**Luiz Carlos Vieira Rodrigues**

# Análise de Transação Oracle em Linux

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Orientador
Prof. Joaquim Quinteiro Uchôa

Lavras
Minas Gerais - Brasil
2004

**Luiz Carlos Vieira Rodrigues**

# Análise de Transação Oracle em Linux

Monografia de Pós-Graduação "*Lato Sensu*" apresentada ao Departamento de Ciência da Computação para obtenção do título de Especialista em "Administração em Redes Linux"

Aprovada em *13 de Dezembro de 2004*

Prof. Fernando Cortez Sica

Prof. Gustavo Guimarães Parma

Prof. Joaquim Quinteiro Uchôa
(Orientador)

Lavras
Minas Gerais - Brasil

# Agradecimentos

Gostaria de deixar meu agradecimento especial ao meu orientador, professor Joaquim Uchôa, pelo profissionalismo com que me ajudou a conduzir este trabalho. E também a amiga Ângela Gontijo que tanto me ajudou com o LaTeX.

# Resumo

O objetivo do estudo que gerou esta monografia foi fazer uma análise de transações de banco de dados Oracle em sistema operacional Linux. A meta é que ao final do texto tenhamos uma análise em etapas e baseada em registros, os anais que poderemos apresentar para confirmar as análises.

*A minha esposa Helena e meus filhos Pedro, Mateus (*em memória*) e Júlia*

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

# O Oracle e O Sistema Operacional Linux

## 1.1 A Análise do Problema

É fato que, durante os últimos anos, ocorreu uma grande migração de serviços que antes rodavam em servidores com sistema operacional proprietário para servidores com sistema operacional Linux, sistema operacional este compatível com o POSIX e licenciado sob a Licença Pública GNU (GPL). A estabilidade, a credibilidade e o desempenho foram os principais fatores para a causa dessa migração. No entanto, mesmo no Linux, problemas podem ocorrer afetando o desempenho das aplicações nele inseridas.

O objetivo deste trabalho é, em primeiro lugar, analisar onde possa estar acontecendo uma contenção no banco de dados Oracle. A melhor maneira de se fazer isso é estudando o arquivo de saída do *trace* da transação que está apresentando o problema. Não importa em que ambiente esta transação esteja inserida (internet, cliente/servidor ou se no próprio servidor) e qual a complexidade que a envolve, sempre estarão presentes neste arquivo todos os elementos para a análise.

Para facilitar esta análise este autor desenvolveu um programa em C++ que lê o arquivo de saída do *trace* do Oracle e faz um diagnóstico com clareza dos problemas de performance que porventura estiverem acontecendo. Este programa mostrará a sequência de ações que o *kernel* do Oracle executou e a quantificação do tempo gasto em cada ação (CPU, rede, E/S, etc.). Com este diagnósitco em mãos, poder-se-á tomar ações específicas em cada caso para solucionar o problema.

Por outro lado o Oracle está sempre executando embaixo de um sistema operacional. É objetivo do próximo capítulo mostrar como pode ser feito um controle simples das principais áreas que afetam o Linux (rede, E/S, processador e memó-

ria). Este controle proporcionará uma análise de desempenho do servidor tanto em seu momento atual como sua evolução através do tempo, ou seja, poder-se-á comparar os gastos nestas áreas por períodos de tempo e saber se eles estão evoluindo ou não.

A abordagem do trabalho será então baseada em registros de comprovação, tanto no contexto do sistema operacional Linux quanto no contexto do banco de dados Oracle. Com isso pode-se dirimir dúvidas a respeito das causas dos problemas que estão afetando a transação.

# Capítulo 2

# Análise da Abordagem de Problemas no Servidor

Antes da análise de quaisquer problemas em serviços específicos no servidor é necessário saber se o servidor em si está em perfeitas condições de oferecer um serviço de qualidade aos seus usuários. Problemas de falta de recurso no servidor podem levar a um tempo de resposta ruim da aplicação. A sobrecarga dos recursos vêm, normalmente, de forma crescente, ou seja, antes uma aplicação que apresentava um tempo de resposta razoável começou apresentar contenções em momentos de pico. É preciso diferenciar, então, quando o problema está nos recursos do servidor e quando ele está localizado dentro do banco de dados. Este capítulo destina-se a armazenar o histórico de utilização dos recursos do servidor.

O núcleo do sistema operacional Linux que é executado no servidor é chamado de *kernel* do sistema operacional. O *kernel* do sistema operacional é usado para implementar a interface entre os processos Linux e todos os dispositivos físicos da máquina, tais como discos, memória e CPU.

Os processos dos usuários interagem com o Linux fazendo chamadas ao sistema. Essas chamadas são interceptadas pelo *kernel* do Linux e processadas de acordo com regras específicas. Podemos ver a diagramação destas chamadas de acordo com a figura 2.1.

Podemos, em uma visão ampla, dividir os problemas que podem acontecer em um servidor em 4 áreas : processamento, memória, rede e entrada/saida (E/S). A seguir será apresentada uma maneira simples de monitorar cada uma delas.

O ponto mais importante das abordagens de monitoramento apresentadas a seguir é possuir meios de armazená-las de forma que possamos consultá-las mais tarde passando filtros de acordo com as necessidades de cada caso. Hoje é praticamente impossível monitorar manualmente vários servidores ao mesmo tempo e

em tempo real, pois problemas podem acontecer simultaneamente em servidores críticos para organização, e para uma boa análise os registros de monitoramento precisam estar armazenados de alguma forma.

## 2.1 Monitoramento do Processador e da Memória

### 2.1.1 Identificando Problemas de Processador e Memória com o utilitário *vmstat*

Uma poderosa ferramenta para análise de problemas de memória e processador no Linux é o *vmstat*. O *vmstat* (MORGAN, 2002) provê estatísticas de utilização dos recursos do sistema. Ele tem como seu primeiro argumento um número que representa o intervalo de tempo em segundos em que serão mostradas as linhas com as informações.

Podemos visualizar um exemplo do comando *vmstat* no Linux na figura 2.2.

As métricas apresentadas por este utilitário são importantes para o monitoramento do servidor, principalmente:

**r** ***(run queue)*** A fila de execução mostra o número de processos prontos para executar, mas que não estão executando.

**b** ***(runnable-but-blocked queue)*** mostra o número de processos que estão prontos para executar mas que não estão executando por causa de um espera por transferência em bloco (geralmente uma operação de E/S de disco).

**si** ***(swap in)*** A operação de swap acontece quando o servidor está experimentando falta de memória RAM. Qualquer número diferente de zero nesta coluna indica atividade excessiva de leitura de RAM do disco.

**us** ***(user CPU)*** Porcentagem do processador que está servindo tarefas do usuário.

**sy** ***(system CPU)*** Quantidade do processador que está servindo tarefas do sistema.

E como interpretar as informações para saber quando o sistema está com contenção de processador? Eis uma regra simples para identificar um servidor com problemas de CPU dadas por (BURLESON, 2002): se o valor da fila de processos em execução (coluna *r* - abaixo do título *procs*) exceder o número de processadores no servidor as tarefas são forçadas a esperar para executar. Outra regra geral dada por (LOUKIDES, 2002): se constantemente a fila de execução (*run queue*) for quatro vezes maior que o número de processadores na máquina, deve-se considerar o aumento do número de processadores disponíveis. Há algumas maneiras para solucionar os problemas de sobrecarga no processador, a saber:

1. Adicionar mais processadores (CPU) ao servidor;

2. Fazer o balanceamento da carga das tarefas do sistema refazendo a programação dos horários dass tarefas mais pesadas em lote para horários onde a carga do processador esteja mais amena;

3. Ajustar as prioridades das tarefas existentes com o comando *renice*.

### 2.1.2 Identificando Alto Uso de Processador com *vmstat*

Pode-se facilmente identificar quando o processador do servidor está sendo altamente utilizado. Basta identificar e somar as colunas **us** (usuário) e **sy** (system): quando atingirem 100 porcento o processador está operando com sua capacidade total.

Vale dizer que a única métrica que identifica um gargalo no processador é quando o valor da fila de execução *r* excede o número de processadores no servidor (BURLESON, 2002).

### 2.1.3 Capturando a Saída do utilitário *vmstat*

É preciso, de alguma forma, criar um mecanismo para monitorar as estatísticas do comando *vmstat*. Sabemos que as contenções de recursos do sistemas podem ser rápidas e passageiras, sendo então muito fácil não estar monitorando quando um gargalo no sistema porventura acontecer.

Uma solução possível para guardar as informações do utilitário *vmstat* é registrar suas informações de performance dentro de uma tabela de um gerenciador de banco de dados, figura 2.3. O armazenamento no banco de dados facilita o grupamento e a totalização das informações por intervalos de tempo (como hora, dia, mês, ano) dos relatórios que serão gerados. Esta técnica além de funcionar para monitorar o servidor em questão, também servirá para capturar as informações de estatísticas de sistema dos outros servidores da rede, tendo assim um ponto centralizado de pesquisa e estatísticas. É interessante que o banco de dados que esteja armazenando estas informações seja um banco de dados separado dos bancos de dados de produção, isto para que as estatísticas possam ser consultadas mesmo em momento de contenção.

Na figura 2.4, um exemplo de script de (BURLESON, 2002), adaptado por este autor, para fazer o armazenamento da saída do utilitário ***vmstat*** em uma tabela do banco de dados Oracle.

Os Problemas de memória podem ser evidenciados pelo indicativo de operações de *page in*. Operações de *page in* causam problemas de performance pelo

simples fato de que o processo sofre um tempo de espera até que sua região de memória seja trazida de volta do disco para a RAM.

Coletando os dados remotamente, pode-se capturar uma visão completa da performance de todos os componentes do ambiente, e não somente do servidor de banco de dados. Isto é importante para os casos onde será preciso investigar as possíveis causas de baixa performance nos sistemas comerciais. Usando as informações do utilitário ***vmstat***, pode-se caminhar até o tempo que aconteceu o gargalo e examinar a carga do servidor naquele momento.

### 2.1.4 Gerando Relatórios com as Informações do *vmstat*

Podemos gerar um gama grande de relatórios com as informações disponíveis registradas dentro do banco de dados. No *Anexo II* encontram-se exemplos de como podem ser gerados estes relatórios.

1. **Relatórios de Exceção:** Relatórios gerados por períodos de tempo específicos onde parâmetros pré-definidos são passados.

