SUPER-AMigOs: um Sistema Tolerante a Falhas para Migração de Objetos em Sistemas Distribuídos

Marcia Pasin, Bruno Jatene, Lenise Cristina Geiss, Taisy Silva Weber
Instituto de Informática, UFRGS
Programa de Pós-Graduação em Computação
Caixa Postal 15064 CEP 91501-970 BRASIL
E-mail: {pasin, jatene, lenise, taisy} @inf.ufrgs.br

RESUMO

Sistemas distribuídos orientados a objetos modelam processos, estruturas de dados (arquivos e diretórios) e serviços como entidades completas, denominadas objetos. Os objetos possuem estado (informações sobre o objeto), métodos (que são usados para alterar o estado do objeto) e podem migrar de uma estação servidora para outra. A migração de objetos permite balanceamento de carga entre servidores e desempenho eficiente ao sistema, mas só faz sentido em redes de longa distância pois o custo de transferir um objeto de uma estação para outra é muito alto. Portanto, é nesessário prover um esquema genérico, independente de plataforma de rede para permitir a migração de objetos. A proposta deste artigo descreve um sistema tolerante a falhas para a migração de objetos em ambientes heterogêneos chamado SUPER-AMigOs.

Palavras-chave: sistemas distribuídos, migração de objetos, tolerância a falhas, comunicação de grupo.

ABSTRACT

Object-oriented distributed systems model processes data structures (files and directories) and services as whole entities called objects. Objects have a state (or object data), methods (used to change the object state) and can migrate between distinct servers stations. Object migration is an important approach of distributed computing systems. The migration approach improves load balancing between servers and efficient performance to the system. The object migration is useful at wide area networks because the cost of object transference is very high. Therefore, is necessary to provide a generic mechanism, with network platform independence to object migration. This paper shows a fault-tolerant system to object migration in heterogeneous environment called SUPER-AMigOs.

Keywords: distributed systems, object migration, fault tolerance, group communication.

1 Introdução

Assume-se sistemas computacionais distribuídos orientados a objetos, onde estações suportam dois tipos principais de objetos: objetos servidores, ou simplesmente servidores, e objetos clientes, ou simplesmente clientes. Servidores fornecem serviços a clientes espalhados pela rede. Clientes invocam métodos de servidores para obter os serviços.

Um mecanismo importante, ocasionalmente provido nestes sistemas, é a migração de objetos entre diferentes estações. Na migração, um servidor é transferido de uma estação (fonte) para outra

(alvo), permitindo balanceamento de carga e aumento da eficiência no sistema distribuído. O maior ganho com a migração de objetos é alcançado em redes de longa distância (WANs ou *wide area networks*), onde o custo de transferência de um objeto entre estações na rede é compensado pela facilidade de acesso a um objeto bem localizado.

A migração automática de objetos é prevista em projetos de sistemas para Internet como o Globe [STE97], W3Objects [ING95] e outros baseados na arquitetura CORBA. Para estes sistemas, a migração, a distribuição e a replicação de objetos podem ser oferecidas como serviços do nível de suporte, transparentes para o usuário. Contudo, quando o sistema distribuído não oferece esta transparência, o programador pode integrar sua aplicação com mecanismos que realizem a migração de objetos. Este estilo de programação pode usar bibliotecas com funções pré-definidas, onde as funções são combinadas ou adaptadas para atingir as necessidades da aplicação. Exemplos de bibliotecas com funções pré-definidas incluem os sistemas Isis [BIR93] e Ensemble [HAY98]. O uso de bibliotecas de programação ainda pode ser justificado pela necessidade de um mecanismo genérico para realizar a migração de objetos, já que a migração só faz sentido em WANs.

A principal contribuição deste trabalho é apresentar um sistema chamado SUPER-AMigOs, com migração automática de objetos usando uma ferramenta de comunicação de grupo para implementar alta confiabilidade. O SUPER-AMigOs é formado por objetos especiais (que são chamados de agentes) que coordenam a migração de objetos servidores em redes heterogêneas.

Este trabalho está organizado da seguinte forma: a seção seguinte descreve o procedimento de migração de objetos. A seção 3 apresenta as propriedades necessárias para a migração de objetos. A seção 4 apresenta o modelo do sistema distribuído considerado. A seção 5 descreve como serão implementados os objetos que formam o sistema SUPER-AMigOs. A seção 6 descreve trabalhos relacionados e a seção final apresenta as conclusões.

2 Migração de Objetos em Sistemas Distribuídos

A migração de um objeto carrega: (a) uma cópia de um objeto em uma estação alvo, que possua as mesmas classes que o objeto possuía na estação fonte; (b) atribui um estado à cópia do objeto, (c) registra o novo objeto na estação alvo e (d) destrói a cópia original [RUB96]. A partir da sinalização de migração recebida por um agente (usuário ou método executado pelo sistema), um objeto que será migrado deve esperar pelo término de todos os seus métodos, isto é, não deve executar mais nenhuma tarefa. O objeto empacota seu estado em uma mensagem (fig. 1) que pode ser enviada e interpretada em outra estação.

