃ¡rio   sistema decorrido
    0.128     0.020     0.455
> system.time({
+ MlinesLB<-clusterApplyLB(cl, ssplt, function(x) apply(x, 1, mean))
+ })
  usuÃ¡rio   sistema decorrido
    0.124     0.028     0.432
>
```

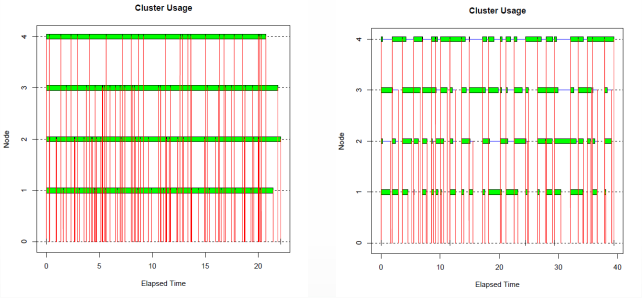
...

```
> A=numeric(0)
> B=numeric(0)
>
> for(i in 1:1000){
+ A[i]=unname(system.time({
+ Mlines<-clusterApply(cl, ssplt, function(x) apply(x, 1, mean))
+ })[3])
+
+ B[i]=unname(system.time({
+ MlinesLB<-clusterApplyLB(cl, ssplt, function(x) apply(x, 1, mean))
+ })[3])
+ }
> mean(A)
[1] 0.488331
> mean(B)
[1] 0.471778
>
```

R em Ambiente Paralelo

doSNOW - *clusterApply()* vs. *clusterApplyLB()*

- Na literatura (McCallum, 20012):



(a) Análise de tempo e carga dos *clustes* via função *clusterApplyLB*

(b) Análise de tempo e carga dos *clustes* via função *clusterApply*

Figura: Gráficos da análise de tempo de comunicação entre *'Master-Workers'*.

R em Ambiente Paralelo
doParallel

O pacote *doParallel* consiste na junção dos pacotes *parallel, multicore* e *snow*. O pacote foi desenvolvido para versão *R 2.14.0* e tem por funcionalidade otimizar as funções dos pacotes integrados, além de gerar de forma integrada números aleatórios.

doParallel ainda tem disponível em seu diretório todas as funções do pacote ***doMC***, que trabalha diretamente em *multicores* (utilizando os núcleos disponíveis como processadores indivisíveis simultâneos).

Principais funções do pacote **doParallel**:

- ***parLapply()*** e ***mclapply()*** - ambas são a versão paralelizada de *laply()*. Têm como *input* dados do tipo *list*, onde cada entrada é utilizada como base de processamento nos *clusters*. Dentro de cada *cluster* será executado a função *laply()* no pedaço por ele gerenciável.

Aplicações
Modelo de Regressão Linear Múltipla (MRLM)

Em notação, o Modelo de Regressão Linear Múltipla é dado por

$$\mathbf{Y}_{[n\times 1]} = \mathbf{X}_{[n\times p]} \cdot \boldsymbol{\beta}_{[p\times 1]} + \boldsymbol{\epsilon}_{[n\times 1]}$$

tal que, $\boldsymbol{\epsilon} = \mathbf{Y} - \hat{\mathbf{Y}}$ e $\hat{\mathbf{Y}} = \mathbf{X}\boldsymbol{\beta}$, pressupondo que $\epsilon_i \sim N(0, \sigma^2)$. A estimação de $\boldsymbol{\beta}$ pode ser realizado via método de mínimos quadrados, máxima verossimilhança ou via resultados da família exponencial para estatísticas suficientes. Em geral, obtém-se

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y} \quad e \quad \hat{\sigma}^2 = \frac{(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}})^T(\boldsymbol{Y} - \boldsymbol{Y}\hat{\boldsymbol{\beta}})}{n-p}.$$

Aplicações
Estimação Paramétrica Via '*bootstrap*'

- **Objetivo:** estimar parâmetros segundo um MRLM por processo de reamostragens, paralelizando as múltiplas iterações do processo; verificar a veracidade dos resultados e obter os tempos de processamento para variação de *clusters* disponíveis.
