Um algoritmo de Membership para o modelo Síncrono Particionado (SPA) de Sistemas Distribuídos com particionamento forte.

Maria Clara Aderne dos Santos[1], **Sérgio Gorender**[1]

[1]Departamento de Ciência da Computação
Instituto de Matemática e Estatística
Programa de Pós-Graduação em Mecatrônica
Universidade Federal da Bahia (UFBA) – Salvador, BA – Brasil

claraaderne@gmail.com, gorender@ufba.br

Abstract. *Group communication services are important building blocks for fault-tolerant distributed systems. In such a service, membership protocols manages the composition of the communication group considering dynamic groups with incoming and outgoing processes. Algorithms have been presented to solve membership in synchronous and partially synchronous distributed systems, tolerating failures, and the impossibility of membership in asynchronous systems has been proven. Hybrid distributed system models, with synchronous and asynchronous components, have been studied, being relevant for representing current systems, such as systems based on IoT, integrating, for example, industrial plants through the internet. In this paper we present an algorithm that solves the membership problem in the distributed systems model SPA (Synchronous Partitioned). The proposed algorithm allows the existence of several intersecting groups. In addition to the algorithm, we present formal proofs of its properties and implementation results in a simulated environment.*

Resumo. *Serviços de comunicação em grupo são importantes blocos de construção para sistemas distribuídos tolerantes a falhas. Neste serviço, o membership faz a gestão da composição do grupo de comunicação considerando grupos dinâmicos com processos entrando e saindo. Têm sido apresentados algoritmos para solucionar o membership em sistemas distribuídos síncronos e parcialmente síncronos, tolerando falhas, tendo sido provada a impossibilidade do membership em sistemas assíncronos. Modelos de sistemas distribuídos híbridos, com componentes síncronos e assíncronos, têm sido estudados, sendo relevantes por representar sistemas atuais, tais como sistemas baseados em IoT, integrando, por exemplo, plantas industriais através da internet. Neste artigo apresentamos um algoritmo que resolve o problema de membership no modelo de sistemas distribuídos SPA (Síncrono Particionado). O algoritmo proposto permite a existência de diversos grupos com interseção entre eles. Além do algoritmo, apresentamos provas formais de suas propriedades e resultados de implementação em ambiente simulado.*

1. Introdução

O Modelo Síncrono Particionado de Sistemas Distribuídos (SPA) foi inicialmente apresentado em [Macêdo and Gorender 2008] e [Macêdo and Gorender 2009], como

um modelo híbrido para sistemas distribuídos tolerantes a falhas, com componentes síncronos e assíncronos. Neste modelo tanto processos como canais de comunicação podem ser síncronos ou assíncronos, permanecendo assim durante a execução do sistema, enquanto não falham. O modelo SPA pode ser caracterizado como um modelo parcialmente síncrono de sistemas distribuídos, modelos que não são síncronos, mas que evitam o resultado da impossibilidade do consenso tolerando falhas em sistemas assíncronos [Fischer et al. 1985] ao considerar algumas características síncronas no sistema [Chandra and Toueg 1996], [Cristian and Fetzer 1999] e [Veríssimo and Casimiro 2002]. A característica fundamental deste modelo é o fato de que processos e canais de comunicação síncronos são agrupados em partições síncronas. Esta característica permitiu que fossem desenvolvidos algoritmos para um detector de defeitos e para um serviço de consenso. Modelos de sistemas distribuídos híbridos, como o SPA, com componentes síncronos e assíncronos, têm sido estudados, sendo relevantes por representar sistemas atuais, como por exemplo sistemas baseados em IoT, integrando plantas industriais através da internet.

Um dos aspectos importantes deste modelo, conforme discutido nestes artigos, é o fato de poder ter um detector de defeitos da classe P (Perfeito) em um sistema que não é síncrono [Macêdo and Gorender 2009]. Este detector é obtido caso o modelo satisfaça a propriedade *strong partitioned synchrony*, ou seja, todos os processos do sistema façam parte de uma partição síncrona. Ainda assim, não teremos um sistema síncrono, pois existirão canais de comunicação assíncronos entre processos deste sistema. Além disto, foi demonstrado que no modelo SPA é possível ter uma detecção de defeitos que, em qualquer situação, satisfaz a propriedade *strong accuracy* [Macêdo and Gorender 2012], ou seja, um detector de defeitos que não faz detecções incorretas.

Foi apresentado para o modelo SPA um algoritmo de consenso eficiente [Gorender and Macêdo 2011], utilizando o detector de defeitos apresentado anteriormente, e tolerando falhas. O detector de defeitos e o consenso são importantes mecanismos para prover tolerância a falhas em sistemas distribuídos. Um mecanismo também importante para a construção de sistemas distribuídos tolerantes a falhas é um serviço de comunicação em grupo.

Um serviço de comunicação em grupo faz a gestão da formação de grupos de comunicação entre os processos do sistema e provê a comunicação entre os membros dos grupos com garantias diversas, tolerando falhas. Este serviço é formado por dois componentes complementares: um serviço de *membership*, responsável pela criação dos grupos de comunicação e pela gestão da composição destes grupos, e um serviço de *multicast*, o qual fornece primitivas de comunicação em grupo *multicast*, podendo fornecer uma comunicação confiável, e com diferentes critérios de ordenamento [Cristian 1991, Chockler et al. 2001, Anceaume et al. 1995].

