View Synchronous Multicast sobre UDP

Ana Beatriz Cruz, Daniel Granhão, Isabel Oliveira
{up201403564,up201406280,up201403560}@fe.up.pt
Sistemas Distribuídos
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
18 de Janeiro de 2019

***Resumo*— *View/Virtual Synchronous Communication* é um paradigma de comunicação baseada em grupos, que facilita o desenvolvimento de serviços baseados em redundância tolerantes a falhas.**

São propostas duas soluções, que constituem variações do algoritmo originalmente proposto por Birman, Kenneth [et al.], mas baseadas em comunicação *Multicast* UDP. A primeira implementa uma alternativa de entrega imediata de mensagens, enquanto que a segunda implementa entrega diferida após garantia de estabilidade das mensagens. Ambas as soluções garantem a propriedade de *Virtual Synchrony*.

A avaliação das implementações é feita recorrendo a simulação e usando como métrica o tempo de mudança de vista, sendo feita uma comparação entre as duas abordagens, evidenciando-se o *trade-off* entre a rapidez de mudança de vista e as garantias de entrega das mensagens.

I. Definição do Problema

A. Contextualização

Tolerância a falhas, a capacidade de um sistema continuar a funcionar corretamente mesmo quando alguns dos seus componentes falham, é uma propriedade indispensável no desenvolvimento de um Sistema Distribuído. Esta pode ser obtida usando técnicas que aproveitam a redundância inerente a este tipo de sistemas.

State Machine Replication constitui uma abordagem genérica ao desenvolvimento de sistemas tolerantes a falhas e consiste em projetar uma máquina de estados, replicá-la em diferentes nós e executar a mesma sequência de operações em todas as réplicas. Porém, garantir a execução da mesma sequência de operações, obriga a que haja alguma forma de consenso, o que pode ser obtido com *multicast* atómico.

B. Objetivos e Trabalho Desenvolvido

O desafio consistiu, então, em desenvolver um protocolo de *multicast* fiável, à semelhança do que é feito em [1], mas baseado em UDP, ao invés de em TCP. Todo o código foi desenvolvido na linguagem JAVA.

View Synchronous Communication engloba dois serviços, *membership service* e *group communication service*, em particular, comunicação *multicast* fiável. É de notar que o âmbito deste projeto contemplou apenas o *design* do segundo serviço.

Assim, o trabalho desenvolvido englobou o *design* e implementação de duas alternativas de *View Synchronous Multicast*, escolha e implementação de métricas adequadas à avaliação deste tipo de protocolos, desenvolvimento de simulações, recolha de dados e análise crítico-comparativa dos mesmos.

C. Propriedades

View Synchronous Multicast baseia-se no conceito de vista, sendo que esta é univocamente definida por um identificador V_i e é composta por um conjunto de processos.

Implementar este protocolo implica garantir as seguintes propriedades:

- *Virtual Synchrony* - variação da propriedade de *Agreement*, podendo também ser considerada uma propriedade de *Atomicity*. Afirma que se dois processos mudam da vista V_1 para V_2, então entregam o mesmo conjunto de mensagens em V_1.
- *Self Delivery* - variação da propriedade de *Validity*. Se um processo correto envia uma mensagem, então entrega-a.

D. Modelo

A implementação desenvolvida tem por base o seguinte modelo:

- Ordem - as implementações efetuadas não garantem qualquer tipo de ordenação.
- Grupo
  - dinâmico - a *membership* do grupo pode ser alterada, à medida que processos de juntam ou saem do grupo, voluntaria ou forçadamente
  - fechado - não há *overlap* entre grupos *multicast* e o processo que envia mensagens *multicast* pertence ao grupo
- Falhas
  - *crash* - assume-se que os processos apenas falham por *crash*, isto é, comportam-se corretamente até determinado instante, a partir do qual deixam de responder a qualquer estímulo.
  - não recuperação - assume-se que um processo em falha não tem possibilidade de recuperar.
- Canal de Comunicação
  - UDP *Multicast* - toda a comunicação é implementada em UDP *Multicast*, pelo que o canal de comunicação não é ponto-a-ponto e não é fiável, no sentido em que pode existir perda de mensagens, não há garantias de ordem e podem existir duplicados.

II. Descrição da Solução

Tendo em conta a implementação dos dois algoritmos que são em seguida descritos, foi necessário definir uma arquitetura que permitisse que cada processo tivesse acesso a um *membership service* básico.

A. Arquitetura

A arquitetura definida encontra-se esquematizada na figura 1. Esta é constituída por um controlador, que desempenha as funções básicas da interface *join* e *leave*, que permitem adicionar ou retirar processos a uma vista, respetivamente, bem como *new-view*, que permite informar todos os elementos que fazem parte de uma vista V_i, da chegada de uma nova vista V_{i+1}. Este controlador comunica com a camada *Group* de cada nó por TCP.