- **Método:** explorar a função *foreach()* no processo iterativo e comparar o seu rendimento com a função *for()* (*loop* em serial).

Aplicações

Resultados da estimação e tempos de processamento via '*bootstrap*'

	clusters = 4		
bootstrap	3.060258	1.779403	2.649605
lm	3.086	1.585	2.715
β	3.0	1.8	2.7

Tabela 1: Resultados de estimação para o vetor paramétrico, aplicação *bootstrap*.

	foreach			**for**
Clusters	4	12	24	
usuário	3.504	3.992	4.132	12.912
sistema	0.336	0.216	0.292	0.028
total	6.936	5.198	4.645	12.943

Tabela 2: Tempos obtidos pela estimação paramétrica via bootstrap para o caso sequencial (*for*) e paralelo (*foreach*).

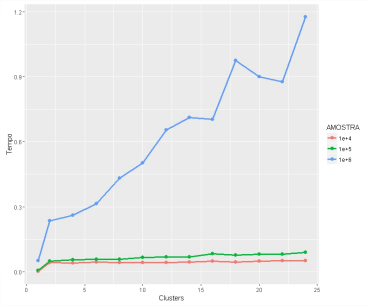
Aplicações

Estimação Paramétrica do Modelo Normal (via Operações Matriciais)

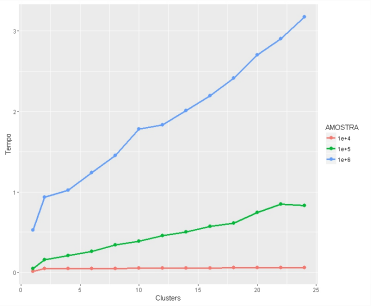
- **Objetivo:** estimar parâmetros segundo um MRLM por operações matriciais, paralelizando cada operação, afim de analisar os tempos e ganhos de processamento para variação de *clusters* disponíveis e tamanhos amostrais.
- **Método:** explorar a função *pMM()* no processo; aumentar o tamanho da amostra, bem como o número de variáveis explicativas.

Aplicações

Resultados - Tempos de processamento para simulação



(a) p=10

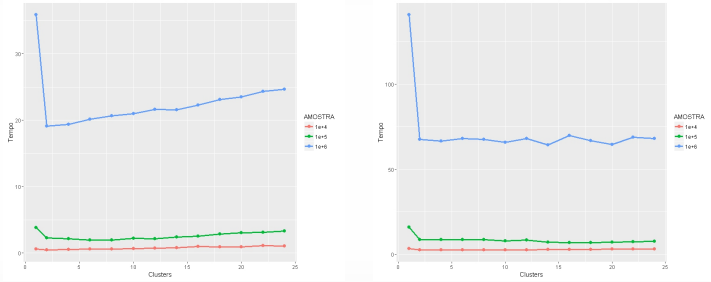


(b) p=100

Figura: Gráficos da análise de tempo de processamento levando em conta a variação do tamanho da amostra e número de '*workers*' ativos.

Aplicações

Resultados - Tempos de processamento para simulação



(a) p=500 (b) p=1000

Figura: Gráficos da análise de tempo de processamento levando em conta a variação do tamanho da amostra e número de '*workers*' ativos.

R em Ambiente Paralelo
Lei de Amdahl

A Lei de Amdahl, criada pelo projetista de computadores Gene Amdahl, explica o aumento de velocidade de um determinado programa como resultado de computação paralela, sendo assim, usada para encontrar a máxima melhora esperada para um sistema.

Leva em conta a análise do *speedup*, em notação

$$S = \frac{T(1)}{T(n)},$$

onde $n \in \mathbb{N}$ denota o numero de *threads* de execução.