O problema do *membership* é caracterizado pela necessidade de manutenção da composição de grupos de comunicação, assumindo que processos podem solicitar entrar e sair destes grupos, ou falhar, e neste caso precisam ser excluídos do *membership*. Para que a comunicação ocorra de forma correta e segura, é preciso que os processos que são membros de um grupo mantenham uma visão consistente de sua composição. Esta gestão da composição dos grupos é efetuada através da manutenção de visões sequenciais (*views*) desta composição instaladas de forma consistente por todos os processos membros. O

membership é considerado um problema de acordo, com características equivalentes ao problema do consenso.

O problema do *membership* em sistemas de comunicação em grupo tem sido investigado em diversos trabalhos [Cristian 1991], [Chockler et al. 2001], [Schiper 2006], [Macêdo and Freitas 2009]. Foram apresentadas soluções para o *membership* em sistemas síncronos em [Cristian 1991] e [Park et al. 2016], entre outros. Em sistemas assíncronos foi provada a impossibilidade do *membership* na ocorrência de falhas [Chandra et al. 1996]. Este resultado é similar ao da impossibilidade do consenso em sistemas assíncronos na presença de falhas [Fischer et al. 1985]. Esta impossibilidade ocorre por ser impossível em um sistema assíncrono diferenciar um processo lento de um que falhou.

Para evitar o resultado da impossibilidade do *membership* em sistemas assíncronos, têm sido apresentadas soluções para o *membership* em diferentes modelos de sistemas, como, por exemplo, para o modelo *Timed Asynchronous* [Fetzer and Cristian 1997]. Outras soluções são apresentadas em [Schiper and Toueg 2006], [Schiper 2006], [Ezhilchelvan et al. 1995], [Keidar et al. 2002], [lin and Hadzilacos 1999], entre outros trabalhos. Foram apresentados serviços de comunicação em grupo para sistemas distribuídos híbridos [Macêdo and Freitas 2009][Macêdo et al. 2011], tais como o Sistema Híbrido e Dinâmico [Gorender et al. 2007].

Neste artigo, apresentamos um protocolo de *membership* para o sistema Síncrono Particionado com a propriedade *Strong Partitioned Synchrony*. Por assumirmos a propriedade *Strong*, o algoritmo utiliza o detector de defeito da classe P [Macêdo and Gorender 2009], além do algoritmo de consenso [Gorender and Macêdo 2011], ambos desenvolvidos para o modelo SPA. Este serviço de *membership* permite a interseção entre grupos de comunicação, com processos podendo ser membros de mais de um grupo ao mesmo tempo. A partir das propriedades assumidas pelo modelo de sistema, e pelo detector de defeitos e consenso, o algoritmo de *membership* satisfaz propriedades de *safety* e *liveness*, garantindo que a visão da composição dos grupos é sempre mantida consistente para todos os seus membros e que as operações relacionadas à gestão dos grupos de comunicação, com processos entrando e saindo destes grupos, são sempre realizadas.

Além do algoritmo e sua descrição e das propriedades e suas provas formais, também apresentamos alguns resultados de experimentos obtidos através da implementação do algoritmo de *membership* em ambiente simulado de sistemas distribuídos [Silva Freitas and de Araújo Macêdo 2014].

O artigo é composto pelas seguintes seções: na seção 2 são apresentados os trabalhos relacionados; o modelo do sistema é apresentado na seção 3; a seção 4 descreve o modelo SPA, juntamente com os serviços de detecção de falhas e consenso; a seção 5 apresenta o algoritmo de *membership* proposto e as provas formais de suas propriedades são apresentadas na seção 5.1; a seção 6 descreve a implementação do protocolo e os resultados de testes realizados; e a seção 7 apresenta as conclusões do trabalho.

2. Trabalhos relacionados

Em sistemas distribuídos síncronos, existem soluções para o problema do *membership* como a apresentada em [Cristian 1991]. [Park et al. 2016] apresenta um protocolo de *membership* para sistemas distribuídos síncronos que executam sobre uma rede cabeada arbitrária, entre outros.

Motivados pelo resultado da impossibilidade do *membership* em sistemas assíncronos na presença de falhas [Chandra et al. 1996], têm sido apresentadas soluções para o *membership* desenvolvidas para sistemas distribuídos parcialmente síncronos, tais como [Ezhilchelvan et al. 1995], [Schiper and Toueg 2006], [Fetzer and Cristian 1997], [Keidar et al. 2002], [lin and Hadzilacos 1999], [Lim and Conan 2014] e [Schiper 2006], entre outros. [Chockler et al. 2001] apresenta uma *survey* de diversos trabalhos sobre comunicação em grupo discutindo enfoques diferentes. [Anceaume et al. 1995] faz uma análise de formalizações propostas para soluções para o *membership*, suas características e problemas.