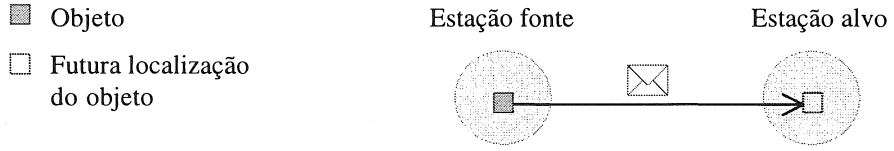

Figura 1 – Transferência do objeto em forma de mensagem

O objeto precisa informar ao serviço de nomes sua nova localização física e pode então ser eliminado da estação fonte. Na estação alvo, uma versão para o novo objeto precisa ser instalada. Esta versão extrai a informação do objeto da estação fonte para atualizar o seu estado e registra-se no serviço de nomes da estação alvo.

Todas as mensagens recebidas pelo objeto durante sua transferência são dispostas em uma FIFO (*first in first out*). Quando a transferência terminar, as mensagens enfileiradas precisam ser redirecionadas e executadas para o novo objeto. Uma mensagem é enviada da estação alvo para a estação fonte, indicando o resultado da migração. Quando a estação fonte receber um resultado de êxito, pode eliminar a sua cópia do objeto (fig. 2), finalizando a migração do mesmo.

Figura 2 – Remoção do objeto na estação fonte

Deve ser observado que falhas que ocorram durante a migração sejam tratadas, sob pena de deixar o sistema distribuído em um estado inconsistente e o objeto migrado indisponível em qualquer estação.

A migração de objetos servidores permite balanceamento de carga entre estações e eficiência ao sistema distribuído (fig. 3). O objeto servidor pode migrar para uma estação mais ociosa que a atual, evitando gargalos e equilibrando a carga operacional das estações no sistema distribuído.

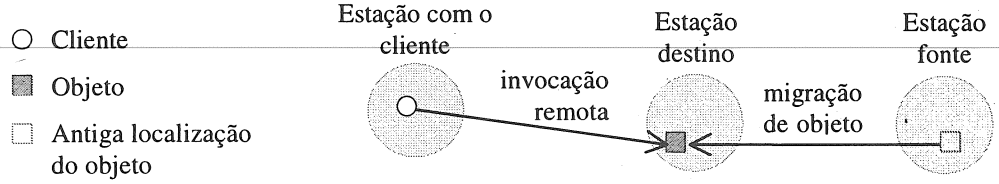

Figura 3 – Migração de objeto entre estações

Para que a migração torne o sistema distribuído eficiente, um objeto em uma rede WAN pode migrar de uma estação distante para outra mais próxima dos clientes. Neste caso, o acesso ao objeto é otimizado, diminuindo o tráfego na rede e o tempo de latência da comunicação remota. Evidentemente que nem sempre a localização física indica menor tráfego, pois apesar de estar mais próximo, o objeto migrado pode ter sido deslocado para uma área com alto tráfego de informações. Essas métricas de desempenho devem ser consideradas para a escolha da melhor estação na rede para um dado objeto em um determinado intervalo de tempo, com vistas à otimização do desempenho da aplicação envolvida. Essa escolha representa um típico problema de otimização e, para evitar a complexidade desses problemas, deve ser tratada por heurísticas.

Um exemplo, que ilustra a migração de objetos, é a realização da matrícula de disciplinas nas universidades. Em uma dada universidade há uma WAN, ligando computadores espalhados em vários *campi*. Todas as informações sobre as disciplinas, que estão armazenadas nos computadores do centro de processamento de dados (CPD), precisam ser consultadas e alteradas durante o período de matrícula. O CPD está localizado no centro da cidade. As matrículas dos cursos são realizadas em vários *campi*, num raio de 10 km do centro. Durante a matrícula, o tráfego na rede da universidade é intenso. Muitas informações são trocadas entre os computadores dos *campi* e do CPD. Uma forma de otimizar a realização da matrícula, é migrar objetos contendo informações sobre disciplinas exclusivas de cada curso para os computadores de cada curso no campus correspondente. Objetos de disciplinas compartilhadas por vários cursos podem migrar de *campus* a *campus* conforme a demanda em dado momento do processo de matrícula. Assim, os computadores de cada curso podem fazer acesso local aos dados, sem precisar consultar as informações no CPD.

Outro exemplo ilustrativo são objetos que representam pacotes turísticos vendidos por uma operadora através de agências de viagens, espalhadas por todas as regiões do globo. A demanda por determinado pacote é imprevisível, assim todos os objetos residem inicialmente no computador da operadora. Eventualmente (devido a algum feriado regional, a variações climáticas, a súbito aumento da massa de salários ou qualquer outro fator imponderável, como uma personalidade local ter elogiado certo roteiro) agências de viagem de uma determinada região inundam a operadora com invocações de determinado pacote. A migração desse objeto para algum computador na região de maior demanda traz benefícios para todo o sistema.

