Dynamic Voltage Scaling in Multitier Web Servers with End-to-End Delay Control

Tibor Horvath and Tarek Abdelzaher and Kevin Skadron and Xue Liu

Universidade Federal Fluminense

Agenda

- Introdução.
- Arquitetura e modelagem.
  - Primeiro algoritmo.
  - Segundo algoritmo.
- Implementação.
- Avaliação.
- Conclusões.

Introdução

Proposta e Objetivos

Dois Algoritmos

- DVS distribuído.
- Uma abordagem ótima.
  - Dentro das hipóteses adotadas.
- Uma abordagem aproximada.
- Ambas focadas em requisitos *soft real time*.

Aplicação a Servidores *Web Multitier*

- Servidores divididos em estágios.
- Exemplo:
  - Conteúdo estático.
  - Conteúdo dinâmico.
  - Bases de dados.
- Em geral, super-dimensionados.

Motivações

Trabalhos Anteriores…

- Foco em:
  - servidores únicos;
  - ou servidores sem hierarquia.

Não Levam em Conta…

- A soma dos atrasos de cada estágio.
- Que as cargas em cada estágio não são uniformes.

Características da Solução

- Trabalha com frequências discretas.
- Modelo de controle com retro-alimentação.
- Pode ser combinado com outras estratégias:
  - Por exemplo, esquemas de desligamento de máquinas ociosas.
- Considera cada estágio como uma única máquina:
  - *Thin Pipeline*.
  - Supõe bom balanceamento de carga.

Arquitetura e Modelagem

Servidor *Multitier*

Processamento Sequencial

- Cada estágio pode ativar tarefas no estágio seguinte.
  - Dependendo do conteúdo.
- Não há paralelismo entre os estágios.
  - Estágios entram em ação de forma sequencial.

Caracterização do Atraso

$$D = \sum_{i}^{N} D_i$$

- D: atraso total do pedido.
- D_i: atraso devido ao estágio i do servidor.

$$D_i = D_i^{CPU} + D_i^{block}$$

- D_i^{CPU}: atraso de processamento.
- D_i^{block}: atraso de operações bloqueantes.
  - I/O, por exemplo.

Se D_i^{block} domina o atraso total...

- Pode-se reduzir a frequência de CPU de maneira mais agressiva.

Primeiro Algoritmo

Hipótese

- Atraso de processamento em um estágio é uma função convexa da utilização da CPU.
- Em outras palavras:
  - Quanto maior a utilização, mais acentuado é o aumento no atraso.
- Hipótese razoável, na prática.

Procedimento

- Se o atraso fim-a-fim excede um limiar máximo:
  - Aumente a frequência do estágio com maior carga.
- Se o atraso fim-a-fim é menor que um limiar mínimo:
  - Reduza a frequência daquele com menor carga.

Primeiro Algoritmo: Análise

Por Causa da Convexidade…

- Aumentar a frequência do estágio mais ocupado implica a máxima redução no atraso.
- Reduzir a frequência do estágio menos ocupado implica o mínimo aumento no atraso.

Simplicidade do Algoritmo

- Não é ótimo.
  - Assume que a economia de energia é proporcional à utilização.
- Mas depende apenas:
  - da utilização de cada CPU.
  - da atraso fim-a-fim de cada pedido.

Condições de Otimalidade

$$W_1 H(U_1) = W_2 H(U_2) = \cdots = W_N H(U_N)$$

Onde...

- $W_i = \left(\frac{A_i \lambda_i^n}{T_i}\right)$
- $H(U_i) = \frac{(1-U_i)^2}{U_i^{n+1}}$
- U_i é a utilização no estágio i.
- λ_i é a taxa de chegada de trabalho no estágio i.
- T_i é o tempo médio de serviço do estágio i.
- A_i e n vêm da equação:

$$P_i = A_i f_i^n + B_i$$

Segundo Algoritmo

- Versão ótima do primeiro algoritmo.
- Periodicamente, cada estágio anuncia sua utilização média.
- Também periodicamente, o primeiro estágio anuncia o atraso médio fim-a-fim D.
- Cada estágio:
  - Calcula $W_i H(U_i)$ para todos os estágios.
  - Se $D > a_{hi} L$, aumenta (para a próxima disponível) a frequência do estágio com menor $W_i H(U_i)$.
  - Se $D < a_{lo} L$, reduz (para a próxima disponível) a frequência do estágio com maior $W_i H(U_i)$.
    - L: prazo máximo.
    - a_{hi} e a_{lo}: limiares (normalizados).

