Balanceamento de Carga nas Redes Inteligentes Distribuídas

N. L. O. Bodart
Departamento de Engenharia da UVV
norminda@uvv.br

E. R. de C. Durães
Departamento de Eng. Elétrica da UNIEST
elisrcd@terra.com.br

R. B. Soares
Departamento de Eng. Elétrica da UFES
rosane@ele.ufes.br

A. S. Garcia
Departamento de Informática da UFES
anilton@inf.ufes.br

Resumo

Este artigo apresenta uma proposta para viabilizar a implementação de serviços banda larga nas Redes Inteligentes distribuídas, com a premissa de que a Arquitetura de Serviços é acessada por meio de um Gateway. Propõe-se a inclusão de um Serviço de Balanceamento de Carga Adaptativo por Demanda, recentemente implementado em um ORB compatível com o CORBA, The ACE ORB (TAO), para otimizar a escalabilidade e a vazão nos servidores e prover mais segurança às aplicações distribuídas.

1. Introdução

As últimas décadas têm revelado a presença crescente de sistemas computacionais distribuídos no ambiente das telecomunicações. No tocante a oferta de serviços baseados no conceito de Redes Inteligentes (RI), as abordagens da teoria de informação mostram a tendência de evolução da camada de serviços, combinando diversos recursos distribuídos.

O processo de crescimento de aplicativos e de recursos para a implementação de sistemas distribuídos vem estimulando, conseqüentemente, estudos voltados para o balanceamento de carga (BC) nas redes envolvidas com a oferta de serviços para os usuários finais. Dependendo do tipo de aplicação pode-se, por exemplo, ser necessário balancear o número de *threads* em uma arquitetura tipo "*three-tier*", a utilização dos processadores, a carga nas interfaces de uma rede, etc. As propostas de BC visam atender vários itens que estão se tornando cada vez mais importantes nas redes distribuídas, quais sejam: Aumentar a escalabilidade dos servidores; melhorar o desempenho; utilização mais eficiente dos recursos.

Os benefícios que podem ser obtidos com as técnicas de BC são a melhoria da escalabilidade e da vazão dos sistemas [1]. A segurança também pode ser melhorada se o sistema se adaptar dinamicamente às mudanças de configuração que surgirem, ou às falhas de software. Tais técnicas também possibilitarão o uso mais eficiente dos recursos, permitindo que novos servidores sejam inseridos somente quando necessário.

Este trabalho apresenta uma proposta de implementação de uma arquitetura de serviços distribuída para as Redes Inteligentes, onde a aplicação de técnicas de BC é de fundamental necessidade, para o balanceamento do número de *threads*. Os enfoques principais são:

- Mostrar a necessidade de implementação de um mecanismo de BC para otimizar a escalabilidade e a vazão nos servidores da arquitetura proposta;
- Propor um serviço de BC mais adequado às especificidades desta arquitetura;
- Demonstrar como esse serviço de BC pode ser combinado com outros, para tornar a implementação da arquitetura mencionada mais atrativa do ponto de vista de utilização dos recursos de software e hardware.

O trabalho foi organizado da seguinte forma: Na seção 2, as políticas de BC são identificadas, contextualizadas e explicadas. Na seção 3, apresenta-se um serviço de BC adaptativo por demanda construído num ORB comercial compatível com o CORBA, denominado *The Adaptive communication environment Orb* (TAO), usado como mecanismo para otimizar os parâmetros de desempenho da arquitetura (escalabilidade e vazão). O modelo de arquitetura de serviços, os cenários em que o serviço de BC é inserido e as razões pelas quais ele foi escolhido são mostrados na seção 4. Na seção 5 apresenta-se a arquitetura de GW proposta. Um exemplo de implementação usando o serviço de BC é apresentado na seção 6 e as principais conclusões na seção 7.

2. BC em Sistemas Distribuídos

Os mecanismos de BC tentam distribuir a carga de trabalho dos clientes eqüitativamente entre servidores,

e estão sendo aplicados em três níveis: de Rede, de Sistema Operacional e de *Middleware* [1].

O BC no nível de rede pode ser feito por roteadores IP e servidores de nomes que servem a um *pool* de máquinas *hosts*. Por exemplo, quando o cliente resolve o nome do *host*, o serviço de nomes pode designar dinamicamente um endereço IP diferente para cada requisição, baseado nas condições de carga. O cliente então cantata o servidor final designado independente do servidor que poderá ser selecionado para a próxima resolução do serviço de nomes. Os roteadores também podem ligar um fluxo TCP para qualquer servidor final, tomando por base as condições atuais de carga e usar então aquela ligação ao longo do fluxo.

