*Estudantes:*

**Arthur Fernandes Minduca de Sousa –** [**fernandes.arthur@gmail.com**](mailto:fernandes.arthur@gmail.com)

**Carlos Eduardo Buarque Cruz Pimentel –** [**cebcp@cin.ufpe.br**](mailto:cebcp@cin.ufpe.br)

**Carlos Henrique Maciel Sobral Timóteo –** [**chmst@cin.ufpe.br**](mailto:chmst@cin.ufpe.br)

**Karina Rodrigues Pereira –** [**krp@cin.ufpe.br**](mailto:krp@cin.ufpe.br)

*Vínculo:*

**Mestrado Acadêmico**

*Disciplina:*

**Avaliação de Desempenho de Sistemas**

*Professor:*

**Paulo Maciel**

*Atividade:*

**Resolução da 3ª Lista de Exercícios**

# Qual é a principal diferença entre *tracing*, *profiling block counting*?

O *tracing* é uma estratégia de medição de eventos baseada na *event-driven*, uma vez que se preocupa em contar as ocorrências do evento. Entretanto, o *tracing* procura armazenar mais informações, registrando o estado do sistema, de modo que o evento possa ser identificado. Por exemplo, se o simulador de redes ns-2 estiver sendo usado para calcular a vazão de uma rede sem fio, ao usar o *tracing* como estratégia de medição, o simulador não registrará apenas o número de vezes que pacotes foram transmitidos e que pacotes foram recebidos, ele poderá registrar juntamente a cada um dos eventos uma série de outras informações como: o momento em que os eventos ocorreram dentro do tempo da simulação, identificadores dos pacotes, etc...

Devido à quantidade de informações que o *tracing* agrega ou pode agregar à contagem do evento que ele está medindo e também do tempo que ele gasta para acessar arquivos de disco ou porções da memória para armazenar essas informações, ele é uma estratégia de medição que potencialmente tem um grande *overhead*. Como conseqüência, esse *overhead* gera muita perturbação à toda medição.

O *profiling* também é uma estratégia de medição de eventos, mas que, por sua vez, procura dar uma visão geral do sistema, caracterizando seu comportamento global através de sumários das métricas de desempenho. Com isso, pode-se verificar que proporções do tempo total da execução do sistema foram gastas com determinados estados, identificando, assim, gargalos no seu funcionamento. Um exemplo é saber, em um programa de computador desenvolvido em C, quais de suas funções estão levando mais tempo para serem executadas e, com isso, providenciar a otimização delas.

Um tipo específico dessa técnica é o *block counting* ou contagem de bloco. Nesse caso, o sistema é dividido em blocos de entrada e saída únicas e em que ações são realizadas de forma ininterrupta, uma vez que se iniciou a execução daquele bloco. A cada bloco, adiciona-se um *overhead* com o que é necessário para realizar a contagem de quantas vezes o sistema passou por lá. Ao final, histogramas são gerados com os dados dos contadores. Vale notar que esse *overhead*, mesmo que relativamente pequeno, causa grande perturbação à medição.

Diante dessas definições, pode-se dizer que, em suma, o *tracing* está preocupado com a quantidade de vezes que um evento específico de um sistema ocorre e em trazer mais informações a respeito dele e a respeito do sistema no momento da ocorrência. Já o *profiling*, em sua denominação *block counting*, preocupa-se em fazer um perfil geral de um sistema de modo que suas diversas partes e a contribuição de cada uma delas nele sejam analisadas. Além disso, o *profiling* do tipo *block counting*, em relação ao *tracing*, causa muito menos *overhead* tanto em termos de aquisição de dados quanto de armazenamento deles, já que só realiza uma contagem. Conseqüentemente, a perturbação do sistema também é menor.

# O que diferencia *profiling block counting* e *sampling profiling.*

O *sampling profiling* gera os sumários das métricas de desempenho de um sistema através de amostras dele. Para colher essas amostras, interrupções são feitas no funcionamento do sistema e informações acerca do seu estado são colhidas. As interrupções podem ser feitas baseadas em intervalos de tempo ou em outro tipo de parâmetro.

Um exemplo de uso *sampling profiling* é quando se quer saber informações acerca de um programa de computador e interrupções na sua execução são realizadas. Nesses momentos de interrupção, o contador de programo, PC, é analisado e uma amostra do que está nele é colhida.

Diante dessa abordagem, o *sampling profiling* gera um sumário com informações que são estimativas estatísticas, diferentemente do *profiling block counting* que tem como saída informações exatas acerca do perfil geral do sistema. Outros aspectos que diferenciam uma estratégia da outra são o *overhead* e, consequentemente, a perturbação: o *sampling profiling* causa *overhead* apenas quando gera uma interrupção, o que torna a perturbação no sistema aleatoriamente distribuída e, se bem projetado o número de amostras/interrupções, essa perturbação é consideravelmente menor do que a resultante do *profiling block counting*.

# Qual é a freqüência de amostragem necessária para garantir que sejam coletas 10 amostras de cada uma das 12 sub-rotinas de um programa que é executado por 30 s? Assuma que cada sub-rotina seja executada durante pelo menos 5% do tempo de execução do programa.

A freqüência de amostragem é o inverso do período de contagem. Para obtermos 10 amostras de cada uma das 12 sub-rotinas () de um programa que executa por 30s (). Considerando que o tempo mínimo de duração de uma sub-rotina é 5% do tempo total, 1,5s, definimos a probabilidade, p, de o programa executar uma sub-rotina x (*p = 0,05*). Aplicando de forma contrária na fórmula de probabilidade temos:

, onde n é o número de amostras.