2. **Relatórios Diários**

3. **Relatórios Horários**

4. **Relatórios de Prevenção:** Relatórios que podem predizer quando mais memória ou processador será requerido para um bom funcionamento do servidor.

As figuras a seguir mostram exemplos das saídas produzidas pelo relatório de exceção de paginação. Nestes relatórios a coluna *runq* representa a fila de execução, a coluna *pg_in* representa a o número de páginas que estão sendo trazidas para memória, a coluna *pg_out* representa o número de páginas que estão sendo levadas para o disco, a coluna *usr* representa a participação dos usuários na utilização do processador e a coluna *sys* representa a participação do sistema no gasto do processador. - figura 2.5, relatório de médias de consumo de recursos por dia da semana - figura 2.6 e, por último, relatório de médias de consumo de recursos por hora do dia - apresentado na figura 2.7.

As saídas produzidas pelos relatórios aqui apresentados são facilmente transportáveis para planilhas. A figura 2.8 mostra um gráfico produzido pela panilha do *OpenOffice* através dos dados da paginação (*page out*) do relatório apresentado na figura 2.7.

## 2.2 Monitoramento da entrada/saída

O monitoramento da entrada/saída é muito importante em um servidor de banco de dados. Do ponto de vista de entrada e saída podemos dizer o seguinte:

1. Acesso a disco é o maior responsável pelo incremento no tempo de resposta. Reduzir acesso a disco irá sempre resultar em melhor performance para a tarefa;

2. A criação de blocos de dados grandes permite acesso a mais dados em uma única operação de entrada/saída;

3. Existe uma importância muito grande na maneira em que a tabela é organizada fisicamente. Reorganizá-la frequentemente para deixar suas linhas na ordem em que as consultas por faixa de valores mais acontecem, reduz enormemente as chamadas *full scan table*( leitura de toda a tabela ).

### 2.2.1 Capturando Informações de Entrada e Saída com o Utilitário *iostat*

O utilitário *iostat* é usado para monitorar os dispositivos de E/S do sistema observando o tempo de atividade em relação à taxa média de transferência. O parâmetro de intervalo especifica a quantidade de tempo em segundos entre cada estatística mostrada. A primeira linha contém estatísticas coletadas desde a inicialização do sistema. Cada linha subsequente possui estatísticas coletadas durante o intervalo.

Com base nos mesmos padrões de captura de informações do utilitário *vmstat*, vamos fazer também a captura dos dados gerados pelo *iostat*, figura 2.9, e registrá-los em uma tabela . No script de (BURLESON, 2002) mostrado na figura 2.10, adaptado por este autor, um exemplo para fazer isso.

### 2.2.2 Gerando Relatórios com as Informações do *iostat*

No figura 2.11 um exemplo da saída do relatório de estatística da utilização (quantidade de *kbytes* lidos e escritos) dos discos agrupados por data. Todos os relatórios exemplificados neste texto podem ser modificados para atender as necessidades específicas que o problema em questão exige.

## 2.3 Monitoramento da Rede

O gerenciamento do tráfego da rede é uma tarefa extensa e penosa que envolve a captura e a análise de estatísticas. A tarefa básica a ser feita é : gerar *traces* com

*sniffers* (utilitários que capturam informações de toda a rede em modo promíscuo) e verificar sua utilização, estatísticas de transmissão, retransmissões etc.

O utilitário básico dos administradores de rede Linux é o *netstat*. Infelizmente este utilitário não tem a capacidade de produzir periodicamente registros instantâneos de utilização, como faz os outros dois utilitários já apresentados *vmstat* e *iostat*.

Por esta razão, vamos abordar o monitoramento do tráfego da rede através da utilização do esquema de estatísticas que é disponibilizado com o banco de dados Oracle.

Através da tabela de eventos disponibilizada pelo Oracle, e capturando fotografias instantâneas (*snapshots*) constantemente (por exemplo, de 30 em 30 segundos) pode-se selecionar a média dos tempos de espera em segundos e a quantidade destes tempos agrupados por hora. Na figura 2.12, (LAWSON, 2004) exemplifica como listar estes tempos.

Na figura 2.13 vemos que a saída produzida por este relatório é de grande ajuda para mostrar os tempos de espera quando a rede está sofrendo contenção por causa de tráfego pesado.

Este relatório oferece uma idéia sobre potenciais problemas que podem estar acontecendo na rede. É claro que a identificação aqui é somente sobre a quantidade de tempo de espera, e quaisquer investigações sobre a real causa do problema da rede terá que ser feita com um utilitário específico para tal.

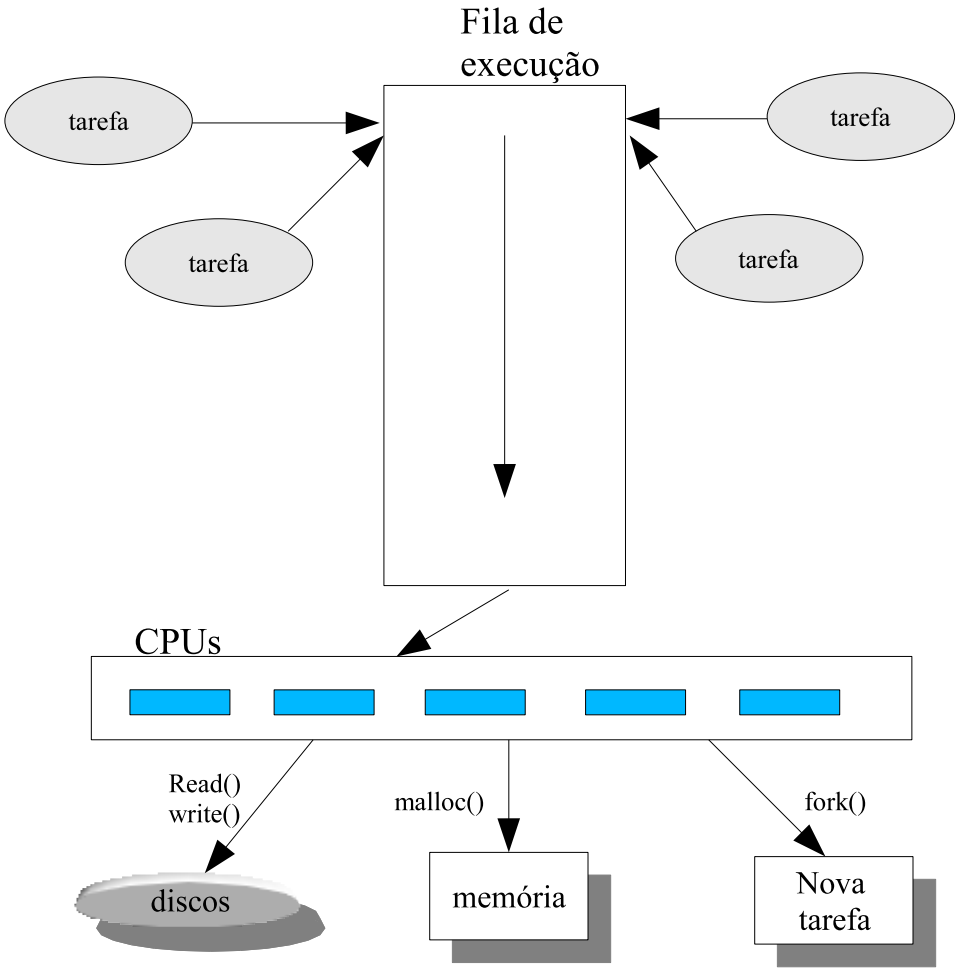


**Figura 2.1:** Fila de Execução

```
procs -----------memory---------- ---swap-- -----io---- --system-- ----cpu----
r  b   swpd   free   buff  cache   si   so    bi    bo   in    cs us sy id wa
0  0  44316  53804  48052 837400    1    1    65    41   33     7  2  1 97  0
0  0  44316  54788  48080 838152    0    0   150    39  440   394  1  1 98  0
3  0  44424  51372  46340 854404    0    0 17598   381  865  1189  6  6 87  0
0  1  44424  49936  46208 854144    0    0   659   234  618   677  9  4 87  0
```

**Figura 2.2:** Comando *vmstat*

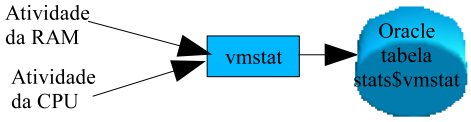


**Figura 2.3:** Armazenando a saída do *vmstat* no Banco de Dados

```
#!/bin/ksh

# First, we must set the environment . . . .
ORACLE_SID=bdprd02
export ORACLE_SID
ORACLE_HOME='cat/etc/oratab|grep\^$ORACLE_SID:|cut -f2 -d':''
export ORACLE_HOME PATH=$ORACLE_HOME/bin:$PATH
export PATH

SERVER_NAME='uname -a|awk '{print $2}'' typeset -u SERVER_NAME
export SERVER_NAME

# coleta a cada minuto (60 seconds) . . . .
SAMPLE_TIME=60

while true
  do
    vmstat ${SAMPLE_TIME} 2 > /tmp/msg$$

    cat /tmp/msg$$|sed 1,2d | awk \
    '{ printf("%s %s %s %s %s %s %s\n", $1, $7, $8, $13, $14, $15, $16) }'
    | while read RUNQUE PAGE_IN PAGE_OUT USER_CPU SYSTEM_CPU
             IDLE_CPU WAIT
        do
            $ORACLE_HOME/bin/sqlplus /nolog <<EOF
            connect perfstat/perfstat;
            insert into perfstat.stats\$vmstat (
                start_date, duration, server_name, runque_waits, page_in,
                page_out, user_cpu, system_cpu, idle_cpu, wait    )
            values  (sysdate, $SAMPLE_TIME, '$SERVER_NAME', $RUNQUE,
                    $PAGE_IN, $PAGE_OUT, $USER_CPU, $SYSTEM_CPU, $IDLE_CPU,
                    $WAIT );
            EXIT
            EOF
        done
  done

rm /tmp/msg$$
```

**Figura 2.4:** Capturando a Saída

page_in > 1
Pode indicar estrangulamento de memória
Sempre que Linux faz um page-in, a memória RAM do servidor
teve seu limite alcançado e páginas de swap serão usadas.

