Avaliação distribuída de desempenho em redes UNIX heterogêneas

Flávio Henrique S. Lima * *Geraldo Amaral Filho*
João Augusto G. Queiroz † *Paulo Lício de Geus*
{fhlima, geraldoa, aqueiroz, paulo}@dcc.unicamp.br

Universidade Estadual de Campinas - UNICAMP
Instituto de Computação
Caixa Postal 6065 CEP 13081-970
Campinas-SP Brasil

Sumário

Este artigo apresenta um sistema para avaliação distribuída de desempenho em redes UNIX heterogêneas. Tal sistema utiliza elementos de coleta distribuídos entre as máquinas do domínio, disponibilizando os dados coletados através de interfaces RPC e de arquivos de dados. O intervalo de coleta é configurável, e os valores retornados correspondem à média dos valores do intervalo, possibilitando assim uma maior representatividade. Os dados disponibilizados dizem respeito a indicadores da CPU, do sistema operacional e dos discos. A parte cliente do sistema utiliza esses dados para inferir e classificar o desempenho das máquinas de um domínio, possibilitando, por exemplo, a execução de uma aplicação na máquina menos carregada. Graças ao uso de interfaces padronizadas, como RPC, o sistema pode ser utilizado na maioria dos sistemas operacionais UNIX e compatíveis.

Abstract

This article presents a distributed performance evaluation system for heterogeneous UNIX networks. This system uses distributed collecting elements spread within the target domain, and makes the data available through RPC interfaces and data files. The collecting interval is configurable, and average values are returned. The collected data corresponds to CPU, operating system and disks parameters. The client side of the system uses these data to classify the performance of the machines, enabling, for instance, the choice of the least loaded machine. This system can be used in the majority of the UNIX and compatible operating systems, provided RPC is available.

*Programa financiado pelo Tribunal Superior do Trabalho
†Programa financiado pelo Centro de Coordenação de Estudos da Marinha em São Paulo

1 Introdução

Nos últimos anos houve um considerável aumento na utilização de plataformas cliente-servidor em redes locais. Essas plataformas têm sido usadas tanto para aplicações científicas como para aplicações comerciais, muitas vezes substituindo os ambientes de mainframe, em uma tendência que ficou conhecida como *downsizing* ou *rightsizing*. Uma grande parte dessas plataformas é baseada no sistema operacional UNIX, devido à sua flexibilidade, robustez e ampla disseminação em todos os meios. O que temos visto, então, é um crescimento em importância das redes baseadas no UNIX.

Em qualquer rede de computadores que execute aplicações críticas, há sempre uma grande preocupação com a avaliação do desempenho das máquinas. Nas redes UNIX esta preocupação é acentuada pelas características do sistema operacional. No UNIX, como na maioria dos sistemas existentes, não existe transparência de máquina. Isto é, o desempenho na execução de um programa depende inteiramente das condições da máquina na qual o usuário executa o programa.

A escolha da máquina mais adequada para executar um programa é freqüentemente complexa, já que há diversos fatores afetando essa decisão. Alguns desses fatores são estáticos, como a velocidade da CPU, e outros são dinâmicos, como a carga do sistema operacional. Esse problema é agravado quando temos uma rede UNIX heterogênea, onde haja diversas máquinas de diferentes portes e arquiteturas. Este problema é conhecido como balanceamento de carga. Com um balanceamento de carga efetivo, obtém-se uma melhor utilização dos recursos disponíveis, resultando em grande economia.

A avaliação de desempenho é também bastante útil para avaliar se os recursos disponíveis nas diversas máquinas estão sendo suficientes, ou se as máquinas estão sobrecarregadas, caso em que a avaliação ajuda a determinar quais recursos devem ser adquiridos. Os dados de desempenho são úteis também quando se deseja contabilizar ou auditorar o uso dos recursos.

Neste trabalho apresentaremos um modelo para avaliação de desempenho em redes UNIX heterogêneas. O modelo desenvolvido é bastante genérico, podendo ser aplicado no suporte a gerenciamento, contabilização, auditoria, previsão de demanda, balanceamento de carga e outros. A implementação desenvolvida permite que aplicações de usuário sejam criadas utilizando os dados de desempenho gerados pelo sistema. Além disso, os usuários podem utilizar uma interface de comando bastante amigável.

