Condições de Conectividade para Realização de Acordo Tolerante a Falhas em Sistemas Auto-Organizáveis*

Fabíola Gonçalves Pereira Greve[1], **Sébastien Tixeuil**[2]

[1] Departamento de Ciência da Computação
Universidade Federal da Bahia
40.170-110 - Salvador - BA - BRASIL

[2]LRI - Laboratoire de Recherche en Informatique
Université Paris-Sud
91405 Orsay Cedex, France

Resumo. *O consenso é um problema fundamental para o desenvolvimento de soluções confiáveis em sistemas auto-organizáveis, tais como redes móveis ad-hoc. Infelizmente, em ambientes clássicos assíncronos, onde o conjunto dos participantes é conhecido, o problema do consenso não tem solução determinística na presença de uma uma única falha de processo. Assim, espera-se que a resolução deste problema em sistemas onde os participantes são desconhecidos, tenha uma complexidade ainda maior. Neste artigo, estudamos condições necessárias e propomos algoritmos suficientes para a resolução do consenso tolerante a falhas em sistemas auto-organizáveis. Tais condições são relacionadas, por um lado, aos requisitos de conectividade do grafo de conhecimento construído pelos nós para comunicação com os seus pares, e por outro lado, aos requisitos de sincronia a serem incorporados ao sistema para tolerar falhas.*

Abstract. *In self-organizing systems, consensus is a fundamental building block to solve agreement problems. It is well known that in classical environments, where identities are known, consensus cannot be solved in the presence of even one process crash. It appears that self-organizing systems are even less favorable because the set of participants are not known. We propose necessary conditions and sufficient algorithms under which fault-tolerant consensus become solvable in these environments. These are related to the synchrony requirements of the environment, as well as the connectivity of the knowledge graph constructed by the nodes in order to communicate with their peers.*

1. Introdução

Redes móveis ad-hoc (ou simplesmente, Manets), redes de sensores e, num contexto diferente, redes entre pares (P2P) não estruturadas são naturalmente auto-organizáveis e dinâmicas. Estas possibilitam acesso universal aos serviços e informações, independentemente da localização e da mobilidade dos pares, eliminando a necessidade de existir uma infra-estrutura estática e um controle centralizado. Adicionalmente, para o respeito da autonomia e da independência de processamento, os nós podem entrar ou sair da rede

*Esse trabalho tem apoio do programa de cooperação internacional CAPES-COFECUB. Fabíola tem apoio do CNPQ e Fapesb-Bahia, Brasil. Sébastien tem apoio do projeto FRAGILE e SOGEA-ANR, França.

a qualquer momento, em conformidade com os seus interesses. Finalmente, devido a escassez de recursos e formas de comunicação, falhas nessas redes são bastante frequentes.

Os algoritmos de acordo são peças fundamentais para o desenvolvimento de sistemas confiáveis e são importantes para lidar com o alto dinamismo e vulnerabilidade das redes auto-organizáveis. Eles possibilitam com que os participantes de uma computação concordem com as ações que serão realizadas, com o intuito de manter a consistência e de fazê-la progredir. O *consenso* fatoriza de maneira genérica essa necessidade de acordo. Informalmente, um grupo de processos atinge o consenso quando todos votam por um valor e os processos corretos (que não falharam) alcançam uma decisão comum sobre um dos valores propostos. Por ex., num contexto móvel, o consenso foi recentemente proposto por [Souissi et al. 2006] para coordenar as atividades de um conjunto de robôs móveis. Em [Cavin et al. 2004], propõe-se o uso do consenso para realizar um *bootstraping* na rede, *i.e.*, decidir quais nós irão prover quais serviços na inicialização do sistema.

Diferentemente das redes clássicas, onde o conjunto de participantes e suas identidades são conhecidos, nas redes auto-organizáveis não se sabe *a priori* quais são os pares com quem se pode colaborar, nem quantos deles estão disponíveis. Os protocolos de consenso clássicos para redes fixas não se adequam mais às exigências desse novo ambiente, pois têm como hipótese o conhecimento *a piori* do universo de todos os nós da rede e sua quantidade. Assim, será preciso promover uma reavaliação das estratégias comumente utilizadas para resolver este problema. É importante destacar que mesmo numa rede clássica, com comportamento assíncrono, o consenso não tem solução determinística na presença de uma única falha de processo [Fischer et al. 1985]. Assim, resolvê-lo num contexto onde o conjunto de nós é desconhecido, tem complexidade ainda maior.

