CMGS - Obtendo eficiência no escalonamento de aplicações memory-intensive em clusters

Luís C. D. Alves[1⋆], Marcos J. Santana[1],
Regina H. C. Santana[1], and Wilnice T. R. Oliveira[2]

[1] Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, São Carlos, SP, Brasil.
[2] Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto,
Universidade de São Paulo, Ribeirão Preto, SP, Brasil
{luisc,mjs,rcs}@icmc.usp.br
{wilnice}@ffclrp.usp.br

Abstract This paper proposes and evaluates a new scheduling policy for heterogeneous cluster systems called CMGS. This policy aims at higher performance of systems with predominantly memory-intensive workloads; however, the results obtained with CPU-bound workloads were also competitive. The scheduling considered is global, dynamic, distributed and not adaptive, using remote execution as its load balancing mechanism. The performance evaluation, of the proposed policy, was performed using statistical analyses and CPU memory based load sharing policy as reference, it is currently highlighted as one of the most efficient for the studied system.

Keywords: Performance evaluation, scheduling policy, computer cluster, single-system image, memory-intensive applications, simulation and remote execution

1 Introdução

Os clusters são formados por estações de trabalho, ou computadores pessoais, conectados por uma rede de comunicação, o que permite a execução de aplicações paralelas [1]. Em tais sistemas, o alto desempenho é obtido através das políticas de escalonamento, cujo objetivo é balancear a carga de trabalho, otimizando a utilização de qualquer recurso que possa influenciar, de maneira significativa, no desempenho geral da plataforma [2].

Na literatura encontram-se trabalhos que descrevem políticas que utilizam tanto recursos de processador, quando de memória, para escalonar aplicações *CPU-bound*. No entanto, com o crescente consumo de memória pelas aplicações, faz-se necessário o estudo de políticas que otimizem a utilização dos *clusters* na presença de cargas *memory-intensive*, garantindo, também, tempos de resposta satisfatórios para as *CPU-bound*.

⋆ Projeto apoiado pela FAPESP (Fundação de Amparo à Pesquisado do Estado de São Paulo), processo n° 06/54471-0.

Assim, neste trabalho é proposta uma nova política de escalonamento, denominada CMGS (CPU and Memory Group Scheduling), que, através de execuções remotas, visa tratar o problema anteriormente descrito. Neste trabalho não é abordada a migração preemptiva de processos, uma vez que esta abordagem não apresenta resultados satisfatórios para a categoria de aplicação estudada [2]. Para comprovar a eficiência da política CMGS serão realizadas comparações estatísticas, com base em dados obtidos através de simulação, com outra política denominada *CPU memory based load sharing*, proposta em [2] e destacada como uma das mais eficientes em [1].

2 Trabalhos relacionados

A comprovação definitiva do ganho de eficiência, ao considerar informações sobre a memória no escalonamento, foi apresentada em [3], através de experimentos abordando o particionamento de sistemas paralelos com partições fixas. Posteriormente bons resultados foram obtidos, principalmente aqueles apresentados em [2], que ao utilizar índices de memória no escalonamento, para dois ambientes distintos, obteve reduções significativas na quantidade de ausências de páginas sofridas pelos processos, diminuindo assim a perda de desempenho total do sistema.

Diferentes métodos para unificar os índices de memória e CPU foram estudados, porém em [4], o uso do algoritmo *Assign-u* apresentou bons resultados e facilidade de implementação. Nesse trabalho os autores reduziram o tempo de resposta das aplicações ao modificar as políticas existentes no ambiente MOSIX e na biblioteca PVM.

Outros trabalhos de destaque são apresentados em [5] e [6]. No primeiro é realizada a migração da melhor tarefa, partindo de uma máquina congestionada para outra com maior quantidade de memória livre. A escolha da melhor tarefa depende da política de seleção utilizada. Já no segundo é proposta uma solução ponto-a-ponto, denominada *Parallel Network RAM*, que procura diminuir a sobrecarga causada pela sincronização e comunicação das aplicações paralelas.

Novos cenários também são estudados, como o proposto em [1], que realiza o balanceamento de carga em um sistema de interligação de *clusters* heterogêneos, através da política LCF *Local Cluster First*.