2 Medição de Desempenho

2.1 Introdução

O monitoramento e a avaliação de desempenho dos nós de uma rede local permitem detectar e resolver problemas operacionais, como o aumento do tempo de resposta das aplicações, o consumo excessivo de recursos de CPU e a existência de "gargalos" em memória e em disco.

A avaliação do desempenho de um sistema computacional exige a definição de métricas

adequadas. Estas devem incluir tanto informações estáticas, sobre a configuração do sistema, quanto dinâmicas, extraídas durante a sua execução. Esses dados podem ser usados para várias aplicações.

Os tipos de informações estáticas são: a descrição do hardware (processadores, recursos de armazenamento/comunicação, barramento, etc) e do software (sistema operacional em uso, prioridades de escalonamento, etc). Estas descrições são raramente modificadas, e auxiliam na interpretação dos dados medidos dinamicamente.

As informações dinâmicas são obtidas através de elementos monitores, durante a operação do sistema. Periodicamente, dados dinâmicos são amostrados e armazenados em arquivos para posterior avaliação.

Algumas ferramentas existentes em ambientes SunOS e Solaris, como `iostat` e `vmstat` [Coc95], coletam e apresentam algumas métricas de desempenho para um único nó da rede (abaixo relacionadas). A combinação, compilação e interpretação dessas métricas dependerão do propósito de sua aplicação, da arquitetura da máquina e do conhecimento e experiência do usuário.

Métricas providas pelos comandos iostat e vmstat

- utilização de CPU
- atividades de paginação
- atividades de disco
- *swapping*
- mudanças de contexto
- carga de trabalho
- interrupções

2.2 Métricas de desempenho

Genericamente, as medidas típicas de desempenho são: vazão, tempo de resposta e utilização[ea84]. O conceito de vazão pode ser definido como a quantidade de trabalho executado em um determinado período de tempo. É comum a confusão com o conceito de largura de banda. Por exemplo, os resultados do *benchmark* SPECrate92 são medidas de vazão. Tempo de resposta é o tempo médio de espera para execução de alguma tarefa. O termo latência é usado no mesmo sentido, para avaliação de um protocolo. Utilização é uma medida que exprime a ocupação de um determinado recurso computacional. Os resultados do comando `iostat`, por exemplo, são medidas de utilização. Essas métricas são inter-relacionadas: a melhoria do tempo de resposta aumenta a vazão, porém a recíproca não é verdadeira.

Segundo Patterson e Hennessy[PH94], o tempo de execução de um programa é a medida mais adequada e confiável para avaliar o desempenho de um sistema computacional. O

tempo de resposta é definido como o tempo total de execução do programa, incluindo acessos a discos, à memória, atividades de E/S e do sistema operacional. A medida do tempo de CPU exclui estes outros fatores envolvidos. O tempo de CPU pode ainda ser dividido em *user time* e *system time*. O primeiro corresponde ao tempo de CPU consumido na efetiva execução do programa , enquanto o segundo é o tempo utilizado pelo sistema operacional na execução de atividades relacionadas ao programa, como a leitura de páginas.

O tempo de CPU para um programa pode ser ainda definido como:

```
Tempo de Execução = número de ciclos de clock / freqüência do clock
```

Essa fórmula permite observar que um projetista pode melhorar o desempenho de uma CPU para um determinado programa, tanto reduzindo o número de *clocks* consumidos pelo programa, quanto aumentando a freqüência do relógio.

Existem outras métricas alternativas, válidas para um contexto restrito. MIPS[1] é a mais popular, por ser de fácil compreensão. Os problemas com o seu uso para comparação de desempenho são [PH94]:

- não podem ser comparados computadores com diferentes conjuntos de instrução
- o seu resultado depende do programa utilizado, assim sendo não existe um único valor para uma máquina

2.3 Benchmarks

Um *benchmark* é um programa ou um conjunto de programas com o propósito de testar e/ou medir o desempenho de um sistema computacional, permitindo a comparação entre diferentes sistemas[Ros76].

Os programas mais apropriados para compor um *benchmark* são as próprias aplicações reais de usuários, porém a dificuldade está na escolha dessas aplicações, as quais devem refletir a carga de trabalho esperada.

O uso de um *benchmark* sintético, como os conhecidos *Whetstone* e *Drystone* para a predição de desempenho pode trazer resultados inadequados [PH94]. Através de otimizações no projeto de hardware e do compilador, esses *benchmarks* produzem um desempenho artificial, porém com resultados inexpressivos para aplicações reais.

