Gerenciamento de Banda Passante em Servidores de Vídeo

Roberto de Almeida Façanha
Autor

Nelson Luís Saldanha da Fonseca
Orientador

Universidade Estadual de Campinas
Instituto de Computação
Unicamp
Caixa Postal 6176
13083-970 — Campinas, SP
e-mail: {facanha,nfonseca}@dcc.unicamp.br

Abstract

Providing video services in large scale demands huge amount of bandwidth. Several techniques have been proposed to reduce this demand. Among them we mention batching and piggybacking. In this paper, we study the use of batching and piggybacking. We also demonstrate the advantages of an integrated scheme over the use of a single bandwidth reduction technique.

Resumo

A provisão de serviços de vídeo em larga escala requer grande quantidade de banda passante. Assim sendo, diversas técnicas vem sendo propostas para a redução desta demanda. Dentre elas, batching e piggybacking têm-se mostrado bastante efetivas. Estuda-se, neste artigo, o uso de batching e de piggybacking bem como monstram-se os benefícios de se adotar estas duas técnicas simultaneamente em um servidor de vídeo.

Palavras-chave: *Batching*, *piggybacking* e vídeo sob demanda.

1 Introdução

Vídeo sob Demanda (*Video on Demand* — VoD) é uma das aplicações multimídia mais promissoras nas Redes Digitais de Serviços Integrados de Faixa Larga (RDSI-FL). Através deste serviço, um usuário pode escolher um título e ter o vídeo enviado através de uma rede de telecomunicações até seu equipamento de exibição.

Entretanto, a transmissão de fluxos de vídeo requer grande largura de banda. Em outras palavras, o oferecimento de serviços de vídeo em larga escala requer a aplicação de técnicas que reduzem esta demanda. Dentre as principais técnicas destacam-se *Batching* [1, 2, 3] e *Piggybacking* [4, 5, 6], que levam em consideração a probabilidade de um conjunto de requisições por um vídeo popular chegar ao sistema dentro de um intervalo de tempo relativamente curto, de forma a atendê-las com um único fluxo.

A técnica de *Piggybacking* baseia-se no fato de que alterações da ordem de 5% na taxa de apresentação de quadros não são perceptíveis aos usuários. Desta forma, as requisições são atendidas imediatamente e obtem-se um fluxo compartilhado através da superposição de fluxos dessincronizados, ou seja, altera-se a taxa de exibição dos fluxos de vídeo até que estes apresentem um mesmo quadro de filme, descartando-se um dos fluxos. Por outro lado, a técnica de *Batching* consiste em reter requisições por um determinado intervalo de tempo, denominado Janela de *Batching*, com o objetivo de agrupar o número máximo de requisições e servir ao grupo com um único fluxo de vídeo, iniciado ao final da Janela de *Batching*.

O objetivo do presente trabalho é avaliar aspectos de desempenho em servidores de VoD, nos quais as técnicas de *Batching* e *Piggybacking* são empregadas. As principais contribuições deste trabalho são:

- Uma nova política de *Batching*;
- Duas novas políticas de *Piggybacking*;
- Um estudo sobre a integração de *Batching* e *Piggybacking*, e;
- Um estudo sobre a complexidade do algoritmo de geração da árvore ótima de mesclagens da política *Snapshot*, resultando na proposição de uma nova heurística para solucionar o problema, denominada *BuildTree*, com complexidade inferior.

O restante deste artigo está organizado como segue. Na seção 2 são apresentados trabalhos relacionados. Os estudos direcionados a aplicação isolada de *Piggybacking* são mostrados na seção 3. A seção 4 apresenta o estudo sobre *Batching*. A integração das técnicas é apresentada na seção 5. Finalmente, na seção 6 finaliza-se o trabalho.

2 Trabalhos Relacionados

Diversas políticas de *Batching* vêm sendo propostas na literatura. Em [3] apresenta-se um estudo considerando as políticas Primeiro a Chegar, Primeiro a ser Atendido (*First Come, First Served* — FCFS) e Maior Tamanho de Fila (*Maximun Queue Length* — MQL). Na política FCFS, todas as requisições são inseridas em uma mesma fila conforme a ordem de chegada e, quando um fluxo torna-se disponível, este é alocado para a primeira requisição da fila, de tal forma que todas as demais requisições pendentes do mesmo vídeo também são atendidas. Por outro lado, no esquema MQL as requisições são inseridas em filas distintas e o vídeo com maior fila é selecionado para alocação.

Em [2], Philip Yu e H. Sachnai introduzem esquemas de escalonamento de fluxos de vídeo, dentre eles *Batch MBQ*, que levam em consideração o tempo de abandono estimado dos usuários.

Define-se um *intervalo mínimo de espera*, após o qual pode-se alocar um fluxo segundo os critérios de maior tamanho de fila ou maior número estimado de abandonos.

Por outro lado, encontram-se na literatura diversas políticas de *Piggybacking*, em que se define a Janela Máxima de Mesclagem, W_m, como a maior distância em quadros, relativa ao início do vídeo (quadro inicial), a partir da qual a mesclagem de dois fluxos ocorre após o último quadro de filme. Em [4, 7], Golubchik *et al.* introduzem as políticas Mesclagem Simples, Par-Ímpar e Gulosa. A política Mesclagem Simples organiza fluxos de um mesmo vídeo, iniciados na Janela Máxima de mesclagem, em grupos de mesclagem, que originam um único fluxo resultante após a ocorrência de todas as mesclagens. A política Par-Ímpar subdivide os fluxos iniciados na Janela Máxima em pares, obtendo uma redução máxima de 50% no número de fluxos do sistema. A política Gulosa realiza o emparelhamento de fluxos enquanto possível antes do final da exibição. Deste modo, obtem-se uma redução ainda maior no número de fluxos no sistema.

Em [5], Philip Yu *al.* introduzem as políticas Mesclagem Simples Generalizada e Algoritmo *Snapshot*. Nestas políticas, assume-se que as chegadas de requisições ao sistema ocorrem conforme um processo de Poisson. Deste modo, deriva-se analiticamente uma Janela Ótima de Mesclagem, W. Na política Mesclagem Simples Generalizada, os fluxos são subdivididos em grupos com o objetivo de gerar um único fluxo resultante, de modo semelhante à política Mesclagem Simples original.

