Sincronia Virtual Otimista em Servidor de Grupos CORBA

Gláucia Oliveira Dias[1], Alba Cristina Magalhães Alves de Melo[1]

[1]Departamento de Ciência da Computação – Universidade de Brasília (UNB)
Caixa Postal 4466 – 70910-900 – Brasília – DF – Brasil
{glaucia,albamm}@cic.unb.br

Resumo

Os grupos de objetos dinâmicos permitem que os objetos entrem ou saiam do grupo quando desejarem. Cada vez que a composição do grupo se altera, uma nova visão é criada. Nesse contexto, a sincronia virtual é uma propriedade muito útil, pois garante que, se dois processos permanecem na mesma visão após uma troca de visão, o mesmo conjunto de mensagens será entregue a eles. A maioria dos protocolos que implementam a sincronia virtual suspendem a atividade de mensagens regulares enquanto a visão está mudando. A sincronia virtual otimista (OVS), ao contrário, permite que mensagens regulares sejam enviadas e recebidas por uma estimativa dos membros que comporão a próxima visão enquanto a visão está se alterando. Ao se estabelecer a nova visão, as mensagens otimistas são avaliadas e somente são entregues as mensagens para as quais a estimativa é correta. Neste artigo, nós propomos e avaliamos uma abordagem para integrar a OVS em um serviço de grupos CORBA chamado OGS (Object Group Service). Os resultados obtidos mostram que a sincronia virtual otimista oferece ganhos muito bons de desempenho em sistemas onde a troca de visão é frequente.

1. Introdução

Os Grupos de Objetos são uma abstração poderosa que permite que um conjunto de objetos seja invocado como uma unidade. Ao invés de enviar mensagens a objetos específicos, a aplicação envia mensagens ao grupo e o serviço de grupo as entrega a todos os membros do grupo, de maneira consistente. Assim, grupos de objetos oferecem uma interface de programação de fácil uso e, além disso, são particularmente úteis para construir aplicações distribuídas tolerantes a falhas.

Os grupos de objetos são geralmente dinâmicos, ou seja, os objetos podem entrar ou sair do grupo a qualquer momento. Neste cenário, o servidor de grupos deve possuir um serviço de membros, responsável por manter o conjunto atual de membros ativos de cada grupo. Cada vez que o conjunto de membros ativos é alterado, cria-se uma nova visão [2], que será gerenciada pelo serviço de membros.

Os serviços de grupo interagem com as aplicações por ao menos três eventos: envio (*send*), recepção (*receive*) e visão (*view*). Quando uma aplicação deseja invocar um grupo, ela executa o evento *send*. As mensagens são recebidas por uma aplicação através do evento *receive*. Finalmente, o evento *view* notifica a aplicação que uma nova visão foi estabelecida.

Geralmente, os serviços de comunicação de grupo oferecem alguma variante da propriedade de sincronia virtual, intercalando notificações de mudança de visão com mensagens regulares. Na sincronia virtual, todo evento está associado com a visão na qual ele ocorre [8]. Sendo assim, em implementações tradicionais, a atividade de mensagens é suspensa sempre que uma nova visão está sendo estabelecida.

A maioria dos serviços de grupos de objetos existentes na literatura decidem sobre a composição de uma nova visão com um protocolo de consenso [2]. Como o tempo necessário para se chegar a um consenso é considerável, já que no caso geral a decisão é tomada em no mínimo duas rodadas [2], o tempo no qual a atividade de mensagens regulares é suspensa pode ser considerado alto.

A Sincronia Virtual Otimista (OVS) foi proposta por [3] e se propõe a reduzir a latência da entrega de mensagens (*message delivery latency*) permitindo que um novo tipo de mensagens – mensagens otimistas – sejam enviadas e recebidas enquanto a visão está mudando. As mensagens otimistas são enviadas e recebidas por um conjunto de membros que o serviço de grupo estima que estarão presentes na próxima visão. Quando a nova visão é estabelecida, o sistema decide se as mensagens otimistas devem ser entregues avaliando um predicado booleano definido pela aplicação. Se o predicado for verdadeiro, a mensagem é entregue. Caso contrário, a mensagem é descartada e a aplicação que enviou a mensagem é notificada.