2.4 Recursos que afetam o Desempenho

A tecnologia de processadores tem desenvolvido rapidamente, enquanto os subsistemas de E/S não têm tido avanços comparáveis. Portanto, estes últimos têm uma função crítica no desempenho de um sistema de computação. A complexidade da arquitetura da máquina é resultado do difícil compromisso custo/benefício. Em um extremo, a CPU monitora todas

[1] Million Instructions per Second

as fases das atividades de E/S, enquanto no outro, controladores inteligentes permitem a transferência independente de dados.

A memória e seu gerenciamento afetam o desempenho de um sistema de computação de duas maneiras [ea84]:

- através do custo associado com o gerenciamento de memória. O *swapping* entre memória principal e secundária demanda CPU e subsistema de E/S; e
- o número de *threads* de controle ativos simultaneamente estabelece um limite superior na proporção dos recursos de processamento que podem ser utilizados concorrentemente. Portanto, compromete a vazão do sistema.

Os sistemas de memória virtual exploram a propriedade de localidade de referência exibida pelos programas, em que apenas uma parte é efetivamente referenciada em um dado intervalo de tempo. Eles utilizam uma combinação de componentes de hardware e software para traduzir endereços virtuais em físicos e vice-versa, permitindo a transferência de dados. Esses sistemas podem empregar técnicas de paginação ou segmentação, ou ainda, a combinação de ambos.

Ao executar tarefas em nível de sistema, o sistema operacional deve escolher os programas e alocar páginas aos processos. Deve ainda escolher e substituir as páginas de acordo com a política adotada. A atividade de paginação tem a particularidade de ser muito dependente das características, das interações e das fases de execução dos programas.

De forma genérica, os processos usuários podem estar em dois estados: *ready* ou *thinking*. Os processos prontos podem ainda estar em dois sub-estados: ativos ou de espera em uma fila. Estes estados sugerem métricas dinâmicas do tipo: número médio de processos prontos, número médio de processos ativos, tamanho da fila e tempo médio de espera.

Como os processadores, muitas das características dos sistemas de E/S são dependentes da tecnologia empregada. O desempenho de tarefas depende de muitos aspectos do sistema computacional: das propriedades dos recursos de E/S, do tipo de barramento para os outros componentes, da hierarquia de memória e do sistema operacional. Este último tem uma importante parcela, porque os sistemas de E/S são compartilhados por múltiplos programas e usam interrupções para comunicar informações sobre as suas operações.

O acesso pelo sistema operacional aos dados armazenados em um disco rígido é efetuado em três etapas. O primeiro passo é posicionar o braço sobre a trilha apropriada. Esta operação é conhecida como *seek*; o tempo para efetuá-la é conhecido como *seek time*. Em seguida, é necessário esperar que o setor desejado da trilha esteja sob a cabeça de leitura/escrita. O tempo associado a esse evento é conhecido como latência de rotação, sendo inversamente proporcional à rotação. O último componente é o tempo de transferência de um bloco de *bits*, sendo função do tamanho do bloco, da velocidade de rotação e da densidade de gravação. Adicionalmente, o controlador ainda acrescenta um componente temporal ao gerenciar a transferência de dados.

3 Arquitetura do Sistema

O sistema operacional UNIX mantém dados sobre o comportamento e a utilização do hardware, discos, memória, rede, etc. A maioria dos sistemas UNIX [UR95] [LMKQ89] permite o acesso a esses dados através dos dispositivos virtuais contidos em `/dev/kmen` e `/dev/vmunix`, que oferecem uma imagem do núcleo do sistema operacional. Essas informações permitem o monitoramento da carga de um nó, servindo de base para a implementação da arquitetura proposta.

A necessidade de projetar um sistema de monitoração flexível, eficiente e genérico exige uma abordagem modular. Dessa forma, a arquitetura básica, derivada do modelo proposto por Jain [Jai91], é composta por módulos com a função de observação, coleta, análise e apresentação (Figura 1). Outras camadas podem ainda ser adicionadas para permitir diferentes formas de apresentação, interpretação, controle e gerenciamento.

