Otimização de Código

Marco A P **D´A**ndrade
mda@rio.pm.org

Objetivos

Destacar alguns pontos que julgo relevantes para avaliados

Utilizar casos reais como referência

Até que pondo devemos otimizar ?

Elementos envolvidos

Esta questão dá um passo além da produção de código!

Devemos considerar todos os elementos

- Dados - Acesso e volume
- Recursos computacionais disponíveis
- Linguagem – recursos e feramentas para medição
- Manutenibilidade versus otimização de código

Elementos envolvidos

Esta questão dá um passo além da produção de código!

Devemos considerar todos os elementos

- Dados - Acesso e volume
- Recursos computacionais disponíveis
- Linguagem – recursos e feramentas para medição
- Manutenibilidade versus otimização de código

Dados – Acesso e Volume

a) O tempo de acesso à informação pode ser melhorado?

b) É possível o fracionar os dados para processamento?

c) Podemos dividir o processamento em multiplas fases com base em dados ou tempo?

d) Poderia ser utilizada alguma forma de cacheamento?

e) Estamos levando o mínimo de dados à memória?

f) Estas ações trazem algum tipo de ganho ? Avalie e priorize!

Elementos envolvidos

Esta questão dá um passo além da produção de código!

Devemos considerar todos os elementos

- Dados - Acesso e volume
- Recursos computacionais disponíveis
- Linguagem – recursos e feramentas para medição
- Manutenibilidade versus otimização de código

Otimizar os recursos computacionais disponiveis

- Tempos de acesso IO (Disco, rede) e Processamento (cpu, memória)
- Custo dos componentes está reduzindo!

Otimizar os recursos computacionais disponiveis

- Tempos de acesso IO (Disco, rede) e Processamento (cpu, memória)
- Custo dos componentes está reduzindo!

Otimizar os recursos computacionais disponiveis

- Tempos de acesso IO (Disco, rede) e Processamento (cpu, memória)
  - Lembre-se, não temos uma situação computacional ideal!
  - Ler dados tem um custo em termos de recursos/tempo!
  - Considere estes elementos como "**custo de sistema**", e lembre-se, sem a troca de hardware só podemos minimizar o uso.

Otimizar os recursos computacionais disponiveis

- Tempos de acesso IO (Disco, rede) e Processamento (cpu, memória)
- Custo dos componentes está reduzindo!

Otimizar os recursos computacionais disponíveis

Custo dos componentes está reduzindo!

- Esta é uma das afirmações mais freqüentes quando toma-se a opção por não otimizar código, mas nem sempre estamos limitados apenas pelo custo!
- Dispomos de meios para fazer a aquisição ?
- O que estamos desenvolvendo pode ser limitado ao hardware ?

Elementos envolvidos

Esta questão dá um passo além da produção de código!

Devemos considerar todos os elementos

- Dados - Acesso e volume
- Recursos computacionais disponíveis
- Linguagem – recursos e feramentas para medição
- Manutenibilidade versus otimização de código

Linguagem – recursos e ferramentas para medição

Vamos fazer uma rápida introdução sobre alguns dos recursos que podem ser explorados

- Benchmark
- Devel::DProf
- Devel::Size
- re (é! re.pm!)

Linguagem – recursos e ferramentas para medição

Vamos fazer uma rápida introdução sobre alguns dos recursos que podem ser explorados

- Benchmark
- Devel::DProf
- Devel::Size
- re (é! re.pm!)

Linguagem – recursos e ferramentas para medição

Benchmark - benchmark running times of Perl code

Este é sem duvida a porta de entrada para a otimização de código em Perl, trazendo ganhos iniciais já com seu teste mais básico.

- Vamos abordar apenas 2 recursos:
  - Medição de tempo - timediff e timestr
  - Comparação de algoritmos - timethese

Linguagem – recursos e ferramentas para medição

Benchmark - benchmark running times of Perl code

Este é sem duvida a porta de entrada para a otimização de código em Perl, trazendo ganhos iniciais já com seu teste mais básico.

- Vamos abordar apenas 2 recursos:
  - Medição de tempo - timediff e timestr
  - Comparação de algoritmos - timethese

Linguagem – recursos e ferramentas para medição

Benchmark – Medição de tempo (User e System)

Nesta abordagem teremos a medição do tempo para executar um pequeno trecho de código:

```
use Benchmark;
$t0 = new Benchmark;
# ... your code here ...
for (  $n=0; $n< 10240000; $n++ ) { $t += $n; }
$t1 = new Benchmark;
$td = timediff($t1, $t0);
print "the code took:",timestr($td),"\n";
```

Resultado básico:

```
the code took: 5 wallclock secs ( 4.63 usr +  0.00 sys =  4.63 CPU)
```


Linguagem – recursos e ferramentas para medição

Benchmark - benchmark running times of Perl code

Este é sem duvida a porta de entrada para a otimização de código em Perl, trazendo ganhos iniciais já com seu teste mais básico.

