Soluções Exatas para o Problema de Replicação e Distribuição de Requisições em Redes de Distribuição de Conteúdos

Tiago Araújo Neves[1], **Luiz Satoru Ochi**[2], **Célio Albuquerque**[2]

[1]Escola de Engenharia Industrial e Metalúrgica de Volta Redonda (EEIMVR) – Universidade Federal Fluminense (UFF) – Volta Redonda - RJ - Brasil

[2]Instituto de Computação (IC) – Universidade Federal Fluminense (UFF) – Niterói – RJ – Brasil

tneves@id.uff.br, {satoru,celio}@ic.uff.br

Abstract. *In a Content Distribution Network (CDN), in order to better serve clients, it is necessary to replicate contents at surrogate servers and distribute requests among such servers. Decisions of where to place replicated contents and how to distribute requests can be modeled as an optimization problem known as the Replica Placement and Request Distribution Problem (RPRDP), which is NP-hard. In this paper we use a model that regards several realistic details that are not treated simultaneously in the literature, such as constraints in server disk space and bandwidth, QoS requirements of requests and changes in the network conditions. Also, we present new approach for the offline version of the problem in which the future dememands are known beforehand. The new approach is an exact method that can find the optimal solutions of the problem. We compared the results obtained with other exact approach exposed in the literature and results show that the new method is able to obtain solutions to problems up to eight times greater than the limits found in the literature.*

Resumo. *Para melhor atender os clientes em uma Rede de Distribuição de Conteúdos (RDC) deve-se replicar os conteúdos em servidores dispersos geograficamente e distribuir as requisições entre estes mesmos servidores. A decisão de onde posicionar os conteúdos replicados e de como fazer a distribuição das requisições pode ser modelada como um problema de otimização chamado de Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR) que pertence à classe de problemas* NP*-Difíceis. Neste artigo, é usado um modelo computacional que considera várias características reais que não são tratadas simultaneamente na literatura, como por exemplo limitações nos discos e banda dos servidores, requisitos de QoS das requisições e mudanças nas condições da rede. Além disso, é apresentada uma nova abordagem de resolução da versão* offline *do problema, onde é suposto que a demanda futura é conhecida* a priori*. A abordagem utilizada é um método exato, que pode encontrar as soluções ótimas do problema. Os resultados obtidos são comparados com os e outra abordagem exata exposta na literatura, mostrando que o método proposto consegue fornecer soluções para problemas até oito vezes maiores que os tamanhos limites encontrados na literatura.*

1. Introdução

Com o crescimento da Internet, a demanda por conteúdos diversos aumentou consideravelmente em todo o mundo. Alguns destes conteúdos, em especial os conteúdos de multimídia, necessitam de algum tipo de suporte especializado, que possa oferecer garantias de Qualidade de Serviço (*Quality of Service* - QoS) como atraso máximo e largura de banda mínima para que as expectativas dos clientes possam ser satisfeitas. Neste sentido, muitos estudos têm sido feitos na tentativa de encontrar maneiras de servir melhor a crescente demanda por conteúdos com restrições de QoS [Tenzakhti et al. 2004].

Um modo de servir os conteúdos multimídia de maneira eficiente é através do uso de Redes de Distribuição de Conteúdos (RDC) [Bakiras and Loukopoulos 2005, Tenzakhti et al. 2004, Zhou and Xu 2007], que são tipicamente redes sobrepostas [Kurose and Ross 2003] usadas para posicionar réplicas dos conteúdos nas proximidades dos clientes reduzindo assim, o atraso, a carga nos servidores e o congestionamento da rede, possibilitando portanto melhorias na qualidade do serviço prestado. Existem empresas como Akamai [aka] e Mirror Image [mir] que se especializaram em prover arquiteturas de RDC para outras empresas, como Adobe, Aude Ag e Fox Interactive, que querem ou precisam distribuir seus conteúdos.

Existem vários problemas de otimização dentro das arquiteturas de RDC. Entre eles estão o Problema de Localização de Servidores (PLS), o Problema de Replicação (PR), o Problema de Posicionamento de Réplicas (PPR) e o Problema de Distribuição de Requisições (PDR). Descrições para o o PLS e para o PR encontram-se em [Bektas et al. 2007, Zhou and Xu 2007]. O PPR pertence à classe de problemas *NP*-difíceis e consiste em encontrar os melhores locais (servidores) dentro da rede sobreposta para colocar os conteúdos replicados de modo a reduzir os custos de transmissão [Aioffi et al. 2005, Zhou and Xu 2007]. O PDR consiste em, dado um conjunto de servidores com réplicas posicionadas, encontrar a melhor distribuição das requisições para o conjunto de servidores de modo a reduzir os custos de transmissão. Este problema, de acordo com um resultado deste artigo apresentado na Seção 4, pertence à classe *P* e tem enorme relevância nos custos dos provedores de RDC.

O problema abordado neste trabalho, denominado *Problema de Posicionamento de Réplicas e Distribuição de Requisições* (PPRDR), é uma variante do PPR e consiste em encontrar os melhores locais para posicionar as réplicas dos conteúdos e definir quais servidores irão atender cada uma das requisições a fim de que as exigências de QoS das mesmas sejam satisfeitas sempre que possível e o tráfego dentro da rede seja minimizado.

Existem poucos trabalhos, segundo o conhecimento dos autores, para o PPRDR como enfocado aqui [Neves et al. 2010]. No entanto, existem vários trabalhos que tratam de problemas semelhantes usando abordagens distintas, tais como [Zhou and Xu 2007] que trata o PR e o PPR de forma isolada, e o apresentado por [Aioffi et al. 2005] que trata os mesmos problemas de maneira conjunta. O PDR é abordado na literatura de duas formas: através do uso do algoritmo de Dijkstra [Wauters et al. 2005] ou através de uma análise de custos [Bektas et al. 2007, Tenzakhti et al. 2004]. Neste trabalho, o PDR é tratado como parte integrante do PPRDR entretanto, devido à alta complexidade do problema, em algumas situações é necessário tratar esta distribuição separadamente, dividindo o PPRDR em dois subproblemas distintos, o PPR e o PDR.