A Sincronia Virtual Otimista já foi integrada por [8] em um serviço de grupos chamado Transis [6]. Os resultados apresentados neste trabalho são muito similares aos nossos. No entanto, Transis é um serviço de grupos proprietátio, que roda somente em estações de trabalho Sun e não segue o padrão CORBA. Tais características fazem com que a sua portabilidade seja difícil bem como a execução em múltiplas plataformas.

Neste artigo, é proposta a integração entre o OVS e um serviço de grupos CORBA para se avaliar os benefícios que a sincronia virtual otimista poderia oferecer neste ambiente. O OVS foi integrado ao Object Group Service (OGS) [2] e foi implementado como um modo de execução adicional do OGS cujo objetivo é reduzir a latência de entrega de mensagens quando a visão está se alterando. Ao nosso conhecimento, esta é a primeira tentativa de se integrar o OVS em um serviço de grupos CORBA.

Nossos resultados mostram que, para um grupo com inicialmente 3 membros e algumas mudanças de visão enquanto um dos membros envia constantemente mensagens ao grupo, obtivemos uma redução de 42% do tempo de execução, para mensagens de 4KB, quando comparado com o protocolo original do OGS. Além disso, foi observado que o tempo de execução das aplicações que usam OVS não é tão suscetível à frequencia de troca de visão quando comparado com as aplicações que usam o protocolo nativo do OGS.

O restante deste artigo está organizado da seguinte maneira. A seção 2 descreve sucintamente os grupos de objetos. A seção 3 apresenta a Sincronia Virtual Otimista. O Serviço de Grupos OGS é descrito na seção 4 e a seção 5 descreve como o OVS foi integrado ao OGS. Alguns resultados experimentais são apresentados e discutidos na seção 6. Por fim, a seção 7 conclui o artigo e apresenta trabalhos futuros.

2. Grupos de Objetos Distribuídos

Um grupo é um conjunto de objetos (ou processos) que agem coletivamente[13]. Pode ser classificado segundo diversas características:

1. *Grupos estáticos ou dinâmicos:* grupos estáticos são criados com um número fixo de membros, que não muda durante o tempo de vida do grupo. Grupos dinâmicos são grupos cuja composição muda ao longo da existência do grupo, i. e., membros podem entrar ou sair do grupo quando desejarem.
2. *Grupos fechados ou abertos:* em grupos fechados, somente os mebros podem enviar mensagens ao seu grupo. Em grupos abertos, qualquer processo ou objeto pode enviar mensagens ao grupo.
3. *Grupos peer ou hierárquicos:* em grupos *peer*, todos os processos são iguais e todas as decisões são tomadas coletivamente. Em grupos hierárquicos, as decisões são tomadas por um único processo, chamado coordenador.

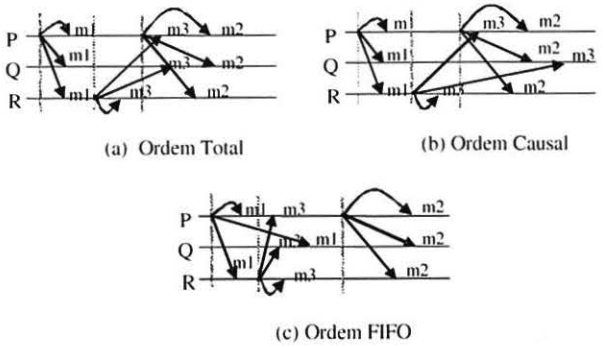
Os Serviços de Grupo gerenciam grupos de objetos e fornecem primitivas para que mensagens sejam enviadas e recebidas por todos os membros do grupo[14], sem que seja necessário que a aplicação tenha conhecimento da identidade, localização ou da quantidade de membros individuais do grupo. Tipicamente integram dois serviços: serviço *multicast* e serviço de membros[12].