Temos que n pode ser definido como:

Aplicando em p, temos:

A freqüência de amostragem é, então:

# Desenvolva um experimento para determinar os seguintes parâmetros da cache de dados do seu computador:

O experimento teórico ideal pode ser definido como a execução de um programa que realiza uma quantidade limitada de acessos ao primeiro nível da memória cache do processador não imediatos, para evitar que o barramento fique ocupado. Um programa contador do tempo de requisição e de resposta para cada operação de leitura/escrita deve medir o tempo total em que uma requisição leva para acessar a memória e retornar (um *hit*); ou acessar a memória, verificar ausência, acessar o nível de memória imediatamente abaixo e retornar (um *miss*). Por fim, é preciso que esse programa contador exiba uma estatística do número de acessos a memória cache e o número de *miss*’s. Sabendo disso, é possível tratar esse experimento de forma similar ao experimento de Bernoulli, em que o tempo pode ser dividido em duas classes: tempo de *miss* e tempo de *hit*. Utilizando as probabilidades de *miss* e *hit*, e o tempo total de operações na memória cache é possível determinar o tempo (*delay*) médio em que o processador fica esperando para o *hit* e o *miss*, de acordo com um intervalo de confiança.

Na prática, programadores têm pouca chance de entender como seus programas interagem com o hardware complexo. Ferramentas são necessárias para ajudá-los com isso. Ferramentas que lidam com a hierarquia de memória e utilização da cache são particularmente importantes, desde que o sistema de memória é um gargalo de desempenho crítico. Abaixo são apresentadas algumas características da arquitetura do processador.

Características do Processador:

Arquitetura Pentium Pro, model 7

Velocidade do Clock 2100 MHz

D1 cache 32 KB, 64B lines

L2 unificada cache 2048KB, 128B lines

**CPU loop + L1 access 4 cycles = 1.95 ns**

**CPU loop + L1 delay 120 cycles = 57.26 ns**

D1 miss latency 88 cycles = 41.93 ns

L2 miss latency 114 cycles = 54.38 ns

D1 replace time 17 cycles

L2 replace time 173 cycles

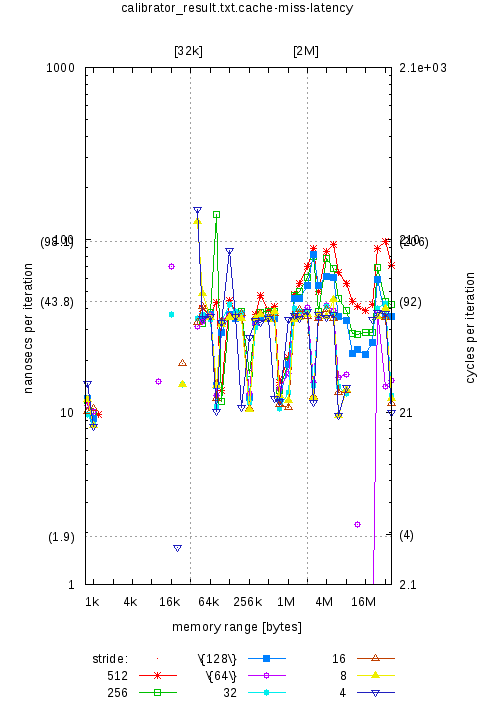
Os resultados desses dados foram obtidos através do CPUID e do Calibrator v0.9e, um micro-benchmark que executa múltiplos acessos dependentes num array com vários tamanhos para estimar os piores casos de latência na memória cache D1 e L2. O tempo de substituição é a penalidade para um miss na cahe com um barramento ocupado, ou seja, quando cada miss é imediatamente seguido por outro. A latência de miss é a penalidade para um miss na cache com um barramento ocioso. O calibrator foi executado sob os seguintes parâmetros:

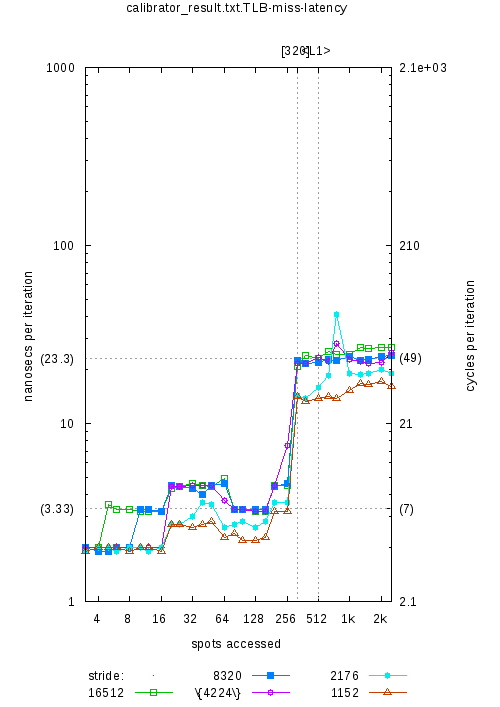
calibrator [MHz] [size] [filename]