Segundo Algoritmo: Análise

Similar ao Primeiro

- Quando o atraso médio é alto, prazos são perdidos.
  - Aumenta-se a frequência de um estágio.
- Quando o atraso médio é baixo, recursos ficam ociosos.
  - Reduz-se a frequência de um estágio.

Diferenças

- Critério de escolha do estágio:
  - Utilização x $W_i H(U_i)$.

Lidando com Múltiplas Classes

Múltiplas Classes de Pedidos

- Diferentes prazos (L_i).
- Diferentes atrasos fim-a-fim (D_i).

Alterações no Algoritmo

- Se $\exists j : D_j > a_{hi} L_j$, aumenta a frequência do estágio com menor $W_i H(U_i)$.
- Se $\exists j : D_j < a_{lo} L_j$, reduz a frequência do estágio com maior $W_i H(U_i)$.

Escolhendo a_{hi}

- É escolhido de forma que:

$$P(D[k+1] > L | D[k] < a_{hi} L) \leq r$$

- $D[k+1]$ e $D[k]$ são as medidas de atraso no instante atual e no próximo.
- r é a taxa tolerável de perda de prazos.

Escolhendo a_{lo}

- Seleção baseada no tamanho de tolerância $(a_{hi} - a_{lo})$.
- Evitar que o intervalo seja pequeno demais.
  - Pode causar oscilações.
  - Exemplo:
    - Uma redução de frequência aumenta muito o atraso médio.
    - O atraso cai abaixo do limiar.
    - Novo ajuste é necessário.

Escolha do Período de Amostragem

Períodos Muito Longos

- Reação do algoritmo é muito lenta.
- O sistema pode perder muitos pedidos ou desperdiçar energia.

Períodos Muito Curtos

- Modificações são muito frequentes.
- Método pode se tornar instável.

Conclusão

- Período deve ser longo o suficiente para que o último ajuste se reflita na amostra corrente.

Implementação

Cenário

Hardware

- Laptops com processador AMD.
- Frequências variando de 532 MHz a 1529 MHz.
- Chaveamento em 100 microssegundos.

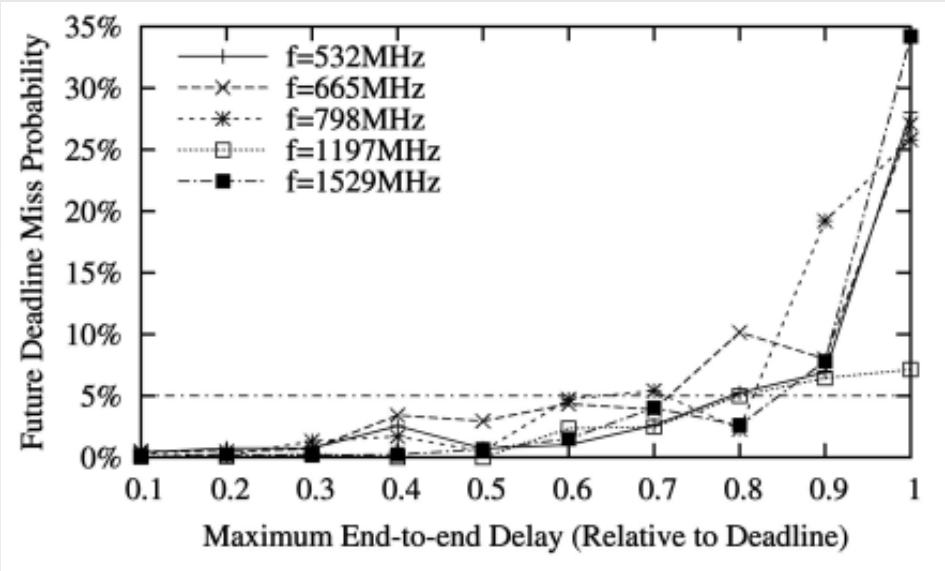
Duas Arquiteturas de *Software*

- TPC-W.
  - Benchmark baseado em transações *Web*.
- Servidor Apache.
  - Primeiro estágio: conteúdo HTML estático.
  - Segundo estágio: CGIs.
  - Terceiro estágio: banco de dados MySQL.

Seleção de Parâmetros (1/2)

Escolha de a_{hi}

- Taxa tolerada de perda de prazos: 5%.
- Experimentalmente, foi calculada a probabilidade condicional.
- Optou-se pelo valor $0,7$.