O objetivo do *multicast* de grupo é permitir que cada objeto receba cópias das mensagens enviadas ao grupo, com algumas garantias de entrega. Estas garantias incluem a concordância sobre o conjunto de mensagens que todo processo no grupo deverá receber e sobre a ordem na qual as mensagens devem ser entregues[5].

O *multicast* de grupo deve prover semântica bem definida sobre a ordem na qual as mensagens serão entregues. As semânticas mais comuns são ordenamento total, causal e FIFO [5]. Sendo *multicast(g,m)* a operação que envia a mensagem *m* a todos os processos do grupo *g*, as semânticas são definidas da seguinte forma:

1. *FIFO*: Caso um processo correto execute *multicast(g,m)* e depois *multicast(g,m')*, então todo processo correto que entregar *(deliver) m'* entregará *m* antes de *m'*.
2. *Causal*: Se *multicast(g,m)* $\rightarrow$ *multicast(g,m')*, onde $\rightarrow$ é a ordem "acontece antes" definida por [9] induzida somente pelas mensagens enviadas entre os membros de *g*, então todo processo correto que entregar *m'* entregará *m* antes de *m'*.
3. *Total*: Caso um processo correto entregue a mensagem *m* antes de entregar *m'*, então qualquer outro processo correto que entrega *m'* entregará *m* antes de *m'*.

A figura 1 ilustra estes ordenamentos para o caso de 3 processos (P, Q e R). Em (1.a), vemos que em todos os processos P, Q e R, a ordem de entrega é a mesma: m1, m3 e depois m2, o que respeita a ordem total. Em (1.b), existe uma relação de causalidade entre m1 e m3, já que m3 só é enviada depois do recebimento de m1. Sendo assim, no ordenamento causal, a ordem m1 $\rightarrow$ m3 deve ser respeitada em todos os processos. No entanto, como os envios de m2 e m3 não são regidos pela causalidade, ordens de entrega diferentes podem ser observadas em P e Q. Em (1.c), somente as mensagens m1 e m2 deve ser entregues na ordem de envio, pois o ordenamento FIFO exige que mensagens enviadas pelo mesmo processo sejam entregues na mesma ordem.



(a) Ordem Total (b) Ordem Causal

(c) Ordem FIFO

Figura 1. *Multicast* total, causal e FIFO.

O serviço de membros mantem uma relação dos membros atuais do grupo. Cada vez que um membro entra ou sai do grupo, uma nova visão é criada. Uma visão contem os membros ativos do grupo em um dado momento.

Em serviços de grupo, os eventos de envio e recepção de mensagens ocorrem no contexto de visão. A maioria dos serviços de grupos oferecem uma propriedade chamada sincronia virtual, que requer que dois processos que participam nas mesmas duas visões consecutivas *i* e *j*, entreguem o mesmo conjunto de mensagens na visão *i* [12,14].

A maioria das variantes da sincronia virtual especificam que uma mensagem deve ser entregue na visão na qual ela foi enviada. Esta propriedade é chamada *group awareness* [12] ou *Sending View Delivery* [14]. A *Group awareness* é uma propriedade importante para aplicações que desejam reduzir a quantidade de informação de contexto que é enviada em cada mensagem e a quantidade de computação necessária para processar uma mensagem.

3. Sincronia Virtual Otimista

Os serviços de grupos interagem com as aplicações através de três tipos de eventos [12]: *send*, *receive*, and *view*. Um evento *send* é emitido da aplicação para o serviço de grupos para que uma mensagem seja enviada. Um evento *receive* é emitido do serviço de grupo para aplicação para que uma mensagem seja recebida. Um evento *visão* é enviado pelo serviço de grupos para notificar a aplicação de que uma troca de visão está sendo feita. Existe ainda um quarto evento que ocorre depois que a aplicação recebeu a mensagem. Este evento é chamado entrega *(deliver)* e é executado pela aplicação para processar a mensagem.

Para garantir a *group awareness*, a maioria dos serviços de comunicação de grupo impede que os processos enviem e recebam mensagens enquanto uma visão está se alterando[12]. Para implementar este bloqueio de mensagens, os eventos *block* e *flush* são adicionados à interface. O evento *block* é enviado pelo serviço de grupos à aplicação para informar que uma troca de visão está ocorrendo. A aplicação responde com o evento *flush* [14].

