UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA
ESPECIALIZAÇÃO EM DESENVOLVIMENTO WEB

JOSÉ LUÍS TADEU LINS CARNEVALLI

MÓDULO AGENTE GERENCIÁVEL DE REPLICAÇÃO DE ARQUIVOS ESTÁTICOS PARA APLICAÇÕES PHP QUE UTILIZAM REDES DE DISTRIBUIÇÃO DE CONTEÚDO

MONOGRAFIA DE ESPECIALIZAÇÃO

LONDRINA
2012

JOSÉ LUÍS TADEU LINS CARNEVALLI

MÓDULO AGENTE GERENCIÁVEL DE REPLICAÇÃO DE ARQUIVOS ESTÁTICOS PARA APLICAÇÕES PHP QUE UTILIZAM REDES DE DISTRIBUIÇÃO DE CONTEÚDO

Monografia de especialização apresentada na Universidade Tecnológica Federal do Paraná – Campus Londrina como requisito parcial para obtenção do título de “Especialista em Desenvolvimento Web”.

Orientador: Prof. Dr. Elias Canhadas Genvigir.

LONDRINA

2012

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM TECNOLOGIA
ESPECIALIZAÇÃO EM DESENVOLVIMENTO WEB

TERMO DE APROVAÇÃO

Título da Monografia:

MÓDULO AGENTE GERENCIÁVEL DE REPLICAÇÃO DE ARQUIVOS ESTÁTICOS PARA APLICAÇÕES PHP QUE UTILIZAM REDES DE DISTRIBUIÇÃO DE CONTEÚDO

por

JOSÉ LUÍS TADEU LINS CARNEVALLI

Esta monografia foi apresentada às 17h00 do dia **01** de **fevereiro** de 2013 como requisito parcial para a obtenção do título de ESPECIALISTA EM DESENVOLVIMENTO WEB. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho _____.

(aprovado, aprovado com restrições ou reprovado)

_____
Prof. Elias Canhadas Genvigir
(UTFPR)

_____
Prof. André Luis dos Santos Domingues
(UTFPR)

_____
Prof. Guilherme Luiz Frufrek
(UTFPR)

Visto da coordenação:

_____
Prof. Thiago Prado de Campos
Coordenador da esp. em Desenvolvimento Web

_____
Prof. Walmir Eno Pottker
Coordenador de Pós-Graduação Lato Senso

DEDICATÓRIA

Dedico este trabalho a todos familiares, amigos e professores que sempre me apoiaram durante este curso, em especial ao meu pai Luiz Carlos, minha mãe Jeusa e minha namorada Carolina.

AGRADECIMENTOS

Aos meus pais, Luiz Carlos e Jeusa, pelo apoio incondicional em todos os momentos.

À minha namorada, Carolina, pela paciência e compreensão.

Ao Professor Elias, pela solicitude durante o acompanhamento e orientação deste trabalho.

Ao coordenador Thiago, por trazer a Londrina professores e profissionais experientes para proporcionar aos alunos uma grande oportunidade de atualização e aprendizado.

EPÍGRAFE

“É preciso ter cuidado.
É preciso saber da limitação do conhecimento do ser humano.”
(Antônio Abujamra)

RESUMO

CARNEVALLI, JOSE LUIS T. L. **Módulo agente gerenciável de replicação de arquivos estáticos para aplicações PHP que utilizam redes de distribuição de conteúdo.** 2012. 65 f. Monografia (Especialização em Desenvolvimento Web), Universidade Tecnológica Federal do Paraná. Londrina, 2012.

Este trabalho apresenta um estudo sobre estratégias de distribuição de requisições entre servidores web por meio da replicação de arquivos estáticos em servidores dispersos geograficamente. Analisa os modelos implementados pelas redes *Google Global Cache* e *Akamai*, bem como os métodos de aquisição *Origin-pull* e *Push*. Por fim, adota o método de aquisição *Push* e desenvolve de um agente de replicação facilmente integrável a uma aplicação em linguagem PHP que permite o gerenciamento de arquivos armazenados remotamente em uma rede de distribuição de conteúdo. Utiliza como metodologia de desenvolvimento uma versão adaptada do Processo Unificado, criando modelos em diagramas UML e codificação orientada a objetos.

Palavras chave: Redes de distribuição de conteúdo. Replicação de arquivos. Otimização de aplicações web.

ABSTRACT

This paper presents a study on strategies for distributing requests among web servers through replication of static files in geographically dispersed servers. Analyzes the models implemented by Akamai and Google Global Cache networks, and the methods of acquisition Origin-pull and Push. Finally, adopts the Push acquisition method and develops a replication agent easily integrated to a PHP application that allows managing files stored remotely on a content distribution network. The Development methodology uses an adapted version of the Unified Process, UML diagrams for creating models and object-oriented coding.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

CDN	Content Delivery Network
CMS	Content Management Systems
CSS	*Cascading Style Sheets*
FTP	File Transfer Protocol
GGC	Google Global Cache
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
IBOPE	Instituto Brasileiro de Opinião Pública e Estatística
JSON	Javascript Object Notation
kbps	Quilobits por Segundo
MD5	*Message-Digest Algorithm 5*
PHP	*PHP: Hypertext Preprocessor*
PTT	Ponto de Troca de Tráfego
REST	Representional State Transfer
RTT	Round Trip Time
SHA	*Secure Hash Algorithm*
SOAP	Simple Object Access Protocol
SSD	Solid-State Drive
UP	*Unified Process*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

No ano de 1994 tinha início no Brasil o serviço de Internet prestado ao consumidor final. Ainda em caráter experimental, apenas cinco mil usuários foram escolhidos para os primeiros testes do serviço que somente no mês de maio de 1995 começou a funcionar definitivamente (GUIZZO, 1999).

Desde então, a cada ano o número de usuários ativos da rede aumentou significativamente. De acordo com levantamento do Instituto Brasileiro de Opinião Pública e Estatística – IBOPE (2012), no segundo trimestre de 2012 o número de usuários que acessam a Internet a partir de domicílios, locais de trabalho, escolas ou *LAN Houses* chegou a 83,4 milhões, representando um crescimento de 1% ante o número apurado no primeiro trimestre do mesmo ano. Quando comparado com o segundo trimestre de 2011, em que foram registrados 77,8 milhões de usuários, o crescimento chega a 7%.

Acompanhando o aumento quantitativo de usuários com acesso à rede, a capacidade do acesso também segue com crescimento. A evolução do uso da Internet tem sido acompanhada pelo aumento do número de usuários que utilizam conexões mais velozes. Enquanto no terceiro trimestre do ano de 2007 a velocidade média das conexões estabelecidas no Brasil era de 704 kbps - quilobits por segundo, no primeiro trimestre de 2012 a velocidade média foi ampliada para 2168 kbps (BELSON; BERGQVIST; MÖLLER, 2012), exibindo um avanço superior a 300%.

Dispondo de uma conexão de maior capacidade, os usuários passam a despender maior tempo navegando por páginas da Internet e acessando conteúdos mais complexos, como imagens, áudio, vídeo e jogos online. Se uma conexão inferior a 1024 kbps é capaz de prejudicar a experiência do usuário no acesso a estes conteúdos com um elevado tempo de carregamento, da mesma forma, conexões de mais elevada velocidade tornam transferências de arquivos mais complexos praticamente imperceptíveis, disponibilizando o arquivo acessado em um intervalo de tempo muito pequeno.

Somados, os aumentos quantitativo e qualitativo do número de conexões estabelecidas com a Internet fizeram com que os serviços mais acessados necessitassem buscar meios de expandir sua capacidade de oferecer conteúdo.

Ao contrário de alguns serviços como telefonia via Internet e compartilhamento de arquivos, que são baseados em uma arquitetura *Peer-to-Peer* onde cada nó computacional realiza tanto as funções de cliente como de servidor, o tráfego do acesso a *websites* que disponibilizam uma grande quantidade de conteúdo é estabelecido por meio de uma arquitetura cliente-servidor (KUROSE; ROSS, 2009), em que qualquer incremento no número de clientes que acessam um determinado recurso disponibilizado, de alguma maneira, trará como consequência um aumento do consumo de recursos computacionais no servidor.

Assim sendo, por meio de uma revisão bibliográfica, o presente trabalho buscará analisar estratégias de distribuição do conteúdo que permitem que empresas como Google, Yahoo! e Facebook disponibilizem em seus produtos serviços de distribuição de mídias digitais, como imagens e vídeos, capazes de armazenar uma grande quantidade de arquivos e, ainda assim, responder de maneira otimizada a milhões de requisições por segundo.

Os conhecimentos adquiridos serão utilizados na implementação de um software capaz de oferecer um serviço de transferência de arquivos entre múltiplos servidores *web*, o qual deverá garantir a execução de regras visando à integridade e à disponibilidade dos arquivos armazenados, atendendo às necessidades específicas de *websites* com grande volume de arquivos de mídia.

2 JUSTIFICATIVA

O dinamismo da Internet delega às aplicações Web a obrigação de fornecer conteúdo a fim de entreter e oferecer algo de valor ao usuário. Qualquer conteúdo acessado por meio da Internet está armazenado em um servidor, cuja capacidade de processamento deve ser capaz de atender um grande número de usuários e mantê-los satisfeitos (COUTINHO et al., 2011).

Muitas vezes, apenas o hipertexto não é suficiente para atingir este objetivo. É necessária a vinculação de conteúdos mais complexos, como imagens digitais, arquivos de áudio, vídeo ou qualquer outro tipo de mídia.

Com o aumento da demanda por conteúdo digital, tornaram-se cada vez mais comuns os sites de conteúdos multimídia, cujo conteúdo principal não está expresso em hipertexto, mas em arquivos binários capazes de ampliar a experiência do usuário e transmitir muito mais informação do que seria possível se expressando exclusivamente de forma textual.

O caráter colaborativo da Internet resultou em uma série de aplicações em que usuários são motivados a produzir conteúdo ao mesmo tempo sobre um determinado tema, fomentando a participação direta de colaboradores e consumidores (KEPLER, 2011). O agora denominado “conteúdo colaborativo” é um dos principais responsáveis pela expansão demasiadamente rápida do conteúdo armazenado em muitas aplicações na Internet. Graças a este conceito, que insere o usuário como agente gerador de conteúdo, diversas aplicações foram concebidas exclusivamente com o intuito de promover o conteúdo publicado pelos seus usuários.

Websites como Wikipedia, Flickr, YouTube, Facebook e Wordpress são exemplos de sistemas voltados exclusivamente para a promoção de conteúdo dos usuários em que não há uma equipe centralizada responsável por sua editoração.

A Figura 1 exibe dados referentes ao volume de produção e acessos ao conteúdo em alguns sites de conteúdo colaborativo dentro de um intervalo de um minuto. O YouTube, responsável pelo armazenamento e reprodução de vídeos *online*, recebe *uploads* que somados representam 30 horas contínuas de vídeo, enquanto 1,3 milhão de acessos são realizados ao acervo de vídeos já publicados.

O Flickr – serviço de galerias de fotos *online* – recebe, em média, três mil novas fotos e 20 milhões de visualizações a fotos publicadas.

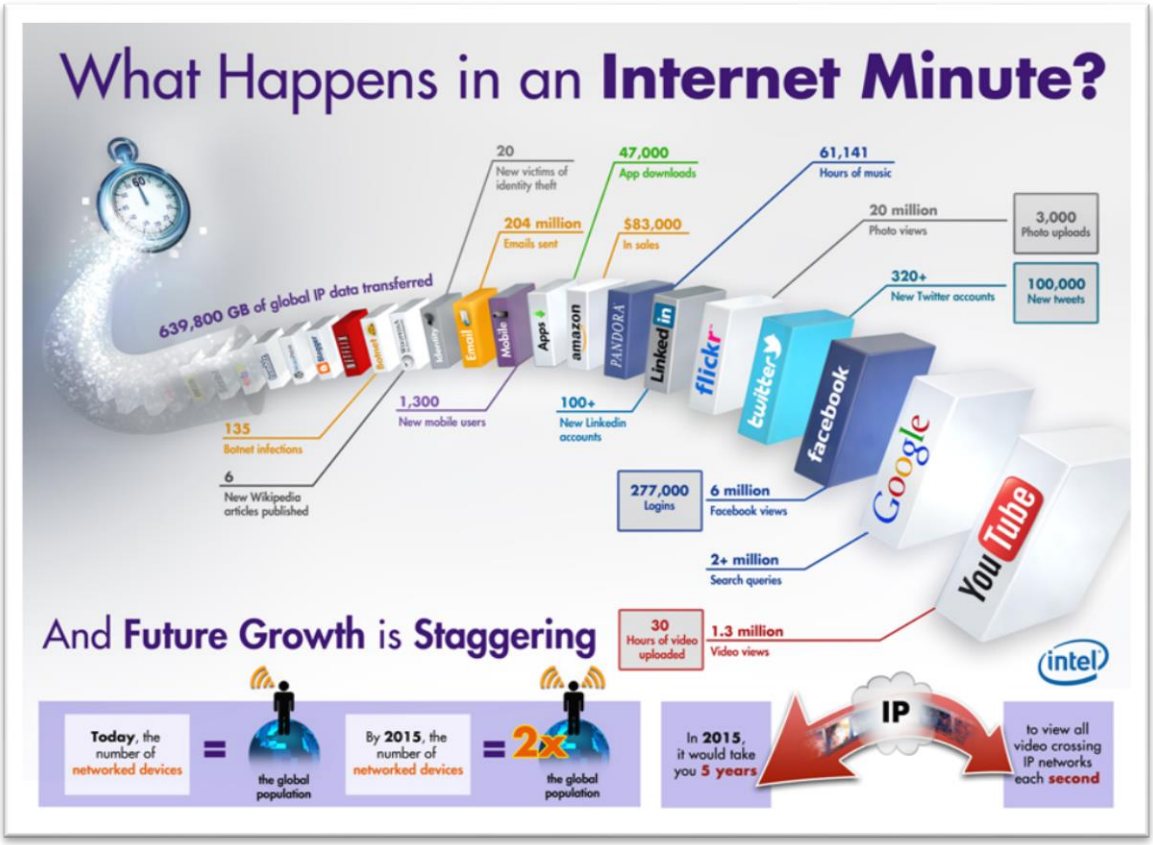

Figura 1 - O que acontece na Internet em um minuto?
Fonte: INTEL (2012)

O sucesso dos sistemas gestores do conteúdo colaborativo implicou no aumento da demanda de recursos computacionais nos servidores que hospedam estas aplicações. As práticas existentes de otimização e *caching* – voltadas para um modelo em que o conteúdo era atualizado por uma quantidade reduzida de pessoas e em maiores intervalos de tempo – necessitaram ser revistas.

Exigindo um constante e ilimitado crescimento do fornecimento de recursos computacionais, os sistemas gestores de conteúdo colaborativo tornaram-se um cenário compatível com a proposta do modelo de computação distribuída, a qual será analisada no item 3.1 deste trabalho sob a perspectiva de sua viabilidade quando aplicada a *websites* cuja principal demanda computacional é decorrente do consumo de arquivos de mídia, como imagens e vídeos.

3 REVISÃO BIBLIOGRÁFICA

Esta seção relaciona os conceitos pertinentes ao cenário de aplicação das redes de distribuição de conteúdo.

3.1 A VIABILIDADE INICIAL DA COMPUTAÇÃO DISTRIBUÍDA

O conceito de computação distribuída pode ser definido como uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e coeso (TANEMBAUM, 2002) e, por meio da adição de novos nós computacionais, torna possível realizar a distribuição de carga utilizando hardwares ou softwares específicos.

Todavia, a despeito dos benefícios oferecidos pelo modelo distribuído, em um cenário real sua utilização pode ser demasiadamente custosa quando a necessidade da aplicação é apenas a otimização do tempo de resposta prejudicado por algum gargalo.

Segundo Do (2007), foi este o cenário observado pelo YouTube em seus primeiros anos de operação. Fundado em fevereiro de 2005, o site de vídeos foi mantido exclusivamente por seus criadores até outubro de 2006, quando foi adquirido pela empresa Google.

O crescimento inicial do site foi acompanhado por uma necessidade constante de ampliação da capacidade computacional de seus servidores, o que se tornou complexo em um cenário altamente dependente de uma infraestrutura baseada em servidores dedicados alugados em *datacenters* compartilhados. Em uma infraestrutura compartilhada não se encontram condições favoráveis à escalabilidade requerida por um modelo distribuído, visto que o locatário não possui a autonomia necessária para controlar o *hardware* e implementar toda a especificação de rede necessária para atender aos seus requisitos (DO, 2007).

No entanto, será inevitável para a maioria dos novos sistemas gestores de conteúdo colaborativo um crescimento inicial semelhante ao experimentado pelo YouTube. Com o alargamento da dependência em relação à capacidade da computação disponível a um aplicativo, os datacenters são cada vez mais vistos como impulsionadores de lucro, tornando-se o controle de custos uma das tarefas

mais delicadas a serem realizadas pelos gerentes de tecnologia da informação (PATRIZIO, 2011).

Diante deste panorama, uma infraestrutura compartilhada é capaz de oferecer um poder computacional suficiente para que o *website* inicie suas atividades sem nenhum investimento inicial na aquisição de *hardware* e, ainda, permite obter um custo de manutenção inferior a qualquer outra modalidade de hospedagem (JENSEN, 2008).