| Servidor | data | hora | runq | pg_in | pg_ot | usr | sys | idl |
|---|---|---|---|---|---|---|---|---|
| TRT18B | 04/11/17 | 22 | 0 | 5 | 0 | 1 | 0 | 100 |
| TRT18B | 04/11/18 | 00 | 0 | 2 | 0 | 1 | 0 | 99 |
| TRT18B | 04/11/18 | 04 | 0 | 4 | 0 | 10 | 2 | 88 |
| TRT18B | 04/11/18 | 08 | 0 | 4 | 1 | 1 | 1 | 98 |
| TRT18B | 04/11/18 | 09 | 0 | 8 | 0 | 3 | 1 | 96 |
| TRT18B | 04/11/18 | 10 | 0 | 13 | 0 | 6 | 3 | 91 |
| TRT18B | 04/11/18 | 11 | 0 | 18 | 0 | 4 | 2 | 95 |
| TRT18B | 04/11/18 | 12 | 0 | 15 | 2 | 5 | 3 | 93 |
| TRT18B | 04/11/18 | 13 | 0 | 25 | 1 | 5 | 2 | 94 |
| TRT18B | 04/11/18 | 14 | 0 | 18 | 7 | 4 | 3 | 93 |
| TRT18B | 04/11/18 | 15 | 0 | 10 | 13 | 6 | 5 | 89 |
| TRT18B | 04/11/18 | 16 | 0 | 7 | 15 | 6 | 5 | 90 |
| TRT18B | 04/11/18 | 17 | 0 | 10 | 3 | 4 | 2 | 94 |
| TRT18B | 04/11/18 | 18 | 0 | 14 | 1 | 4 | 1 | 96 |
| TRT18B | 04/11/18 | 19 | 0 | 12 | 5 | 23 | 5 | 72 |
| TRT18B | 04/11/18 | 20 | 0 | 21 | 0 | 4 | 1 | 95 |
| TRT18B | 04/11/18 | 21 | 0 | 14 | 0 | 0 | 0 | 100 |
| TRT18B | 04/11/18 | 23 | 0 | 2 | 0 | 4 | 1 | 95 |
| TRT18B | 04/11/19 | 00 | 0 | 6 | 0 | 2 | 0 | 98 |
| TRT18B | 04/11/19 | 02 | 0 | 2 | 0 | 0 | 0 | 100 |
| TRT18B | 04/11/19 | 04 | 0 | 2 | 82 | 15 | 2 | 83 |
| TRT18B | 04/11/19 | 08 | 0 | 4 | 0 | 1 | 1 | 98 |
| TRT18B | 04/11/19 | 09 | 0 | 5 | 1 | 3 | 1 | 96 |
| TRT18B | 04/11/19 | 10 | 0 | 10 | 7 | 5 | 2 | 93 |
| TRT18B | 04/11/19 | 11 | 0 | 9 | 6 | 4 | 2 | 94 |
| TRT18B | 04/11/19 | 12 | 0 | 6 | 4 | 4 | 2 | 94 |
| TRT18B | 04/11/19 | 13 | 0 | 4 | 4 | 4 | 2 | 94 |
| TRT18B | 04/11/19 | 14 | 0 | 5 | 4 | 4 | 2 | 94 |
| TRT18B | 04/11/19 | 15 | 0 | 5 | 6 | 4 | 2 | 93 |
| TRT18B | 04/11/19 | 16 | 0 | 5 | 6 | 5 | 4 | 91 |
| TRT18B | 04/11/19 | 17 | 0 | 5 | 4 | 5 | 4 | 91 |
| TRT18B | 04/11/19 | 18 | 0 | 5 | 4 | 5 | 3 | 92 |
| TRT18B | 04/11/19 | 19 | 1 | 4 | 3 | 7 | 3 | 90 |
| TRT18B | 04/11/19 | 20 | 0 | 4 | 3 | 6 | 3 | 91 |
| TRT18B | 04/11/19 | 21 | 0 | 3 | 3 | 5 | 3 | 93 |

**Figura 2.5:** Relatório de Paginação

| data | runq | pg_in | pg_ot | usr | sys | idl |
|---|---|---|---|---|---|---|
| Domingo | 0 | 0 | 1 | 1 | 0 | 99 |
| Segunda | 0 | 1 | 2 | 3 | 1 | 96 |
| Terça | 0 | 1 | 1 | 2 | 1 | 97 |
| Quarta | 0 | 1 | 1 | 2 | 1 | 97 |
| Quinta | 0 | 1 | 1 | 2 | 1 | 97 |
| Sexta | 0 | 3 | 4 | 4 | 2 | 94 |
| Sábado | 0 | 1 | 1 | 1 | 1 | 98 |

**Figura 2.6:** Relatório de diário

| Data | Hora | runq | pg_in | pg_ot | usr | sys | idl |
|---|---|---|---|---|---|---|---|
| 2004/11/09 | 00 | 0 | 1 | 1 | 2 | 1 | 98 |
| 2004/11/09 | 01 | 0 | 1 | 2 | 2 | 1 | 98 |
| 2004/11/09 | 02 | 0 | 1 | 1 | 2 | 1 | 97 |
| 2004/11/09 | 03 | 0 | 1 | 1 | 2 | 1 | 98 |
| 2004/11/09 | 04 | 0 | 1 | 1 | 3 | 1 | 97 |
| 2004/11/09 | 05 | 0 | 1 | 1 | 2 | 1 | 98 |
| 2004/11/09 | 06 | 0 | 1 | 1 | 2 | 1 | 98 |
| 2004/11/09 | 07 | 0 | 1 | 1 | 2 | 1 | 98 |
| 2004/11/09 | 08 | 0 | 2 | 2 | 3 | 1 | 97 |
| 2004/11/09 | 09 | 0 | 1 | 1 | 5 | 2 | 95 |
| 2004/11/09 | 10 | 0 | 1 | 2 | 4 | 1 | 96 |
| 2004/11/09 | 11 | 0 | 1 | 1 | 4 | 1 | 96 |
| 2004/11/09 | 12 | 0 | 1 | 2 | 4 | 1 | 96 |
| 2004/11/09 | 13 | 0 | 1 | 1 | 4 | 1 | 96 |
| 2004/11/09 | 14 | 0 | 3 | 9 | 7 | 3 | 93 |
| 2004/11/09 | 15 | 0 | 5 | 12 | 7 | 3 | 93 |
| 2004/11/09 | 16 | 0 | 6 | 7 | 5 | 2 | 95 |
| 2004/11/09 | 17 | 0 | 4 | 1 | 3 | 1 | 97 |
| 2004/11/09 | 18 | 0 | 5 | 1 | 3 | 1 | 97 |
| 2004/11/09 | 19 | 1 | 2 | 2 | 9 | 2 | 91 |

**Figura 2.7:** Média Paginação/Hora

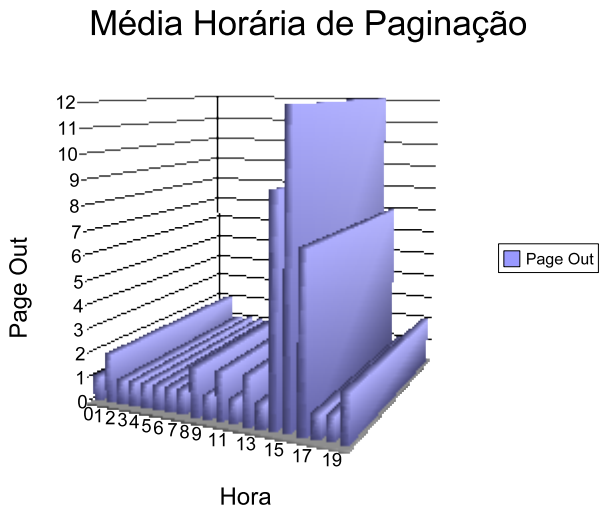


**Figura 2.8:** Média Horária de Paginação

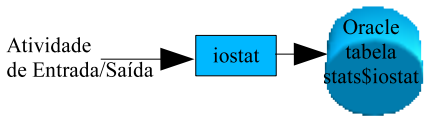


**Figura 2.9:** Armazenando a saída do iostat no Banco de Dados

```
#!/bin/ksh

while true
do
   iostat -x  300 1|\
      sed 1,6d|\
      awk  '{ printf("%s %s %s\n", $1, $8, $9) }' |\
   while read HDISK VMSTAT_IO_R VMSTAT_IO_W
   do

      echo $HDISK
      echo $VMSTAT_IO_R
      echo $VMSTAT_IO_W

      sqlplus perfstat/perfstat <<!
      insert into
         perfstat.stats\$iostat
      values
        (sysdate, 300, '$HDISK',
         nvl(to_number('$VMSTAT_IO_R'),0),
         nvl(to_number('$VMSTAT_IO_W'),0) );
      exit
!

   done
   sleep 300

done
```

**Figura 2.10:** Capturando E/S

| Disco | Data | SUM_KB_READ | SUM_KB_WRITE |
|---|---|---|---|
| /dev/sda | 2004/11/09 | 31,492 | 12,021 |
| | 2004/11/10 | 67,161 | 26,390 |
| /dev/sda1 | 2004/11/09 | 0 | 0 |
| | 2004/11/10 | 0 | 0 |
| /dev/sda5 | 2004/11/09 | 79 | 108 |
| | 2004/11/10 | 168 | 229 |
| /dev/sda6 | 2004/11/09 | 66 | 428 |
| | 2004/11/10 | 139 | 943 |
| /dev/sda7 | 2004/11/09 | 31,334 | 11,040 |
| | 2004/11/10 | 66,820 | 24,237 |
| /dev/sda8 | 2004/11/09 | 14 | 444 |
| | 2004/11/10 | 34 | 980 |

**Figura 2.11:** Quantidade de E/S por Disco

```
sqlplus /nolog <<EOF
connect perfstat/perfstat
set pages 999;

column mydate heading 'Yr. Mo Dy Hr' format a13;
column event format a30;
column waits format 999,999;
column secs_waited format 999,999,999;
column avg_wait_secs format 99,999;

select
        to_char(snap_time,'yyyy-mm-dd HH24') mydate,
        e.event,
        e.total_waits - nvl (b.total_waits,0) waits,
        ((e.time_waited_micro - nvl (b.time_waited_micro,0))/100) /
        nvl ((e.total_waits - nvl(b.total_waits,0)),.01) avg_wait_secs
from
        stats\$system_event b,
        stats\$system_event e,
        stats\$snapshot sn
where
        e.snap_id = sn.snap_id
  and   b.snap_id = e.snap_id - 1
  and   b.event = e.event
  and   e.event like 'SQL*Net%'
  and   e.total_waits - b.total_waits > 100
  and   e.time_waited_micro - b.time_waited_micro > 100
;
/
EXIT
EOF
```

