Escalando o AdvPL

Marcelo Camargo <marcelo.camargo@ngi.com.br> - 15/07/2016

Este documento provém de uma pesquisa com o objetivo de escalar o *webservice* utilizado pelo aplicativo **MNT NG**. Visa abordar otimizações simples que possuem um impacto enorme e exponencial na performance, mostrar comparativos e apresentar uma solução para a nova arquitetura do *webservice*.

Modelo atual

O modelo atual de sincronização do *webservice* trabalha com múltiplos métodos, chamados paralelamente/assincronamente, que acessam os dados modificados das tabelas do *Protheus* e usam um *parser* interno para gerar JSON e repassar ao aplicativo. O modelo tornou-se problemático por diversos fatores. Um deles foi a limitação de *threads* devido ao esquema de licenciamento e de gerenciamento de memória; outro, o *overhead* causado quando trabalhamos com **milhares** de itens contidos na base. Tornou-se extremamente problemático manter a conexão ativa e consistente, e trafegar diversos *megabytes* de dados dessa maneira. Também houve uma gama de problemas ao gerar JSON a partir disso, por conta do acúmulo desnecessário na memória ao fazê-lo.

Modelo proposto

Propõe-se o modelo de pacotes para processamento e sincronização. Em primeiro lugar, o processamento no Protheus seria síncrono, mas, ainda assim, mais rápido do que assíncrono por conta do custo de *handshake* para conexão e do número limitado de *threads*. Ideia-se, portanto, gerar pacotes criptografados e compactados que mantém, internamente, os arquivos gerados para sincronização.

Garbage collector

Os pacotes gerados precisariam ser removidos após cada sincronização, que deve ser baseada no id da *thread* e da sessão. Não unicamente no id da *thread* porque esta é reaproveitada. Faz-se necessário escrever uma função de entrada que busque os arquivos temporários mais velhos que 24 horas e os remova.

Otimização nos *aliases*

Por padrão, o acesso de dados de uma TRB/banco usando `->` são custosos quando realizados múltiplas vezes. O valor não fica na memória de fácil acesso, e tem-se um *delay* considerável. O ideal é que, quando tem-se múltiplos usos do mesmo campo, crie-se uma variável que o armazene e utilize ela, com o valor armazenado na *heap*.

Otimização na concatenação de *strings*

Em AdvPL, *strings* são imutáveis, isto é, cada concatenação gera uma nova *string* com o valor anterior mais o valor atual, e mantém o antigo ainda acumulado na *stack*. O custo disso é exponencial e a complexidade é `O(n ^ 2)`. Conseguimos apresentar e escrever um modelo de complexidade linear (`O(1)`), de maneira que a complexidade **total** seja `O(n)`.

Compactação eficiente

O tamanho de um pacote gerado contendo 14.000.000 de caracteres é de aproximadamente 14 *megabytes* de tamanho. Usando compactação `TAR` para agrupar arquivos e `GZip` para comprimi-los, conseguimos reduzir o tamanho do arquivo em **99.5%**, deixando-o com somente `220kb`.

Testes

Nos resultados de nossos testes, conseguimos uma otimização de **até** 240 vezes. Um processo de sincronização da `STJ`, que levava, facilmente, mais de 1 hora, foi reduzido para um tempo entre 14 e 15 segundos. Abaixo, serão apresentados os testes e implementações.

O operador `+` também significa `+` custo

Cada operação de `+` cria uma *string* intermediária e a deixa acumulada na *stack*. Quando trabalhamos com concatenação de *strings* onde damos *append* a uma *string* relativamente grande, Isso tem um custo, literalmente, exponencial.

Usando `+`:

```
Function StrCat
  Local cStr := ''
  Local nI

  For nI := 1 To 1000000
    cStr += Str( nI )
  Next nI
```

```
  Return
```

Dessa maneira, temos algo similar a:

```
''
'1'
'12'
'123'
'1234'...
```

Estes valores estão sempre sendo acumulados e gerando novos valores. `+` continua sendo rápido para concatenações pequenas, mas esse não é, definitivamente, o caso. Podemos resolver alocando um espaço na memória e posicionando um ponteiro. Toda vez que formos concatenar, apenas jogamos nosso *buffer* a partir do ponteiro atual. Podemos simular esse modelo usando as *wrapper* functions de C, como `fseek`, `fwrite`, `fclose` e `fread`:

```
#include "fileio.ch"

Function StrCat
  // Ponteiro inicial
  Local nPointer := FCreate( 'buffer.bin' )
  // Memória atualmente alocada
  Local nMalloc
  Local nI
  Local cStr := ''

  For nI := 1 To 1000000
    // Ponteiro posicionado no final. Append na memória
    FWrite( nPointer, Str( nI ) )
  Next nI

  // Fechar o ponteiro atual
  FClose( nPointer )

  // Abrimos o buffer para utilização
  nPointer := FOpen( 'buffer.bin' )
  // Devemos pegar o tamanho alocado
  nMalloc := FSeek( nPointer, 0, FS_END )
  // Lemos por referência, passando o tamanho dos bytes
  FRead( nPointer, cStr, nMalloc )
  FClose( nPointer )

  // Deletar vestígios físicos
  If File( 'buffer.bin' )
    FErase( 'buffer.bin' )
  EndIf

  Return
```

As diferenças de performance são gritantes e exponenciais, podendo chegar a **horas**. Implementamos uma classe que simula esse processo de maneira mais simples, mas 4 vezes mais lenta que o modelo acima (ainda assim, centenas de vezes mais rápida que +), cujo código fonte segue ao fim deste documento.