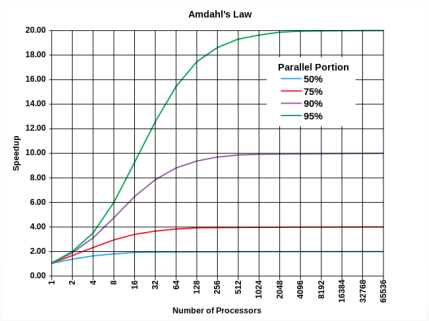
A Lei de Gustafson-Barsis, criada pelo cientista da computação John L. Gustafson, aborda as lacunas da Lei de Amdahl , que é baseada na suposição de um fixo tamanho do problema. Ela diz-nos que o '*speedup*' máximo que uma aplicação paralela com p processadores pode obter é:

$$S(P) = P - \alpha(P - 1)$$

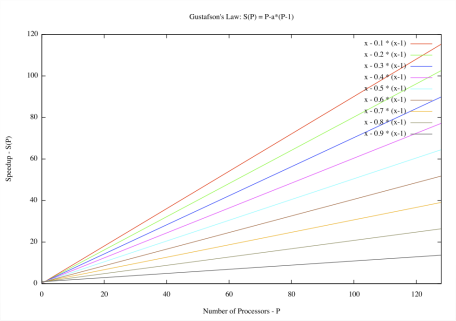
ond P é o número de processadores disponíveis e α é a parte não paralelizável do programa.

R em Ambiente Paralelo

Ilustração gráfica das Leis de Amdahl e Gustafson



(a) Lei de Amdahl



(b) Lei de Gustafson

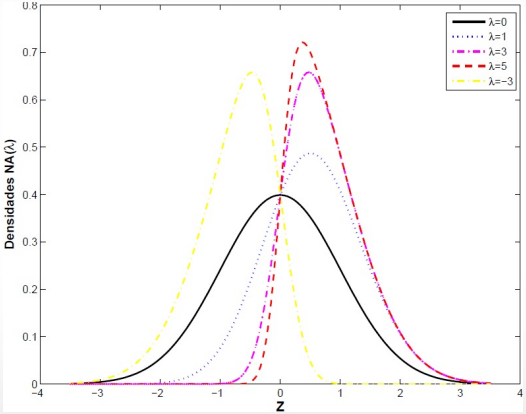
Figura: Gráficos dos '*speedups*' para as Leis de Amdahl e Gustafson.

SN - Definição e propriedades

1. $f(y) = 2\phi(y|\mu, \sigma^2)\Phi_1\left(\frac{\lambda(y-\mu)}{\sigma}\right), \quad y \in \mathbb{R}$ (Azzalini, 1985)
   Notação: $Y \sim SN(\mu, \sigma^2, \lambda)$.
2. Representação estocástica (Henze, 1986): Se $U, V \sim N(0,1)$, independentes, então $Y = \mu + \sigma\left(\delta|U| + (1-\delta^2)^{1/2}V\right) \sim SN(\mu, \sigma^2, \lambda)$
3. Representação hierárquica: $Y|Z = z \sim N(\mu + \sigma\delta, \sigma^2(1-\delta^2))$, $Z = |U| \sim HN(0,1)$, com $\delta = \frac{\lambda}{(1+\lambda^2)^{1/2}}$

Aplicação em Modelos Normais Assimétricos

Gráficos da função densidade de probabilidade (fdp)



Conclusão

Os resultados obtidos, mostraram que a capacidade de processamento em paralelo é limitada, uma vez que está atinge um ponto de otimização no processo, evidenciando fortemente os efeitos da Lei de Amdhal. Mas, mesmo com limitação na melhora de processamento, ainda sim os estudos se mostraram de grande valia, já que facilitam as análises sobre grandes bases de dados (*Big Data*), principalmente em casos que o número de variáveis é excessivamente grande.

Referências

McCallum, Q. E. and Weston, S. (2012). Parallel R. O'REILLY, United States of America, 1st edition.

Emerson, J. W. and Kane, M. J. (2014). The r package bigmemory: Supporting eficient computation and concurrent programming with large data sets. Journal of Statistical Software, Volume VV.

R Core Team (2015). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Matloff, N. (2014), Parallel Computing for Data Science: With Examples in R, C++ and CUDA , Chapman & Hall/CRC The R Series (Book 28), University of California, Davis.

Agradecimentos

UNIVERSIDADE
FEDERAL DE JUIZ DE FORA