**Figura 2.12:** Listando os Tempos de Espera da Rede

```
Yr. Mo Dy Hr  EVENT                          WAITS AVG_WAIT_SECS
------------- ------------------------------ -------- -------------
2004-11-08 01 SQL*Net message from client       4,390        92,937
2004-11-08 03 SQL*Net message to client         2,996             0
2004-11-08 04 SQL*Net break/reset to client       264             0
2004-11-08 05 SQL*Net message from client       3,038        39,565
2004-11-08 07 SQL*Net break/reset to client       264             0
2004-11-08 08 SQL*Net message from client     167,689        12,077
2004-11-08 09 SQL*Net message to dblink           271             0
2004-11-08 09 SQL*Net more data from client       143           270
2004-11-08 16 SQL*Net more data to client       3,664             1
2004-11-08 17 SQL*Net message to dblink           223             0
2004-11-08 21 SQL*Net break/reset to client       264             1
2004-11-08 22 SQL*Net message to client        12,712             0
2004-11-08 23 SQL*Net message to client        14,149             0
2004-11-09 04 SQL*Net message from client     146,619           423
2004-11-09 05 SQL*Net break/reset to client       264             1
2004-11-09 08 SQL*Net message to client       121,241             0
2004-11-09 13 SQL*Net more data to client       2,984             1
2004-11-09 13 SQL*Net message from dblink         773            12
2004-11-09 13 SQL*Net break/reset to client       296            14
2004-11-09 18 SQL*Net message to client       285,884             0
2004-11-09 19 SQL*Net more data to client       3,512             0
2004-11-09 19 SQL*Net message from client     222,212        16,921
```

**Figura 2.13:** Tempos de Espera de Rede

# Capítulo 3

# Análise da Abordagem de Problemas no Oracle

## 3.1 Não Concordância Com as Principais Causas do Problema

Técnicos geralmente são questionados a respeito de causas de lentidão no funcionamento do sistema como um todo, ou de parte do sistema, ou de uma função do sistema. Ou então são questionados por esses mesmos itens mas somente em determinados intervalos de tempo do dia. Se várias pessoas forem questionadas sobre as causas do sistema estar lento, provavelmente a discordância entre elas será quase total.

De acordo com (MILLSAP; HOLT, 2003), para se obter sucesso na análise dos problemas que podem ocorrer dentro do banco de dados é essencial termos uma boa compreensão de como o banco mostra as informações a respeito de sua performance. Através do arquivo de saída do *trace* podemos compreender como *kernel* do Oracle interage com o sistema operacional. É, de fato, o sistema operacional que aloca os recursos que o núcleo do Oracle necessita para sua existência. E é o próprio sistema operacional que provê as estatísticas de tempo que o Oracle usa para descrever sua própria performance.

Normalmente o analista responsável pelo desempenho do banco de dados Oracle é o primeiro a ser questionado nos itens descritos no parágrafo anterior. Muitas destas vezes ele se questiona quanto à correção da parametrização das opções de inicialização ou de funcionamento do gerenciador de banco de dados. Às vezes faz experiências de todo o tipo quanto a aumento de área de buffers, aumento de área de classificação, diminuição do valor de um item aqui ou aumento do valor

de um item acolá; mas ao final ele vê um frustante resultado de todos seus ajustes: *o usuário não notou diferença alguma no tempo de resposta*.

A experiência deste autor na área em questão é que os ajustes feitos nos parâmetros do banco de dados refletem nele como um todo e são de sintonia fina - não vão melhorar enormemente uma aplicação da noite para o dia. E ainda, a maioria dos problemas que o usuário final enfrenta são problemas que será resolvido fora do banco de dados. E no caso do problema estar centrado no banco de dados, será resolvido com análise da aplicação, ou talvez, mais especificamente, da transação em si.

A compreensão da estrutura do banco de dados e dos parâmetros de inicialização e funcionamento do banco de dados é essencial para qualquer analista que queira fazer ajustes no banco, e por certo precisará fazê-lo para cada banco que for criado. Mas na análise dos problemas com aplicação, a verdade é que precisamos analisar o problema de uma visão bem ampla, partindo do servidor como um todo, e descendo em níveis da aplicação, até chegar na transação em si.

De acordo com (LAWSON, 2004) as causas de problemas com performance no banco de dados se encaixam nas seguintes categorias:

- Projeto da aplicação e sua interação com o banco de dados;
- Projeto do banco de dados;
- Indexação;
- Parâmetros de inicialização do banco de dados;
- Problemas causados pela concorrência com programas em lote;
- Hardware (incluindo problemas de rede).

E qual a melhor maneira de resolver estes problemas? Mais uma vez citando (MILLSAP; HOLT, 2003):

> *"I believe that the best way to begin the study of Oracle operational data is with a tour of Oracle's extended SQL trace output. SQL trace output is unsurpassed as an educational and diagnostic aid, because it presents a linear sequential recorded history of what the Oracle Kernel does in response to an application's demands upon the database*".

```
sqlplus /nolog<<EOF
connect perfstat/perfstat
alter session set max_dump_file_size=unlimited;
alter session set timed_statistics=true;
alter session set events '10046 trace name context forever, level 12';
alter session set tracefile_identifier='TRACE_EXE_LAVRAS';
select * from spddba01.saj18t003 where nome_parte like 'LUIZ CARLOS VIEIRA%';
exit
EOF
```

**Figura 3.1:** Ativando o *Trace*

## 3.2 A Análise da Saída do *Trace* do Oracle

Baseando-se nos princípios na sessão anterior vistos, vamos, a partir deste momento, analisar a saída do *trace* do Oracle e propor uma solução (um programa em C++) que consegue ler esta saída e montar uma estrutura de dados que possibilitará diversos tipos de consultas.

Os comandos SQL da figura 3.1 mostram como ativar o *trace* no Oracle.

Com o exemplo dos comandos da figura 3.1 pode-se visualizar o arquivo de *trace* do Oracle na figura 3.2. Para achar onde o Oracle colocou o arquivo de saída do *trace*, basta ir no diretório de saída de *traces* do usuário que é informado no arquivo de parâmetros de inicialização. No nosso exemplo o arquivo estará com o nome: bdprd02_ora_26246_TRACE_EXE_LAVRAS.trc.

### 3.2.1 Como o Oracle Faz Sua Medição

Saber como o Oracle faz sua medição de performance não é uma tarefa difícil. O sistema operacional Linux possui uma ferramenta chamada ***strace***. O ***strace*** é simples de usar. Por exemplo, na figura 3.3 verifica-se diretamente o que o núcleo do Oracle está fazendo.

Neste exemplo, pode se ver o que o processo do Linux de número 1825 está fazendo. Aplicando o utilitário *strace* no momento em que o Oracle está fazendo o *trace* de alguma transação, pode-se ver claramente que ele se utiliza da função ***gettimeofday()*** para mostrar as estatísticas de tempo no arquivo de saída que é gerado. O tempo transcorrido entre suas ações é computado como no pseudo-código abaixo:

```
t0 = gettimeofday;   # marca o tempo imediatamente antes de fazer algo
  faz algo;
t1 = gettimeofday;   # marca o tempo imediateamente depois de fazê-lo
t = t1 - t0;         # mostra t, que é o tempo transcorrido da ação
```

Em sistemas operacionais compatíveis com o POSIX, o núcleo do Oracle obtém as informações da CPU através da função ***getrusage()***.

```
EXEC #1:c=0,e=14330,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=1074894149642183
WAIT #1: nam='SQL*Net message to client' ela= 4 p1=1650815232 p2=1 p3=0
WAIT #1: nam='SQL*Net message from client' ela= 398 p1=1650815232 p2=1 p3=0
=====================
PARSING IN CURSOR #2 len=76 dep=0 uid=869 oct=3 lid=869 tim=1074894149646560
                                                hv=2723982836 ad='55702a40'
select * from spddba01.saj18t003 where nome_parte like 'LUIZ CARLOS VIEIRA%'
END OF STMT
PARSE #2:c=10000,e=3730,p=0,cr=0,cu=0,mis=1,r=0,dep=0,og=1,tim=1074894149646553
BINDS #2:
EXEC #2:c=0,e=147,p=0,cr=0,cu=0,mis=0,r=0,dep=0,og=1,tim=1074894149646809
WAIT #2: nam='SQL*Net message to client' ela= 3 p1=1650815232 p2=1 p3=0
WAIT #2: nam='db file sequential read' ela= 30 p1=7 p2=27658 p3=1
WAIT #2: nam='db file sequential read' ela= 20 p1=7 p2=30652 p3=1
WAIT #2: nam='db file sequential read' ela= 9293 p1=7 p2=29481 p3=1
FETCH #2:c=0,e=9695,p=3,cr=4,cu=0,mis=0,r=1,dep=0,og=1,tim=1074894149656602
WAIT #2: nam='SQL*Net message from client' ela= 72110 p1=1650815232 p2=1 p3=0
WAIT #2: nam='db file sequential read' ela= 23 p1=6 p2=108898 p3=1
WAIT #2: nam='SQL*Net message to client' ela= 3 p1=1650815232 p2=1 p3=0
WAIT #2: nam='db file sequential read' ela= 18 p1=6 p2=111778 p3=1
WAIT #2: nam='db file sequential read' ela= 18 p1=6 p2=114871 p3=1
WAIT #2: nam='db file sequential read' ela= 17 p1=6 p2=106932 p3=1
WAIT #2: nam='db file sequential read' ela= 16 p1=6 p2=108561 p3=1
WAIT #2: nam='db file sequential read' ela= 18 p1=6 p2=109093 p3=1
WAIT #2: nam='db file sequential read' ela= 19 p1=6 p2=111817 p3=1
WAIT #2: nam='db file sequential read' ela= 19 p1=6 p2=106768 p3=1
WAIT #2: nam='db file sequential read' ela= 18 p1=6 p2=114073 p3=1
WAIT #2: nam='db file sequential read' ela= 20 p1=6 p2=115127 p3=1
WAIT #2: nam='db file sequential read' ela= 20 p1=6 p2=106918 p3=1
WAIT #2: nam='db file sequential read' ela= 18 p1=6 p2=107835 p3=1
WAIT #2: nam='db file sequential read' ela= 19 p1=6 p2=110380 p3=1
FETCH #2:c=0,e=913,p=13,cr=14,cu=0,mis=0,r=13,dep=0,og=1,tim=1074894149729790
WAIT #2: nam='SQL*Net message from client' ela= 47221 p1=1650815232 p2=1 p3=0
XCTEND rlbk=0, rd_only=1
STAT #2 id=1 cnt=14 pid=0 pos=1 obj=49370 op='TABLE ACCESS BY INDEX ROWID OBJ#(49370)
                                                (cr=18 pr=16 pw=0 time=9706 us)'
STAT #2 id=2 cnt=14 pid=1 pos=1 obj=49373 op='INDEX RANGE SCAN OBJ#(49373)
                                                (cr=4 pr=3 pw=0 time=9641 us)'
```