É interessante notar que nem disciplinas nem pacotes turísticos podem sumir de um sistema devido a falhas no processo de migração. Assim, a necessidade de tolerância a falhas nesse processo é óbvia.

3 Principais Propriedades para a Migração de Objetos

O mecanismo de migração de objetos em sistemas distribuídos precisa ter as propriedades de tolerância a falhas e integridade referencial. A tolerância a falhas permite que problemas durante a migração não tornem o objeto indisponível ou extraviem o objeto migrado. A integridade referencial, controla todas as requisições de acesso a objetos, mantendo o serviço para os clientes, apesar da migração de objetos.

3.1 Tolerância a Falhas Durante a Migração de Objetos

Uma forma de prover a transparência desejada é implementar a migração como uma operação ACID (*all-or-nothing, consistency, isolation and durability*), seguindo o modelo de sistema distribuído orientado a transações [LYN94]. A atomicidade (*all-or nothing*) significa que a migração precisa ser finalizada com o novo objeto operando no novo endereço, ou então o objeto precisa estar disponível na antiga estação. A consistência (*consistency*) precisa ser mantida nos acessos realizados pelos clientes, durante a migração do objeto. O isolamento (*isolation*) garante que mensagens que chegam durante a migração não influirão no procedimento da migração. Durabilidade (*durability*) é a propriedade na qual o objeto migrado precisa ser armazenado em memória estável, para que possa ser removida a cópia na antiga estação.

A atomicidade pode ser alcançada usando uma primitiva de comunicação *unicast* atômica, garantindo o sucesso no transporte do objeto entre estações. A consistência é alcançada através da integridade referencial, garantindo que não seja modificado o estado do objeto durante a migração. O isolamento também utiliza a integridade referencial mantendo uma pilha das mensagens recebidas pelo objeto que está sendo migrado. A disponibilidade de uma memória estável na estação alvo garante a durabilidade do objeto. Essas características poderiam ser obtidas também por réplicas dos objetos servidores, mas a implementação de réplicas para objetos servidores foge ao escopo desse artigo.

A ocorrência e a possibilidade de recuperação de falhas em objetos durante a migração devem ser transparentes para os clientes. A maior dificuldade de implementar tolerância a falhas, principalmente em WANs, é distinguir com precisão estações falhas de estações lentas [CAU95], já que é difícil estabelecer qualquer relação entre atrasos na comunicação e velocidade relativa de processos. Os valores admissíveis para os tempos aceitáveis de espera (*timeouts*) devem ser cuidadosamente determinados, de acordo com as características de cada rede.

Adicionalmente às operações ACID usadas para tolerar falhas, pode-se também tratar de segurança do objeto, transferindo, para a estação que recebeu o objeto, as permissões de acesso que o objeto possuía antes da migração.

3.2 Integridade Referencial Durante a Migração de Objetos Ativos

Nem sempre a migração de um objeto pode ser realizada logo após o disparo pelo agente supervisor de migração. Se o objeto estiver ativo, isto é, se algum cliente estiver acessando o objeto, os acessos precisam ser tratados.

A integridade referencial é a propriedade que trata a consistência de acessos a objetos antes, durante e após sua migração. Para implementar integridade referencial após a migração, uma fila de mensagens precisa ser formada durante a migração, ou os clientes, que fizeram requisições aos objetos, precisam ser informados que o objeto está momentaneamente indisponível. Essa informação evita futuros fracassos em tentativas de acesso. Isso obriga alterações nos clientes, para ajustá-los ao novo endereço do objeto. Para evitar este inconveniente, podem ser usados *stubs*.

Quando o objeto terminar a migração, um *stub* pode ser deixado na antiga localização informando que o objeto foi migrado, e qual é o seu novo endereço. Um *stub* (fig. 4) é gerado na antiga estação, para redirecionar as invocações ao novo endereço do objeto. Assim, para um cliente, que faz requisições à estação fonte, o objeto migrado aparece como se ainda estivesse na estação fonte.

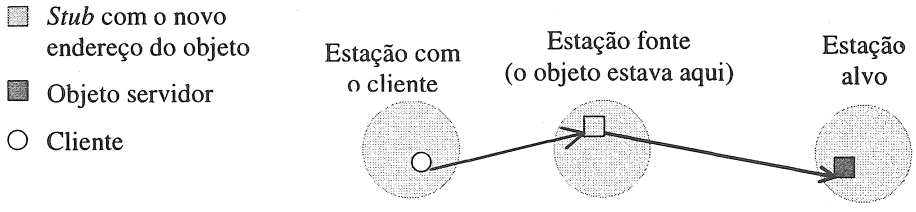

Figura 4 – Referências são redirecionadas automaticamente

Quando o cliente receber a nova localização do objeto, e quando o sistema não detectar mais nenhuma referência ao objeto na antiga estação fonte, poderá ser realizada a *coleta de lixo* para eliminar as referências e *stubs* que não são mais úteis.