O PPRDR, que pertence à classe *NP*-Difícil, é um problema de grande aplicação prática e para tratá-lo duas abordagens exatas para a sua versão *offline* são usadas: uma formulação matemática e um algoritmo de enumeração, proposto neste artigo, e que é capaz de resolver instâncias até 8 vezes maiores que os limites expostos pela literatura. O objetivo do uso destas abordagens distintas para a versão *offline* é encontrar limites de melhor qualidade e realizar um estudo mais aprofundado a respeito da melhor forma para tratar este problema real, que é tipicamente encontrado em ambientes de RDC. Para ambientes com alto grau de imprevisibilidade, o uso de modelos dinâmicos *offline* não é totalmente adequado, visto que estes modelos utilizam conhecimento da demanda futura e sabem, *a priori*, todas as mudanças que ocorrerão, tirando muitas vantagens disto. Supor a existência de tais conhecimentos em um ambiente de redes não é razoável, uma vez que não há como antever o volume de demandas nem sua origem, bem como não é possível prever com absoluta certeza as mudanças nos enlaces e o surgimento de conteúdos. Entretanto, os modelos *offline* são capazes de resolver bem os problemas e fornecem um bom parâmetro de comparação para os modelos *online*, que são mais adequados aos ambientes com alto grau de imprevisibilidade pelo fato de que os dados só são conhecidos com o passar do tempo. Uma outra razão para estudar os modelos *offline*, é que heurísticas para a versão *online* do problema nem sempre conseguem encontrar as soluções ótimas e, em alguns casos, as soluções produzidas por tais heurísticas são muito ruins [Neves 2011a].

Uma outra contribuição deste artigo é tratar de maneira conjunta o PR, o PPR e o PDR para arquivos extensos, analisando não apenas os custos para associar as requisições aos servidores, mas também uma série de questões que até então não haviam sido abordadas de maneira conjunta, como banda mínima para os clientes, carga nos servidores e presença de múltiplos conteúdos. Além dessas questões, uma série de outras, relacionadas ao fato dos conteúdos serem extensos, como possibilidade de uma requisição ser atendida por mais de um servidor ao mesmo tempo e a possibilidade do atendimento a uma requisição se estender por vários períodos de tempo também são consideradas.

O PPRDR que considera de maneira conjunta a replicação, o posicionamento das réplicas e distribuição eficiente das requisições, ainda é pouco explorado pela literatura apesar de sua visível importância econômica, fato este que justifica e motiva o desenvolvimento de técnicas eficientes para resolver este problema. Além disso, outra grande motivação deste trabalho é a possibilidade de melhorar a qualidade e a eficiência dos serviços prestados pelos provedores de RDC. Estas duas metas vão ao encontro da demanda mundial por melhor aproveitamento de recursos vivenciada neste século, onde simplesmente apresentar soluções para os problemas não é mais suficiente. É necessário apresentar soluções eficientes, que satisfaçam as necessidades dos clientes e que aproveitem de maneira racional os recursos disponíveis.

O restante do trabalho está organizado da seguinte maneira. Na Seção 2 é feita uma descrição PPRDR. A Seção 3 expõe uma formulação matemática para o problema. Já na Seção 4 é apresentado um modelo de fluxo em rede para resolver o PDR. A Seção 5 descreve o algoritmo enumerativo proposto. A Seção 6 expõe os resultados obtidos, bem como uma análise dos mesmos, e na Seção 7 encontram-se as conclusões do trabalho.

2. O Problema de Posicionamento de Réplicas e Distribuição de Requisições

O PPRDR consiste em encontrar o melhor posicionamento para as réplicas dentro da rede sobreposta e redistribuir as requisições entre os servidores, com o objetivo de reduzir a carga da rede sem violar as restrições de QoS das requisições. Este problema difere do PPR pelo fato de que as requisições são tratadas individualmente, e não de maneira aglomerada. Assim, requisições originadas de um mesmo ponto para um mesmo conteúdo podem ser tratadas por servidores diferentes caso isso seja vantajoso.

Para lidar com a natureza dinâmica do problema (surgimento de requisições, conteúdos, etc.) o horizonte de planejamento é discretizado em partes menores, chamados períodos, que tem duração, para efeito deste trabalho, de 60 segundos, porém estudos para analisar a influência dos tamanhos dos períodos e o *trade-off* que pode surgir entre estes tamanhos e os custos operacionais das RDCs serão alvo de estudos futuros.

Em cada período, podem surgir novas requisições e conteúdos. Alguns conteúdos podem ser removidos, algumas requisições podem deixar de existir, caracterizando o fim do atendimento e as condições da rede, tais como RTT (*Round Trip Time*) e atrasos nos canais, podem mudar.

Uma requisição, neste trabalho, consiste em uma identificação, um conteúdo exigido e quatro informações relacionadas à QoS: atraso máximo, banda mínima, banda máxima e atraso local. Considere que cada cliente se conecta a um servidor da RDC (chamado de servidor origem), faz uma requisição e, quando esta última for plenamente atendida, o cliente é desconectado. O atraso local mencionado anteriormente representa o atraso entre o computador do cliente e o servidor origem; O atraso máximo representa o valor máximo de atraso tolerado pelo cliente. As bandas mínima e máxima representam respectivamente o que o cliente deseja e qual a capacidade máxima deste cliente em termos de banda. Estes quatro elementos de QoS permitem que cada requisição seja tratada de modo diferenciado pela RDC. Existem trabalhos na literatura [Bartolini et al. 2003] que tratam as requisições forma aglomerada, onde as requisições são agrupadas e tratadas de forma conjunta como “carga”. Como no modelo usado aqui cada uma das requisições possui necessidades diferentes em termos de QoS esta aglomeração não é possível sem a criação de classes de requisições. Classes de requisições não são sempre a melhor saída uma vez que o uso desta estratégia obriga os clientes a se submeterem à uma das classes existentes, podendo ocorrer, em alguns casos, que nenhuma das classes satisfaça por completo as expectativas dos clientes. Além disso, o uso de aglomeração pode dificultar o atendimento parcial das demandas (quando não é possível atender todas as requisições) uma vez que fica difícil determinar quais as requisições devem ser atendidas dentro da demanda aglomerada. Estes dois limitadores para a aglomeração de requisições estimulam o tratamento individual das requisições, que é a abordagem utilizada aqui.

A redistribuição de requisições é usada como mecanismo de balanceamento de carga na rede com objetivo de reduzir a carga nos servidores e melhorar a qualidade do serviço. Para fazer a redistribuição de requisições no PPRDR alguns fatores precisam ser considerados. Uma requisição pode ser redirecionada apenas para servidores que possuem réplicas do conteúdo desejado pela requisição. As restrições de QoS também precisam ser averiguadas. Se após a redistribuição é verificado que estas restrições são violadas, então a solução resultante da redistribuição é de baixa qualidade e deve ser penalizada. Em alguns casos, pode ser necessário alterar o posicionamento e número de

réplicas de todos os conteúdos na RDC para otimizar a distribuição de requisições.