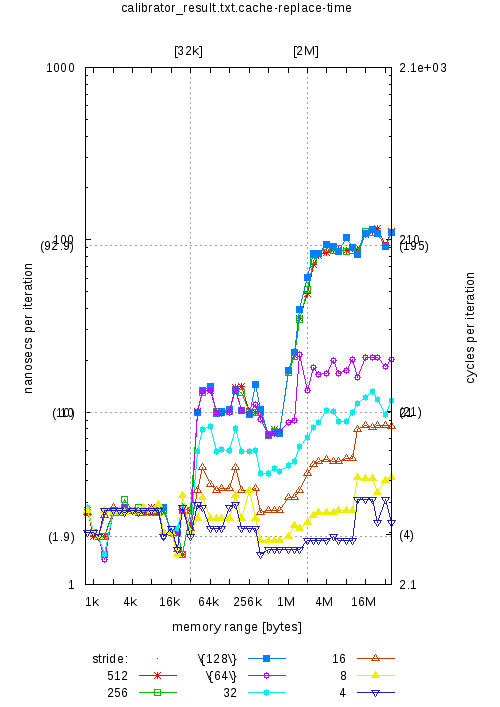
calibrator 2100 50M calibrator\_result

O *delay* de um *hit* na memória cache (no nível 1 de dados) está representado pelo valor *CPU loop+L1 access*. Já, o *delay* de um *miss* na memória cache (no nível 1 de dados) com a busca no nível seguinte da cache está representado por *CPU loop + L1 delay*.

Nos gráficos a seguir, são representados a latência de um miss, a latência de um miss na TLB (um buffer que mantém uma cópia dos endereços físicos dos *frames* de memória acessadas mais recentemente) e o tempo de substituição. No eixo vertical é expresso o tempo em nanosegundos por iteração ou ciclos por iteração, no eixo horizontal é expresso o intervalo de memória utilizado na simulação. As linhas coloridas indicam o tamanho do array testado.







Outra alternativa para a medição dos *delays* de *hit* e *miss* na cachê é através da execução de programas que fazem acesso a dados carregados na memória. Após isso, pode ser utilizado um programa, como o Valgrind, para realizar o *profiling* da memória cache do processador para verificar o número de acessos totais e o número de *miss*.

Programas de Experimento:

//fast.c

#include <stdlib.h>

#include <stdio.h>

int main(void)

{

int h, i, j, a[1024][1024];

for (h = 0; h < 10; h++)

for (i = 0; i < 1024; i++)

for (j = 0; j < 1024; j++)

a[i][j] = 0; // !!

return 0;

}

//low.c

#include <stdlib.h>

#include <stdio.h>

int main(void)

{

int h, i, j, a[1024][1024];

for (h = 0; h < 10; h++)

for (i = 0; i < 1024; i++)

for (j = 0; j < 1024; j++)

a[j][i] = 0; // !!

return 0;

}

O primeiro programa faz uma travessia linha por linha, executa em 0.060s (tempo do usuário, medido pelo comando /usr/bin/time). O segundo programa faz uma travessia coluna por coluna, e executa em 0.572s, ou seja, 9.53 vezes mais lento. Em ambos os programas, o array de 4MB é muito grande para caber na cache. Na primeira versão, a travessia é feita pelas linhas da cache, então somente a cada 16 acessos ocorre um miss no primeiro nível da cache (D1). Na segunda, a travessia é feita através das linhas de cache, então cada acesso no array causa uma miss na cache.

Executando o comando /usr/bin/time e o Valgrind no modo Cachegrind, temos os seguintes resultados que justificam a diferença de tempo.

Para o programa fast.c:

Tempo de Duração do Processo: 0.060s

Command: ./fast.o

I refs: 84,068,013

I1 misses: 616

L2i misses: 612

I1 miss rate: 0.00%

L2i miss rate: 0.00%

**D refs: 52,521,932 (42,010,585 rd + 10,511,347 wr)**

**D1 misses: 656,487 ( 963 rd + 655,524 wr)**

L2d misses: 656,393 ( 878 rd + 655,515 wr)

**D1 miss rate: 1.2% ( 0.0% + 6.2% )**

L2d miss rate: 1.2% ( 0.0% + 6.2% )

L2 refs: 657,103 ( 1,579 rd + 655,524 wr)

L2 misses: 657,005 ( 1,490 rd + 655,515 wr)

L2 miss rate: 0.4% ( 0.0% + 6.2% )

Para o programa low.c:

Tempo de Duração do Processo: 0.572s

Command: ./low.o

I refs: 84,068,003

I1 misses: 616

L2i misses: 612

I1 miss rate: 0.00%

L2i miss rate: 0.00%

**D refs: 52,521,933 (42,010,586 rd + 10,511,347 wr)**

**D1 misses: 10,486,777 ( 961 rd + 10,485,816 wr)**

L2d misses: 10,486,684 ( 877 rd + 10,485,807 wr)

**D1 miss rate: 19.9% ( 0.0% + 99.7% )**

L2d miss rate: 19.9% ( 0.0% + 99.7% )

L2 refs: 10,487,393 ( 1,577 rd + 10,485,816 wr)

L2 misses: 10,487,296 ( 1,489 rd + 10,485,807 wr)

L2 miss rate: 7.6% ( 0.0% + 99.7% )

Para entender melhor os resultados apresentados pelo Cachegrind é importante ler o manual do usuário do Valgrind.

1. O delay do sistema de memória quando ocorre um hit na cache de dados,

O *delay* de um *hit* na memória cache (no nível 1 de dados) está representado pelo valor *CPU loop+L1 access.*

1. O delay do sistema de memória quando ocorre um miss na cache de dados, e

O *delay* de um *miss* na memória cache (no nível 1 de dados) com a busca no nível seguinte da cache está representado por *CPU loop + L1 delay*.

1. Proponha um modelo simples que represente o delay médio do sistema de memória observado por um programa, dada a hit rate. Utilize os parâmetros observados em a) e b).