Dentre as políticas de *Piggybacking*, a política *Snapshot* [5, 8] é a que busca minimizar o número de quadros exibidos por um conjunto de fluxos de vídeo de forma mais eficiente.

A computação realizada pelas políticas de *Piggybacking* pode ser vista como uma árvore binária em que as folhas correspondem aos fluxos, os nós internos são as mesclagens intermediárias e a raiz é a mesclagem final formando o fluxo resultante do conjunto (Figura 1). Estas representações gráficas das estratégias de mesclagem de um conjunto de fluxos são chamadas de **Árvores de Mesclagens**. 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** [5, 8, 9]:

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

o que implica que o número de árvores cresce rapidamente inviabilizando uma 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 em quadros e denotadas por $f_1, f_2, \ldots, f_n$, em que $f_1 \geq f_2 \geq \cdots \geq f_n$, em um determinado instante de tempo. Sejam S_{min} e S_{max}, as taxas mínima e máxima, repectivamente, de apresentação, em quadros por segundo, de um vídeo e 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 de filme. 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

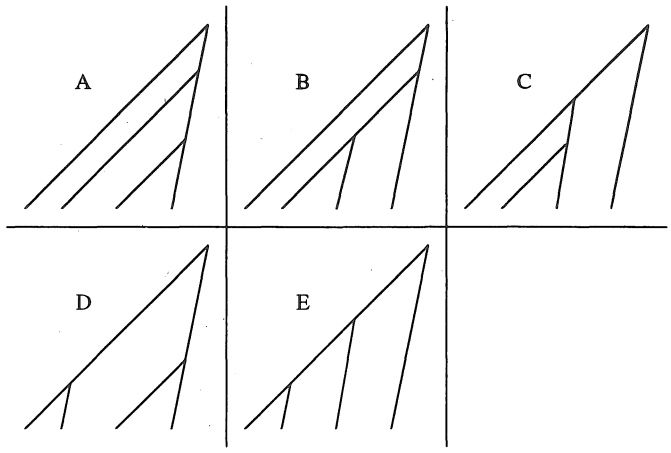


Figura 1: Possíveis árvores de mesclagens para quatros fluxos.

cada fluxo é dado por:

$$C(i,i) = L - f_i \tag{3}$$

Para se obter o custo $C(i,j)$ mínimo é necessário que o princípio de otimalidade seja satisfeito, ou seja, "*existe um fluxo k, com $i \leq k < j$, tal que as subárvores esquerda $(i,\ldots,k)$ e direita $(k+1,\ldots,j)$ também são ótimas*"x. 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:

$$C(i,k) + C(k+1,j) - \max(L - P(i,j), 0) \tag{4}$$

Portanto, a política ótima para os fluxos $i,\ldots,j$ possui subárvores $i,\ldots,k^*$ e $k^*+1,\ldots,j$, em que:

$$k^* = \text{argmin}_{i \leq k < j}\{C(i,k) + C(k+1,j) - \max(L - P(i,j), 0)\} \tag{5}$$

Deste modo, o custo para o 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*, 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 [5, 8]. 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.

3 Novas Políticas de *Piggybacking*

Nesta seção são introduzidas duas novas políticas de *Piggybacking*, S^2 e Híbrida. Adicionalmente, é apresentado um estudo sobre o algoritmo para a construção da árvore ótima de mesclagem da política *Snapshot* com o objetivo de se reduzir sua complexidade.

3.1 A Política S^2

A política *Snapshot* foi proposta de modo a minimizar o número de quadros exibidos pelos fluxos iniciados em intervalos *Snapshot*. Verifica-se que, quanto menor o intervalo médio entre requisições, menor o tamanho da janela ótima de mesclagem, podendo haver uma ou várias janelas ótimas contidas em W_m. A possibilidade de mesclagem entre dois fluxos separados por no máximo W_m quadros, 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 iniciados pelo sistema numa janela máxima alterada de mesclagem, dada por W'_m ($W'_m \leq W_m$). 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 dos fluxos resultantes dos intervalos *Snapshot*.

O funcionamento da política S^2 é descrito a seguir: aplica-se primeiramente o algoritmo para a determinação da árvore ótima de mesclagem sobre os fluxos iniciados nos intervalos I (como proposto originalmente) e, em seguida, aplica-se novamente o algoritmo sobre os fluxos resultantes dos intervalos ótimos. A segunda aplicação do algoritmo ocorre ao final de intervalos, denotados por I_{S^2}, cuja 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).

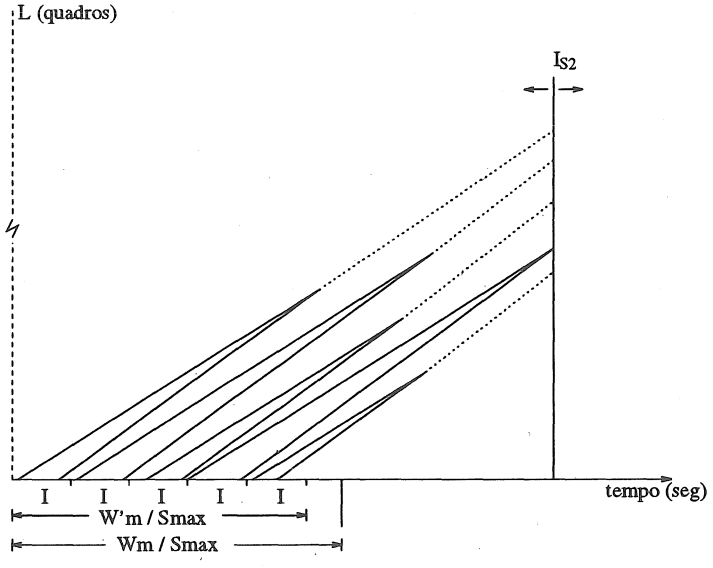
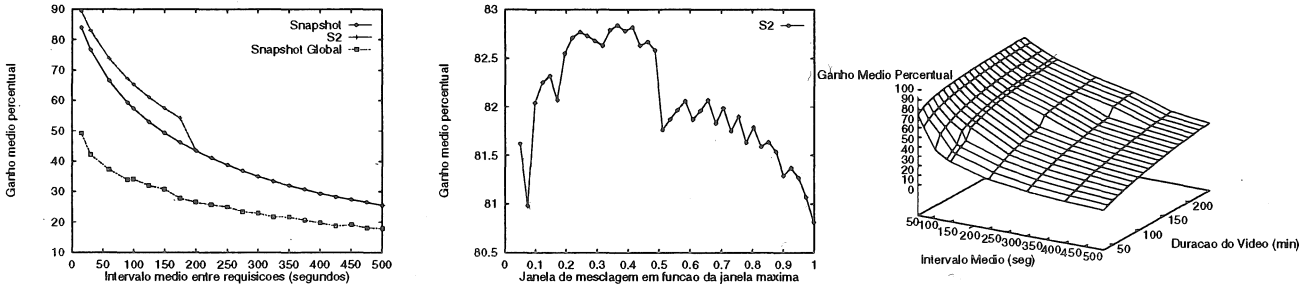