O posicionamento das réplicas está ligado às capacidades de armazenamento dos servidores e aos custos de comunicação entre os mesmos. Neste trabalho, os servidores possuem limites de banda e capacidades de disco heterogêneas. No início, os conteúdos se encontram em um único servidor (chamado origem do conteúdo), podendo um ou mais conteúdos ter o mesmo servidor de origem, e só depois eles são replicados pela rede. Devido à existência de custos associados com o transporte destas réplicas, a relação custo-benefício entre a redução de carga na rede e o custo de transporte das réplicas deve ser analisada antes da replicação. Estes fatores tornam o posicionamento das réplicas um problema não trivial, no qual uma decisão precipitada pode acarretar grandes custos, inviabilidades nos requisitos de QoS ou ambos.

Neste trabalho, as requisições possuem demanda divisível, o que significa que uma requisição pode ser atendida por múltiplos servidores simultaneamente, que é uma abordagem comum em redes *Peer-to-Peer*. Isto possibilita o melhor uso dos recursos, utilizando ao máximo as bandas dos servidores com menor custo de entrega de conteúdos, reduzindo o custo total. Além disso, os conteúdos possuem tamanhos heterogêneos.

Mudanças na rede podem ocorrer ao longo do tempo através da mudança no atraso de comunicação entre servidores. Claro que a mudança em um enlace da rede pode provocar mudança no custo de comunicação entre vários servidores, sendo necessário recalcular os custos para todos os servidores que usam o enlace alterado para se comunicar. Como mecanismos eficientes para efetuar estes cálculos não fazem parte do escopo desta trabalho considera-se que o custo de comunicação entre cada par de servidores afetados será informado quando ocorrer uma mudança na rede.

Outras duas características do problema que também podem ser tratadas de maneira dinâmica são o surgimento e a remoção de conteúdos dentro da RDC. Assim, para este trabalho, optou-se por modelar o surgimento e remoção de conteúdos da RDC através da atribuição de um tempo de vida para cada conteúdo.

Para modelar a dinâmica de requisições considera-se que estas podem surgir e terminar dentro de um horizonte de planejamento. Uma requisição possui demanda por um conteúdo, mas esta demanda não é persistente. Uma vez que o conteúdo tenha sido obtido pelo cliente, a requisição é encerrada, o que caracteriza o fim do atendimento. O mais lógico a se pensar, neste caso, é que a requisição possui demanda por um conteúdo por um determinado número de unidades de tempo. Em ambientes de rede com QoS, uma qualidade mínima é exigida pelas requisições, contudo geralmente não há imposições sobre a qualidade máxima. Isto leva a um ponto chave da discussão sobre as demandas. Se o sistema da RDC for mantido de modo que todas as requisições tenham somente seus requisitos mínimos atendidos, ele (o sistema de RDC) pode estar sendo subutilizado.

Garantir os requisitos mínimos não quer dizer que o sistema da RDC não pode oferecer mais qualidade aos seus usuários. Alguns servidores podem ter banda sobrando e capacidade para atender ainda melhor os clientes. Por exemplo, uma requisição que exige uma banda mínima de 5 kbps para um conteúdo de 100 KB é atendida por um servidor que tem, no momento, 50 kbps de banda não utilizada. Se o sistema fosse mantido de modo que somente as exigências mínimas fossem atendidas, o servidor atenderia a requisição com apenas 5 kbps de sua banda, levando 20 segundos. Porém, se o sistema enviar mais

que o mínimo, apenas dois segundos seriam necessários. Contudo, para prover este tipo de atendimento otimizado é necessário que o servidor conheça o limite máximo da banda da requisição. Por exemplo, não faz sentido o servidor enviar o conteúdo utilizando os seus 50 kbps de banda excedente se o máximo que a conexão da requisição suporta é de 10 kbps. Considerar este tipo de atendimento otimizado torna o problema mais realista e abre amplas frentes de trabalho.

Em alguns trabalhos [Aioffi et al. 2005], os autores consideram que um período é suficientemente extenso para atender todas as requisições que surgem naquele período. Isto é possível devido ao fato de que os conteúdos tratados nestes trabalhos são geralmente pequenos (até dois Megabytes), e os períodos de tempo suficientemente grandes para que todas as requisições pendentes sejam atendidas. Assim, pode-se considerar que as requisições não se propagam de um período para outro. Tais suposições não podem ser feitas para arquivos grandes (vídeos, por exemplo). Frequentemente, estes conteúdos consomem muito tempo para sua transferência, o que torna irreal a possibilidade de supor um intervalo de tempo suficientemente grande para que todas as requisições sejam atendidas. Também há o fato de que é impossível prever com exatidão em que instante de tempo dentro do período as requisições irão chegar. Isto implica que, se uma requisição chegar no meio de um período de tempo ela terá que esperar até o próximo período para começar a ser atendida. Caso se estipule um intervalo de tempo muito grande, uma requisição que se encontra na situação de espera pode ter que aguardar durante muito tempo. Por isso, o ideal é que os períodos de tempo sejam tão curtos quanto possível, fazendo com que as requisições se propaguem para outros períodos e diminuindo o tempo de espera para que uma requisição comece a ser atendida.

Assim sendo, as características do PPRDR que é abordado neste trabalho são: 1) Servidores Capacitados - existência de limites, heterogêneos, de banda e espaço em disco nos servidores; 2) Múltiplos Conteúdos - existência de mais de um conteúdo na rede; 3) Dinâmico - mudanças na rede, nos conteúdos e nas demandas ocorrem ao longo tempo; 4) *Offline* - mudanças que ocorrem ao longo do tempo são conhecidas *a priori*; 5) Atendimento por Múltiplos servidores - possibilidade de atendimento de uma demanda por mais de um servidor; 6) QoS - presença de requisitos de banda e atraso nas requisições e 7) Atendimento maximizado - tentativa de atender melhor os clientes sempre que possível.

3. Formulação Matemática

Nesta seção é apresentada uma formulação matemática para o PPRDR. Seja R o conjunto de requisições a serem atendidas, S o conjunto de servidores da RDC, C o conjunto de conteúdos a serem replicados e T o conjunto de períodos de tempo. Sejam também as constantes $\delta = 60$ a duração do período (em segundos), $origem(i)$ o servidor de origem da requisição i, $G(i)$ o conteúdo exigido pela requisição i, $ld(i)$ o atraso local da requisição i (em segundos), BR_i a banda mínima da requisição i (em bytes/segundo), BX_i banda máxima da requisição i (em bytes/segundo), TD_i o atraso máximo permitido pela requisição i. Além disso sejam L_k o tamanho do conteúdo k (em bytes) B_k período em que o conteúdo k é disponibilizado, E_k período em que o conteúdo k é removido da RDC, O_k o servidor origem do conteúdo k, AS_j o espaço em disco disponível no servidor j (em bytes) e MB_j a banda máxima do servidor j (em bytes/segundo). Sejam ainda $atraso(j, l, t)$ o atraso de comunicação entre dois servidores (em segundos) no período t e $RTT(j, l, t)$ o tempo que uma mensagem (ou pacote) leva para percorrer o caminho de

um servidor j para um outro servidor l e voltar ao primeiro em um período t dado por $RTT(j,l,t) = atraso(j,l,t) + atraso(l,j,t)$ (expresso em segundos).