O modelo que represente o delay médio do sistema de memória observado por um programa pode ser definido como a seguir. Os parâmetros do modelo são as taxas e os atrasos, sendo que os atrasos podem ser definidos a partir da execução do Calibrator e dependem da arquitetura da máquina que está executando o sistema. As taxas podem ser extraídas a partir do Valgrind mediante o *profiling* da memória cache.

Cachedelay = Hitrate\* Hitdelay + Missrate\* Missdelay

Missrate = (1 - Hitrate)

# Estude o gprof (GNU Profile, http://www.cs.utah.edu/dept/old/texinfo/as/gprof\_toc.html), descreva-o e aplique a um exemplo de sua escolha.

Em se tratando de programas de computador, realizar o *profiling* é importante, pois permite que gargalos na execução do código sejam identificados e tratados.

Dentre diversas ferramentas que são capazes de realizar o *profiling* de programas, tem-se o gprof. O gprof, ou GNU Profiler, utiliza a técnica de *profiling block counting*: para cada função do programa sendo analisado, ele adiciona código no começo e no final dela a fim de coletar informações de quantas vezes cada uma das funções foi chamada e quanto do tempo total foi gasto em cada uma.

Levando em consideração um programa em C ou C++, ao utilizar o gprof para analisá-lo, os seguintes passos devem ser seguidos:

*1. Compilar:* gcc programa.c -o programa.out -pg

A opção -pg fará com que sejam adicionados os trechos de código, às funções, necessários para o *profiling*.

O mesmo deve ser feito ao linká-lo.

*2. Execute o programa normalmente*

Depois do programa executado, um arquivo chamado gmon.out será gerado. Esse arquivo conterá todas informações sobre o *profiling* realizado no seu programa.

*3. Interpretar a informação no gmon.out:* gprof programa.out gmon.out > programa.stats

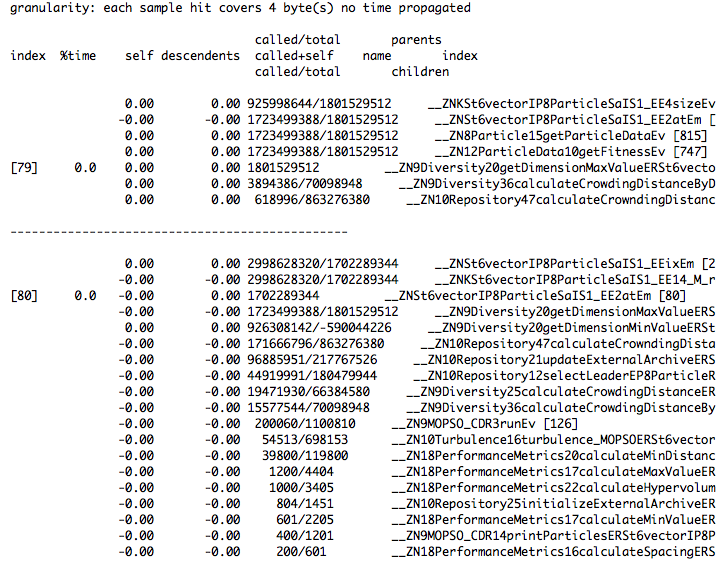
  Em programa.stats, estarão contidos duas tabelas e algumas informações que explicam os dados gerados. A primeira tabela é chamada de Call Graph, que reúne informações de quanto tempo foi gasto em cada função e em cada uma das funções que ele chamou. Com isso, é possível identificar chamadas de funções que estão levando muito tempo para serem executadas.

A segunda tabela, a Flat Profile, mostra quanto tempo foi utilizado em cada função e o seu número de chamadas. Ela é uma forma mais resumida e direta de observar o que é apresentado na Call Graph.

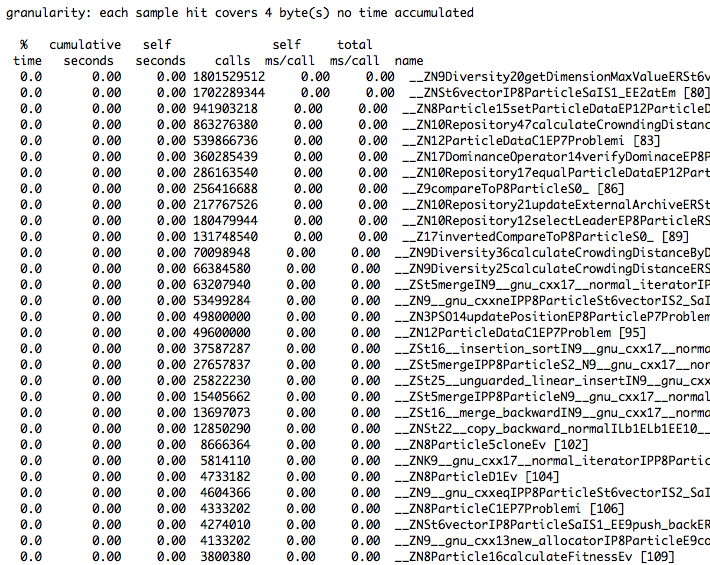
Para finalidades de teste, o gprof foi aplicado a um programa em C++ que implementa o algoritmo de otimização multiobjetiva MOPSO-CDR (*Multiple Objective Particle Swarm Optimization with Crowding Distance and Roulette*) . O algoritmo foi executado considerando 10.000 iterações, 2 objetivos a serem otimizados e um enxame com 20 partículas em um espaço de busca com 4 dimensões.

A máquina na qual o programa foi executado possui o sistema operacional Mac OS X 10.6, processador Intel Core 2 Duo de 2,1 GHz, 1GB de memória RAM DDR2 e barramento de 667 MHz. O tempo gasto para executar o programa foi de aproximadamente 10 minutos.

