4
Mecanismo de Tolerância a Falhas para Sistemas de Gerenciamento de Workflow

Nos capítulos anteriores apresentamos as características de sistemas de gerenciamento de *workflow*, tipos de replicação para prover tolerância a falhas e abordagens para replicação em sistemas distribuídos. Com tudo isso que foi discutido, é possível determinar os requisitos de um mecanismo de tolerância a falhas para um sistema de gerenciamento de *workflow* e observar se é possível e qual tipo de replicação ou abordagem de sistema de replicação é o mais indicado para esse mecanismo.

Para determinar esses requisitos e especificar o mecanismo iremos considerar que as ações realizadas nos *workflows* são idempotentes, ou seja, o resultado final da realização de uma ação mais de uma vez será o mesmo que se a ação for realizada um única vez. Além disso iremos supor que não vão ocorrer erros de comunicação e toda mensagem enviada será recebida corretamente.

4.1
Requisitos

Nesta seção iremos apresentar os requisitos necessários para que um sistema de gerenciamento de *workflow* seja tolerante a falhas. Consideramos que apenas o servidor de execução precisa ser replicado. Isso acontece porque esse componente contém o motor do sistema, responsável pela execução dos processos e pelo acesso às informações relevantes a eles. Por outro lado, o cliente de modelagem não precisa ser tolerante a falhas, pois é usado na fase de modelagem dos processos e a sua interrupção não interfere na realização desses processos. Sendo assim, os requisitos não irão tratar a tolerância a falhas de recursos externos que uma ação possa utilizar.

O mecanismo deve ser capaz de lidar com operações não determinísticas. Esse requisito é necessário porque sistemas de gerenciamento de *workflow* podem modelar ações cujo resultado dependa de recursos externos a ele. Além dessas ações, os processos podem utilizar intervalos de tempo ou agendamento de tarefas, dependendo assim do relógio da máquina. Outro tipo de indeterminismo observado nesses sistemas vem da interação com usuário. Por exemplo,

comandos de usuário que ativem condições precisam ser consumidos pelo sistema para que não seja interpretado por mais tempo que o necessário. Além dessas, a interação do sistema com elementos externos pode não ser determinística, uma vez que valores lidos podem ser alterados de forma arbitrária. Essa base de dados pode apresentar valores baseados em informações do mundo real, como estoque, que são alteradas arbitrariamente.

O mecanismo deve ser capaz de lidar com operações que possam ter efeito colateral. Esse requisito está associado ao fato de ações dos *workflows* poderem realizar alterações no mundo real. Sendo assim, algumas ações só podem ser realizadas uma única vez.

O mecanismo deve ser capaz de atender de forma transparente clientes de execução. Esse requisito está associado ao fato de aplicações externas serem utilizadas para o monitoramento e controle dos processos. Essas aplicações não precisam saber que o servidor de execução é replicado e devem observar o mesmo comportamento quando acessam um servidor de execução com e sem replicação.

As réplicas deverão ser independentes do mecanismo. Esse requisito é necessário para que não haja interferência da replicação do sistema na realização dos processos, ou seja, falhas identificadas incorretamente no mecanismo não podem interferir na execução dos processos.

O tempo para recuperação deve ser minimizado. Esse requisito visa a manutenção da operação do sistema pelo maior tempo possível, mesmo na presença de falhas. Caso uma réplica falhe, as ações dos processos não podem ser significativamente atrasadas. Em processos produtivos atrasos podem ser custosos resultando, por exemplo, em multas ou perda de cliente por atrasos nas entregas. Além disso, atrasos na recuperação podem levar o sistema produtivo para níveis críticos. Por exemplo, a demora para fechar uma válvula de abastecimento de um tanque pode deixar o nível do tanque próximo ao nível máximo e causar o desligamento emergencial da produção.

O número de réplicas não pode ser fixo. Esse requisito está associado a garantia de que novas réplicas podem ser adicionadas ao sistema a qualquer momento. Essa possibilidade é necessária para casos em que novas máquinas ficam disponíveis para replicação do sistema. Dessa forma, o sistema não precisa ser interrompido para incluir uma nova réplica.