**Figura 3.2:** Arquivo de Saída do *Trace*

```
$ strace - p 1825

open("/proc/stat", O_RDONLY)            = 15
fstat64(15, {st_mode=S_IFREG|0444, st_size=0, ...}) = 0
old_mmap(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x42656000
read(15, "cpu  2069215 8812 35868505 82705"..., 4096) = 756
read(15, "", 4096)                      = 0
close(15)                               = 0
munmap(0x42656000, 4096)                = 0
gettimeofday({1100204429, 718582}, {240, 0}) = 0
gettimeofday({1100204429, 718865}, {240, 0}) = 0
gettimeofday({1100204429, 719181}, {240, 0}) = 0
```

**Figura 3.3:** Saída do comando *strace*

Na figura 3.4 têm-se um pseudo-código de como o Oracle faz a medição de seus tempos de execução e de consumação de CPU:

Para ativar o *trace* do em uma sessão do Oracle é necessário emitir os seguintes comandos :

```
alter session set max_dump_file_size=unlimited;
alter session set timed_statistics=true;
```

```
procedure dbcall {
    e0 = gettimeofday;          # marca o tempo
    c0 = getrusage;             # obtém estatística de uso
    ....                        # executa uma chamada ao banco
    c1 = getrusage;             # obtém estatística de uso
    e1 = gettimeofday;          # marca o tempo
    e = e1 - e0;                # tempo transc. chamada ao banco
    c = (c1.utime + c1.stime)
      - (c0.utime + c0.stime);  # tempo total de CPU da chamada ao banco
    print(TRC,....);            # imprime um PARSE, EXEC, FETCH, etc.
}

procedure wevent {
    ela0 = gettimeofday;        # marca o tempo
    ...                         # executa o tempo de espera aqui
    ela1 = gettimeofday;        # marca o tempo
    ela = ela1 - ela0;          # ela é o tempo transc. do tempo de espera
    print(TRC,"WAIT....");      # imprime a linha de WAIT
}
```

**Figura 3.4:** Marcação de Tempo no Oracle

```
alter session set events '10046 trace name context forever, level 12';
```

Uma das razões do sucesso da alta performance do gerenciador de banco de dados Oracle é o detalhamento e a transparência em que o tempo de resposta é mostrado. Com os dados da saída do *trace* e ainda as várias visões fixas que vêm junto com o banco, é possível saber o porquê dos tempos de resposta de uma aplicação.

## 3.3 Os Elementos da Saída do Trace

Um resumo dos principais elementos da saída do arquivo de trace e de seus significados será descrito a seguir.

**Número do Cursor** Cada linha emitida na saída do *trace* corresponde a uma ação executada pelo *kernel* do Oracle. Cada linha usa a sequência de caracteres *#ID* para identificar a qual cursor a ação executada está se referenciando.

**Identificação da Sessão e do Tempo** A linha começando com o padrão *** indica o tempo obtido do sistema. A identificação de uma sessão no oracle é feita pelas variáveis de número de sessão mais um número de identificação dentro da sessão. No *trace*, a linha contendo o padrão ***SESSION ID:(*m.n*), sendo *m* o número da sessão e *n* um número serial dentro da sessão.

**A Linha Com a Identificação de *PARSING IN CURSOR #ID*** Contém informação a respeito do cursor. O texto entre esta linha e a linha que contém o texto *END OF STMT* corresponde ao cursor da aplicação.

Cada linha com a identificação do cursor contém as informações sobre o cursor, como:

1. ***len*** O tamanho do texto do SQL;
2. ***dep*** O nível de recursividade do cursor. Sendo o nível *dep = n + 1* um filho de algum cursor de nível *dep = n*. Várias ações podem motivar uma recursividade do SQL, incluindo chamadas que requerem informações do dicionário do Oracle, disparos de gatilhos;
3. ***uid*** O usuário que está compilando o SQL;
4. ***tim*** Tim é um valor expresso em microsegundos ( $1\ \mu_s = 0.000.001$ segundos ). O *tim* é o exato valor do dado pela função *gettimeofday()*;
5. ***database calls*** Uma chamada ao banco de dados é uma subrotina do *kernel* do Oracle. Cada chamada ao banco de dados contém as seguintes informações:

   ***c*** Tempo total consumido pela CPU para processar a chamada ao banco de dados;

   ***e*** Quantidade de tempo transcorrido durante a chamada ao banco de dados;

   ***p*** Número de blocos obtidos através de leitura solicitada via sistema operacional. Pode ser leitura no disco ou no cache;

   ***cr*** Número de blocos obtidos pela chamada em *modo de consistência*. Uma chamada nesta modalidade pode motivar leituras nos blocos de *undo*, que são armazenados nos segmentos de *rollback*;

   ***cu*** Número de blocos obtidos pela chamada em *modo corrente*. Uma chamada nesta modalidade lê diretamente o conteúdo de um bloco sem se preocupar se ele está no meio de uma atualização ainda sem confirmação de gravação;

   ***mis*** Número de leituras não encontradas no buffer ( *cache miss* );

   ***r*** Número de linhas retornadas pela chamada ao banco de dados;

   ***dep*** O nível de recursividade do cursor;

   ***og*** O tipo de otimização em efeito no tempo da chamada ao banco de dados. O Oracle usa os valores mostrados na tabela 3.1;

   ***tim*** Mesmo significado descrito anteriormente.

**Eventos de Espera (*wait events*)** Um evento de espera no Oracle é mostrado para cada tipo de espera encontrado quando a opção de *trace* estiver ativa. Na tabela 3.2 pode-se visualizar os eventos, e seus significados, que mais ocorrem

| *Valor* | *Tipo de Otimização* |
|---|---|
| 1 | ALL ROWS |
| 2 | FIRST ROWS |
| 3 | RULE |
| 4 | CHOOSE |

**Tabela 3.1:** Tipos de Otimização do Oracle.

no banco de dados de acordo com (LAWSON, 2004). Para cada evento de espera temos informações do tipo:

1. ***nam*** O nome que é dado ao evento de espera, sendo um nome relativo a cada porção do *kernel* do Oracle responsável por esta parte do tempo de resposta;
2. ***ela*** O tempo transcorrido na duração do evento;
3. ***p1, p2, p3*** Cada um destes três parâmetros têm significados próprios dentro de cada evento de espera. Para sabermos seu significado precisamos emitir o seguinte SQL:

```
Select Name, Parameter1, Parameter2, Parameter3
  from V$event_name
order by name
```

Os tempos de espera podem aparecer no arquivo de saída do *trace* antes da chamada ao banco de dados que os motivou. Isso acontece por que o *kernel* do Oracle mostra as linhas do *trace* somente quando o evento em questão estiver terminado. Então, se uma operação de leitura no banco requer três operações de leitura no sistema operacional, os três eventos de espera destas leituras feitas pelo sistema operacional serão mostradas antes da informação total da chamada ao banco de dados.

**Marcação de Final de Transação** Quando uma transação SQL fizer uma confirmação da transação ( *commit* ) ou uma não confirmação da transação (*rollback*) o Oracle emitirá no arquivo de saída do *trace* a seguinte linha:

```
XCTEND rlbk=0, rd_only=0
```

Sendo que *rlbk* terá o valor 1 se a transação fizer um *rollback* e *rdonly* terá o valor 1 se a transação não alterar nada no banco de dados.

Tabela 3.2: Eventos de Espera do Oracle

| Evento de Espera | Descrição |
|---|---|
| enqueue | O processo está esperando pelo liberamento de algum recurso |
| library cache pin | O processo quer examinar algum objeto e certificar que ninguém consiguirá alterá-lo neste tempo. |
| library cache load lock | O processo está esperando pela oportunidade de carregar um objeto ou parte de um objeto na biblioteca de cache. |
| latch free | O processo está esperando por um *latch* (termo usado quando um grande número de processos estão competindo pelo acesso a algum objeto interno) |
| buffer busy waits | O processo quer acesso a um bloco de dados que não está na memória, mas que outro processo já requisitou uma E/S para trazê-lo. |
| control file sequential read | O processo está esperando para acessar blocos do arquivo de controle. |
| control file parallel write | O processo está esperando que termine sua requisição de E/S em paralelo para os arquivos de controle. |
| log buffer space | O processo está esperando por espaço livre no buffer do log. (geralmente acontece quando as aplicações produzem *redo* (imagem anterior à alteração de algum registro do banco) mais rápido que o processo de descarregamento dos buffers de log em disco conseguem consumí-los. |
| log file sequentia read | O processo está esperando para trazer blocos do arquivo de log ativo para memória. |
| log file parallel write | O processo está esperando para escrever blocos para um grupo de arquivos de log ativos. |
| log file sync | O processo está esperando que processo responsável pela escrita de log termine de escrever os buffers para o disco. |
| db file scattered read | O processo emitiu uma requisição de E/S para ler uma série de blocos contíguos de um arquivo de dados para a área de buffer e está esperando a operação completar. Tipicamente nas consultas *(*full scan) de índices e tabelas. |