Macêdo e Freitas apresentam em [Macêdo and Freitas 2009] e [Macêdo et al. 2011] o *Timed Causal Blocks* (TimedCB), um serviço de comunicação em grupo para sistemas distribuídos híbridos, auto-gerenciáveis, baseado no conceito dos *causal blocks*, o qual foi apresentado em [Ezhilchelvan et al. 1995]. Este trabalho assume um modelo híbrido e dinâmico de sistemas distribuídos, formado por canais de comunicação e processos que podem ser síncronos ou assíncronos. Também assume a existência de um oráculo de QoS, além de serviços de detecção de defeitos e consenso, como descritos em [Gorender et al. 2005]. Embora também seja um modelo de sistemas distribuídos híbrido, o modelo apresentado em [Gorender et al. 2005] é diferente do Síncrono Particionado, com características, comportamentos e propriedades diferentes.

O algoritmo de *membership* que apresentamos neste artigo foi desenvolvido para o modelo SPA, utilizando um serviço de detecção de defeitos da classe P, além de um serviço de consenso, conforme desenvolvidos para este modelo. Embora em um sistema que não é síncrono, o algoritmo apresentado permite efetuar de forma consistente a gestão de grupos de comunicação.

3. Modelo de Sistema

Assumimos um sistema distribuído composto por um conjunto de processos (Π), os quais se comunicam através de troca de mensagens via canais de comunicação (C). Existem canais de comunicação interligando cada 2 processos. Os processos e os canais de comunicação podem ser síncronos ou assíncronos, e este estado não é alterado durante a execução do sistema. Os processos e canais de comunicação formam um grafo completo não direcionado $DS = \{\Pi, C\}$. O modelo assume a existência de um oráculo temporal que realiza o mapeamento de canais e processos para valores síncronos e assíncronos. O oráculo é definido pela função de *QoS* (*quality-of-service*) que, por sua vez, é definida para os canais e processos no seu momento de criação sem alteração ao longo do tempo. O sistema satisfaz as definições e propriedades do modelo SPA de sistemas distribuídos, conforme descrito na próxima seção.

Assumimos a existência de uma primitiva de comunicação em grupo *reliable multicast* (*rmulticast*()), através da qual mensagens são enviadas para todos os processos

membros de um grupo de forma confiável, ou seja, é garantida a entrega das mensagens. Também assumimos que esta primitiva satisfaz a propriedade Sincronia Virtual (*Virtual Synchrony*) conforme descrita em [Chockler et al. 2001], ou seja, "se processos p e q instalam uma mesma *view* v na mesma *view* anterior v', então qualquer mensagem recebida por p em v' é também recebida por q em v'". Desta forma, é garantido que processos membros de um grupo recebem as mesmas mensagens enviadas por *rmulticast*, nas mesmas *views*.

4. Modelo SPA

Apresentado em [Macêdo and Gorender 2008] e [Macêdo and Gorender 2009], o Modelo Síncrono Particionado de Sistemas Distribuídos (SPA) é um modelo híbrido, possuindo componentes (processos e canais de comunicação) síncronos e assíncronos. Os processos e canais de comunicação formam partições síncronas, as quais são definidas como sub-grafos completos contidos no grafo do sistema (DS) no qual todos os componentes, processos e canais de comunicação são síncronos. Além disto, assume-se que para cada partição síncrona, não existe outro sub-grafo completo do grafo do sistema que contenha a partição síncrona, fora o próprio grafo DS.

O modelo satisfaz uma propriedade de particionamento, a qual pode assumir os valores *strong partitioned synchrony* ou *weak partitioned synchrony*. A propriedade é assumida como sendo *strong* quando todos os processos do sistema (pertencentes ao conjunto Π) fazem parte de alguma partição síncrona. A propriedade é assumida como sendo *weak* quando existem processos que são assíncronos, e que não fazem parte de nenhuma partição síncrona.

Todas as partições síncronas possuem ao menos um processo correto, que não falha durante a execução do sistema. Os demais processos podem falhar por parada sem produzir ações futuras. Também é assumido que os canais de comunicação são confiáveis, ou seja, não perdem ou alteram mensagens. A comunicação entre processos de diferentes partições síncronas, ou entre estes e processos assíncronos, se dá através de canais de comunicação assíncronos.

Neste trabalho assumimos que o modelo satisfaz a propriedade *strong partitioned synchrony*, ou seja, todos os processos são síncronos e pertencem a partições síncronas.

Foi também apresentado em [Macêdo and Gorender 2008] e [Macêdo and Gorender 2009] um detector de defeitos para o modelo SPA, o qual satisfaz as propriedades *strong accuracy* e *strong completeness*, pertencendo à classe de detectores P (*Perfect*), caso o modelo satisfaça a propriedade *strong partitioned synchrony*. Em [Gorender and Macêdo 2011] foi apresentado um algoritmo de consenso para o modelo SPA, que utiliza o detector de defeitos. Este algoritmo de consenso satisfaz as propriedades de terminação, validade e acordo uniforme. Como assumimos que o modelo satisfaz a propriedade *strong partitioned synchrony*, e desta forma temos um detector de defeitos da classe P, o consenso tolera falhas de até n - p processos, sendo n o número de processos no sistema, e p o número de partições síncronas. Portanto, podem falhar todos os processos menos aqueles que por definição do sistema devem permanecer corretos, um por partição síncrona.