Seleção de Parâmetros (2/2)

Escolha de a_{lo}

- Concluiu-se que $0,3$ é um intervalo suficiente.
  - Não permite oscilações do método.
- Escolha específica para os processadores considerados.

Escolha do Período de Amostra

- $200ms$.
- Poucas requisições, considerando capacidade do sistema.
- Ao mesmo tempo, baixo *overhead*.
  - Apenas 5 execuções por segundo.

Outras Considerações Sobre Parâmetros

Período de Amostragem

- Em baixa carga, $200ms$ é muito curto.
  - Poucos pedidos.
  - Média não é representativa.
- Neste caso, período de amostragem sobe para 4 a 10 segundos.

Falta de Sincronia

- Problema não é serio, assumindo:
  - Utilização média e atraso médio não mudam muito rápido.
- No pior caso, não há consenso e dois estágios agem.
  - Apenas aumenta o efeito do ajuste.

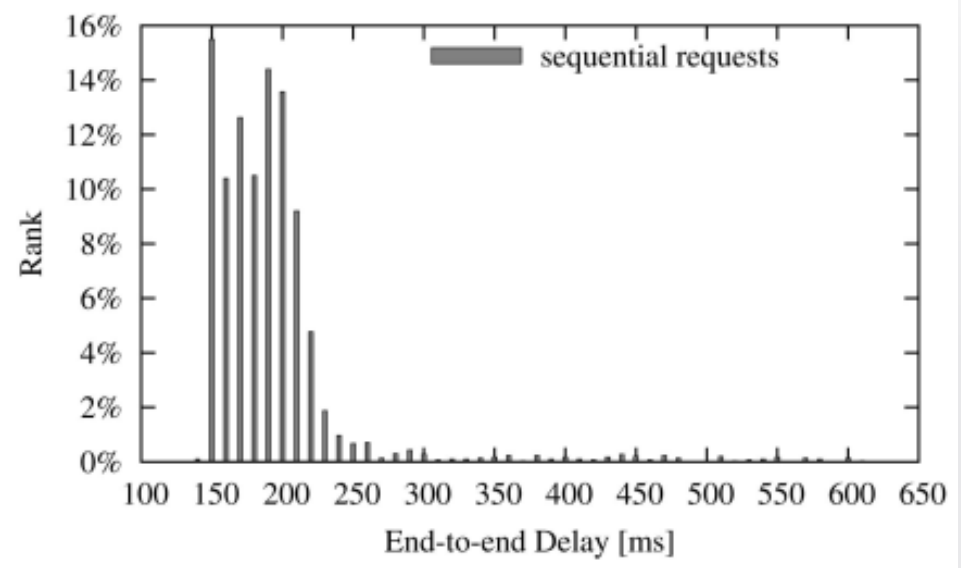
Avaliação

Propostas Avaliadas

Quatro

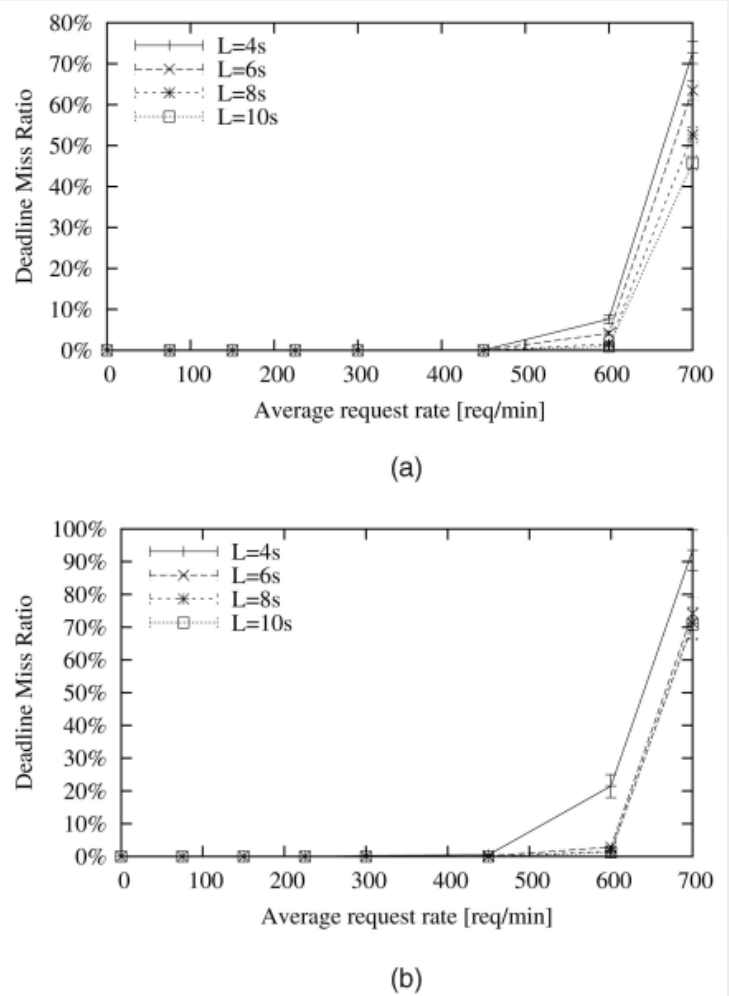
- *Baseline*.
  - Sempre a maior frequência.
  - Processador desligado quando ocioso.
- *Independent DVS*.
  - Decisões locais, apenas.
  - Algoritmo PAST.
- *Feedback DVS*.
  - Algoritmo simplificado.
- *Weighted Feedback DVS*.
  - Algoritmo ótimo.