A Formulação, denominada *FD*, utiliza as seguintes variáveis: x_{ijt} variável contínua que representa a fração do conteúdo solicitado pela requisição i entregue pelo servidor j no período t; A variável binária y_{kjt} assume 1 se o conteúdo k está replicado no servidor j no período t, e 0 caso contrário; A variável contínua b_{it} é o *backlog* da requisição i no período t, ou seja, é o montante da demanda de i que deveria ser atendido em t mas que é postergado para algum período seguinte; E a variável binária w_{kjlt} que assume 1 se o conteúdo k é copiado pelo servidor j a partir do servidor l no período t, e 0 caso contrário.

Além das constantes já definidas, as seguintes definições são utilizadas: D_{it} é a demanda da requisição i no período t (em bytes); c_{ijt} é o custo de atender a requisição i pelo servidor j, no período t, dado por $c_{ijt} = (\omega + RTT(origin(i), j, t)) \times BR_i$ se $\omega \leq TD_i$, Senão $c_{ijt} = ((\omega + RTT(origin(i), j, t)) \times BR_i) + 1000 \times (\omega - TD_i) + 1000$ onde $\omega = atraso(origem(i), j, t) + ld(i)$; p_{it} penalidade por fazer *backlog* da requisição i no período t dada por $p_{it} = max(c_{ijt}) \times 2, \forall i \in R, \forall t \in T$; E h_{kjlt} custo de replicar o conteúdo k no servidor j a partir do servidor l no período t. Nos experimentos feitos, este custo é sempre dado pelo tamanho do conteúdo k, ou seja, $h_{kjlt} = L_k \forall k \in C$.

A função objetivo, exposta em (1), minimiza o custo de entrega dos conteúdos para os clientes bem como a quantidade de *backlog* feitos ao longo do tempo e o custo de replicação. As restrições (2) associam as variáveis x_{ijt} e b_{it}, dizendo que a soma das quantidades entregues no período atual, mais a quantidade que se ficará devendo para o período posterior é igual à demanda do período atual mais o que se ficou devendo do período anterior. As restrições (3) exigem que o fluxo total entregue por um servidor seja menor ou igual à sua capacidade máxima. As restrições (4) impedem que seja entregue ao cliente uma banda maior do que ele suporta. As restrições (5) exigem que uma requisição seja plenamente atendida. As restrições (6) condicionam que uma requisição só pode ser atendida por um servidor que possua uma réplica do conteúdo exigido. As restrições (7) e (8) controlam o número de réplicas de um conteúdo, afirmando que no mínimo uma réplica deve existir durante o tempo de vida do conteúdo e que nenhuma réplica pode existir fora do tempo de vida. As restrições (9) e (10) fazem com que, no período de submissão, apenas o servidor origem de um conteúdo possua uma réplica. As restrições (11) garantem que réplicas não surjam espontaneamente. As restrições (12) exigem que uma replicação ocorra a partir de um servidor que possua o conteúdo replicado. As restrições (13) dizem que a soma dos tamanhos dos conteúdos em um servidor não pode exceder o espaço em disco disponível. As demais restrições são as de integralidade e não negatividade. Esta formulação é apresentada pelos autores deste trabalho em [Neves et al. 2010] e transcrita aqui por razões didáticas.

$$Min \quad \sum_{i \in R} \sum_{j \in S} \sum_{t \in T} c_{ijt} x_{ijt} + \sum_{i \in R} \sum_{t \in T} p_{it} b_{it} + \sum_{k \in C} \sum_{j \in S} \sum_{l \in S} \sum_{t \in T} h_{kjlt} w_{kjlt} \tag{1}$$

$S.a.$

$$\sum_{j \in S} L_{G(i)} x_{ijt} - b_{i(t-1)} + b_{it} = D_{it}, \forall i \in R, \forall t \in [B_{G(i)}, E_{G(i)}], \tag{2}$$

$$\sum_{i \in R} L_{G(i)} x_{ijt} \leq \delta MB_j, \ \forall j \in S, \forall t \in T, \tag{3}$$

$$\sum_{j \in S} L_{G(i)} x_{ijt} \leq \delta B X_i, \ \forall i \in R, \forall t \in T, \tag{4}$$

$$\sum_{j \in S} \sum_{t \in T} x_{ijt} = 1, \ \forall i \in R, \tag{5}$$

$$y_{G(i)jt} \geq x_{ijt}, \ \forall i \in R, \forall j \in S, \forall t \in T, \tag{6}$$

$$\sum_{j \in S} y_{kjt} \geq 1, \ \forall k \in C, \forall t \in [B_k, E_k], \tag{7}$$

$$y_{kjt} = 0, \ \forall k \in C, \forall j \in S, \forall t \notin [B_k, E_k], \tag{8}$$

$$y_{kO_kB_k} = 1, \ \forall k \in C, \tag{9}$$

$$y_{kjB_k} = 0, \ \forall k \in C, \forall j \in \{S | j \neq O_k\}, \tag{10}$$

$$y_{kj(t+1)} \leq \sum_{l \in S} w_{kjlt}, \ \forall k \in C, \forall j \in S, \forall t \in T, \tag{11}$$

$$y_{kjt} \geq w_{kljt}, \ \forall k \in C, \forall j, l \in S, \forall t \in T, \tag{12}$$

$$\sum_{k \in C} L_k \ y_{kjt} \ \leq \ AS_j, \ \forall j \in S, \forall t \in T, \tag{13}$$

$$x_{ijt} \in [0, 1], \ \forall i \in R, \forall j \in S, \forall t \in T, \tag{14}$$

$$y_{kjt} \in \{0, 1\}, \ \forall j \in S, \forall k \in C, \forall t \in T, \tag{15}$$

$$b_{it} \geq 0, \ \forall i \in R, \forall t \in T, \tag{16}$$

$$w_{kjlt} \in \{0, 1\}, \ \forall j, l \in S, \forall k \in C, \forall t \in T. \tag{17}$$