Apesar dos trabalhos presentes na literatura, este artigo se destaca pela abordagem inovadora na forma de combinar índices de carga, e pelos índices utilizados estarem disponíveis a partir do pseudo-sistema de arquivos /proc do sistema operacional GNU/Linux, permitindo a sua coleta no nível de usuário, evitando trocas de contextos excessivas.

3 A política CMGS

O escalonamento baseado em grupos, desenvolvido neste trabalho, consiste em realizar n seleções, onde cada uma das n etapas analisará um índice diferente. Após cada etapa, os nós que não atendem aos requisitos pré-estipulados são

descartados. Assim, a n-ésima seleção indicará a máquina alvo que receberá o processo. Tal método difere-se dos comumente encontrados na literatura, cuja análise de cada índice é feita separadamente e a decisão tomada por apenas um dos índices analisados, dependendo do estado da plataforma no momento do escalonamento.

A política CMGS utiliza o conceito de grupos, onde $n = 2$. A primeira etapa consiste em selecionar as máquinas cujas quantidades de páginas ativas sejam menores que a quantidade média de páginas ativas nas máquinas do *cluster*. A segunda etapa consiste em escolher, dentre as máquinas previamente selecionadas na etapa anterior, aquela que apresenta a menor fila de prontos. O tamanho da fila de prontos é constituído da quantidade de processos no núcleo do sistema operacional, acrescido dos processos enviados para execução remota que se encontram em trânsito na rede de comunicação.

O escalonamento abordado pela política CMGS pode ser descrito, de acordo com a taxonomia de Casavant e Kuhl [7], como global, dinâmico, distribuído e não adaptativo. O algoritmo a seguir expressa o funcionamento dessa política, dado que p denota a quantidade de páginas ativas, sendo representado por p_i a quantidade de páginas ativas na máquina i do *cluster*, sendo este composto por N nós, cujo i-ésimo nó é denotado por no_i. Por p_origem denota-se a quantidade de páginas ativas na máquina que está realizando o balanceamento de carga.

```
Algoritmo CMGS
    p_media = média de p dos N nós;
    if(p_origem > p_media) {
        for(i = 0; i < N; i++){
            if(p_i < p_media)
                Selecionar No_i;
        }
        Enviar para execução remota no nó,
          entre os selecionados, com a menor fila de prontos;
     }
     else
         Executar localmente;
fim algoritmo
```

4 Modelos

O *cluster* a ser avaliado foi modelado através de rede de filas, utilizando-se simulação baseada em *traces* para resolver tal modelo. Como *software* de simulação utilizou-se o simulador baseado em processos Simclux [8].

O modelo de redes de filas desenvolvido descreve um *cluster* executando um sistema de imagem única. Assim, além de não conhecer a quantidade de máquinas que constitui o sistema, o usuário pode executar a sua aplicação em qualquer máquina da plataforma, uma vez que tais ambientes utilizam sistemas de arquivos distribuídos, ou pelo menos os simulam.

Cada máquina do *cluster* é modelada com um subsistema composto por três centros de serviços: processador, disco e *interface* de rede. Nesse modelo o processo é tratado como um *token*, que circula pelo sistema a fim de consumir seu tempo de execução. Na Figura 1 se encontra o modelo adotado.

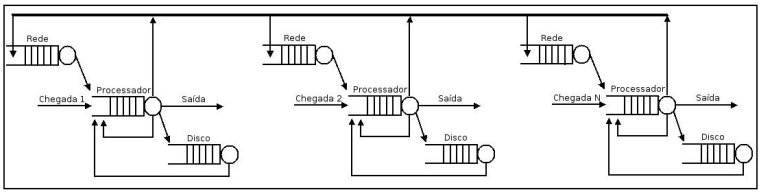

Fig. 1. Modelo de rede de filas do sistema de imagem única.

Os arquivos de *traces*, utilizados neste trabalho, são os mesmos disponibilizados por [9] e utilizados em [2] e [1]. Estes arquivos foram disponibilizados originalmente no repositório *Parallel Workloads Archive* [3].

Nos *traces* utilizados as entradas: momento de submissão, nó de destino, tempo de processamento e nome da aplicação foram retiradas dos arquivos originais, porém, a quantidade de memória utilizada foi gerada através de uma distribuição de Pareto, com objetivo de melhor caracterizar as cargas propostas. Tal distribuição é a que melhor descreve o comportamento da utilização de memória por aplicações[10, 9, 1]. A função computacional utilizada foi obtida em [11].