Carga Sintética



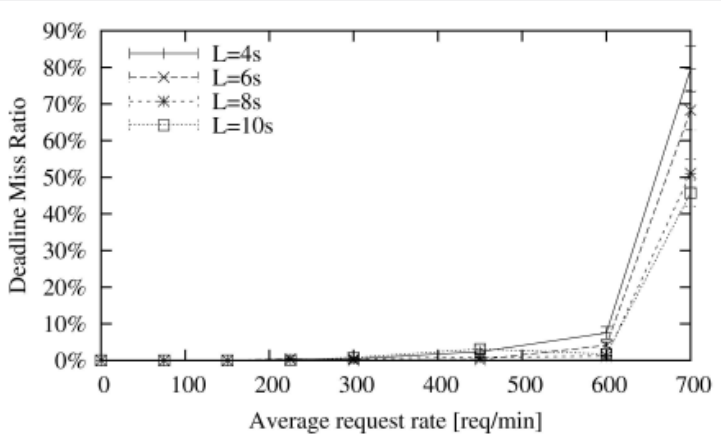
- Maior parte leva 300 ms ou menos.
  - Sem concorrência.

Perda de Prazos X Carga (1/2)

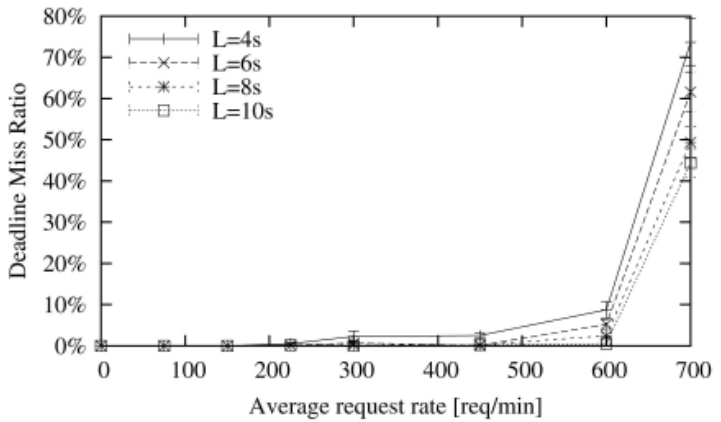


- No *baseline*, saturação começa em 450 req/min.
- *Independent DVS*: similar.
  - Tem crescimento mais rápido para L=4.

Perda de Prazos X Carga (2/2)



