A Política de *Piggybacking* S^2

Roberto A. Façanha *Nelson L. S. Fonseca*

Relatório Técnico IC–98-07

Abril de 1998

A Política de Piggybacking S^2

Roberto A. Façanha *Nelson L. S. Fonseca*

Instituto de Computação
UNICAMP
Caixa Postal 6176
13083-970 — Campinas — SP
{facanha,nfonseca}@dcc.unicamp.br

Abstract

In a video-on-demand system, users expect to watch the film right after its selection. However. such a short response time is feaseble only if there is available bandwidth. Several techniques have been proposed to reduce the huge bandwidth demand on video servers. In this paper, we introduce the Piggybacking policy S^2 and an heuristic to reduce the complexity of generate the tree of superimposed video streams.

Resumo

Em ambientes de vídeo sob demanda, espera-se que após seleção do filme a exibição do mesmo inicie em curto intervalo de tempo. Isto é possível se existir banda passante disponível para satisfazer às requisições dos usuários. Diversas técnicas foram propostas com o objetivo de reduzir a grande demanda de banda passante em servidors de vídeo. Neste artigo, a política S^2 e uma heurística para a redução da complexidade de geração da árvore de mesclagens de fluxos de vídeo são introduzidas e analizadas.

1 Introdução

As aplicações de vídeo, como por exemplo Vídeo sob Demanda (*Vídeo on Demand* — VoD), estão entre as de maior potencial comercial nas futuras Redes Digitais de Serviços Integrados de Faixa Larga (RDSI-FL). Porém, dado que estas aplicações demandam uma grande quantidade de banda passante, sua disponibilização em larga escala requer a utilização de técnicas de redução da demanda de banda passante. Técnicas de redução da demanda de banda passante levam em consideração a probabilidade de um conjunto de requisições por vídeos populares chegarem ao sistema

dentro de um intervalo de tempo relativamente curto possibilitando o fornecimento de um único fluxo para as diversas requisições.

A técnica de *Batching* [1] dispara um fluxo ao final de um determinado intervalo de tempo, denominado Janela de Batching, agrupando todas as requisições pendentes no sistema por um certo vídeo até aquele instante. A grande desvantagem desta técnica é o retardo introduzido entre a requisição e a exibição do filme, podendo resultar no abandono do usuário.

A técnica de *Piggybacking* [2, 3, 4] baseia-se no fato de que alterações da ordem de 5% na taxa de exibição não são perceptíveis aos usuários. Desta forma, as requisições são atendidas imediatamente e o fluxo unificado é obtido através da superposição de fluxos de-sincronizados, ou seja, altera-se a taxa de exibição dos fluxos de vídeo de tal forma que estes venham a exibir um mesmo quadro de filme em um determinado instante descartando um dos fluxos após a sincronização.

Este artigo introduz a política S^2 [5], uma generalização da política Algoritmo *Snapshot* [3, 4], que considera a mesclagem dos fluxos resultantes de intervalos *Snapshot*. Adicionalmente, propõe-se uma heurística para a redução da complexidade de construção da árvore de mesclagens de fluxos de vídeo.

O restante deste artigo está organizado como segue. Na seção 2 a política *Snapshot* é apresentada. As seções 3 e 4 introduzem, respectivamente, a política S^2 e a heurística proposta. A seção 5 apresenta as conclusões. Finalmente, os apêndices A e B apresentam demonstrações e o código da heurística.

2 A Política Algoritmo *Snapshot*

Dentre as políticas de *Piggybacking*, a política *Snapshot* [3, 4] busca minimizar o número de quadros exibidos por um conjunto de fluxos de vídeo.

A computação realizada pelas políticas de *Piggybacking* pode ser vista como uma árvore binária nas quais as folhas correspondem aos fluxos, os nós intermediários são as mesclagens e a raiz é a mesclagem final formando o fluxo resultante do conjunto (Figura 1). Portanto, o número de possíveis árvores formadas a partir de um conjunto de fluxos constitui-se no número de políticas de *Piggybacking* potencialmente ótimas e é dado pelo $(n-1)$-ésimo número de **Catalan**, ou seja:

$$Catalan(n-1) = \frac{(2n-2)!}{(n-1)!n!} \tag{1}$$

o que implica que o número de árvores cresce muito rapidamente inviabilizando a busca exaustiva da estratégia ótima e sua árvore binária correspondente.