Continua na pág. seguinte

**Tabela 3.2 – continuação da pág. anterior**

| Evento de Espera | Descrição |
|---|---|
| db file sequential read | O processo emitiu uma requisição de E/S para ler um bloco de um arquivo de dados para a área de buffer e está esperando a operação completar. |
| db file parallel read | O processo emitiu múltiplas requisições de E/S em paralelo para ler blocos de arquivos de dados para a área de buffer e está esperando que todas as operações se completem. |
| db file parallel write | O processo emitiu múltiplas requisições de descarregamento dos buffers para o disco está esperando que todas as operações completem. |
| direct path read, direct path write | O processo emitiu requisições assíncronas de E/S que não passam pelos buffers e está esperando que todas as operações completem. É um evento típico de operações de classificação. |

### 3.3.1 Medição do Tempo de Resposta no Oracle

O *kernel* do Oracle emite duas categorias de tempo no arquivo de *trace*:

1. Tempo consumido nas chamadas ao banco de dados;

2. Tempo consumido entre as chamadas ao banco de dados.

O tempo de resposta total (***e***) é o tempo gasto com CPU (***c***) somado ao tempo consumido nos eventos de espera. No arquivo de saída do *trace* temos o tempo consumido durante toda a chamada ao banco de dados reportado no campo ***e***. De acordo com (MILLSAP; HOLT, 2003), formalmente pode-se escrever:

$$e \approx c + \sum_{chbd} ela$$

O *kernel* do Oracle também emite os tempos de espera que ocorrem entre as chamadas ao banco de dados. Exemplos de eventos de espera incluem:

```
SQL*Net message from client
SQL*Net message to client
single-task message
pmon timer
smon timer
```

De acordo com (MILLSAP; HOLT, 2003) existe uma relação entre relação entre os campos do arquivo de *trace* ***c, e*** e ***ela***:

$$R = \sum e + \sum_{chbd} ela$$

$$\approx [\sum c + \sum_{nachbd} ela] + \sum_{entrechbd} ela$$

$$\approx \sum c + \sum ela$$

Deduz-se que o somatório dos tempos consumidos entre as chamadas ao banco de dados e o tempo consumido dentro do banco de dados

$$\sum_{nachbd} ela + \sum_{entrechbd} ela$$

compõem a totalização dos tempos de espera (*elapsed*). Somando este resultado ao somatório do tempo gasto com a CPU (***c***) temos o tempo total levado pela aplicação para completar seu trabalho.

Um complicador que surge com o somatório que está proposto acima é a dupla contagem relacionada com os SQL recursivos. Como dito anteriormente a cada cursor temos nível de profundidade associado, mostrado como ***dep=n*** (sendo n = 0,1,2 ...). Uma chamada ao banco de dados com ***dep=n+1*** é o filho recursivo da primeira subsequente ***dep=n*** chamada ao banco de dados listado no arquivo de *trace*. A figura 3.5 mostra este relacionamento.

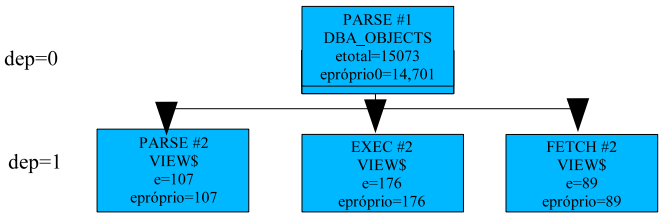


**Figura 3.5:** Níveis de Contagem

Para cada chamada ao banco de dados, os campos descritos no arquivo de *trace* ***c, e, p , cr*** e ***cu***, possuem a estatística de consumo de toda sua descendência incluída nele.Então o valor das estatísticas relacionadas a um nó ***dep=k*** é o valor emitido para aquela chamada menos o somatório dos valores emitidos em sua descendência ***dep=k+1***. O que (MILLSAP; HOLT, 2003) representa com a fórmula:

$$S = S_k - \sum_{filhos} S_{k+1},$$

onde $S_i$ é o valor da estatística no conjunto *(c, e, p, cr, cu)* emitido pelo *kernel* do Oracle para o nível de profundidade *i*.

O tempo de resposta total para o arquivo de saída emitido pelo *trace* ficaria assim:

$$R = \sum_{dep=0} c + \sum ela$$

No ***Anexo I***, este autor apresenta a sua solução para ler o arquivo de saída do *trace* e carregar os elementos significativos da análise dos dados, através de um programa escrito em C++. Após carregar os elementos siginificativos do arquivo em uma estrutura de dados, possibilitará consultas diversas, como por exemplo, percentuais de participação de cada evento no tempo total, contabilização individual dos tempos de cada cursor, texto dos cursores, etc.

Por certo os analistas que estão acostumados a procurar por explicações para o tempo de resposta da aplicação em um mar de literatura, vão acostumar a fazê-lo de uma maneira muito mais simples e direta com este programa.

A representação da estrutura de dados que armazena as informações, pode ser vista na figura 3.6: uma lista encadeada de cursores representados pelos retângulos maiores, cada qual com um ponteiro para uma lista encadeada de eventos de espera, representada pelos retângulos menores, e vetores de contabilização dentro de cada cursor.

A figura 3.7 reproduz, a título de exemplo, o resultado da execução do programa tendo com entrada um arquivo de *trace* do Oracle. Os eventos de espera que foram encontrados estão listados juntamente com suas participações no tempo total:

A figura 3.8 reproduz, a títuto de exemplo, o resultado parcial da execução do programa quando é mostrado todos os cursores e a participação de seus tempos de espera no total geral.

Para todo arquivo de saída de *trace* analisado, sempre haverá alguma diferença entre o que o *kernel* do Oracle contabiliza e o tempo total transcorrido. Esta diferença está representada na figura 3.7 como um evento “Tempo não Contabilizado“. As causas que levam a essa diferença não serão discutidas neste texto, mas podem ser encontradas em (MILLSAP; HOLT, 2003), página 150.

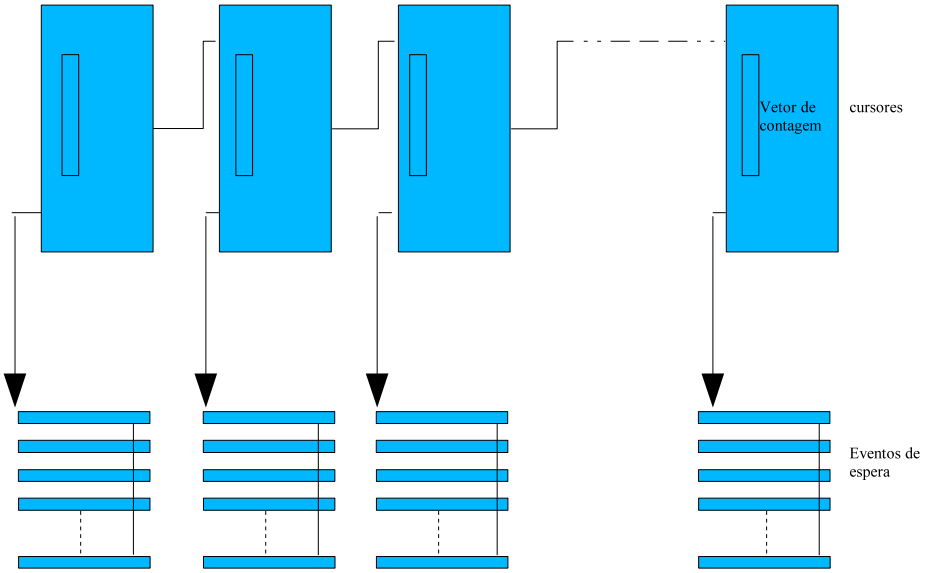


**Figura 3.6:** Estrutura de Armazenamento

| Eventos | Pct |
|---|---|
| SQL*Net message to client | 0.00% |
| SQL*Net message from client | 4.39% |
| rdbms ipc reply | 0.02% |
| enqueue | 8.79% |
| PX Deq: Join ACK | 0.07% |
| PX Deq: Parse Reply | 0.54% |
| PX Deq: Execute Reply | 70.26% |
| PX qref latch | 0.68% |
| PX Deq: Signal ACK | 3.89% |
| db file sequential read | 4.29% |
| Cpu | 4.76% |
| Não Contabilizado | 2.32% |
| Total | 100.00% |

**Figura 3.7:** Participação do Tempos de Espera no Total

```
Cursor n. 3 (  select ts#,file#,block#,nvl(bobj#,0),nvl(tab#,0),intcols,nvl(clucols,0),
audit$,flags,pctfree$,pctused$,initrans,maxtrans,rowcnt,blkcnt,empcnt,avgspc,chncnt,
avgrln,analyzetime, samplesize,cols,property,nvl(degree,1),nvl(instances,1),avgspc_flb,
flbcnt,kernelcols,nvl(trigflag, 0),nvl(spare1,0),nvl(spare2,0),spare4,spare6
from tab$ where obj#=:1 )

db file sequential read                                                  0.64%
```

**Figura 3.8:** Totalização por Cursor

# Conclusão

Após a implementação de todos os métodos sugeridos por este texto, verificou-se maior segurança na análise dos problemas que ocorrem no dia a dia de um setor de informática com vários servidores instalados, cada qual executando serviços específicos. E, melhor que isso, uma análise que pode ser demonstrada através das informações que ficaram registradas por estes métodos.

Com o registro das informações de performance dos servidores em tabelas do banco de dados, pôde-se fazer um controle sobre causas dos problemas diversos que ocorrem nestes servidores, e para cada caso, uma ação específica pôde ser tomada.

Quanto aos problemas que estão localizados dentro do banco de dados, pôde-se verificar o que exatamente estava afetando o tempo de resposta da aplicação. E isso foi feito com um programa em C++, ou do contrário o analista teria que passar horas e horas analisando uma saída nada agradável aos olhos humanos. E ainda, não é nada fácil ficar verificando linha a linha de código de *trace*, principalmente quando este arquivo alcança alguns milhares de linhas.

Após verificar o que aqui está proposto, pode-se garantir que muito tempo será economizado, principalmente o longo tempo de leituras em manuais e entendimentos de centenas de views que o Oracle disponibiliza para acompanhamento de performance e que às vezes deixa o analista responsável por um diagnóstico um tanto quanto confuso.

Assim, o objetivo do trabalho - ter uma primeira análise dos problemas que podem estar afetando o tempo de resposta da aplicação - foi plenamente alcançado, mesmo que essa análise leve a pesquisas mais aprofundadas em outros aspectos do banco de dados ou do sistema operacional.

# Referências Bibliográficas

BURLESON, B. K. *Oracle9i Unix Administration Handbook.* [S.l.]: McGraw-Hill/Osborne, 2002.