O tempo para detecção da falha deve ser adaptável. Esse requisito está associado à necessidade do mecanismo se adaptar às características da rede e do sistema que está sendo replicado. O intervalo de tempo irá determinar a frequência em que o mecanismo verifica a disponibilidade das réplicas. A definição de intervalos pequenos permite a detecção rápida da falha, mas

implica em mais ciclos de processamento gastos na execução do mecanismo e na troca de mensagens. Por outro lado, a definição de intervalos grandes deixará o sistema mais tempo no estado de falha, potencializando os danos da falha na realização dos processos.

Uma lista com os requisitos identificados pode ser vista na tabela 4.1.

Ser capaz de lidar com operações não determinísticas.
Ser capaz de lidar com operações que possam ter efeito colateral.
Ser capaz de atender de forma transparente clientes de execução.
As réplicas deverão ser independentes do mecanismo.
O tempo para recuperação deve ser minimizado.
O número de réplicas não pode ser fixo.
O tempo para detecção da falha deve ser adaptável.

Tabela 4.1: Lista de Requisitos para o Mecanismo.

4.2 Especificação

Nessa seção iremos especificar as características do mecanismo baseados nos requisitos apresentados anteriormente. Como falhas na tolerância a falhas não devem interromper a execução dos processos, iremos utilizar no mecanismo um componente replicador independente do servidor de execução. O replicador ficará encarregado das funções de tolerância a falhas, cabendo ao servidor de execução disponibilizar uma interface de acesso ao estado global. Um replicador precisará conhecer a localização do servidor de execução para poder replicá-lo. O relacionamento entre os replicadores, servidores de execução e elementos externos estão representados na figura 4.1

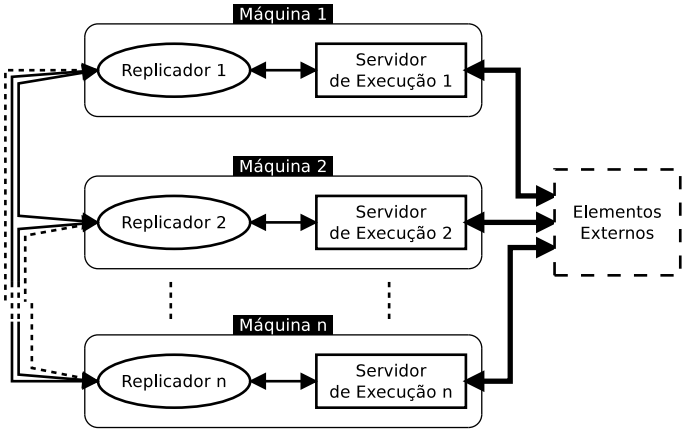

Figura 4.1: Diagrama com relacionamento entre replicadores, servidores de execução e elementos externos.

Como discutido em 2, o motor do servidor de execução lida com operações não-determinísticas e operações com efeitos colaterais. Sendo assim, a replicação do estado global deve ser passiva, apenas um servidor de execução realiza as operações e atualiza o estado dos demais. Como a maioria das ações não pode ser duplicada ou perdida o mecanismo deve usar a replicação passiva quente que atualiza o estado das demais réplicas a cada operação realizada.

Cada replicador deverá ser capaz de manter a execução do mecanismo mesmo que um replicador falhe durante a operação, ou seja, os replicadores devem ser tolerantes a falhas. Para isso, os replicadores devem realizar a detecção de falhas, gerenciamento de grupo e replicação do estado de forma distribuída, mantendo todas as informações necessárias para continuar replicando o sistema.

Quando um novo replicador for adicionado, sua réplica precisará receber o estado completo. Sendo assim, o servidor de execução deve ser capaz de informar seu estado completo, para que um replicador do grupo seja capaz de obter e enviar o estado para a nova réplica. Analogamente, para que essa réplica apresente o mesmo estado global dos demais, o servidor de execução deve ser capaz de aplicar um novo estado.

Como o estado será replicado usando replicação passiva quente, para que o servidor de execução possa fazer a atualização incremental do seu estado, o replicador deve ser capaz de encaminhar atualizações parciais do estado. Essas atualizações serão enviadas em mensagens de atualização que permitem que menos tempo seja gasto para atualizações pequenas no estado global, enviando apenas o que foi alterado desde o último estado enviado. Para isso, o servidor de execução tem que ser capaz de notificar o replicador de mudanças no estado e, para aplicar as mensagens de atualização, o servidor de execução tem que ser capaz de aplicar a atualização parcial do seu estado.