5. Membership no SPA

O protocolo de *membership* que apresentamos a seguir foi desenvolvido para o modelo SPA com a propriedade *Strong Partitioned Synchrony* (Particionamento Forte). O algoritmo utiliza os serviços de detecção de defeitos e de consenso desenvolvidos para o modelo SPA, conforme descritos na seção anterior. Este serviço de *membership* permite a criação e exclusão de novos grupos de comunicação, e a inclusão e exclusão de processos nos grupos existentes. A cada modificação em um grupo é gerada uma nova visão (*view*) deste grupo, compartilhada por todos os processos membros do grupo. O serviço permite a interseção entre os grupos, com processos podendo fazer parte de mais de um grupo ao mesmo tempo.

Quando um novo grupo é criado, é atribuída uma identificação representada pela variável g. Identificações de grupo são assumidas serem únicas. Cada grupo tem sua composição representada por *views*, as quais recebem identificações únicas em cada grupo. Estas identificações também indicam a sequência entre as *views* instaladas em cada grupo. Cada *view* mantém a lista de processos que compõem o grupo naquele instante. Novas *views* são instaladas para um grupo quando alguma alteração é realizada em sua composição.

No algoritmo temos as seguintes variáveis:

- g - identificação única de um grupo;
- v_{kg}^{g} - composição da *view* atual do grupo g;
- kg - identificação da *view* atual do grupo g;
- p_i - processo i, identificado como o processo que está executando o algoritmo localmente;
- p_j - processo j, um outro processo qualquer.

O protocolo está apresentado no Algoritmo 1 e é dividido em oito tarefas numeradas de 0 a 7. As tarefas 1 a 7 são concorrentes, sendo que a tarefa 7 pode ter diversas execuções concorrentes, uma para cada grupo do qual o processo faz parte. O serviço é executado em módulos distribuídos, um para cada processo do sistema, sendo que todos os processos executam o mesmo algoritmo. A *Task* 0 é responsável por preparar o funcionamento do serviço, chamando a *Task* 2 para acessar o detector de defeitos e criando a lista de grupos *group-list* que irá conter uma referência para os grupos nos quais o processo é um membro.

Algorithm 1 Módulo do serviço de *membership* para o processo p_i.

Task 0 - init
1: detectFaulty
2: new $group - list$
Task 1 - createGroup()
3: new group g such that g is a unique value
4: insert g in $group - list$
5: new view v_{kg}^{g} such that $kg = 1$
6: $v_{kg}^{g} \leftarrow p_i$
7: new $tasklist - g$
8: run instalView($tasklist - g$) concurrently
9: **return** g **and** (kg, v_{kg}^{g})

Task 2 - detectFaulty
10: **while true do**
11: **for each** group g **do**
12: **for each** $p_j \in v_{kg}^g$ such that $p_j \in fault$ **do**
13: insert $\{p_j, g, \text{“}FD\text{”}\}$ in $tasklist - g$
14: **end for**
15: **end for**
16: **end while**
Task 3 - joinGroup(g)
17: rmulticast(newView(p_i, g, “$join$”))
Task 4 - leaveGroup(g)
18: rmulticast(newView(p_i, g, “$leave$”))
19: insert $\{p_i, g, \text{“}leave\text{”}\}$ in $tasklist - g$
Task 5 - installView($tasklist - g$)
20: $pi\_in\_g =$ **true**
21: **while** pi_in_g **do**
22: wait for $tasklist - g <> vazio$
23: select the first task $\{p_j, g, task\}$ in the $tasklist - g$
24: **if** ($task =$ “$join$”) **then**
25: $\{CP, p_j, task\} \leftarrow$ consensus(idconsensus(g, $kg + 1$), $v_{kg}^g \cup \{p_j\}$, p_j, $task$)
26: **else**
27: $\{CP, p_j, task\} \leftarrow$ consensus(idconsensus(g, $kg + 1$), $v_{kg}^g - \{p_j\}$, p_j, $task$)
28: **end if**
29: **if** ($task$ = “$leave$”) **and** ($p_j = p_i$) **then**
30: delete v_{kg}^g
31: delete $tasklist_g$
32: $pi_in_g =$ **false**
33: **else**
34: $kg = kg + 1$
35: $v_{kg}^g = CP$
36: delete it from $tasklist - g$
37: **if** ($task =$ “$join$”) **then**
38: send to p_j (joingroup(CP, g, k, $tasklist - g$))
39: **end if**
40: **end if**
41: **end while**
Task 6 - newView()
42: **when** rdeliver(newView(p_j, g, $task$)) **do**
43: **if** ($task$ = “$join$” **and** $p_j \notin v_{kg}^g$) or ($task$=”$leave$” **and** $p_j \in v_{kg}^g$) **then**
44: insert $\{p_j, g, task\}$ in $tasklist - g$
45: **end if**
46: **end when**
Task 7 - joingroupmessage()
47: **when** rdeliver(joingroup(CP, g, k, $tasklist - g$))
48: **if** ($\nexists v_{kg}^g$ such that $p_i \in v_{kg}^g$) **then**
49: new group g such that $kg = k$

50: new view $v_{kg}^{g} = CP$
51: new $tasklist - g$
52: run instalview($tasklist - g$) concurrently
53: **end if**
54: **end when**

A *Task* 1 é chamada por um processo quando este deseja criar um novo grupo. A tarefa cria um novo grupo com a identificação única g, e com o processo que executa a tarefa (p_i) como membro único em sua primeira view, v_{kg}^{g}. A tarefa também cria para o grupo uma lista *tasklist-g*, para as tarefas pendentes de alteração do grupo, e chama a *Task* 5 para executar estas alterações, de forma concorrente aos demais grupos dos quais o processo p_i faz parte. O novo grupo é inserido na lista *group-list*. A *Task* 2 executa o tempo todo de forma concorrente com as demais tarefas do algoritmo e faz uma consulta contínua ao serviço de detecção de defeitos do sistemas. Para cada processo que é detectado como tendo falhado, e que faz parte de algum dos grupos de comunicação no qual o processo p_i também faz parte, uma ação para que o processo detectado seja excluído do grupo é inserida na *tasklist-g* do grupo.