5 Planejamento de experimento

Para desenvolver este trabalho adotou-se, como planejamento de experimento, o modelo fatorial completo 2^3 [12]. Os fatores são: a) Política de escalonamento, b) Plataforma distribuída e c) Carga de Trabalho submetida ao sistema.

5.1 Políticas de escalonamento

O primeiro nível deste fator corresponde a política *CPU memory based load sharing*, definida em [2]. Esta política utiliza tanto informações de memória quanto de CPU. O algoritmo a seguir retrata o seu funcionamento, onde ML_j representa a quantidade de memória ocupada no nó j, RAM_j indica a quantidade de memória física residente no nó j, L_j é o tamanho da fila de processos prontos na máquina j e finalmente CT é o tamanho máximo da fila de processos prontos.

```
Algoritmo CPU Memory based load sharing
    if(ML_j < RAM_j)
```

[3] http://www.cs.huji.ac.il/labs/parallel/workload/

```
        index_j = L_j;
    else
        index_j = CT;

    if(index_j < CT)
        Executar o processo localmente;
    else
        Executar o processo remotamente
          no nó com a menor fila de prontos;
fim algoritmo
```

O segundo nível corresponde a política CMGS, já descrita anteriormente na SEÇÃO 3.

5.2 Plataformas

Como níveis deste fator foram selecionadas duas plataformas heterogêneas, que serão denominadas plataforma I e II. A primeira apresenta heterogeneidade no poder de processamento e a segunda na quantidade de memória disponível.

A Tabela 1 apresenta, na segunda coluna, os valores adotados, em MIPS (*Millions of Instructions Per Second*), para o poder de processamento de cada um dos nós da Plataforma I. Já na terceira coluna é apresentada a quantidade de memória DRAM adotada para cada nó da Plataforma II.

Table 1. Poder de processamento e quantidade de memória DRAM dos nós que constituem as Plataformas I e II.

Nó	Velocidade processadores (MIPS) Plataforma I	Quantidade de memória disponível (MB) Plataforma II
0	500	512
1	1000	1024
2	2000	2048
3	500	512
4	1500	1536
5	500	384

Para a Plataforma I, a quantidade de memória disponível em cada nó é de 512 MB, enquanto cada nó da Plataforma II possui 1000 MIPS de poder de processamento.

Para ambas as plataformas adotou-se: 10 ms para tratamento de uma ausência de página; 100 Mbps para taxa de transmissão da rede; 100 ms para realizar uma execução remota; 60% da quantidade de memória requisita para *working set* das aplicações; 0,1 ms para custo de troca de contexto; e, finalmente, utilizou-se $\sigma = 10$. Esta constante é utilizada na Equação 1 que define o momento em que uma ausência de página deverá ocorrer. Ao adotar $\sigma = 10$, define-se que uma

ausência de página ocorrerá a cada dez milhões de instruções, caso a quantidade de memória disponível no nó seja menor que o *working set* do processo escalonado. De acordo com [2], em uma plataforma real tem-se $1 \leq \sigma \leq 10$, sendo assim, o valor adotado representa o pior caso, o que torna a simulação do sistema conservadora, evitando resultados fantasiosos.

$$\sigma * \frac{Mem.\ Solicitada_j^i}{Mem.\ Alocada_j^i} \tag{1}$$

Em 1, o numerador $mem.\ Solicitada_j^i$ indica a quantidade de páginas solicitadas pelo processo i na máquina j e $mem.\ Alocada_j^i$ a quantidade de páginas alocadas ao processo i na máquina j.

Em ambas as plataformas os nós que as compõem são interligados por duas redes isoladas de comunicação. A primeira possui a função de comunicação com o usuário de envio de processos para execução remota. A segunda rede, denominada rede de sincronização, é utilizada para a troca de informações de carga e de sincronização entre as máquinas do sistema. Devido a presença de ambas as redes, adotou-se uma política de informação orientada a mudança de estado [8].

5.3 Cargas de trabalho

Definiram-se dois níveis para as cargas de trabalho. Na primeira, denominada "A", pretende-se caracterizar aplicações mistas, com a presença tanto de aplicações *CPU-bound* quanto *memory-intensive*. Já a segunda, denominada "B", reflete aplicações unicamente *memory-intensive*.