4. O Problema de Distribuição de Requisições

A formulação apresentada na Seção 3 consegue resolver instâncias (problemas teste) de até 50 servidores [Neves 2011a], contudo, mesmo para estas instâncias, consideradas de pequeno porte, existe dificuldade para provar a otimalidade das soluções encontradas. Neste sentido, uma das propostas para tentar fornecer soluções exatas para o PPRDR, é segmentar este problema em dois subproblemas, o Problema de posicionamento de Réplicas (PPR) e o Problema de Distribuição de Requisições (PDR), tratado nesta seção. O PDR consiste em, dado um conjunto de servidores com réplicas dos diversos conteúdos já posicionadas, dividir da melhor maneira possível as requisições respeitando as capacidades dos servidores e tentando atender a QoS dos clientes sempre que possível. Em [Neves et al. 2010] é apresentada uma formulação matemática para este problema. Esta seção apresenta uma nova abordagem exata de resolução através do uso de um modelo de fluxo em rede. Para isso um Problema de Fluxo de Custo Mínimo (PFCM) [Ahuja et al. 1993] é resolvido em uma rede específica, onde os vértices representam requisições, servidores, origem de fluxo e destino de fluxo. Demandas específicas são associadas aos vértices de acordo com o que eles representam. Os arcos que ligam os vértices possuem um custo associado, uma capacidade e uma classe. O custo é o valor pago para transmitir uma unidade de fluxo pelo arco e a capacidade representa a quantidade máxima de fluxo que o arco pode transportar. Já as classes dos arcos representam as relações entre os vértices. Como os vértices representam entidades diferentes do problema a relação entre eles nem sempre é a mesma, e esta diferença entre os relacionamentos é modelada pelas classes de arcos. Uma solução para o PFCM nesta rede é equivalente à uma solução para o PDR. A rede mencionada pode ser construída através dos seguintes passos: 1) Seja $R* \subseteq R$ o conjunto de requisições ativas (requisições recém chegadas ou não atendidas por completo) em um período. Criar um par de vértices i' e i'' para

cada requisição $i \in R*$; 2) Criar um vértice para cada servidor; 3) Criar um vértice que representa um servidor de capacidade infinita, i.e., um servidor de *backlog* (SB); 4) Criar vértices para a origem (FS) e destino (FF) do fluxo; 5) Estabelecer demandas para os vértices da seguinte maneira: FQ unidades em FS, $-FQ$ em FF e zero nos demais vértices, onde $FQ = \sum_{i \in R*} D_i$, $D_i = \delta BX_i + B_i$ e B_i é o *backlog* da requisição i nos períodos anteriores ao período em consideração. Em outras palavras, FQ é a soma das demandas dos clientes; 6) Criar arcos ligando FS a cada vértice i''. Este arcos pertencem à classe *a* e têm custo zero e capacidade igual à demanda de cada cliente; 7) Criar arcos ligando os vértices i'' aos vértices i'. Para cada requisição, um arco é criado ligando os dois vértices que a representam. Estes arcos pertencem à classe *f*, têm custo zero e capacidade igual δBX_i; 8) Criar arcos ligando os vértices i' aos vértices que representam os servidores para modelar o atendimento. Estes arcos podem pertencer a duas classes. Arcos da classe *b* representam o atendimento viável, logo vão ligar uma requisição aos servidores que possuem uma réplica do conteúdo solicitado. Estes arcos Têm custo $c_{ij}/L_{G(i)}$ e capacidade δBX_i. Arcos da classe *c* modelam o atendimento inviável, logo ligam as requisições aos servidores que não possuem o conteúdo desejado. Estes arcos possuem capacidade δBX_i e um custo muito alto. É importante mencionar que neste modelo só existe um arco ligando uma requisição e um servidor. Em outras palavras, não pode haver um arco da classe *b* e um arco da classe *c* partindo de uma requisição para um mesmo servidor; 9) Ligar os vértices i'' ao servidor de *backlog*. Estes arcos pertencem à classe *d* e têm custo p_i e capacidade infinita; 10) Por último, é necessário criar arcos ligando os servidores (incluindo o servidor de *backlog*) ao vértice destino do fluxo. Estes arcos tem capacidade δMB_j, custo zero e pertencem à classe *e*.

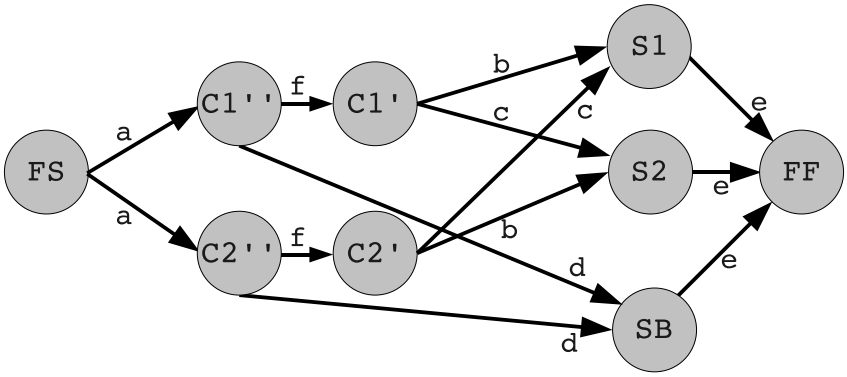

Figura 1. Modelo de Fluxo em Rede para o PDR

A Figura 1 mostra a rede construída usando os passos mencionados para uma instância com duas requisições ($C1$ e $C2$) e dois servidores ($S1$ e $S2$). Neste exemplo $S1$ possui uma réplica do conteúdo desejado por $C1$ e $S2$ possui uma réplica do conteúdo desejado por $C2$. Depois que todos os dez passos estão concluídos, basta resolver o PFCM nesta rede usando um algoritmo apropriado [Ahuja et al. 1993] e a solução resultante é equivalente a uma solução para o PDR.