Os *detectores de falhas* reunem as condições de sincronia necessárias e suficientes para a resolução do consenso em redes clássicas, onde o conjunto de participantes é conhecido [Chandra and Toueg 1996]. Informalmente, um detector de falhas é um conjunto de oráculos distribuídos que fornece dicas aos processos sobre quais deles estão falhos. Os detectores $\diamondsuit\mathcal{S}$ representam a classe mais fraca de detectores a permitir uma resolução do consenso num sistema assíncrono sujeito a falhas e à condição que uma maioria de processos esteja correta no sistema [Chandra et al. 1996]. Tais detectores podem cometer uma quantidade arbitrária de erros, mas, apesar da sua imprecisão, eles nunca irão comprometer as propriedades de exatidão do algoritmo de consenso que o utiliza como módulo subjacente. Neste caso, os algoritmos de consenso são considerados *indulgentes* em relação aos detectores [Guerraoui 2000]. Ou seja, eles foram projetados para tolerar o comportamento arbitrário destes oráculos durante períodos de assincronismo e instabilidade da rede. Adicionalmente, qualquer um dos algoritmos indulgentes que resolvem o consenso, também resolvem o consenso uniforme. Na versão uniforme, garante-se a uniformidade das decisões tomadas pelos processos, sejam eles corretos ou não.

[Cavin et al. 2004] introduziram um novo problema de nome CUP (*consenso with unknown participants*), que contempla a mesma definição do consenso clássico, excetuando-se a hipótese feita sobre o conhecimento do conjunto de processos do sistema, representado por Π. Ou seja, para resolver CUP, os processos não têm conhecimento de Π. Entretanto, para que seja possível realizar qualquer computação distribuída, os nós devem, de alguma maneira, adquirir um conhecimento parcial dos demais processos no sistema com os quais eles podem colaborar. A abstração de *detectores de participação* foi então

proposta com o intuito de alimentar os nós com este subconjunto [Cavin et al. 2004]. Tais detectores são oráculos distribuídos que provêm dicas aos processos sobre quais deles participam da computação. Por exemplo, numa rede sem fio, uma forma de implementar tais detectores seria através do uso da comunicação por difusão, inerente a este tipo de rede. Assim, para cada nó, é possível construir uma visão local formada por nós na área de cobertura da difusão (1-*hop*). Com base no grafo de conhecimento estabelecido pelos detectores de participação de todos os nós, Cavin *et al.* definem condições de conectividade necessárias e suficientes para a resolução do CUP no sistema assíncrono, porém considerando um cenário *sem falhas dos nós*.

O problema FT-CUP (CUP *tolerante a falhas*) foi posteriormente estudado por [Cavin et al. 2005]. Tomando como base as condições de conectividade mínimas para resolver CUP, os autores provam que para resolver FT-CUP será necessário enriquecer o sistema com detectores de falhas perfeitos ($\mathcal{P}$). Um detector perfeito nunca comete erros e somente pode ser implementado num sistema síncrono. Assim, para resolver FT-CUP, num cenário em que as relações de conhecimento entre os processos são as mais fracas possíveis, será necessário utilizar as mais fortes condições de sincronia. Ocorre que tais exigências de sincronismo forte competem com a natureza altamente dinâmica e totalmente descentralizada das redes auto-organizáveis, móveis e dinâmicas. Além disso, mesmo com o uso de detectores perfeitos, quando as menores condições de conectividade são consideradas, provou-se que a resolução da versão uniforme do FT-CUP é impossível [Cavin et al. 2005].