Deste modo, a política *Snapshot* constrói a árvore ótima de mesclagem de fluxos de vídeo da seguinte forma: considere um conjunto de n fluxos de um mesmo vídeo (o qual é composto de L quadros) e suas posições dadas por $f_1, f_2, \ldots, f_n$, em que

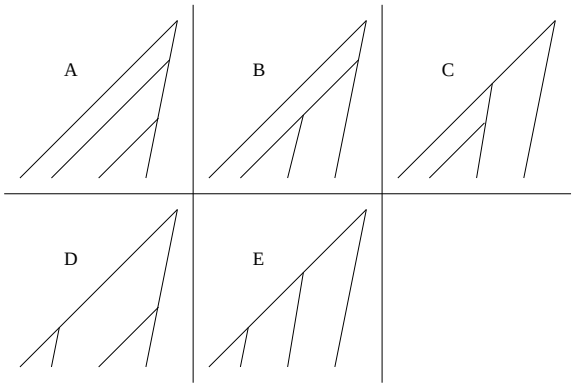


Figura 1: Possíveis árvores de mesclagens para quatro fluxos.

$f_1 \geq f_2 \geq \cdots \geq f_n$ em um determinado instante de tempo T. Definem-se as "velocidades" S_{min} e S_{max}, dadas em quadros por segundo, como as taxas mínima e máxima, respectivamente, de apresentação dos quadros dos fluxos do vídeo. Sejam i e j dois fluxos entre 1 e n, com $i \leq j$. Denota-se $P(i,j)$ como a posição de mesclagem (em quadros), na qual os fluxos i e j apresentam o mesmo quadro. Uma vez que o fluxo i possui velocidade S_{min} e o fluxo j S_{max}, a posição de mesclagem é dada por:

$$P(i,j) = f_i + \frac{S_{min} \cdot (f_i - f_j)}{S_{max} - S_{min}} \tag{2}$$

Seja $C(i,j)$ o custo de uma política e $\mathcal{A}(i,j)$ sua árvore binária correspondente. O custo para cada fluxo é dado por $C(i,i) = L - f_i$.

Para se obter o custo $C(i,j)$ mínimo é necessário que o princípio de otimalidade seja satisfeito, ou seja: *Existirá um fluxo k, com $i \leq k < j$, tal que as subárvores à esquerda e à direita também são ótimas.* Estas subárvores são denotadas por $\mathcal{A}(i,k)$ e $\mathcal{A}(k+1,j)$. O custo da árvore que corresponde ao conjunto de fluxos $i,\ldots,j$ é dado por $C(i,k) + C(k+1,j) - \max(L - P(i,j), 0)$.

Portanto, a política ótima para os fluxos $i,\ldots,j$ possui subárvores $i,\ldots,k^*$ e $k^*+1,\ldots,j$, em que $k^* = \text{argmin}_{i \leq k < j}\{C(i,k) + C(k+1,j) - \max(L - P(i,j), 0)\}$.

Deste modo, o custo do conjunto dos n fluxos, $C(1,n)$, pode ser calculado de forma *bottom-up* através de um algoritmo de programação dinâmica.

Assim como as demais políticas de *Piggybacking*, a política *Snapshot* atribui velocidades aos fluxos que são disparados pelo sistema dentro de um intervalo (neste caso o Intervalo *Snapshot* que é denotado por I). O intervalo *Snapshot* é dado por $I = W/S_{max}$, em que W é o valor ótimo da janela de mesclagem derivado para a política mesclagem simples generalizada [3, 4]. Dentro do intervalo I, a política *Snapshot* comporta-se como a política mesclagem simples. Ao final do intervalo os

procedimentos de otimização descritos anteriormente são aplicados.