Considerando que muitos dos gargalos em aplicações baseadas em imagens ou vídeo estão situados nas unidades de armazenamento e no tráfego da rede (DO, 2007), a utilização de uma rede de distribuição de conteúdo para arquivos de mídia pode representar uma solução que vai ao encontro dos requisitos de um sistema gestor de conteúdo colaborativo em fase inicial, respeitando às restrições estabelecidas por seu ambiente compartilhado e postergando a necessidade de maiores investimentos voltados à migração da infraestrutura de hospedagem para um ambiente mais custoso e complexo.

3.2 CACHE VERSUS VOLUME DE DADOS

A largura de banda, o tempo para propagação, a velocidade e o nível de carga dos computadores, tanto do cliente como dos servidores, são fatores determinantes para a definição do tempo médio de resposta de uma aplicação web (WATANABE, 2000).

Em páginas dotadas de elevado número de elementos estáticos, tais como imagens, arquivos de folhas de estilo em cascata e arquivos com código *Javascript*, a utilização de um *proxy* reverso entre o servidor web e seus clientes pode representar um fator de melhoria significativa na minimização do tempo de recuperação destes arquivos, fazendo com que sua velocidade de carregamento seja aumentada entre trezentos e mil vezes (VARNISH, 2012).

Configurado para operar na porta que recebe as requisições dos clientes – usualmente a porta número 80 – o *proxy* reverso intermedeia todas as solicitações que antes seriam encaminhadas diretamente ao servidor web.

Conforme esquematizado na Figura 2, o servidor de *cache* executando a aplicação Varnish possui ao seu dispor um determinado espaço para armazenamento, mantendo uma estrutura de arquivos otimizada para recuperar rapidamente os arquivos mais frequentemente requisitados. Assim sendo, para toda requisição recebida (1) é verificado se primeiramente o arquivo solicitado está disponível em *cache* (2). Em caso afirmativo, configura-se o denominado "*cache hit*", em que o próprio servidor de *cache* responderá à requisição (3), que será entregue ao usuário (4) sem sequer ter sido processada pelo servidor de aplicação.

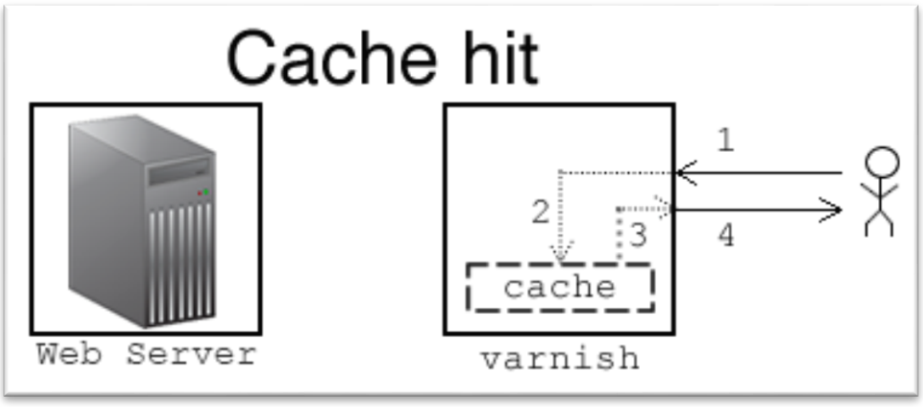

Figura 2 - Cache Hit: Requisição é respondida diretamente pelo cache, sem necessidade de processamento pelo servidor web.
Fonte: DRUPAL (2010).

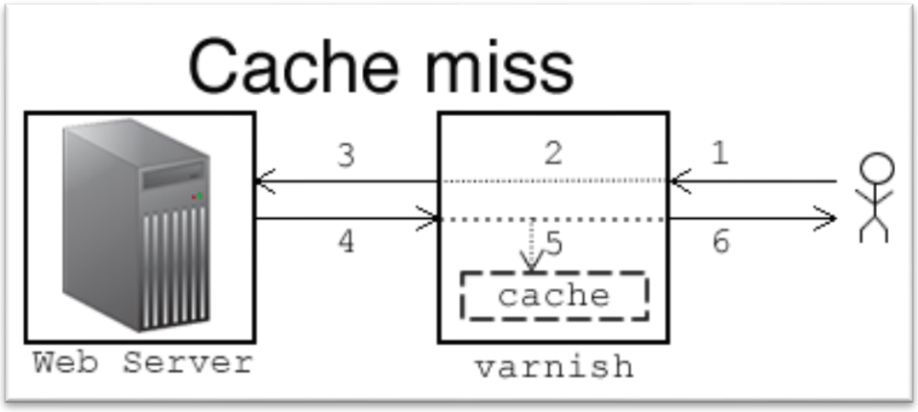

Figura 3 - Cache miss: não houve sucesso na leitura do arquivo a partir do cache e a requisição passa a ser tratada pelo servidor web.
Fonte: DRUPAL (2010).

Por sua vez, caso o arquivo solicitado não esteja disponível em *cache*, configura-se o "*cache miss*", conforme demonstrado na Figura 3, em que o servidor de *cache* permite que a requisição chegue até o servidor web (2 e 3), onde será

processada e cuja resposta passará novamente pelo servidor de *cache* (4), o qual poderá optar por armazená-la em sua estrutura de arquivos (5) a fim de servir mais rapidamente às próximas requisições ao mesmo arquivo. Finalmente a resposta é entregue ao usuário (6).

A relação estatística entre os acertos (*cache hit*) e o número total de requisições recebidas é denominada *hit ratio* (WATANABE, 2000). Em um cenário ideal, em que o *cache* seria capaz de armazenar a totalidade do conjunto de dados disponibilizado pelo servidor web, haveria um *hit ratio* de 100%.

Porém, armazenar um volume muito grande de arquivos em *cache* pode não ser uma implementação eficaz por:

- Comprometer o desempenho do próprio serviço de *cache*, sobrecarregando-o com estruturas de dados com número excessivo de nós, o que aumentará o consumo de processamento para a busca e recuperação do arquivo solicitado (VARNISH, 2012).
- Elevação dos custos com hardware, tendo em vista que arquivos mantidos em *cache* tenderão a estarem armazenados em unidades de armazenamento de maior velocidade, como memória RAM ou discos *Solid-State Drive* – SSD.

Assim sendo, para aplicações baseadas em arquivos de mídia, tais como imagens ou vídeos, a efetividade de um serviço de *cache* pode ser comprometida pela disparidade entre o amplo volume de dados mantido pela aplicação e o reduzido espaço disponível para *cache*. A Figura 4 demonstra por meio de uma curva em um plano cartesiano o comportamento da taxa de acerto do *cache* (*cache hit ratio*) em função do porcentual total que o espaço disponível para *cache* representa quando comparado ao total de dados armazenados pela aplicação. Enquanto nota-se grande efetividade do *cache* enquanto o conjunto de dados é menor e mais próximo do tamanho disponibilizado para *cache*, conforme se dá a redução da diferença entre o total armazenado e o tamanho do *cache*, há a tendência de estabilização da curva e a taxa de acerto passa a se estagnar.

A redução da efetividade do *cache* quando aplicado em sistemas com grande volume de dados demonstra que, mesmo sendo um recurso vital em aplicações com grande volume de acessos, o *cache* por si só não é capaz de

proporcionar o resultado esperado em termos de desempenho e redução do tempo de resposta.

Todavia, ainda assim, os mecanismos de *cache* são uma ferramenta importante nestas aplicações quando são levados em consideração os conceitos de *hot data* e *cold data*, em que, respectivamente, apenas uma fração do conteúdo total é acessada com maior frequência em um determinado momento, enquanto a maior parte é acessada apenas casualmente ou com uma frequência muito menor (CAULKINS, 2012).

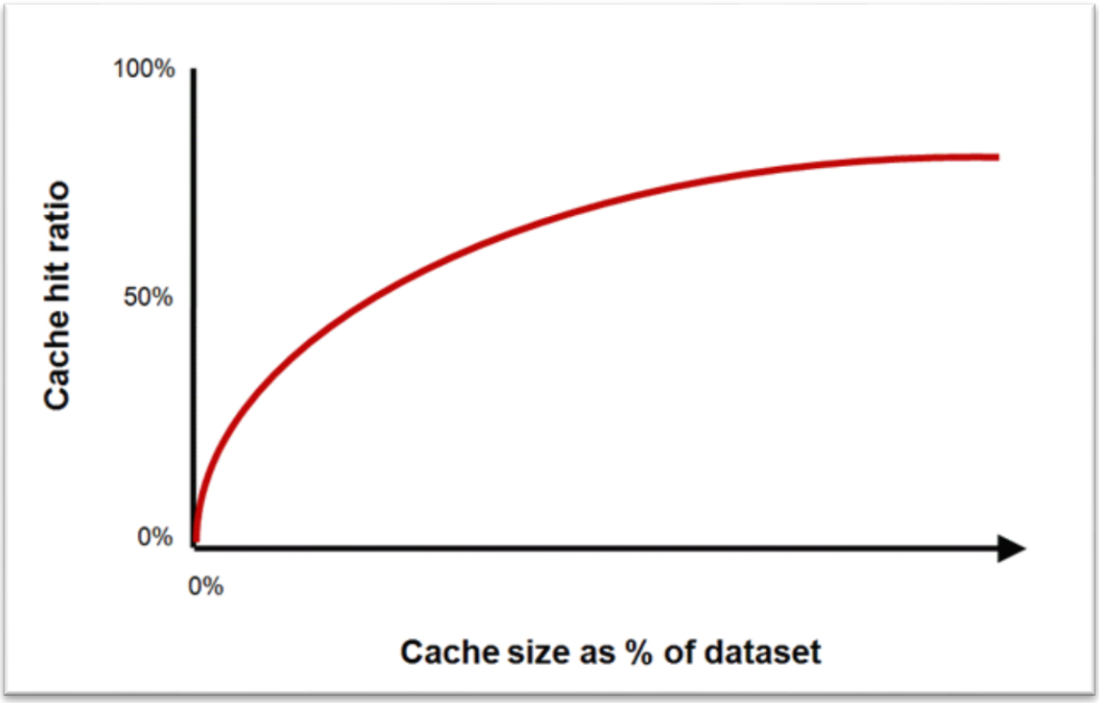

Figura 4 - A redução da efetividade do cache ante a ampliação do conjunto de dados hospedados pela aplicação web.
Fonte: CAULKINS (2012)

Deste modo, se evidencia a importância do correto dimensionamento e otimização de um mecanismo de *cache* capaz de estar focado nos dados mais frequentemente acessados em um determinado momento, não permitindo que dados menos acessados interfiram e prejudiquem seu desempenho, ainda que sua capacidade de armazenamento represente uma pequena fração do total de dados da aplicação (CAULKINS, 2012).

O *Google Global Cache* é um exemplo de mecanismo de *cache* capaz de conciliar *caching* e redes de distribuição de conteúdo a fim de prover arquivos de

mídia de forma geograficamente otimizada. Seu funcionamento será analisado no item 3.3.1 deste trabalho.

3.3 ESTRATÉGIAS DE DISTRIBUIÇÃO DO CONTEÚDO

Apesar da velocidade média de conexão à Internet manter um ritmo de expansão acelerado, a proximidade entre o usuário e o conteúdo ainda representa um impacto significativo no tempo de carregamento das páginas, principalmente quando é levado em consideração que cerca de 80% a 90% deste tempo é gasto realizando o download de todos os componentes estáticos que fazem parte dela (SOUDERS, 2007).

Ainda para Souders (2007, p. 18), a fim de distribuir conteúdo disperso geograficamente, seria precipitado um redesenho da aplicação para que esta passasse a funcionar dentro de uma arquitetura distribuída, uma vez que o conteúdo dinâmico, aquele demarcado por meio do HTML, é transferido por meio de apenas uma requisição. Por sua vez, uma página HTML poderá desencadear dezenas de outras requisições para *download* de seus componentes estáticos, como imagens, arquivos *Flash*, folhas de estilo CSS, entre outros.

Desta forma, enquanto a utilização de uma arquitetura distribuída traria ao usuário a redução do tempo de resposta de apenas uma requisição HTTP, em contrapartida, a utilização de uma rede de distribuição de conteúdo seria capaz de otimizar o tempo de carregamento de todos os outros arquivos estáticos que fazem parte desta página, proporcionando uma redução significativa no tempo de carregamento total e, ainda assim, dispensando qualquer necessidade de redesenho da aplicação para que inclua sistemas avançados de replicação de banco de dados ou sincronização de sessões, o que seria indispensável caso adotasse uma arquitetura distribuída.

As redes de distribuição de conteúdo podem ser definidas como uma coleção de servidores web distribuídos em múltiplas localidades geográficas com o intuito de oferecer conteúdo ao usuário de uma maneira mais eficiente (SOUDERS, 2007).

A eficiência pode ser definida em termos de:

- *Performance*, resultando na minimização do tempo de carregamento final ao usuário, por meio da redução dos efeitos da latência que pode ser alcançada pela seleção de um servidor em rede mais próxima à rede do usuário;
- Distribuição de carga, buscando utilização mais eficiente dos recursos oferecidos, balanceando o tráfego entre múltiplos nós de fornecimento de conteúdo com base no tempo de resposta de cada servidor em dado momento;
- Racionalização do uso de recursos, fazendo com que requisições a arquivos de mídia que requerem um maior tráfego de dados sejam direcionadas a servidores localizados em áreas com maior largura de banda ou menor custo de transferência.

A Tabela 1 demonstra o impacto da distância entre o usuário e o servidor sobre o tempo de download de um arquivo de 4 gigabytes. A distância é um fator diretamente relacionado à variação de *Round-Trip Time* – RTT, tempo que pode ser verificado por meio de um comando *ping*, além de impactar significativamente sobre o número de pacotes perdidos e reduzir drasticamente a taxa de transferência. A implicação é a tendência de obter um maior tempo necessário para download do arquivo conforme se amplia a distância entre o usuário e o servidor que o hospeda.

Distância entre o usuário e o servidor	RTT	Perda de Pacotes	Taxa de Transferência	Tempo de Download para 4GB
Local Até 160km	1,6 ms	0,6%	44 Mbps	12 minutos
Regional De 804 a 1609 km	16 ms	0,7%	4 Mbps	2,2 horas
Cross-continent Cerca de 4828 km	48 ms	1%	1 Mbps	8,2 horas
Multi-continent Cerca de 9656 km	96 ms	1,4%	0,4 Mbps	20 horas

Tabela 1 - Efeito da distância sobre a taxa de transferência e tempo de download

Fonte: NYGREN, SITARAMAN, SUN (2010).

Na sequência, nos tópicos 3.3.1 e 3.3.2 serão analisados os modelos de distribuição do conteúdo adotados, respectivamente, pelas redes *Google Global Cache* e *Akamai*.

3.3.1 GOOGLE GLOBAL CACHE

O *Google Global Cache* - GGC possibilita que provedores de serviços de Internet otimizem sua infraestrutura de rede e reduzam os custos de transferência de dados de longa distância por meio do armazenamento local de conteúdo estático dos sites do Google dentro de sua própria rede (GOOGLE GLOBAL CACHE, 2012).

A implementação do GGC consiste na instalação de um conjunto de servidores no datacenter da companhia operadora da rede. O número e a capacidade dos servidores são dimensionados de acordo com a demanda de usuários e o número de localidades atendidas. Os servidores são remotamente gerenciados pelo Google (GOOGLE GLOBAL CACHE, 2012).

Preferencialmente, a instalação do GGC é realizada em operadoras próximas a Pontos de Troca de Tráfego – PTT, o que beneficia não apenas os usuários da operadora que hospeda os servidores de *cache* da empresa, mas também todos os usuários das demais operadoras que utilizam o mesmo ponto.

Almada (2009) aponta como as principais vantagens da utilização do GGC:

- Redução do tráfego internacional, que se traduz em uma maior rentabilização da capacidade das empresas provedoras de serviços de Internet, levando em consideração o alto custo da transferência de dados por meio de banda internacional.
- Resposta mais rápida e transparente aos usuários finais por meio do *cache*;
- Robustez, uma vez que o nó do GGC possui vários níveis de redundância e, em caso de falhas, as requisições serão transparentemente encaminhadas ao servidor principal.