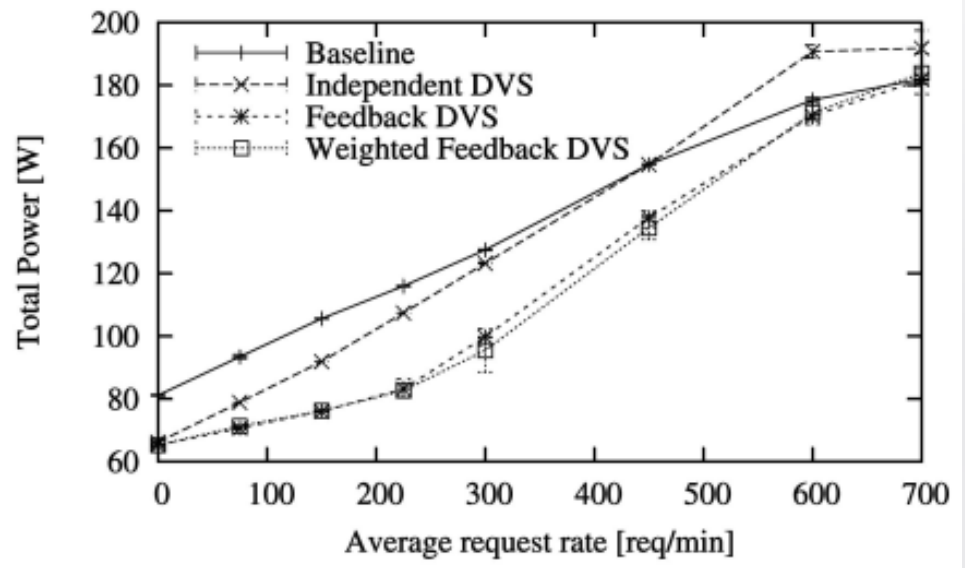
(a)



(b)

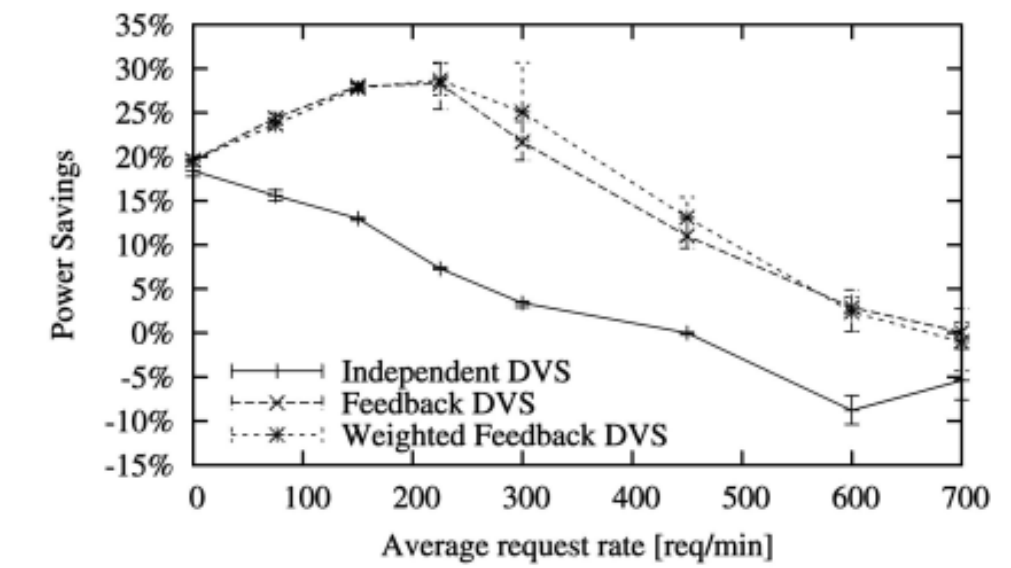
- Saturação começa mais cedo.
  - Mas ainda dentro da tolerância de 5%.

Consumo X Carga (1/2)



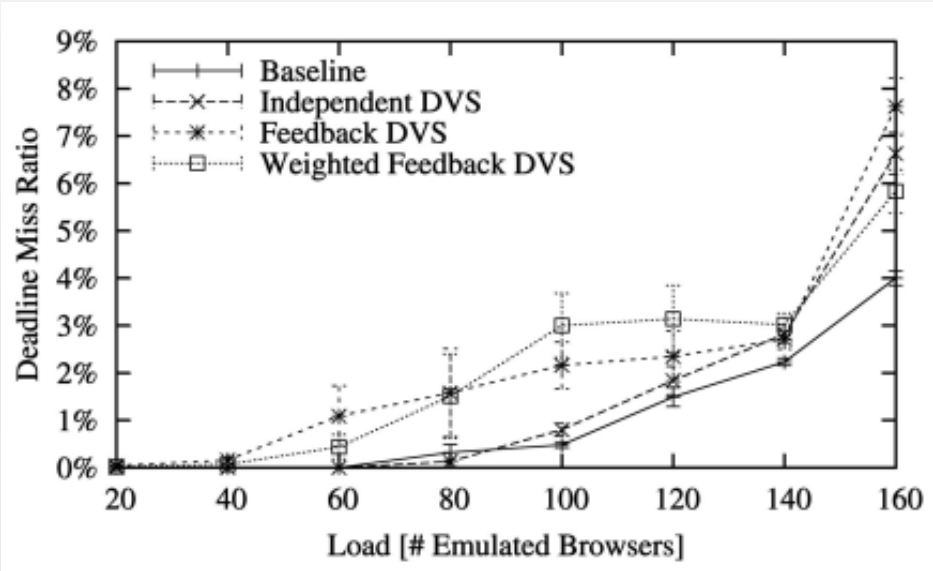
- *Baseline* economiza 100 W em relação ao pico.