O dimensionamento da janela de mesclagem é um fator importante porque quando o tamanho da janela é grande, um número maior de fluxos pode ser mesclado em um único fluxo, porém as mesclagens ocorrem próximas do final do vídeo; por outro lado, quando o tamanho da janela é muito pequeno, as mesclagens tendem a ocorrer próximas do ínicio do vídeo, porém, em número reduzido. Pode-se perceber que ambas as situações não favorecem à redução do número de quadros exibidos por um conjunto de fluxos. Aggarwal *et. al* [3, 4] apresentam um método analítico para otimizar o tamanho da janela de mesclagem o qual leva em consideração a taxa de chegada prevista, assumindo que as chegadas são modeladas por um processo de Poisson com parâmetro λ. Deste modo, o tamanho de W otimiza o número e a posição, relativamente ao tamanho do vídeo, em que mesclagens ocorrem; e, deste modo, otimiza a demanda de banda passante.

3 A Política S^2

A política *Snapshot* foi proposta de modo a minimizar o número de quadros exibidos dos fluxos contidos em um intervalo I, de tal modo que é necessário aguardar o final deste intervalo. A otimização realizada envolve apenas os fluxos que chegaram durante o último intervalo *Snapshot*. Pode-se verificar que, quanto menor o intervalo médio entre requisições (maiores taxas de chegadas) as janelas ótimas de mesclagem serão cada vez menores, podendo haver uma ou várias janelas ótimas contidas numa mesma janela máxima de mesclagem. A possibilidade de mesclagem entre dois fluxos separados por no máximo W_m quadros, em que W_m é janela máxima de mesclagem, permite que ganhos adicionais aos da política *Snapshot* possam ser obtidos. Assim sendo, propõe-se a política S^2 que busca otimizar o número de quadros exibidos por um conjunto de fluxos disparados pelo sistema numa janela máxima alterada de mesclagem, W'_m, $(W'_m \leq W_m)$ e não somente no intervalo I. A janela máxima alterada é constituída por um número inteiro de janelas ótimas, ou seja, a janela máxima alterada possui $\lfloor W_m/W \rfloor$ janelas ótimas. Em outras palavras, a política S^2 introduz um segundo nível de mesclagens, isto é, a mesclagem das resultantes dos intervalos I da política *Snapshot*.

O funcionamento da política S^2 é como se segue: aplica-se primeiramente o algoritmo *Snapshot* sobre os fluxos que chegam ao sistema nos intervalos *Snapshot* (como proposto originalmente) e em seguida, aplica-se novamente o *Snapshot* sobre os fluxos resultantes dos fluxos dos intervalos ótimos. É interessante enfatizar que, de acordo com a definição da própria política *Snapshot*, atribui-se a velocidade S_{max} a estes fluxos resultantes com exceção daquele gerado pela primeira janela ótima contida em W'_m. Outro aspecto importante é que ao contrário do que ocorre com o primeiro nível de otimização, o ponto de aplicação do procedimento de otimização não ocorre ao final

de intervalos de duração fixa. Estes intervalos são denotados por I_{S^2} e sua duração é determinada pelo padrão das requisições por vídeos em cada janela W'_m (Figura 2).

Uma generalização natural da política S^2 seria considerar n níveis de otimização. No entanto, os ganhos obtidos com a implantação destes níveis seriam praticamente nulos dado que os fluxos nestes níveis estariam separados por valores bem próximos a W_m quadros (ou ainda maiores) o que implica em mesclagens próximas do final do vídeo. Portanto, a introdução de níveis extras de mesclagens não proporciona reduções efetivas de banda passante.

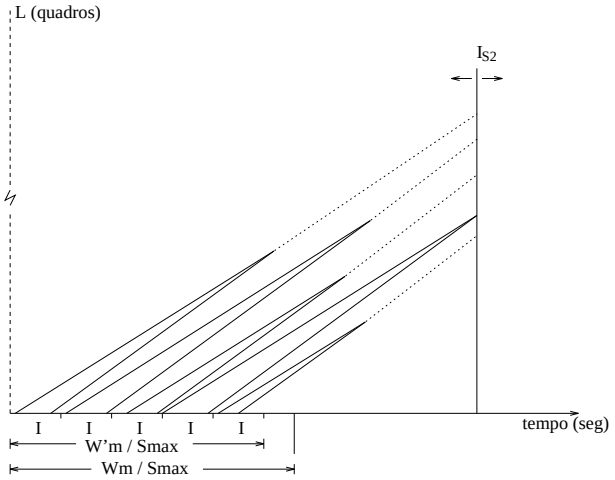


Figura 2: Possível situação da política S^2 na qual o último fluxo resultante gerado não corresponde ao fluxo resultante da última janela ótima contida na janela W'_m.