LAWSON, C. *The Art and Science of Oracle Performance Tuning*. [S.l.]: Apress, 2004.

LOUKIDES, G. D. M. . M. *System Performance Tuning*. [S.l.]: O'Reilly, 2002.

MILLSAP, C.; HOLT, J. *Optimizing Oracle Performance*. [S.l.]: O´Reilly, 2003.

MORGAN, A. G. *The Linux PAM System Administrators' Guide; Draft v0.76*. [S.l.]: Linux-PAM, 2002. URL: `http://www.us.kernel.org/pub/linux/libs/pam/`.

SICA J. Q. UCHôA, L. E. S. F. C. *Administração de Redes Linux*. [S.l.]: Ufla/Faepe, 2003.

# Apêndice A

# Anexo I

## A.1 Criando Uma Estrutura de Dados Através da Saída do *Trace*

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <iomanip>
#include <string.h>
#include <stdio.h>
using namespace std;


// Programa que tem como entrada o arquivo produzido pelo trace do Oracle
// e que monta uma estrutura de listas encadeadas com as informacoes obtidas
// deste arquivo

// Autor : Luiz Carlos Vieira Rodrigues      Dezembro, 2004

const int num_niveis = 20;
struct accounts {
        float cpu;
        float ela; };

class WaitsNode {   // Estrutura de ligação com tipos de espera
public:
  WaitsNode() {
    proxwait=0;
    tipowait="";
    valorwait=0;
  }
  WaitsNode(string p_waittxt,float p_vlrwait, WaitsNode * p_npw=0) {
    tipowait = p_waittxt;
    proxwait = p_npw;
    valorwait = p_vlrwait;
  }
  string tipowait;
  float  valorwait;
  WaitsNode * proxwait;
};

class LnWaits {        // Classe que manipula a estrutura de espera
public:
  LnWaits() {
    inicio = fim = 0;
  }
  ~LnWaits() {
     for (WaitsNode *p; !vazia(); ) {
         p = inicio->proxwait;
```

```
        delete inicio;
        inicio = p;
     }
  }
  bool ExWait(string p_waittxt, float p_vlrwait=0) {
     WaitsNode * p = inicio;
     for (p = inicio; p != 0 && !(p->tipowait == p_waittxt); p = p->proxwait);
     if (p !=0 ) {
        p->valorwait=p->valorwait+p_vlrwait;
     }
     return p != 0;
  }
  float TotalWait() {
     float total = 0;
     WaitsNode * p = inicio;
     for (p = inicio; p != fim; p = p->proxwait)
           total = total + p->valorwait;
     return total;
  }

  int vazia() {
     return inicio == 0;
  }
  void AdInicio(string p_waittxt, float p_vlrwait) {
     WaitsNode * PtrAux;
     inicio = new WaitsNode(p_waittxt,p_vlrwait,inicio);
  }
  float PrintWaitNo(float p_total=0) {
     float total_parcial=0;
     char sz[101]="";
     WaitsNode * p = inicio;
     for (p = inicio; p != fim; p = p->proxwait) {
           total_parcial=total_parcial+p->valorwait;
           cout.setf(ios::left,ios::adjustfield);
           cout << setw(50) << p->tipowait ;
           cout.setf(ios::right,ios::adjustfield);
           cout << fixed << setprecision(2) << setw(10) <<
           (p->valorwait/p_total)*100 << "%" << endl;
     }
     return total_parcial;
  }
  void AccTotal(LnWaits * p_acumula_wait=0) {
     LnWaits * q = p_acumula_wait;
     WaitsNode * p = inicio;
     for (p = inicio; p != 0; p = p->proxwait)
        if ( q->ExWait(p->tipowait,p->valorwait) == false ) {
              q->AdInicio(p->tipowait,p->valorwait);
        }
  }

private:
  WaitsNode *inicio, *fim;
};

class Accdml {
   // Classe para manipulação da estrutura de cursores e suas totalizações
  public:
      Accdml() {
        pw = new LnWaits();
        proxcursor=0;
      }
      Accdml(string p_cursortxt=" ", int p_numcursor=1, Accdml * p_npw=0) {
        pw = new LnWaits();
        for (int i=0;i<num_niveis;i++) {
                niveis_account[i].cpu=0;
                niveis_account[i].ela=0;
        }
        cursortxt = p_cursortxt;
        proxcursor = p_npw;
        numcursor = p_numcursor;
      }

      bool ExWait(string p_waittxt, float p_vlrwait) {
        return pw->ExWait(p_waittxt, p_vlrwait);
      }
      void AdInicio(string p_waittxt, float p_vlrwait) {
```

```
      pw->AdInicio(p_waittxt, p_vlrwait);
    }
    float TotalWait() {
       return pw->TotalWait();
    }
    int RetNumCursor() {
       return numcursor;
    }
    Accdml *  ExCursor(int p_numcursor, Accdml * inicio_cursores) {
      Accdml * p = inicio_cursores;
      for (p = inicio_cursores;
           p != 0 && !(p->numcursor == p_numcursor);
           p = p->proxcursor);
        return p;
    }
    Accdml * AdCursor(string p_cursortxt,
                      int p_numcursor,
                      Accdml * inicio_cursores) {
      Accdml * p = inicio_cursores;
      p = new Accdml(p_cursortxt, p_numcursor, p);
      return p;
    }
    void AcCpu(float p_vlrcpu, int p_nivel) {
       niveis_account[p_nivel].cpu=niveis_account[p_nivel].cpu + p_vlrcpu;
    }
    void AcEla(float p_vlrela, int p_nivel) {
       niveis_account[p_nivel].ela=niveis_account[p_nivel].ela + p_vlrela;
    }
    void AtCursor(string p_cursortxt) {
      cursortxt = cursortxt + p_cursortxt;
    }
    float TotalCpu(Accdml * p_inicio_cursores) {
      Accdml * p = p_inicio_cursores;
      float totalcpu=0;
      for (p = p_inicio_cursores; p != 0 ; p = p->proxcursor) {
         totalcpu=totalcpu+p->niveis_account[0].cpu;
      }
      return totalcpu;
    }
    float TotalEla(Accdml * p_inicio_cursores) {
      Accdml * p = p_inicio_cursores;
      float totalela=0;
      for (p = p_inicio_cursores; p != 0 ; p = p->proxcursor) {
         totalela=totalela+p->niveis_account[0].ela;
      }
      return totalela;
    }
    void WaitsNos(Accdml * p_inicio_cursores,float p_total=0) {
      Accdml * p = p_inicio_cursores;
      for (p = p_inicio_cursores; p != 0 ; p = p->proxcursor) {
         cout << "Cursor n. " << p->numcursor
              << " ( " << p->cursortxt << " )" << endl;
         cout << endl;
         p->pw->PrintWaitNo(p_total);
         cout << endl;
      }
    }
    float TotalWaits(Accdml * p_inicio_cursores,
                     LnWaits * p_inicio_total_waits) {
      Accdml * p = p_inicio_cursores;
      LnWaits * q = p_inicio_total_waits;
      for (p = p_inicio_cursores; p != 0 ; p = p->proxcursor) {
         p->pw->AccTotal(q);
      }
    }
    float PrintWait(float total=0,LnWaits * p_inicio_total_waits=0) {
      LnWaits * p = p_inicio_total_waits;
      return p->PrintWaitNo(total);
   }

private:
    LnWaits * pw;
    accounts niveis_account [num_niveis];
    string cursortxt;
    int numcursor;
    Accdml * proxcursor;
```

```
};


void setstring (char szOut [], char szIn []);
float retvalor (char strin [], char p, char stop);
float retvalor_wait (char strin [], int pos);
int ret_pos (string s, string sub);
string retmot_wait (string strin , int pos);

int main () {


   Accdml * idml;
   Accdml * idml_aux;
   Accdml *inicio_cursores=0, *fim_cursores=0;
   idml = new Accdml("",1);
   LnWaits * totalwaits;
   totalwaits = new LnWaits();
   float total_parcial = 0;
   LnWaits *inicio_total_waits=totalwaits;
   inicio_cursores=idml;
   fim_cursores=idml;
   int linecount=0;
   float cpu=0;
   float ela=0;
   float total=0;
   int v_numcursor=0;
   string v_strcursor;
   int j=0;
   int k=0;
   char aux[15]="";
   char buffer[1024];
   char sz[101]="";
   char valores[7][20] = {  "PARSE #",
                            "EXEC #",
                            "FETCH #",
                            "UNMAP #",
                            "SORT UNMAP #",
                            "PARSING IN CURSOR #",
                            "WAIT #" };
   char * pch;
   ifstream filein ("filein.txt");
   ofstream fileout ("fileout.txt");
   if (! filein.is_open()) { cout << "erro io open"; exit(1); }
   while (! filein.eof() )
   {
    filein.getline (buffer,1024);
    for ( int i=0; i<=6 ;i++ )
      if ( strstr (buffer,valores[i]) ) {
        v_numcursor=static_cast<int>(
                        retvalor_wait (buffer,ret_pos(buffer,valores[i])+
        ret_pos(valores[i],"#")+1));
        if ( i!=6 ) {
                v_nivelcursor=static_cast<int>(
                        retvalor_wait (buffer,ret_pos(buffer,"dep=")+4));
        }
        idml_aux=idml->ExCursor(v_numcursor,inicio_cursores);
        if ( idml_aux == 0 ) {
           idml_aux=idml->AdCursor(" ",v_numcursor,inicio_cursores);
           inicio_cursores=idml_aux;
        }
        idml=idml_aux;
        if ( i==5 ) {   // aramazenamento do cursor
              v_strcursor="";
              filein.getline (buffer,1024);
              v_strcursor=buffer;
              pch=strstr(buffer,"END OF STMT");
              while (( pch!=NULL) && (! filein.eof() )) {
                   filein.getline (buffer,1024);
                   v_strcursor=v_strcursor+buffer;
                   pch = strstr (buffer,"END OF STMT");
              }
              idml->AtCursor(v_strcursor);
        } else if ( i==6 ) { // linha de wait - não possui profundade nem accounts
                if (idml->ExWait( retmot_wait (buffer,ret_pos(buffer,"nam=")+5) ,
                retvalor_wait (buffer, ret_pos(buffer,"ela=")+5) ) == false) {
```

