ORPIS: um Modelo de Consistência de Conteúdo Replicado em Servidores Web Distribuídos

Cristiano Cachapuz e Lima*
Universidade da Região da Campanha
cristiano@urcamp.tche.br

Adenauer Correa Yamin†
Universidade Católica de Pelotas
adenauer@ucpel.tche.br

Cláudio Fernando Resin Geyer
Universidade Federal do Rio Grande do Sul
geyer@inf.ufrgs.br

Resumo

O surgimento de novas aplicações que utilizam o protocolo HTTP nas suas transações e a crescente popularidade da World Wide Web *(WWW) estimularam pesquisas pelo aumento do desempenho de servidores* Web. *Para tal, a alternativa proposta neste trabalho é utilizar um* Web cluster, *isso é, um conjunto de servidores* Web *distribuídos que espalham a carga de requisições entre vários computadores, atuando como um só associado a uma estratégia de replicação de conteúdo. Um dos problemas centrais a ser resolvido em* Web clusters *é como manter a consistência das réplicas de conteúdo entre os equipamentos envolvidos. Este trabalho aborda esse problema e tem por objetivo propor um modelo de manutenção da consistência de conteúdo em servidores* Web *distribuídos com características de transparência e autonomia, denominado* One Replication Protocol for Internet Servers *(ORPIS).*

1. Introdução

Desde sua introdução a *World-Wide Web* (ou simplesmente *Web*) evoluiu de um modelo simples de arquitetura cliente-servidor para um modelo distribuído sofisticado. Essa evolução foi ocasionada pelo crescimento exponencial de diversas aplicações anteriormente inexistentes: bibliotecas digitais, educação à distância, áudio e vídeo sob demanda e comércio eletrônico. Essas aplicações ocasionaram um aumento enorme do tráfego na Internet. Alguns sítios *Web* populares podem receber milhões de acessos por dia, resultando em tempos de resposta altos. A sobrecarga pode acontecer devido à saturação da largura de banda da rede, do processador ou da memória principal do servidor *Web* ou, até mesmo, da redução da capacidade de conexão do servidor à rede. A lista de pedidos TCP (*Transfer Control Protocol*) do servidor pode ficar sobrecarregada, ocasionando uma degradação no desempenho [1]. Dessa forma, soluções são necessárias para atender os pedidos de maneira eficiente e com desempenho admissível.

Uma das soluções possíveis é o uso de servidores *Web* distribuídos. Essa abordagem espalha a carga de pedidos HTTP (*HyperText Transfer Protocol*) entre vários computadores conectados atuando como um só com o objetivo de proporcionar um melhor desempenho: um *Web cluster*. Um servidor *Web* distribuído exporta um nome lógico único e o informa para o mundo externo. Cada um dos componentes do *cluster* possui uma réplica do conteúdo a ser oferecido por esse servidor *Web*. Esse *cluster* pode estar instalado fisicamente em um mesmo local ou distribuído geograficamente em diferentes pontos da Internet.

Um problema fundamental em sistemas distribuídos é a manutenção da consistência das réplicas. As cópias do conteúdo *Web* devem permanecer consistentes, para que todos os servidores replicados possam coordenar suas leituras e escritas [6]. Um dos aspectos a serem considerados nessa abordagem é a política de sincronização da atualização das réplicas dos dados no *Web cluster*. Quando da atualização das páginas de um dos servidores, os outros componentes do *cluster* devem refletir exatamente o mesmo conteúdo.

Esse trabalho tem por objetivo apresentar um modelo de gerenciamento da consistência de conteúdo em servidores *Web* distribuídos – *Web clusters*, com características de transparência e autonomia. Essa ferramenta foi denominada ORPIS (*One Replication Protocol for Internet Servers*).

A ferramenta ORPIS pretende ser uma solução extremamente portável e que pode ser livremente usada por administradores de sítios *Web* distribuídos – os protótipos da ferramenta originada a partir do modelo foram implementa-

*Mestre pela Universidade Federal do Rio Grande do Sul
†Doutorando na Universidade Federal do Rio Grande do Sul

dos utilizando ferramentas de software livre, criando uma solução economicamente viável e acessível a qualquer interessado. O modelo também pode ser usado em ambientes de computação distribuída com estrutura não fixa (ex: *grids* computacionais) já que permite a configuração de diversos componentes de forma dinâmica.

A seção dois deste trabalho apresenta conceitos envolvendo o tema replicação de conteúdo em servidores *Web* distribuídos, suas características e problemas inerentes à sua implementação e gerenciamento. Essa seção também apresenta os trabalhos relacionados à manutenção da consistência de réplicas. A proposição do modelo ORPIS é feita na terceira seção, onde são discutidas sua arquitetura e suas principais características. A seção quatro descreve a implementação feita no protótipo do modelo ORPIS. A quinta seção apresenta os trabalhos relacionados. A seção seis apresenta as conclusões e os possíveis trabalhos futuros.