Para avaliar o impacto da introdução de um segundo nível de otimização, foi realizado um estudo comparativo através de simulação entre as políticas S^2, *Snapshot* original e a política *Snapshot* Global, que considera todas as requisições de um mesmo filme sem dividir o tempo em intervalos *Snapshot*. A função objetivo utilizada neste estudo difere da considerada em [4], a qual contabiliza apenas o número de quadros exibidos após o intervalo I. A função objetivo utilizada neste artigo contabiliza o número total de quadros apresentados por um conjunto de fluxos, ou seja, reflete a otimização realizada no intervalo I, assim como a realizada posteriormente ao intervalo. A função objetivo é dada por: $C(i,i) = L$ para um fluxo i e para um conjunto de fluxos $1, \ldots, n$:

$$C(1,n) = Quadros_I + \sum_{k=\text{argmin}_{i \le k < j}} (C(i,k) + C(k+1,j) - \max(L - P(i,j), 0)) \quad (3)$$

em que $Quadros_I$ é o somátorio dos quadros apresentados pelos m fluxos descartados, $Q(l)$, para $l = 1, \ldots, m$, antes do final dos intervalos *Snapshot*, dado por:

$$Quadros_I = \sum_{l=1}^{m} Q(l) \tag{4}$$

3.1 Resultados Numéricos

Os resultados apresentados neste artigo foram obtidos via simulação de eventos discretos. Utiliza-se o método de replicações para calcular um intervalo de confiança com nível de confiança de 95%, porém nos gráficos plotam-se apenas os valores médios para facilitar a visualização. Assume-se que as chegadas são modeladas por um processo de Poisson e utiliza-se $S_{min} = 28.5$ e $S_{max} = 31.5$ em quadros/segundos. Os gráficos mostram a redução percentual média no número de quadros apresentados em decorrência da aplicação das políticas de *Piggybacking*.

Foram realizados três experimentos: *i*) análise das políticas sob diversas taxas de chegadas, *ii*) análise de sensibilidade da política S^2 em relação ao tamanho da janela de mesclagem e *iii*) análise do efeito da duração de um filme e da taxa de chegadas sobre a política S^2.

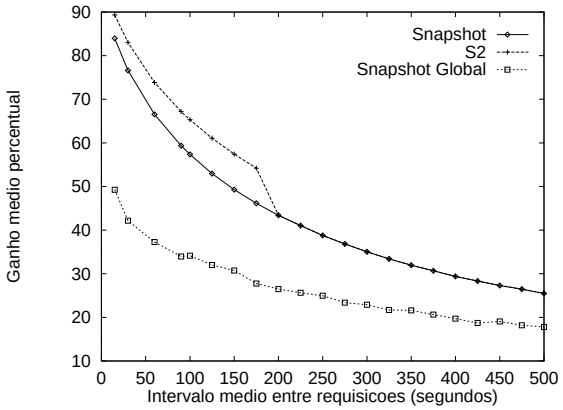


Figura 3: Comparação entre as políticas *Snapshot* original, S^2 e *Snapshot* Global.

A Figura 3 ilustra o comportamento das políticas sob diferentes taxas de chegada, isto é, varia-se o intervalo médio de requisição de 15 a 500 segundos. Pode-se verificar que, à medida que os intervalos médios entre chegadas assumem valores maiores, os ganhos com adoção de *Piggybacking* são reduzidos. Para altas taxas, a introdução de um segundo nível de otimização propicia ganhos de até 8% maiores que os alcançados pela política *Snapshot*.