Figura 2: Possível situação da política S^2 na qual o último fluxo resultante gerado não corresponde ao fluxo resultante da última janela ótima contida na janela W'_m.

3.1.1 Resultados Numéricos

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*. Utiliza-se o método de replicações independentes para se calcular um intervalo 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. Os gráficos mostram a redução percentual média no número de quadros apresentados em decorrência da aplicação das políticas de *Piggybacking*.



(a) (b) (c)

Figura 3: Avaliação da política S^2: (a) variação da taxa de chegadas sobre as políticas S^2, *Snapshot* original e Global, (b) variação do tamanho da janela de mesclagem sobre S^2 e (c) variação da taxa de chegadas e do tamanho do vídeo sobre S^2.

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

A Figura 3-b mostra a influência da variação do tamanho da janela de mesclagem sobre a política S^2, considerando-se intervalo médio entre chegadas de 30 segundos e vídeo de 2 horas. Observa-se que a política S^2 é insensível à variação da janela de mesclagem, devido ao segundo nível de otimização, isto é, quando a janela de mesclagem assume valores muito pequenos, ocorrem poucas mesclagens no primeiro nível de otimização; porém, no segundo nível, os fluxos resultantes do primeiro nível são superpostos, proporcionando assim, uma redução ainda maior no número de quadros apresentados.

A Figura 3-c 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 diversas taxas de chegadas de requisições, cujos intervalos médios assumem valores entre 15 a 500 segundos. Alcançam-se reduções no número de quadros exibidos que variam de 9% para vídeos curtos até 93% para filmes longos (4 horas de duração).

3.2 A Política Híbrida

A política *Snapshot* foi definida de modo a obter a máxima redução de quadros exibidos por um conjunto de fluxos iniciados nos intervalos *Snapshot*. No interior destes, os fluxos comportam-se conforme a política Mesclagem Simples, ou seja, formam um grupo de mesclagem com o primeiro fluxo lento e os demais rápidos. Ao final dos intervalos, formula-se um problema de otimização para minimizar o número de quadros apresentados pelos fluxos que permanecem no sistema até estes instantes.

Com base neste comportamento, define-se a política Híbrida, que atua sobre os fluxos, no interior dos intervalos *Snapshot*, à medida em que são iniciados. A política considera o tempo decorrido entre o início dos fluxos alterando suas velocidades de apresentação dinamicamente, com o objetivo de se reduzir ainda mais o número de quadros exibidos por estes. Ao final do intervalo *Snapshot* aplica-se o algoritmo de geração da árvore ótima de mesclagem. A política Híbrida alcança seus objetivos, dado que, reduz o número de quadros apresentados no interior de intervalos *Snapshot*.

3.3 Redução da Complexidade de Geração da Árvore Ótima de Mesclagem da Política *Snapshot*

O problema de otimização formulado pela política *Snapshot*, com o objetivo de obter a árvore ótima de mesclagem dos fluxos de um intervalo I, é solucionado através de um algoritmo de programação dinâmica, cuja complexidade é de $\Theta(n^3)$, em que n representa o número de fluxos que alcançam o final de um intervalo I. Porém, em virtude da natureza de tempo real do problema (recepção e exibição contínuas dos quadros de um vídeo), deseja-se obter uma solução que seja atrativa sob os aspectos de custo (otimização dos recursos do sistema) e de desempenho.

Neste sentido, duas abordagens foram investigadas: *i*) utilização de algoritmos propostos para o problema da parentização ótima do produto de matrizes (solução através da triangulação de polígonos convexos), em virtude das analogias entre os problemas da parentização ótima e construção da árvore ótima de mesclagens [10, 11, 12, 13, 14]; e *ii*) proposição de um novo algoritmo para a construção da árvore de mesclagens.

A primeira abordagem não se mostra viável, dado que requer a redução do problema de construção da árvore ótima de mesclagens no problema da multiplicação de matrizes, e esta redução não se mostra clara. Com isso, analisa-se a segunda abordagem.

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:

$$k^* = \text{argmin}_{i \leq k < j}\{|P(i,k) - P(k,j)|\} \quad (6)$$

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 6) 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 3.1 *A heurística BuildTree possui complexidade* $O(n^2)$.

Prova:

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

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 4-a. Em seguida, reaplica-se o algoritmo sobre os subconjuntos $(i = 1, \ldots, j = 4)$ e $(i = 5$ e $j = 6)$. A Figura 4-b ilustra a árvore de mesclagens final contruída pelo algoritmo (neste caso, igual a arvore gerada pelo algoritmo de programação dinâmica).