O BC baseado em sistema operacional é oferecido pelos sistemas operacionais distribuídos por meio de clusterização, compartilhamento de carga e mecanismos de migração de processos. A clusterização é um tipo eficiente que consegue alta confiabilidade e desempenho, combinando computadores para melhorar a potência do sistema de processamento geral. Os processos podem então ser distribuídos transparentemente entre computadores no *cluster*. Os *clusters* normalmente requerem compartilhamento de cargas e migração de processos. Os mecanismos de migração de processo fazem o BC nos processadores ou nós de rede transferindo o estado de um processo entre os nós. A transferência dos estados requer um suporte significativo da infraestrutura para manipular as diferenças de plataformas entre os nós. Ele também pode limitar a aplicabilidade das linguagens de programação, como do JAVA, por exemplo.

O BC executado na camada *Middleware* pode ser realizado por-sessão, por-requisição ou por-demanda. A carga de trabalho do cliente é distribuída eqüitativamente sobre os vários servidores na rede, a fim de melhorar os tempos de resposta às requisições dos clientes. Neste trabalho, o BC é feito no nível de *Middleware*, particularmente sobre ORBs CORBA, em aplicações de serviços de Redes Inteligentes distribuídas. Esses ORBs permitem que os clientes invoquem operações nos objetos distribuídos sem a preocupação com a localização do objeto, a linguagem de programação, a plataforma do sistema operacional, os protocolos de comunicação e interconexão, e o hardware. O ORB pode determinar qual cliente será encaminhado para determinada réplica de objeto em um determinado servidor. Uma réplica é definida como uma instância duplicada de um objeto em particular, em um servidor gerenciado por um BC.

De acordo com Othman [1], existem várias limitações nas arquiteturas de BC baseadas em rede e em sistemas operacionais, tais como falta de flexibilidade e de adaptabilidade. A falta de flexibilidade ocorre devido à inabilidade para tomar decisões de BC em tempo real. A falta de adaptabilidade acontece devido à ausência de realimentação proveniente de um dado conjunto de réplicas, bem como devido à inabilidade de controlar se e quando uma dada réplica deverá aceitar requisições adicionais.

2.1. Formas de BC nos ORBs

Um balanceador de carga tem a finalidade de ligar uma requisição de cliente a uma réplica, e a cada momento tomar uma decisão de BC em relação a seleção dessa réplica. Prover mecanismos para executar essas ligações dos clientes, significa modificar os serviços CORBA padrões, os protocolos e interfaces proprietárias ou usar mensagens específicas no protocolo padrão entre os ORBs, o *General Inter-ORB Protocol* (GIOP).

A ligação dos clientes pode ser feita de acordo com sua granularidade, ou seja, por-sessão, por-requisição ou por-demanda. No tipo de ligação por-sessão, as requisições de clientes continuarão sendo emitidas à mesma réplica durante uma sessão. No tipo de ligação por-requisição, cada requisição de cliente será remetida a uma réplica potencialmente diferente.

No tipo de ligação por-demanda, as requisições dos clientes podem ser ligadas à outra réplica sempre que o balanceador de carga julgar necessário. Este método força um cliente a emitir sua próxima requisição a uma réplica diferente da qual ele está enviando sua requisição no momento.

2.2. Categorias de BC nos ORBs

Ao projetar um serviço de BC, é importante selecionar um algoritmo apropriado que decidirá qual réplica processará cada requisição de entrada. Por exemplo, aplicações nas quais todas as requisições geram uma carga quase idêntica podem usar um algoritmo simples *round-robin*, que será explicado mais adiante [1], enquanto que aplicações nas quais a carga gerada por cada requisição não pode ser prevista antecipadamente requer algoritmos mais avançados. Em geral, pode-se classificar as políticas de BC em duas categorias:

- Não-adaptativa: pode usar políticas não adaptativas, como um algoritmo simples de *round-robin* ou um algoritmo de randomização [1], para selecionar qual réplica controlará uma requisição em particular;
- Adaptativa: pode usar políticas adaptativas que utilizam informações em tempo real para selecionar a réplica que controlará uma requisição particular.

2.3. Arquiteturas de BC

Combinando os níveis, formas e categorias descritas anteriormente, pode-se criar arquiteturas alternativas de BC. A Figura 1 ilustra três arquiteturas que estão sendo mais exploradas [1].