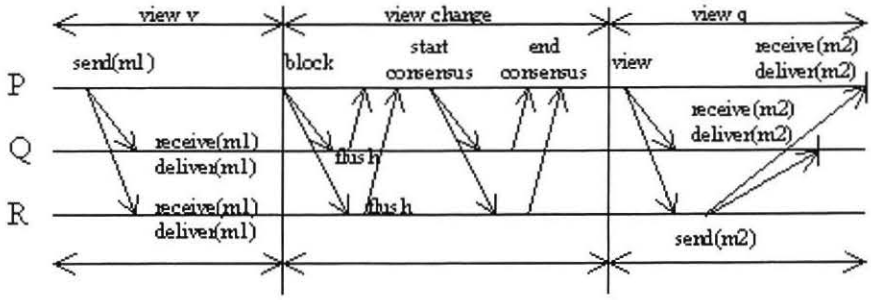
A Sincronia Virtual Otimista (OVS) é um tipo de comunicação de grupo que oferece a *group awareness* através de uma abordagem otimista, permitindo que as aplicações enviem e recebam mensagens mesmo durante a troca de visão[12]. No OVS, o evento *block* é substituído pelo evento *visão otimista* (*optView*), que contem uma estimativa do conjunto de membros que pertencerão à próxima visão[12]. Quando a aplicação recebe o evento *optView*, ela envia o evento *flush* e entra no modo otimista. Neste modo, as aplicações ainda recebem mensagens que foram enviadas na visão que está sendo terminada e podem enviar mensagens otimistas que serão entregues na próxima visão. Quando o serviço de grupos entregar uma nova visão à aplicação, a aplicação volta ao modo normal e envia um evento *viewAck* ao serviço de grupo para marcar o final do modo otimista. No modo normal, a aplicação envia mensagens regulares que serão entregues na mesma visão.

Quando uma nova visão é estabelecida, o serviço de grupo verifica se as mensagens otimistas devem ser entregues ou não. Isso é feito através da avaliação de um predicado, *MessageCondition*, para cada mensagem otimista. Caso o predicado seja verdadeiro, a mensagem é entregue. Caso contrário, a mensagem é descartada e o evento *discardedMessage* é enviado à aplicação.

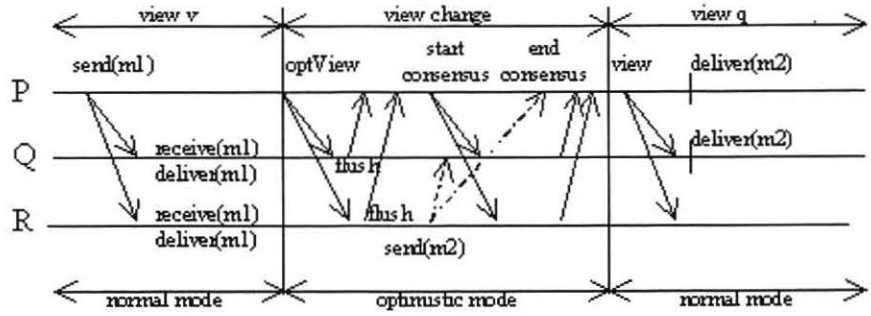
A figura 2 [15] ilustra a diferença entre um sistema que oferece a sincronia virtual da maneira tradicional (2.a) e um sistema que usa a sincronia virtual otimista (2.b). Inicialmente, a visão ativa é a visão *v*. Quando uma visão está estabelecida, os processos podem enviar e receber mensagens, que são imediatamente entregues à aplicação, respeitadas as restrições de ordem.

Quando o processo P notifica a mudança de visão (2.a), ele envia um evento *block* para cada membro, que é respondido com um evento *flush*. Depois deste momento, nenhum processo pode enviar mensagens e o consenso é iniciado para decidir a nova visão. Após o término do consenso, os processos recebem a nova visão e podem enviar e receber mensagens.