2. Servidores web distribuídos

O núcleo desta proposta é colocar diversos servidores a trabalhar de forma cooperativa, deste modo, o total de requisições HTTP pode ser distribuída entre estes equipamentos interconectados. Um mecanismo de alocação com base em uma polítca previdamente definida, seleciona o servidor mais adequado a ser utilizado no momento. Devido ao fato de que os servidores tornaram-se menores e mais baratos e também com o aumento do desempenho das redes de comunicação, hoje em dia é possível agrupá-los de modo que pareçam aos usuários apenas um único sistema. Esse grupo de servidores é conhecido como um agregado (*cluster*). Segundo [11], um "*cluster* de estações" é constituído de um conjunto de computadores, denominados nós, interconectados por uma rede de alta velocidade, e classificados como multicomputadores.

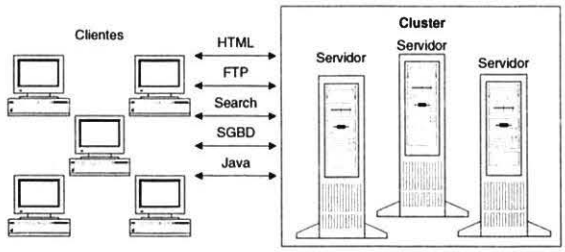

Figura 1. Arquitetura de um *Web cluster*, traduzido de [8]

O agregado (figura 1) funciona para o cliente como um sistema único, no qual fica potencializado o desempenho. Em um sistema de alto desempenho do tipo *mainframe*, o administrador precisa gerenciar apenas uma máquina. No agregado, o administrador deve instalar software em vários servidores, configurar e operar cada um deles, e garantir que os dados, tais como o conteúdo do sítio *Web* seja o mesmo em cada sistema. Isso pode ser um desafio e é a razão primária pela qual as tecnologias de replicação de dados estão se tornando importantes para o mercado [9].

Existem três tipos de soluções de agregados: de rede, de dados e de processos. Agregados de redes gerenciam a interface da rede ao agregado. Tipicamente, esses elementos incluem sobreposição de IP, *heartbeat*, balanceamento de carga e várias formas de monitoração de serviços e dispositivos. *Heartbeat* é a função de verificar se um determinado componente do agregado ainda se encontra funcionando, através de envio e retorno de pacotes [10]. Agregados de dados gerenciam o acesso a recursos de armazenamento compartilhados do agregado. Agregados de processos permitem que vários nós participem no processamento de uma única aplicação.

Servidores *Web* distribuídos necessitam de mecanismos de decisão para definir qual servidor replicado que deverá responder às requisições dos clientes. A situação ideal é descobrir o servidor replicado mais apropriado com o qual o cliente em questão possa se comunicar com o melhor desempenho possível. Essa otimização é uma política normalmente estabelecida por proximidade, mas também pode ser baseada em outros critérios, como, por exemplo, a carga de pedidos. As formas mais usuais existentes são [3]:

- ligações de navegação – a maneira mais simples de comunicação entre clientes e réplicas. Esse mecanismo usa URLs individuais dentro das páginas que apontam para os servidores replicados. O cliente seleciona manualmente a ligação do servidor replicado que deseja usar;

- redirecionamento de HTTP – os clientes são redirecionados para um servidor replicado ótimo através do uso dos códigos de resposta do protocolo HTTP: "302 *Found*" ou "307 *Temporary Redirect*". O cliente estabelece comunicação com um dos servidores replicados, e este pode escolher aceitar o serviço ou redirecionar o cliente novamente;

- redirecionamento através de DNS – o *Domain Name Service* (DNS) oferece um sofisticado mecanismo de comunicação entre clientes e réplicas. Isso é possível pelos servidores DNS que ordenam os endereços IP resolvidos, baseado em políticas de serviço. Quando um cliente converte o nome de um servidor, o servidor DNS ordena os endereços IP dos servidores replicados, iniciando pela melhor réplica e terminando na réplica menos apropriada.

2.1. Consistência em servidores *Web* distribuídos

Um dos aspectos centrais a serem considerados em servidores *Web* distribuídos é a política de manutenção da consistência das réplicas do conteúdo. Quando da atualização das páginas de um dos servidores, os outros componentes do *cluster* devem refletir o mesmo conteúdo.

Ao se acrescentar servidores *Web* distribuídos, vem à tona a seguinte questão: deve ser replicado todo o conteúdo, isto é, manter várias cópias dos mesmos dados ou simplesmente particionar o conteúdo entre diversos servidores. Ao se escolher a primeira opção, deve-se estar ciente da preocupação adicional em manter esse conteúdo consistente entre os diferentes servidores.