Consumo X Carga (2/2)



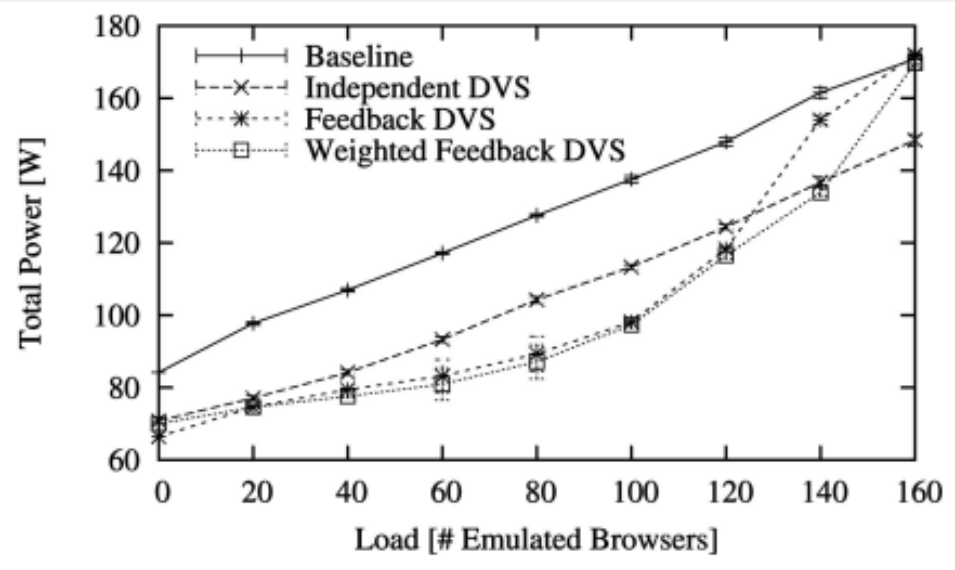
- Em carga média, *Feedback* resulta em 30% de economia.
  - Versão simplificada é ligeiramente pior.
- Em baixa carga, CPU quase sempre inativa.
- Em alta carga, frequências são sempre altas.

Perda de Prazos X Carga (TPC-W)



- Todas as políticas dentro da tolerância até 140 clientes.
  - Com 160, apenas o *baseline*.

Consumo X Carga (TPC-W)



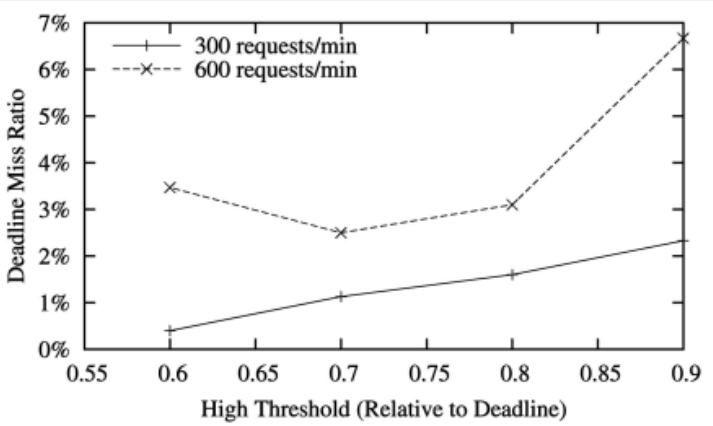
- Em carga média, bons resultados do *Feedback*.
- Em carga alta, *Independent DVS* consome menos.
  - Mas com maior taxa de perda de prazos.