Cada nó possui uma camada *Group*, responsável por receber a vista mais atual e uma camada *VSM*, responsável pelo serviço de *View Synchrony*. A camada da aplicação usufrui dos serviços disponibilizados pela camada de *VSM* para poder comunicar com os nós pertencentes à vista, por *Multicast*.

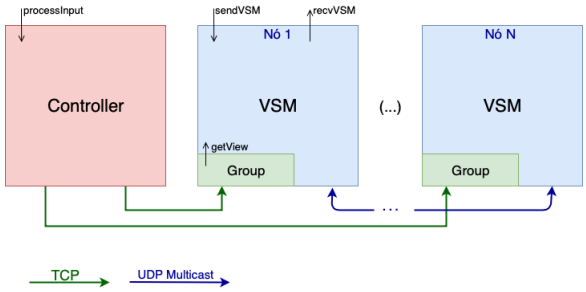

Figura 1: Arquitetura da solução proposta

B. VSM com entrega imediata - EI

Considera-se que um processo p se encontra num estado normal de funcionamento quando pode receber e enviar novas mensagens. Um processo p muda para um estado de mudança de vista quando se encontra em espera uma nova vista, V_{i+k}, para ser instalada.

Uma vez que com comunicação UDP *Multicast* as mensagens enviadas podem chegar ao destino repetidas, quando é recebida uma nova mensagem, verifica-se se é ou não repetida. Caso seja, é descartada.

Assim, o algoritmo EI executado por um processo p é o abaixo descrito:

1) Havendo uma nova vista, V_{i+k}, para ser instalada, novas mensagens não são enviadas. p verifica se há mensagens que foram entregues e que ainda estão não estáveis. Caso se verifique, p reenvia essas mensagens juntamente com os *acknowledges* das mensagens, ack_m, recebidas até ao momento. Depois das retransmissões, se ainda existirem mensagens não estáveis após um *timeout*, essas mensagens voltam a ser retransmitidas. Na situação de p não ter mensagens não estáveis, então este envia uma mensagem $flush_V$.
2) Quando um processo p recebe uma mensagem m, verifica a que vista esta pertence. Caso pertença a uma vista passada, é descartada, mas caso pertença a uma vista futura, é armazenada até que esta seja instalada, para ser posteriormente entregue. No caso da mensagem pertencer à vista atual, p confirma a receção de m, enviando um ack_m, e armazena m até à mudança de vista seguinte.
3) No caso de p receber um ack_m, verifica se já recebeu a mensagem correspondente a esse ack_m na vista em que se encontra. Esse *acknowledge* é adicionado à mensagem m e de seguida o processo verifica se m já ficou estável ou se falta receber $acks_m$ de outros nós.
4) Ao receber um $flush_V$, p compara o estado das mensagens do processo p_j que enviou esse $flush_V$, e verifica se correspondem às mesmas mensagens recebidas pelo processo p antes da chegada da nova vista. Se o estado das mensagens não for o mesmo nos dois processos, então o $flush_V$ é descartado. Caso o estado das mensagens seja o mesmo, então p guarda o $flush_V$ e verifica se já recebeu os $flush_V$ de todos os processos que pertencem a $V_{i+1} \cap V_{i+k}$. Após receber um $flush_V$, p envia ainda um ack_f a confirmar a receção do $flush_V$.
5) Durante a mudança de vista, caso p não tenha recebido os ack_f do seu $flush_V$ após um *timeout*, este retransmite o seu $flush_V$, e aguarda novamente a chegada das mensagens em falta. Quando o processo receber todos os $flush_V$ de todos os nós que pertencem a $V_{i+1} \cap V_{i+k}$ e todos os ack_f, então p pode mudar de vista.

Não é especificado um número máximo de retransmissões, pois não se encontra aqui contemplado o *membership service* e, por isso, não existe mecanismo de deteção de falhas. Caso estivesse disponível um serviço que permitisse um processo p suspeitar da falha de outro, então após algumas retransmissões de m sem obter ack_m, p suspeitaria dos processos dos quais não obteve ack.

C. VSM com entrega diferida - ED

O algoritmo ED executado por um processo p é o abaixo descrito:

1) Quando p recebe uma mensagem m relativa à vista atual, envia um ack_m que informa da receção de m e guarda a mesma até esta se tornar estável. Se m for de uma vista anterior é descartada e se for de uma vista futura é guardada para ser processada mais tarde. p guarda, também, uma cópia de todas as mensagens recebidas na vista atual para que possa descartar duplicados.
2) Quando p recebe ack_m de todos os nós na vista atual, m torna-se estável para p e m é entregue.
3) Quando p recebe uma nova vista V_{i+k}, adiciona a mesma a uma fila FIFO e enquanto a fila não está vazia, novas mensagens não são enviadas. Também envia um $flush_V$ que identifica todas as mensagens estáveis para p no momento de envio.
4) Ao receber $flush_V$ é enviado uma mensagem ack_f. Quando p recebe $flush_V$ de todos os processos em

$V_{i+1} \cap V_{i+k}$, torna estáveis e entrega quaisquer mensagens que estejam identificadas em pelo menos uma das mensagens $flush_V$. Todas as mensagens que continuam não estáveis são, então, descartadas e V_{i+k} é instalada quando forem recebidos ack_f de todos os outros nós.