Sincronização é o processo de assegurar que os dados estejam consistentes entre todos os servidores no *cluster*. Em alguns sistemas, pode não ser necessário que os dados estejam sincronizados em tempo real. Nesse caso, um procedimento que replica e sincroniza os arquivos a cada hora pode ser o suficiente.

A arquitetura cliente-servidor da *Web* não tem por foco manter os usuários atualizados com as últimas versões das páginas. Os usuários têm que pressionar o botão "atualizar" de seus navegadores para garantir que têm as últimas informações, e os navegadores são forçados a comunicarem-se com os servidores para confirmar se a cópia local de um recurso ainda é válida. Portanto, a arquitetura de sistema da *Web* permite uma semântica de consistência mais "relaxada" em função da disponibilidade e do desempenho.

Nos ambientes de *Web cluster*, o objetivo principal da replicação é permitir o balanceamento de carga entre os componentes, além de permitir tolerância a falhas quando um dos servidores deixa de funcionar. Nesta perspectiva, é desejável que os detalhes da replicação sejam escondidos dos usuários finais, proporcionando transparência de localização, isto é, o fato de o usuário usar uma réplica sem perceber [13].

2.2. Replicação

A replicação é uma técnica de redundância empregada para melhorar o nível de disponibilidade em sistemas distribuídos. A replicação aumenta a disponibilidade dos dados e processos e o desempenho geral do sistema, já que o tempo para atender a uma requisição será menor se uma réplica mais adequada (menos ocupada ou com menor custo de acesso) for usada.

Em sistemas baseados no modelo cliente-servidor, um servidor único pode atender a vários clientes, e um aumento da carga de trabalho no servidor pode acarretar tempos de resposta elevados. Em tais circunstâncias, replicar os dados ou servidores pode aumentar o desempenho.

O mínimo necessário para que haja replicação é que as diferentes cópias de um mesmo objeto residam em máquinas independentes umas das outras. Isso permite que se atinjam alguns requisitos necessários à tolerância a falhas: a disponibilidade de uma réplica não pode ser afetada pela disponibilidade das demais réplicas.

A redundância é normalmente introduzida pela replicação de componentes ou de serviços. Apesar de a replicação ser uma idéia intuitiva, rapidamente compreensível, sua implementação é complexa. Replicar um serviço em sistemas distribuídos exige que cada réplica do serviço mantenha um estado consistente. Essa consistência é garantida através de um protocolo de replicação específico [5].

Replicação tem sido pesquisada em diversas áreas, especialmente em sistemas distribuídos (especialmente na área de tolerância a falhas) e em bancos de dados por razões de desempenho. Nessas duas áreas, as técnicas e mecanismos usados são similares. Porém alguns mecanismos conceitualmente idênticos, na prática, tornam-se muito diferentes [14].

2.2.1. Replicação de conteúdo *Web*.

Pode-se distinguir três formas de replicação de conteúdo em servidores *Web* distribuídos [3]:

- replicação em lotes – o servidor replicado a ser atualizado é quem inicia a comunicação com um servidor original. A comunicação é estabelecida em intervalos baseados em transações enfileiradas que são agendadas para processamento posterior. As políticas de agendamento variam, mas, normalmente, ocorrem em um determinado intervalo de tempo. Uma vez que a comunicação é estabelecida, conjuntos de dados são copiados para o servidor replicado;

- replicação por demanda – os servidores replicados obtêm o conteúdo quando necessário devido à demanda do cliente. Quando um cliente solicita um recurso que não esteja no conjunto de dados do servidor replicado, é feita uma tentativa de atender ao pedido, buscando o recurso do servidor original e retornando-o para o cliente que o solicitou;

- replicação sincronizada – os servidores replicados cooperam usando estratégias sincronizadas e protocolos especializados de réplica para manter os conjuntos de dados replicados coerentes. Estratégias de sincronização variam desde fortemente coerentes (alguns poucos minutos) até fracamente coerentes (algumas horas). As atualizações ocorrem entre as réplicas baseadas nas restrições de tempo do modelo de coerência empregado e são feitas, geralmente, apenas através dos dados que foram modificados.

3. ORPIS: concepção e modelagem

ORPIS é um modelo de replicação de conteúdo para ser utilizado em servidores *Web* geograficamente distribuídos (*clusters Web* distribuídos), os quais têm o objetivo de oferecer mais desempenho no atendimento às requisições, principalmente em situações de sobrecarga. Um módulo anexado ao servidor *Web* encarrega-se de desviar os pedidos para o servidor mais adequado para atender às requisições. O conteúdo (objetos *Web*) é replicado entre os diversos servidores componentes do *Web cluster* distribuído ("comunidade ORPIS"). Um módulo encarrega-se de detectar as atualizações feitas nos objetos *Web*, baseando-se em políticas de intervalos de tempo. Tendo detectado as atualizações feitas, inicia-se o processo de sincronização entre as réplicas.