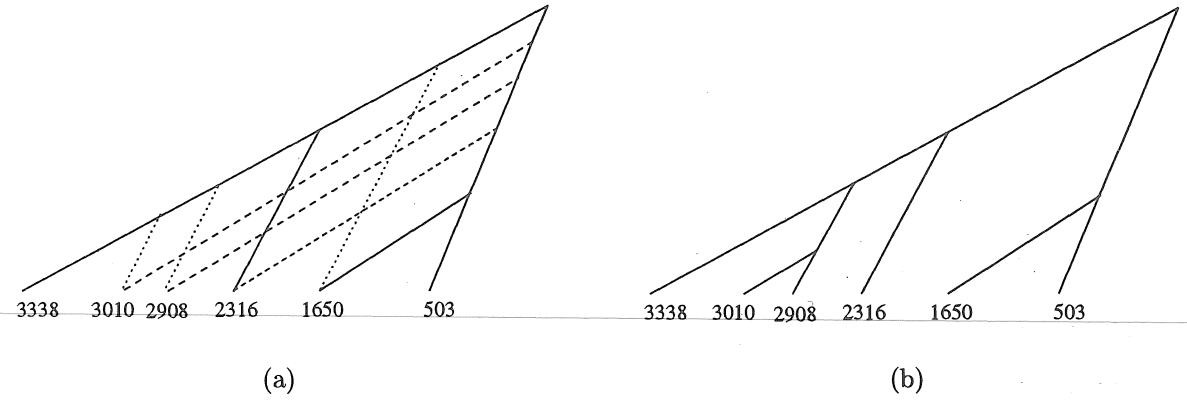


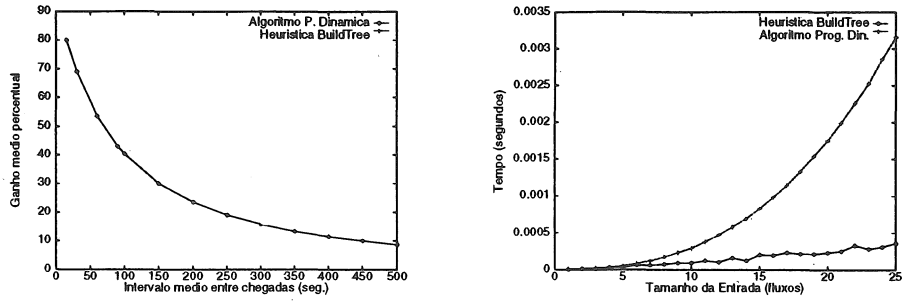
Figura 4: Construção da árvore de mesclagens pela heurística: (a) Análise dos segmentos de mesclagem para o primeiro particionamento e (b) Árvore Final.

Em nossos experimentos a heurística foi implementada utilizando-se busca sequencial do fluxo k através do conjunto $i, \ldots, j$. Considera-se que, com a utilização de busca binária em substituição à sequencial, obtem-se uma redução significativa no tempo de execução da heurística proposta. No Apêndice A, apresenta-se a heurística implementada em linguagem C.

A heurística descrita constitui-se numa aproximação da solução ótima, fazendo-se necessário avaliar a precisão em relação à esta. Com este objetivo, ambos algoritmos são aplicados 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.

A Figura 5-a 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 quando comparada à solução ótima. Isto se deve ao fato de que, na maioria dos casos, a heurística obtém o resultado ótimo; de modo que, considerando-se casos individuais, a diferença percentual mostra-se muito pequena assumindo valores no máximo iguais a 5,6%. Porém, a diferença média mostra-se desprezível, em alguns casos possuindo valores da ordem de 10^{-7} para taxas elevadas e nula para valores mais baixos de taxa, como pode-se constatar através da coincidências das curvas.

No gráfico da Figura 5-b, instâncias de tamanho 1 até 25 (fluxos), observadas em experimentos anteriores, foram fornecidas como entrada para a computação dos tempos de processamento de ambos os algoritmos. Observa-se em todos os casos que o tempo de execução da heurística foi



(a) Custo Médio (b) Tempos de Execução

Figura 5: Comparação entre a heurística *BuildTree* e o algoritmo de programação dinâmica da política *Snapshot*.

no máximo igual ao tempo do algoritmo de programação dinâmica, nunca excedendo-o. Este experimento foi realizado num microcomputador PC 486 Dx4-100MHz com Linux 2.0.0.

4 Uma Nova Política de Batching

Batching constitui-se numa alternativa interessante para a redução da demanda de banda passante, uma vez que diversas requisições podem ser agrupadas antes do início da exibição do vídeo. Esta seção tem como objetivo apresentar um estudo direcionado a *Batching*, em que uma nova política é introduzida e avaliada. Inicialmente, apresentam-se métricas para avaliação de um sistema com *Batching* (seção 4.1), em seguida a nova política é introduzida (seção 4.2) e avaliada (seção 4.3).

4.1 Métricas de Avaliação

Num sistema de VoD, sujeito a um processo de chegada de requisições $\mathcal{P}$, com tempo médio entre requisições $1/\lambda$, um usuário escolhe o vídeo, k, que deseja assistir com probabilidade p_k, $1 \leq k \leq V$, em que V é o número de programas armazenados no servidor. A requisição deste usuário é inserida na fila correspondente, onde permanece até que um fluxo seja alocado ou o usuário decida abandonar o sistema sem atendimento.

Em um sistema com *Batching*, o principal parâmetro de avaliação constitui-se no número de usuários suportados. Os parâmetros seguintes podem ser utilizados na avaliação de sistemas com *Batching*:

- **Probabilidade de Abandono**: É definida como a razão entre o número de abandonos pelo número total de requisições em um determinado intervalo de tempo.
- **Retardo Médio de Atendimento**: É definido como o tempo decorrido entre a chegada da requisição e seu atendido através da alocação de um fluxo.

- **Injustiça** (*Unfairness*): Seja $P_A(i)$ a probabilidade de abandono para o vídeo i e $\overline{P} = \sum_{i=1}^{V} P_A(i)/V$ a probabilidade média de abandono, define-se a injustiça no sistema como:

$$Unfairness = \sqrt{\frac{\sum_{i=1}^{V}(P_A(i) - \overline{P})^2}{V-1}}. \quad (7)$$

A Equação 7 determina a variância das probabilidades de abandono no sistema. Objetiva-se minimizar os abandonos tanto nas filas de vídeos populares como dos não populares, tornando assim, o sistema justo. No entanto, os critérios definidos por certas políticas de *Batching*, com o objetivo de se maximizar o número de usuários suportados, podem impedir a homogeneidade dos valores das probabilidades de abandono, isto é, estes critérios podem beneficiar somente os vídeos populares. Quando isto ocorre, avaliam-se as diferenças entre os índices percentuais de perdas nas filas dos diversos vídeos através de sua variância. Por isso afirma-se que o sistema tende a ser mais justo quanto mais $P_A(i)$ aproxima-se de $\overline{P}$ para todo $1 \leq i \leq V$, ou seja, a probabilidade de abandono de requisições para cada vídeo aproxima-se de média das probabilidades de abandono.