Para se inscrever no grupo, um novo replicador precisa conhecer apenas a localização de um dos replicadores do grupo de que deseja fazer parte. Sendo assim, qualquer replicador poderá adicionar um novo replicador ao grupo.

Para que o mecanismo se adapte a diferentes demandas de tempo de resposta, o intervalo para identificação de falha deve ser parametrizado. Adicionalmente, todos os replicadores do mesmo grupo devem apresentar o mesmo intervalo de tempo para identificação de falha.

Como a prioridade do mecanismo é a execução do processo, servidores de execução devem ser capazes de inibir a troca de servidor principal caso identifique que o mesmo ainda está em execução. Sendo assim, para que identificação de falha possa ser inibida, os servidores de execução devem ser capazes de confirmar a falha de um outro servidor de execução. A lógica para

determinar se um servidor de execução continua em operação fica a cargo do sistema de gerenciamento de *workflow* e deve ser padronizada para que todos os servidores de execução cheguem ao mesmo resultado ao realizá-la.

As decisões estabelecidas para o projeto podem ser observadas na tabela 4.2.

O componente replicador será independente do servidor de execução.
O mecanismo usará a replicação passiva quente.
O grupo de replicadores será tolerante a falhas.
O servidor de execução deve ser capaz de informar seu estado completo.
O servidor de execução deve ser capaz de aplicar um novo estado.
O replicador será capaz de encaminhar atualizações parciais do estado.
O servidor de execução deverão notificar mudanças no estado.
O servidor de execução deverão aplicar atualização parcial do estado.
Qualquer replicador poderá adicionar um novo replicador ao grupo.
O intervalo para identificação de falha será parametrizado.
Os servidores de execução deverão confirmar a falha de um outro servidor.

Tabela 4.2: Lista de Decisões de Projeto para o Mecanismo.

4.3 O Mecanismo de Tolerância a Falhas

Nesse trabalho estabelecemos interfaces para um mecanismo de tolerância a falhas para sistemas de gerenciamento de *workflow*. Implementamos o mecanismo usando CORBA e o desenvolvemos com o foco na tolerância de falhas de parada, ou seja, falhas como perda de mensagens, particionamento e falhas bizantinas [11] não terão suporte.

O mecanismo se baseia em um componente replicador externo ao sistema de gerenciamento de *workflow* que interage com o motor do servidor de execução do sistema que, por sua vez, é a réplica propriamente dita. Para que o mecanismo seja tolerante a falhas, cada replicador fica responsável por gerenciar a lista de replicadores do grupo, encaminhar mensagens de atualização (no caso do replicador principal), detectar falhas nos demais replicadores e eleger uma nova réplica principal em casos de falhas.

A seguir serão descritos em detalhes o replicador e as modificações que devem ser feitas no motor que será replicado para que esse possa ser usado pelo mecanismo.

4.3.1 O Replicador

O replicador possui quatro funções principais: gerenciamento de grupos, detecção de falhas, localização de serviço e replicação do estado. Essas funções se

relacionam durante a operação do sistema e não é possível descrever umas das funções sem mencionar características ou o funcionamento das demais. Sendo assim, a descrição das funções pode fazer referência a características de funções ainda não descritas. Além disso, as funções são realizadas de forma distribuída entre as réplicas, em alguns casos existindo o papel de um coordenador. Na figura 4.2 é possível observar a interface de um replicador usada na interação com o motor e entre replicadores.

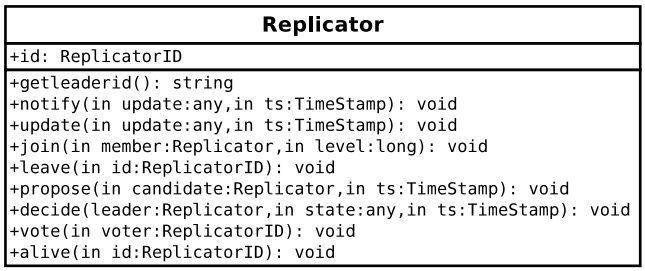

Figura 4.2: Interface dos replicadores em detalhe.