Devido a complexidade do algoritmo e, consequentemente, implementação, as tabelas geradas foram muito grandes de modo que  serão apresentadas aqui de forma resumida. Primeiramente, a Call Graph:



Agora a Flat Profile:



Observando esses trechos das duas tabelas e também se o arquivo for observado por inteiro, facilmente pode-se notar que os tempos calculados foram zero devido ao truncamento. Isso acontece pelo fato de, apesar de ter grande número de iterações, o algoritmo gasta pouco tempo em cada uma delas, mais precisamente 0,06 segundos. Diante do grande número de funções no código e do nível de precisão dos dados do *profiling*, elas acabam aparecendo como se tivessem gasto 0 segundo.

Entretanto, pode-se facilmente observar que funções são chamadas dentro de cada função e quantas chamadas são realizadas a cada uma delas. Nesse caso, se uma otimização fosse considerada, ela deveria ser realizada nas funções que tiveram o maior número de chamadas.

# Faça um resumo do Capítulo 1 do manual “A Handbook of Techniques and Tools” (http://www.orau.gov/pbm/handbook/handbook\_all.pdf).

O primeiro capítulo do livro “A Handbook of Techniques and Tools” tem como foco dispor informações que auxiliem uma organização a identificar quais tipos de medidas ela necessita, além de prover um guia a respeito do que realmente deve ser medido e dar a capacidade de criar um sistema de medição em qualquer nível de abstração.

Abaixo, discriminamos as seções do capítulo para facilitar a leitura e modularizar o conteúdo.

**Processo de Medição de Desempenho**

O processo de medição pode ser visto como um sistema global de gestão, com o objetivo de atingir conformidade com os requisitos do produto ou serviço a ser prestado para o cliente. Além disso, o processo de medição está relacionado a otimização através da busca por uma melhor eficiência e eficácia do processo ou produto. Essas medidas ocorrem em um ciclo contínuo e escalável, permitindo expansão e melhoria no processo de trabalho ou produto à medida que novas técnicas são descobertas e implementadas.

Um processo de medição nos garante informações importantes sobre os produtos, serviços, e processos que os produzem. Informam o quão bem está o processo, se os objetivos estão sendo alcançados, se os clientes estão satisfeitos, etc. O processo de medição é analisado através de sua eficácia, eficiência, qualidade, prazos estabelecidos, produtividade e segurança.

Medições trazem uma série de benefícios. Permitem identificar se produto ou serviço está de acordo com os requisitos do cliente, ajuda a entender e avaliar os processos utilizados, permite que as decisões sejam tomadas sobre fatos, e não preposições. Dessa forma, se não for possível medir uma atividade, também não será possível garantir o controle sobre ela. E sem o devido controle, é difícil gerenciar o processo.

Os sistemas medidores de desempenho seguem alguns princípios fundamentais: medir somente o que é importante e agregar valor à satisfação do cliente, focar nas necessidades do cliente, e envolver empregados no design e implementação das medições. O loop de feedback resultante desse processo informa o que é necessário para alcançar o objetivo, e é caracterizado pela presença de um Sensor (avalia o desempenho atual), que reporta o desempenho para um trabalhador responsável, que por sua vez compara o desempenho obtido com o desejado. Quando a diferença demanda uma decisão, a pessoa responsável pela decisão deve direcionar no que for preciso para que o próximo valor obtido se aproxime do desejado.

O processo de medição de desempenho é baseado em etapas bem definidas. Identificar o fluxo do processo é uma etapa de extrema importância, pois o fluxo deve corresponder à realidade do sistema medido.

Em seguida, deve-se identificar a atividade crítica a ser medida, que é a atividade que mais faz sentido de ter um sensor para realizar medições individuais.

Após essa etapa, devem-se estabelecer metas ou padrões a serem alcançados, importante para definir o que realmente deve ser considerado uma medição de sucesso.

Estabelecer medidores de desempenho a partir da necessidade de realizar medições individuais é a etapa que se segue. Aqui o importante é identificar pontos que devem ser tratados individualmente.

Após essa etapa, deve-se identificar pessoas responsáveis para cumprirem papéis dentro do processo de medição, e em seguida trabalhar na coleta de dados.

Em seguida, deve-se reportar o desempenho atual da medição, exibindo os indicadores de forma legível para disponibilizá-los em forma de relatório. Após essa etapa, deve-se comparar o resultado obtido com o desejado. Após a comparação, verificar se correções são necessárias e aplicá-las, caso necessário. No caso de haver necessidade de aplicar correções, deve-se fazer com que a correção ajude a medição a encontrar os valores desejados.

Por último, mesmo em sistemas que obtiveram sucesso, mudanças podem ser necessárias, e junto com as mudanças novas metas podem surgir, e talvez as novas metas demandem novas medições.

**Desenvolvendo medições de desempenho – Uma abordagem sistemática**

Medidores de desempenho podem ser agentes efetivos no que diz respeito a lidar com mudanças. Isso evita que mudanças aconteçam de forma imprevisível com objetivo de lidar com algum evento ocorrido. Nesse contexto, a medição de desempenho permite que diferenças entre os valores obtidos e desejados sejam possíveis indicadores de mudanças.