A *Task* 3 é chamada por um processo p_i quando este solicita a sua entrada em um grupo g. Esta solicitação é enviada para todos os processos do grupo g através da primitiva *rmulticast*, para que estes processos instalem uma *view* com o processo p_i como membro. A *Task* 4 é chamada por um processo p_i quando este solicita a sua saída de um grupo g do qual faz parte. A solicitação é enviada por *rmulticast* para todos os processos do grupo e a ação de saída é inserida na *tasklist-g* do grupo.

A *Task* 5 possui várias execuções concorrentes, uma para cada grupo no qual o processo p_i faz parte, sendo chamada pela *Task* 1 ou pela *Task* 7. Esta tarefa chama o serviço de consenso com a primeira ação da lista *tasklist-g* para o grupo g e verifica o resultado do consenso. Se a ação acordada foi a exclusão do próprio processo p_i, este exclui a *tasklist-g* e a $view$ do grupo e encerra a execução concorrente desta tarefa para este grupo. Em caso contrário, a nova *view* acordada no consenso é instalada e a ação de alteração executada é excluída de *tasklist-g*. Se a ação executada tiver sido um *join* é enviada uma mensagem com a composição do grupo para o processo p_j que solicitou a inclusão.

A *Task* 6 é executada quando o processo recebe uma mensagem *newView* solicitando a instalação de nova *view* por uma ação de entrada de um processo em um grupo (*join*) ou por uma ação de saída de um processo de um grupo (*leave*). É feita uma verificação se a solicitação é consistente e a ação é inserida na *tasklist-g* do grupo. Por fim, a *Task* 7 é executada por um processo que solicitou sua entrada em um grupo. Quando a inclusão é executada e os processos membros do grupo instalam uma nova *view* com o processo solicitante como membro do grupo, enviam para este uma mensagem *joingroup*. Nesta tarefa, o processo que solicitou a inclusão aguarda pelas mensagens *joingroup* e cria, a partir das mensagens, a *view* para o grupo, uma *tasklist-g*, e inicia nova execução concorrente da *Task* 5.

5.1. Provas formais de propriedades

O protocolo de *membership* apresentado satisfaz propriedades de *liveness* e *safety*, conforme definidas em trabalhos clássicos da área, tais como [Chockler et al. 2001]. As pro-

priedades de *liveness* possuem o objetivo de garantir que operações solicitadas são realizadas pelo serviço de *membership*, com a instalação de novas *views* que representem a execução destas operações. Com estas propriedades provamos que as operações iniciadas são corretamente executadas, efetuando a entrada e saída de processos no *membership* dos grupos. As propriedades de *safety* visam provar que em todas as situações as operações sempre geram resultados consistentes, e que não ocorrem estados indesejáveis. Apresentamos primeiro as propriedades de *safety* e depois as de *liveness*, além das provas formais mais importantes, quando o espaço permitir.

5.1.1. Propriedades de *safety*

Apresentamos nos teoremas a seguir as propriedades Integridade (*self-inclusion*), Acordo Uniforme e Validade.

Teorema 5.1.1 (Integridade (*self-inclusion*))**.** *Se um processo p instala a view $v(g)$ então $p \in v(g)$.*

Demonstração. A prova é desenvolvida por contradição, assumindo que o processo p instala a *view* $v(g)$, do grupo g, e que p não pertence a esta *view*, ou seja, $p \notin v(g)$. Existem duas possibilidades para p instalar a *view* $v(g)$: 1) p_i já é membro da grupo g, tendo instalado a *view* anterior, e executa *Task* 5 - *installView* para o grupo g, instalando a *view* $v(g)$ ou 2) p_i não faz parte do grupo, mas solicitou *join* executando a *Task* 3, e depois de o *join* ter sido efetuado pelos membros do grupo, p executa a *Task* 7, na qual instala a *view* $v(g)$.