O gerenciamento de grupos do replicador é responsável por manter a lista de replicadores atualizada. Cada replicador mantém uma lista com os demais membros conhecidos do grupo. Essa lista é utilizada para detecção de falhas, descrita à frente. Um novo replicador, para se inscrever no grupo, envia um pedido para um replicador ativo. Quando um replicador recebe o pedido de inscrição de um novo membro, ele encaminha esse pedido para os membros conhecidos, se inscreve no novo membro e adiciona o novo membro a sua lista de replicadores conhecidos. O replicador principal, quando recebe esse pedido, pede um novo estado para o motor do sistema e informa que a execução deve ser pausada. Assim que o estado é obtido, o replicador principal envia esse estado e adiciona o novo replicador a sua lista. Após a inclusão do novo membro na sua lista o replicador envia um comando para o motor retomar a sua execução. O gerenciamento de grupos também é responsável pela remoção de um membro. Nesse caso, essa remoção deve ser feita explicitamente, a partir do identificador do replicador. Caso o membro sendo removido seja o atual líder do grupo, uma nova eleição será iniciada. Os passos de uma eleição são descritos posteriormente na função de replicação de estado.

A função de detecção de falhas do replicador também é realizada por todos os membros do grupo. O algoritmo usado para determinar qual replicador está em falha se baseia no algoritmo de detector de falhas não-confiáveis para sistemas distribuídos confiáveis [14]. Os replicadores são configurados com um intervalo para *timeout* a partir do qual a ausência de comunicação de um dos replicadores irá indicá-lo como suspeito de falha. Para executar esse algoritmo,

cada replicador possui duas co-rotinas. Uma co-rotina envia, para todos os membros conhecidos, uma mensagem de *keepalive*. Cada replicador, ao receber uma mensagem de *keepalive*, armazena o tempo em que recebeu a mensagem e o remetente. A segunda co-rotina da detecção de falhas verifica o último momento em que cada replicador enviou uma mensagem de *keepalive*. Caso esse momento tenha acontecido há um tempo maior que o intervalo para identificação de falhas, o replicador correspondente é suspeito de falha e removido do grupo.

A função localização de serviço é feita por qualquer replicador e permite que um cliente de execução obtenha as informações da execução dos processos mesmo que se conecte a um motor associado a um replicador secundário. Essa função é usada para evitar que a interface de interação com um cliente de execução para monitoração dos processos tenha que ser alterada para lidar com as funções de tolerância a falhas ou que motores réplicas tenham que ser modificados para gerar informações de monitoração a partir das atualizações de estado. Isso porque, ao receber uma conexão de um cliente de execução, os motores réplica utilizam a localização de serviço para encaminhar essa conexão para o motor principal que estará executando e gerando as informações de monitoração sem alterações. Dessa forma, a interface entre o motor e o cliente de execução se mantém inalterada. Além disso, até que haja uma falha na réplica principal, o cliente de execução não precisa saber que existem réplicas do motor ou qual é a principal, de forma que seus comandos e a obtenção de informações sejam feitos diretamente no motor que está executando as operações. Apesar na conexão ser transparente, caso haja uma falha na réplica principal o cliente de execução precisa se reconectar e conhecer a localização de uma das outras réplicas.

Finalmente, a função de replicação do estado é responsável pelas tarefas de eleição de nova réplica principal em caso de falhas, gerenciamento de mensagens de atualização de estado, obtenção e definição de estado. Um replicador iniciado marcado como líder é considerado o replicador principal e o motor associado executa as ações. Para isso, o replicador se registra no motor para obter as atualização de estado a cada ação executada. A medida que novos replicadores vão sendo inscritos no grupo, o replicador principal passa a encaminhar as mensagens de atualização para eles. As mensagens de atualização são acompanhadas de um identificador de estado, chamado de *timestamp*. Esse identificador é usado para determinar, na eleição, qual o replicador que possui o estado mais recente e deverá ser a nova réplica principal.

Os replicadores lidam com falhas de acordo com o papel desempenhado pelo replicador em falha. Caso um replicador secundário seja suspeito de falha, ele é simplesmente removido da lista de membros. Quando o replicador

principal é suspeito de falha, é necessário iniciar eleição entre os replicadores secundários para determinar qual deve substitui-lo.