A Figura 4 mostra o comportamento da política S^2 em função do tamanho da janela de mesclagem para o intervalo médio de chegadas de 30 segundos e a duração

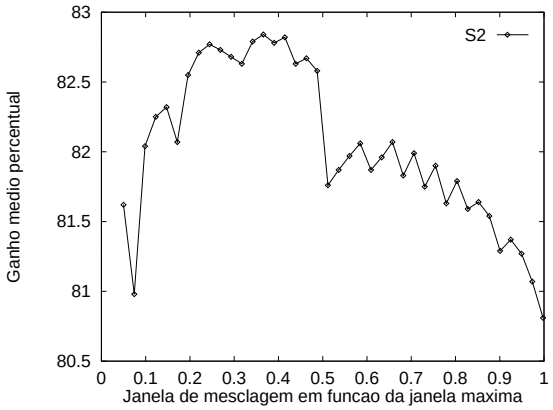


Figura 4: Influência do tamanho da janela sobre a política S^2.

do vídeo em 2 horas. Pode-se verificar que a política S^2 é insensível à variação da janela de mesclagem. Isto se deve ao fato de que o segundo nível de otimização "recupera" eventuais perdas devido à variação do tamanho da janela de mesclagem. Em outras palavras, quando a janela de mesclagem assume valores muito pequenos, ocorrem poucas mesclagens no primeiro nível de otimização, o que implica uma pequena redução no número de quadros exibidos. Porém, uma vez que existe um segundo nível, os fluxos resultantes do primeiro nível de mesclagens ainda podem ser superpostos, proporcionando assim uma redução ainda maior no número de quadros apresentados. À medida que a janela de mesclagem assume valores próximos aos da janela máxima, os ganhos são obtidos já no primeiro nível de otimização.

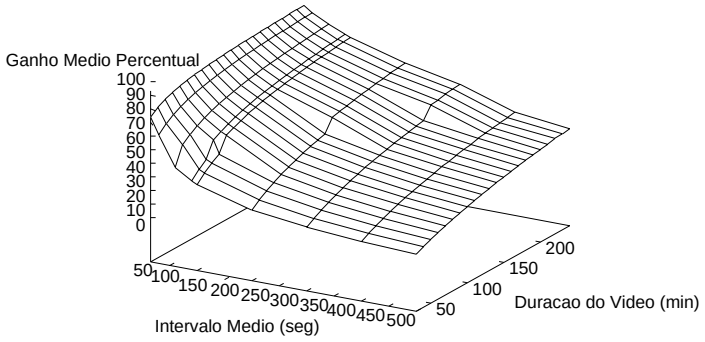


Figura 5: Influência da duração do vídeo sobre a política S^2.

A Figura 5 ilustra o efeito da variação da duração do vídeo em conjunto com à variação do intervalo médio entre requisições sobre a política S^2. Neste caso, vídeos com duração de 30 minutos até 4 horas são submetidos a diversos intervalos médios

entre chegadas, os quais assumem valores que variam de 15 a 500 segundos. Nota-se que obtém-se maiores reduções no número de quadros exibidos para filmes mais longos e submetidos a maiores taxas de chegada. A redução na demanda de banda passante varia desde 9% para vídeos com 30 minutos de duração e intervalo médio entre requisições de 500 segundos chegando até a 93% para filmes longos (4 horas de duração) com intervalo entre chegadas de 15 segundos.

4 Redução da Complexidade de Geração da Árvore Ótima de Mesclagens da Política *Snapshot*

Num sistema de VoD em que se emprega a técnica de *Piggybacking*, a política *Snapshot* constitui-se numa alternativa atrativa sob o aspecto de otimização da demanda de banda passante. Porém, a árvore ótima de mesclagens é construída por um algoritmo de programação dinâmica que possui complexidade $\Theta(n^3)$, no qual n representa o número de fluxos disparados num intervalo I. Adiciona-se a este fato o progresso contínuo dos fluxos paralelo à execução do algoritmo, ou seja, deseja-se que a árvore de mesclagens seja contruída antes que o contexto caracterizado pelas posições dos fluxos seja alterado.

Observou-se que a árvore de mesclagens pode ser construída de forma *Top Down*, diferentemente do algoritmo de programação dinâmica que a constrói de forma *Bottom Up*. Com vistas à redução da complexidade, elaborou-se uma heurística que utiliza a estratégia de divisão e conquista na construção da árvore de mesclagens. O princípio básico da heurística consiste em dividir sucessivamente o conjunto de fluxos a ser mesclado em duas partes computando-se os custos de cada subgrupo até que se obtenha a árvore de mesclagens. O critério de divisão utilizado leva em consideração os segmentos da árvore de mesclagem (número de quadros de vídeo dado pela posição de mesclagem) entre o primeiro fluxo e um fluxo intermediário (fluxo divisor) e entre este e o último fluxo. Em outras palavras, determina-se o fluxo que representaria o ponto de divisão do conjunto em subárvores potencialmente ótimas. Deste modo, para um conjunto de fluxos $i, \ldots, j$, existe um fluxo k, com $i \leq k < j$ que minimiza o módulo da diferença dos comprimentos dos segmentos $P(i,k)$ e $P(k,j)$, dado por:

$$k^* = \text{argmin}_{i \leq k < j}\{|P(i,k) - P(k,j)|\} \tag{5}$$

Após a determinação de k^*, o conjunto original de fluxos $i, \ldots, j$ é particionado nos subconjuntos $i, \ldots, k^*$ e $k^*+1, \ldots, j$, se $P(i,k^*) < P(k^*,j)$, caso contrário, $i, \ldots, k^*-1$ e $k^*, \ldots, j$, reaplicando-se o critério de divisão em ambos. O algoritmo considera dois casos particulares em cujas entradas possuem somente dois e três fluxos. Nestes casos simplifica-se a análise para se obter melhor desempenho. No primeiro caso, o algoritmo calcula apenas a posição de mesclagem dos fluxos, no segundo comparam-se os segmentos $P(i,i+1)$ e $P(i+1,j)$ e descarta-se o de maior valor.

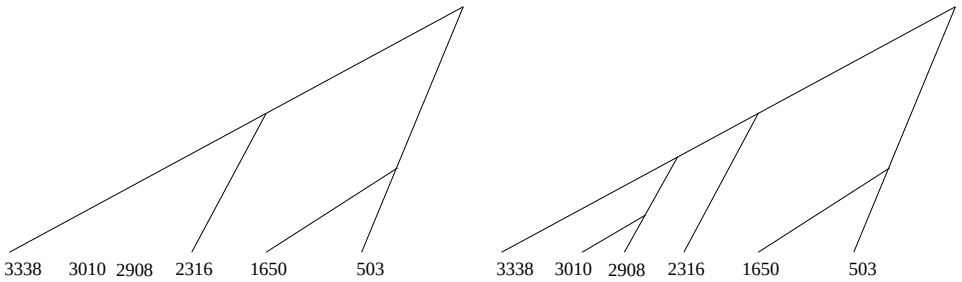
Deste modo, o custo para um conjunto de fluxos $i, \ldots, j$ é obtido através do somatório dos comprimentos dos segmentos da árvore de mesclagem construída, o qual representa o custo dos $n-1$ fluxos a menos do fluxo resultante, adicionado do número de quadros do fluxo resultante.

No processo de determinação do fluxo divisor (Equação 5) são necessárias $O(n)$ operações. Pode-se utilizar o artifício descrito a seguir para se reduzir o número de operações realizadas: "*O fluxo divisor é o primeiro cujo segmento de mesclagem do fluxo inicial com o posterior do divisor é maior que o segmento de mesclagem do fluxo posterior do divisor com o último fluxo*". No pior caso, $k = j-1$, $O(n)$ operações são realizadas.

Teorema 4.1 *A heurística BuildTree possui complexidade* $O(n^2)$.

Prova: Apêndice A.

Para uma melhor compreensão da heurística, considere o exemplo a seguir. Suponha que em um determinado intervalo *Snapshot* seis fluxos são iniciados e que, ao final do mesmo, suas posições sejam: 3338, 3010, 2908, 2316, 1650 e 503. Sejam i, o primeiro fluxo; k, o fluxo divisor e j, o último fluxo do conjunto, particionam-se os seis fluxos nos conjuntos $(1 \Longleftrightarrow 4)$ e $(5 \Longleftrightarrow 6)$, $(i = 1,\ k = 4,\ j = 6)$, dado que o quarto fluxo é o primeiro fluxo em que $P(i, k+1) \geq P(k+1, j)$, compondo-se a árvore da Figura 6-a. Em seguida, reaplica-se o algoritmo sobre os subconjuntos $(i = 1, \ldots, j = 4)$ e $(i = 5$ e $j = 6)$. A Figura 6-b ilustra a árvore de mesclagens final contruída pelo algoritmo (neste caso, igual a arvore gerada pelo algoritmo de programação dinâmica).