4 Modelo do Sistema Distribuído para Migração de Objetos

A migração de objetos ocorre em um sistema distribuído (fig. 5), composto por objetos servidores e objetos clientes:

a) *objetos clientes*: fazem invocações dos métodos remotos localizados nos objetos servidores;

b) *objetos servidores*: contêm os métodos utilizados pelos objetos clientes. Podem migrar de uma estação para outra, mudando o seu endereço físico, de acordo com a quantidade de invocações que recebem dos clientes e métricas de carga e desempenho das estações hospedeiras.

Além destes objetos, este modelo também inclui outros objetos:

c) *agente de migração de objeto*: responsável por monitorar estações informando ao agente supervisor de migração (d) a necessidade de migração de um objeto servidor ou a disponibilidade de receber um novo objeto servidor. Nesse trabalho, o agente de migração será renomeado para Agente de MIGração de Objetos (AMigO);

d) agente SUPERvisor de migração (supervisor): possui a visão geral de todo o sistema e detecta qual o melhor local para a migração de um objeto servidor e qual o custo da transferência do objeto de um lugar para outro;

e) *stubs*, objetos que redirecionam invocações de clientes utilizados para manter a integridade referencial do sistema, após a execução do algoritmo de migração.

Cada estação do sistema possui um AMigO associado, que está sempre ativo. O AMigO contém um histórico sobre a carga da sua estação. As informações antigas do histórico podem ser comparadas localmente com as informações atuais, para detectar a necessidade da migração ou a disponibilidade de receber um novo objeto servidor. Através do histórico, o AMigO atualiza o supervisor por demanda, com as informações mais recentes do histórico. Assim, o AMigO pode ajudar o supervisor decidir sobre futuras migrações. Periodicamente, as informações mais antigas do histórico são removidas para evitar o armazenamento de grande volume de dados.

O supervisor possui a visão geral do sistema para identificar a viabilidade de migração de um determinado objeto servidor. A visão geral do sistema é atualizada pelos AMigOs, através de demanda do último estado da estação ao qual está associado. O supervisor detecta qual o melhor local para a migração de um objeto servidor e qual o custo da transferência do objeto de um lugar para outro. O algoritmo de migração é disparado automaticamente pelo supervisor, mas é executado pelo próprio objeto servidor.

O supervisor poderia ser um único objeto para controlar todas as estações, entretanto, para evitar o ponto único de falha (*single point of failure*), ele é replicado em um número pequeno de estações. As réplicas são coordenadas pelo protocolo de réplicas ativas [SCH93], onde todas as réplicas são aptas a receber as informações dos AMigOs. Vale a pena ressaltar que é importante manter todas as réplicas do supervisor mutuamente consistentes, de forma que a migração de um objeto servidor seja conhecida pelas outras réplicas. As réplicas do agente supervisor de migração poderão migrar mais de um objeto simultaneamente, haja visto que são réplicas ativas e pertencem ao mesmo grupo;

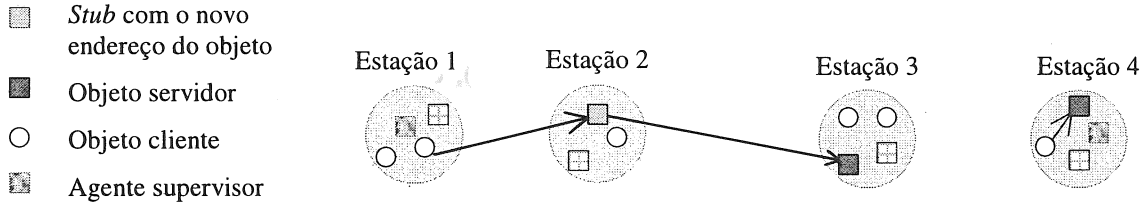

Figura 5 – Objetos no sistema distribuído

Os objetos servidores estão localizados em estações específicas, de acordo com as métricas coletadas pelo agente supervisor no sistema distribuído. As réplicas do supervisor estão localizadas em pontos estratégicos no sistema distribuído, esperando informações dos AMigOs. Neste universo de objetos, os clientes representam a grande maioria de elementos no sistema. Falhas podem acontecer em todos os tipos de objetos descritos e dividem-se em falhas de *crash* em estações e falhas de comunicação entre

estações.

Falhas de comunicação representam quebra de conexão entre os segmentos de rede. Falhas de perda ou duplicação de mensagens serão tratadas pelo sistema através do protocolo de rede. Nas falhas de *crash,* a estação pára de processar informações; não envia nem recebe mais mensagens. Falhas de *crash* podem ocorrer nas seguintes situações:

a) as falhas de *crash* em estações não geram nenhum problema para o funcionamento dos AMigOs. Cada estação possui um AMigO associado. Obviamente que se a estação falhou, todos os objetos existentes naquela estação irão parar de executar juntamente com o AMigO associado;

b) falhas em estação com réplica do supervisor de migração não comprometem o sistema, pois as demais réplicas do supervisor continuam provendo este serviço. Em caso de falha em alguma estação com réplica, o grupo deverá refazer sua visão e, se necessário, adicionar novas réplicas ao grupo garantindo funcionamento eficiente do sistema;

c) falhas em estações contendo apenas objetos clientes não afetam o sistema de migração e não necessitam de nenhum tratamento específico;

d) falhas em estações com servidores ou com *stubs* precisam ser tratadas por procedimentos específicos, que fogem ao escopo deste artigo, por exemplo usando *checkpointing* ou replicação por primário-*backup* [GUE97].