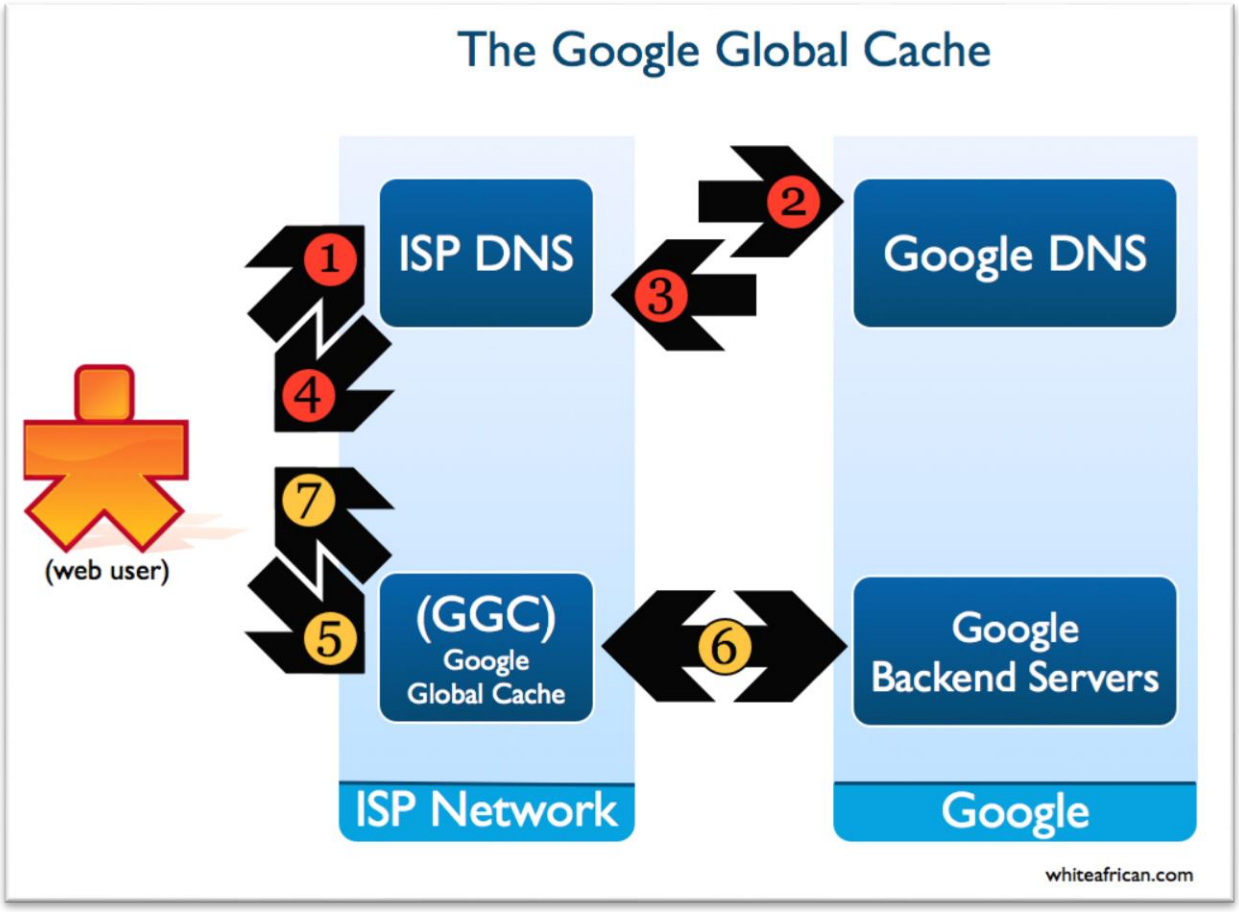

Figura 5 - Funcionamento do Google Global Cache
Fonte: HERSMAN (2008).

Na Figura 5 é possível visualizar a estratégia de funcionamento implementada pelo *Google Global Cache*. Enquanto as setas com números entre 1 e 4 representam um fluxo normal de requisições, as setas com números entre 5 e 7 demonstram a sequência de requisições com a existência de servidores de *cache* inseridos na rede do provedor de serviços de Internet. O fluxo bidirecional no número 6 será executado apenas quando a requisição não puder ser resolvida pelo servidor de *cache*, o que representa um *cache miss*, situação descrita no item 3.2 deste trabalho.

3.3.2 AKAMAI

A empresa Akamai é responsável pela administração de redes de distribuição de conteúdo que respondem de 15% a 30% de todo o tráfego da Internet, o que representa um volume de dados aproximado de 8 terabytes por segundo (AKAMAI, 2012).

Ao contrário do *Google Global Cache*, descrito no item 3.3.1, os servidores da Akamai não fazem parte de uma rede destinada a atender apenas aos produtos de uma empresa. A empresa disponibiliza serviços de distribuição de conteúdo dispersos em várias regiões do planeta, em parceria com um grande número de provedores de serviços de Internet.

Entre os serviços que fazem uso das redes de distribuição de conteúdo Akamai estão a rede social Facebook, os múltiplos serviços oferecidos pelo site Yahoo! e a loja online iTunes, da Apple, responsável pela venda de mídias digitais como aplicativos, músicas e vídeos.

Em todas as situações, a entrega de arquivos estáticos é beneficiada pela rede Akamai, que possibilita por meio de sua estratégia de *caching* que as transferências ocorram, sempre que possível, a partir da rede com maior proximidade do cliente.

3.4 MÉTODOS DE AQUISIÇÃO DO CONTEÚDO

As redes de distribuição de conteúdo não atuam isoladas, uma vez que dependem de um servidor de aplicação, o qual é responsável pelo armazenamento inicial dos arquivos que serão replicados por elas.

Deste modo, será necessário garantir que o conteúdo disponibilizado pelo servidor da aplicação possa ser transferido para os servidores da rede de distribuição de conteúdo. Com o intuito de realizar esta transferência, existem duas possíveis abordagens com diferentes características de facilidade de uso, flexibilidade e controle.

A escolha de um modelo de aquisição do conteúdo está diretamente ligada à natureza da aplicação, sendo possível até mesmo combinar ambos modelos a fim de otimizar o desempenho da distribuição de arquivos e reduzir o número de requisições sobre o servidor de aplicação, bem como o tempo de resposta para o usuário. Os modelos *Origin-Pull* – baseado em carga preterida – e *Push* – baseado em carga antecipada – serão detalhados nos itens 3.4.1 e 3.4.2, respectivamente.

3.4.1 ORIGIN-PULL

O modelo *Origin-Pull* é dotado de grande facilidade de configuração, uma vez que permite que o proprietário do *website* apenas reescreva as URLs de arquivos estáticos a fim de que estas passem a apontar para a rede de distribuição de conteúdo ao invés de um arquivo local no servidor (BAILEY, 2010).

Assim sendo, conforme mostra a Figura 6, na primeira vez que houver uma requisição a determinado arquivo (1), a CDN irá receber a requisição do usuário e fará uma nova solicitação ao servidor de aplicação a fim de obter o arquivo que foi pedido (2 e 3). A partir de então, a CDN passará a manter uma cópia deste arquivo, o que evitará que novas solicitações iguais cheguem ao servidor de aplicação, ficando a cargo da rede de distribuição de conteúdo fornecer tal arquivo, conforme demonstra a Figura 7 (BAILEY, 2010).

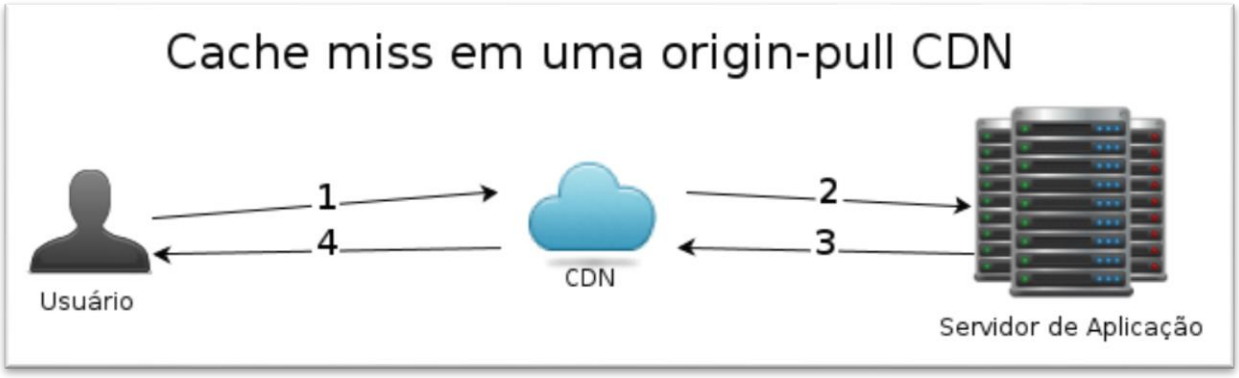

Figura 6 - Cache miss em uma origin-pull CDN

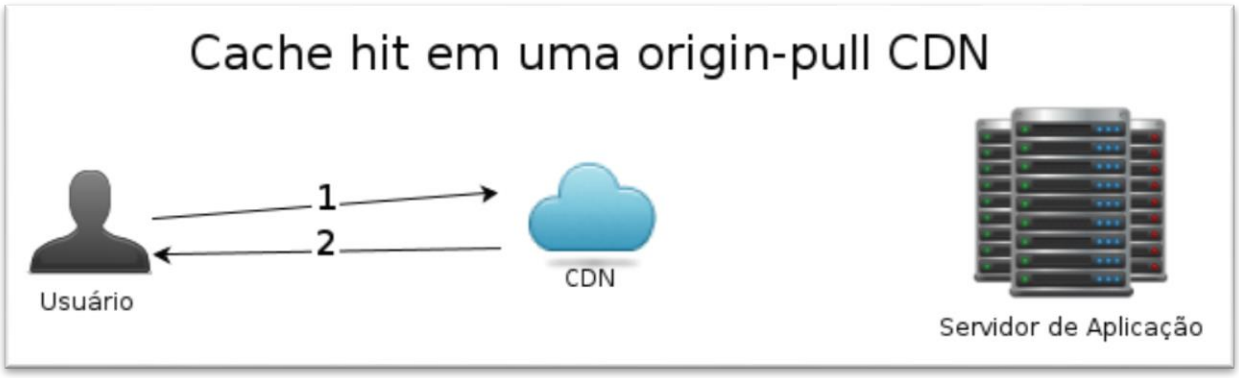

Figura 7 - Cache hit em uma origin-pull CDN

Neste modelo, o servidor de aplicação não possui qualquer referência para a rede de distribuição de conteúdo, o que implica em maior facilidade de

configuração, mas, em contrapartida, oferece menor capacidade de flexibilização da rede para atender a necessidades específicas e impossibilita um controle de versão sobre mudanças que podem acontecer em um arquivo, podendo levar a um estado inconsistente de sincronização em que a CDN manterá uma cópia desatualizada de um arquivo que foi modificado pelo servidor de aplicação (BAILEY, 2010).

O tempo de resposta à primeira requisição também é um fator negativo para redes que funcionam no modelo *origin-pull*. Enquanto as requisições seguintes serão atendidas pela cópia mantida em *cache*, a primeira requisição deverá buscar o arquivo diretamente no servidor de aplicação, o que inevitavelmente aumentará o tempo de espera até que o arquivo seja enviado ao usuário.

3.4.2 PUSH

No modelo *Push*, o servidor de aplicação possui total controle sobre a rede de distribuição de conteúdo, sendo este o único responsável por enviar, atualizar e remover arquivos da CDN (BAILEY, 2010). Algumas redes que funcionam neste modelo suportam o gerenciamento de arquivos por meio de serviço FTP – *File Transfer Protocol* ou por meio de protocolos como REST – *Representional State Transfer* ou SOAP – *Simple Object Access Protocol*.

Conforme mostra a Figura 8, o usuário apenas passa a enviar requisições (3) por um arquivo à rede de distribuição a partir do momento em que este arquivo já foi transferido até ela por meio de uma ordem do servidor de aplicação (1) e este já recebeu a confirmação do armazenamento na CDN (2).

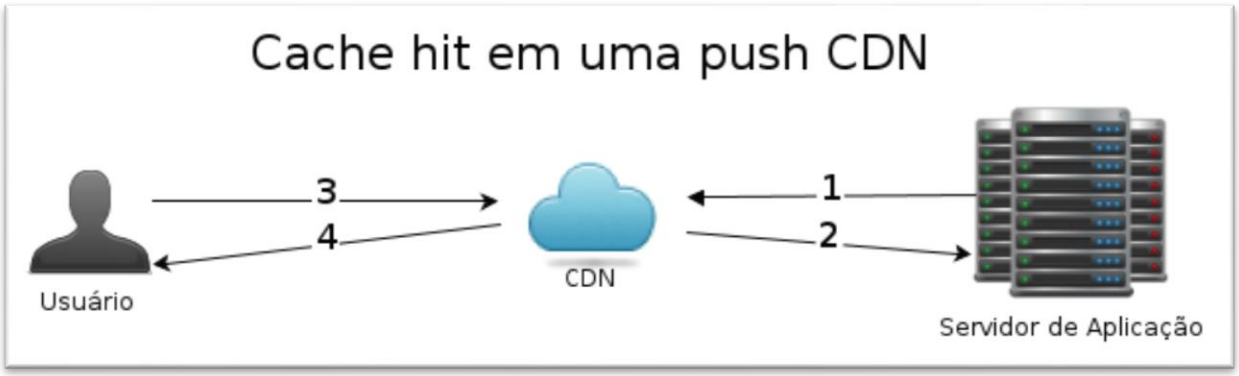

Figura 8 - Cache hit em uma push CDN

O maior controle do servidor sobre a CDN garante a maximização da flexibilidade em relação à forma com que a CDN ajudará a servir o conteúdo, sendo possível garantir a consistência de versionamento de um mesmo arquivo entre o servidor de aplicação e a rede de distribuição do conteúdo.

Além disso, por meio de uma rede em modelo *Push*, será possível balancear o tráfego em uma estratégia definida pela aplicação, o que é muito útil em sites cujo maior volume de tráfego é direcionado a arquivos presentes em atualizações recentes. Assim, pode-se garantir que a rede trabalhará exclusivamente com os arquivos mais importantes naquele momento, fazendo um uso mais eficiente do *cache* disponibilizado pela CDN.

A principal desvantagem de uma CDN em operação no modelo *Push* é a necessidade da aplicação gerenciar as URLs de todos os arquivos estáticos, afinal, é de total responsabilidade do servidor de aplicação estabelecer o local onde cada arquivo está armazenado.

4 PROJETO

O presente trabalho propõe a implementação de um agente de replicação para servidores de mídia, permitindo que aplicações que sofrem com os problemas apresentados no item 2 se beneficiem da distribuição do conteúdo entre dois ou mais servidores, minimizando a possibilidade de sobrecarga no servidor de aplicação devido às requisições para acesso aos arquivos estáticos.

Na sequência, a Figura 9 exibe o modelo de carregamento baseado em um único servidor. Após o envio do hipertexto, novas requisições são feitas ao mesmo servidor que enviou o conteúdo HTML.

Na Figura 10, um modelo de conteúdo distribuído. O servidor mestre faz uso dos métodos de roteamento do agente de replicação e gera URLs para o conteúdo presente em outros servidores HTTP destinados exclusivamente ao provimento de mídia. Assim, o único contato entre o cliente e o servidor HTTP mestre se dá no fornecimento do conteúdo de hipertexto. O documento de hipertexto gerado contém todas as URLs já definidas para que o download do conteúdo externo a este arquivo seja feito a partir dos servidores de mídia, minimizando o número de requisições submetidas ao servidor mestre.

Deste modo, o servidor mestre – o qual executa a regra de negócio da aplicação, podendo concentrar até um sistema gerenciador de banco de dados – é poupado da sobrecarga ocasionada pelas conexões recebidas, leitura em disco e envio da resposta para cada requisição de arquivo estático solicitado pelos clientes.

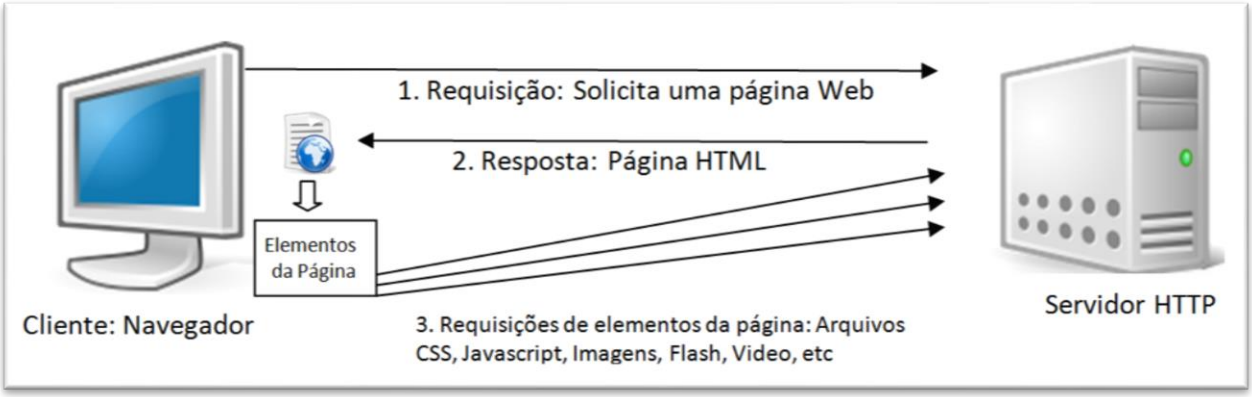

Figura 9 - Requisições no carregamento de uma página e de seus elementos em um modelo de servidor único.