- Vamos abordar apenas 2 recursos:
  - Medição de tempo - timediff e timestr
  - Comparação de algoritmos - timethese

Linguagem – recursos e ferramentas para medição

Benchmark – Comparação de algoritmos - timethese

Outra abordagem muito interessante é verificar, dentre alternativas, qual tem o melhor resultado para nossa necessidade!

```
use Benchmark qw( cmpthese );
cmpthese( -1, { a => "++\$i", b => "\$i *= 2" } ) ;
```

Resultado básico:

```
        Rate    b    a
b  6327831/s   -- -58%
a 15238820/s 141%   --
```


Linguagem – recursos e ferramentas para medição

Vamos fazer uma rápida introdução sobre alguns dos recursos que podem ser explorados

- Benchmark
- Devel::DProf
- Devel::Size
- re

Linguagem – recursos e ferramentas para medição

Devel::DProf - a Perl code profiler

A finalidade deste módulo é identificar quais as subrotinas que consomem mais tempo na execução de sua aplicação.

A abordagem é basicamente:

perl ***-d:DProf*** script.pl

Linguagem – recursos e ferramentas para medição

Devel::DProf - a Perl code profiler

Para demonstrar sua funcionalidade, preparei um código qualquer, que le todos os PODs armazenados em um diretorio, armazena em um ARRAY, contabiliza as 10 palavras mais utilizadas:

```
Carregadas 166254 linhas!
  ----------------------
31546 the
31340
17044 to
14322 a
12338 of
11528 is
11038 and
 9656 in
 7026 for
 6088 that
  ----------------------
```


Linguagem – recursos e ferramentas para medição

Devel::DProf - a Perl code profiler

```
use Devel::Size qw(size total_size);
  foreach my $File ( glob('C:\Perl\lib\pod\*') ) {
      unless ( open(F, $File) ) { warn "Falha ao abrir arquivo: $File $!";  next; }
      LoadFile(*F);
  }
  CountWord();
  TopTenWords();
 my @x = keys %Words;

sub TopTenWords {
 my $Max=10;
  print "  ----------------------\n";
  foreach ( sort { $Words{$b} <=> $Words{$a} } keys %Words ) {
      next if ( /[\W\s]/ );
      printf "%5s %s\n", $Words{$_}, $_;
      last unless ( --$Max );
  }
  print "  ----------------------\n";
}

sub LoadFile {
 my $fh = shift;
  while ( defined( $_ = <$fh> ) ) { push(@Data, $_); }
}

sub CountWord {
  foreach my $l ( @Data ) {
      map { $Words{$_}++ } split(/\s+/, $l);
  }
}
```


Linguagem – recursos e ferramentas para medição

Devel::DProf - a Perl code profiler

Como resultado da avaliação Devel::DProf temos a contabilização do tempo de execução e a distribuição por subrotinas:

```
Total Elapsed Time = 2.780628 Seconds
  User+System Time = 2.561628 Seconds
Exclusive Times
%Time ExclSec CumulS #Calls sec/call Csec/c  Name
 47.5   1.219  1.219      1   1.2190 1.2190  main::CountWord
 26.2   0.672  0.672      1   0.6720 0.6720  main::TopTenWords
 12.2   0.313  0.313    160   0.0020 0.0020  main::LoadFile
 0.62   0.016  0.016      2   0.0080 0.0080  DynaLoader::BEGIN
 0.59   0.015  0.015      1   0.0150 0.0150  File::Glob::doglob
 0.00   0.000  0.000      1   0.0000 0.0000  File::Glob::GLOB_BRACE
```


Linguagem – recursos e ferramentas para medição

Devel::DProf - a Perl code profiler

Neste levantamento fica evidente que os melhores resultados serão obtidos com eventual otimização em CountWord, TopTenWords ou LoadFile...

Em um primeiro esforço, que não acrescentei nos slides:

- acrescentei o módulo Benchmark para distinguir tempos de User e Sys
- Unifiquei LoadFile e CountWords, descartando o uso do array @Data...
  Trazendo para memória o mínimo possível de informação

```
Versão  1:  2 wallclock secs ( 1.59 usr +  0.13 sys =  1.72 CPU)
        2:  2 wallclock secs ( 1.34 usr +  0.16 sys =  1.50 CPU)
```

```
sub LoadFile {
 my $fh = shift;
  while ( defined( $_ = <$fh> ) )
{ push(@Data, $_); }
}

sub CountWord {
  foreach my $l ( @Data ) {
      map { $Words{$_}++ }
split(/\s+/, $l);
  }
}
```

```
sub CountWord {
 my $File = shift;
  return unless ( open(F, $File) );
  while ( defined( my $l = <F> ) ) {
      map { $Words{$_}++ }
              split(/\s+/, $l);
  }
}
```