As falhas de comunicação podem ocorrer devido à falha de um segmento da rede, onde ocorrerá a divisão do sistema em duas ou mais partições. Esse tipo de falha deve ser tratada pelo grupo de réplicas do supervisor. Estas réplicas devem estar consistentes entre si, e diante de uma falha de comunicação, o grupo pode ser particionado. Portanto, o esquema de comunicação das réplicas deve prover métodos para garantir que a consistência seja alcançada durante a recomposição do grupo.

5 Implementando Agentes de Migração

O algoritmo de migração é a parte principal do sistema de migração, que compreende os agentes supervisor e AmigOs, além dos demais objetos. O algoritmo de migração é executado pelo objeto servidor após solicitação do agente supervisor (fig. 6).

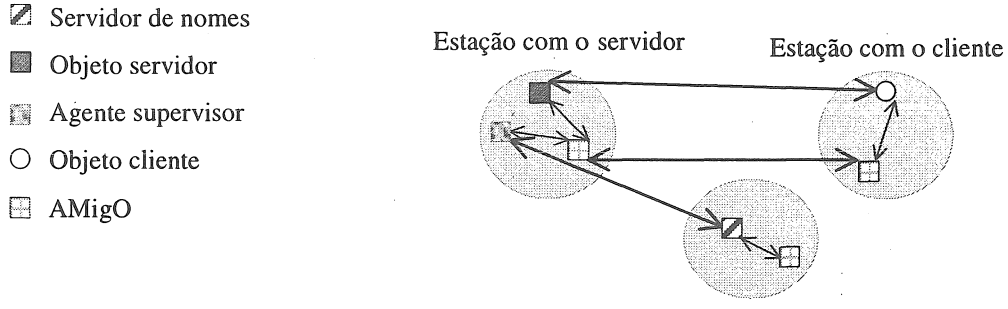

Figura 6 – Ambiente da migração com possíveis trocas de mensagens

Como resultado final do algoritmo de migração, o supervisor informa ao servidor de nomes o endereço referencial da estação destino. A análise de qual a melhor estação para o objeto migrar é executada pelo supervisor. Os principais parâmetros utilizados para avaliar a capacidade de uma nova estação receber o objeto incluem o desempenho da estação, a localização mais eficiente dentro do sistema e a disponibilidade de canais de comunicação. Para análise de decisões de migração, três fatores [RUB96] devem ser considerados pelo supervisor: *forças gravitacionais do objeto, forças atrativas do objeto* e *forças externas ao objeto.*

a) *forças gravitacionais do objeto*: indicam que o objeto deveria estar onde ele está. Por exemplo, o tamanho do objeto, dependência da execução de métodos em dispositivos locais ou dependências de comunicação com outros objetos locais;

b) *forças atrativas do objeto*: são as propriedades do objeto e métodos que sugerem que o objeto deve migrar para um determinado local. Exemplo de forças atrativas incluem dependências de dados, comunicação ou dispositivos que um método necessita para sua execução e que estão localizados em determinado local;

c) *forças externas do objeto*: influenciam nas decisões de migração e relacionam-se ao estado corrente do sistema. Neste caso, por exemplo, uma alta carga na rede irá inibir o procedimento de migração, enquanto que a disponibilidade de estações pouco utilizadas irá promovê-lo.

Através da observação destes fatores, o supervisor decide sobre a migração de um objeto. Para comunicação entre as réplicas, que estão dispersas no sistema, pode ser usada uma ferramenta de comunicação de grupo para modelar o supervisor como um grupo de réplicas ativas. A abstração de grupos permite uma estrutura adequada para implementar réplicas consistentes.

5.1 Comunicação de Grupo

Um grupo é um conjunto de objetos que cooperam para realizar uma tarefa, segundo o sistema ou a especificação do usuário. Um objeto que faz parte de um grupo é chamado membro. Cada membro do grupo possui memória local própria e não tem acesso direto aos dados de outro objeto. Em decorrência disto, objetos trocam informações através de mensagens, para realizar a requisição de serviços. A propriedade fundamental dos grupos é a atomicidade. Quando uma mensagem é enviada para o grupo, todos os seus membros devem recebê-la, na forma de comunicação do tipo um-para-muitos. A comunicação um-para-muitos pode ser de dois tipos: quando uma estação envia uma mensagem para estações específicas (*multicast*) ou quando uma estação envia uma mensagem para todas as outras estações, sem a possibilidade de indicar destinatários (*broadcast*). Estes dois tipos de comunicação contrastam com a comunicação ponto-a-ponto ou *unicast*, onde um objeto envia uma mensagem para outro objeto.