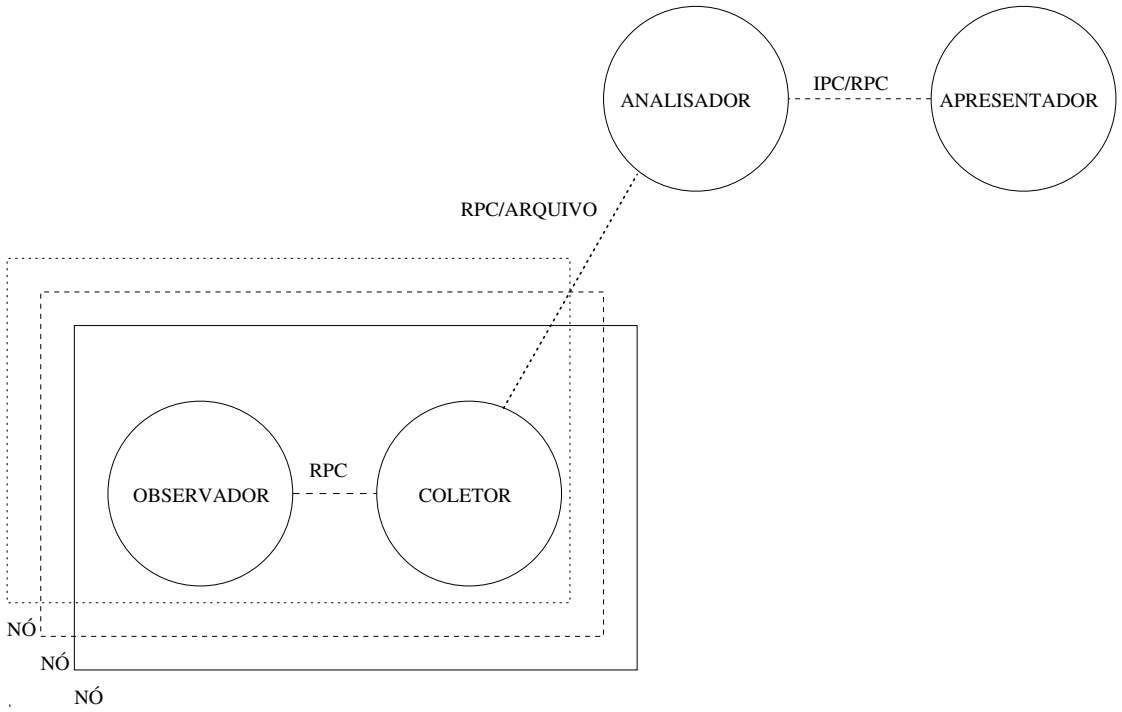

Figura 1: Arquitetura Proposta

Os módulos observadores, residentes em cada nó, adquirem dados de desempenho do *kernel* sem nenhum cálculo adicional. A sua implementação deve ser planejada de forma cuidadosa, para evitar o custo inerente ao processo de monitoração.

Os módulos coletores extraem dados periodicamente dos observadores. Realizam cálculos adicionais, disponibilizando esses dados em um arquivo para um outro módulo, para um outro programa, ou ainda para um *script*.

O módulo analisador avalia os dados adquiridos pelos coletores, compilando-os de acordo

com o propósito da aplicação. O seu escopo corresponderá ao domínio da rede estabelecido na fase de especificação. O módulo de apresentação permite a visualização dos dados coletados e analisados pelos usuários.

A comunicação entre os componentes da arquitetura em diferentes nós exige o suporte de uma tecnologia de *middleware* adequada, permitindo que os projetistas preocupem-se apenas com os aspectos de implementação. O serviço de RPC desenvolvido pela Sun foi o protocolo adotado. As chamadas entre os módulos devem ser preferencialmente assíncronas, não interferindo no processo de observação, coleta e análise. Como alternativa ao RPC, a comunicação entre os módulos analisador e apresentador pode ser feita usando-se alguma técnica de IPC (*Inter Process Communication*), já que estes módulos geralmente estão no mesmo nó. Pode-se ainda juntar os módulos analisador e apresentador em um único programa.

4 Aspectos de Implementação

O sistema possui dois objetivos principais: disponibilizar dados para análise de desempenho em redes UNIX heterogêneas, e demonstrar a utilização desses dados em uma aplicação de avaliação de desempenho. Para que o sistema possa ser utilizado em diferentes implementações UNIX, optamos por utilizar a linguagem C para desenvolvimento, e o protocolo RPC para a disponibilização dos dados. Além do RPC, os dados também são disponibilizados na forma de arquivos comuns, o que possibilita seu uso em aplicativos simples, como *shell scripts*.