Mostraremos dois exemplos de fontes e suas diferenças de performance. O primeiro, usa a metodologia padrão de AdvPL. O segundo, usa ponteiros na memória.

Modelo padrão

```
Static Function GetSyncPackage( oWS )
    Local cThread := ThreadId()
    Local xBuffer
    Local lHasError := .F.
    Local cFile
    Local nMalloc := 0
    Local xReg := ""
    Local nI
    Local cQuery
    Local cAliasQry
    Local xOpt
    Local xOpt2
    Local xResult := ''

    // Criar o arquivo e pegar seu ponteiro
    cFile := "pkg_" + AllTrim( Str( cThread + Randomize( 0, 10000 ) ) ) +
".json"
    cZip := cFile + ".tar.gz"
    xBuffer := FCreate( cFile )

    // Quando dá erro...
    If xBuffer == -1
        FClose( xBuffer )
        oWS:SetResponse( FError() )
        Return
    EndIf

    // Posicionar na ST9
    cAliasQry := GetNextAlias()
    cQuery := "SELECT STJ.TJ_ORDEM, TJ_CODBEM FROM " + RetSQLName( "STJ" ) + "
STJ WHERE "
    cQuery += "STJ.TJ_FILIAL = '" + xFilial( "STJ" ) + "' AND STJ.D_E_L_E_T_
<> '*'"
    cQuery := ChangeQuery( cQuery )

    dbUseArea( .T., "TOPCONN", TCGenQry( Nil, Nil, cQuery ), cAliasQry, .F.,
.T. )

    ConOut( Time() )
    While !EoF()
        // Acumular os valores no índice do arquivo na memória
        xOpt := ( cAliasQry )->TJ_ORDEM
        xOpt2 := ( cAliasQry )->TJ_CODBEM
```

```
        For nI := 1 To 20
            xResult += xOpt + xOpt2
        Next nI
        ( cAliasQry )->( dbSkip() )
    End
    ( cAliasQry )->( dbCloseArea() )
    ConOut( Time() )

    Return
```

Modelo com ponteiros

```
Static Function GetSyncPackage( oWS )
    Local cThread := ThreadId()
    Local xBuffer
    Local cFile
    Local nI
    Local cQuery
    Local cAliasQry
    Local xOpt
    Local cZip
    Local aFiles

    // Criar o arquivo e pegar seu ponteiro
    cFile := "pkg_" + AllTrim( Str( cThread + Randomize( 0, 10000 ) ) ) +
".json"
    cZip := cFile + ".tar.gz"
    xBuffer := FCreate( cFile )

    // Quando dá erro...
    If xBuffer == -1
        FClose( xBuffer )
        oWS:SetResponse( FError() )
        Return
    EndIf

    // Posicionar na ST9
    cAliasQry := GetNextAlias()
    cQuery := "SELECT STJ.TJ_ORDEM FROM " + RetSQLName( "STJ" ) + " STJ WHERE
"
    cQuery += "STJ.TJ_FILIAL = '" + xFilial( "STJ" ) + "' AND STJ.D_E_L_E_T_
<> '*'"
    cQuery := ChangeQuery( cQuery )

    // Abrir em modo compartilhado e readonly!
    dbUseArea( .T., "TOPCONN", TCGenQry( Nil, Nil, cQuery ), cAliasQry, .T.,
.T. )

    ConOut( Time() )
    While !EoF()
        // Acumular os valores no índice do arquivo na memória
```

```
            xOpt := ( cAliasQry )->TJ_ORDEM
            For nI := 1 To 20
                FWrite( xBuffer, xOpt )
            Next nI
            ( cAliasQry )->( dbSkip() )
        End
        ( cAliasQry )->( dbCloseArea() )
        ConOut( Time() )

        // Fechar o ponteiro e matar o arquivo de memória
        FClose( xBuffer )

        TarCompress( { cFile }, cFile + ".tar"  )
        GZCompress( cFile + ".tar", cZip )
        Download( oWS, cZip, cZip )

        // Tenta limpar o lixo deixado
        aFiles := { cFile, cFile + ".tar", cZip }
        For nI := 1 To Len( aFiles )
            If File( aFiles[ nI ] )
                FErase( aFiles[ nI ] )
            EndIf
        Next nI

        Return
```

Diferenças em segundos

Padrão	Otimizado
3600+	14

Links úteis

http://stackoverflow.com/questions/15400508/string-concatenation-complexity-in-c-and-java