Grupos podem ser estáticos ou dinâmicos [GUE97]. No grupo estático, o número de membros permanece constante durante sua existência. Quando um membro falha, o grupo estático não altera sua configuração. Um grupo dinâmico comporta entrada (*join*) e saída (*leave*) de membros, tornando necessário mecanismos para gerenciá-lo. Quando um membro falha, o sistema retira este membro do grupo através da *troca de visão do grupo*. A visão do grupo $v_I(g_x)$ descreve quais elementos fazem parte do grupo x, em um instante i.

Para auxiliar o programador a implementar aplicações distribuídas com confiabilidade usando grupos, existem muitas ferramentas de comunicação de grupos. Uma destas ferramentas, é o Ensemble [HAY98]. O Ensemble é versátil pois pode ser executado sobre os sistemas operacionais Linux, Solaris e Windows NT. A aplicação distribuída pode ser codificada em Java, C++ entre outras linguagens de programação.

5.2 Implementando o sistema SUPER-AMigOs

No sistema SUPER-AmigOs, cada estação no sistema possui um AMigO associado, que coleta informações sobre a carga operacional e sobre as mensagens trocadas nas invocações remotas. Os AMigOs enviam as informações que julgam mais importantes e mais recentes ao supervisor, que as usa para decidir quando a migração de um objeto servidor deve acontecer e para onde o objeto deve ir.

Os AMigOs não são amigos entre si. Isto é, os AmigOs não trocam informações uns com os outros. Apenas recolhem informações da estação, à qual estão associados, e as remetem ao supervisor. Os históricos dos AMigOs não são idênticos, pois refletem diferentes estados de diferentes estações. Para uma estação com um cliente, o AMigO associado verifica localmente se o tempo de latência do servidor remoto é aceitável. O AMigO do cliente dispara uma comunicação com o supervisor quando o tempo de resposta excede o valor aceitável. O supervisor pode esperar por outra comunicação do AMigO ou dos outros AMigOs, ou questionar o AMigO da estação com o servidor sobre a carga da sua estação hospedeira. O AMigO da estação do servidor, tendo controle sobre a quantidade de invocações que está recebendo dos demais clientes e a carga da sua estação, pode informar essas métricas ao supervisor, que pode então decidir sobre a conveniência da migração. Para a decisão final, o supervisor deve conhecer a carga dos potenciais candidatos à estação alvo.

Como já foi mencionado, as réplicas do supervisor estão dispostas em pontos estratégicos do sistema. Ele é um agente replicado e precisa comunicar com os AMigOs para decidir a localização final do objeto a ser migrado. Para associar réplicas do supervisor, será usada uma ferramenta de comunicação de grupo [HAY98].

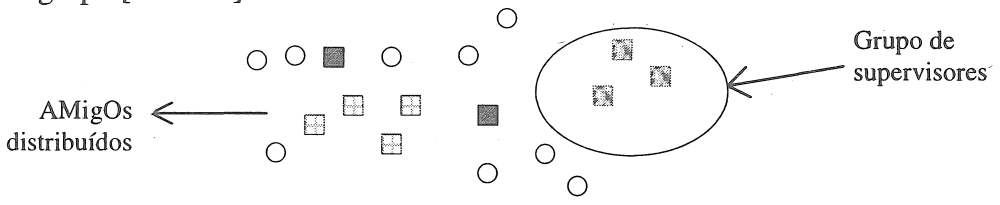

Figura 7 - Objetos do modelo de migração no sistema distribuído

De acordo com a disposição de grupos do Ensemble [HAY98], todas as réplicas do supervisor participam de um mesmo grupo (fig. 7) chamado SUPER, que possui um endereço lógico. O grupo SUPER é dinâmico, e os objetos/réplicas podem entrar e sair do grupo a qualquer instante. Uma restrição do Ensemble é o uso de grupos fechados, por questão de segurança: apenas membros de um grupo podem enviar mensagens para este. Assim, para que um objeto troque informações com um grupo, precisa antes tornar-se membro. Após trocar informações, este objeto abandona o grupo. Outra opção é implementar todos os objetos não membros do grupo como objetos CORBA usando o Maestro [VAY98] para implementar os objetos membros. Assim, os objetos clientes podem se comunicar com objetos implementados no Maestro através do protocolo IIOP, usando uma interface padronizada. A vantagem desta opção, é a possibilidade de tratar objetos em redes heterogêneas.

Por exemplo, durante a decisão de uma possível migração, um AMigO pode entrar no SUPER para expor o estado dos objetos servidores da estação à qual ele pertence. O AMigO é detectado pelo grupo de supervisores como não sendo um supervisor, e em virtude disso, é permitido a ele somente a troca de informações necessárias para atualizar o estado do SUPER sobre o comportamento do sistema.

O sistema (fig. 8) funciona da seguinte forma: o AMigO entra periodicamente no SUPER para reportar o que está acontecendo na estação, comunica-se com o mesmo através de um *multicast*, e abandona o grupo. Para que um objeto servidor migre, este precisa receber uma mensagem *unicast* do SUPER. As réplicas do agente supervisor são regidas pelo protocolo de réplicas ativas [SCH93]. No protocolo de réplicas ativas, todas as réplicas têm os mesmos direitos e podem estabelecer comunicação com os AMigOs. Cada réplica processa a mensagem, atualizando o seu estado. O protocolo de réplicas ativas é uma abordagem distribuída. A sua principal vantagem sobre a abordagem centralizada, é que a ocorrência de falhas em uma das réplicas é totalmente transparente para os demais objetos. Isso motivou a escolha desta estratégia para implementar o SUPER.