Um dos objetivos do modelo ORPIS é ser transparente. Transparência é a capacidade de esconder do usuário e do programador da aplicação a separação dos componentes em um sistema distribuído, para que assim o sistema seja percebido como algo inteiro e não como um conjunto de componentes independentes [4].

O modelo ORPIS é baseado na forma de replicação descrita em [14] como multiprimária, pois permite que o cliente envie as requisições para qualquer membro do *cluster*. Essa forma de replicação permite que qualquer uma das réplicas existentes possa ser eleita como a primária. As atualizações no conteúdo do sítio Web também podem ser feitas em qualquer servidor distribuído. A não existência de um servidor central permite essa liberdade.

O modelo funciona com o conceito de **proprietários** de páginas. O conteúdo total do *Web cluster* é replicado em todos os componentes do *cluster*, porém a atualização de um objeto nas páginas somente é autorizada pelo seu proprietário, tendo ele feito a atualização em qualquer uma das réplicas. Não é permitido que o proprietário de uma página atualize páginas pertencentes a outro proprietário. Esse pressuposto impede a existência de conflitos entre as réplicas.

O servidor *Web* que tem instalado o componente Gerente de Replicação do modelo ORPIS, ao detectar uma atualização em seu sistema de arquivos, passa, então, a ser o primário, responsável pela atualização dos demais membros, enviando mensagens de atualizações através do mecanismo de transporte de dados. A troca de informações a respeito de atualizações do conteúdo é realizada entre as réplicas através de trocas de mensagens do servidor primário com as réplicas *backups*, até que todos os servidores estejam consistentes. Os servidores primários são chamados de nós propagadores, e os servidores *backups* são denominados nós receptores.

No modelo ORPIS, o conteúdo é totalmente replicado entre os componentes da comunidade, cada réplica funcionando como um "espelho" das demais.

O modelo ORPIS usa uma estratégia de propagação otimista porque não existe sincronismo no envio das atualizações para as réplicas.

3.1. Componentes do modelo

A figura 2 apresenta os principais componentes do modelo ORPIS. Como se pode notar, os componentes do modelo ORPIS são executados no nível do usuário, sem necessidade de reconfiguração do sistema operacional ou do servidor *Web*.

As seções seguintes descrevem os vários módulos que compõem o modelo ORPIS.

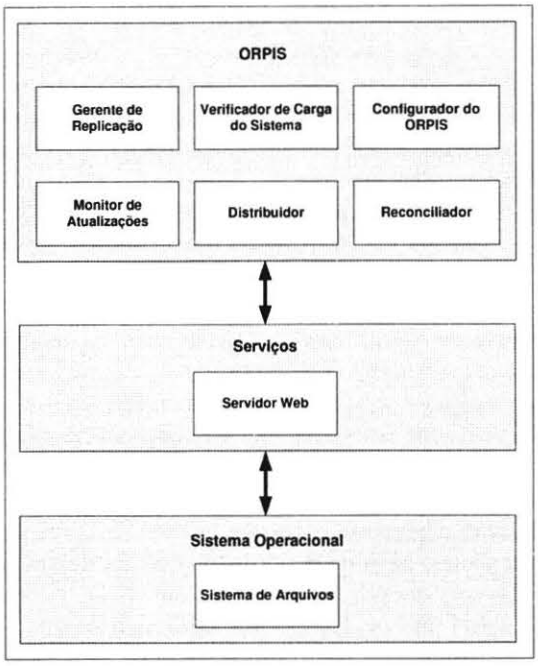

Figura 2. Componentes do modelo ORPIS

O **Verificador de Carga do Sistema** analisa a fila de requisições e calcula a carga de requisições que atualmente está sendo imposta ao servidor em um determinado momento. Sua concepção foi baseada em [7] e [1].

Uma forma simples de decidir se o servidor está sobrecarregado é através da monitoração do tempo de resposta das requisições ao servidor. Esse tempo de resposta é obtido através do envio de requisições HTTP ao servidor e da medição do tempo de resposta às mesmas. A maneira que permite uma melhor medição é o envio das requisições HTTP a partir de uma sessão no próprio computador que possui o servidor Web. Dessa forma, não existe o tempo necessário para que os pacotes circulem pela rede. Um tempo de resposta pequeno pode indicar que o servidor não está sobrecarregado.

De acordo com [1], deve-se encontrar um ponto de *threshold*, que é estabelecido através de um acordo de QoS (*Quality of Service*). Na ausência desse tipo de medida, o ponto de *threshold* (T) é obtido através do tamanho máximo da fila de requisições (Q) e do tempo médio de resposta (S). Se for estabelecido, por exemplo, que 75% de ocupação da fila é um indicativo de sobrecarga, o Verificador de Carga do Sistema deve ser ajustado para considerar o servidor sobrecarregado quando a fila de requisições estiver em 75% de sua capacidade ($T = 0.75QS$). Pode-se obter o parâmetro Tempo Médio de Resposta através de testes de desempenho do servidor.