(a) Primeiro Particionamento (b) Árvore Final

Figura 6: Construção da árvore de mesclagens pela heurística.

Em nossos experimentos a heurística foi implementada utilizando-se busca sequencial do fluxo k através do conjunto $i, \ldots, j$. A utilização de busca binária em substituição à sequencial implica em uma complexidade de $O(n \log n)$. No Apêndice B, apresenta-se a heurística implementada em linguagem C.

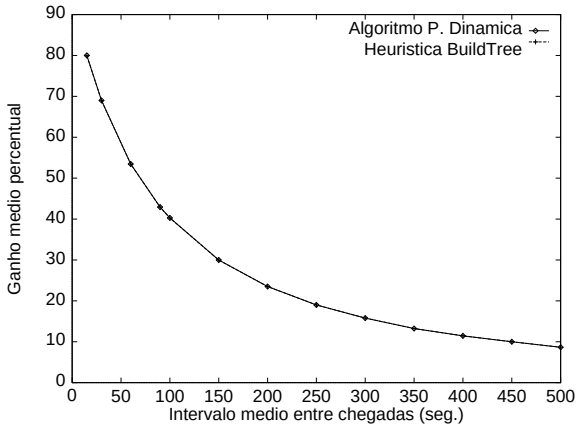


Figura 7: Comparação dos custos computados pela heurística e pelo algoritmo de programação dinâmica.

4.1 Resultados Numéricos

A heurística descrita constitui-se numa aproximação da solução ótima, fazendo-se necessário avaliar a precisão em relação à esta. Com este objetivo, utilizaram-se ambos algoritmos ao final de intervalos *Snapshot* de modo a se comparar os custos das árvores de mesclagens geradas. Neste experimento as requisições foram modeladas através de um processo de Poisson e o intervalo médio entre requisições variou de 15 a 500 segundos. Para cada taxa de chegadas o número de intervalos *Snapshot* simulados totalizou 10^5.

A Figura 7 mostra as curvas do ganho médio percentual dos algoritmos de programação dinâmica e da heurística. Pode-se observar que a heurística *BuildTree* é bastante precisa em relação à solução ótima. Isto se deve ao fato de que, na maioria dos casos, a heurística obtém a solução ótima, nos demais a diferença percentual mostra-se muito pequena assumindo valores no máximo iguais a 5.6%.

No gráfico da Figura 8, instâncias de 1 até 25 fluxos (observadas em outros experimentos) foram fornecidas como entrada para a computação dos tempos de processamento de ambos os algoritmos. Observou-se em todos os casos que o tempo de execução da heurística foi no máximo igual ao tempo do algoritmo de programação dinâmica, nunca excedendo-o. Este experimento foi realizado num computador PC 486 Dx-4 100MHz com 16MB de memória sobre Linux 2.0.0.

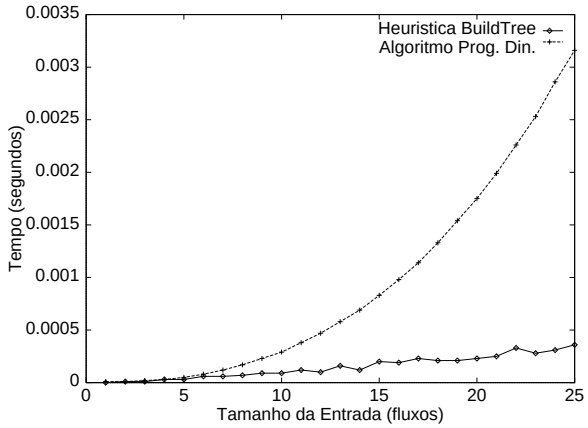


Figura 8: Comparação entre os tempos de execução da heurística e algoritmo de programação dinâmica.

5 Conclusões

Este artigo introduz a política S^2, a qual acrescenta um nível de otimização em relação à política *Snapshot* e uma heurística aproximada, que busca a redução da complexidade da construção da árvore de mesclagens dos fluxos disparados num intervalo *Snapshot*. Simulações foram feitas e verificou-se que a política S^2 proporciona resultados superiores aos obtidos pela política *Snapshot*, além disso, nota-se que a heurística proposta é bastante precisa em relação à solução ótima (algoritmo de programação dinâmica), porém com complexidade inferior.