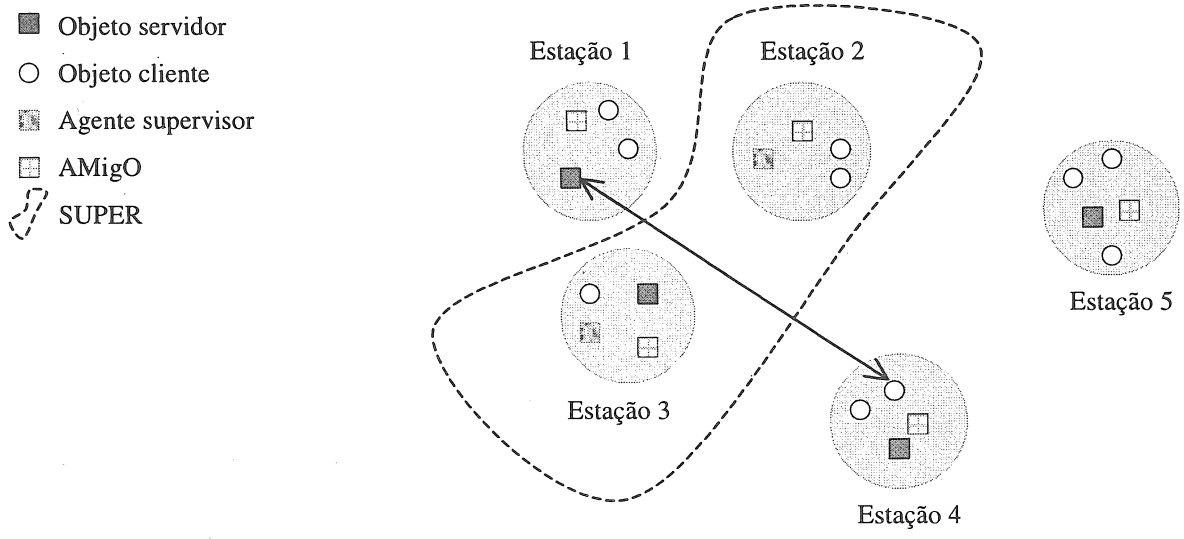

Figura 8 – Ambiente de migração de objeto

Uma outra alternativa de implementação para prover melhor eficiência no balanceamento de carga, é migrar as réplicas do SUPER. Isso pode ser feito adicionando-se novos membros ao grupos do mesmo tipo, atualizando seu estado e eliminando outras réplicas que são membros do grupo. Evidentemente, a migração de réplicas do SUPER só poderá ser efetivada quando a nova localização da réplica for mais conveniente ao desempenho do sistema.

Adicionalmente, o SUPER só dispara a migração de qualquer objeto após estabelecer contato (*unicast*) com sucesso com a sua futura estação hospedeira. Esta restrição implica que uma localização canditada precisa estar operacional, e realmente apta a receber o objeto.

6 Trabalhos Relacionados

A migração de objetos é encontrada no W3Objects [ING95] e no Piranha [MAF96], entre outros sistemas. A migração no Piranha não é inteligente, como a do W3Objects, e precisa ser acionada pelo usuário. Contudo, estes sistemas oferecem a migração como um serviço do sub-sistema, diferenciando-se da proposta deste trabalho.

O W3Objects é projetado para a Internet. Os recursos do sistema tratados como objetos e podem migrar de uma estação para outra. A migração é um procedimento robusto: a integridade referencial é garantida mantendo referências válidas dentro de objetos e entre objetos após a migração. Quando um objeto migra, um *stub* é deixado na estação fonte para invocar automaticamente o objeto migrado. Um contador de referências é incrementado a cada migração do objeto. Quando o *stub* é removido, o contador é decrementado. Um objeto só poderá ser removido quando o sistema indicar que não utilizará mais o objeto e não houver mais referências de outros objetos para este objeto. Um *stub,* que não é mais referenciado, é eliminado para reduzir o tamanho da cadeia de referências. Os *stubs,* que continuam sendo referenciados, não podem ser removidos e, por isso, a cadeia de referências pode se tornar longa demais representando diversos pontos de falha e comprometendo a eficiência do sistema. O W3Objects elimina este problema usando um mecanismo chamado *callback* [CAU95]. O projeto do W3Objects também trata quebra de conexão entre objetos e permite qualidade ao serviço. O sistema fornece informação de quais objetos são correntemente referenciados para impedir que os mesmos sejam eliminados.