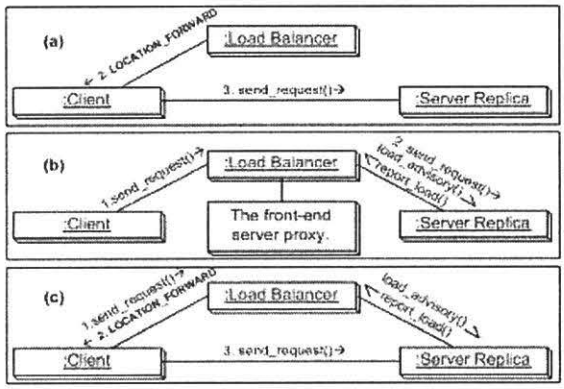

Figura 1. Arquiteturas para BC

Na arquitetura de BC por-sessão não-adaptativa (Fig. 1a), um balanceador de carga é projetado no CORBA de forma a selecionar a réplica designada quando uma sessão de cliente-servidor é estabelecida pela primeira vez, ou seja, quando um cliente obtém uma referência para um objeto CORBA (chamado aqui de réplica) e se conecta àquele objeto.

Diferentes clientes podem ser direcionados a diferentes réplicas de objeto usando um recurso de ativação, como um Repositório de Implementação do CORBA, ou um Serviço de Localização, como o Serviço de Nomes.

Essa arquitetura de balanceamento pode satisfazer as necessidades de transparência das aplicações, aumentando a segurança do sistema e a interoperabilidade do CORBA. Porém, arquiteturas de BC por-sessão não-adaptativas não são adequados quando necessita-se que a manipulação dinâmica das requisições de operação dos clientes seja feita adaptativamente.

Na arquitetura de BC por-requisição adaptativa (Fig. 1b), existe um servidor que é um proxy, o qual recebe todas as requisições dos clientes. Neste caso, este servidor contém o balanceador de carga. Ele seleciona uma réplica no servidor final de acordo com sua política de BC e envia a requisição para aquela réplica. O servidor proxy espera a resposta da réplica chegar, e a retorna para o cliente. Mensagens de consulta de carga são enviadas do balanceador para as réplicas, fazendo com que elas aceitem as requisições ou as redirecionem para o balanceador de carga.

O principal benefício do uso dessa arquitetura é o aumento da escalabilidade e a equidade. Por exemplo, o servidor intermediário pode examinar a carga em cada réplica antes de selecionar a réplica para cada requisição, distribuindo a carga eqüitativamente. Portanto, ela é indicada para balanceamentos adaptativos. No entanto, ela introduz *overhead* devido à inclusão de mensagens de consulta [2].

Outro problema é que em certas aplicações, como é o caso do exemplo apresentado neste trabalho é importante que o servidor intermediário possa executar outras tarefas, enquanto aguarda a resposta do servidor final, o que não é possível com essa arquitetura.

Na arquitetura de BC por-demanda adaptativa (Fig. 1c), o cliente recebe inicialmente uma referência para o objeto do balanceador de carga. Usando um mecanismo chamado *LOCATION_FOWARD*, o balanceador pode direcionar a requisição inicial do cliente para uma determinada réplica. Os clientes continuarão usando a nova referência para objeto, que é obtida como parte da mensagem *LOCATION_FORWARD*, para se comunicarem com esta réplica diretamente até que eles recebam outros redirecionamentos ou terminem suas conversações.

Essa arquitetura pode monitorar a carga da réplica continuamente, diferentemente das arquiteturas não-adaptativas. Usando a informação de carga, um balanceador poderá determinar como distribuí-la de maneira equilibrada, redirecionando o cliente para a réplica menos sobrecarregada. A informação sobre o estado da carga (nominal ou sobrecarga) pode ser definida pelo usuário.

Dependendo do projeto podem surgir problemas, devido ao fato de que as réplicas no servidor precisam ser preparadas para receber mensagens de um balanceador de carga, e que os clientes precisam ser redirecionados para o balanceador.

Embora essas mudanças sejam necessárias, e não afetem a lógica da aplicação, os desenvolvedores de aplicativos precisarão modificar os componentes de inicialização e ativação dos servidores CORBA para responder às mensagens de consulta mencionadas anteriormente.

Entretanto, para aplicações em que o servidor intermediário precise dar vazão, simultaneamente, para muitas requisições, independente do recebimento dos resultados das requisições, provenientes do processamento nos servidores finais, essa arquitetura parece ser a mais adequada.

3. Uma Visão do Serviço de BC do TAO

O ORB conhecido como TAO, foi projetado para suportar aplicações exigências de QoS [3]. Ele

implementa um serviço de BC, o qual facilita o desenvolvimento de aplicações distribuídas em ambientes heterogêneos, pois o mesmo provê transparência para as aplicações, alta flexibilidade, escalabilidade e vazão além de ser adaptável em tempo real. O serviço de BC foi projetado conforme ilustrado na Figura 2, e compreende dos seguintes componentes [3]:

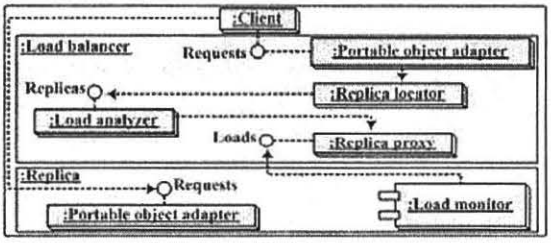

Figura 2. Interações dinâmicas no BC do TAO

- *Replica locator*: Este componente identifica quais réplicas receberão determinadas requisições e também liga os clientes às réplicas identificadas. O localizador de réplica pode ser portável, usando mecanismos do adaptador de objetos do padrão CORBA (o POA);
- *Load monitor*: Monitora as cargas em uma dada réplica, reporta a carga da réplica ao balanceador e responde às consultas que o balanceador envia. Ele também processa essas consultas e informa às réplicas quando elas devem aceitar requisições;
- *Load analyzer*: Este componente decide qual réplica receberá a próxima requisição do cliente. O localizador de réplica obtém a referência para a réplica do analisador de carga e então envia a requisição para a réplica;
- Replica proxy: Cada objeto que o serviço de BC do TAO gerencia, se comunica com ele através de um único proxy. O balanceador de carga usa esses proxies para distinguir as identidades dos objetos. Por exemplo, duas referências para um objeto são equivalentes se elas se referem ao mesmo objeto. Se elas não se referirem ao mesmo objeto, então elas não são equivalentes ou o ORB não consegue determinar;
- *Load balancer*: Este componente é um mediador que integra todos os componentes descritos acima. Ele provê uma interface para o BC sem expor os clientes a interações complexas entre os componentes que se integram.

Selecionar uma réplica usando uma política de balanceamento não adaptativo pode carregar as réplicas não uniformemente. Em contrapartida, selecionar uma réplica usando uma política de balanceamento adaptativo para cada requisição pode causar *overhead* e latência excessiva. Então, para evitar essas duas situações, o serviço de balanceamento de carga do TAO provê uma solução híbrida através da implementação de dois algoritmos de BC:

- *Round-Robin*: Este algoritmo não adaptativo é direto e não considera a carga. Em vez disso, ele simplesmente envia uma requisição para a próxima réplica no grupo de objetos que estão sendo balanceados.
- Dispersão mínima: Este algoritmo é adaptável e mais sofisticado que o algoritmo round-robin. A meta deste algoritmo é assegurar que as cargas caiam dentro de uma certa tolerância, tentando minimizar a diferença entre a carga em cada réplica e a média dessas cargas.

A Figura 3 ilustra a execução do serviço de BC do TAO, explicado como segue:

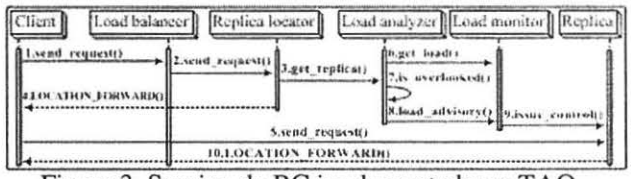

Figura 3. Serviço de BC implementado no TAO

1. Um cliente obtém uma referência para aquilo que parece ser uma réplica e invoca uma operação. Na realidade, o cliente invoca transparentemente as requisições em seu balanceador de carga.
2. Após receber a requisição de um cliente, seu adaptador de objetos envia a requisição para o localizador de réplica.
3. Depois, o localizador de réplica consulta o analisador de carga se há uma réplica adequada no servidor.
4. O localizador de réplica então, redireciona transparentemente o cliente para a réplica escolhida.
5. Os clientes continuarão a enviar requisições diretamente para a réplica escolhida até que o balanceador de carga detecte uma alta carga naquela réplica. A comunicação direta entre cliente e a réplica elimina o *overhead* adicional incorrido nas arquiteturas de BC por requisição.
6. O balanceador monitora a carga da réplica. Dependendo da política de troca de informações que foi configurada, o monitor de carga informará ao balanceador qual é a carga ou o balanceador questionará o monitor pela carga na réplica.
7. Assim, como o balanceador coleta cargas, o analisador analisa as cargas na réplica.
8. Se uma réplica se torna sobrecarregada, o balanceador pode remeter dinamicamente o cliente para outra réplica menos carregada. Para atingir um certo grau de transparência, o balanceador do

TAO, não se comunica com a aplicação do cliente quando a está transferindo para outra réplica. Em vez disso, o balanceador expede uma consulta para o monitor de carga da réplica.

9. O monitor de carga envia uma mensagem de controle para a réplica. Dependendo do seu conteúdo, a mensagem induzirá a réplica a aceitar ou redirecionar requisições.
10. Quando instruído pelo monitor de carga, a réplica usa a mensagem do *GIOP LOCATION_FORWARD* para redirecionar a próxima requisição enviada por um cliente de volta para o balanceador de carga.
11. Neste ponto, o ciclo do balanceamento de carga inicia novamente.