Na figura 2.b, o processo P notifica a troca de visão enviando o evento *optview* aos membros. Os membros entram no modo otimista. Neste modo, podem enviar mensagens que são marcadas como otimistas, pois serão recebidas pelos processos, porém não serão entregues [12]. Após o término do consenso, os processos recebem a nova visão e retornam ao modo normal. Neste momento, as mensagens otimistas recebidas para as quais o predicado é verdadeiro são entregues à aplicação.



(a) View change in a traditional protocol



(b) View change in OVS

Figura 2. Comportamento da Troca de Visão em dois protocolos que oferecem a sincronia virtual [15].

4 O Serviço de Grupos OGS

O Serviço de Grupos OGS (*CORBA Object Group Service*) foi proposto por[7]. A arquitetura proposta no OGS é composta por 4 serviços, que incluem:

1. *Serviço de Mensagens*: implementa comunicação ponto a ponto e multicast não blocante e confiável.
2. *Serviço de Monitoramento*: monitora objetos e detecta falhas.
3. *Serviço de Consenso*: permite que um conjunto de objetos alcance uma decisão em comum acordo.
4. *Serviço de Gerenciamento de Grupo*: oferece multicast de grupo e gerencia os membros.

No restante desta seção, somente os serviços de consenso e gerenciamento de grupo serão discutidos, pois o projeto da sisncronia virtual otimista para o OGS envolveu diretamente somente estes serviços. Informações sobre os outros serviços podem ser encontradas em [7].

O Serviço de Consenso permite que um conjunto de membros de um grupo entre em acordo, resolvendo o problema clássico de consenso em sistemas distribuídos [3]. Este problema pode ser definido em termos de duas primitivas: *propose(v)* e *decide(v)*, onde *v* é um valor. Todos os participantes corretos podem propor um valor inicial e devem concordar sobre um valor final, relacionado aos valores propostos.

O consenso é utilizado no OGS para implementar o multicast totalmente ordenado e para gerenciar os membros do grupo. Cada instância do consenso decide sobre um conjunto totalmente ordenado de mensagens ou sobre os membros que devem entrar ou sair do grupo (troca de visão).

A implementação do consenso no OGS é baseada no algoritmo de Chandra and Toueg [4], que usa o paradigma do coordenador rotativo [2] e se executa em rodadas assíncronas. Em cada rodada, um processo diferente age como coordenador. Na primeira rodada, estimativas são enviadas ao coordenador. A seguir, o coordenador propõe um valor e o envia aos membros (*propose*). Caso os membros concordem com o valor proposto, enviam um

ACK ao coordenador e, finalmente, o coordenador envia a decisão final ao membro (*decide*).

O OGS oferece também uma versão otimizada do consenso. No consenso otimizado, o coordenador não espera a maioria das propostas mas simplesmente faz a proposta com a sua própria estimativa [7], economizando, assim, uma rodada. Caso a estimativa do coordenador esteja incorreta, o protocolo tradicional do consenso é executado.

O Serviço de Gerenciamento de Grupo é o principal serviço do OGS pois implementa o *multicast* e a gerência de membros do grupo [7].

O multicast de grupo oferece suporte para o envio de mensagens *multicast* aos membros do grupo. Possui primitivas que permitem que o grupo seja invocado, ao invés de objetos paraticulares. Tanto os grupos abertos como fechados são implementados pelo OGS. As primitivas de *multicast* no OGS são classificadas de acordo com o seu grau de confiabilidade e de garantias de ordenamento.

O *multicast* confiável garante que todos os membros corretos do grupo entreguem o mesmo conjunto de mensagens [2]. Tanto o *multicast* confiável como o não confiável são oferecidos pelo OGS. As seguintes garantias de ordem são oferecidas pela versão corrente do OGS: nenhuma garantia (*unordered*), FIFO, total. Dentre estes ordenamentos, somente o ordenamento total oferece a sincronia virtual.

Um serviço de membros (*Group membership*) gerencia os objetos do grupo. O OGS suporta grupos dinâmicos, assim, objetos podem entrar ou sair do grupo em qualquer momento ou podem ser removidos do grupo por causa de falhas. Os objetos que desejam entrar no grupo contactam o serviço de membros, que atualiza a lista de membros ativos do grupo.