Para tais cargas adotou-se os seguintes valores para k, p e α:

- Aplicações mistas: k = 64 MB; P = 256 MB; $\alpha = 1$.
- Aplicações *memory-intensive*: k = 128 MB; P = 256 MB; $\alpha = 5$.

As variáveis k, p, e α, descritas em [11], são utilizadas para definir, respectivamente, o menor e o maior valor de memória a ser gerado e o "peso da cauda" da distribuição.

6 Resultados

As análises realizadas, mantendo-se uma confiança de 95%, obtidas com vinte interações, são baseadas nas seguintes variáveis de resposta: a) Perda de desempenho, definida pela razão entre o tempo total gasto pelos processos e o tempo total de CPU dos mesmos, b) Ausências de páginas e c) Execuções remotas.

6.1 Influência dos fatores

A partir da soma dos quadrados [12], observa-se que a política e a plataforma são os fatores que mais influenciam na perda de desempenho 53,96%. Já a carga

de trabalho influi 8,14% e 5,54% menos que a política e a plataforma, respectivamente.

Na variável ausências de páginas, a influência da política aumenta para 28,78%. Nesta variável de resposta, a influência da carga de trabalho reduziu 9,81%, influindo 18,97% menos que a política. Ainda nas ausências de páginas, a política e a plataforma são os fatores que mais influenciam 65,96%, apresentando um crescimento de 12% se comparado à influência na variável perda de desempenho.

É importante observar que o maior impacto apresentado pela política foi registrado na variável execução remota 93,42%, sendo que a plataforma e a carga de trabalho influenciam em apenas 5,04% e 0,95%, respectivamente.

6.2 Desempenho na Plataforma I

O gráfico disposto na Figura 2.a apresenta a média da perda de desempenho durante as simulações.

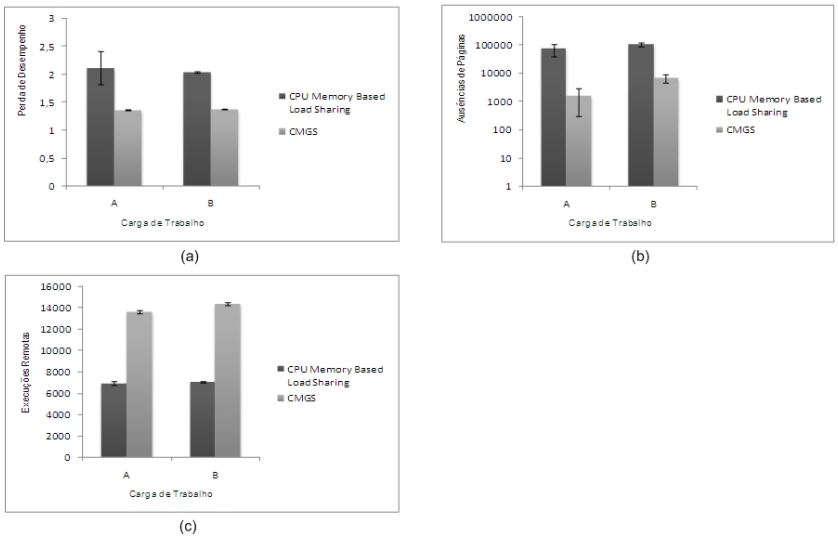

Fig. 2. Resultados obtidos na Plataforma I: a) Perda de desempenho, b) Ausência de páginas e c) Quantidade de execuções remotas.

Por não haver sobreposição dos intervalos de confiança, é possível afirmar que a política CMGS apresenta melhores resultados para plataformas com heterogeneidade de CPU, tanto para cargas mistas quanto para cargas *memory-intensive*.

Para cargas mistas a política baseada em grupo apresentou aproximadamente 35,85% menos perda de desempenho que a política *CPU memory based load sharing*, já para a carga *memory-intensive* a diferença de desempenho foi de 32,35%.

As vantagens de desempenho apresentadas são conseqüências da quantidade de ausências de páginas, sofridas pelos processos, sob o escalonamento da política *CPU memory based load sharing*, e pela pequena quantidade de execuções remotas realizadas, como é possível observar nas Figuras 2.b e 2.c, respectivamente.