4. Aplicação do BC nas RIs Distribuídas

As RIs são constituídas por vários sistemas bem definidos, que se comunicam por meio da Rede de Sinalização por Canal Comum No. 7 (SCC7), usando o *Intelligent Network Application Protocol* (INAP). Os Pontos de Acesso aos Serviços (PAS) são as centrais de comutação digitais com capacidades para acessar a lógica e os dados de serviços centralizados nos Pontos de Controle de Serviços (PCS).

Este modelo de rede, adicionado ao surgimento de novos serviços do tipo banda larga, tem agravado os problemas que as operadoras de rede enfrentam, tais como: pouca flexibilidade na oferta de serviços mais sofisticados; difícil portabilidade dos serviços; difícil interoperabilidade entre sistemas de fabricantes diferentes; baixa escalabilidade dos servidores; difícil reutilização do software; difícil integração com outros tipos de redes e fraca personalização dos serviços [4].

Para enfrentar tais problemas e proporcionar a abertura da rede ao mercado de serviços banda larga, optou-se por um processo de substituição gradual dos seus elementos, usando técnicas de orientação a objetos e processamento distribuído, suportado por uma plataforma compatível com o CORBA. Esse processo está ilustrado na Figura 4.

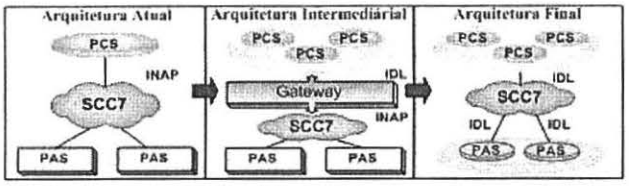

Figura 4. Um Caminho de Evolução para as RIs

Inicialmente a introdução de objetos distribuídos ocorre na camada de serviços, em substituição aos atuais Pontos de Controle de Serviços (PCSs), preservando a estrutura de acesso (PASs). Assim, os PCSs passam a ser distribuídos em servidores CORBA e acessados pelos PASs da RI convencional através de um *gateway* (GW).

Na segunda e última etapa ocorre a substituição de todos os elementos da RI por objetos de serviço CORBA, mantendo a rede SCC7 como rede de transporte entre os diferentes nós CORBA.

Na sua fase final, a arquitetura deverá também prover os recursos necessários para que seus usuários possam acessar e usar os serviços da RI convencional.

Assim, existirão dois cenários em que um GW deverá ser usado, como ilustra a Figura 5. Um cenário, cenário 1, corresponde ao acesso dos usuários da rede de telefonia convencional aos serviços da rede distribuída, ilustrado como caminho 1. O outro cenário, cenário2, corresponde ao acesso dos usuários da rede distribuída aos serviços da rede convencional, através do caminho 2. Como foi demonstrado em um trabalho anterior [5], o modelo de GW proposto [6] tem sua arquitetura simétrica, o que possibilita as comunicações para os dois cenários mencionados.

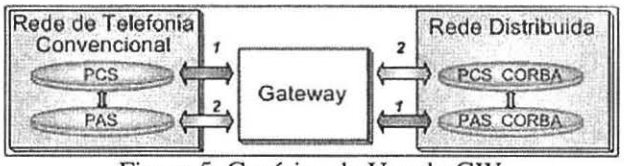

Figura 5. Cenários de Uso do GW

Independente do contexto de uso do GW, a rede se configura numa arquitetura em camadas, como ilustrado na Figura 6.

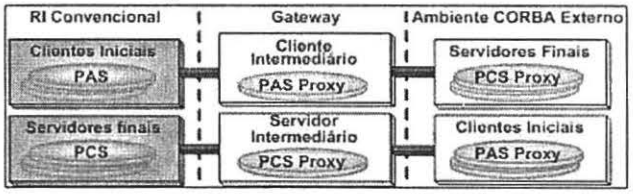

Figura 6. Arquitetura em Camadas

4.1. Análise do Cenário 1

Os ORBs compatíveis com o CORBA normalmente funcionam na base de requisições síncronas e processamentos também síncronos. Para atender as especificidades das aplicações de telecomunicações (tais como aplicações de RI) houve a necessidade de se criar um serviço específico, denominado Serviço de Mensagens do CORBA, em que as requisições são feitas de forma assíncrona. Tal serviço foi usado no projeto do GW a fim de conferir um certo grau de escalabilidade ao modelo.

Entende-se por escalabilidade, especificamente para este Cenário, a capacidade de se beneficiar de múltiplos servidores finais e de um objeto PAS Proxy

no GW de manipular muitas requisições agregadas, como ilustra a Figura 7.