O ambiente escolhido para a implementação é composto por diversas máquinas de arquitetura Sun e compatíveis: SPARCstation 1 e 2, SPARCclassic, SPARCstation 10 e Axil 320 (SPARCstation 20). São utilizados nesse ambiente os sistemas operacionais SunOS e Solaris. Todas as máquinas estão conectadas em uma rede ethernet, onde a distância máxima entre as estações é de dois *hops*. Isto é, para que um pacote saia de uma estação e chegue até outra, ele atravessa no máximo dois roteadores. Por causa dessa configuração, simplificaremos o modelo assumindo que o tráfego da rede não afeta as características de desempenho relativo entre os servidores. Em redes onde o retardo de entrega de pacotes é significativo, esta assertiva deve ser revista.

A primeira etapa de implementação consistiu em determinar como seria feita a aquisição dos dados de performance, representada pelo módulo observador. As opções seriam obter os dados diretamente do *kernel* ou através do daemon `rstatd` (explicado a seguir). A obtenção de dados diretamente do *kernel* tem a vantagem de ser mais rápida, mas por outro lado, é altamente dependente do sistema operacional, possuindo pouca ou nenhuma portabilidade.

O *daemon* `rstatd` extrai as estatísticas diretamente do *kernel* e as disponibiliza através de uma interface de RPC, explicada em seguida. O `rstatd` está disponível em várias implementações UNIX, como AIX, HP-UX, SunOS e Solaris. Assim, decidiu-se utilizá-lo como módulo observador para a aquisição dos dados. O daemon `rstatd` é invocado através de uma interface de RPC [Sun94], retornando a seguinte estrutura:

```
struct statstime {
      int cp_time[RSTAT_CPUSTATES];
      int dk_xfer[RSTAT_DK_NDRIVE];
      u_int v_pgpgin;
      u_int v_pgpgout;
      u_int v_pswpin;
      u_int v_pswpout;
      u_int v_intr;
      int if_ipackets;
      int if_ierrors;
      int if_oerrors;
      int if_collisions;
      u_int v_swtch;
      long avenrun[3];
      rstat_timeval boottime;
      rstat_timeval curtime;
      int if_opackets;
};
```

Esta estrutura contém dados de desempenho relativos a: utilização de CPU, utilização de disco, paginação, *swap* e utilização da rede. Todos esses dados são armazenados de forma acumulativa, e alguns deles devem ser multiplicados por escalas de fundo.

Dentro da arquitetura estabelecida há um módulo coletor em cada máquina responsável por extrair periodicamente os dados do `rstatd`, realizar alguns cálculos visando obter dados mais significativos, e disponibilizar esses dados trabalhados através de uma interface RPC. O intervalo de coleta dos dados é configurável durante a instalação. De acordo com as características do ambiente computacional, como a duração média dos jobs executados, pode-se configurar adequadamente o intervalo. Os valores retornados pelo coletor correspondem à média aritmética dos valores obtidos no intervalo.

Internamente ao *kernel* (e também no `rstatd`) os dados são guardados de forma acumulativa. Graças a isto, para calcular a média aritmética dos valores, basta fazer uma coleta no início e outra no final do intervalo, dividindo-se a diferença entre esses dois valores pelo tamanho do intervalo (dado em segundos). Pelo número de coletas ser bastante reduzido, a carga no sistema gerada pelo software é bastante reduzida. Com um intervalo de coleta de 30 segundos, o software (quando ativo) consome aproximadamente 0.5% da CPU de uma SPARCstation II.

Uma vez calculados todos os indicadores de desempenho, eles são disponibilizados em cada máquina de duas formas: através de uma interface RPC e através de arquivos. A interface RPC pode ser consultada sob demanda pelas aplicações clientes através de programas que tenham acesso ao protocolo RPC, como em linguagem C. A interface RPC possui a seguinte definição em XDR (*eXternal Data Representation*).

```
struct stats {
      int cpu;
      float disk;
      float load1;
      float load5;
      float load15;
};
program GETSTATS {
        version GETSTATSVER {
                stats GETSTATSPROC (void) = 1;
        } = 1;
} = 0x20000099;
```

O parâmetro de CPU indica a porcentagem da CPU que esteve livre (*idle*) durante o intervalo de coleta. O parâmetro de disco indica a atividade de todos os discos (número médio de transferências por segundo) durante o intervalo de coleta. Os parâmetros de carga indicam a carga média do sistema operacional no último minuto, nos últimos cinco minutos e nos últimos quinze minutos.