Ao enviar processos para execução remota somente quanto o tamanho máximo da fila de prontos é atingido, ou quanto a quantidade de memória solicita é maior ou igual a quantidade de memória existente no nó, a política *CPU memory based load sharing* permite que alguns nós do sistema sejam sobrecarregados, enquanto outros permanecem com carga moderada ou até mesmo sem carga alguma. Isso não ocorre com a política CMGS que realiza em torno de 49% e 50,86% mais execuções remotas para as cargas mista e *memory-intensive*, respectivamente.

Esse comportamento da política CMGS deve-se à sua característica de realizar o balanceamento de carga desde a primeira submissão de processos ao sistema, assim, as ausências de páginas são retardadas e compartilhadas de maneira uniforme por todos os nós.

O balanceamento em etapas é outro fator favorável à política baseada em grupo, uma vez que as máquinas com processadores de menor potência tendem a receber processos para execução remota, apenas se a quantidade de memória disponível estiver acima da média do cluster e, se a sua fila for a menor dentre as máquinas previamente selecionadas. Esta característica evita o acúmulo de processos em máquinas com processadores de baixa capacidade.

6.3 Desempenho na Plataforma II

Na plataforma II, as perdas de desempenho sofridas pela política *CPU memory based load sharing* foram maiores que as registradas na Plataforma I, como pode ser observado na Figura 3.a.

Nesta plataforma a política CMGS foi superior à política *CPU memory based load sharing* em cerca de 78,40% e 96,68% para as cargas mista e *memory-intensive*, respectivamente. É possível perceber ainda que a política baseada em grupo apresentou reduções entre 2,90% e 6,63% na perda de desempenho, ao comparar os resultados da Plataforma II, com aqueles obtidos na Plataforma I. Para este mesmo cenário, a outra política aumentou a perda de desempenho entre 63,95% e 94,94%.

Essa vantagem, obtida pela política baseada em grupo, deve-se à seleção inicial pela quantidade de memória disponível nas máquinas, o que torna a política adaptável a heterogeneidade de memória presente na plataforma. Por outro lado, a política *CPU memory based load sharing* permite que máquinas, com pequena capacidade de armazenamento, recebam um grupo de processos que consumam uma grande quantidade de memória, levando a plataforma a sofrer uma grande quantidade de ausência de páginas, como pode ser observado na Figura 3.b.

Tanto na Figura 3.a, quanto na Figura 3.b, é possível perceber o grande intervalo de confiança registrado para a carga *memory-intensive* na política *CPU memory based load sharing*. Esse fato demonstra que esta política torna-se extremamente instável para cargas deste tipo, principalmente à medida que a sobrecarga do sistema é ampliada.

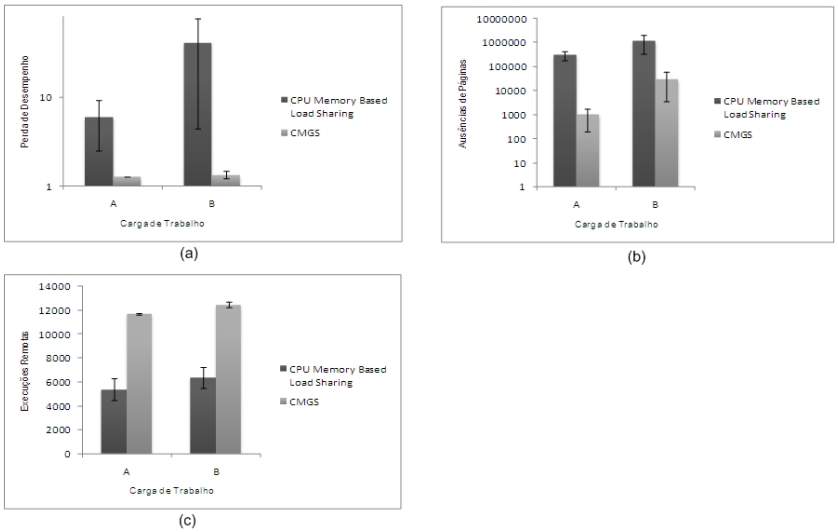

Fig. 3. Resultados obtidos na Plataforma II: a) Perda de desempenho, b) Ausência de páginas e c) Quantidade de execuções remotas.

Ao analisar os resultados das simulações, percebe-se que o fato da política *CPU memory based load sharing* enviar poucos processos para execução remota, como pode ser observado no gráfico da Figura 3.c, juntamente com o balanceamento tardio, leva a mesma a acumular uma grande quantidade de processos na fila de *swap*, incrementando ainda mais a perda de desempenho.