Este modelo é capaz de lidar com o *backlog* de períodos anteriores porque permite que uma quantidade de fluxo maior que a capacidade das requisições atinja o primeiro vértice das requisições. Como a quantidade de fluxo FQ é igual à soma das demandas, todos os arcos da classe *a* são saturados, o que significa que um fluxo igual à demanda de

i atinge i''. Os arcos da classe *f* tem capacidade igual à capacidade máxima de *download* das requisições, logo, se uma requisição possui demanda maior que sua capacidade, a demanda excedente deve passar pelos arcos da classe *d*, que representam o *backlog*. Para esclarecer o funcionamento do modelo o seguinte exemplo deve ser observado: Suponha que no primeiro período uma requisição i com capacidade de download de 200 bytes recebe apenas 190. Os 10 bytes restantes são o *backlog* de i no período 1 e são passados como demanda de i para o período seguinte. Como a demanda da requisição é dada por $D_i = \delta BX_i + B_i$, $D_i = 210$ no segundo período. Agora suponha que 195 bytes são entregues no período 2. Como pode ser calculado, o valor do *backlog* de i no segundo período é 15 bytes e assim sucessivamente. Caso o tamanho do conteúdo solicitado seja de 300 bytes, a demanda de i pode ser plenamente atendida no terceiro período e, caso novos *backlogs* não aconteçam, i pode ser removida do conjunto de requisições ativas. O fluxo que chega em um vértice do tipo i' deve seguir por arcos das classes *b* ou *c*. Se não há arcos da classe *b* disponíveis, um algoritmo de PFCM deve transferir o fluxo que dos arcos da classe *f* para arcos da classe *d*. As restrições de banda nos servidores são asseguradas pelos arcos da classe *e*. Quando um servidor excede sua capacidade, um algoritmo de PFCM deve transferir o fluxo excedente para outros arcos da classe *b* ou para arcos da classe *d* se necessário. Apesar de arcos da classe *c* nunca serem utilizados em uma solução ótima do PFCM, eles são necessários porque alguns algoritmos para este problema utilizam arcos caros no processo de construção da solução [Ahuja et al. 1993].

A complexidade computacional do modelo de fluxo em rede pode ser calculada da seguinte maneira: O modelo utiliza $2|R|$ vértices para as requisições , $|S| + 1$ vértices para os servidores e 2 vértices adicionais para origem e destino de fluxo. Logo a criação dos vértices leva $2|R| + |S| + 1 + 2$ passos, que tem uma complexidade $\mathrm{O}(|R| + |S|)$ no pior caso. O modelo também utiliza $|R|$ arcos da classe *a*, $|R|$ arcos da classe *f*, $|R|$ arcos da classe *d* e $|S| + 1$ arcos da classe *e*. Para as classes *b* e *c* juntas $|S||R|$ arcos são criados, resultando em $3|R| + |S| + 1 + |S||R|$ arcos que tem complexidade $\mathrm{O}(|S||R|)$. Como o número total de passos para criar a rede é dado pelo número de arcos mais o número de vértices, a complexidade do processo de criação da rede é dada por $\mathrm{O}(|R| + |S|) + \mathrm{O}(|S||R|) = \mathrm{O}(|S||R|)$, que é uma complexidade polinomial. Como a criação da rede pode ser feita em tempo polinomial e existem algoritmos polinomiais para o PFCM [Ahuja et al. 1993], fica claro que o PDR pode ser resolvido em tempo polinomial e logo pertence à classe P de problemas.

5. Um Algoritmo Enumerativo para o PPRDR

Como já mencionado, formulação *FD* encontra dificuldades na resolução de algumas instâncias com 50 servidores e praticamente é incapaz de resolver instâncias maiores do que isso. As maiores dificuldades foram encontradas nas instâncias em que os recursos (espaço em disco e banda nos servidores) são mais escassos e os canais de comunicação são assimétricos. Já nas instâncias de maior porte o principal problema encontrado é a alta demanda por memória da formulação *FD* [Neves 2011a]. Considerando a questão do consumo de memória, um algoritmo de enumeração para a resolução do PPRDR, chamado *Smart Enumeration Algorithm* (*SEA*), é proposto nesta seção. O principal objetivo da proposta deste algoritmo é encontrar soluções ótimas ou limites de boa qualidade para as instâncias onde *FD* é pouco eficaz e não obter um algoritmo exato eficiente quando comparado com outros métodos exatos.

A estratégia do *SEA* é a mesma utilizada pelo algoritmo de *branch-and-bound* [Zionts 1974], que consiste em descrever o espaço de soluções através de uma árvore binária de decisões e explorar esta árvore para encontrar o ótimo global. Um ponto chave para este tipo de estratégia é como explorar a árvore de maneira eficiente. Para isso, o *SEA* utiliza mecanismos de poda, que impedem a exploração de ramos que possam levar à soluções ruins, garantindo que as soluções ótimas não são descartadas.

A árvore que descreve o espaço de soluções do *SEA* representa possíveis posicionamentos de réplicas em servidores e pode ser obtida através dos seguintes passos: 1) Criar tuplas $< t, s, c >$ para ser os nós da árvore, onde $t \in T$, $s \in S$ e c pertence ao conjunto de conteúdos ativos em t (Conteúdos que possuem t em seu tempo de vida). Estas tuplas devem ser armazenadas em uma lista l; 2) Ordene l em ordem ascendente de i) períodos, ii) servidores e iii) conteúdos; 3) Suponha que a notação $x : xs$ representa uma lista com nó cabeça x e cauda xs. Seja $f(ramo, x : xs)$ uma função recursiva que recebe um ramo e uma lista de nós, remove o elemento na cabeça da lista (x), liga este elemento em $ramo$ criando dois sub-ramos, chamados de $ramoS$ e $ramoN$, que representam, respectivamente, a inserção do conteúdo $x.c$ no servidor $x.s$ no período $x.t$ e a **não** inserção de $x.c$ em $x.s$ no período $x.t$. A função f então se invoca recursivamente para ambos os ramos, usando a lista xs como parâmetro ($f(ramoS, xs)$ e $f(ramoN, xs)$). A função f retorna quando a lista recebida como parâmetro é vazia.

A árvore construída pelos passos acima é binária e completa com altura dada por $|T| \times |S| \times |C|$ e $2^{|T|\times|S|\times|C|}$ folhas. Uma solução completa para o posicionamento de réplicas do PPRDR é obtida percorrendo todo o caminho entre a raiz e uma folha, o que significa que para atingir uma solução completa para o posicionamento de réplicas, $|T| \times |S| \times |C|$ decisões binárias precisam ser tomadas. Em cada nó, exitem apenas dois possíveis caminhos para atingir uma folha: $ramoS$ e $ramoN$. Caso $ramoS$ seja escolhido, para um nó $< t, s, c >$, significa que o conteúdo c está posicionado no servidor s no período t. Caso o caminho escolhido seja $ramoN$, isto significa que o conteúdo c **não** está posicionado no servidor s no período t.

A solução ótima do PPRDR pode ser obtida ao através de uma busca em todos os ramos da árvore. Contudo, esta árvore pode se tornar demasiadamente grande mesmo para instâncias de pequeno porte, o que torna a construção de toda a árvore uma opção pouco razoável. Para explorar a estrutura de um modo mais eficiente optou-se por construir a árvore sob demanda, o que significa que apenas um ramo é utilizado por vez.