O Piranha é implementado em CORBA, usando conceitos da programação de grupos. Os objetos podem migrar de uma estação para outra quando o usuário selecionar o comando adequado no menu. A migração é usada para manutenção preventiva: se a estação for desativada, o objeto que está na estação pode migrar para outra localização. A migração do objeto é realizada através da criação de uma réplica, que é sincronizada com o objeto da estação fonte, e em seguida é eliminado. Sincronização virtual [BIR93] é usada para controlar as mensagens perdidas durante a migração.

Rubin [RUB96] implementa migração de objetos para o DCE (Distributed Computing Environment). Foi Rubin quem dividiu os objetos de migração em agentes de migração (MA) e gerentes de migração (MM). Os agentes estão localizados em todas as estações e os gerentes em estações selecionadas. Os gerentes são responsáveis por coletar informações do sistema e equivalem aos supervisores de migração deste trabalho. Os agentes de migração de Rubin, equivalem aos AMigOs. Porém, Rubin não descreve como os agentes de migração e os gerentes/supervisores estabelecem o consenso para ativar a migração. Neste trabalho, isso é feito usando uma ferramenta de comunicação de grupo que permite alto grau de confiabilidade no sistema.

7 Conclusões

O modelo de programação orientado a objetos permite uma maneira conveniente para estruturar sistemas distribuídos, possibilitando a implementação de objetos em redes heterogêneas. Serviços do sistema invocados por um grande número de clientes dispersos são encapsulados em objetos, que podem migrar de uma estação para outra. A migração permite eficiência de acesso através do balanceamento de carga entre estações.

O uso de uma ferramenta de comunicação de grupo permite modelar grupos de objetos com confiabilidade, utilizando a transparência de um único endereço lógico. Além disso, possibilita a implementação da migração de algum membro do grupo, mediante atualização do estado, e a utilização de réplicas para garantir maior disponibilidade dentro do sistema.

Esse artigo apresentou um modelo de sistema baseado em migração de objetos. A idéia é prover um grupo de agentes supervisores replicados, auxiliados por agentes de migração distribuídos por toda a rede, com a função de coletar informações locais sobre a aplicação de métricas de desempenho de todos os objetos existentes nas estações. Mediante esse monitoramento, o grupo de agentes supervisores decide a viabilidade da migração de um determinado objeto servidor.

8 Bibliografia

[BIR93] BIRMAN, Kenneth; VAN RENESSE, R. Reliable distributed computing with the Isis Toolkit. IEEE Computer Society Press. 1993.

[CAU95] CAUGHEY, S. J.; SHRIVASTAVA, S. K. Architectural support for mobile objects in large scale distributed systems. Proceedings of the IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS), Lund, Sweden, Aug. 1995. http://arjuna.ncl.ac.uk/arjuna/papers/shadows-iwooos95.ps.

[FAB95] FABRE, Jean-Charles; NICOMETTE, Vincent; PÉRENNOU, Tanguy. Implementing fault tolerant applications using reflexive object-oriented programming. Proceedings of the 25th IEEE International Symposium on Fault-Tolerant Computing, Pasadena, USA, June 1995. pp.489-498.

[GUE97] GUERRAOUI, R.; SCHIPER, A. Software-based replication for fault tolerance. IEEE Computer, V.30, N. 4. pp.68-74. April, 1997.

[HAY98] HAYDEN, Mark G. The Ensemble System. A dissertation presented to the Faculty of the Graduate School of Cornell University, Jan. 1998. http://www.cs.cornell.edu/Info/Projects/HORUS.

[ING95] INGHAM, D. B.; LITTLE, M. C; CAUGHEY, S. J. and SHRIVASTAVA, S. K. W3Objects: Bringing Object-Oriented Technology to the Web. World Wide Web Journal, Issue 1, pp. 89-105: Proceedings of the 4th International World Wide Web Conference, Boston, USA, Dec. 1995. http://w3objects.ncl.ac.uk.

[LYN94] LYNCH, N.; MERRIT, M.; WEIHL, W.; FEKETE, A. Atomic Transactions. Morgan Koffmann, 1994.

[MAF96] MAFFEIS, Silvano. PIRANHA – A Hunter Crashed CORBA Objects. Jan. 1996. http://www.cs.cornell.edu/Info/People/maffeis/electra.html.

[RUB96] RUBIN, Ryan.; KISSIMOV, Valentin. Object Migration in the Distributed Computing Environment. ECCOP 96. II Workshop on Mobility and Replication. Linz, Austria. July 1996. http://www.diku.dk/distlab/workshops/wmr96/pgm.html.

[SCH93] SCHNEIDER, F. B. Replication management using the state machine approach. In: MULLENDER, Sape (Ed.). Distributed Systems. 2. ed., New York: ACM Press, 1993. p. 169-198.

[STE97] STEEN, Maarten van; HOMBURG, Philip; TANENBAUM, Andrew S. The architectural design of Globe: a wide-area distributed system. Internal report IR-422. March 1997. http://www.cs.vu.nl/~steen/globe/publications.html.

[VAY98] VAYSBURD, Alexey. Building reliable interoperable distributed objects with the Maestro Tools. A dissertation presented to the Faculty of the Graduate School of Cornell University, May 1998.