```
                idml->AdInicio( retmot_wait (buffer,ret_pos(buffer,"nam=")+5) ,
                retvalor_wait (buffer, ret_pos(buffer,"ela=")+5) );
                  }
        }
        else {
          idml->AcCpu( retvalor (buffer, 'c',',') , v_nivelcursor );
          idml->AcEla( retvalor (buffer, 'e',',') , v_nivelcursor );
        }
      }
   }

// Mostrando os totais gerais

 total=idml->TotalCpu(inicio_cursores)+idml->TotalEla(inicio_cursores);
 cout.setf(ios::left,ios::adjustfield);
 cout << setw(50) << "Eventos ";
 cout.setf(ios::right,ios::adjustfield);
 cout << setw(10) << "        Pct" << endl;
 cout << setfill('-') << setw(62) << " " << setfill(' ') << endl;
 idml->TotalWaits(inicio_cursores,inicio_total_waits);
 total_parcial=idml->PrintWait(total,inicio_total_waits);

 cout.setf(ios::left,ios::adjustfield);
 cout << setw(50) << "Cpu ";
 cout.setf(ios::right,ios::adjustfield);
 cout << fixed << setprecision(2) << setw(10)
      << (idml->TotalCpu(inicio_cursores)/total)*100 << "%" << endl;
 cout.setf(ios::left,ios::adjustfield);
 cout.setf(ios::left,ios::adjustfield);
 cout << setw(50) << "Não Contabilizado" ;
 cout.setf(ios::right,ios::adjustfield);
 cout << fixed << setprecision(2) << setw(10)
 << ((total-(total_parcial+idml->TotalCpu(inicio_cursores)))/total)*100
 << "%" << endl;
 cout.setf(ios::left,ios::adjustfield);
 cout << setw(50) <<  "Total ";
 cout.setf(ios::right,ios::adjustfield);
 cout << setw(11) << "100.00%" << endl;
 cout << endl;
 cout << "Participação na Totalizacao por Cursor" << endl;
 cout << endl;

// Mostrando a participacao na totalizacao por cursor

 idml->WaitsNos(inicio_cursores,total);


   return 0;
}

void setstring (char szOut [], char szIn [])
{
  int n=0;
  do {
    szOut[n] = szIn[n];
  } while (szIn[n++] != '\0');
}

float retvalor (char strin [], char p, char stop)
{   int j=0;
    int k=0;
    char aux[15];
    setstring (aux," ");
    while ( strin[j]!=p ) j++;
    while ( strin[j+2]!=stop ) {
        aux[k] = strin[j+2];
        j++;
        k++;
    }
    aux[k]='\0';
    return atof (aux);
}

int ret_pos (string s, string sub)
{
  return s.find(sub,0);
```

```
}

float retvalor_wait (char strin [], int pos)
{
    int j=pos;
    int k=0;
    char aux[15];
    setstring (aux," ");
    while ( strin[j]=='0' or
            strin[j]=='1' or
            strin[j]=='2' or
            strin[j]=='3' or
            strin[j]=='4' or
            strin[j]=='5' or
            strin[j]=='6' or
            strin[j]=='7' or
            strin[j]=='8' or
            strin[j]=='9' ) {
        aux[k] = strin[j];
        j++;
        k++;
    }
    aux[k]='\0';
    return atof (aux);
}

string retmot_wait (string strin, int pos)
{
    return strin.substr(pos, ret_pos(strin,"' ")-pos);
}
```

# Apêndice B

# Anexo II

## B.1 Gerando Relatórios de Exceção

```
sqlplus /nolog <<EOF
connect perfstat/perfstat;
set lines 80;
set pages 999;
set feedback off;
set verify off;

column server_name    heading 'Servidor'
column data    heading 'data       hora' format a20
column c2      heading runq   format 999
column c3      heading pg_in  format 999
column c4      heading pg_ot  format 999
column c5      heading usr    format 999
column c6      heading sys    format 999
column c7      heading idl    format 999
column c8      heading wt     format 999

ttitle 'run queue > 2|Pode indicar Cpu sobrecarregada|
Quando a fila de execução excede o número de Cpus|
no servidor, tarefas estarão esperando por serviço.';

select
 server_name,
 to_char(start_date,'YY/MM/DD    HH24') data,
 avg(runque_waits)       c2,
 avg(page_in)            c3,
 avg(page_out)           c4,
 avg(user_cpu)           c5,
 avg(system_cpu)         c6,
 avg(idle_cpu)           c7
from
perfstat.stats\$vmstat
WHERE
runque_waits > 2
and start_date > sysdate-3
group by
 server_name,
 to_char(start_date,'YY/MM/DD    HH24')
ORDER BY
 server_name,
 to_char(start_date,'YY/MM/DD    HH24')
;

ttitle 'page_in > 1|Pode indicar estrangulamento de memória|
Sempre que Linux faz um page-in, a memória RAM do servidor |
```

```
teve seu limite alcançado e as páginas de swap serão usadas.';

select
 server_name,
 to_char(start_date,'YY/MM/DD    HH24') data,
 avg(runque_waits)       c2,
 avg(page_in)            c3,
 avg(page_out)           c4,
 avg(user_cpu)           c5,
 avg(system_cpu)         c6,
 avg(idle_cpu)           c7
from
perfstat.stats\$vmstat
WHERE
page_in > 1
and start_date > sysdate-3
group by
 server_name,
 to_char(start_date,'YY/MM/DD    HH24')
ORDER BY
 server_name,
 to_char(start_date,'YY/MM/DD    HH24')
;

ttitle 'user+system CPU > 70%|Indica períodos com CPU com máxima utilização.|
Períodos de 100% utilização são somente uma |
preocupação quando a fila de execução excede o número de CPUs no servidor.';
select
 server_name,
 to_char(start_date,'YY/MM/DD    HH24') data,
 avg(runque_waits)       c2,
 avg(page_in)            c3,
 avg(page_out)           c4,
 avg(user_cpu)           c5,
 avg(system_cpu)         c6,
 avg(idle_cpu)           c7
from
perfstat.stats\$vmstat
WHERE
(user_cpu + system_cpu) > 70
and start_date > sysdate-2
group by
 server_name,
 to_char(start_date,'YY/MM/DD    HH24')
ORDER BY
 server_name,
 to_char(start_date,'YY/MM/DD    HH24')
;
/
EXIT
EOF
```

## B.2 Gerando Relatório Diário

```
sqlplus /nolog <<EOF
connect perfstat/perfstat;
set pages 9999;
set linesize 80;
set feedback off;
set verify off;

column my_date heading 'date' format a20
column c2      heading runq   format 999
column c3      heading pg_in  format 999
column c4      heading pg_ot  format 999
column c5      heading usr    format 999
column c6      heading sys    format 999
column c7      heading idl    format 999
column c8      heading wt     format 999
```

```
select
 case to_char(start_date,'d')
     when '1' then 'Domingo'
     when '2' then 'Segunda'
     when '3' then 'Terça'
     when '4' then 'Quarta'
     when '5' then 'Quinta'
     when '6' then 'Sexta'
     when '7' then 'Sábado'
 end my_date,
 avg(runque_waits)       c2,
 avg(page_in)            c3,
 avg(page_out)           c4,
avg(user_cpu + system_cpu)          c5,
 avg(system_cpu)         c6,
 avg(idle_cpu)           c7
 -- avg(wait_cpu)           c8
from
   stats\$vmstat
group  BY
 to_char(start_date,'d')
order by
 to_char(start_date,'d')
;
/
EXIT
EOF
```

## B.3 Gerando Relatório Horário

```
sqlplus /nolog <<EOF
connect perfstat/perfstat;
set pages 9999;
set linesize 80;
set feedback off;
set verify off;

column data    heading 'Data       Hora'
column c2      heading runq   format 999
column c3      heading pg_in  format 999
column c4      heading pg_ot  format 999
column c5      heading usr    format 999
column c6      heading sys    format 999
column c7      heading idl    format 999
column c8      heading wt     format 999

select
 to_char(start_date,'yyyy/mm/dd   HH24') data,
 avg(runque_waits)       c2,
 avg(page_in)            c3,
 avg(page_out)           c4,
 avg(user_cpu + system_cpu)           c5,
 avg(system_cpu)         c6,
 avg(idle_cpu)           c7
from
   stats\$vmstat
where to_char(start_date,'ddmmyyyy')=to_char(sysdate,'ddmmyyyy')
group  BY
 to_char(start_date,'yyyy/mm/dd   HH24')
order by
 to_char(start_date,'yyyy/mm/dd   HH24')
;
/
EXIT
EOF
```

# Apêndice C

# Anexo III

## C.1 Ferramentas de Auxílio Rápido

Incluí este anexo como um auxílio rápido para visualização do que está ocorrendo com o Oracle, quando somente um usuário está reclamando e quando o problema é generalizado.

### C.1.1 Quando Uma Sessão Está Travada

Quando algum usuário está com sua sessão travada por alguma razão, podemos utilizar as visões do Oracle para determinar em tempo real onde está a causa do problema. Um SQL para isto é apresentado abaixo:

```
col sid format 999990
col seq# format 999,990
col event format a26
select sid, seq#, event, state, seconds_in_wait seconds
from v$session_wait
where sid=xx
```

O resultado do SQL acima pode ser visualizado na figura C.1, quando um usuário está tentando alterar alguma linha de alguma tabela que outro usuário já está alterando e ainda não finalizou a transação.

### C.1.2 Quando Todos Reclamam do Tempo de Resposta

Se por acaso o problema é geral e não é só uma pessoa que enfrenta problemas com o tempo de resposta ruim, e se o sistema permitir a execução de um SQL, pode-se fazer a seguinte consulta ao banco:

```
    SID     SEQ# EVENT                     STATE                SECONDS
------- -------- ------------------------- ------------------- ----------
     29       68 enqueue                   WAITING              1945684
```

**Figura C.1:** Sessão Travada

```
EVENT                                                                                SESSIONS
-------------------------------------------------------------------------------- ----------
rdbms ipc message                                                                         7
SQL*Net message from client                                                               3
pmon timer                                                                                1
smon timer                                                                                1
```

**Figura C.2:** Tempos de Espera

```
break on report compute
sum of sessions on report
select event,count(*) sessions
  from v$session_wait
 where state='WAITING'
 group by event
order by 2 desc; ,
```

para saber como estão os tempos de espera no Oracle, que produz o resultado da figura C.2.