Este algoritmo não apresenta a propriedade de *self-delivery*, mas garante na mesma *virtual synchrony*.

III. Resultados

Para avaliar ambos os algoritmos descritos foi usada como métrica principal o tempo de mudança de vista. Este foi medido de forma independente por cada processo, tendo como início o momento em que recebe uma nova vista e como fim o momento em que instala a vista.

A. Parâmetros variados

Foram impostas condições iniciais ao sistema e o mesmo foi iniciado imediatamente antes de mudar de vista. Nas condições iniciais foram variados 3 parâmetros:

- Número de mensagens não estáveis - N_{ne}
- Número de mensagens estáveis - N_e
- Número de processos - N_p

Em todas as execuções dos algoritmos cada processo foi iniciado com um igual N_{ne} e N_e antes da mudança de vista. As mensagens que se encontravam estáveis eram as mesmas em todos os processos e as não estáveis eram distintas. Assim, N_e = 10 significa que existem 10 mensagens estáveis comuns a todos os processos, enquanto que N_ne = 10 significa que existem 10 mensagens não estáveis por processo.

B. Condições de simulação

Todas as simulações foram realizadas no mesmo computador. O computador em questão é um Macbook (Retina, 12-inch, Early 2016) com processador 1,2 GHz Intel Core m5 e 8GB de RAM. Durante as simulações foram desligados o máximo de processos não relacionados com as mesmas.

C. Gráficos

As figuras 2, 3 e 4 representam os dados obtidos com as simulações que foram descritas.

Cada gráfico representa os resultados obtidos com os dois algoritmos implementados e, para cada um, apresenta-se, também, a curva ajustada às médias dos dados, para que melhor se possa inferir o tipo de relação entre as variáveis em análise.

IV. Análise Crítica

A escolha entre um dos dois algoritmos apresentados depende das características desejadas para o mesmo.

A. Atraso introduzido a cada mensagem

Uma das vantagens que o algoritmo EI apresenta em relação ao ED é o facto de que as mensagens são entregues imediatamente após a sua receção, enquanto que no ED são apenas entregues quando se tornam estáveis. Isto leva a que o algoritmo EI apresente um menor atraso de entrega durante a operação normal.