As mensagens provenientes dos PASs convencionais são encaminhadas ao GW, convertidas na linguagem apropriada ao ambiente CORBA. No GW elas são direcionadas a um objeto PAS Proxy o qual representa o PAS convencional no domínio CORBA, que requisita uma operação em um objeto PCS CORBA no servidor final.

Com o Serviço de Mensagens, um mesmo PAS Proxy pode manipular várias requisições para um mesmo PCS CORBA e/ou para outros PCSs dentro do mesmo servidor. O Serviço de Mensagens implementa um método conhecido como *Asynchronous Method Invocation* (AMI) [7], como ilustrado na Figura 7.

Figura 7. Um Modelo CORBA para AMI.

Quando um cliente invoca uma operação assíncrona em um objeto (1) (2), ele passa uma referência como parâmetro para um objeto local denominado *Reply Handler object*. Quando a resposta retorna (3), o ORB chama este objeto (4) para notificar à aplicação que a resposta está disponível. Neste modelo, a referência ao objeto *Reply Handler* não é passada ao servidor, mas é armazenada no ORB do cliente. Quando a resposta do servidor chega, o ORB do cliente a despacha por uma operação *callback* apropriada no *Reply Handler* provido pela aplicação cliente.

Através desta estratégia um cliente pode fazer várias requisições em objetos residindo em um ou mais servidores sem a necessidade de bloquear o sincronismo das respostas. Além disso, os clientes podem iniciar longos métodos de invocação concorrentemente, sem incorrer em sobrecarga que seria associada ao uso de um encadeamento separado para cada requisição.

O uso do Serviço de Mensagens, no entanto, não é suficiente para garantir altos índices de escalabilidade ao modelo do GW em condições de sobrecarga. Neste caso, um serviço de BC seria aplicável, pois possibilitaria a criação de outros objetos PAS Proxy e, conseqüentemente, a criação de várias *threads*, quando a demanda de requisições o exigisse. Sendo assim, apresenta-se como proposta para este Cenário a utilização do serviço de BC como implementado no TAO para viabilizar e garantir um bom desempenho do GW. Este mesmo serviço também seria aplicável no projeto de implementação dos servidores finais, servindo como uma metodologia para a criação de réplicas (PCSs CORBA) sempre que for necessário.

4.2 Análise do Cenário 2

Enquanto o AMI foi feito sob medida para o Cenário 1, ele não produz o mesmo efeito no Cenário 2 em que o cliente está no domínio CORBA (PAS) e o servidor final (PCS) se encontra na RI convencional. Visto que o AMI só atua em aplicações de clientes, ele não tem efeito quando o GW opera como servidor intermediário. Neste caso, pelo método convencional do CORBA, ele bloquearia seus recursos para cada associação em que se envolvesse, enquanto o servidor final estivesse processando as operações.

Neste novo contexto há a necessidade de incluir algum modelo de concorrência que garanta também altos índices de vazão, imprescindíveis para as aplicações de RI.

Entende-se por vazão a capacidade do GW (como servidor intermediário) processar muitas respostas por unidade de tempo, provenientes da RI convencional.

Como solução para garantir altos índices de vazão foi inserido no modelo um serviço de processamento assíncrono de mensagens baseado num método recentemente desenvolvido, chamado de *Asynchronous Method Handling* (AMH) [8].

O AMH foi desenvolvido para contornar as limitações dos métodos convencionais, para arquiteturas em camada como esta, que manipulam grandes volumes de tráfego e com elevado número de clientes, como é o caso. Ele pode ser compreendido como o AMI estendido para os servidores, permitindo que os servidores intermediários processem novas requisições sem ter que esperar pelas respostas dos servidores finais. Neste caso a escalabilidade também é melhorada, pelo fato de que não há necessidade de gerar um número muito grande de *threads*, quando for o caso, e os servidores não terão que necessariamente manipular as requisições na ordem de chegada, independente do modelo de concorrência que estiver sendo usado (*single-threaded* ou *multi-threaded*).

Aliando o AMH ao AMI, torna-se possível aumentar a escalabilidade e a vazão do cenário 2 (GW2), como ilustra a Figura 8.

O cliente invoca uma operação no servidor intermediário. O servidor usa o AMH para criar um objeto chamado de *ResponseHandler* e armazenar nele a informação sobre a requisição do cliente. O servidor intermediário retorna o controle ao ORB imediatamente, processa a requisição e invoca uma operação AMI no servidor final (e retorna o controle ao ORB). Quando a resposta do servidor final fica disponível, ele a despacha para o *ReplyHandler* (do

modelo AMI). O *ReplyHandler* usa o contexto da informação armazenada no *ResponseHandler* para especificar os parâmetros de saída e o *ResponseHandler* retorna a resposta ao cliente.