Pode-se observar que os indicadores apresentados cobrem a maioria das variáveis de desempenho. Procurou-se reduzir o número de indicadores a um mínimo, já que se torna exponencialmente mais complexo calcular o desempenho quando aumentam os indicadores. Deve-se considerar também que algumas variáveis são diretamente derivadas de outras. Por exemplo, o número de transferências dos discos cobre tanto as tranferências de dados quanto as transferências motivadas por paginação (*swap*).

A interface de arquivo deve ser consultada quando a aplicação desenvolvida não possui acesso a RPC. Um exemplo disto seria um programa em shell scriptou PERL que, consultando os dados de desempenho, decida em que máquina executar um determinado programa. Os arquivos de dados de desempenho possuem os mesmos indicadores que a interface RPC, em formato texto. Os arquivos são nomeados na forma *nome-da-maquina.stat*. Por haver nomes diferenciados para os dados de cada máquina, é possível gravá-los em um *filesystem* compartilhado (via NFS), possibilitando o acesso imediato em todas as máquinas. Caso não se queira fazer isto, pode-se utilizar comandos UNIX remotos para ler os arquivos, como `rcp`, `rsh` ou mesmo `ftp`.

5 Aplicação exemplo

Nesta seção é mostrada uma aplicação que utiliza os dados de desempenho, aqui chamada de `lscs` (*list computing servers*).

O objetivo do `lscs` é quantificar e classificar o desempenho de máquinas UNIX, visando identificar as máquinas mais adequadas para executar jobs naquele instante. Ela foi desen-

volvida utilizando a linguagem C e as chamadas ao coletor explicadas anteriormente. O `lscs` corresponde aos módulos analisador e apresentador da arquitetura proposta.

De posse dos dados de desempenho, o maior problema é justamente determinar a importância relativa de cada um deles, o que também depende do tipo de aplicação que se pretende executar. Para aplicações de cálculo intensivo, por exemplo, os indicadores de CPU e carga do sistema operacional são mais significativos, enquanto que, para aplicações de E/S intensivo, os indicadores de disco são mais importantes.

Para o cálculo final do desempenho, é preciso haver um quantificador do desempenho absoluto da máquina. Esse valor não é fácil de obter, depende da arquitetura da máquina, e é um valor estático. Por causa dessas considerações, decidiu-se colocar esse valor como um parâmetro do arquivo de configuração, ao lado do nome da máquina monitorada. Esse valor deve ser obtido de um *benchmark* adequado às características do ambiente. Por exemplo, em um ambiente de computação científica, este valor pode ser dado em Mflops, enquanto que em um ambiente de banco de dados, pode ser dado em TPS (Transações por Segundo).

Nesta implementação optou-se por definir um cálculo padrão de desempenho, e ainda possibilitar que o peso relativo seja modificado (através de diretivas `#define` no código-fonte). Na interface da aplicação são mostrados os valores de cada indicador para facilitar o entendimento. Depois de aplicados os pesos relativos, os valores são somados para chegar a um único número, que é o indicador final do desempenho da máquina. Com base neste número, as máquinas são ordenadas e apresentadas ao usuário.

O `lscs` possui dois modificadores (*switches*) de linha-de-comando. O primeiro deles, `-l`, indica que se deseja uma listagem longa, onde são mostrados todos os valores utilizados no cálculo, além de um cabeçalho explicativo. Na ausência do modificador `-l`, somente são mostrados os nomes das máquinas em ordem decrescente de desempenho, um nome por linha. Isto é útil quando a saída do programa for utilizada como entrada de outro programa (*pipe*), como `rexec` ou `rsh`. O modificador `-nX` limita a saída do programa às `X` máquinas de maior desempenho. Por exemplo, pode-se querer saber somente qual a máquina mais rápida no momento. Os dois modificadores podem ser usados sozinhos ou combinados. Um exemplo de utilização do sistema seria executar uma compilação demorada, que possa ser executada em qualquer máquina. Neste caso, teríamos o seguinte comando em C shell :

```
% rsh `lscs -n1` gcc <parâmetros>
```

O comando `lscs` indicaria a máquina menos carregada, onde a compilação seria mais rápida. A saída do `lscs` seria usada como parâmetro do comando `rsh`. Este é um exemplo simples de como obter balanceamento de carga sem necessitar de suporte equivalente do sistema operacional.