4.2 A Política *Look-Ahead-Optimize-Batch* — LAO-*Batch*

A política LAO-*Batch* [15] assume uma janela de *Batching* dependente do usuário, diferentemente de outras políticas, que a consideram um parâmetro global do sistema. Em outras palavras, cada usuário tem associado a ele um tempo (aleatório) máximo, T, de espera na fila que ocorre segundo a distribuição Normal, $T \sim N(\mu, \sigma^2)$.

A alocação de fluxos pode ocorrer quando o tempo máximo de espera de um usuário (janela de *Batching*) esgota, isto é, quando um usuário está prestes a abandonar o sistema. As requisições são inseridas nas filas conforme a ordem cronológica de término da janela de *Batching*, ou seja, a primeira requisição de uma fila é aquela cuja janela de *Batching* esgota mais cedo.

O critério de alocação de fluxos é o mesmo para todos os vídeos (independentemente de sua popularidade).

i) Se o número de vídeos com requisições pendentes é menor ou igual ao número de canais atualmente disponíveis no sistema, aloca-se um fluxo para o vídeo cuja janela de *Batching* esgotou;

ii) Se o número de filas com requisições pendentes é maior que o número de fluxos disponíveis, define-se uma *Janela de Estudo* — JE, na qual são ordenados os próximos eventos de possíveis alocações e liberações de fluxo. A JE tem como limite inferior o tempo corrente e como limite superior o instante no qual termina o maior tempo de espera de uma requisição na cabeça de uma das filas (*head of the queue*). Com base na JE, formula-se um problema de otimização para a maximização do número de usuários atendidos pelo sistema (Figura 6).

Levando-se em consideração os possíveis instantes de alocação de canais, bem como de liberação dos mesmos dentro da JE, resolve-se um problema de programação linear inteira para determinar se é vantajoso ou não alocar um canal para o vídeo cujo intervalo de *Batching* esgotou (ou seja, o vídeo que originou a JE em questão). Aloca-se um fluxo para o referido vídeo se e somente se

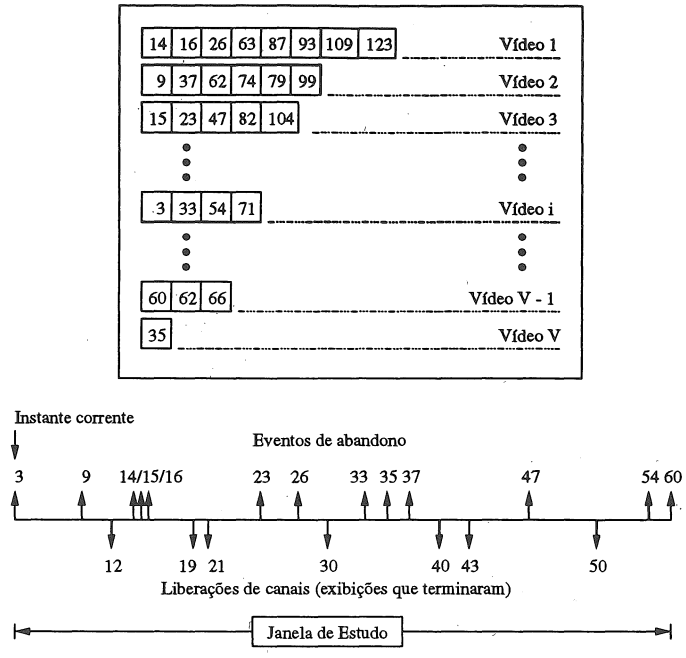


Figura 6: Cenário de filas com requisições pendentes e sua janela de estudo correspondente.

a alocação no presente instante de tempo não acarreta uma perda maior de usuários, por falta de canais disponíveis, em outros instantes de tempo compreendidos entre os limites da janela de estudo. O problema de programação linear inteira pode ser descrito como:

$$\max \sum_{k=1}^{V} \sum_{i=1}^{L_k} (q_k - i + 1) \times x_{ki}$$

Sujeito a:

$$\sum_{i=1}^{L_k} x_{ki} \leq 1, \qquad k = 1 \ldots V$$

$$x_{ki} + \sum_{\substack{\hat{k}=1 \\ \hat{k} \neq k}}^{V} \sum_{j=1}^{L_k} x_{\hat{k}j} \leq A + R_{ki} \qquad \begin{array}{c} t_{\hat{k}j} < t_{ki} \\ k = 1 \ldots V \\ i = 1 \ldots L_k \\ t_{ki} \in \Gamma \end{array}$$

$$x_{ki} \in \{0, 1\}$$

em que : e_{ki} - momento no qual se esgota o intervalo de Batching do i-ésimo usuário da k-ésima fila, isto é, o momento em que o usuário esta prestes a abandonar o sistema.

$$\Gamma = \left[\min_{k}\{e_{k1}\} \ , \ \max_{k}\{e_{k1}\} \right]$$

em que e_{k1} é o instante no qual o primeiro usuário da k-ésima fila abandonará o sistema.

Γ: Janela de Estudo,

V: Número total de vídeos armazenados no servidor,

q_k: Número de requisicões pendentes na k-ésima fila, ou seja, tamanho da k-ésima fila, $0 \leq q_k \leq \infty$,

L_k: L-ésimo usuário da k-ésima fila tal que:

$$\begin{cases} e_{kL} = \max_{i}\{e_{ki}\} \ , \ e_{kL} \leq \max_{j}\{e_{j1}\} \ , \ e_{kL+1} > \max_{j}\{e_{j1}\} \ , & q_k > L \\ e_{kL} = \max_{i}\{e_{ki}\} \ , \ e_{kL} \leq \max_{j}\{e_{k1}\} \ , & q_k = L \end{cases}$$

t_{ki}: Momento de abandono do i-ésimo da k-ésima fila, isto é, o momento em que estoura a sua janela de *Batching* (tolerância à espera para assistir a um filme), $t_{ki} \in \Gamma$,

x_{ki}: Variável inteira 0-1 tal que, se o seu valor é 1 aloca-se um canal para a k-ésima fila no instante t_{ki}; caso contrário, não se aloca um canal para a k-ésima fila no instante t_{ki},

R_{ki}: Número de canais liberados desde o início da Janela de Estudo até o instante $t_{ki}, 0 \leq R_{ki} \leq N_{max}$,

N_{max}: Capacidade do Servidor (número total de canais), e

A: Número de canais disponíveis no início da Janela de Estudo $(\min\{e_{k1}\}), 0 \leq A \leq N_{max}$.