O passo inicial da eleição se dá com os replicadores se candidatando por meio de uma proposta. Essa proposta contém o seu identificador e o *timestamp* da última atualização recebida por esse replicador. Caso um motor réplica identifique que o motor principal ainda está em execução, o replicador correspondente não enviará a sua proposta, ou seja, a eleição não poderá ser concluída até que todos os motores confirmem a falha do motor principal. Eventualmente, todos os replicadores recebem as propostas dos demais e, ao identificarem que possuem as propostas de todos os demais membros, determinam a melhor.

A determinação da melhor proposta leva em consideração o *timestamp* e o identificador do candidato, sendo eleito o candidato com o maior *timestamp* e, em caso de empate, com o identificador mais baixo. O critério do maior *timestamp* é usado para minimizar a quantidade de atualizações perdidas. Dessa forma, mesmo que uma atualização tenha alcançado apenas uma das réplicas é possível continuar a execução dos processos a partir do estado que contém essa atualização. Um replicador envia seu voto para replicador que apresentou a melhor proposta, informando o indicador do replicador que votou. Essa informação é necessária caso, durante a eleição, o candidato identifique a falha de um de seus eleitores e seja necessário desconsiderar o voto correspondente. Durante a eleição, a cada voto recebido o replicador verifica se possui todos os votos disponíveis e, sendo o caso, envia uma mensagem de decisão para os demais, se tornando o replicador principal. Nesse momento, o replicador principal recém-eleito inicia a execução do motor associado a partir do estado que possui.

4.3.2 Modificações no Motor

Associado a um replicador existe um motor replicado que faz o papel de réplica no mecanismo. Esse motor precisa disponibilizar maneiras de um replicador se registrar e gerenciar o estado dos seus *workflows*. Como esperado, isso implica em mudanças na aplicação para que funcione com tolerância a falhas. A interface esperada para o motor replicado pode ser observada na figura 4.3

Em relação a replicação do estado, o motor deve ser capaz de: informar seu estado, substituir seu estado, informar uma atualização parcial no estado, aplicar uma atualização parcial no seu estado e atribuir novos identificadores a estados diferentes. O estado informado deverá conter todos os dados relevantes

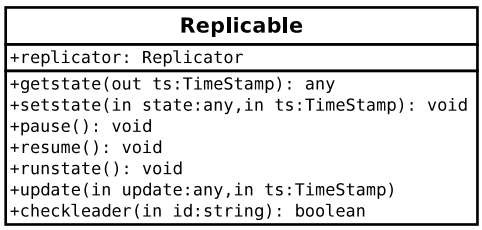

Figura 4.3: Interface com métodos necessários para utilização de um motor como réplica pelo mecanismo.

para a execução dos *workflows*, além das *threads* que estão executando ações em paralelo em um *workflow*. Informações de atualização parcial são utilizadas para reduzir o volume das mensagens trocadas. Elas permitem que no lugar de enviar o estado completo a cada alteração no estado, apenas a parte alterada seja enviada. Para determinar o estado mais novo, o motor associa identificadores aos estados obtidos e às atualizações de estado enviadas. Esse identificador deve ser enviado junto com o estado informado e será usado para determinar, no caso da falha de uma réplica principal, qual réplica secundária possui o estado mais recente. Sendo assim, esse identificador de estado deve ser, preferencialmente, numérico, para que a definição de nova réplica principal seja simplificada.

O envio da mensagem de atualização deve ser feito sempre que alguma ação for concluída. Como esse momento é conhecido apenas pelo motor, ele deve possuir um meio de comunicação para enviar notificações de atualização para o replicador associado. Além do envio de notificações, quando um cliente de execução se conecta a um motor réplica, esse meio de comunicação é utilizado para a localização de serviço. Quando necessário o motor pode buscar uma referência para o motor principal que será repassada para o cliente de execução. Finalmente, o motor deverá permitir que a execução dos processos fique em espera, dando suporte a operação de retomada da execução quando o replicador associado for eleito replicador principal.