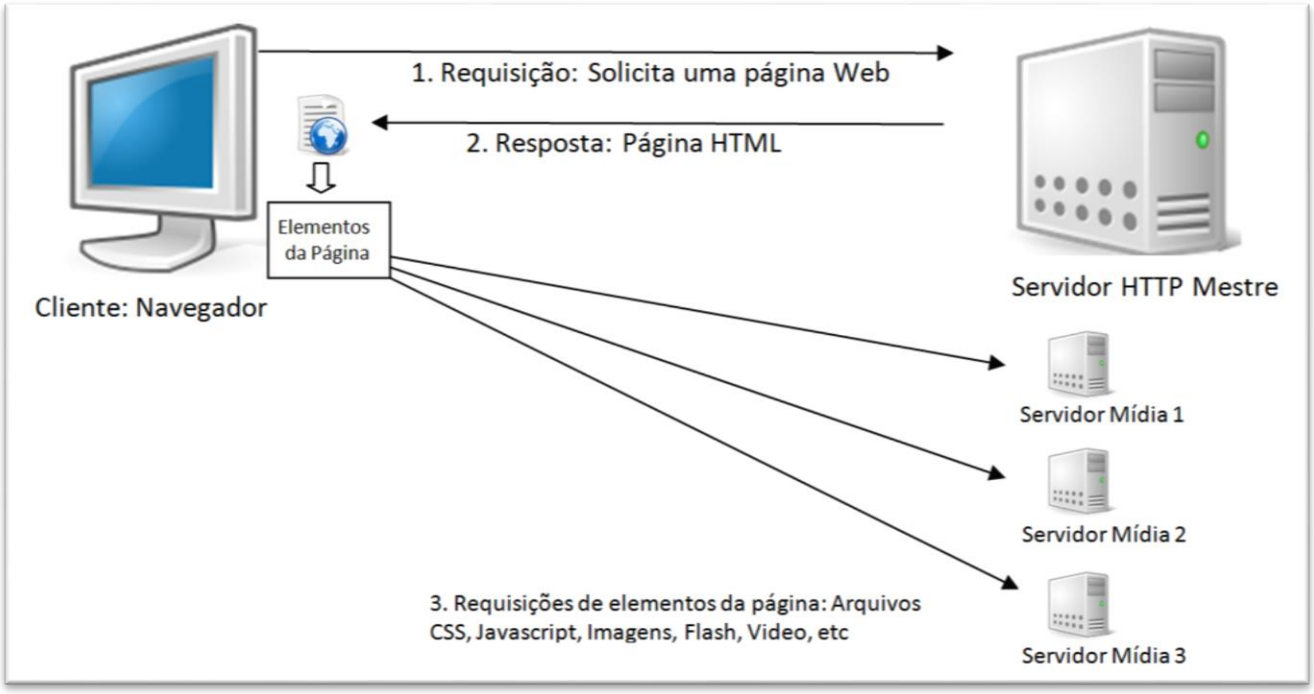

Figura 10 - Requisições e carregamento de uma página e de seus elementos com a implementação do conteúdo distribuído entre servidores de mídia.

4.1 OBJETIVOS GERAIS

O quadro 1 apresenta os objetivos gerais do trabalho.

ID	Objetivo	Descrição
1	Replicar o conteúdo de um servidor web mestre para um ou mais servidores web escravos.	Permitir que seja feita a replicação parcial do conteúdo entre um servidor de aplicação mestre e os demais servidores escravos.
2	Oferecer mecanismos para verificação de integridade dos arquivos transferidos.	Oferecer meios para verificar se os arquivos foram transferidos com sucesso, sem perdas.
3	Disponibilizar métodos de auditoria a partir do servidor mestre.	Permitir que o servidor mestre realize auditorias do conteúdo replicado entre os servidores escravos, a fim de verificar a integridade dos arquivos em caso de uma eventual falha.
4	Oferecer um mecanismo de roteamento para geração de URLs	Com a linguagem PHP, oferecer à aplicação objetos que permitam definir URLs que apontem para os arquivos em seus respectivos destinos.
5	Permitir que o servidor mestre obtenha informações sobre a disponibilidade atual dos servidores escravos.	Garantir que o servidor mestre tenha informações sobre a disponibilidade de cada um dos servidores escravos a fim de direcionar o tráfego para servidores redundantes em caso de falha em um dos nós.

Quadro 1 - Objetivos do Sistema

4.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos podem ser traduzidos em funcionalidades que o serviço oferecerá à aplicação.

O Quadro 2 lista os objetivos específicos para o agente de replicação que será desenvolvido. Para cada objetivo, um nível de relevância será atribuído, em que os indispensáveis serão classificados como essenciais, os de importância média serão classificados como importantes e os complementares receberão a categoria “desejável”.

Os objetivos específicos estão relacionados aos objetivos gerais relacionados no Quadro 1 e serão enumerados como seus subitens.

Necessidades	Categoria
1.1 Permitir que o servidor de aplicação solicite o armazenamento de um ou mais arquivos em uma rede de distribuição de conteúdo.	Essencial
1.2 Permitir que o servidor de aplicação solicite a exclusão de um ou mais arquivos em uma rede de distribuição de conteúdo.	Essencial
1.3 Executar a cópia de arquivos a partir de uma URL indicada pelo servidor de aplicação no momento da sincronização.	Essencial
1.4 Possibilitar que, por meio do agente de replicação, a aplicação possa transferir arquivos para um número ilimitado de redes diferentes.	Desejável
2.1 Realizar testes de integridade nos arquivos transferidos por meio da comparação entre *chechsums* calculados na origem e no destino.	Essencial
2.2 Permitir que a aplicação seja notificada em caso de falhas em alguma operação.	Essencial
3.1 Garantir que todos os arquivos armazenados na rede de distribuição de conteúdo possam ser indexáveis pelo agente de replicação.	Importante
3.2 Coletar informações sobre arquivos armazenados na rede de distribuição a fim de fornecê-los em relatórios para o servidor de aplicação quando solicitado.	Importante
4.1 Garantir que todo arquivo armazenado esteja acessível por meio de um identificador único.	Essencial
4.2 Notificar o servidor de aplicação sobre o status de todas as operações realizadas a fim de garantir que as URLs de acesso a todos os arquivos permaneçam sempre atualizadas.	Essencial
5.1 Implementar um mecanismo de detecção de indisponibilidade da rede a fim de permitir que a aplicação desvie seu tráfego para outra rede em operação redundante.	Importante

Quadro 2 - Objetivos específicos

4.3 LIMITES DA SOLUÇÃO

Os limites representam possíveis funcionalidades que poderiam ser implementadas neste sistema, porém, devido à necessidade de adequação ao tempo e aos recursos disponíveis para o desenvolvimento, não foram incluídas na presente versão.

O Quadro 3 lista os limites estabelecidos ao escopo deste projeto.

ID	Limitação	Justificativa
01	O agente de replicação não suportará transferências assíncronas.	Pelo fato da biblioteca CURL não disponibilizar uma interface assíncrona, não haverá nesta versão suporte a transferências assíncronas.
02	Não haverá nenhum tipo de tratamento de erros de transferência.	Por se tratar de um agente destinado a redes que operam em um modelo PUSH, detalhado no item 3.4.2 deste trabalho, o controle sobre a manipulação de erros será totalmente delegado à aplicação, cabendo ao agente de replicação apenas notificar quanto à sua ocorrência.
03	O agente não será responsável pelo gerenciamento do uso de recursos nos servidores de mídia.	Por se tratar de um agente destinado a redes que operam em um modelo PUSH, detalhado no item 3.4.2 deste trabalho, todo gerenciamento sobre a utilização de recursos, como o espaço em disco, nos servidores de mídia será de responsabilidade da aplicação.
04	O agente não realizará desvios de tráfego na ocorrência de indisponibilidade de um dos servidores de mídia.	O agente apenas notificará a indisponibilidade de um servidor de mídia à aplicação quando for solicitada uma verificação de status da rede. A partir da notificação, caberá à aplicação desviar o tráfego para outros servidores de acordo com seu plano de redundância de arquivos.
05	O agente não realizará um controle transacional das operações.	Quando o servidor de aplicação enviar uma lista de arquivos para processamento de inclusão ou exclusão, poderão ocorrer falhas em um ou mais arquivos da lista sem que as transferências pendentes ou bem sucedidas sejam abortadas ou revertidas. Caberá ao agente apenas notificar, por meio de log de operação, o status de cada uma das operações realizadas.

Quadro 3 - Limites da solução

4.4 RESTRIÇÕES

Restrições ou requisitos não funcionais delimitam a forma com que o sistema realizará seus requisitos funcionais (WASLAWICK, 2004).

O Quadro 4 exibe as restrições impostas sobre o projeto, implementação e funcionalidades do sistema.

ID	Restrição	Natureza
1	Os servidores mestre e escravos deverão estar interligados por meio de uma rede TCP/IP.	Ambiente
2	Os servidores mestre e escravos deverão possuir sistemas operacionais Linux.	Ambiente
3	Serão realizados testes utilizando o sistema operacional Linux CentOS versão 6.0 x86-64.	Ambiente
4	Os servidores deverão possuir instalado o PHP versão 5.3.17 ou superior, com a biblioteca CURL habilitada.	Ambiente
5	Serão realizados testes de acesso utilizando os servidores web Apache 2.2 e NGinx 1.0.	Ambiente
6	Não será implementado mecanismo de monitoramento para desvio de requisições em caso de queda de um dos servidores escravos.	Funcionalidade
9	Os checksums serão realizados com algoritmo MD5 implementado pela função md5_file() da linguagem PHP.	Funcionalidade
10	O sistema não implementará mecanismo de balanceamento de carga baseado no consumo de recursos do servidor. Todo balanceamento será definido pelas regras estabelecidas pelo administrador do sistema para a distribuição dos arquivos.	Funcionalidade
11	O sistema não apresentará interface gráfica, tendo sua configuração realizada por meio de arquivos de texto simples.	Funcionalidade
12	As funções de sincronização dependerão do agendamento de tarefas no sistema operacional (*crontabs*).	Compatibilidade
13	As trocas de mensagens entre os servidores e as próprias configurações do sistema serão realizadas por meio de arquivos em formato JSON – Javascript Object Notation.	Compatibilidade
14	Todos os arquivos do sistema estarão com codificação UTF-8.	Compatibilidade
15	O correto processamento dos arquivos JSON dependerá da implementação das funções json_encode e json_decode, nativas do PHP 5.3.	Compatibilidade

Quadro 4 - Restrições da solução

4.5 TECNOLOGIAS UTILIZADAS

Este tópico listará as tecnologias empregadas no desenvolvimento deste trabalho, buscando identificar elementos que justifiquem sua utilização.

4.5.1 PHP

PHP – PHP: *Hypertext Preprocessor* é uma linguagem de programação amplamente utilizada em *scripts* voltados às mais diversas finalidades, especialmente no desenvolvimento de aplicações voltadas para a Internet (PHP, 2012).

Conforme demonstra a Figura 11, a linguagem possui ampla aceitação no mercado de linguagens de programação para web. A alta popularidade deve-se também à grande oferta de sistemas CMS – *Content Management Systems* criados com PHP, tais como WordPress, Drupal, Joomla, Magento, Moodle, entre outros.

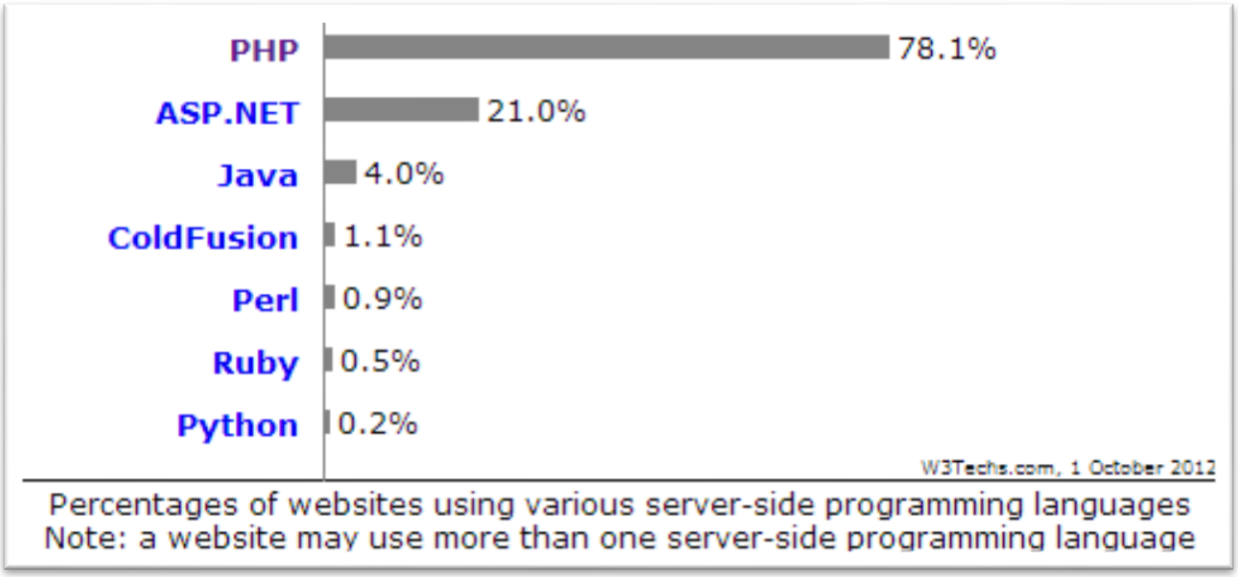

Figura 11 - Estatísticas sobre a utilização de linguagens de programação *server-side* em *websites*.
Fonte: W3Techs.com (2012)

4.5.2 JSON

JSON – *JavaScript Object Notation* é um formato para armazenamento e troca de informação textual. É menor, mais rápido e pode ser analisado mais facilmente que o XML – *eXtensible Markup Language* (W3SCHOOLS.COM, 2012).

Apesar de utilizar a sintaxe da linguagem *Javascript* para descrever objetos, o JSON é totalmente independente de linguagens e plataformas, podendo ser manipulado por meio dos *parsers* e bibliotecas disponíveis para as mais diversas linguagens de programação.

Desde a versão 5.2.0, a linguagem PHP incorporou a extensão JSON à sua lista de extensões empacotadas e compiladas por padrão (PHP, 2012), passando, assim, a fazer parte do núcleo da linguagem.

Desde então, as funções *json_encode()* e *json_decode()* trouxeram uma grande facilidade na leitura e escrita de arquivos neste formato, sendo capazes de, respectivamente, transformar um *array* ou objeto para uma representação JSON ou carregar os dados de um JSON em um objeto ou *array*.

4.6 METODOLOGIA DE DESENVOLVIMENTO

A metodologia deve ser capaz de guiar a atividade de desenvolvimento, que consiste em um conjunto de atividades executadas para transformar um conjunto de requisitos em um sistema de software (SCOTT, 2003).

Na realização deste trabalho será adotada uma instância do Processo Unificado (*Unified Process* - UP), um processo de desenvolvimento que conta com uma estrutura genérica de processo que pode ser customizada por meio da adição ou remoção de atividades, com base nas necessidades específicas e nos recursos disponíveis para um projeto (SCOTT, 2003).

Utilizando os principais fundamentos do modelo de processo, como a direção por casos de uso, o foco na arquitetura e o ciclo de vida iterativo e incremental, são estipuladas atividades a serem desenvolvidas produzindo artefatos – documentação e código fonte – de valor para este projeto. Atividades e artefatos considerados dispensáveis são removidos para um melhor aproveitamento do tempo disponível.

O Quadro 5 lista os artefatos que serão gerados para cada fase do processo de desenvolvimento. O caráter iterativo e incremental permite que artefatos sejam retrabalhados, resultando na geração de novas versões sempre que necessário, independentemente da fase em que o projeto se encontre.

Fase	Artefatos
Concepção	Diagrama de Casos de Uso; Descrição de Cenários;
Elaboração	Diagrama de Atividades; Diagrama de Classes; Diagrama de Sequência;
Construção	Código fonte do software da instância mestre; Código fonte do software das instâncias escravas.
Transição	Documentação da codificação.

Quadro 5 - Artefatos gerados em cada fase do processo de desenvolvimento.

4.6.1 O PROCESSO UNIFICADO

O Processo Unificado (*Unified Process* - UP) é um processo de desenvolvimento que conta com uma estrutura genérica de processo que pode ser customizada por meio da adição ou remoção de atividades, com base nas necessidades específicas e nos recursos disponíveis para um projeto (SCOTT, 2003).

Por possuir uma especificação consideravelmente ampla e generalizada, com o intuito de atender projetos das mais diversas naturezas e tamanhos, foi necessário adequar este modelo de processo para este projeto, que é dotado de um número reduzido de atividades e desenvolvido individualmente. Neste sentido, artefatos que seriam úteis para auxiliar a comunicação entre grupos de trabalho em projetos com diversos profissionais trabalhando simultaneamente puderam ser desconsiderados a fim de tornar o tempo disponível livre para a confecção de artefatos que possuam um valor relevante dentro do cenário da aplicação.

A despeito da customização nos artefatos gerados, as três principais características do Processo Unificado foram consideradas para a ordenação das atividades, sendo elas:

- **Dirigido por casos de uso**: Um caso de uso é uma sequência de ações, executadas por um ou vários atores, os quais podem ser pessoas ou elementos externos automatizados, e pelo próprio sistema, que produz um ou vários resultados de valor para um ou mais atores. A expressão “dirigido por casos de uso” faz referência ao fato de se utilizar os casos de uso para dirigir todo o trabalho de desenvolvimento, desde a captação inicial e negociação junto aos *stakeholders* até a aceitação do código (SCOTT, 2003).

- **Centrado na arquitetura**: Um processo de *software* centrado na arquitetura define uma estratégia de compreender o sistema em sua modelagem de componentes, descrevendo como estes componentes interagem entre si, como eles são moldados, construídos e rastreados em todo o desenvolvimento do software (ARLOW et al., 2002).

- **Iterativo e Incremental**: Cada iteração é um subprojeto que resulta em uma versão do sistema, podendo esta ser liberada interna ou externamente. Tem-se como objetivo que cada subprojeto ofereça uma melhora incremental sobre o anterior. (SCOTT, 2003).

As atividades realizadas durante as fases e workflows, bem como os artefatos gerados, serão detalhados no tópico 5 .

5 DESENVOLVIMENTO

O desenvolvimento do sistema seguiu as fases estabelecidas pela metodologia de desenvolvimento adaptada a partir do Processo Unificado, descrita no item 4.6.