A função objetivo traduz o ganho obtido no sistema com a solução do problema, que leva em consideração todas as requisições inseridas nas filas

A restrição representada pela Equação 8 determina que, para o conjunto de requisições pendentes numa fila, o número total de canais alocados é 1. Isto é, se na solução de um determinado problema a variável $x_{ki} = 1$, a solução obtida indica que as requisições $1, \ldots, i-1$ não são atendidas (abandonando o sistema) e que, no instante t_{ki}, ocorrerá a alocação de um fluxo para o vídeo k suportando as requisições $i, \ldots, q_k$.

O limite superior no número total de fluxos alocáveis é imposto pela restrição da Equação 9. Num determinado instante (t_{ki}), aloca-se um fluxo se o montante utilizado até então é inferior à soma dos fluxos inicialmente disponíveis com os liberados até este instante .

A solução do problema leva em consideração todos os eventos dentro da JE (eventos de possível alocação e de liberação de canais). Mais precisamente, o problema de otimização formulado indica se é atrativo (ou não) para o sistema alocar um fluxo para o vídeo cuja requisição representa o primeiro evento da JE, isto é, a requisição cuja janela de *Batching* esgotou. Em outras palavras, para um vídeo k, que origina a janela de estudo, aloca-se um fluxo se $x_{k1} = 1$, caso contrário, a alocação é retida e a primeira requisição da fila do vídeo k abandona o sistema; de modo que, para cada requisição cujo intervalo de *Batching* esgota, define-se uma nova janela de estudo e formula-se um novo problema.

4.3 Resultados Numéricos

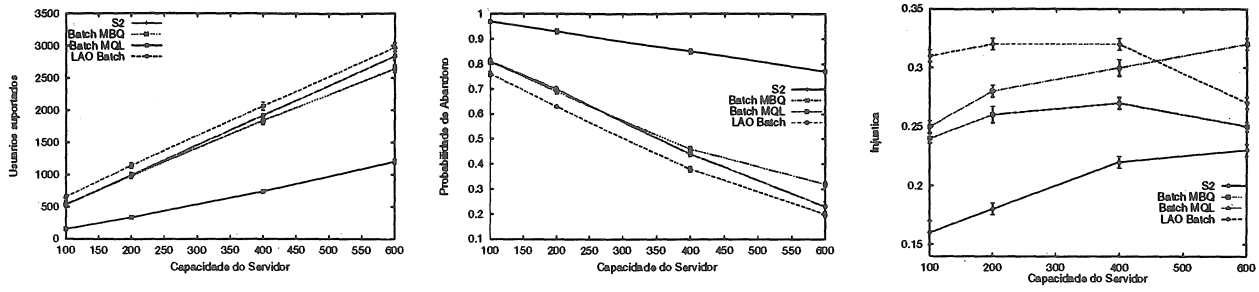
Na política proposta considera-se que o tempo entre chegadas de requisições é exponencialmente distribuído com média $1/\lambda$. A probabilidade de escolha de um vídeo é dada pela distribuição Zipf com parâmetro $\theta = 0{,}271$, valor derivado da aloação de filmes em lojas de vídeo [3]. Assume-se que cada usuário espera durante um intervalo de tempo cuja duração é definida pela distribuição normal (*Gaussiana*) com média μ e variância σ^2, ou seja, $T \sim N(\mu, \sigma^2)$ [2]. Deste modo, considera-se que após T minutos o usuário abandona o sistema sumariamente. Utilizou-se o método das replicações independentes para calcular um intervalo com nível de confiança de 95%.

Nos experimentos de simulação, compara-se a política LAO-*Batch* com as políticas *Batch* MBQ e com uma variação da política MQL. Utiliza-se a política MBQ pois esta demonstrou desempenho superior ao das políticas MQL e FCFS [2]. Para a política MQL modificada, assume-se um critério de escalonamento semelhante ao primeiro caso da política LAO-*Batch*, isto é, alocações somente podem ocorrer quando uma janela de *Batching* esgota. Adicionalmente, compara-se a efetividade das políticas de *Batching* em relação a efetividade de *Piggybacking*. Com este objetivo, utiliza-se a política S^2, uma vez que o desempenho desta mostra-se superior ao das demais políticas de *Piggybacking* [6].

Assume-se que o número de vídeos armazenados no servidor é $V = 100$, a duração média de cada vídeo é de 2 horas (120 minutos) e a taxa de chegadas é de 50 requisições por minuto. A capacidade do servidor assume valores entre 100 e 600 fluxos e o tempo de abandono dos usuários é modelado pela distribuição normal com média $\mu = 10$ minutos e variância $\sigma^2 = 2{,}5$ minutos, $T \sim N(10,\ 2{,}5)$.

No gráfico da Figura 7-a observa-se que a política LAO-*Batch* suporta um número de usuários superior ao das demais políticas consideradas, dado que consegue prever se uma alocação em um certo momento vai produzir um número maior (menor) de usuários aceitos pelo sistema dentro da JE. Com a política MQL, quando o servidor está saturado, podem haver várias filas aptas para alocação. No entanto, quando um fluxo torna-se disponível sua realocação é imediata. Quanto à política *Batch* MBQ, que possui um intervalo mínimo de espera entre alocações, ω, podem haver desistências sumárias quando o tempo de abandono de um usuário é menor que ω, reduzindo assim seu desempenho. Pode-se observar também que todas as políticas de *Batching* apresentam desempenho superior ao da política de *Piggybacking* S^2.

No gráfico da Figura 7-b analisa-se probabilidade de abandono dos usuários. A política LAO-



(a) (b) (c)

Figura 7: Comparação entre as políticas de *Batching* e a política S^2: (a) usuários suportados, (b) probabilidade de abandonos e (c) *unfairness*.