Esse módulo somente é acionado pelo Gerente de Replicação quando do início do processo de sincronização, que o utiliza como forma de detectar a carga de trabalho que está sendo imposta no servidor *Web*.

O **Monitor de Atualizações** faz varredura passiva no sistema de arquivos indicado e tem como finalidade o monitoramento de inclusões, exclusões e modificações feitas em diretórios, arquivos ou sistemas de arquivos completos. Varredura passiva identifica objetos atualizados através da comparação do estado atual de um dado replicado e um estado conhecido anteriormente [6].

O Monitor de Atualizações possui dois modos de execução, indicados através de parâmetros quando de sua invocação:

- criação da Base de Dados do Sistema de Arquivos – varre o sistema de arquivos e diretórios indicado em um arquivo de configuração e constrói a Base de Dados do Sistema de Arquivos;
- checagem da Base de Dados do Sistema de Arquivos com o Sistema de Arquivos – neste modo, o Monitor de atualizações varre o Sistema de Arquivos e compara as informações obtidas com a Base de Dados do Sistema de Arquivos, criada na última vez em que o Monitor de atualizações foi executado.

O processo de sincronização é iniciado pelo **Gerente de Replicação**, em momentos pré-determinados, ou manualmente, através da requisição do usuário. Ele tem a função de acionar o Verificador de Carga. Se o servidor não estiver sobrecarregado, o Gerente de Replicação inicia o Monitor de Atualizações, que faz a varredura do sistema de arquivos. Se há atualizações a serem propagadas, o Gerente de Replicação tem o papel de incluí-las na Fila de Objetos a Propagar.

O módulo **Distribuidor** é responsável pelo contato do nó propagador com o módulo Reconciliador no nó remoto. No momento da sincronização, o Distribuidor do nó propagador estabelece a comunicação com o Reconciliador do nó remoto. O Distribuidor envia a lista de atualizações para o Reconciliador do nó receptor.

Reconciliador é o módulo responsável pela comunicação nos nós receptores. Tem por objetivo enviar respostas de conexão e recepção de listas de atualizações, feitas pelo módulo Distribuidor dos nós propagadores.

O módulo **Configurador do ORPIS** é uma aplicação autônoma que permite incluir, excluir e alterar endereços de servidores *Web* que fazem parte da comunidade ORPIS. Suas funções permitem definir: os servidores *Web* envolvidos, o tempo de operação do reconciliador (operações de sincronização) e quais diretórios são replicados e em quais máquinas do *Web cluster* ocorreria a replicação dos mesmos.

3.2. Relacionamento entre os componentes

A figura 3 apresenta o diagrama de relacionamentos entre os módulos componentes do modelo ORPIS funcionando em um nó propagador.

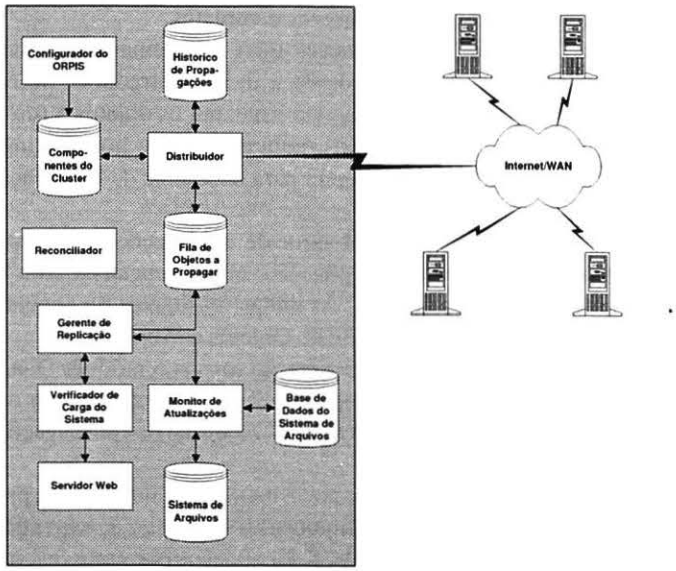

Figura 3. Arquitetura do modelo ORPIS em um nó propagador

3.3. Funcionalidade do modelo – nó propagador (algoritmo)

Esta seção apresenta a descrição do funcionamento do modelo ORPIS em um servidor que inicia o processo de manutenção da consistência, o denominado nó propagador (primário).

O Gerente de Replicação é acionado em intervalos regulares, de acordo com uma programação do sistema *cron* da máquina onde está instalado. O Gerente de Replicação inicia o Verificador de Carga do Sistema.