Uma abordagem estrutural cria um raciocínio para a seleção de medição e, caso seja necessário, realizar alterações no processo de trabalho. O desenvolvimento de medidores de desempenho possui uma relação direta com o *Total Quality Management*. As principais tarefas que compõem o coração do desenvolvimento de medidores de desempenho (através de uma discriminação das atividades do processo da AT&T) são:

* Descrever os resultados desejados
* Descrever o processo do trabalho principal
* Identificar os principais resultados necessários
* Estabelecer metas de desempenho para o resultado
* Definir medidas para as metas
* Selecionar as métricas apropriadas

A etapa de descrição do resultado desejado tem como objetivo identificar em alto nível de abstração a meta a ser alcançada. É uma etapa importante, pois o resultado desejado guia a decisão estratégica da empresa. Dessa forma, decisões táticas do que é o negócio e como deve ser feito, além de saber o que deve ser medido está tudo relacionado a esta etapa.

A próxima etapa é a de descrição do trabalho principal envolvido. Para aumentar as chances de alcançar o objetivo, deve-se ter completo do sistema medido. Não é uma etapa fácil de ser avaliada, pois aqui é importante entender a estrutura operacional subjacente ao processo. Entretanto, interconexões entre funções nem sempre são claramente definidas. Aqui deve-se identificar quais atividades serão necessárias e como elas estão interligadas.

Identificar os principais resultados necessários é próxima etapa. Aqui, devem-se identificar quais produtos e serviços serão produzidos nas atividades do processo, ou seja, dá uma razão de ser para cada tarefa inclusa no processo. A identificação dos produtos é de suma importância para que as medições possam ser desenvolvidas.

Em seguida, tem-se a etapa de estabelecer metas para os resultados. Essa etapa define o que é um sucesso de medição. Esta etapa também define o que deve ser feito caso o resultado obtido não seja satisfatório em relação ao resultado desejado. Nessa etapa, deve-se garantir que as pessoas envolvidas são capazes de avaliar suas contribuições.

A definição de medidas para as metas é a próxima etapa. Nesta etapa deve-se definir que método será utilizado para fazer *tracking* do progresso. Como não existem fórmulas para a seleção de métricas de desempenho, algumas características são desejáveis. O medidor deve expressar resultados, e não as atividades necessárias para gerar o resultado. O medidor deve ser baseado em dados que podem ser medidos, e deve estar diretamente relacionado a meta de desempenho. Deve ser prático e de fácil entendimento, entre outras características.

Selecionar as métricas apropriadas é a sexta etapa deste fluxo. Aqui todos os pontos específicos que devem ser medidos devem ser identificados através da análise de demonstrações de medições em relação às metas para identificar as unidades que devem ser pontos de medição.

**Desenvolvimento métricas de desempenho – Abordagem da Universidade da Califórnia**

As Métricas de desempenho devem ser construídas para estimular a melhoria do desempenho, eficácia, eficiência, e níveis apropriados de controle interno. Elas devem incorporar “as melhores práticas” relacionadas ao desempenho sendo medido e a análise de custo/ risco/ benefício, quando apropriado.

O Departamento de Energia promulgou um conjunto de orientações do Gerenciamento da Qualidade Total que indica que métricas de desempenho devem lidar com uma avaliação quantitativa de ganhos em: satisfação do cliente, desempenho organizacional e excelência da força de trabalho.

Os elementos chaves das métricas de desempenho para atender essas orientações devem abordar os seguintes elementos chaves: alinhamento com a visão organizacional, qualidade do produto, a entrega oportuna, a redução de custo e/ou prevenção, redução do tempo de ciclo, satisfação do cliente, satisfazer requisitos de DOE e satisfazer compromissos.

***O Processo***

O primeiro passo no desenvolvimento de métricas de desempenho é envolver pessoas responsáveis pelo trabalho a ser medido. Uma vez que essas pessoas são identificadas e envolvidas, é necessário: identificar processos críticos de trabalho e requisitos dos clientes, identificar os resultados críticos desejados e alinhá-los aos requisitos dos clientes, desenvolver medidas para o processo de trabalho crítico ou resultados críticos, estabelecer metas de desempenho - especificadas em objetivos, critérios e medidas -, padrões ou *benchmarks*.

O teste SMART (S=Especifico, M=Mensurável, A=Atingível, R=Realístico, T=Oportuno/*Timely*) é freqüentemente utilizado para prover uma referência rápida para determinar a qualidade de uma métrica de desempenho particular.

***Tipos de Métricas***

Métricas de desempenho de qualidade permitem para a coleta de dados significativa para a análise de tendências e as taxas de variação ao longo do tempo.

Métricas “sim/não” são utilizadas em certas situações, geralmente envolvendo tendências, que estabelecem, linhas de base, ou metas, inicio dos casos. Porque não há nenhuma calibração válida do nível de desempenho para este tipo de medida, eles devem ser utilizados com moderação.

***Classificação das Métricas de Desempenho***

|  |  |  |
| --- | --- | --- |
| Medida de... | Mede | Representado como taxa de... |
| Eficiência | Habilidade de uma organização realizar uma tarefa | Entrada atual/ entrada planejada |
| Eficácia | Habilidade de uma organização planejar as saídas dos seus processos | Saída atual/ saída planejada |
| Qualidade | Se uma unidade de trabalho foi feita corretamente. Critério para definir “correção” são estabelecidos pelos clientes. | Número de unidades produzidas corretamente/ número total de unidades produzidas. |
| Oportunidade | Se uma unidade de trabalho foi feita na hora certa. Critério para definir “no tempo” são estabelecidos pelos clientes. | Número de unidades produzidas no tempo/ número total de unidades produzidas. |
| Produtividade | A quantidade de um recurso usado para produzir uma unidade de trabalho. | Saídas/ entradas |

Algumas perguntas servem como uma lista de checagem para determinar a qualidade da métrica de desempenho que foi definida.