Batch possui os índices de probabilidade de abandono mais baixos dentre todas consideradas, sendo um reflexo da maximização do número de usuários suportados, e a política S^2 apresenta índice de abandonos superior àqueles obtidos pelas políticas de *Batching*. Em outras palavras, em decorrência da maior eficiência de *Batching* em relação a *Piggybacking*, observa-se um número maior de abandonos para a política S^2 que para as políticas de *Batching*.

O grau de injustiça é considerado no gráfico da Figura 7-c. Observa-se que a política LAO-*Batch* apresenta os níveis de injustiça mais elevados, uma vez que privilegia os vídeos populares em detrimento dos não populares. Por outro lado, a política S^2 proporciona os menores índices de injustiça entre os vídeos. Isto se deve à alocação de fluxos obedecer a seqüência de chegada das requisições, ou seja, emprega-se o esquema FCFS para a alocação de fluxos, não havendo privilégios para qualquer um dos vídeos. Percebe-se um nível crescente de injustiça para a política S^2 porque, à medida em que se aumenta o número de fluxos num servidor, aloca-se uma porcentagem maior destes para os vídeos populares devido à maior freqüência de chegadas de suas requisições, as quais têm maior chance de encontrar fluxos disponíveis para alocação.

5 Abordagem Integrada de *Batching* e *Piggybacking*

Nas seções anteriores analisou-se como políticas de *Piggybacking* e de *Batching* podem ser aplicadas em um sistema de VoD. Nesta seção, examinam-se os fatores que influenciam sua integração, e como esta integração atua sobre aspectos como o número de usuários suportados, probabilidade de abandono e justiça do sistema.

5.1 Integrando *Batching* e *Piggybacking*

Um esquema integrado de *Batching* e *Piggybacking* sugere naturalmente um modelo composto por dois níveis, no qual a base realiza o controle da estratégia de alocação e gerenciamento das filas de requisições; enquanto no topo, controlam-se os fluxos em exibição no sistema. As funções atribuídas à base podem ser desempenhadas por uma política de *Batching*, e as do segundo nível,

por uma política de *Piggybacking*. O esquema pode conter uma "interface" para o intercâmbio de informações entre os níveis, Figura 8.

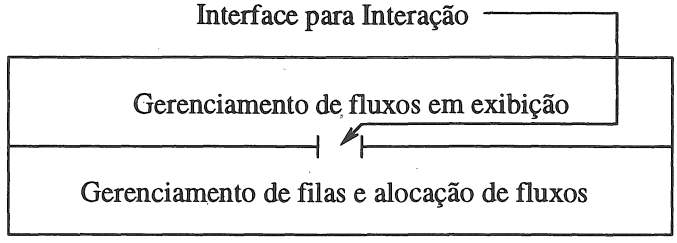


Figura 8: Esquema genérico para integração de *Batching* e *Piggybacking*.

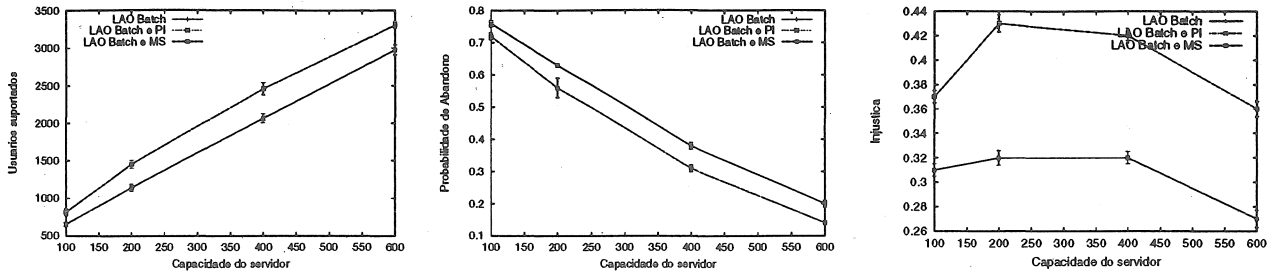
As políticas de *Piggybacking* possuem a limitação de somente poder superpor fluxos, antes do final da exibição do vídeo, se estes estiverem distantes no máximo W_m quadros. Esta restrição deve ser especialmente observada na presença de *Batching*, que favorece ao distanciamento dos fluxos de um vídeo. Uma característica, que de certo modo agrava esta restrição, consiste no fato de que algumas políticas de *Batching* impõem intervalos mínimos entre a alocação de fluxos, como por exemplo, *Batch* MBQ. Com a determinação de intervalos muito amplos, a possibilidade de mesclagem de fluxos praticamente inexiste, ou seja, mesmo com a presença de políticas de *Piggybacking* tem-se a utilização quase que exclusiva de *Batching*.

Neste sentido, a política LAO-*Batch* apresenta-se como uma opção atrativa para integração com Piggybacking pois: *i*) seu desempenho mostra-se superior em relação às demais políticas analisadas, e *ii*) inexistência de intervalos de duração fixa entre eventos de alocação de fluxo. Isto se deve ao fato do intervalo de *Batching* ser definido como o tempo de abandono dos usuários. Quanto às políticas de *Piggybacking*, foram utilizadas no presente trabalho as políticas Mesclagem Simples e Par-Ímpar. Tal escolha deve-se, basicamente, ao fato de ambas as políticas realizarem toda a sua computação sobre fluxos iniciados no maior intervalo de quadros possível, isto é, a Janela Máxima de mesclagem.

5.2 Resultados Numéricos

A avaliação da implementação conjunta de *Batching* e *Piggybacking* foi realizada através de simulação, utilizando-se o modelo descrito na seção 4.3. Pela análise apresentada na seção 4.3, em que se constata grau de efetividade maior de *Batching* que de *Piggybacking*, pode-se concluir que se abordagem integrada é vantajosa em relação ao emrpego isolado de *Batching*, também o é em relação a *Piggybacking*.