Para determinar a carga atual do servidor *Web*, o Verificador de Carga do Sistema envia cinqüenta requisições HTTP para o servidor *Web*. Essas cinqüenta requisições são cronometradas, estabelecendo-se uma média dos tempos obtidos. Se o servidor *Web* obtiver uma média elevada, o Verificador de Carga do Sistema declara o servidor como sobrecarregado e envia um valor verdadeiro (*true*) ao Gerente de Replicação indicando que, no momento, não é possível iniciar o processo de sincronização de servidores. O processo de sincronização não continua.

Se o servidor *Web* não estiver sobrecarregado, o Verificador de Carga do Sistema retorna um valor falso (*false*) ao Gerente de Replicação, indicando que o servidor *Web* não está sobrecarregado e que o processo de sincronização pode continuar sua execução.

Então o Gerente de Replicação dispara o Monitor de atualizações, que tem como finalidade o monitoramento de inclusões, exclusões e modificações feitas em diretórios, arquivos ou sistemas de arquivos completos.

O Monitor de atualizações varre o sistema de arquivos em busca de alterações desde a última varredura. Se encontrar atualizações, gera uma lista com os objetos a serem propagados para as demais réplicas. Se não houve atualizações, envia uma mensagem para o Gerente de Replicação informando-o do fato.

A lista gerada pelo Monitor de atualizações é manipulada pelo Gerente de Replicação, que a concatena com os componentes do *cluster*. As linhas resultantes da concatenação são incluídas na Fila de Objetos a Propagar.

Então o Gerente de Replicação invoca o módulo Distribuidor. Esse módulo é responsável pela propagação dos objetos nos nós propagadores e pela recepção das propagações nos nós receptores.

O módulo Distribuidor envia mensagens de início de propagação para todos os componentes do *cluster* e, logo após, aguarda as respostas de ACK (aceitação) por parte dos nós receptores. Se algum nó não responder em um determinado intervalo de tempo, o Distribuidor atualiza a base de dados Componentes do *Cluster*, informando que aquele componente do *cluster* não foi contatado. O Distribuidor tenta contato com esse nó novamente por mais três vezes. Após essas três tentativas, esse componente é classificado como indisponível, e a base de dados Componentes do *Cluster* é atualizada.

Após ter recebido as respostas de ACK dos nós receptores, o Distribuidor passa a propagar todas as referências aos objetos a serem replicados, a partir da Fila de Objetos a Propagar. A cada resposta de ACK dos nós receptores, o Distribuidor apaga a linha com a referência correspondente ao objeto da Fila de Objetos a Propagar.

O Distribuidor mantém uma base de dados Histórico de Propagações, com o objetivo de manter um histórico de todas as operações realizadas. Esse Histórico de Propagações é utilizado com o objetivo de manter a consistência de um servidor que porventura venha a ficar demasiado tempo fora da comunidade ORPIS, por falta de conectividade da rede ou por falhas de software ou hardware.

3.4. Funcionalidade do modelo – nós receptores (algoritmo)

Esta seção apresenta a descrição do funcionamento do ORPIS em um servidor receptor (*backup*), que é atualizado a partir do servidor propagador (primário).

O módulo Reconciliador de um nó receptor recebe a mensagem de que um nó propagador está iniciando o processo de propagação. Imediatamente, o Reconciliador do nó receptor suspende qualquer atividade de propagação, informando ao Gerente de Replicação do nó receptor que não inicie qualquer processo de atualização até que o processo que foi iniciado pelo nó propagador esteja encerrado. O Reconciliador envia uma resposta ao Distribuidor do nó propagador, que solicitou o início da propagação, informando que está pronto para receber as propagações. Somente após o processo de sincronização estar encerrado, o Reconciliador envia uma mensagem ao Gerente de Replicação para que retome suas atividades de propagação aos demais nós.

O Reconciliador recebe várias mensagens contendo as informações necessárias para que seu sistema de arquivos seja sincronizado com o do nó propagador (referências). Cada uma dessas mensagens é armazenada na Fila de Objetos a Propagar. O Distribuidor do nó propagador envia uma mensagem final informando que não há mais objetos a serem propagados.

Então o Reconciliador chama o Gerente de Replicação, que, por sua vez, chama o Verificador de Carga do Sistema. Caso o Verificador de Carga do Sistema informe que o servidor *Web* está sobrecarregado, uma nova tentativa de busca dos objetos remotos é agendada para alguns momentos depois.

No caso de o Verificador de Carga do Sistema indicar que o servidor *Web* não está sobrecarregado, o Reconciliador passa a ler a Fila de Objetos a Propagar e envia uma requisição via protocolo FTP ao nó propagador, requisitando o objeto. O Reconciliador grava no seu sistema de arquivos o objeto enviado. Cada linha da Fila de Objetos a Propagar é processada pelo Reconciliador e removida. A base de dados Histórico de Propagações mantém um *log* de todas as atualizações feitas nos objetos do sistema de arquivos, tanto as feitas a partir do próprio *host*, quanto as feitas por outros componentes do *cluster*.