1. p_i executa as linhas 34 e 35 da *Task* 5 - *installView* para o grupo g - para que esta tarefa esteja em execução pelo processo p_i para o grupo g, e as linhas 34 e 35 sejam executadas instalando a nova *view* do grupo, é preciso que p_i faça parte do grupo e de seu *membership*, e neste caso *Task* 5 já está em execução, e que a condição do comando if da linha 29 dê um resultado falso, e para tal, ou a alteração executada no *membership* não foi um *leave*, ou o processo excluído não foi o processo p_i em execução, portanto, neste caso, p_i já faz parte do grupo, e como não executou um *leave*, não foi retirado de seu *membership*.
2. p_i executa as linhas 49 e 50 do algoritmo, na *Task* 7, instalando o grupo g e a nova *view* deste grupo - p_i executa a *Task* 7 ao receber uma mensagem *joingroup* de algum outro processo, p_j, sendo que um processo p_j envia uma mensagem *joingroup* para um processo p_i ao executar a linha 38 na *Task* 5 do algoritmo; esta linha só é executada se a ação executada foi um *join*, sendo a mensagem enviada para o processo que foi inserido no *membership*, e a composição deste *membership* enviada na mensagem foi o resultado do consenso chamado na linha 25, e que passa a ter o novo processo, neste caso p_i, em sua composição, portanto, p_i faz parte da nova *view* instalada.

Como em ambos os casos nos quais um processo p_i instala uma *view* para um grupo, este processo faz parte do *membership* do grupo, contradizendo a suposição inicial da prova, e provando o teorema. □

Teorema 5.1.2 (Acordo uniforme)**.** *Processos corretos pertencentes a um mesmo grupo g instalam a mesma sequência de views $v_i(g), v_{i+1}(g), \ldots, v_n(g)$, a partir do momento em que se juntam ao grupo.*

Demonstração. A prova é realizada por indução sobre o índice i de uma *view* $v_i(g)$.

Passo básico: se dois processos corretos p_x e p_y são membros de um grupo g e instalam a *view* $v_1(g)$, e não saem do grupo, irão instalar a *view* $v_2(g)$, posterior à *view* $v_1(g)$.

Assumindo p_x e p_y membros do grupo g, o processo p_x instala a *view* $v_1(g)$ executando a *Task* 5 linhas 34 a 36, e que o processo p_y entrou no grupo g após executar um *join* processado nesta mesma *view*, a qual é instalada pelo processo ao executar a *Task* 7, linhas 49 a 52. Como os processos p_x e p_y são corretos e não solicitam sair do grupo, ambos executam a *Task* 5, chamam o consenso da linha 25 ou da linha 27, e executam as linhas 34 a 36 do algoritmo, instalando a próxima *view* $v_2(g)$ (segundo a linha 34 - $kg = kg + 1$, sendo kg o número de identificação da *view*).

Passo da indução: se dois processos corretos p_x e p_y são membros do grupo g e instalam uma *view* $v_i(g)$, e não saem do grupo, p_x e p_y irão instalar a *view* $v_{i+1}(g)$, imediatamente posterior à *view* $v_i(g)$.

Assumimos que os processos p_x e p_y são membros do grupo g e instalaram a *view* $v_i(g)$. Provamos que, como os processos são corretos e não solicitam sair do grupo, ambos irão instalar a *view* $v_{i+1}(g)$, posterior à *view* $v_i(g)$. Logo após instalar a *view* $v_i(g)$, executando a *Task* 5, linhas 21 a 41, os processos continuam a execução desta tarefa, na próxima iteração do *while* da linha 21, uma vez que nenhum dos dois processos saiu do grupo g. Os processos irão consultar a próxima operação de suas listas *tasklist-g* (como o serviço de comunicação é confiável e a primitiva multicast satisfaz a propriedade Sincronia Virtual, todas as mensagens de *join* e *leave* chegam aos processos membros do grupo, sendo inseridas em suas *tasklist-g*, embora podendo ser em ordens diferentes), e chamar o consenso na linha 25 ou 27. Como o consenso satisfaz a propriedade de terminação, os processos terminam a execução do consenso decidindo pelo mesmo valor para instalação da *view*. Após o consenso, como os processos não solicitam sair do grupo g, executam as linhas 34 a 38 do algoritmo, instalando a próxima *view* do grupo, de numeração $kg = kg + 1$ (linha 34), e sendo $kg = i$, a próxima *view* será igual a $i + 1$.

Assim, para qualquer índice i de uma *view* $v_i(g)$, dois processos corretos, membros de um grupo g e que permanecem no grupo, p_x e p_y, a próxima *view* instalada será sempre de índice $i + 1$.

□

Teorema 5.1.3 (Validade)**.** *Toda operação executada pelo serviço de membership é requisitada por algum processo e é executada apenas uma vez.*

Devido à falta de espaço deixamos esta prova para uma outra publicação.

5.2. Propriedades de *liveness*

Apresentamos nos teoremas a seguir as propriedades de Terminação e Não-trivialidade. A propriedade Terminação aqui apresentada é similar à propriedade Execução das Operações, conforme encontrada em alguns trabalhos.

Teorema 5.2.1 (Terminação)**.** *Se um processo correto p executa joinGroup para um grupo g, então, em algum momento caso permaneça correto e g existindo, p se torna membro do grupo g e instala uma view $v(g)$ deste grupo (o mesmo vale para leavegroup).*

Demonstração. Esta prova será desenvolvida por contradição assumindo que um processo correto p execute *joingroup*(g) (*Task* 3), solicitando ser membro do grupo g, que o grupo g permaneça ativo, e que o processo p nunca instale uma *view* $v(g)$ deste grupo, passando a ser membro de g.