Número de Mudanças de Frequência

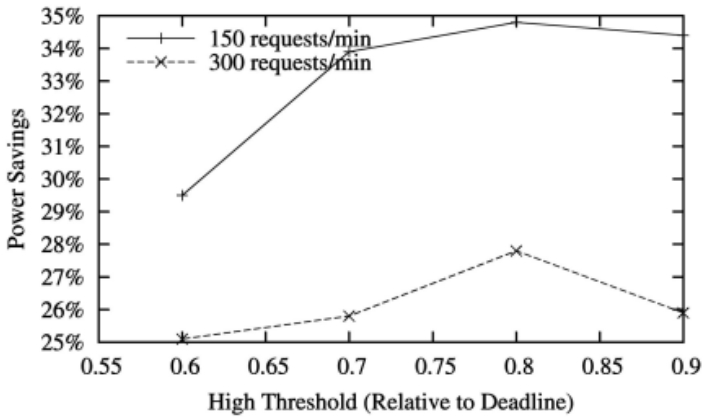
Load	*Adjustments/minute*
20	0.5
40	1.3
60	4.1
80	9.2
100	20.8
120	25.0
140	18.8
160	3.5

- Segundo os autores, grande variabilidade mesmo em carga constante.
  - O que causa variabilidade?
  - Pedidos heterogêneos?

Sensibilidade: Limiar Superior



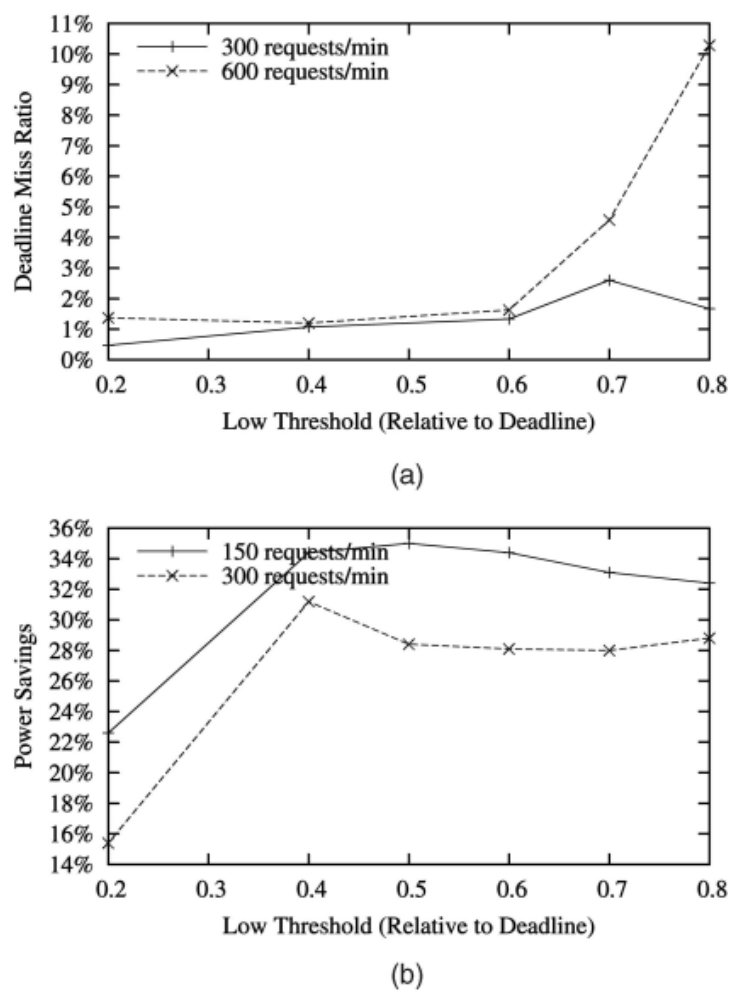
(a)



(b)

- 0,7 é uma boa escolha:
  - menor perda de prazos que os maiores.
  - maior economia que os menores.

Sensibilidade: Limiar Inferior



- 0,6 (*deadzone* de 0,3), tem baixa perda de prazos.
- A partir de 0,4, boa economia.

Conclusões

Conclusões

Experimentos

- O esquema proposto mantém baixa perda de prazos, com economia "não trivial".
- A economia ótima não ocorre com utilizações balanceadas.
  - Mas com as utilizações ponderadas.
- Mas o algoritmo simplificado traz bons resultados.
  - Cerca de 30% de economia.

Trabalhos Futuros

- Mais servidores por estágio.
- Incorporar controle de outros estados do sistema.