Após o processo de consistência estar encerrado, o Gerente de Replicação invoca o Monitor de atualizações no modo Criação da Base de Dados do Sistema de Arquivos.

4. Implementação

O protótipo do modelo ORPIS foi implementado no laboratório de computação da URCAMP, em Bagé. Os equipamentos escolhidos foram dois microcomputadores K6-II, de 500 MHz, ambos com 64 MB de RAM. Como plataforma de sistema operacional, optou-se pelo GNU Linux, com as seguintes características: distribuição Conectiva Linux 6, *kernel* versão 2.2.17-14cl. O servidor *Web* utilizado foi o Apache, versão 1.3.14-6cl. A forma de comunicação adotada entre os módulos do modelo ORPIS é através de *sockets*. O recurso de direcionamento de requisições HTTP foi obtido através da recompilação do servidor *Web* Apache, com a inclusão do módulo `mod_backhand` [12].

4.1. Classes

A classe **Configurador** foi concebida como uma aplicação que se beneficia do modelo *Common Gateway Interface* (CGI), com trechos executados no servidor e no cliente. Foi implementado através um programa escrito em *PHP: Hypertext Preprocessor* (PHP) e HTML, com acesso a banco de dados MySQL. A utilização do configurador do ORPIS deve ser iniciada através da execução de um navegador *Web*. Deve-se informar o endereço *http://nome_do_servidor/orpis/*. A primeira tela permite a inclusão de novos servidores, modificação ou remoção de servidores existentes, e ainda existe uma opção de visualizar todos os servidores cadastrados. Os requisitos para instalação do protótipo do configurador ORPIS são os seguintes componentes configurados em um servidor Linux: servidor *Web* Apache (a partir da versão 1.3), servidor de banco de dados MySQL (a partir da versão 3.23) e módulos do interpretador de scripts PHP (a partir da versão 4.04). A figura 4 apresenta a tela de inclusão de novos servidores. A classe **Distribuidor** é responsável por informar aos diversos componentes do *cluster* as propagações que devem ser efetuadas. Ela tem a responsabilidade de conectar um nodo propagador a um nodo receptor e informá-lo sobre quais objetos deverão ser sincronizados.A classe **Gerente** é responsável por gerenciar o processo de propagação do conteúdo para os demais componentes do *cluster*. A classe **Reconciliador** é encarregada de efetuar a comunicação nos nodos receptores. Tem por objetivo enviar respostas de conexão e recepção de listas de atualizações, feitas pela classe Distribuidor do nodo propagador. A comunicação entre essas duas classes é implementada através de mensagens via *sockets*. A classe **Verificador** faz o teste de carga no servidor *Web* do componente do *cluster*. A classe **Monitor** tem a função de fazer a varredura passiva do sistema de arquivos, buscando atualizações que tenham sido feitas. Essa verificação é feita através da comparação dos objetos armazenados no sistema de arquivos com os que estão armazenados na base de dados do sistema de arquivos.

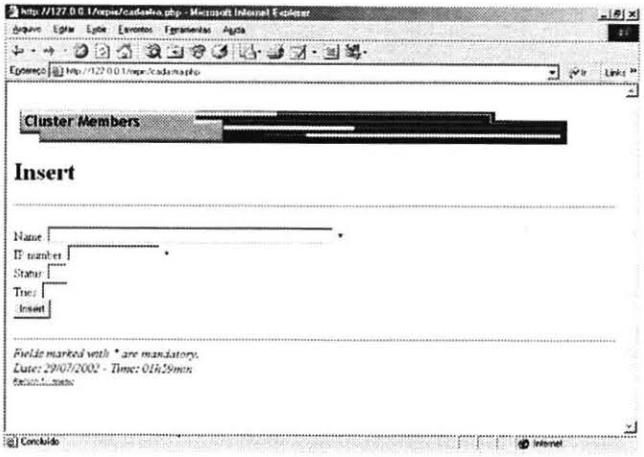

Figura 4. Tela da opção Inclusão do configurador do ORPIS

5. Trabalhos relacionados

Durante o desenvolvimento deste trabalho, foi possível identificar e estudar alguns trabalhos e ferramentas cujos objetivos encontram-se dentro do escopo desta pesquisa, isto é, ferramentas de replicação de conteúdo:

- *Rdist* – seu objetivo é manter cópias idênticas de arquivos em várias estações através dos protocolos remote shell (rsh) ou rcmd. Rdist pode ser obtida no sítio *http://www.magnicomp.com/rdist/*;
- *Rsync* – pretende ser uma substituta mais rápida e mais flexível ao comando rcp. A ferramenta é baseada no algoritmo rsync, usado para identificar os trechos que são diferentes entre dois arquivos que foram atualizados independentemente, e pode ser obtida em *http://rsync.samba.org*;
- *Mirrordir* – faz o espelhamento de uma árvore de diretórios em todos os seus detalhes. Também implementa seus próprios *sockets* seguros para transferência de arquivos com encriptação forte. Esta ferramenta pode ser obtida em *http://mirrordir.sourceforge.net*.