O serviço de membros possui dois protocolos: *view change* e *state transfer*. O protocolo de mudança de visão é executado toda vez que a composição do grupo é alterada. Ele garante que todo membro correto do grupo receba uma nova lista contendo os atuais membros do grupo. As trocas de visão são totalmente ordenadas entre si. O protocolo de transferência de estado é uma operação atômica que acontece durante a troca de visão, quando um membro entra em um grupo. Consiste em entregar o estado correto do grupo ao novo membro. Este protocolo garante que o estado contido em todos os membros do grupo permanece consistente através das trocas de visão. O protocolo de troca de visão somente termina após o término da transferência de estado[7].

5. Integração da Sincronia Virtual Otimista no OGS

Para se fazer a integração entre a sincronia virtual otimista e o OGS [7], as seguintes mudanças foram feitas no OGS:

1. Alteração da estrutura da mensagem para incluir a identificação de mensagens otimistas.
2. Adição de código para identificar os modos normal e otimista.
3. Adição do tratamento das mensagens otimistas.

1. *Modificação da Estrutura da Mensagem:* as mensagens do OGS são enviadas em uma estrutura de dados, que é composta pela identificação da mensagem, identificação do emissor, número da resposta, semântica de ordenamento, versão e dados. Um novo campo foi incluído, tipo da mensagem, que permite que o sistema faça a distinção entre uma mensagem otimista e uma mensagem normal. Este campo também é utilizado pelo serviço de gerenciamento de grupos para informar aos membros da mudança entre o modo normal e otimista e vice-versa. A nova estrutura da mensagem é mostrada abaixo.

```
    /* Definition of Message Type. Included
for optimistic  messages */
        enum MessageType{
            NORMAL,
            OPTIMISTIC,
            CHANGE_OPTIMISTIC,
            CHANGE_NORMAL
        };

    /* Internal message. */
      struct Message {
        MessageId id_;
        GroupAccessManager sender_;
        NumReplies nb_replies_;
        Ordering ordering_;
        MessageType type_;
        unsigned long view_version_;
        any data_;
    };
```

2. *Identificação dos modos normal e otimista*: no OGS original, quando uma troca de visão é reconhecida, uma mensagem totalmente ordenada é enviada aos membros do grupo. Neste momento, o consenso é executado para decidir sobre a nova visão. Na nossa proposta, os membros começam no modo normal e, quando recebem a mensagem de troca de visão, entram no modo otimista. Após o término do consenso, o modo otimista é terminado e os membros voltam ao modo normal.

3. *Tratamento de mensagens otimistas:*
   3.1 Antes de enviar uma mensagem, o processo verifica se o modo corrente é o otimista e, neste caso, marca a mensagem como otimista.

3.2 Quando uma mensagem é recebida por um processo, três situações podem ocorrer: (a) se o tipo da mensagem é normal, ela é tratada segundo o OGS original; (b) caso o membro esteja no modo otimista, a mensagem é colocada em um buffer de mensagens otimistas; (c) se o membro está no modo normal e uma mensagem otimista é recebida, o método *boolean message_condition (newView, optView, msg)* é executado. Através deste método, a aplicação determina se a mensagem pode ou não ser entregue.

3.3 Quando uma nova visão é entregue, cada mensagem contida no buffer de mensagens otimistas é analisada para verificar se o emissor é membro da nova visão. Caso afirmativo, o método *boolean message_condition (newView, optView, msg)* é executado. Caso contrário, a mensagem é descartada e a aplicação emissora é notificada.

6. Resultados Experimentais

O OVS foi implementado utilizando o JDK 1.1.8 e o Visibroker for Java 3.2 e integrado ao OGS. O sistema operacional utilizado foi o Microsoft Windows NT 4.0.

Para avaliar nossa proposta, foram executados experimentos em 4 máquinas: 3 Athlon 900MHz, com 128MB RAM e um Pentium II 550 MHz que atuou exclusivamente como servidor de nomes. Todas as máquinas estavam conectadas por um hub Ethernet 10Mb/sec e os testes foram executados em modo dedicado.