Os *workflows* das fases de concepção, elaboração, construção e transição foram trabalhados e produziram artefatos de modelagem, documentação e código fonte. Na sequência serão exibidos os artefatos gerados durante o desenvolvimento, em que a realização de uma iteração completa que foi capaz de gerar uma primeira versão funcional do *software*.

5.1 CONCEPÇÃO

A fase de concepção preocupou-se com a delimitação do escopo do sistema, definindo restrições e limitações, bem como elucidando os principais requisitos para o desenvolvimento. Neste sentido, o foco esteve na definição das principais funcionalidades que a versão produzida nesta iteração deve presentar.

Usuários e funcionalidades são representados por atores e casos de uso no Diagrama de Casos de Uso apresentado na Figura 12.

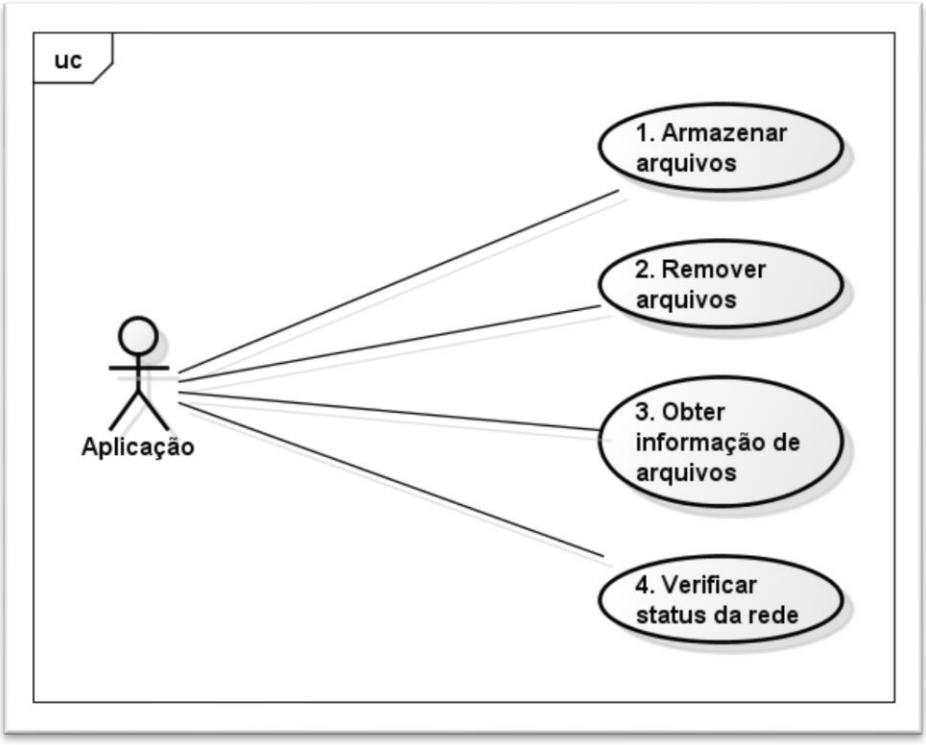

Figura 12 - Diagrama de Casos de Uso

O ator denominado “Aplicação” representa o único usuário do agente de replicação, que será o sistema do website que utilizará as funções disponibilizadas.

As funcionalidades tangíveis ao usuário são “Armazenar arquivos”, “Remover Arquivos”, “Obter Informações de arquivos” e “Verificar status da rede”. Cada caso de uso é posteriormente expandido e detalhado por meio de um modelo de cenário, que representa uma descrição textual em nível de detalhamento capaz de guiar as fases posteriores nas atividades de análise, projeto e implementação.

Cada cenário recebe um identificador único, sequencial e hierárquico em relação ao caso de uso que representa. O título expressa claramente a ação que será realizada, que pode ser mais bem conhecida pelo campo destinado ao resumo. Os principais atores interessados nos resultados gerados pela ação são descritos nos campos “Ator principal” e “Ator secundário”. Pré-condições determinam a forma com que o ambiente deve se encontrar para que esta ação possa ser realizada, assim como pós-condições descrevem ações que deverão ser realizadas ou estados que deverão ser assumidos após a execução do fluxo de ações. As validações que deverão ser efetuadas também são listadas detalhadamente.

A seguir, os quadros de 6 a 9 detalham os modelos de cenários para os casos de uso identificados.

Identificação:	1.1
Título:	Armazenar arquivos a partir de uma lista
Ator Principal:	Servidor de Aplicação do Website
Ator Secundário:	Não há
Resumo:	Este cenário descreve a entrega de uma lista de arquivos do servidor de aplicação para um servidor da rede de distribuição de conteúdo.
Pré-Condições:	- Os arquivos devem estar armazenados localmente no servidor de aplicação ou em qualquer outro local acessível via requisição HTTP; - Deve ser possível o cálculo do checksum dos arquivos utilizando o método de algoritmo de hash MD5 implementado pela função md5_file da linguagem PHP. - O servidor de aplicação deve fornecer um identificador único para cada arquivo; - Deve existir uma conexão de rede ativa entre ambos servidores;
Validações:	

FLUXO DE AÇÕES	
Ações do Ator	Ações do Sistema
1. Uma requisição GET é enviada solicitando a sincronização e indicando a URL onde será realizada a leitura dos dados no servidor de aplicação.	
	2. A solicitação é recebida.
	3. Os parâmetros da solicitação são validados.
	4. É realizada uma requisição GET para buscar os dados dos arquivos a serem transferidos.
	5. Os dados dos arquivos a serem transferidos são validados.
	6. Para cada arquivo, é realizada uma requisição GET para o download a partir do servidor de aplicação.
	7. Verifica-se a existência da estrutura de diretórios necessárias para o armazenamento do arquivo (path). Não havendo tal estrutura, ela deverá ser criada.
	8. É realizado o cálculo do checksum do arquivo.
	9. Ao término do processamento de todos os arquivos, os resultados são reunidos e retornados ao servidor de aplicação.
10. O resultado final da operação é recebido pelo servidor de aplicação.	

Pós-Condições:	- Será retornada pelo servidor da CDN para o servidor da aplicação uma representação JSON contendo o status da operação de cada um dos arquivos processados.
Observações:	- Caso seja realizada uma tentativa de enviar um arquivo com identificador já existente, o arquivo anterior será substituído.

Quadro 6 - Cenário "Armazenar arquivos a partir de uma lista" para o caso de uso "Armazenar Arquivos"

Identificação:	2.1
Título:	Remover uma lista de arquivos
Ator Principal:	Servidor de Aplicação do Website
Ator Secundário:	Não há
Resumo:	Este cenário descreve o processamento de uma ordem para exclusão de uma lista de arquivos da rede de distribuição de conteúdo.
Pré-Condições:	- O servidor de aplicação deve fornecer um identificador único para o arquivo; - Deve existir uma cópia do arquivo na rede de distribuição de conteúdo; - Deve existir uma conexão de rede ativa entre ambos servidores;
Validações:	

FLUXO DE AÇÕES	
Ações do Ator	Ações do Sistema
1. Uma requisição GET é enviada solicitando a exclusão e indicando a URL onde será realizada a leitura dos dados no servidor de aplicação.	
	2. A solicitação é recebida.
	3. Os parâmetros da solicitação são validados.
	4. É realizada uma requisição GET para buscar os dados dos arquivos a serem excluídos.
	5. Os dados dos arquivos a serem excluídos são validados.
	6. É realizada a exclusão de cada um dos arquivos listados.
	7. Verifica-se se há diretórios vazios entre o diretório raiz da aplicação e o diretório em que o arquivo se encontrava. Existindo diretórios sem nenhum arquivo, estes deverão ser removidos.
	8. Ao término da exclusão de todos os arquivos da lista, os resultados são reunidos e retornados ao servidor de aplicação.
9. O resultado final da operação é recebido pelo servidor de aplicação.	

Pós-Condições:	- Será retornada pelo servidor da CDN para o servidor da aplicação uma representação JSON contendo o status da operação de cada um dos arquivos processados.
Observações:	- Caso seja realizada uma tentativa de exclusão a um arquivo não existente nenhum erro será gerado.

Quadro 7 - Cenário "Remover uma lista de arquivos" para o caso de uso "Remover Arquivos"

Identificação:	3.1
Título:	Obter informação de um ou mais arquivos
Ator Principal:	Servidor de Aplicação do Website
Ator Secundário:	Não há
Resumo:	Este cenário descreve o procedimento para a obtenção de informações sobre um arquivo. As informações são voltadas para a realização de testes de consistência entre os arquivos da rede de distribuição de conteúdo e o servidor de aplicação.
Pré-Condições:	- O servidor de aplicação deve fornecer um identificador único para o arquivo; - Deve existir uma cópia do arquivo na rede de distribuição de conteúdo; - Deve existir uma conexão de rede ativa entre ambos servidores;
Validações:	

FLUXO DE AÇÕES	
Ações do Ator	Ações do Sistema
1. Uma requisição GET é enviada solicitando a listagem de informações e indicando a URL onde será realizada a leitura dos dados no servidor de aplicação.	
	2. A solicitação é recebida.
	3. Os parâmetros da solicitação são validados.
	4. É realizada uma requisição GET para buscar os dados dos arquivos a terem informações listadas.
	5. Para cada arquivo na listagem, as informações sobre seus atributos são recuperadas.
	6. Ao término do processamento de todos os arquivos da lista, os resultados são reunidos e retornados ao servidor de aplicação.
7. O resultado final da operação é recebido pelo servidor de aplicação.	

Pós-Condições:	- Será retornada pelo servidor da CDN para o servidor da aplicação uma representação JSON contendo o status da operação de cada um dos arquivos processados. - Caso um arquivo da lista não seja encontrado na rede de distribuição de conteúdo, o sistema deverá identificar a falha e reportar o erro por meio dos dados fornecidos.
Observações:	

Quadro 8 - Cenário "Obter informação de um ou mais arquivos" para o caso de uso "Obter informação de arquivos"

Identificação:	4.1
Título:	Verificar status da rede de distribuição de conteúdo
Ator Principal:	Servidor de Aplicação do Website
Ator Secundário:	Não há
Resumo:	Este cenário permite que o servidor de aplicação monitore a rede de distribuição de conteúdo. Em caso de alguma indisponibilidade, o servidor de aplicação poderá realizar alteração automáticas em suas URLs para direcionar o tráfego para outra rede.
Pré-Condições:	- Deverá ser configurado um timeout em milissegundos para a operação.
Validações:	

FLUXO DE AÇÕES	
Ações do Ator	Ações do Sistema
1. Uma requisição GET é enviada para a rede de distribuição de conteúdo.	
	2. A solicitação é recebida.
	3. A solicitação é respondida com o status atual da rede.
4. O resultado final da operação é recebido pelo servidor de aplicação.	

Pós-Condições:	- Será retornada pelo servidor da CDN para o servidor da aplicação uma representação em forma de String contendo o status da operação de cada um dos arquivos processados. - Caso o timeout definido seja atingido sem a conclusão da operação, será retornado um estado de falha.
Observações:	

Quadro 9 - Cenário "Verificar status da rede de distribuição de conteúdo" para o caso de uso "Verificar status da rede"

5.2 ELABORAÇÃO

Durante a fase de elaboração, os requisitos identificados na fase de concepção, que foram especificados pelos Casos de Uso e Modelos de Cenários, transformam-se em diagramas voltados à modelagem estrutural e comportamental do sistema. Com este objetivo, foram desenvolvidos os diagramas de classes, com foco na modelagem estrutural das classes com seus atributos e métodos, e diagramas de atividades e sequência, com ênfase na análise da ordenação de atividades e na troca de mensagens entre objetos.

Na Figura 13 é possível visualizar o Diagrama de Classes do módulo mestre do agente de replicação. O módulo mestre é a divisão do sistema que poderá ser acoplada à aplicação para que, por meio de seus objetos, seja possível executar ordens de manipulação de arquivos entre o servidor de aplicação e a rede de distribuição de conteúdo.

A classe *CDNApplicationManager* será responsável pela interface externa do agente de replicação para com a aplicação, disponibilizando métodos para a execução de operações, tais como:

- *send(fileListURL : String)*, cuja função é solicitar a coleta de um ou mais arquivos presentes no servidor de aplicação para que sejam armazenados na rede de distribuição de conteúdo. O parâmetro de entrada para este método é a URL para uma lista de objetos da classe *File*, listados conforme a especificação do formato JSON. A lista de objetos deverá ser montada pela aplicação, de acordo com os arquivos que serão transferidos;
- *purge(fileListURL : String)*, cuja função é solicitar a remoção de um ou mais arquivos que estejam armazenados na rede de distribuição de conteúdo. Como parâmetro de entrada, a função espera uma URL para uma lista em formato JSON com objetos da classe *File* com o campo *id* preenchido com o identificador único do arquivo que foi especificado no momento do seu envio;
- *info(fileListURL : String)*, método que executa a verificação de informações de arquivos que estão incluídos em uma lista de objetos *File* codificada em formato JSON.
- *status()*, cujo objetivo é obter informações sobre a disponibilidade atual da rede de distribuição de conteúdo.

A classe *File* é uma abstração de um arquivo armazenado fisicamente no servidor de aplicação. No entanto, apenas as informações relevantes para a rede de distribuição de conteúdo são armazenadas em seus atributos. Os atributos armazenam um identificador único (*id*), o nome do arquivo (*name*), o endereço relativo de armazenamento a partir do diretório raiz da aplicação (*path*), o domínio do servidor em que se encontra (*domain*), seu tamanho em número de bytes (*size*), uma *string* de 32 caracteres correspondente ao *checksum* realizado por meio do algoritmo MD5 (*hash*) e a data de criação do arquivo (*created*). Por fim, o método *getResume*() oferece uma maneira direta de obter todas as informações do arquivo em uma *string* no formato JSON.

Por sua vez, a classe CDN representa um ou mais servidores de mídia que estão sendo gerenciados pelo agente. As informações armazenadas pelos objetos que abstraem os servidores escravos são um nome único (*name*), um

endereço de acesso (*url*), uma chave de acesso (*key*) e um tempo limite de espera por cada operação realizada (*timeout*). Os métodos da classe fazem parte do seu mecanismo de carga automática a partir de um arquivo de configuração local.

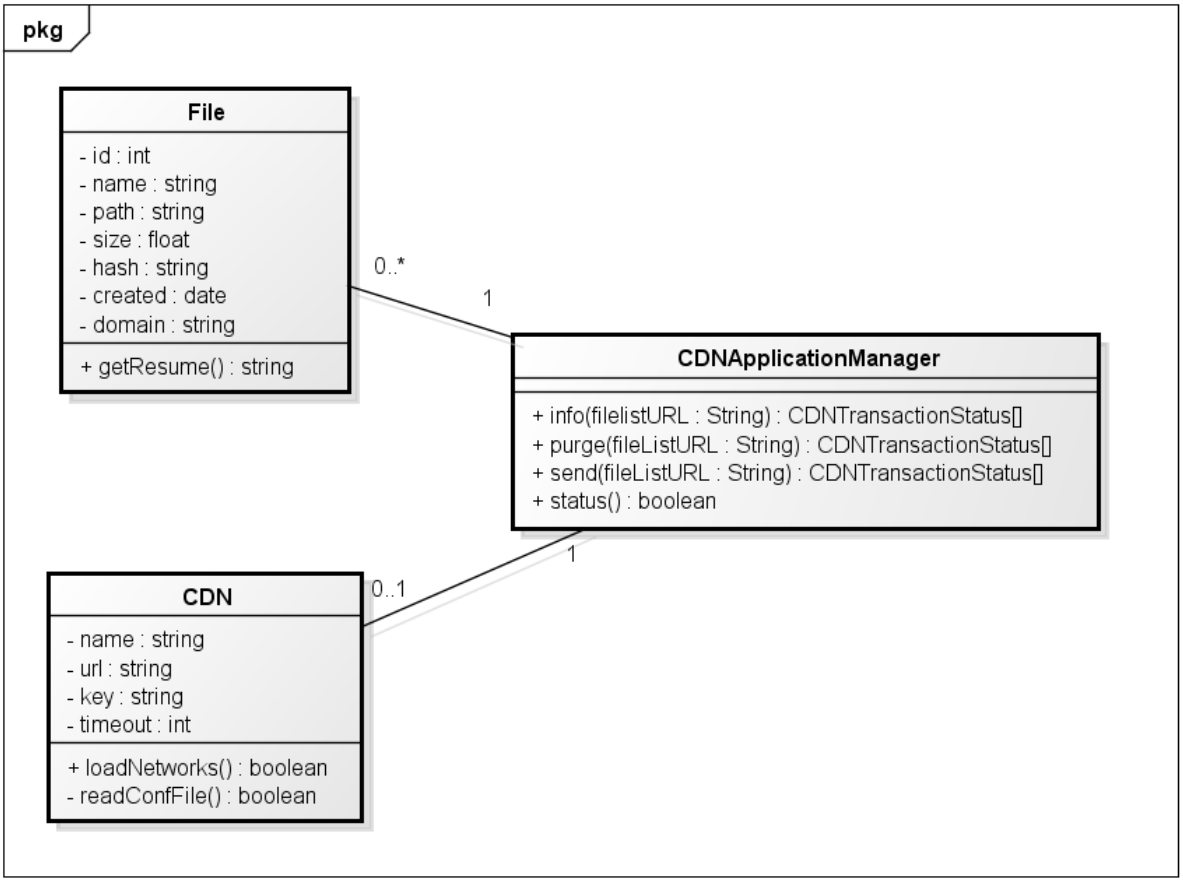

Figura 13 - Diagrama de Classes do módulo mestre.