Agradecimentos

Este trabalho foi parcialmente financiado pelo CNPq/Protem-III ALMADEM, CNPq e Fapesp, processo Nº 96/09739-1.

Referências

[1] Dan, Sitaram, and Shahabuddin. Dynamic batching policies for an on-demand video server. *Multimedia Systems*, 4:112–121, 1996.

[2] Golubchik, C. S. Lui, and Muntz. Reducing I/O Demand in Video-on-Demand Storage Servers. *ACM Sigmetrics*, pages 25–36, 1995. Ottawa, Canadá.

[3] Aggarwal, Joel L. Wolf, and Yu. On Optimal Piggyback Merging Policies for Video-on-Demand Systems. Technical Report RC 20337 (90078), IBM Research Division, Fevereiro 1996.

[4] Aggarwal, Joel L. Wolf, and Yu. Adaptive Piggybacking Schemes for Video-on-Demand Systems. Technical Report RC 20635 (91350), IBM Research Division, Novembro 1996.

[5] Façanha, Nelson Fonseca, and Rezende. Reduzindo a Demanda de Banda Passante em servidores de Vídeo. In *II Seminário Franco-Brasileiro em Sistemas Informáticos Distribuídos: Arquiteturas Multimídia para as Telecomunicações*, pages 308–319, Novembro 1997.

A Complexidade da Heurística *BuildTree*

$$
\begin{aligned}
T(n) &= k + T(k) + T(n-k) \\
T(n) &= (n-1) + T(n-1) + T(1), \qquad \text{fazendo } T(1) = c_1 \\
&= (n-1) + [(n-2) + T(n-2) + c_1] + c_1 = \\
&\quad 2 \times n - 3 + 2 \times c_1 + T(n-2) \\
&= 2 \times n - 3 + 2 \times c_1 + (n-3) + T(n-3) + c_1 = \\
&\quad 3 \times n - 6 + 3 \times c_1 + T(n-3) \\
&= 3 \times n - 6 + 3 \times c_1 + (n-4) + T(n-4) + c_1 \\
&= 4 \times n - 10 + 4 \times c_1 + T(n-4) \\
&= x \times n - \sum_{i=1}^{x} i + x \times c_1 + T(n-x), \qquad \text{no pior caso } x = n \\
&= n^2 - \sum_{i=1}^{n} i + n \times c_1 + c_2, \qquad \text{fazendo } T(n-n) = T(0) = c_2 \\
&= n^2 - \frac{n \times (n-1)}{2} + n \times c_1 + c_2 \\
&= n^2 - \frac{n^2}{2} + \frac{n}{2} + n \times c_1 + c_2 \\
&= \frac{n^2}{2} + \left(c_1 + \frac{1}{2}\right) \times n + c_2 \\
T(n) &= O(n^2)
\end{aligned}
$$

□

B Código da Heurística *BuildTree*

Algoritmo 1 Heurística de Construção da Árvore de Mesclagem.

```
int BuildTree(int Posicao[], int i, int j) {
    int n = j - i + 1, // Número de fluxos.
        k = i,         // Fluxo divisor.
        Custo = 0,     // Custo da árvore.
        MPik,          // Posição de Mesclagem dos fluxos i e k.
        MPkj;          // Posição de Mesclagem dos fluxos k + 1 e j.
    switch(n) {
        case 0:
        case 1:  return(0);
        case 2:  return(MergePosition(Posicao[i], Posicao[j]));
        case 3:  MPik = MergePosition(Posicao[i], Posicao[i + 1]);
                 MPkj = MergePosition(Posicao[i + 1], Posicao[j]);
                 return(((MPik  > MPkj) ? MPkj : MPik) +
                        MergePosition(Posicao[i], Posicao[j]));
        default: while (MergePosition(Posicao[i], Posicao[k + 1]) <
                        MergePosition(Posicao[k + 1], Posicao[j]))
                   k++;
                 Custo  = BuildTree(Posicao, i, k);
                 Custo += BuildTree(Posicao, k + 1, j);
                 Custo += MergePosition(Posicao[i], Posicao[j]);
                 return(Custo);
    }
}
```