Ao executar a *Task* 3, *joinGroup*(g), p envia uma mensagem *newView*(p_i, g, "*join*") por *multicast* para todos os processos membros do grupo g (linha 17). Como a comunicação é confiável, todos estes processos irão receber esta mensagem executando a *task* 6 - *newView* com os processos membros de g inserindo a ação p, g, "*join*" em sua *tasklist-g* (linhas 42 a 46). Quando esta ação for a primeira da *tasklist-g* de algum processo (linha 23), esse passará a chamar o consenso (linha 25), propondo esta ação. Como todos os processos membros do grupo g possuem esta ação em suas *tasklist-g*, no pior caso, quando esta entrada for a primeira em todas as *tasklist-g*, o consenso decidirá por ela, e a *view* gerada terá o processo p como membro, sendo igual à composição da *view* anterior mais o processo p. Como a ação executada foi um *join*, todos os processos que executaram o consenso executam a linha 38, enviando uma mensagem *joingroup*(CP, g, kg, *tasklist-g*) com a sua *tasklist-g*. Ao receber todas estas mensagens *joingroup*(CP, g, kg, *tasklist-g*), executando a *Task* 7 - *joingroupmessage*, p irá instalar a nova *view* deste grupo, executando as linhas 47 a 54. Em consequência, em algum momento, p irá fazer parte do grupo g, instalando uma *view* deste grupo, o que contradiz a suposição inicial. □

A prova com relação á operação *leavegroup* não será apresentada por falta de espaço.

Teorema 5.2.2 (Não-trivialidade)**.** *Se um processo p se junta a um grupo g e ele é, ou se torna, indefinidamente alcançável por um processo $q \neq p$, sendo que q já pertence ao grupo g, então, a partir de algum momento t, p estará para sempre nas views $v_i(g)$ que q instalar.*

Devido à falta de espaço esta prova não será apresentada aqui.

6. Implementação e Resultados

Foi desenvolvida uma implementação do protocolo de *membership* em ambiente simulado utilizando o *framework* HDDSS (*Hybrid and Dynamic Distributed System Simulator*) [Silva Freitas and de Araújo Macêdo 2014]. Foram implementados no simulador os algoritmos do detector de defeitos e de consenso, conforme descritos para o modelo SPA, e o algoritmo de *membership* apresentado neste artigo.

O ambiente de simulação foi configurado com canais de comunicação confiáveis, sem perdas de mensagens, e um comportamento determinístico para os canais síncronos, e probabilístico para os canais assíncronos. Todos os processos são síncronos, satisfazendo a propriedade *Strong Partitioned Synchyrony*. Os experimentos foram executados com um único grupo de comunicação, para a validação do funcionamento do protocolo. Foram utilizados dois cenários, ambos com duas partições síncronas. No primeiro cenário utilizamos 2 processos por partição, totalizando 4 processos e no segundo cenário, utilizamos 4 processos por partição, totalizando 8 processos.

Em cada experimento realizado, após a criação do grupo, são solicitadas operações de *joingroup* e de *leavegroup*, às vezes em sequência, às vezes ao mesmo tempo. A eficiência avaliada para o protocolo foi calculada em termos do tempo para a execução de

cada operação, desde a sua solicitação até a instalação de uma nova *view* pelos processos do grupo com a operação realizada. No caso de operação gerada pela detecção de um processo (*detectFaulty*), calculamos o tempo a partir da detecção da falha por um primeiro processo no grupo, até a instalação da *view*. Para cada cenário fizemos 50 repetições da execução da simulação e, a partir dos dados destas execuções chegamos aos valores apresentados nas tabelas 1 e 2. O tempo no simulador é calculado em termos de unidades de tempo do relógio interno do *framework*.

Tabela 1. Dados de média e desvio padrão da eficiência do processamento das operações para o cenário A

Cenário A			
Média Eficiência *Join* (4 processos)	30,16	Desvio Padrão Eficiência *Join* (4 processos)	9,78
Média Eficiência *leave* (4 processos)	26,84	Desvio Padrão Eficiência *leave* (4 processos)	21,48
Média Eficiência *faulty* (4 processos)	16,54	Desvio Padrão Eficiência *faulty* (4 processos)	14,87

Tabela 2. Dados de média e desvio padrão da eficiência do processamento das operações para o cenário B

Cenário B			
Média Eficiência *Join* (8 processos)	67,64	Desvio Padrão Eficiência *Join* (8 processos)	40,88
Média Eficiência *leave* (8 processos)	44,80	Desvio Padrão Eficiência *leave* (8 processos)	49,86
Média Eficiência *faulty* (8 processos)	44,32	Desvio Padrão Eficiência *faulty* (4 processos)	37,90

Avaliamos que quanto maior o número de processos em um grupo, maiores os tempos de processamento para cada operação, devido, principalmente ao tempo de troca de mensagens no consenso para estabelecimento da composição da nova *view*. A variação no número de processos no grupo, à medida em que processos entram e saem, gerou uma variação grande nos tempos para execução das operações, implicando em um desvio padrão alto. O número de processos impacta principalmente na execução do consenso, devido ao alto número de mensagens sendo trocadas, gerando maiores tempos para a transmissão das mensagens nos canais assíncronos.

Os resultados apresentados são preliminares, sendo que ajustes estão sendo realizados no sentido de otimizar a implementação do algoritmo no simulador, e novos experimentos serão realizados para próximos trabalhos, com maior número de processos, e a criação de diversos grupos em diferentes cenários.