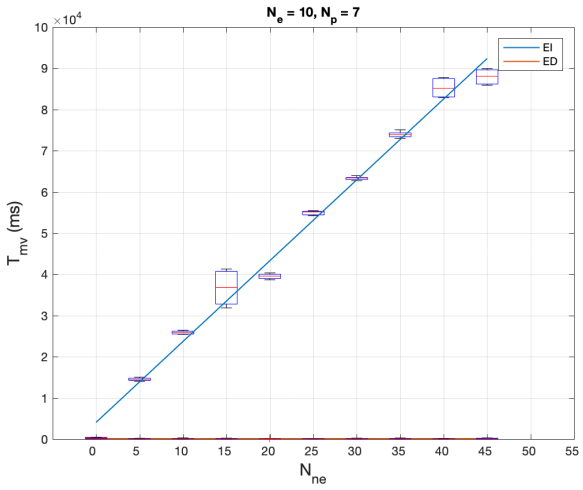

Figura 2: T_{mv} vs. N_{ne}

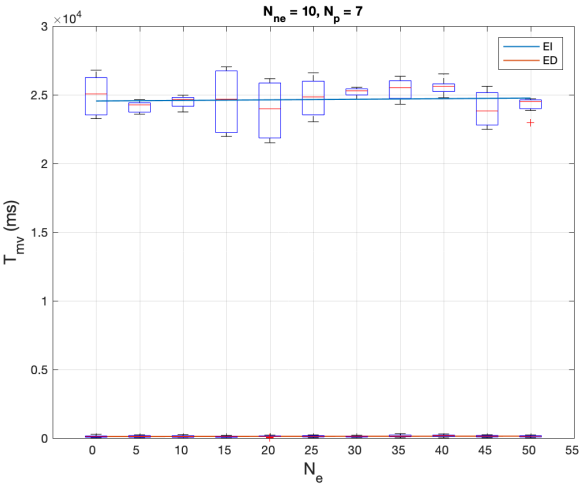

Figura 3: T_{mv} vs. N_e

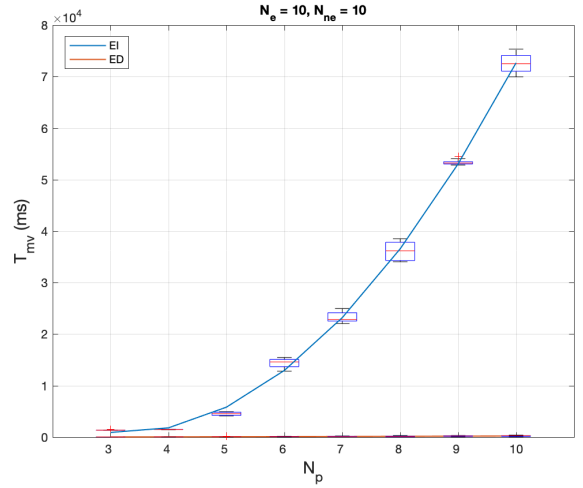

Figura 4: T_{mv} vs. N_p

B. Fiabilidade

Como foi referido na definição do algoritmo ED, ao contrário do EI, este não apresenta a propriedade de *self-delivery* e, por isso, não é garantido que uma mensagem enviada por um processo correto seja entregue. Isto leva a que seja possível que mensagens sejam perdidas mas, nesse caso, nenhum processo entrega a mensagem. Deste modo, se se pretender que a entrega seja fiável, essa propriedade terá de ser adicionada numa camada superior, caso se utilize o algoritmo ED.

C. Tempo de mudança de vista - T_{mv}

Como o algoritmo EI contempla retransmissões de mensagens não estáveis durante a mudança de vista e o ED apenas as descarta, sempre que existam mensagens não estáveis num processo, quando se inicia uma mudança de vista, o algoritmo ED será sempre mais rápido que o EI a instalar a vista, como se conseguiu confirmar nas medições efetuadas.

T_{mv} varia com o número de mensagens que foram enviadas na vista anterior e também com o número de processos.

1) Número de mensagens não estáveis - N_{ne}: no algoritmo EI, T_{mv} cresce linearmente com N_{ne}, enquanto que no ED não varia. Se se assumir que a comunicação UDP *multicast*, em que ambos se baseiam, apresenta uma probabilidade fixa de perda de pacotes, então T_{mv} irá também subir linearmente com o aumento do número de mensagens enviadas na vista anterior.

2) Número de mensagens estáveis - N_e: ambos os algoritmos demoram mais a instalar uma vista se na vista anterior tiverem sido enviadas um maior número de mensagens, mesmo que estas se encontrem estáveis em todos os processos. O atraso adicional, que decorre de um maior número de mensagens estáveis, é desprezável em ambos os algoritmos em comparação com o atraso adicional que ocorre no algoritmo EI com um maior N_{ne}.

Este aumento relativamente pequeno pode ser explicado pelo facto de as mensagens de $flush_V$ identificarem as mensagens estáveis do processo que as enviou e cada nó demorar mais a processar as mensagens $flush_V$ recebidas.

3) Número de processos - N_p: com o aumento de N_p, T_{mv} aumenta exponencialmente nos dois algoritmos. Isto deve-se ao facto de que todas as comunicações entre os processos serem realizadas em *multicast* e, por isso, o número de mensagens trocadas crescer exponencialmente.

D. Conclusão

Conclui-se, então, que, caso não seja tolerável perder mensagens, terá de se optar obrigatoriamente pelo algoritmo EI e, que se for importante reduzir a um mínimo o atraso a que cada mensagem está sujeita, também se deve tomar a mesma decisão. Por outro lado, caso a perda de mensagens seja tolerável, é vantajosa a escolha do algoritmo ED, pois leva a mudanças de vista quase imediatas em comparação com o EI, garantindo na mesma *virtual synchrony*.

V. Organização do Grupo

Cada um dos elementos do grupo contribuiu com pesos idênticos para o resultado final, ou seja, a percentagem atribuída a cada um é de 33,33%. Apesar de todos terem estado envolvidos de alguma forma em todas as tarefas, cada um dedicou-se mais para algumas em particular, como se descreve em seguida.

- Beatriz Cruz - desenvolvimento do algoritmo EI, teste do algoritmo EI, apresentação, relatório
- Daniel Granhão - desenvolvimento do algoritmo EI, desenvolvimento do algoritmo ED, apresentação, relatório
- Isabel Oliveira - implementação da simulação, recolha de resultados, apresentação, relatório

Referências

[1] Kenneth Birman, André Schiper and Pat Stephenson, "Lightweight Causal and Atomic Group Multicast", *ACM Transactions on Computer Systems*, vol. 9, no. 3, pp. 272-314, August 1991.