**Índices de Desempenho**

Um índice de desempenho é uma ferramenta gerencial que permite que múltiplos conjuntos de informações sejam traduzidos em uma medição global. Isso permite que informações de desempenho sejam obtidas de fontes diferentes simultaneamente. Um indicador de desempenho é o resultado de uma análise de uma medida de desempenho. Já um índice representa uma medição estatística de como uma variável (ou conjunto de variáveis) se comportam ao longo do tempo.

A vantagem de utilizar índices como ferramenta de gerenciamento reside em sua habilidade em capturar informação contida em um grande número de variáveis e representa-la em um único valor. A criação de um índice não é um processo sistemático, mas alguns existem conceitos que devem ser considerados na hora de criar um índice. Alguns índices serão apresentados adiante.

O DOE é um índice simples de calcular. Basicamente, é uma combinação linear de parâmetros com pesos predefinidos. Index = (W1xP1)+..+(WnxPn), onde W são pesos constantes e P são medições individuais. Na determinação dos pesos, variáveis como valor do dólar, análise do risco, análise de custo/benefício, opiniões de expert, etc. No caso do DOE, variáveis como número de óbitos, números de dias ausentes ao trabalho, número de casos de dias perdidos são variáveis pesadas.

Outro exemplo de índice baseado em combinação linear é o do Westinghouse Handford Conduct of Operations Event Index. Diferentemente do anterior, este não utiliza pesos. Basicamente, é um índice que mede o número de certos tipos de ocorrências reportadas por 200.000 horas trabalhadas por uma organização. O índice é calculado por: Index = (A+B+C+D+E+F+G)/(H/200.000), onde A,B,..G são variáveis como deficiência de treinamento, falhas de procedimento, etc, enquanto H representa a relação pessoas/hora trabalhada.

A Eastman Kodak utiliza uma metodologia diferente. Basicamente, envolve operações envolvendo o mapeamento do alcance de desempenho de várias métricas em uma escala fixa através de uma matriz de desempenho com passos predefinidos:

1. Selecionar indicadores relacionados à área ao qual se deseja criar o índice.
2. Determinar a importância relativa e o impacto de cada indicador selecionado.
3. Estabelecer um patamar para o indicador (e.g. em uma escala de 1 a 10, 7 poderia ser o valor).
4. Determinar a meta para cada medida.
5. Determinar uma meta de expansão
6. Determinar metas imediatas, que geralmente são valores intermediários entre o patamar e a meta.
7. Determinar valores para os níveis abaixo do patamar, pois é possível que o desempenho esteja abaixo dele.
8. Atribuir valor para o(s) nível(is) restante(s).
9. Analisar a matriz e fazer mudanças necessárias.
10. Desenvolver um sistema para pontuar e exibir os resultados

Outro método consiste em determinar alterações incrementais em um número de indicadores, relativos a um período de tempo base, e então calcular a média dessas alterações. Essa técnica é comumente utilizada em índices econômicos. As alterações incrementais são chamadas de “desempenho relativo”, ou seja, desempenho durante determinado período relativo ao desempenho durante o período de tempo base.

A média de desempenhos relativos é obtida através de uma média geométrica, sendo Index = antilog(sum(log Pi/P0)/n, onde Pi representa o indicador obtido pelo indicador de desempenho ‘i’, P0 é o patamar base e ‘n’ representa o número de indicadores de desempenho na série.

# Estude o PERFMON do Windows, descreva-o e aplique a um exemplo de sua escolha.

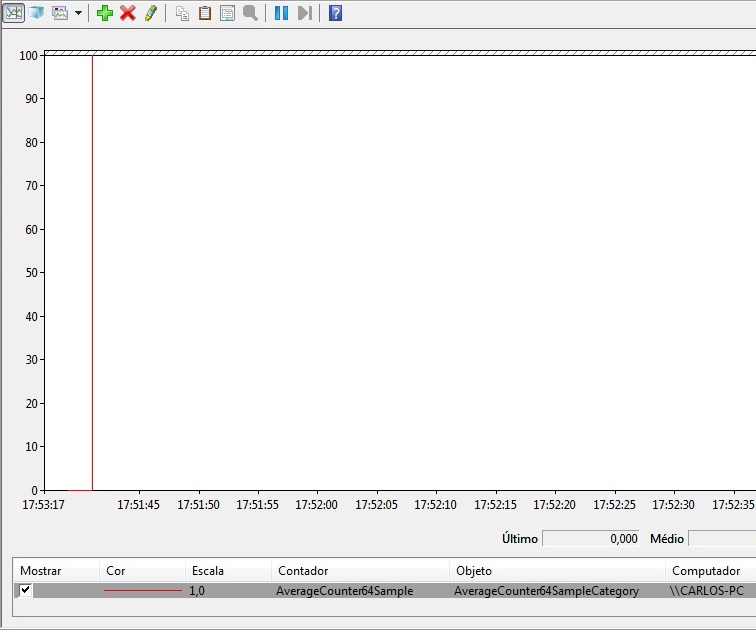
O PERFMON é uma ferramenta destinada à análise de desempenho dos sistemas que estão operando sobre o Windows. Esta ferramenta contém uma listagem de objetos, que podem ser medidos através de seus respectivos contadores de desempenho.

A ferramenta dispõe de recursos que permitem detecção de pontos de afunilamento. Um afunilamento ocorre quando um recurso alcança o limite de sua capacidade, reduzindo o desempenho de todo sistema. Entre os possíveis recursos que causam afunilamento, destacam-se o disco físico, memória, processo, CPU e rede. Se qualquer um desses recursos for utilizado em excesso, o servidor ou aplicativo poderá ficar muito lento ou até falhar.