7. Conclusões

Neste artigo apresentamos um protocolo de *membership* para o modelo Síncrono Particionado de sistemas distribuídos, com a propriedade *Strong Partitioned Synchrony* sendo satisfeita, ou seja, todos os processos pertencem a partições síncronas. O protocolo utiliza um detector de defeitos da classe P e um algoritmo de consenso. O protocolo satisfaz propriedades de *safety* e *liveness*, sendo que algumas de suas provas formais foram apresentadas.

Embora em um sistema não síncrono, o uso de um detector de defeitos da classe P, além de algoritmo de consenso desenvolvido para executar com o mesmo detector, permitem o desenvolvimento de um serviço de *membership* simples e consistente.

Em trabalhos futuros serão efetuados novos experimentos, em ambiente simulado, com o objetivo de obter uma mais ampla validação deste algoritmo. O serviço de *membership* será também estendido para também funcionar no modelo SPA caso a propriedade *Weak Partitioned Synchrony* seja satisfeita, existindo processos assíncronos, que não são membros de partições síncronas.

Referências

Anceaume, E., Charron-Bost, B., Minet, P., and Toueg, S. (1995). On the Formal Specification of Group Membership Services. Research Report RR-2695, INRIA. Projet REFLECS.

Chandra, T. D., Hadzilacos, V., Toueg, S., and Charron-Bost, B. (1996). On the impossibility of group membership. In *Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing*, pages 322–330.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267.

Chockler, G. V., Keidar, I., and Vitenberg, R. (2001). Group communication specifications: a comprehensive study. *ACM Computing Surveys (CSUR)*, 33(4):427–469.

Cristian, F. (1991). Reaching agreement on processor-group membrship in synchronous distributed systems. *Distributed Computing*, 4(4):175–187.

Cristian, F. and Fetzer, C. (1999). The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):642–657.

Ezhilchelvan, P. D., Macêdo, R. A., and Shrivastava, S. K. (1995). Newtop: a fault-tolerant group communication protocol. In *Proceedings of 15th International Conference on Distributed Computing Systems*, pages 296–306. IEEE.

Fetzer, C. and Cristian, F. (1997). A failure aware membership service. In *Proceedings of the 16th Symposium on Reliable Distributed Systems*, page 157. IEEE.

Fischer, M. J., Lynch, N. A., and Paterson, M. S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382.

Gorender, S., Macedo, R., and Raynal, M. (2005). A hybrid and adaptive model for fault-tolerant distributed computing. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 412–421. IEEE.

Gorender, S. and Macêdo, R. J. d. A. (2011). Consenso distribuído eficiente no modelo síncrono particionado. In *Anáis do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pages 587–600.

Gorender, S., Macedo, R. J. d. A., and Raynal, M. (2007). An adaptive programming model for fault-tolerant distributed computing. *IEEE Transactions on Dependable and Secure Computing*, 4(1):18–31.

Keidar, I., Sussman, J., Marzullo, K., and Dolev, D. (2002). Moshe: A group membership service for wans. *ACM Transactions on Computer Systems*, 20.

Lim, L. and Conan, D. (2014). Partitionable group membership for mobile ad hoc networks. *Journal of Parallel and Distributed Computing*, 74(8):2708–2721.

lin, k. and Hadzilacos, V. (1999). Asynchronous group membership with oracles. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 70–93.

Macêdo, R. J. d. A. and Freitas, A. E. S. (2009). A generic group communication approach for hybrid distributed systems. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 102–115. Springer.

Macêdo, R. J. d. A., Freitas, A. E. S., et al. (2011). A self-manageable group communication protocol for partially synchronous distributed systems. In *2011 5th Latin-American Symposium on Dependable Computing*, pages 146–155. IEEE.

Macêdo, R. J. d. A. and Gorender, S. (2008). Detectores perfeitos em sistemas distribuídos não síncronos. In *Anais do IX Workshop de Teste e Tolerância a Falhas (WTF 2008)*.

Macêdo, R. J. d. A. and Gorender, S. (2009). Perfect failure detection in the partitioned synchronous distributed system model. In *2009 International Conference on Availability, Reliability and Security*, pages 273–280. IEEE.

Macêdo, R. J. d. A. and Gorender, S. (2012). Exploiting partitioned synchrony to implement accurate failure detectors. *International Journal of Critical Computer-Based Systems 4*, 3(3):168–186.

Park, S., Yoo, S., Kim, Y., Lee, S., and Kim, D. (2016). A message efficient group membership protocol in synchronous distributed systems. In *Information Technology: New Generations*, pages 1249–1254. Springer.

Schiper, A. (2006). Dynamic group communication. *Distributed Computing*, 18(5):359–374.

Schiper, A. and Toueg, S. (2006). From set membership to group membership: A separation of concerns. *IEEE Transactions on Dependable and Secure Computing*, 3(1):2–12.

Silva Freitas, A. E. and de Araújo Macêdo, R. J. (2014). A performance evaluation tool for hybrid and dynamic distributed systems. *ACM SIGOPS Operating Systems Review*, 48(1):11–18.

Veríssimo, P. and Casimiro, A. (2002). The timely computing base model and architecture. *IEEE Transactions on Computers*, 51(8):916–930.