Os testes foram feitos para avaliar duas características: o overhead introduzido pelo OVS quando não há trocas de visão e os ganhos de desempenho obtidos a medida em que a freqüência de troca de visão aumenta. Para tanto, foi medido o tempo total de execução para que um cliente envie 50 mensagens a um grupo composto inicialmente por 3 membros nos seguintes casos: sem troca de visão, com duas e com três trocas de visão intercaladas com mensagens regulares. Os protocolos considerados foram a ordem total do OGS (TO), a ordem total do OGS com consenso otimizado (OTO) e a sincronia virtual otimista (OVS). Os resultados obtidos para mensagens de 4KB e 8KB são mostrados nas tabelas 1 e 2, respectivamente. As figuras 3 e 4 apresentam estes resultados em um gráfico.

Tabela 1. Tempo de execução (ms) para o envio de 50 mensagens a 3 membros com 0, 2 e 3 trocas de visão intercaladas (tamanho da mensagem = 4KB)

Message Ordering Semantics	Number of view changes Zero	2	3
Total Order (TO)	13229	22072	26398
Optimized Total Order (OTO)	7581	19107	25947
Optimistic Virtual Synchrony (OVS)	13250	14872	15071

Tabela 2. Tempo de execução (ms) para o envio de 50 mensagens a 3 membros com 0, 2 e 3 trocas de visão intercaladas (tamanho da mensagem = 8KB)

Message Ordering Semantics	Number of view changes Zero	2	3
Total Order (TO)	19157	27049	29232
Optimized Total Order (OTO)	14410	24996	27029
Optimistic Virtual Synchrony (OVS)	19257	22503	26508

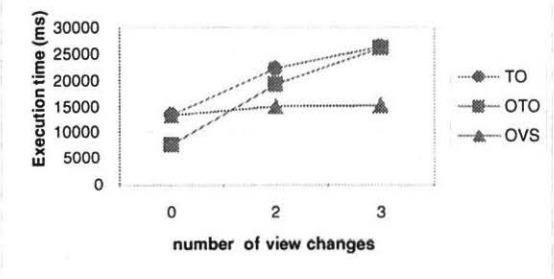


Fig. 3. **Tempo de Execução x No. de trocas de visão para mensagens de 4K**

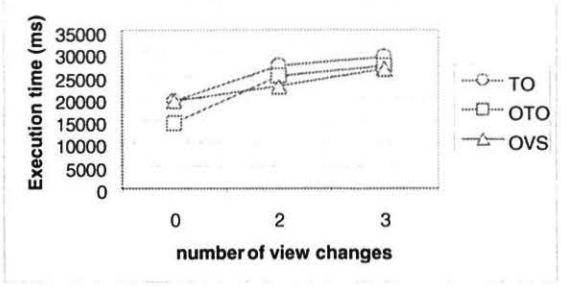


Fig. 4. **Tempo de Execução x No. de trocas de visão para mensagens de 8K**

Os resultados apresentados nas figuras 3 e 4 mostram que o OVS apresenta um melhor desempenho que os outros dois protocolos, quando trocas de visão ocorrem. Este comportamento pode ser facilmente explicado já que, em TO e OTO, a atividade de mensagens regulares é suspensa enquanto uma troca de visão está sendo decidida. Com o OVS, ao contrário, as mensagens podem ser enviadas e recebidas enquanto a visão está mudando; somente a entrega de mensagens é retardada.

Pode também ser observado na figura 3 que, à medida que o número de trocas de visão aumenta, o OVS apresenta baixo acréscimo no tempo de execução total da aplicação. Os outros dois protocolos são mais sensíveis à frequência de trocas de visão e, por esta razão, são menos adequados quando a escalabilidade é considerada.

O ganho em desempenho obtido pelo OVS com mensagens de 8KB (figura 4) foi menor que o ganho obtido no caso precedente (figura 3). Analisando este cenário, pode ser observado que o tamanho da mensagem é o dobro do caso precedente e o tempo necessário para se obter o consenso não foi alterado, já que temos o mesmo número de membros. Sendo assim, menos mensagens otimistas são enviadas enquanto ocorre a troca de visão e, conseqüentemente, o ganho em desempenho apresentado pelo OVS é menor, porém ainda existe.