Na Figura 14, o Diagrama de Classes do módulo escravo, que é um serviço instalado em cada um dos servidores de mídia da rede de distribuição de conteúdo e será responsável por responder às requisições enviadas pelo servidor de aplicação.

A classe *CDNAgent* é a responsável por processar as solicitações enviadas pelo servidor de aplicação, implementando métodos complementares a cada um dos métodos presentes na classe *CDNApplicationManager* do módulo mestre. Os métodos disponibilizados são:

- *receive(fileListURL : String)*, responsável por receber as solicitações e coletar arquivos no servidor de aplicação. Recebe por parâmetro a URL para download de uma lista em formato JSON, a partir da qual obterá

informações sobre os arquivos que deverá transferir e armazenar, realizando as validações de integridade necessárias;

- *purge(fileListURL : String)*, método que recebe como parâmetro de entrada uma lista em formato JSON enviada pelo método de mesmo nome no módulo mestre. A partir da lista com a descrição dos arquivos, realiza a remoção daqueles que foram listados no servidor escravo.
- *info(fileListURL : String)*, responsável por responder ao método com mesmo nome no módulo mestre, fornecendo as informações do objeto *File* por meio de uma lista em formato JSON.
- *status()*, responsável pela análise do *status* de um servidor da rede de distribuição de conteúdo, retornando uma *string* com o atual estado do mesmo.
- *buildTransactionReport(status : CDNTransactionStatus[])* é um método responsável pelo registro do estado de cada procedimento realizado, compilando objetos *CDNTransactionStatus* em uma única lista em formato JSON para que esta possa ser encaminhada ao servidor de aplicação.

Os métodos privados createPath(), deleteEmptyDirs() e isEmptyDir() realizam, respectivamente, operações de criação, exclusão e verificação da hierarquia de diretórios em que se encontra um arquivo manipulado pelos métodos públicos.

A classe *CDNFile* apresenta as mesmas propriedades da classe *File* presente no módulo mestre, contudo, implementa métodos adicionais para operações de manipulação com os arquivos presentes no servidor de mídia.

A classe *IOHelper* apresenta métodos utilitários para o download de arquivos remotos, que podem ser listas em formato JSON ou qualquer tipo de arquivo estático que, por solicitação do método *receive*() da classe *CDNAgent*, deverá ser transferido e armazenado localmente neste servidor.

Por fim, a classe *CDNTransactionStatus* armazena, para cada arquivo envolvido em uma dada operação, o estado de sucesso ou falha, reunindo informações que serão compiladas pelo método *buildTransactionReport*() da classe *CDNAgent* e, ao término da operação solicitada, transferidas ao servidor de

aplicação para que este possa gerenciar possíveis falhas ocorridas durante o processamento da lista de arquivos.

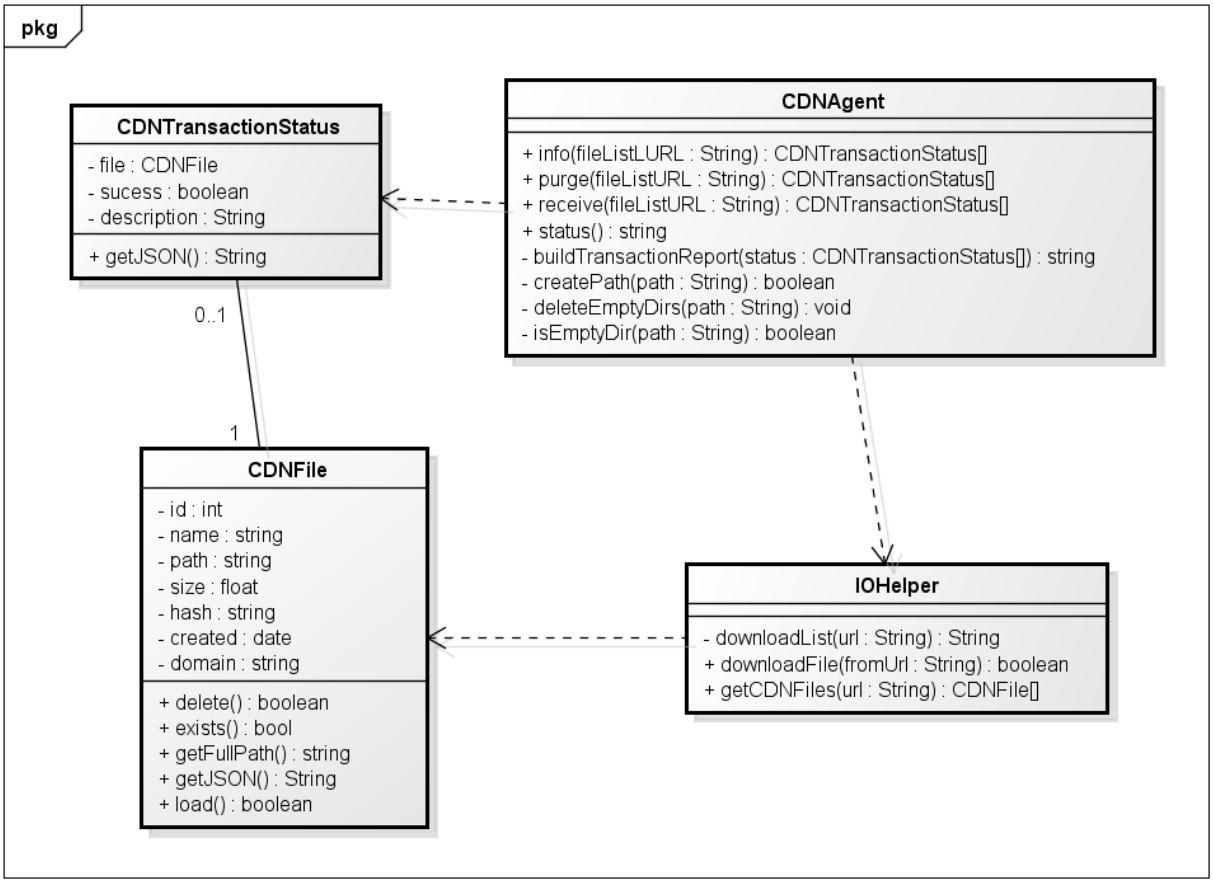

Figura 14 - Diagrama de Classes do módulo escravo.

Para a modelagem comportamental do sistema, foram utilizados diagramas de atividades e diagramas de sequência. A análise comportamental foi desenvolvida a partir da descrição dos modelos de cenários criada na fase de concepção, apresentada no item 5.1.

Na Figura 15 é apresentado o diagrama de atividades correspondente ao cenário “Armazenar uma lista de arquivos”, elaborado durante a fase de concepção e apresentado no Quadro 6. Na sequência, a Figura 16 representa as atividades necessárias para a implementação do cenário “Remover uma lista de arquivos”, a Figura 17 demonstra as atividades para a execução do cenário “Obter informação de um ou mais arquivos” e a Figura 18 enumera as atividades para a construção do cenário “Verificar status da rede de distribuição de conteúdo”.