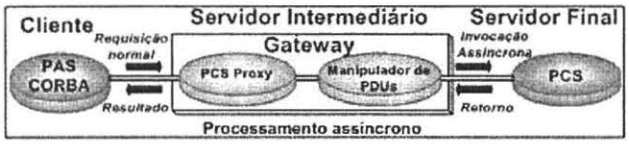

Figura 8. Processamento e invocação assíncrona.

A implementação conjunta dos métodos AMI e AMH dentro de um serviço de mensagens assíncronas foi proposta e implementada para o Cenário em questão [9]. Entretanto, o modelo pode ser otimizado com a inclusão de um serviço de BC adaptativo por demanda para gerar objetos PCS Proxy no GW quando as condições de sobrecarga surgirem.

5. Novo Serviço de Suporte para o GW

O modelo de arquitetura de GW proposto está indicado na Figura 9.

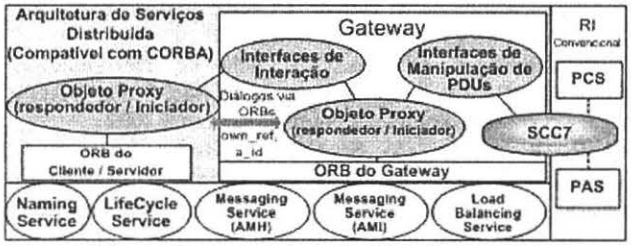

Figura 9. Arquitetura de GW Proposta

Ele é composto de um conjunto de objetos, usados para prover as operações de interfaceamento entre os ambientes externos compreendidos da RI convencional e de CORBA, respectivamente. Ele também contém serviços de suporte, dentre eles, o Serviço de BC implementado juntamente com o os métodos AMI e AMH, que deverão compor o Serviço de Mensagens. Além das vantagens citadas anteriormente no uso de tais serviços, pode-se acrescentar a possibilidade de contar com esses recursos instalados no ORB. A padronização do método AMH e do serviço de BC pelo *Object Management Group* (OMG) também é uma possibilidade real.

6. Um exemplo de implementação

Para o primeiro Cenário apresentado, está sendo desenvolvida uma implementação do serviço de BC adaptativo sob demanda no GW (para a geração dos PASs Proxy) e no Servidor (para a geração dos PCSs CORBA), parte integrante da proposta apresentada neste artigo. Conforme ilustra a Figura 10, as *threads* serão geradas de acordo com a demanda de tráfego.

Figura 10. Inclusão do Serviço de BC no Cenário 1

Quando o tráfego entre clientes PASs e um PASProxy se torna intenso, é necessário que sejam criadas réplicas do PASProxy no GW, para garantir que o sistema não fique sobrecarregado. A implementação do serviço AMI permite que um PASProxy mantenha vários diálogos provenientes de diferentes usuários da RI convencional com um mesmo objeto PCS CORBA. O serviço de BC neste caso, vem aumentar a escalabilidade e vazão, conseguidas com o serviço AMI, através da geração de outros PCSs CORBA para se comunicarem com aquele PASProxy, ou criar outros PASs Proxy para fluir o tráfego em direção a um determinado endereço no servidor.

Na Figura 10, um cliente PAS na RI convencional envia uma invocação de operação, *InitialDP* por exemplo, em um PCS que após processamento retorna uma operação *Connect*. Com o serviço de BC, o cliente invoca transparentemente a operação em seu balanceador (intermediário) de carga, que localiza um objeto (PASProxy) adequado no GW ou gera uma réplica se for necessário, e fica monitorando sua carga. O PASProxy designado invoca a mesma operação (porém na notação IDL) em um PCS no ambiente CORBA que, de modo análogo, invoca transparentemente a operação em seu balanceador de carga. O balanceador (final) localiza um objeto PCS adequado no servidor final ou gera uma réplica se for o caso, e fica monitorando sua carga. O PASProxy invoca o método *sendc_message* (este é o prefixo pelo qual se reconhece que a mensagem é assíncrona) para emitir a operação assíncrona, usando para isso o serviço de mensagens AMI, passando como parâmetros, entre outros:

- own_ref, que é a referência ao próprio objeto para que os resultados das operações sejam retornados corretamente;
- a_id, é o identificador da associação, para que o objeto possa conversar com outros objetos sem perder a referência da conexão;
- "InitialdP", é a primeira operação INAP normalmente invocada.

No segundo Cenário, é proposta a implementação do serviço de BC no GW para a geração econômica de

PCSs Proxy. O serviço de BC, neste caso, é uma estratégia para otimização da escalabilidade e vazão nas RIs, com os serviços de mensagens AMI e AMH conjuntamente implementados. Esta associação é ilustrada na Figura 11.