A análise de desempenho no Performance Monitor é realizada através de contadores de desempenho. Um contador de desempenho é uma métrica que permite a análise de alguma informação. O próprio SO já dispõe de vários contadores nativamente. Entretanto, algumas linguagens de programação integradas com o Windows, como C# por exemplo, permite a criação de contadores de desempenho customizados dentro da própria aplicação. Essa flexibilidade permite que novas métricas sejam adicionadas no PERFMON e viabiliza um melhor acompanhamento do estado do sistema.

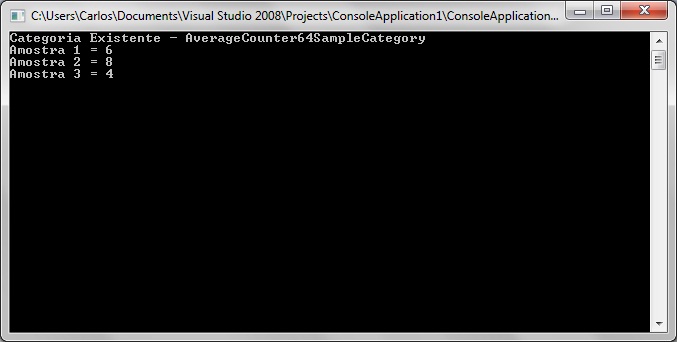
**Aplicação Prática – Descrição:** Utilizamos uma aplicação genérica que processa itens durante uma operação. A quantidade de itens gerados é um número aleatório entre 1 e 10. Desenvolvemos nessa aplicação um contador de desempenho customizado, chamado *AverageCounter64SampleCategory,* que computa a quantidade de itens que foram processados, em média, durante uma operação. Um contador desse tipo pode exibir desde a taxa dos itens que são processados (e.g. bytes enviados) até o número de operações concluídas.

Antes de executar a aplicação, o PERFMON estava ocioso (Figura 1).

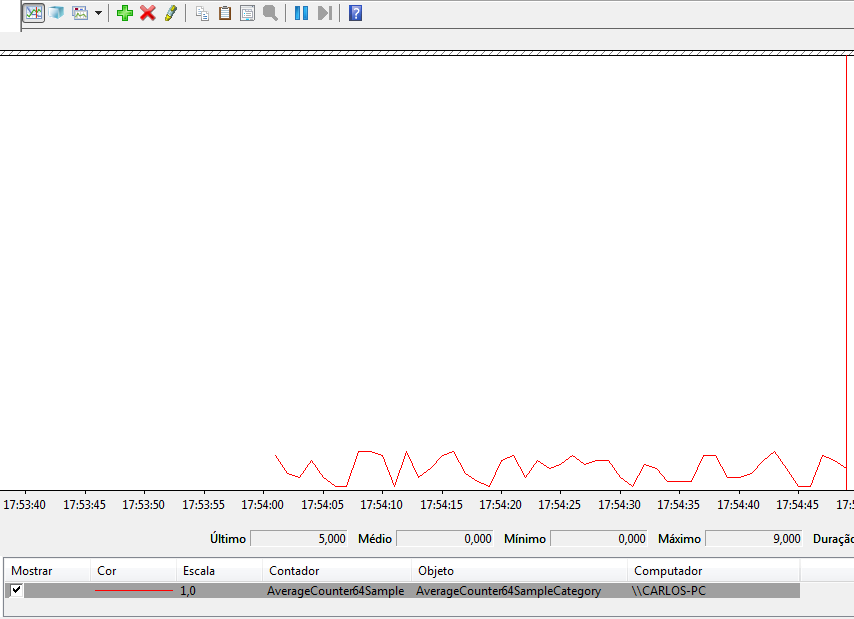


**Figura 1**: Contador de desempenho ocioso

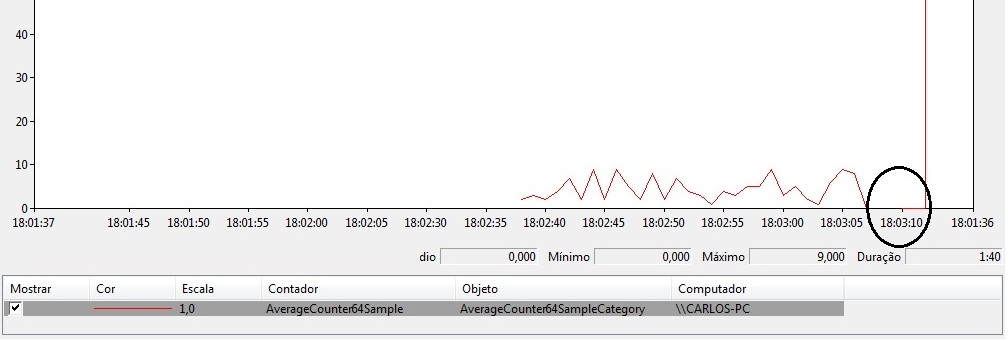
A execução do sistema imprime os valores do contador no console (Figura 2) e imprime no PERFMON (Figura 3). Quando a aplicação termina de executar, o contador de desempenho volta a zerar (Figura 4).



**Figura 2**: Aplicação em execução.



**Figura 3**: Exibição do contador de desempenho enquanto o sistema está em execução.



**Figura 4**: Término da execução do programa. O círculo destaca o momento exato em que os itens pararam de ser processados (contador passa a exibir sempre zero).