A seguir é mostrado um exemplo da saída do comando `lscs -l`, quando executado em um subconjunto de máquinas de diferentes portes, no IC/UNICAMP, com um intervalo de coleta de 30 segundos.

```
-------------------------------------------
Hosts List
(performance descending order)
-------------------------------------------
name            score   load    idle   tps
xingu    :      786     4.47    12     40.0
jaguari  :      702     1.65    34     14.1
atlantic :      213     0.54    71      3.3
ttl      :      140     0.67    70      2.2
marumbi  :      43      2.58    0       2.7
```

A saída do `lscs -l` classifica os *hosts* em ordem descendente de *score*. Este *score* é calculado com base nos outros parâmetros mostrados, no indicador de performance absoluta e nos pesos relativos anteriormente citados. São mostrados também os valores médios no intervalo da carga do sistema (*load*), da porcentagem de CPU livre (*idle*) e do número médio de transferências por segundo de todos os discos de cada *host* (tps acumulado).

6 Conclusão

Neste trabalho foram estudados os diversos fatores envolvidos na análise do desempenho de máquinas UNIX conectadas em rede. Foram aqui identificadas as principais métricas de desempenho nesse ambiente e seus inter-relacionamentos.

Como resultado desse estudo, foi desenvolvida uma arquitetura de avaliação distribuída de desempenho, cujo objetivo é disponibilizar dados de desempenho das máquinas de um domínio. Esta arquitetura permite uma maior independência entre as fases de observação, coleta, análise e apresentação de resultados. A arquitetura torna possível o acesso aos dados coletados por uma interface de RPC, ou ainda através de arquivos de dados.

Para validar a arquitetura proposta, foi desenvolvida uma aplicação-exemplo, chamada `lscs`. Esta aplicação classifica o desempenho das máquinas baseada em uma métrica estática, fornecida pelo usuário, e métricas dinâmicas, fornecidas pela arquitetura proposta. Pode-se configurar intervalos de coleta, e também modificar o peso relativo dos diversos elementos analisados, como E/S ou CPU.

Neste projeto confirmamos a viabilidade de implementar sistemas de avaliação de desempenho como processos de usuário, e ainda com pequeno impacto sobre a carga de processamento dos nós. Por utilizar processos de usuário, este modelo pode ser implementado na maioria dos sistemas UNIX. Tais sistemas são essenciais no suporte a atividades de gerenciamento, contabilização, auditoria, previsão de demanda e balanceamento de carga.

Pela experiência obtida com a implementação de uma aplicação que classifica o desempenho dos nós de um domínio, pudemos verificar que tais aplicações são bastante complexas, já que os usuários possuem diferentes necessidades de computação, e conseqüentemente, diferentes métricas e prioridades.

Referências

[Coc95] A. Cockroft. *Sun Performance and Tuning.* SunSoft, 1995.

[ea84] E.D. Lazowska et all. *Quantitative System Performance.* Prentice-Hall, Inc., 1984.

[Jai91] R. Jain. *The Art of Computer System Performance Analysis-Techniques for Experimental Design, Measurement, Simulation and Models.* John Wiley & Sons, 1991.

[LMKQ89] S.L. Leffler, M.K. McKusic, M.J. Karels, and J.S. Quarterman. *The Design and Implementation of the 4.3BSD UNIX Operating System.* Addison-Wesley, 1989.

[PH94] D.A. Patterson and J.L. Hennessy. *Computer Organization & Design The Hardware/Software Interface.* Morgan Kaufmann Pub., Inc., 1994.

[Ros76] S. Rosen. *Lectures on the Measurement and Evaluation of Performance of Computing Systems.* SIAM, 1976.

[Sun94] Sun Microsystems. *Sun Networking Programming*, 1994.

[UR95] Rodrigo C. Uchôa and Noemy R. Rodrigues. Suporte para monitoramento e controle de carga em sistemas distribuídos. *13o Simpósio Brasileiro de Redes de Computadores*, 1995.