A confirmação de falhas no motor principal pode não ser possível em todos os casos, mas o motor precisa disponibilizar um método, mesmo que esse confirme qualquer suspeita de falha. Casos que contam com elementos externos comuns a todos os motores réplicas podem usar esses elementos para confirmar a suspeita de falha do motor principal. Por exemplo, se o sistema possuir uma base de dados única, é possível reservar campos da base de dados para esse fim. Nesse exemplo, o motor principal poderia zerar um campo em intervalos de tempo pré-definidos e um procedimento na base de dados poderia incrementar esse valor a cada segundo. Sendo assim, bastaria que motores réplica verifiquem

se esse valor está acima de um limite para confirmar a suspeita de falha do motor principal. Apesar de ser semelhante a identificação de falhas, essa lógica não substituiria a identificação de falhas feita nos replicadores pois, além de depender de um elemento externo comum, a comunicação com esse elemento pode ser custosa e interferir significativamente no funcionamento da base de dados.

4.4 Exemplo de uso

Nessa seção será descrito um exemplo com os passos necessários para a inicialização do mecanismo com alguns replicadores e réplicas. O exemplo irá também considerar o caso da falha de uma réplica secundária e a sua recuperação. Por fim, o exemplo irá considerar o caso da falha da réplica principal.

O primeiro passo, é inicializar o replicador principal. Esse replicador deve ser inicializado com o parâmetro `leader`, além da localização do seu motor réplica e o intervalo para indicação de falhas. Por ser o replicador principal o motor réplica deverá esta em execução. Nesse ponto o replicador principal está aguardando a inscrição de novos replicadores.

O segundo replicador é instanciado indicando a sua réplica e pelo menos um replicador, no caso o replicador principal. Nesse caso, após ao registro no seu motor réplica, o novo replicador irá fazer um pedido de inscrição no replicador principal. Ao receber o pedido, o replicador principal irá se inscrever no novo membro, obter o estado da sua réplica, incluir o novo replicador na sua lista de membros e enviar uma mensagem informando: o replicador principal, o estado do motor principal e o identificador do estado. O novo replicador irá incluir o replicador principal na sua lista de membros e, ao receber a mensagem com o estado, irá aplicar o estado em seu motor réplica. Nesse momento o novo replicador e o replicador principal iniciam o envio de mensagens de *keepalive*. Além disso, o replicador principal estará enviando mensagens de atualização que serão encaminhadas para o motor réplica pelo replicador secundário. Essas mensagens de atualização possuem apenas as modificações realizadas desde o último estado enviado e o identificador do novo estado. Nesse ponto, os dois replicadores estão verificando, a cada intervalo de tempo, o momento em que a última mensagem de *keepalive* do outro foi recebida para determinar a suspeita de falha. A figura 4.4 representa a sequência de chamadas para a partida dos replicadores e início da replicação do estado.

Um terceiro replicador deve ser instanciado indicando a localização do motor réplica e de um dos dois replicadores instanciados. Supondo que esse

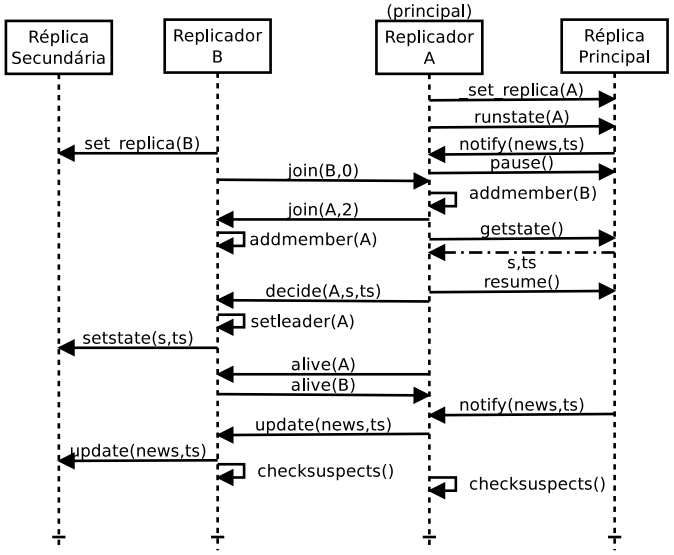

Figura 4.4: Sequência de chamadas de para partida de dois replicadores e início da replicação.

replicador tem apenas a localização do replicador secundário, seu pedido de inclusão no grupo é encaminhando para o replicador principal. Os dois replicadores que já estavam em operação enviam uma mensagem de inclusão de volta, sendo que o replicador principal só adiciona o novo replicador quando obtém o estado atual do seu motor. Durante a obtenção do estado, o motor principal é mantido em espera. Após enviar o estado do motor principal para o novo replicador, o replicador principal retoma a execução do motor principal. Nesse ponto, o replicador principal passa a encaminhar as mensagens de atualização para os dois replicadores conhecidos. Todos os replicadores passam a enviar as mensagens de *keepalive* para os demais e todos verificam o tempo em que a última mensagem de *keepalive* foi recebida de cada replicador conhecido para determinar a suspeita de falha. A figura 4.5 representa a sequência de chamadas para partida de um terceiro replicador.