6. Conclusão

O modelo ORPIS adota uma estratégia de propagação otimista porque não existe sincronismo no envio das atualizações para as réplicas. Outro ponto a ser salientado é a compatibilidade do modelo ORPIS aos servidores *Web* existentes, sem a necessidade de modificação em suas configurações. Essa característica também se aplica aos editores e atualizadores dos diretórios do servidor *Web*.

O modelo pode ser usado em ambientes de computação distribuída com estrutura não fixa (ex: *grids* computacionais) já que permite a configuração de diversos componentes de forma dinâmica.

O uso de software de código aberto no desenvolvimento do protótipo proporcionou um rápido acesso às ferramentas necessárias (sistema operacional, linguagens e gerenciador de banco de dados), com possibilidade de alteração nos códigos fonte como uma alternativa de customização.Os vários componentes do modelo ORPIS ainda estão em desenvolvimento e várias funcionalidades estão previstas para serem disponibilizadas. Como trabalhos futuros, destacar-se-ia:

- desenvolvimento de um módulo anexado ao servidor *Web* que desvie as requisições dos usuários para o servidor replicado que apresente as melhores condições de acesso ou que esteja mais disponível, baseado em métricas relativas ao desempenho da rede e à carga imposta no momento;
- aperfeiçoamentos no protótipo, através da otimização do código ou até mesmo geração de código nativo da plataforma de hardware, proporcionando um melhor desempenho;
- geração de carga de trabalho sintética, simulando o acesso simultâneo de vários clientes e medição do desempenho do servidor *Web*, com o objetivo de simular situações possíveis de acontecerem em um ambiente de produção.

Referências

[1] Tarek F. Abdelzaher and Nina Bhatti. Web content adaptation to improve server overload behavior. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1563–1577, 1999.

[2] Yair Amir. *Replication Using Group Communication Over a Partitioned Network*. PhD thesis, Hebrew University of Jerusalem, Jerusalem, 1995.

[3] I. Cooper, I. Melve, and G. Tomlinson. Internet web replication and caching taxonomy, 2000. 2000. Disponível em: <ftp://ftp.nordu.net/internet-drafts/draft-ietf-wrec-taxonomy-05.txt>. Acesso em: 20 ago. 2000.

[4] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed systems: concept and design*. Addison-Wesley, Harlow, 3 edition, 2001.

[5] Xavier Défago, Andre Schiper, and Nicole Sergent. Semi-passive replication. In *Symposium on Reliable Distributed Systems*, pages 43–50, West Lafayette, 1998. West Lafayette, IEEE.

[6] Todd Ekenstam, Charles Matheny, Peter L. Reiher, and Gerald J. Popek. The bengal database replication system. *Distributed and Parallel Databases*, 9(3):187–210, 2001.

[7] Patrick Killelea. *Web Performance Tuning*. O'Reilly, Newton, 1998.

[8] Daniel A. Menascé and Virgilio A. F. Almeida. *Capacity Planning for Web Performance: metrics, models, and methods*. Prentice-Hall, Englewood Cliffs, 1998.

[9] Polyserve. Data replication for high availability web server clusters, 2000. 2000. Disponível em: <http://www.polyserve.com/support/downloads/white_papers/data_replication_for_high_availability_web_servers.pdf>. Acesso em: 20 nov. 2000.

[10] Alan Robertson. Linux-HA heartbeat system design. In USENIX, editor, *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, October 10–14, 2000, Atlanta, Georgia, USA*, Berkeley, 2000. Berkeley, USENIX.

[11] L. M. Sato. Evolução e tendências da programação paralela. In *Escola Regional de Alto Desempenho (ERAD 2001)*, Gramado, 2001. Gramado, SBC.

[12] Theo Schlossnagle. mod_backhand: A load balancing module for the apache web server. In *ApacheCon 2000*, Orlando (Florida), 2000. APACHECON2000, Orlando, The Apache Software Foundation.

[13] A. S. Tanenbaum. *Distributed Operating Systems*. Prentice-Hall, Upper Saddle River, 1995.

[14] M. Wiesmann, F. Pedone, and A. Schiper. A systematic classification of replicated database protocols based on atomic broadcast. In *Proceedings of the* 3^{rd} *Europeean Research Seminar on Advances in Distributed Systems (ERSADS'99)*, Madeira Island (Portugal), 1999. BROADCAST Esprit WG 22455, Madeira Island, Esprit WG 22455.