Apesar de uma busca ramo a ramo na árvore ser perfeitamente possível, pode levar muito tempo para explorar toda a estrutura. Devido a isto, alguns mecanismos de poda são utilizados com o objetivo de evitar a exploração desnecessária de alguns ramos da árvore e assim reduzir o tempo computacional necessário para encontrar as soluções ótimas. Uma descrição detalhada destes mecanismos pode ser encontrada em [Neves 2011a]. É importante mencionar que os mecanismos de poda propostos somente evitam a exploração de ramos que levem à soluções inviáveis ou à soluções piores do que a melhor solução obtida até o momento, e que nenhum destes mecanismos descarta ramos onde podem ser encontrados soluções ótimas para o PPRDR.

Uma vez que se obtém os posicionamentos de réplicas para cada período, o PDR é resolvido utilizando o modelo de fluxo em rede apresentado na Seção 4, calculando

os custos para cada período e o conjunto de requisições ativas para o período seguinte. Assim, para cada solução parcial do PPR (soluções para um período), um PFCM é resolvido, gerando uma solução parcial para o PPRDR. Como o *SEA* é capaz de gerar todos os possíveis posicionamentos de réplicas, o algoritmo é caracterizado como método exato enumerativo, pois consegue encontrar as soluções ótimas através da enumeração parcial de variáveis. Uma descrição mais detalhada do *SEA* pode ser obtida em [Neves 2011a].

6. Resultados Computacionais

Os algoritmos apresentados foram implementados em C++ usando g++ versão 4.3 e executados em um Quad-Core com 2.83 GHz/core, 8 Gigabytes de RAM usando Linux (kernel 2.6). Para resolver as formulações e os problemas de fluxo em rede o resolvedor CPLEX 11.2 [cpl 2008] foi utilizado. As instâncias usadas nos testes computacionais estão disponíveis no site do projeto LABIC [lab 2005], juntamente com um relatório técnico sobre as mesmas [Neves 2011b].

Tabela 1. Melhores Resultados Exatos: Instâncias de Pequeno Porte

(a) Instâncias com Restrições Fracas de Recurso

Instancia	Tempo (s)	F.O.	Método
10A01	1.21	3.00E+005	FD
10A02	1.04	2.72E+005	FD
10A03	1.15	2.53E+005	FD
10A04	1.28	3.49E+005	FD
10A05	0.84	2.65E+005	FD
20A01	3.47	2.00E+006	FD
20A02	8.93	2.14E+006	FD
20A03	7.50	2.07E+006	FD
20A04	6.76	2.08E+006	FD
20A05	6.23	2.10E+006	FD
30A01	20.77	2.10E+006	FD
30A02	14.86	2.16E+006	FD
30A03	15.25	2.15E+006	FD
30A04	17.51	2.22E+006	FD
30A05	19.12	2.24E+006	FD
50A01	59.98	4.63E+005	FD
50A02	63.00	8.21E+005	FD
50A03	62.16	8.78E+005	FD
50A04	60.19	6.49E+005	FD
50A05	58.57	7.54E+005	FD
10B06	6.02	2.00E+006	FD
10B07	5.00	2.14E+006	FD
10B08	4.86	2.07E+006	FD
10B09	4.98	2.08E+006	FD
10B10	5.07	2.10E+006	FD
20B06	29.43	4.20E+006	FD
20B07	27.58	4.33E+006	FD
20B08	31.04	4.29E+006	FD
20B09	29.43	4.43E+006	FD
20B10	28.57	4.48E+006	FD
30B06	92.25	6.56E+006	FD
30B07	80.79	6.34E+006	FD
30B08	73.50	6.56E+006	FD
30B09	72.12	6.46E+006	FD
30B10	68.56	6.46E+006	FD
50B06	278.16	1.12E+007	FD
50B07	269.80	1.09E+007	FD
50B08	357.13	1.05E+007	FD
50B09	347.39	1.10E+007	FD
50B10	315.46	1.10E+007	FD

(b) Instâncias com Restrição Fortes de Recurso

Instância	Tempo (s)	F.O.	Método
10C11	38.22	2.10E+006	FD
10C12	9.14	2.16E+006	FD
10C13	15.14	2.15E+006	FD
10C14	30.30	2.22E+006	FD
10C15	32.93	2.24E+006	FD
20C11	666.83	4.78E+006	FD
20C12	230.75	4.42E+006	FD
20C13	317.04	4.50E+006	FD
20C14	520.43	4.67E+006	FD
20C15	175.98	4.55E+006	FD
30C11	433.04	6.70E+006	FD
30C12	211.01	6.39E+006	FD
30C13	196.27	6.61E+006	FD
30C14	157.09	6.53E+006	FD
30C15	188.90	6.52E+006	FD
50C11	523.88	1.13E+007	FD
50C12	379.95	1.09E+007	FD
50C13*	1229.67	1.15E+007	FD
50C14*	1239.84	1.18E+007	FD
50C15	666.31	1.12E+007	FD
10D16	8133.05	8.74E+006	FD
10D17	24547.20	8.83E+006	FD
10D18	2337.77	8.74E+006	FD
10D19	3448.68	8.53E+006	FD
10D20	4035.07	8.62E+006	FD
20D16**	10802.10	1.81E+007	FD
20D17**	10800.38	1.73E+007	FD
20D18**	10803.62	1.77E+007	FD
20D19**	10801.41	1.71E+007	FD
20D20**	10800.35	1.74E+007	FD
30D16**	10801.40	2.66E+007	FD
30D17**	10802.10	2.60E+007	FD
30D18**	10800.90	2.54E+007	FD
30D19**	10806.20	2.69E+007	FD
30D20**	10801.60	2.76E+007	FD
50D16**	10800.00	2.58E+008	SEA
50D17**	10800.60	4.13E+007	FD
50D18**	10800.40	4.80E+007	FD
50D19**	10801.20	2.41E+008	FD
50D20**	10802.70	4.48E+007	FD

A Tabela 1 sumariza os resultados obtidos para as instâncias de pequeno porte. Para cada tabela, na primeira coluna estão as instâncias tratadas expressas no formato xLy, sendo x o número de servidores, L a classe da instância [Neves 2011b] (onde A é a classe com mais recursos e D a com menos recursos) e y a identificação da instâncias. A segunda coluna mostra o tempo computacional, em segundos, do melhor resultado. A terceira coluna mostra o valor da função objetivo da solução encontrada e a última coluna

mostra qual o algoritmo que conseguiu esta solução. Na Tabela 1(a) estão os resultados para as instâncias com pouca restrição de recursos. Note que nestes casos, a formulação *FD* sempre consegue os melhores resultados. Já na Tabela 1(b) estão os resultados para as instâncias onde os recursos são mais escassos. Dentre estas instâncias, algumas (as que possuem a letra D entre os números) também apresentam canais assimétricos. Note que o *SEA* já aparece como melhor algoritmo para uma das instâncias, mostrando que mesmo para casos pequenos a formulação *FD* tem dificuldades para resolver o problema. Nas instâncias marcadas com "*", as soluções encontradas são ótimos não provados por limitações de memória no computador utilizado. Já nas instâncias marcadas com "**" as soluções encontradas são ótimos não provados devido ao fato de que o algoritmo atingiu o tempo limite permitido, que é de 10800 segundos nestes experimentos. Um fato importante a ser mencionado é que o *SEA* consegue encontrar as mesmas soluções que *FD* para diversos casos, contudo, *FD* encontra estas soluções em menos tempo computacional.