Supondo a falha em um dos replicadores secundários, eventualmente o replicador principal e o outro replicador secundário irão suspeitar da falha devido ao intervalo desde que a última mensagem de *keepalive* foi recebida. Como era um replicador secundário, os dois replicadores irão identificar que o membro suspeito não era importante, realizando apenas a remoção deste da sua lista de membros. Ao se recuperar, esse replicador deve fazer um novo pedido de inscrição no grupo para voltar a fazer parte da replicação.

Supondo então, que existem novamente três replicadores e o replicador principal falha, os dois replicadores secundários irão, eventualmente, suspeitar dessa falha. Nesse caso, durante a suspeita os dois replicadores irão identificar

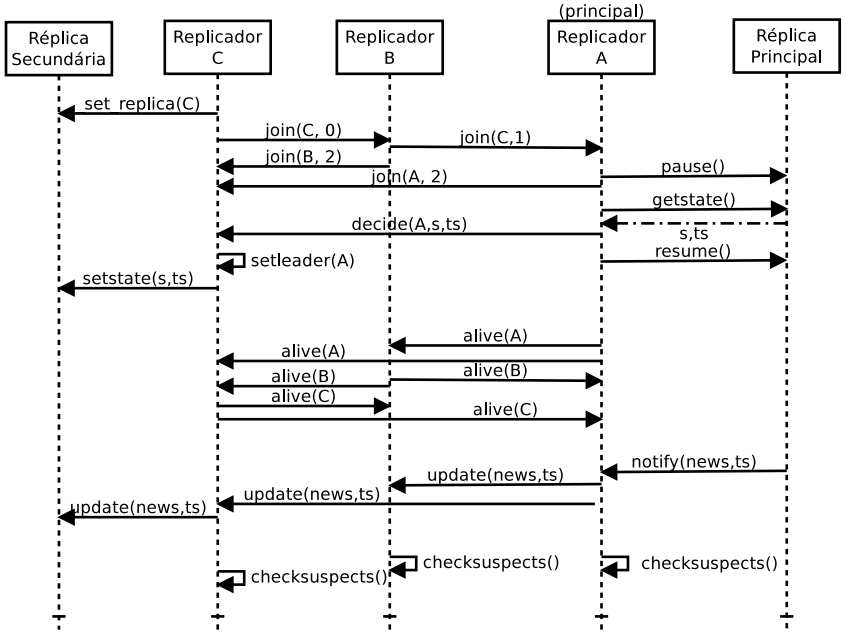

Figura 4.5: Sequência de chamadas para entrada de um replicador secundário adicional.

que o membro suspeito era o replicador principal e tentar confirmar a falha com o seu motor réplica associado. Uma vez que a falha for confirmada, os dois replicadores enviam uma proposta para o outro se candidatando a novo replicador principal. Essa proposta contém o seu identificador e o identificador do último estado aplicado pela sua réplica. Ao receberem a proposta, eles verificam se receberam todas as propostas possíveis e, se for o caso, comparam os identificadores de estado conhecidos, incluindo seu próprio identificador. Uma vez definido o melhor candidato, o replicador envia seu voto podendo, no caso do replicador com o menor identificador, votar em si próprio. Nesse exemplo, o replicador que receber dois votos (o próprio e o do outro replicador) irá ser o novo replicador principal. Ao se tornar o novo replicador principal, o replicador envia a mensagem de decisão para o outro, incluindo uma referência para si próprio, o estado atual da sua réplica e o identificador desse estado. Em seguida, o replicador principal envia o comando para a sua réplica iniciar a execução dos processos a partir do último estado informado. A sequência de chamadas com suspeita de falha e eleição de nova réplica principal podem ser observados na figura 4.6.