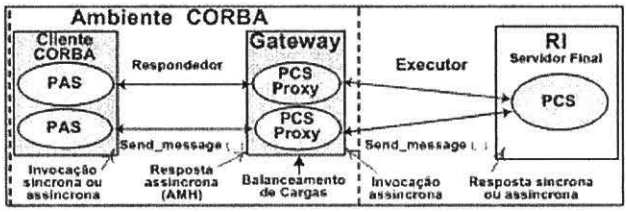

Figura 11 Inclusão do Serviço de BC no Cenário 2

O cliente (PAS CORBA), invoca uma operação no servidor intermediário, que delega a mesma requisição ao servidor final para processamento. Implementando o serviço de BC, o cliente invoca transparentemente a operação em seu balanceador de carga, que localiza um objeto (PCSProxy) adequado no GW ou cria uma réplica, e monitora sua carga.

Um objeto *reply handler* (criado pelo método AMI) no GW possui uma referência para outro objeto *response handler* (criado pelo método AMH), também no GW. Essa referência para o objeto é gerada para a invocação assíncrona. Quando o *reply handler* é invocado, ele usa essa referência para retornar o resultado ao cliente. Quando o servidor final é invocado assincronamente uma referência para o *reply handler* é passada como primeiro parâmetro no método *sendc_*, seguido dos parâmetros definidos na interface do executor.

Nos dois Cenários apontados onde são acrescentados os serviços de BC, se a réplica ficar sobrecarregada as requisições são remetidas dinamicamente para outra réplica menos sobrecarregada ou cria-se mais uma. Essas manipulações são transparentes para as aplicações do cliente. O serviço de BC sugerido é do TAO, pois [2]: Suporta um modelo de BC orientado a objeto; permite transparência para as aplicações dos clientes; permite transparência para as aplicações dos servidores; maximiza a escalabilidade e distribui dinamicamente e eqüitativamente a carga; aumenta a fidelidade do sistema e, causa pouco *overhead.*

7 . Conclusões

A tendência atual de transformação dos sistemas de telecomunicações em grandes ambientes distribuídos tem revelado a necessidade premente de adaptação das técnicas de implementação desses sistemas às aplicações em tempo real as quais exigem alto desempenho (altos índices de escalabilidade e vazão), com alta confiabilidade (segurança). Neste contexto, este trabalho apresenta como contribuição principal uma proposta de ferramenta para otimizar tais parâmetros de desempenho e confiabilidade nas Arquiteturas de Serviços Distribuídas, previstas como as próximas gerações das RIs.

Foi demonstrada a necessidade de implementação de uma estratégia de BC para proporcionar a manipulação de grandes volumes de tráfego pelos servidores da arquitetura de redes, para em conjunto com os métodos AMI e AMH alcançarem altos níveis de desempenho para os aplicativos das RIs.

Espera-se apresentar em futuro trabalho resultados da aplicação do modelo proposto para as RIs distribuídas, bem como avaliar algumas questões importantes, tais como: taxa de overhead gerada, as metodologias para a geração de threads, entre outras.

8. Referências

[1] O. Othman, C. O'Ryan, D. C. Schmidt, "Strategies for CORBA Middleware-Based Load Balancing", *IEEE DS Online*, Volume 2, Number 3, March 2001.

[2] O. Othman, C. O'Ryan, D. C. Schmidt, "The Design and Performance of na Adaptive CORBA Load Balancing Service", www.cs.wustl.edu/~schmidt/PDF.

[3] O. Othman, C. O'Ryan, D. C. Schmidt, "Designing an Adaptive CORBA Load Balancing Service Using TAO", *IEEE DS Online*, Volume 2, Number 4, March 2001.

[4] R. B. Soares, A. S. Garcia, "Redes Inteligentes: Alternativas para a Introdução de Sistemas Distribuídos", XIX Simpósio Brasileiro de Telecomunicações, Fortaleza CE, Setembro 2001.

[5] R. B. Soares, A. S. Garcia, "Uma Proposta de Metodologia para a Integração de CORBA na RI", XIV Congresso Brasileiro de Automação, Setembro 2002.

[6] OMG Document Number: telecom/98-10-03, "Interworking between CORBA and TC Systems", October 1998.

[7] A. B. Arulanthu et al, "The Design and Performance of a Scalable ORB Architecture for CORBA Asynchronous Messaging", Proceedings of the Middleware 2000 Conference, ACM/IFIP, April 2000.

[8] D. Brunsch et al, "Designing an Efficient and Scalable Server-side Asynchrony Model of CORBA", Proceedings of the ACM SIGPLAN Workshop on Optimization on Middleware and Distributed Systems (OM 2001) Snowbird, Utah, June 18, 2001.

[9] R. B. Soares, "Estratégias de Introdução de Sistemas Distribuídos nas Redes Inteligentes", Tese de Doutorado, UFES, Vitória ES, 2003.