Linguagem – recursos e ferramentas para medição

Vamos fazer uma rápida introdução sobre alguns dos recursos que podem ser explorados

- Benchmark
- Devel::DProf
- Devel::Size
- re

Linguagem – recursos e ferramentas para medição

Devel::Size - Perl extension for finding the memory usage of Perl variables

Utilizando a primeira versão do código exemplo, acrescentei uma verificação da area de memória utilizada para armazenar os dados:

```
use Devel::Size qw(total_size);
 printf "Dados processados: %s bytes\n”.
          \t\tMemoria ocupada: Array: %s - %s linhas" .
          "\t\tHASH: %s %s palavras\n\n",
     $Size,
     total_size(\@Data),   scalar(@Data),
     total_size(\%Words),  scalar(@x);

Dados processados: 5.404.772 bytes
          Memoria ocupada:
          Array: 10.443.484 - 166.254 linhas
          HASH: 4.779.018 85.548 palavras
```

Apesar de rápida abordagem, este módulo nos demonstra a memoria alocada para a manipulação dos dados.

Linguagem – recursos e ferramentas para medição

Vamos fazer uma rápida introdução sobre alguns dos recursos que podem ser explorados

- Benchmark
- Devel::DProf
- Devel::Size
- re

Linguagem – recursos e ferramentas para medição

re - Perl pragma to alter regular expression

Como o tema de hoje é otimização de código, passo apenas uma pista sobre depuração, e otimização de Expressões regulares!

perldebguts - Guts of Perl debugging - Seção "*Debugging regular expressions*"

```
use re qw(debug);


use re qw(debug);
@x = ( $Data =~ /(?:perl)/g );
no re qw(debug)

Compiling REx `(?:perl)'
size 3 Got 28 bytes for offset annotations.
first at 1
   1: EXACT <perl>(3)
   3: END(0)
anchored `perl' at 0 (checking anchored isall) minlen 4
Offsets: [3]
        4[4] 0[0] 9[0]
```


Linguagem – recursos e ferramentas para medição

Vamos fazer uma rápida introdução sobre alguns dos recursos que podem ser explorados

- Benchmark
- Devel::DProf
- Devel::Size
- re (é! re.pm!)

Dados – Acesso e Volume

Caso 1 – Processamento diario de aproximados 50K arquivos transferidos via FTP

- Em intervalos determinados cada ponto de execução faz upload de Logs de transações e falhas de conexão, armazenando em um diretorio

- Processamento diário, uniformização de dados e junção em arquivo único.

Resultado: Logs diários de aproximados 5MB.

Este processo aparentemente simples tinha um ponto de falha... você saberia determinar?

R: Numero de entradas em um diretório...
Espaço em disco... arquivos médios de 1KB em unidades de alocação de 4K !

Dados – Acesso e Volume

Caso 1 – Processamento diario de aproximados 50K arquivos transferidos via FTP

a) O tempo de acesso à informação pode ser melhorado?

50K arquivos armazenados em único diretorio... extremamente lento!

b) É possível o fracionar os dados para processamento?

Processos intermediarios podem transferir para outra area ?

c) Podemos dividir o processamento em multiplas fases com base em dados ou tempo?

Podemos processar com maior regularidade, e gerar log intermediario ?

d) Poderia ser utilizada alguma forma de cacheamento?

e) Estamos levando o mínimo de dados à memória?

f) Estas ações trazem algum tipo de ganho ? Avalie e priorize!

Dados – Acesso e Volume

Caso 2 – Avaliação de área ocupada pelos usuários de correio Click21

Problema:

Identificar maiores usuários e traçar perfil de utilização

Dados envolvidos:

1.000.000 de usuários, quota de 500MB – maildir (1 arquivo por mensagem)

Acesso aos dados via NFS - otimização para acessos concorrentes

Abordagem

Processamento por multiplos processos (forks)

Registro pontual em base de dados

Resultado

Dados disponíveis com uma defasagem, aceitável, de até 36 horas

Dados – Acesso e Volume

Caso 2 – Avaliação de área ocupada pelos usuários de correio Click21

a) O tempo de acesso à informação pode ser melhorado?

NFS é otimizado para acessos concorrentes...

Somente dividindo processos - forks

b) É possível o fracionar os dados para processamento?

Carregar listas individuais por processo é essencial.

Isolados grupos de usuarios por processo

c) Podemos dividir o processamento em multiplas fases com base em dados ou tempo?

Processamento ininterrupto – Agregado a outras funções essenciais

- (é recomendado)

d) Poderia ser utilizada alguma forma de cacheamento?

Dados armazenados em banco de dados para consulta sob demanda.

e) Estamos levando o mínimo de dados à memória?

f) Estas ações trazem algum tipo de ganho ? Avalie e priorize!

Duvidas?

Marco A P **D´A**ndrade
mda@rio.pm.org

Obrigado!

Marco A P **D´A**ndrade
mda@rio.pm.org