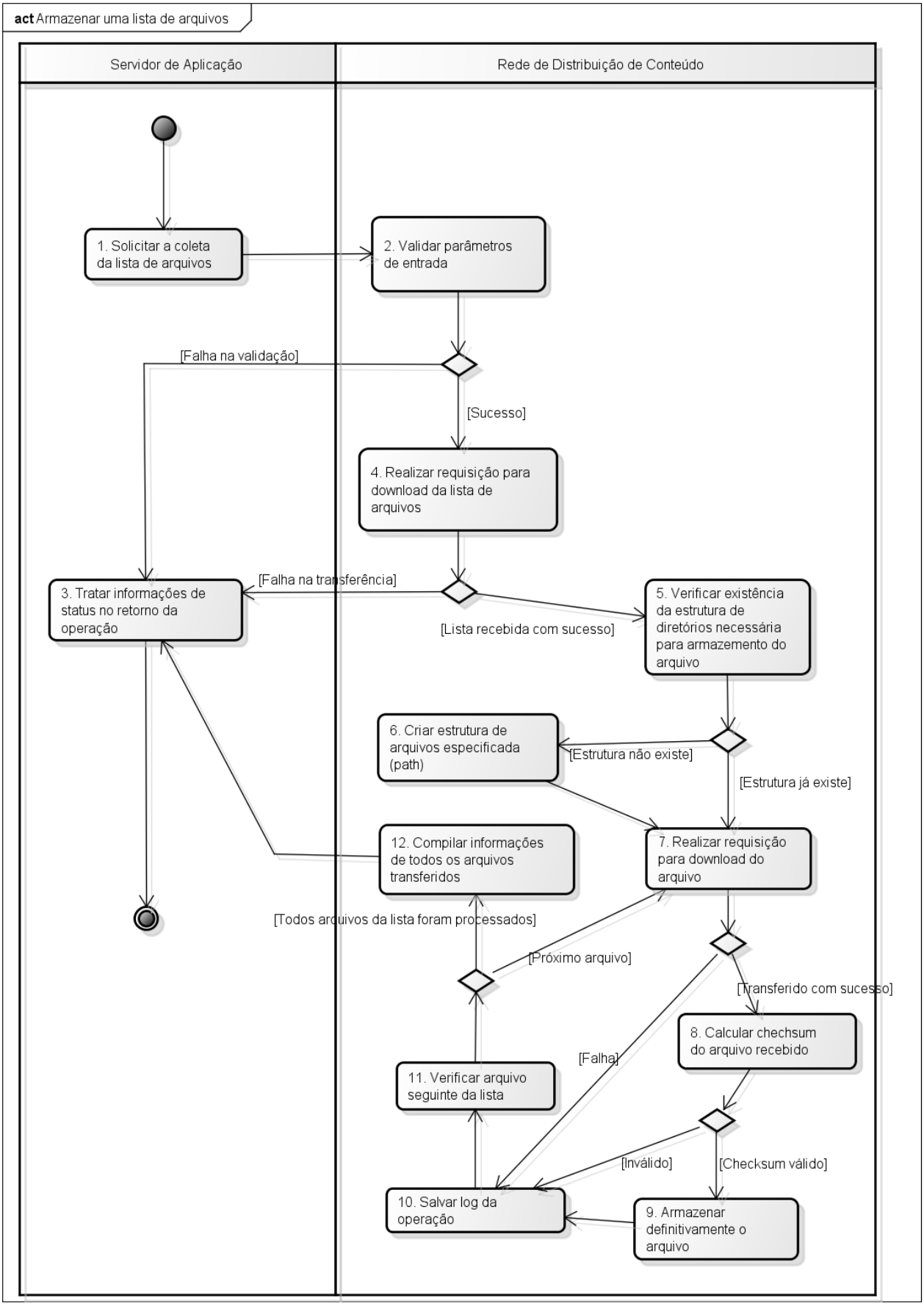

Figura 15 - Diagrama de Atividades: Armazenar uma lista de arquivos

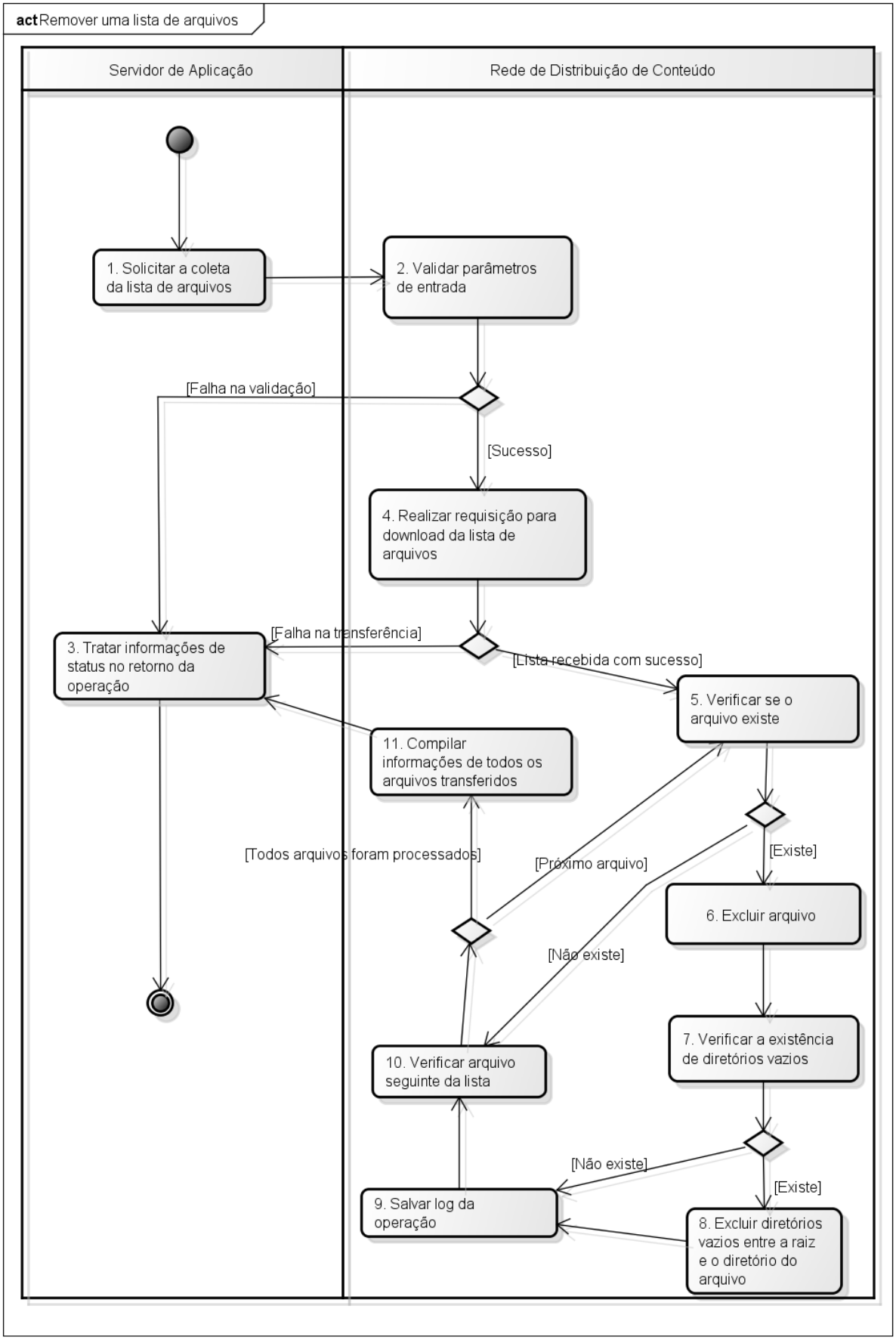

Figura 16 - Diagrama de Atividades: Remover uma lista de arquivos

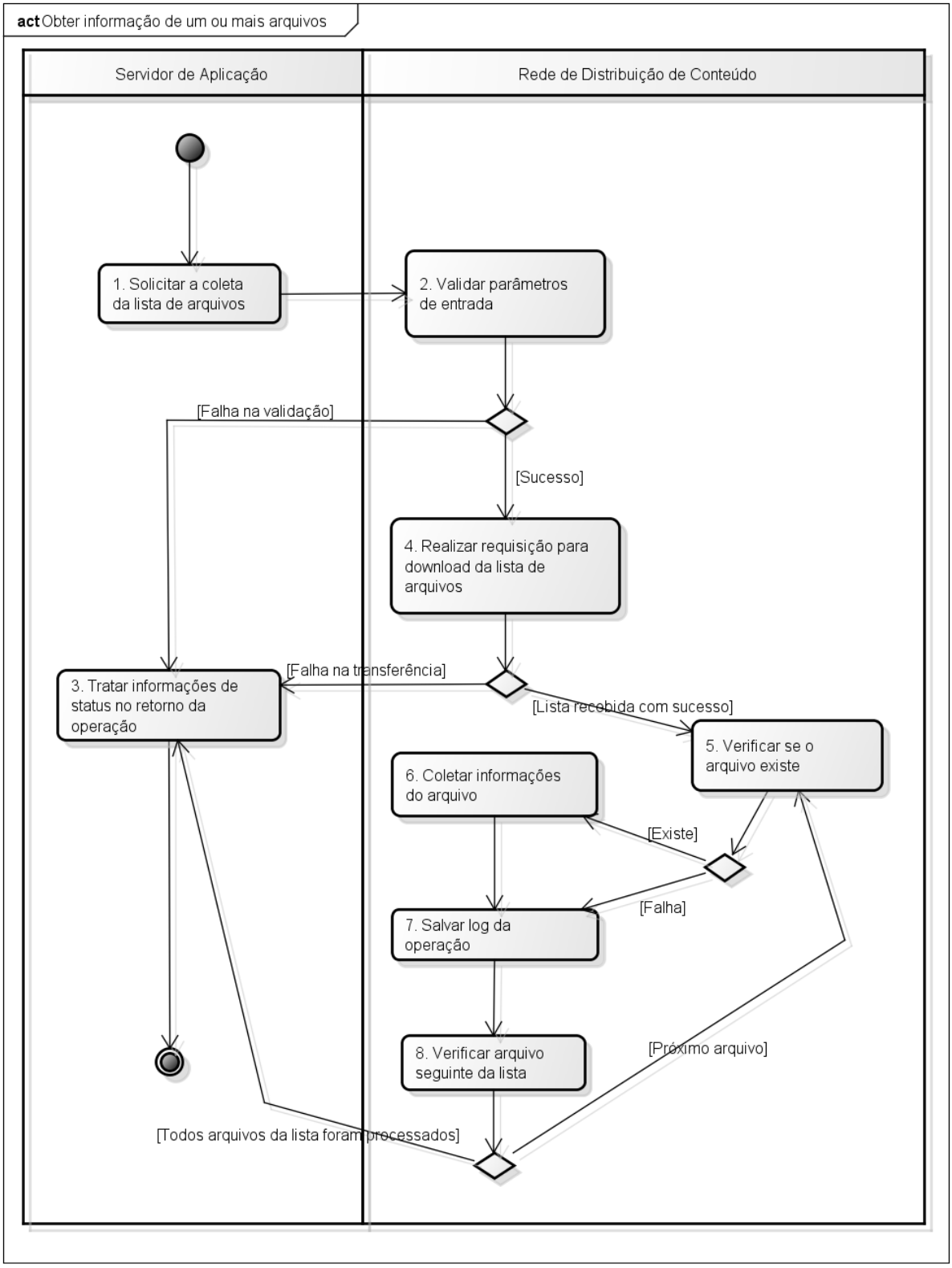

Figura 17 - Diagrama de Atividades: Obter informação de um ou mais arquivos

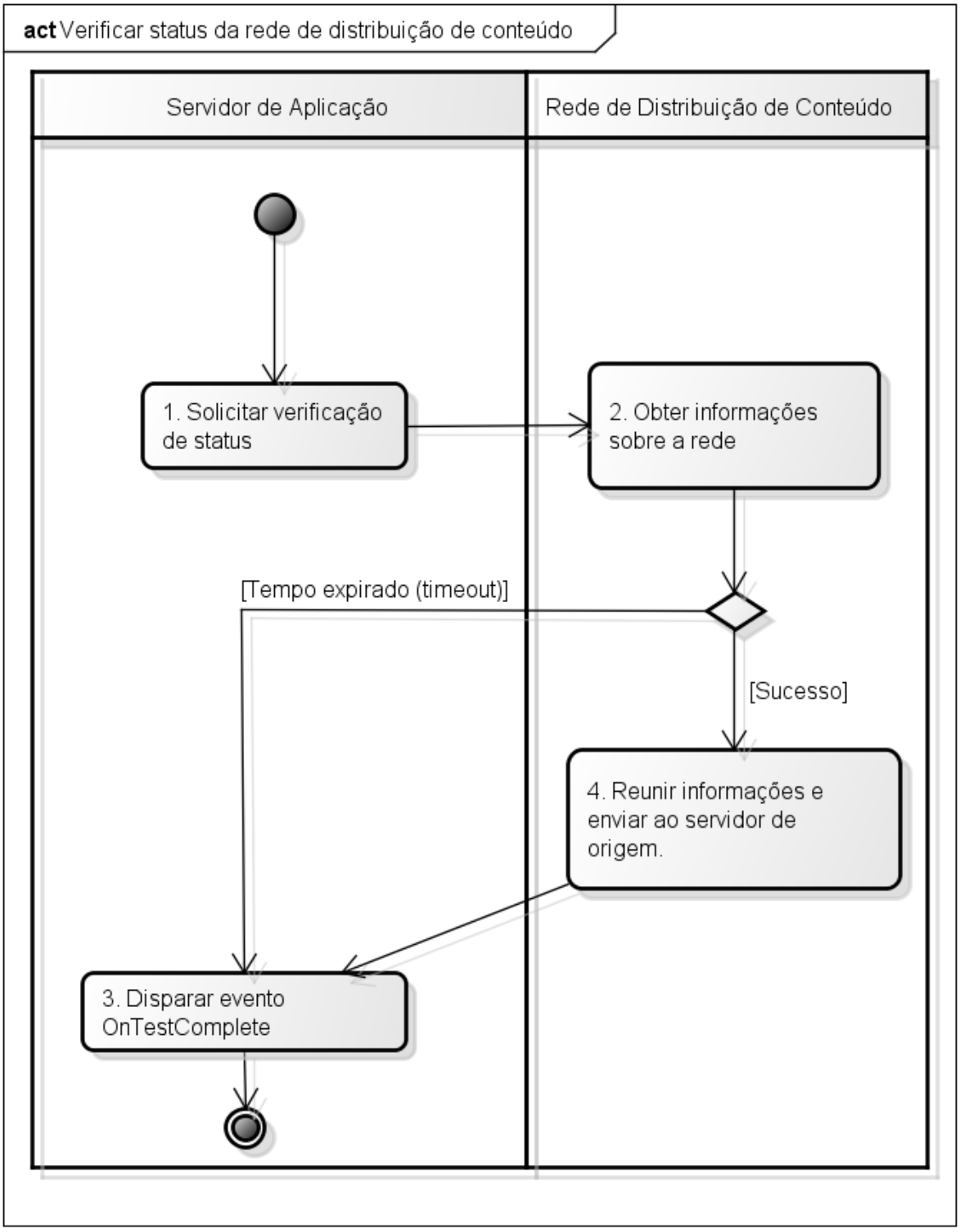

Figura 18 - Diagrama de Atividades: Verificar status da rede de distribuição de conteúdo