Tabela 2. Melhores Resultados Exatos: Instâncias de Grande Porte

(a) Instâncias com $|S| \leq 200$

Instância	Tempo (s)	F.O.	Método
100C01**	10801.20	2.38E+007	SEA
100C02**	10802.03	2.25E+007	SEA
100C03**	10801.00	2.37E+008	SEA
100C04**	10803.10	2.27E+007	SEA
100C05**	10800.01	2.26E+007	SEA
100D16**	10801.04	2.74E+009	SEA
100D17**	10802.53	2.14E+009	SEA
100D18**	10801.13	1.71E+009	SEA
100D19**	10803.81	2.10E+009	SEA
100D20**	10800.76	2.61E+009	SEA
200C01**	10802.18	2.00E+009	SEA
200C02**	10801.94	3.64E+009	SEA
200C03**	10800.16	2.80E+009	SEA
200C04**	10801.37	3.26E+009	SEA
200C05**	10803.14	5.04E+009	SEA

(b) Instâncias com $|S| > 200$

Instância	Tempo (s)	F.O.	Método
300C01**	10803.00	6.09E+009	SEA
300C02**	10801.52	1.25E+010	SEA
300C03**	10802.00	1.52E+010	SEA
300C04**	10801.68	1.26E+010	SEA
300C05**	10800.41	1.63E+010	SEA
400C01**	10801.98	1.97E+010	SEA
400C02**	10800.73	3.06E+010	SEA
400C03**	10802.37	2.90E+010	SEA
400C04**	10800.41	2.85E+010	SEA
400C05**	10801.08	2.58E+010	SEA

A Tabela 2 mostra os resultados obtidos para as instâncias de grande porte. Note que o *SEA* consegue os melhores resultados em todos os casos. Também é possível notar que em nenhuma destas instâncias o ótimo foi provado, contudo vale lembrar que o uso de abordagens exatas não serve apenas para encontrar as soluções ótimas, serve também para conseguir limites de boa qualidade e para extrair padrões de comportamento para serem incorporados em outros algoritmos.

7. Conclusões

Este trabalho descreve o Problema de Posicionamento de Réplicas e Distribuição de Requisições (PPRDR), expõe uma formulação matemática para a versão *offline* deste problema, propõe um modelo de fluxo em rede para a resolução de uma parte do PPRDR e um algoritmo enumerativo (*SEA*) como alternativa exata de resolução para grandes instâncias do problema, onde o uso da formulação matemática é limitado. O *SEA* é baseado na decomposição do PPRDR em dois subproblemas menores, o PPR e o PDR. Neste algoritmo, o PPR é resolvido por enumeração e o PDR é resolvido através do modelo de fluxo em rede mencionado. Uma análise da complexidade do modelo de fluxo em rede também é feita, mostrando que o PDR pode ser resolvido em tempo polinomial. Os resultados computacionais mostram que o *SEA* consegue melhor desempenho que a formulação matemática para todas as instâncias de grande porte e para uma instância de pequeno porte. Além disso, o modelo computacional tratado neste trabalho inclui uma

série de aspectos que não são tratados na literatura de maneira simultânea, tais como dinâmica nas condições da rede, surgimento e remoção de conteúdos e QoS.

Como Trabalhos futuros pretende-se fazer uso de heurísticas para a versão *online* do problema, usando os resultados aqui expostos como base de comparação.

Referências

AKAMAI. World Wide Web, www.akamai.com. Acessado em 03/2008.

Mirror Image. World Wide Web, www.mirror-image.com. Acessado em 01/2009.

(2005). LABIC. World Wide Web,http://labic.ic.uff.br/.

(2008). ILOG S.A., CPLEX 11 user's manual.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. (1993). *Network Flows:Theory, Algorithms, and Applications*. Prentice Hall, 1 edition.

Aioffi, W., Mateus, G., Almeida, J., and Loureiro, A. (2005). Dynamic content distribution for mobile enterprise networks. *IEEE Journal on Selected Areas in Communications*, 23(10):2022–2031.

Bakiras, S. and Loukopoulos, T. (2005). Combining replica placement and caching techniques in content distribution networks. *Computer Communications*, 28:1062–1073.

Bartolini, N., Presti, F., and Petrioli, C. (2003). Optimal dynamic replica placement in content delivery networks. Em *The 11th IEEE International Conference on Networks 2003. ICON2003* , pages 125–130.

Bektas, T., Oguz, O., and Ouveysi, I. (2007). Designing cost-effective content distribution networks. *Computers & Operations Research*, 34:2436–2449.

Kurose, J. and Ross, K. (2003). *Computer Networking: a Top-Down Approach Featuring the Internet*. Addison-Wesley.

Neves, T. (2011a). *Redes de Distribuição de Conteúdos: Abordagens Exatas, Heurísticas e Híbridas*. Tese de Doutorado, Universidade Federal Fluminense - UFF.

Neves, T. (2011b). Synthetic instances for a management problem in content distribution networks. Relatório Técnico, UFF.

Neves, T., Drummond, L., Ochi, L., Albuquerque, C., and Uchoa, E. (2010). Solving replica placement and request distribution in content distribution networks. *Electronic Notes in Discrete Mathematics*, 36:89–96.

Tenzakhti, F., Day, K., and Ould-Khaoua, M. (2004). Replication algorithms for the world-wide web. *Journal of Systems Architecture*, 50:591–605.

Wauters, T., Coppens, J., Dhoedt, B., and Demeester, P. (2005). Load balancing through efficient distributed content placement. Em *Next Generation Internet Networks*, pages 99–105.

Zhou, X. and Xu, C.-Z. (2007). Efficient algorithms of video replication and placement on a cluster of streaming servers. *Journal of Network and Computer Applications*, 30:515–540.

Zionts, S. (1974). *Linear and Integer Programming*. Prentice-Hall.