7. Conclusões e Trabalhos Futuros

Neste artigo, foi discutida e avaliada uma abordagem para a integração entre a sincronia virtual otimista e um serviço de grupos CORBA (OGS). Nós sustentamos que, por permitir que a atividade de mensagens regulares ocorra simultaneamente com as trocas de visão, a sincronia virtual otimista oferece um bom compromisso entre o desempenho e as garantias oferecidas à aplicação distribuída.

A sincronia virtual otimista foi proposta como um novo modo de execução no OGS. As modificações foram feitas de maneira modular, o que trouxe mudanças mínimas na estrutura original do OGS. Os resultados obtidos com a nossa abordagem mostram melhorias de desempenho muito boas quando comparados aos protocolos originais do OGS que oferecem sincronia virtual: ordem total e ordem total otimizada.

Como trabalhos futuros, pretendemos implementar um serviço de near-video on demand sobre o OVS+OGS para avaliar os ganhos que seriam obtidos com uma aplicação real onde a freqüência de trocas de visão tende a ser média-alta. Também como trabalho futuro, pretendemos investigar as modificações que devem ser feitas na nossa abordagem para que seja possível que o OGS+OVS se execute em um ambiente de computação sem fio (wireless). Mais especificamente, sera estudado como o OVS poderia ser utilizado de maneira consistente em uma rede particionada, incorporando ao mesmo o conceito de sincronia virtual estendida [10].

8. Referências

1. Agarwal, D. A., Chevassut, O., Thompson, M. R., Tsudik, G.: An Integrated Solution for Secure Group Communication in Wide-Area Networks. In Proceedings of the 6th IEEE Symposium on Computers and Communications, Hammamet, Tunisia (July 2001)
2. Attiya, H., Welch, J.: Distributed Computing: Fundamentals, Simulations and Advanced Topics. McGraw Hill Pub Co., England (1998) 451
3. Barborak, M., Malek, M., Dahbura, A.: The consensus problem in distributed computing. ACM Computing Surveys, Vol.25(2). (June 1993) 171-220
4. Chandra, T. D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM, Vol.43(2). (1996) 225-267
5. Coulouris, G., Dollimore,J., Kindberg, T.: Distributed Systems Concepts and Design. (2000) 153-158, 436-451, 556-565
6. Dolev, D., Malki, D.: The Transis Approach to High Availability Cluster Communication. Communications of the ACM (April 1996)
7. Felber, P.: The CORBA Object Group Service. Lausanne, EPFL (1998)
8. Keidar, I., Khazan,R.: A Client-Server Approach to Virtually Synchronous Group Multicast: Specifications and Algorithms. Proc. of the Int. Conf. on Distributed Computing Systems (2000)
9. Lamport, L.: Time, Clocks and the Ordering of Events in a Distributed System. Comunications of the ACM, Vol.21. (July 1978) 558-564.
10. Moser, L.E., Amir, Y., Melliar-Smith, P.M., Agarwal, D.A.: Extended Virtual Synchrony. The 14th IEEE International Conference on Distributed Computing Systems (ICDCS) (June 1994) 56-65.
11. OMG: The Common Object Request Broker: Architecture and Specification. OMG (February 1998)
12. Sussman, J., Keidar, I., Marzullo, K.: Optimistic Virtual Synchrony. MIT Technical Report MIT-LCS-TR –792 (November 1999)
13. Tanenbaum, A. S.: Distributed Operating Systems (1995) 99-115
14. Vitenberg, R., Keidar, I., Chockler, G. V., Dolev, D.: Group Communication Specifications: A Comprehensive Study. MIT Technical Report MIT-LCS-TR-790 (September 1999).
15. Dias, G. O., Melo, A. C. M., "Integrating Optmistic Virtual Synchrony to a CORBA Object Group Service", In: Proc. Of the 4th DOA (Distributed Objects and Applications), LNCS 2519, October, 2002, Irvine, USA, p711-722.