4.5 Limitações do Mecanismo

Devemos deixar claro que o mecanismo proposto não tem como objetivo ter o menor impacto no desempenho possível e sim avaliar uma forma de

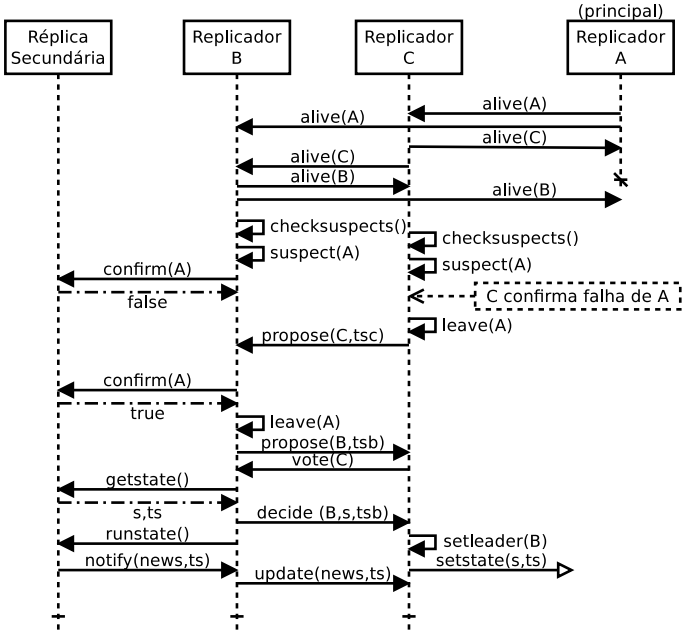

Figura 4.6: Sequência de chamadas para eleição de nova réplica.

replicar um sistema de gerenciamento de *workflow*. Sendo assim, iremos identificar a seguir algumas limitações do mecanismo e discutir brevemente linhas de investigação de como superá-las.

O impacto da pausa na execução dos processos será avaliado no estudo de caso apresentado no próximo capítulo. Entretanto, caso seja necessário, esse impacto pode ser reduzido modificando o mecanismo para que os replicadores acumulem as mensagens de atualização que não podem ser aplicadas. Dessa forma, ao receber um estado completo, o novo replicador aplica, na sua réplica, as mensagens de atualização acumuladas desde a obtenção do estado da réplica principal.

Enviar mensagens de atualização a cada operação pode sobrecarregar a rede em alguns casos. Para evitar esse comportamento, o mecanismo poderia suportar a marcação de ações críticas na realização do processo que estariam associadas à atualização do estado das réplicas. Nesse caso, o replicador principal poderia acumular mensagens de atualização até que uma ação marcada fosse alcançada, substituindo o envio contínuo de mensagens curtas pelo envio de um conjunto mensagens em pontos específicos do processo.

O acúmulo de mensagens poderia ser usado também para reduzir o tempo gasto na eleição. Nesse caso, a obtenção e envio do estado completo para os replicadores poderiam ser substituídos pelo envio de mensagens de atualização desde o estado indicado pelo *timestamp* da proposta dos replicadores.

O número de réplicas em geral deve ser pequeno, mas em casos que o número de réplicas for grande, a necessidade de enviar as mensagens de

atualização para todas as réplicas pode sobrecarregar a rede e atrasar de forma significativa a execução dos *workflows*. Para evitar esse atraso, os replicadores poderiam ser ordenados em um *heap* e, a partir do replicador principal, cada um encaminharia mensagens de atualização para outros dois replicadores. Nesse caso, um replicador com índice i enviaria mensagens de atualização apenas para os replicadores no índice $2i$ e $2i+1$. O replicador principal teria índice 1 e enviaria mensagens de atualização para apenas dois replicadores secundários, liberando o motor para continuar a replicação assim que as mensagens de atualização fossem recebidas por duas réplicas. A ordenação dessa estrutura poderia se basear no identificador dos replicadores ou usar alguma informação da rede em que eles se encontram. Imaginando que a detecção de falhas passe a usar um *timeout* específico para cada replicador, esse valor poderia ser usado na ordenação, ou seja, replicadores mais distantes poderiam apresentar *timeouts* maiores que replicadores da rede local. Além disso, o valor do *timeout* poderia ser adaptado durante a execução, sendo aumentado caso o replicador correspondente seja suspeito de falhas.