7 Conclusões

Este trabalho propôs uma nova política de escalonamento para aplicações *memory-intensive*, comprovando sua eficiência através da avaliação de desempenho realizada com uma segunda política difundida na literatura, que também utiliza informações de memória como índice de escalonamento.

Para realizar a avaliação utilizou-se modelagem por redes de fila e, posteriormente, empregou-se simulação baseada em *trace* para resolver o modelo proposto. Os *traces* utilizados foram disponibilizados em [9] e [2] e utilizados recentemente em [1] e [8].

Para realizar o estudo proposto, utilizou-se o planejamento de experimento fatorial completo 2^3, com os fatores: Política de escalonamento, plataforma distribuída e carga de trabalho. A partir dos experimentos concluiu-se que dentre os fatores citados a política de escalonamento é aquele que mais influi no desempenho da plataforma. Todavia, a plataforma distribuída também possui impacto significativo, principalmente nas variáveis perda de desempenho e ausências de páginas. Sendo assim, conclui-se também que a escolha da política é de fundamental importância para o bom desempenho da plataforma distribuída. Nesse

sentido, a política CMGS apresentou-se superior à política *CPU memory based load sharing* em todas as combinações realizadas.

A maior redução na perda de desempenho obtida pela política CMGS foi registrada na carga do tipo *memory-intensive* na Plataforma II. Essa vantagem, que foi de aproximadamente 96,68%, deve-se ao fato da desta política adaptar-se à heterogeneidade existente, através da seleção em duas etapas, onde as máquinas com memória disponível abaixo da média do sistema são descartadas como potenciais receptoras de processos enviados para execução remota.

Outra vantagem apresentada pela política baseada em grupo refere-se a sua capacidade de escalonar diferentes tipos de cargas, enquanto a política *CPU memory based load sharing* é destinada apenas à cargas *CPU-bound*. Além disso, a política CMGS, apesar de apresentar uma implementação mais complexa, é extremamente estável, evitando a oscilação dos resultados obtidos.

References

1. Wang, Y.M.: Local cluster first load sharing policy for heterogeneous clusters. J. Inf. Sci. Eng. **23**(2) (2007) 497–510
2. Xiao, L., Chen, S., Zhang, X.: Dynamic cluster resource allocations for jobs with known and unknown memory demands. IEEE Trans. Parallel Distrib. Syst. **13**(3) (2002) 223–240
3. Peris, V.G.J., Squillante, M.S., Naik, V.K.: Analysis of the impact of memory in distributed parallel processing systems. In: SIGMETRICS '94: Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems, New York, NY, USA, ACM Press (1994) 5–18
4. Amir, Y., Awerbuch, B., Barak, A., Borgstrom, R.S., Keren, A.: An opportunity cost approach for job assignment in a scalable computing cluster. IEEE Trans. Parallel Distrib. Syst. **11**(7) (2000) 760–768
5. Barak, A., Braverman, A.: Memory ushering in a scalable computing cluster (1997)
6. Oleszkiewicz, J., Xiao, L., Liu, Y.: Effectively utilizing global cluster memory for large data-intensive parallel programs. IEEE Trans. Parallel Distrib. Syst. **17**(1) (2006) 66–77
7. Casavant, T.L., Kuhl, J.G.: A taxonomy of scheduling in general-purpose distributed computing systems. IEEE Trans. Softw. Eng. **14**(2) (1988) 141–154
8. Alves, L.C.D.: Políticas de escalonamento memory-intensive para aplicações distribuídas. Master's thesis, ICMC, USP, São Carlos, SP (Junho 2008)
9. Xiao, L., Zhang, X., Qu, Y.: Effective load sharing on heterogeneous networks of workstations. In: IPDPS '00: Proceedings of the 14th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, IEEE Computer Society (2000) 431
10. Harchol-Balter, M., Downey, A.B.: Exploiting process lifetime distributions for dynamic load balancing. ACM Trans. Comput. Syst. **15**(3) (1997) 253–285
11. Crovella, M., Harchol-Balter, M., Murta, C.: Task assignment in a distributed system: Improving performance by unbalancing load. Technical report, Boston, MA, USA (1997)
12. Jain, R.: The art of computer systems performance analysis: Techniques for experimental design, measurement, simulation and modeling. 1 edn. John Wiley, New York (1991)