No gráfico da Figura 9-a comparam-se as curvas do número de usuários suportados em função da capacidade do servidor para a política LAO-*Batch* isoladamente e de sua implementação conjunta com as políticas Mesclagem Simples e Par-Ímpar. Observa-se que com a integração das políticas, o número de usuários suportados supera o obtido com a utilização exclusiva de *Batching*. Isto ocorre porque, com a introdução de *Piggybacking*, ocorrem liberações antecipadas de fluxos (através das mesclagens). Nota-se também o comportamento idêntico nas implementações de LAO-*Batch* com as políticas Mesclagem Simples e Par-Ímpar. A coincidência das curvas é proporcionada pelo



(a) (b) (c)

Figura 9: Comparação a abordagem integrada com a política LAO-*Batch*: (a) número de usuários suportados, (b) probabilidade de abandono e (c) *unfairness*.

tempo médio de abandono ($\mu = 10$ minutos), o qual impede o diferenciamento das políticas de *Piggybacking*, isto é, a política Mesclagem Simples realiza no máximo o emparelhamento de fluxos. Em outras palavras, o distanciamento entre os fluxos impede a formação de grupos de mesclagem na política Mesclagem Simples.

Na Figura 9-b analisa-se a probabilidade de abandono. Nota-se que há uma queda no número de usuários que abandonam o sistema sem atendimento, isto se deve a dois fatores: (*i*) liberação antecipada de fluxos com as mesclagens, permitindo o atendimento de novas requisições; e (*ii*) a realocação destes fluxos para o atendimento de vídeos populares, que claramente concentram um número de pedidos maior.

Os índices de injustiça são ilustrados no gráfico da Figura 9-c. Observa-se que a abordagem integrada de *Batching* e *Piggybacking* favorece ao aumento da injustiça no sistema, uma vez que os fluxos liberados antecipadamente são realocados conforme os critérios da política LAO-*Batch*, que privilegia os vídeos populares. Em síntese, alocam-se mais fluxos para os vídeos populares reduzindo-se as desistências em suas filas, ao mesmo tempo em que os índices de abandono de usuários que requisitam vídeos de menor popularidade não sofrem alterações significativas.

6 Conclusões

O serviço de Vídeo sob Demanda requer grande largura de banda para a transmissão de fluxos de vídeo, de modo que para oferecer este serviço a uma população de usuários numerosa, faz-se necessário o emprego de técnicas como *Batching* e *Piggybacking*.

Neste trabalho apresenta-se um estudo sobre o emprego de *Batching* e *Piggybacking*. O estudo compreende a proposição e avaliação de novas políticas de ambas as técnicas, bem como da avaliação da integração de algumas destas políticas. São traçados comentários sobre os fatores que influenciam a abordagem integrada e sobre a escolha das políticas utilizadas. Observa-se que a utilização de um esquema integrado de *Batching* e *Piggybacking* é vantajoso, pois permite ao servidor suportar um número de usuários maior que o obtido com a utilização de ambas as técnicas isoladamente. Adicionalmente, são avaliados aspectos teóricos sobre o algoritmo de programação dinâmica utilizado na determinação da árvore ótima de mesclagem pela política Algo-

ritmo *Snapshot*, resultando na proposição da heurística *BuildTree*, que possui custo computacional inferior ao do algoritmo de programação dinâmica da política *Snapshot*.

Nos problemas investigados, foram considerados diversos cenários, entretanto apenas um subconjunto destes encontra-se neste trabalho. Finalmente, pode-se dizer que os experimentos realizados evidenciam a extensão dos resultados contidos na literatura contribuindo, deste modo, para o avanço da pesquisa sobre as técnicas de redução da demanda de banda passante, mais especificamente *Batching* e *Piggybacking*.

Referências

[1] Philip S. Yu, Joel L. Wolf, and Hadas Shachnai. Design and Analysis of a Look-Ahead Scheduling Scheme to Support Pause-Resume for Video-on-Demand Applications. *Multimedia Systems*, 3(4):137–149, Setembro 1995.

[2] Hadas Shachnai and Philip S. Yu. The Role of Wait Tolerance in Effective Batching: A Paradigm for Multimedia Scheduling Schemes. Technical Report RC 20038 (88607), IBM Research Division, T. J. Watson Research Center, Abril 1995.

[3] Asit Dan Dinkar Sitaram and Perwez Shahabuddin. Dynamic Batching Policies for an on-demand Video Server. *Multimedia Systems*, 4:112–121, 1996.

[4] Leana Golubchik, John C. S. Lui, and Richard Muntz. Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-on-Demand Storage Servers. *Multimedia Systems*, 4(3):140–155, 1996.

[5] Charu C. Aggarwal, Joel Wolf, and Philip S. Yu. On Optimal Piggybacking Merging Policies for Video-on-Demand Systems. In *Proceedings of the ACM Sigmetrics*, volume 24, pages 200–209, New York, Maio 1996. ACM Press.

[6] Roberto de A. Façanha and Nelson L. S. da Fonseca. A Política de Piggybacking S^2. In *Anais do IV Simpósio Brasileiro de Sistemas Multimídia e Hipermídia*, pages 125–136, Rio de Janeiro, RJ, Maio 1998.

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

[8] Charu C. Aggarwal, Joel L. Wolf, and Philip S. Yu. Adaptive Piggybacking Schemes for Video-on-Demand Systems. Technical Report RC 20635 (91350), IBM Research Division, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, Novembro 1996.

[9] Martin Gardner. Catalan numbers. *Scientific American*, pages 120–124, Junho 1976.

[10] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*, chapter 16. The MIT Press, 3ª edition, 1991.

[11] T. C. Hu and M. T. Shing. Some theorems about Matrix Multiplication. In *Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, pages 28–35. IEEE Computer Society, 1980.

[12] T. C. Hu and M. T. Shing. Computation of Matrix Chain Products. Part I. *SIAM Journal on Computing*, 11(2):362–373, Maio 1982.

[13] T. C. Hu and M. T. Shing. Computation of Matrix Chain Products. Part II. *SIAM Journal on Computing*, 13(2):228–251, Maio 1984.

[14] T. C. Hu and M. T. Shing. An $O(n)$ Algorithm to Find a Near-Optimum Partition of a Convex Polygon. *Journal of Algorithms*, 2:122–138, 1981.

[15] Nelson Luís Saldanha da Fonseca and Roberto de Almeida Façanha. Maximizando o Número de Usuários em Servidores de Vídeo sob Demanda. Technical Report IC-98-30, Instituto de Computação, Universidade Estadual de Campinas, UNICAMP, Agosto 1998. Submetido ao SBRC'99.

A 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);
      }
}
```