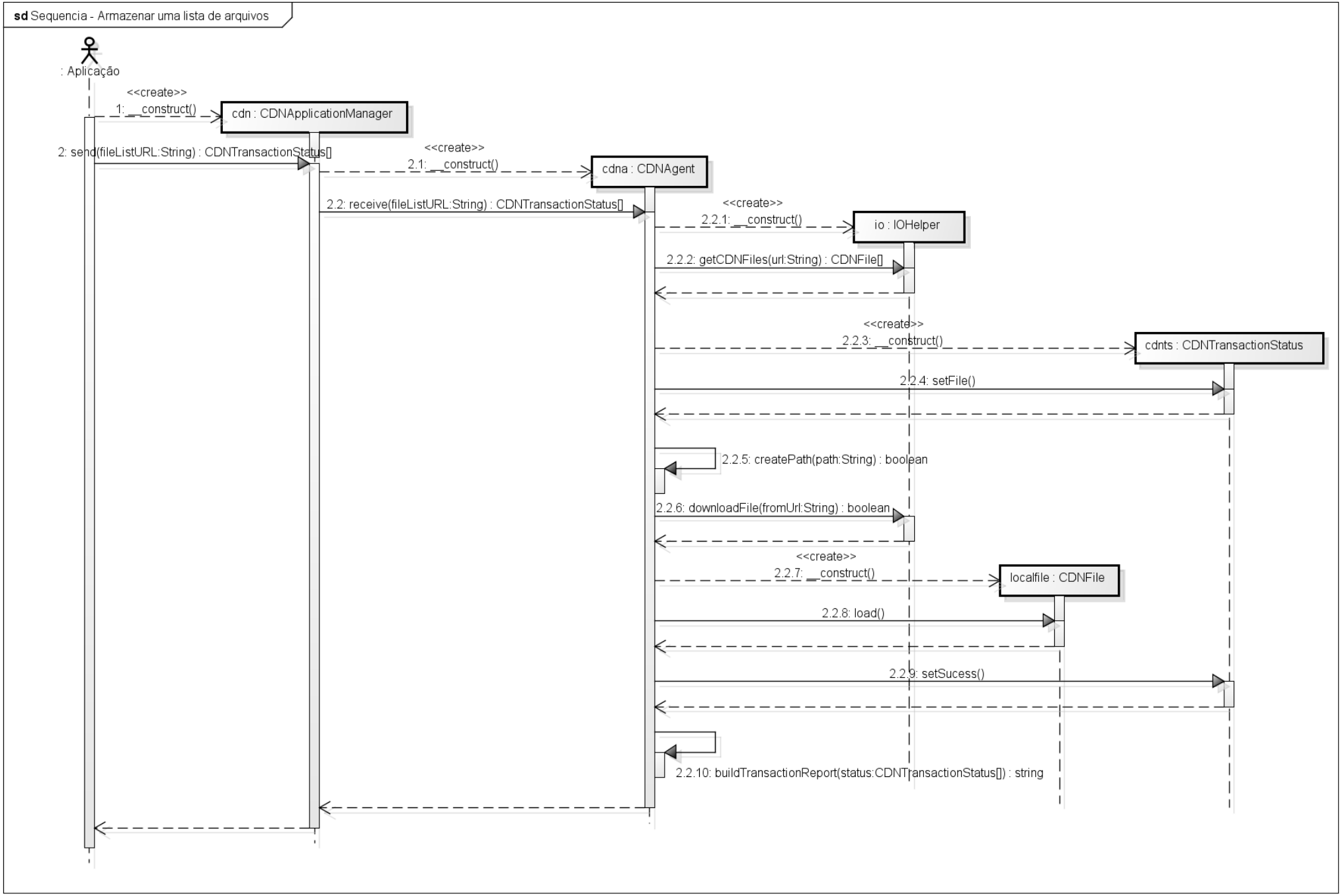

Figura 19 - Diagrama de Sequência: Armazenar uma lista de arquivos

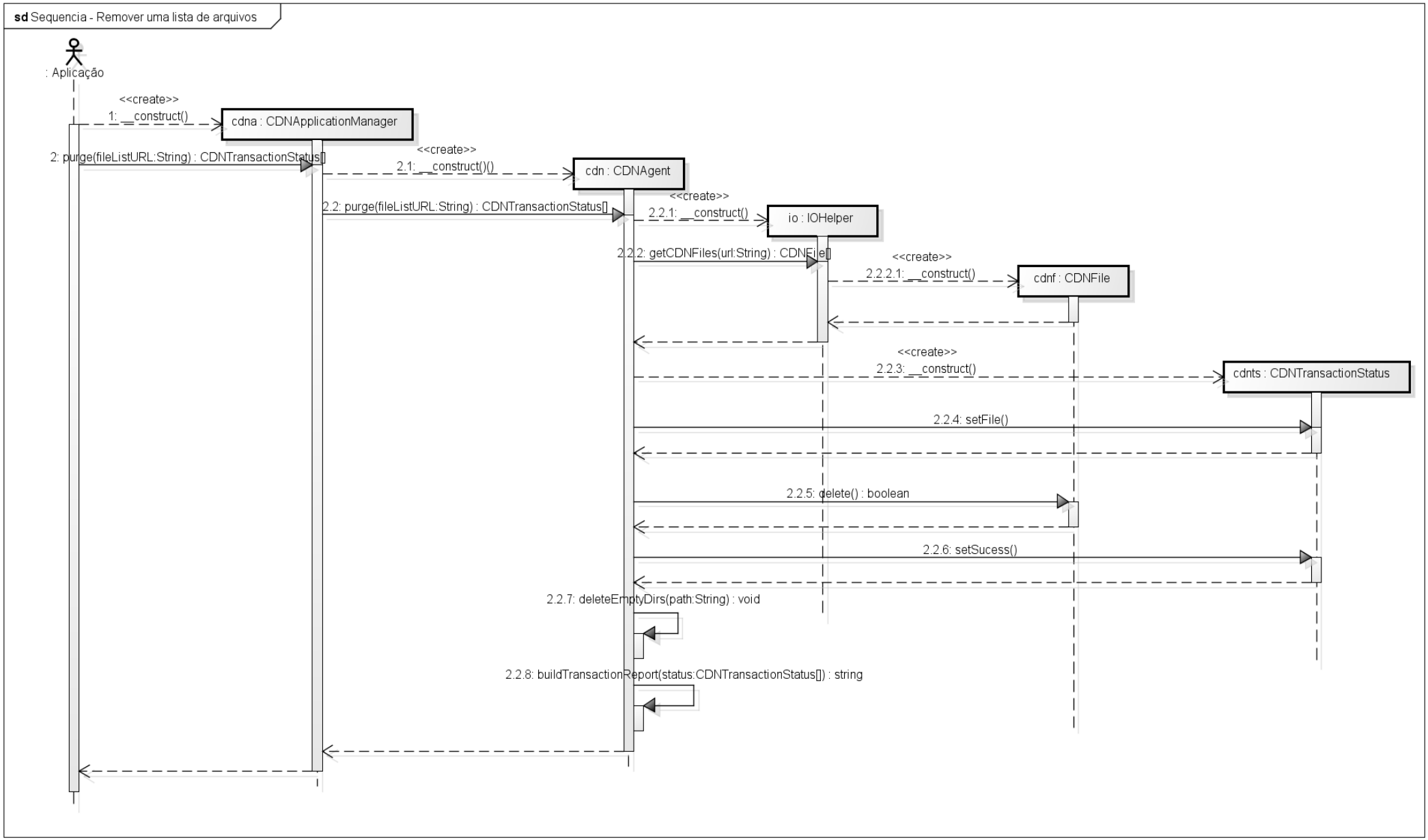

Figura 20 - Diagrama de Sequência: Remover uma lista de arquivos

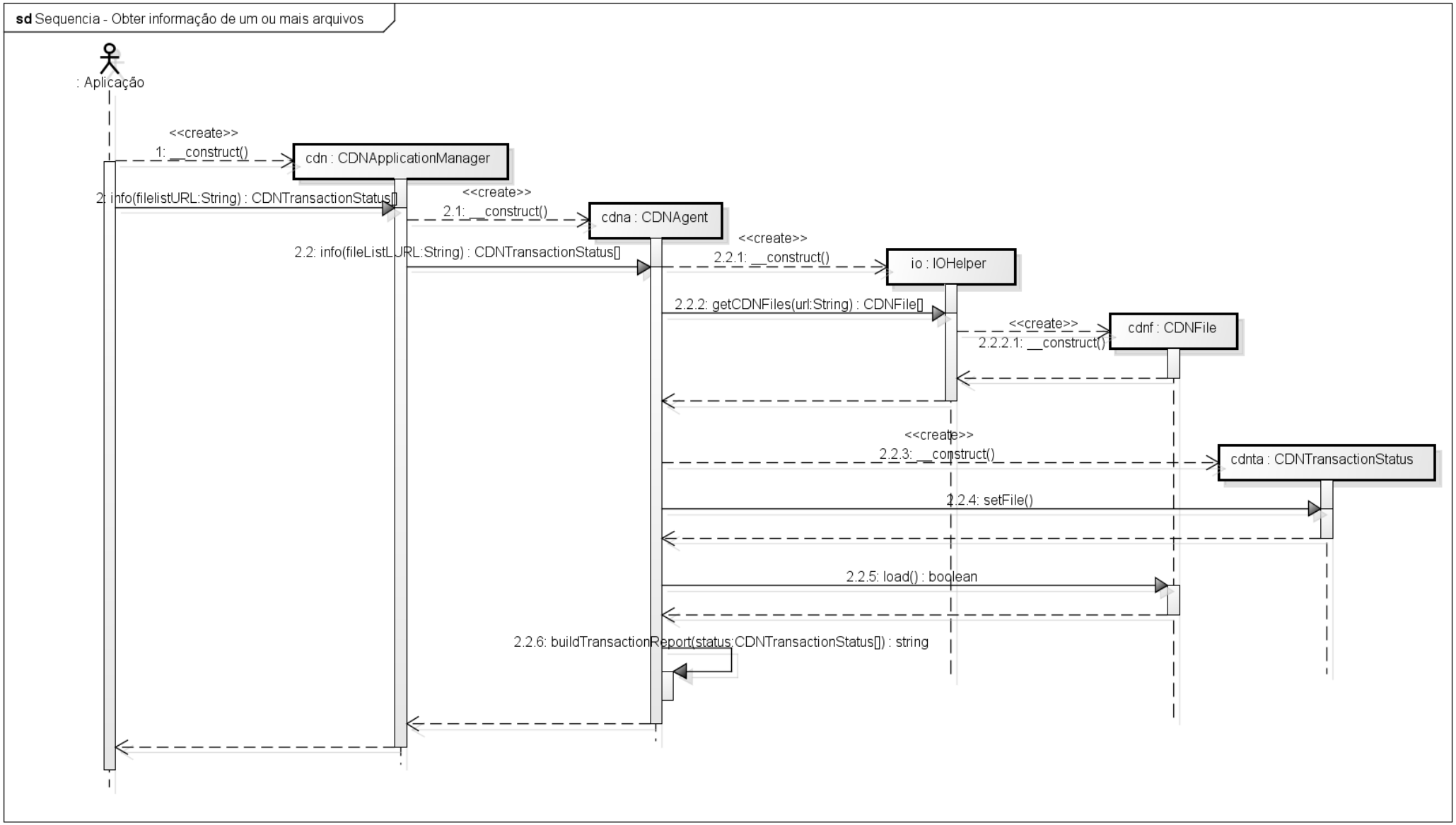

Figura 21 - Diagrama de Sequência: Obter informação de um ou mais arquivos

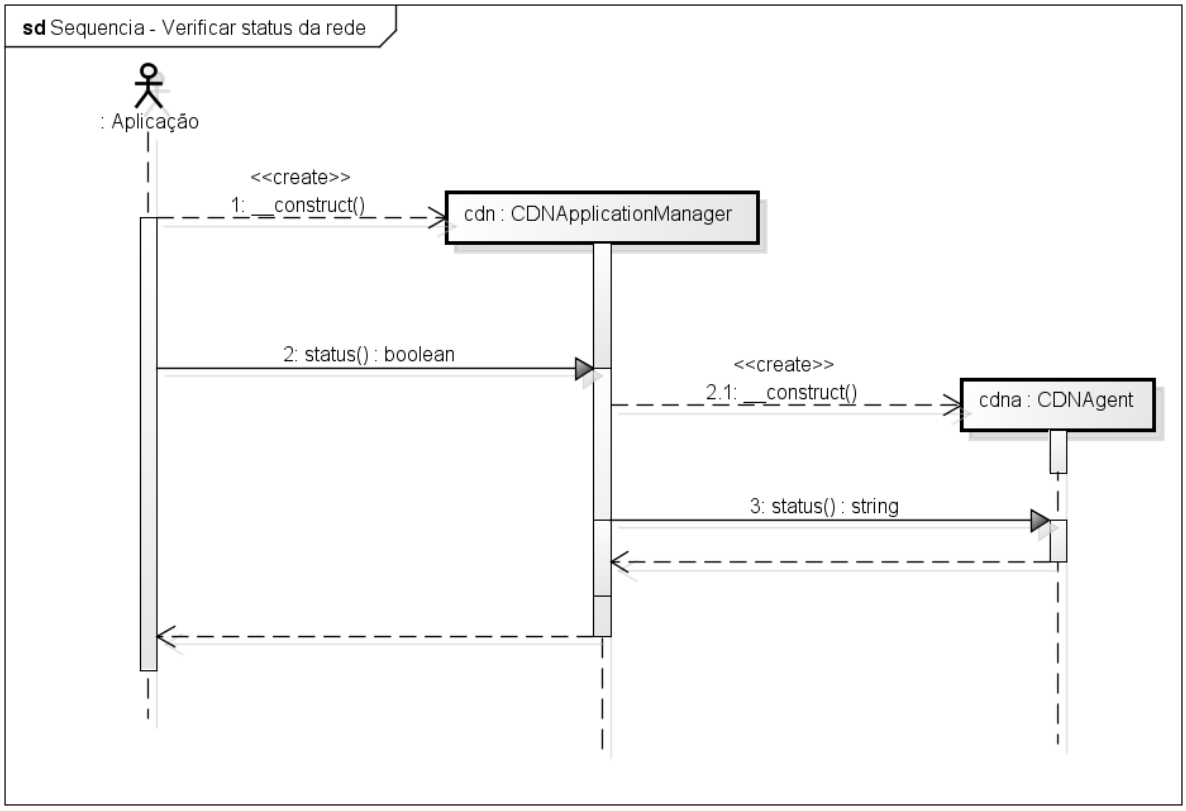

Figura 22 - Diagrama de Sequência: Verificar status da rede

5.3 CONSTRUÇÃO

Na fase de construção, os esforços se concentraram na codificação das estruturas elaboradas nas fases anteriores. Arquivos com código PHP foram criados a fim de expressar as classes e seus respectivos atributos e métodos. Os modelos de cenários guiaram a sequência de atividades com o apoio dos diagramas de atividades e sequência e também puderam ser utilizados como caso de testes a fim de garantir a correta implementação de cada um dos casos de uso.

A Figura 23 demonstra a implementação do método *receive* da classe *CDNAgent*, presente no módulo localizado na rede de distribuição de conteúdo. A modelagem comportamental deste método foi apresentada pelo diagrama de atividades na Figura 15 e a troca de mensagens entre objetos é representada pelo diagrama de sequência da Figura 19.

```
public function receive($fileListURL){
    require_once('CDNFile.php');
    require_once('IOHelper.php');
    require_once('CDNTransactionStatus.php');
    $io = new IOHelper();
    $files = $io->getCDNFiles($fileListURL);
    $statuses = array();
    foreach($files as $file){
        $status = new CDNTransactionStatus();
        $status->setFile($file);
        $this->createPath($file->getPath());
        if($io->downloadFile($file->getDomain().$file->getPath().
                $file->getName(), $file->getPath().$file->getName())){
            $localFile = new CDNFile();
            $localFile->setPath($file->getPath());
            $localFile->setName($file->getName());
            $localFile->load();
            $checksum = ($localFile->getHash()==$file->getHash());
            $status->setSucess($checksum);
        }else{
            $status->setSucess(false);
        }
        array_push($statuses, $status);
    }
    return $this->buildTransactionReport($statuses);
}
```

Figura 23 - Codificação do método receive da classe CDNAgent

5.4 TRANSIÇÃO

A transição é a fase que marca o final de uma iteração dentro do processo de desenvolvimento. Com a codificação finalizada na fase de construção, foram realizados testes de integração entre os módulos mestre e módulos escravos em ambiente de desenvolvimento e, posteriormente, em ambiente que simula um ambiente real de produção.

Por fim, foi gerada a documentação das classes que compõem o sistema desenvolvido a fim de fornecer o conhecimento necessário sobre seus métodos públicos para que possam ser utilizados pela aplicação a que serão integrados.

6 CONSIDERAÇÕES FINAIS

Este capítulo relata as experiências obtidas por meio da pesquisa sobre redes de distribuição de conteúdo e pelo desenvolvimento de um agente de replicação para servidores de arquivos estáticos para *websites*. Os aspectos que não puderam ser realizados neste trabalho, bem como possíveis melhorias às funcionalidades já agregadas ao sistema criado, serão descritos na forma de trabalhos futuros.

6.1 CONCLUSÃO

As redes de distribuição de conteúdo são um recurso inevitável para sites com grande volume de tráfego, principalmente para aqueles cujo principal conteúdo é baseado em arquivos estáticos como imagens, vídeos ou qualquer tipo de arquivo que não seja o próprio hipertexto. Nestes cenários há um grande número de requisições para o download destes arquivos sempre que uma página é carregada, sobrecarregando o servidor de aplicação pelo uso excessivo de processos por parte do servidor web e também pela alta taxa de utilização da unidade de disco para busca e leitura de arquivos.

Os benefícios proporcionados por uma rede de distribuição de conteúdo são perceptíveis tanto pelos usuários como pelos administradores de um *website*, uma vez que o tempo de carregamento de arquivos estáticos tenderá a ser reduzido graças à proximidade geográfica entre a rede do usuário e a rede que hospeda o servidor de arquivos. Da mesma forma, a utilização de uma rede de distribuição de conteúdo desonera o servidor de aplicação, permitindo uma utilização mais racional de recursos de processamento, memória e, principalmente, acesso a disco, por meio do desvio de grande parte de requisições para outros servidores dispersos geograficamente. Desta forma, outros serviços hospedados no servidor de aplicação terão maior eficiência na utilização de processos que dependem de sua disponibilidade de recursos computacionais, como, por exemplo, um serviço de banco de dados que fará uso frequente de operações de escrita e leitura de informações na unidade de armazenamento.

Além disso, a flexibilidade proporcionada por algumas formas de distribuição de conteúdo colaborara para atender a necessidades específicas de

algumas aplicações web como, por exemplo, a necessidade de desviar um maior volume de tráfego para locais em que há maior largura de banda para transferência ou, até mesmo, menor custo por gigabyte transferido, o que se torna uma possibilidade atraente em localidades em que os recursos de infraestrutura de rede ainda são precários e demasiadamente custosos.

Por sua vez, a adoção de uma rede de distribuição de conteúdo requer uma análise minuciosa a fim de detectar os padrões de consumo de tráfego, tendo em vista a origem geográfica das requisições e quais arquivos são mais frequentemente solicitados, a fim de implantar um modelo capaz de combinar uma solução otimizada, corretamente dimensionada e que demande o menor número possível de alterações na codificação já existente no *website*.

6.2 TRABALHOS FUTUROS

A versão do agente de replicação desenvolvida neste trabalho realiza as funcionalidades essenciais para a transferência de arquivos com garantias de integridade necessárias para assegurar que o arquivo tenha sido copiado sem perdas. Todavia, a partir das funções básicas será possível agregar novas funcionalidades ou aperfeiçoar muitas das funcionalidades já existentes.

Novas versões das funções de envio, exclusão e obtenção de informações de arquivos poderão incluir funcionalidades para o envio assíncrono, sem que haja a necessidade do servidor de aplicação permanecer no aguardo de uma resposta para cada requisição enviada. Desta forma, seria possível incluir chamadas aos objetos do agente de replicação diretamente dentro do código da aplicação e um processo realizaria sincronizações agendadas periodicamente. Por outro lado, este aperfeiçoamento exige um estudo de viabilidade, tendo em vista o aumento da complexidade do agente devido à necessidade de coordenar solicitações e notificações de resposta.

Outro recurso consideravelmente relevante é a agregação de um mecanismo que possibilite novas tentativas de transferência para um arquivo cuja transferência falhou. Esta possibilidade seria interessante uma vez que reduziria o

número de erros que serão reportados à aplicação, já que simples erros de transferência devido a indisponibilidades momentâneas do arquivo ou da rede poderiam ser contornados facilmente, sem a necessidade de envolver o servidor de aplicação para que este realize novas tentativas de transferência.

Por fim, poderá ser analisada também a possibilidade da criação de grupos de redes, de modo a permitir que um arquivo possa ser transferido para duas ou mais redes de distribuição diferentes a partir de uma única solicitação da aplicação.

7 CRONOGRAMA

Etapa / Atividade	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out
Revisão Bibliográfica / Estado da Arte	X	X	X	X							
Projeto de Interação e Interface de Comunicação			X								
Desenho, Implementação e Codificação			X	X	X	X	X				
Testes					X	X	X				
Análise e Avaliação dos Testes						X	X				
Aplicação de Resultados							X				
Escrita da Monografia				X	X	X	X	X	X	X	X

Quadro 10 - Cronograma Planejado

Etapas Fixas:

- 15/Outubro a 05/Novembro de 2012 – entrega da monografia para banca
- 16/Novembro a 30/Novembro de 2012 – apresentação da monografia
- 11/Dezembro de 2012 – entrega da versão final da monografia

8 REFERÊNCIAS BIBLIOGRÁFICAS

AKAMAI. **Facts & Figures**. Disponível em: <http://www.akamai.com/html/about/facts_figures.html>. Acesso em 22 set. 2012.

ALMADA, Silvio. **Implementação do Google Global Cache (GGC)**. Associação Angolana de Provedores de Serviços de Internet, Luanda, 30 dez. 2009. Disponível em:<http://www.aapsi.og.ao/index.php?option=com_content&view=article&id=29:projecto-ggc&catid=7&Itemid=25>. Acesso em: 21 set. 2012.

ARLOW, J; NEUSTADT, I. **UML and the Unified Process**: Practical Object-Oriented, Analysis & Design. London: Addison Wesley, 2002.

BAILEY, JONATHAN. **Push Vc. Pull: The Tale of Two CDNs**. WhoIsHostingThis?, Londres, 30 jun. 2010. Disponível em: < http://www.whoishostingthis.com/blog/2010/06/30/cdns-push-vs-pull/>. Acesso em: 23 out. 2012.

BELSON, David; BERGQVIST, Svante; MÖLLER, Richard. **Q1 2012 State of the Internet Webinar**. Disponível em: <http://www.akamai.com/stateoftheinternet/>. Acesso em: 08 set. 2012.

CAMPOS, Marcelo Costa. **Interoperabilidade entre sistemas distribuídos utilizando Web Services**. Disponível em: <http://www.inst-informatica.pt/servicos/informacao-e-documentacao/biblioteca-digital/arquitectura-e-desenvolvimento-de-aplicacoes/interoperabilidade/interoperabilidade-entre-sistemas-distribuidos>. Acesso em: 01 dez. 2011.

CAULKINS, Jason. **Understanding the Nature of the "Cache Hit Ratio Curve"**, Part 1. Disponível em: <http://www.dataram.com/blog/?p=112>. Acesso em: 14 set. 2012.

COMITÊ GESTOR DA INTERNET NO BRASIL. **Dimensões e características da Web brasileira. Um estudo do .gov.br**. 2010. Disponível em: <http://cgi.br/publicacoes/pesquisas/govbr/cgibr-nicbr-censoweb-govbr-2010.pdf>. Acesso em: 01 dez. 2011.

COUTINHO et al. **Estratégias de balanceamento de carga em servidores Web transacionais**. Disponível em:<http://ufmg.academia.edu/BrunoCoutinho/Papers/602629/Estrategias_de_Balanceamento_de_Carga_em_Servidores_Web_Transacionais>. Acesso em: 01 dez. 2011.

DO, Cuong. **YouTube Scalability**. [s.l.]: YouTube, 2007. Disponível em: < http://www.youtube.com/watch?v=w5WVu624fY8>. Acesso em: 12 set. 2012.

DRUPAL and Varnish, a Quick Intro. Disponível em: < http://deglos.com/blog/2010/09/22/drupal-and-varnish-quick-intro>. Acesso em 13 set. 2012.

GOOGLE Global Cache. Disponível em: <http://ggcadmin.google.com/ggc>. Acesso em: 12 set. 2012.

GUIZZO, Érico. **Internet: o que, o que oferece, como conectar-se**. [s.l.] : Ática, 1999.

HERSMAN, Erik. **Google Kenya and the Google Global Cache**. White African, Quênia, 04 jul. 2008. Disponível em: <http://whiteafrican.com/2008/07/04/google-kenya-and-the-google-global-cache/>. Acesso em: 21 set. 2012.

INSTITUTO BRASILEIRO DE OPINIÃO PÚBLICA E ESTATÍSTICA. **Acesso à Internet no Brasil chega a 83,4 milhões de pessoas**. 2012. Disponível em: <http://www.ibope.com.br/pt-br/noticias/Paginas/Acesso-a-internet-no-Brasil-chega-a-83-milhoes-de-pessoas.aspx>. Acesso em: 07 set. 2012.

INSTITUTO DE PESQUISA ECONÔMICA APLICADA. **Panorama da Comunicação e das telecomunicações no Brasil**. Brasília: IPEA, 2010. v. 2.

INTEL. **What happens in an Internet minute?** Disponível em: < http://www.intel.com/content/www/us/en/communications/internet-minute-infographic.html>. Acesso em: 12 set. 2012.

JENSEN, JOE. **What type of Web Hosting is best for you?** Disponível em: <http://networkbits.net/web/web-hosting-types/>. Acesso em: 14 set. 2012.

JUNIOR, Massilon; HUMBERTO, Luis; RODRIGUES, Iran Miranda. **Problemas de performance em servidores Web muito acessados. Escalabilidade e portabilidade**. Disponível em: <http://www.di.ufpe.br/~flash/resultados/eventos/workshopais972/servidoresweb/servweb.html>. Acesso em: 01 dez. 2011.

KEPLER, João. **A revolução do conteúdo colaborativo**. 2011. Disponível em: <http://www.ojornalweb.com/2011/10/26/a-revolucao-do-conteudo-colaborativo/>. Acesso em: 01 dez. 2011.

KUNG, Fabio. **O novo desafio dos servidores Web**. 2006. Disponível em: <http://blog.caelum.com.br/servidores-web-e-nio/>. Acesso em 01 dez. 2011.

KUROSE, James F.; ROSS, Keith W. **Computer Networking: A Top-Down Approach**. 5. Ed. Boston: Addison Wesley, 2009.

LOBATO, Elvira. **Banda larga no Brasil é mais cara e pior, aponta estudo**. FOLHA DE SÃO PAULO, São Paulo, 29 mar. 2011. Disponível em: <http://www1.folha.uol.com.br/mercado/895270-banda-larga-no-brasil-e-mais-cara-e-pior-aponta-estudo.shtml>. Acesso em: 01 dez. 2011.

NYGREN, Erik; SITARAMAN, Ramesh K.; SUN, Jennifer. **The Akamai Network: A Platform for High-Performance Internet Aplications**. ACM SIGOPS Operating Systems Review, vol. 44, n. 3, jul. 2010. Disponível em: < http://www.akamai.com/dl/technical_publications/network_overview_osr.pdf>. Acesso em: 24 set. 2012.

PATRIZIO, Andy. **Controle dos Custos Operacionais em um Data Center**. Disponível em: http://content.dell.com/br/pt/corp/d/large-business/rein-data-center-operational-costs. Acesso em: 12 set. 2012.

PHP.NET. **Javascript Object Notation**. Disponível em: < http://www.php.net/manual/pt_BR/book.json.php>. Acesso em: 01 out. 2012.

SCOTT, Kendall. **O processo unificado explicado.** Porto Alegre: Bookman, 2003.

SOUDERS, Steve. **High Performance Web Sites**. California (EUA): O'Reilly, 2007.

SPAGNUOLO, Sérgio. **Número de usuários ativos na Internet cresce 12% no Brasil**. 2011. Disponível em: <http://br.reuters.com/article/internetNews/idBRSPE7AR08020111128>. Acesso em: 01 dez. 2011.

TANENBAUM, Andrew S.; VAN STEEN, Maarten. **Sistemas distribuídos:** princípios e paradigmas. 2. ed. São Paulo, SP: Pearson Prentice Hall, 2007. 402 p.

VARNISH CACHE. **About**. Disponível em: <https://www.varnish-cache.org/about>. Acesso em: 13 set. 2012.

W3SCHOOLS.COM. **JSON Tutorial**. Disponível em: <http://www.w3schools.com/json/default.asp>. Acesso em: 01 out. 2012.

W3TECHS.com. **Usage of server-side programming languages for websites**. Disponível em: <http://w3techs.com/technologies/overview/programming_language/all>. Acesso em: 01 out. 2012.

WASLAWICK, Raul S. **Análise e Projeto de Sistemas de Informação Orientados a Objetos**. São Paulo: Campus Elsevier, 2004.

WATANABE, Cláudia S. **Introdução ao Cache Web**. Disponível em: <http://www.rnp.br/newsgen/0003/cache.html>. Acesso em 13 set. 2012.