StringBuilder.prw

```
#include "fileio.ch"
#include "protheus.ch"

//-------------------------------------------------------------------
/*/{Protheus.doc} Classe otimizada para concatenação de strings sem
```

```
o overhead de memória padrão do AdvPL, usando apenas processos
inerentes do próprio sistema operacional
@author Marcelo Camargo
@since 14/07/2016
/*/
//------------------------------------------------------------------
Class StringBuilder
    // ID da thread atual para identificar o processo
    Data cThreadId
    // ID único para evitar deadlock em processos rodando na mesma
    // thread
    Data cUUID
    // Localização física do arquivo após ser liberado da memória
    // virtual
    Data cFileName
    // Ponteiro para o arquivo virtual atual
    Data nPointer
    // Tamanho do arquivo em bytes
    Data nMalloc
    // Determina se a memória virtual já foi liberada e o conteúdo
    // foi armazenado fisicamente
    Data lBuilt

    Method New( cInit ) Constructor
    Method Append( cBuffer )
    Method Build()
    Method Read()
    Method Location()
    Method Dispose()
EndClass

//------------------------------------------------------------------
/*/{Protheus.doc} New
Constrói o buffer, podendo receber um valor inicial

@author Marcelo Camargo
@since 14/07/2016
@param cInit, O valor inicial do nosso buffer
@return StringBuilder
/*/
//------------------------------------------------------------------
Method New( cBuffer ) Class StringBuilder
    // Inicializamos os identificadores
    Self:lBuilt    := .F.
    Self:cThreadId := AllTrim( Str( ThreadID() ) )
    Self:cUUID     := AllTrim( Str( Randomize( 0, 100000 ) ) ) + ;
                      AllTrim( Str( Randomize( 0, 100000 ) ) )
    Self:cFileName := CurDir() + "/tmp/strbuilder" + Self:cThreadId +
Self:cUUID + ".bin"

    If .Not. ExistDir( CurDir() + "tmp" )
        MakeDir( CurDir() + "tmp ")
    EndIf

    // Criamos o arquivo virtualmente
```

```
    Self:nPointer := FCreate( Self:cFileName )

    If Self:nPointer == -1
        UserException( "StringBuilder: Falha ao alocar espaço na memória. Erro
" + AllTrim( Str( FError() ) ) )
        Return Self
    EndIf

    // Caso seja provido um valor inicial, o alocamos
    If cBuffer != Nil
        FWrite( Self:nPointer, cBuffer )
        Self:nMalloc := FSeek( Self:nPointer, 0, FS_END )
    Else
        Self:nMalloc := 0
    EndIf

    Return Self

//-------------------------------------------------------------------
/*/{Protheus.doc} Append
Adiciona uma string ao buffer, localizando o ponteiro final

@author Marcelo Camargo
@since 14/07/2016
@param cBuffer, string a adicionar
@return Nil
/*/
//-------------------------------------------------------------------
Method Append( cBuffer ) Class StringBuilder
    FWrite( Self:nPointer, cBuffer )
    Self:nMalloc := FSeek( Self:nPointer, 0, FS_END )
    Return

//-------------------------------------------------------------------
/*/{Protheus.doc} Build
Constrói os dados gerados em arquivo físico e liberada a memória
virtual da heap

@author Marcelo Camargo
@since 14/07/2016
@return Nil
/*/
//-------------------------------------------------------------------
Method Build() Class StringBuilder
    FClose( Self:nPointer )
    Self:lBuilt := .T.
    Return

//-------------------------------------------------------------------
/*/{Protheus.doc} Read
Lê, como string, o conteúdo da memória física. Precisa que Build tenha
sido anteriormente chamado

@author Marcelo Camargo
@since 14/07/2016
```

```
@return String
/*/
//-------------------------------------------------------------------
Method Read() Class StringBuilder
    Local cBigStr := ''

    If .Not. Self:lBuilt
        UserException( "StringBuilder: Build precisa ser invocado antes de
Read" )
    EndIf

    FRead( Self:nPointer, cBigStr, Self:nMalloc )

    Return cBigStr

//-------------------------------------------------------------------
/*/{Protheus.doc} FileName
Retorna o nome do arquivo físico para

@author Marcelo Camargo
@since 14/07/2016
@return String
/*/
//-------------------------------------------------------------------
Method Location() Class StringBuilder
    Return Self:cFileName

//-------------------------------------------------------------------
/*/{Protheus.doc} Dispose
Limpa todo o conteúdo da memória física e virtual. Deve ser chamado
quando as operações com strings terminarem. Trabalha como um GC
interno

@author Marcelo Camargo
@since 14/07/2016
@return Nil
/*/
//-------------------------------------------------------------------
Method Dispose() Class StringBuilder
    Self:nMalloc := 0

    // Caso não tenha sido construído, precisamos liberar o ponteiro
    If .Not. Self:lBuilt
        FClose( Self:nPointer )
    EndIf

    If File( Self:cFileName )
        FErase( Self:cFileName )
    EndIf

    Return
```