Neste trabalho, mostramos ser possível resolver o FT-CUP e a sua versão uniforme quando o sistema é enriquecido com requisitos menores de sincronia. Em particular, desejamos resolver o FT-CUP com as hipóteses de sincronia necessárias para resolver o consenso em redes clássicas, ou seja, $\diamond\mathcal{S}$. Para tanto, será preciso exigir do sistema um grau de conectividade maior entre os seus processos. Sendo assim, determinamos condições de conectividade sobre o grafo de conhecimento que são suficientes, mas também necessárias para a resolução do FT-CUP. Tais condições são expressas através de novas classes de detectores de participação e a sua suficiência é atestada através de algoritmos que resolvem o problema. De fato, se o sistema satisfaz as condições de conectividade aqui estudadas, qualquer algoritmo de consenso indulgente, anteriormente proposto para redes clássicas [Chandra and Toueg 1996], poderá ser utilizado como algoritmo subjacente, tanto para resolver o FT-CUP, quanto o FT-CUP uniforme.

O resto deste artigo contempla as seguintes seções: a seção 2 apresenta o modelo, define o problema de consenso para redes desconhecidas e apresenta abstrações para a sua resolução; a seção 3 apresenta os algoritmos e descreve as condições necessárias e suficientes para resolução do FT-CUP e FT-CUP uniforme; a seção 4 conclui o trabalho.

2. Modelo

Redes, Processos e Canais. Consideramos um sistema distribuído formado por um conjunto finito Π de $n > 1$ processos[1], $\Pi = \{p_1, \ldots, p_n\}$. Numa *rede conhecida*, todo processo do sistema conhece Π e n. Numa *rede desconhecida*, um processo p_i conhece apenas um subconjunto Π_i de Π. Os processos comunicam-se pela emissão e recepção de mensagens através de canais confiáveis, que não alteram, nem duplicam as mensagens

[1]Os termos "processo", "nó" e "vértice"serão utilizados indistintamente neste trabalho.

que ali trafegam. Um processo p_i apenas pode enviar mensagens para outro processo p_j se $p_j \in \Pi_i$. Evidentemente que, se p_i envia uma mensagem para p_j, tal que $p_i \notin \Pi_j$, então, p_j, após receber a mensagem, pode adicionar p_i em Π_j. A partir deste conhecimento, p_j pode enviar mensagens para p_i. Implicitamente, considera-se que existe uma camada de roteamento confiável responsável por encaminhar as mensagens entre os processos que se conhecem. Nenhuma hipótese temporal existe no que diz respeito à realização das ações efetuadas pelos processos ou pelos canais. Ou seja, o sistema é *assíncrono*. Um processo pode falhar por parada (*crash*), através de um colapso brusco ou saída deliberada (*switched off*). Inicialmente, todos os processos estão ativos e se comportam conforme a sua especificação. Quando um processo falha, torna-se inativo e mantém-se nesse estado durante todo o resto da execução. Um processo que nunca está inativo é considerado *correto*; de outra maneira, ele é considerado um processo *faltoso*. Seja f o número máximo de processos autorizados a falhar no sistema, então f é conhecido por todos os processos.

Consenso Clássico. No consenso, cada processo correto p_i propõe um valor v_i e todos os processos corretos devem "decidir" por um único valor v, dentre os propostos. Formalmente, ele é definido pelo seguinte conjunto de propriedades [Chandra and Toueg 1996].

- C_Terminação: todo processo correto decide de maneira definitiva;
- C_Validade: se um processo decide v, então v foi proposto por algum processo;
- C_Acordo: dois processos corretos não decidem diferentemente.

Consenso Uniforme. Garante-se a uniformidade da decisão. Logo, o acordo modifica-se:
- C_Acordo_Uniforme: dois processos, *corretos ou não*, não decidem diferentemente.

Consenso em Redes Desconhecidas. Pretende-se resolver o problema do consenso em redes desconhecidas, sujeitas a falhas. Três variantes são definidas:
CUP (Consenso com Participantes Desconhecidos). O objetivo é resolver o consenso em redes desconhecidas, considerando-se que os processos *não* podem falhar;
FT-CUP (CUP Tolerante a Falhas). O objetivo é resolver o consenso em redes desconhecidas, considerando-se que até f processos podem falhar (onde f é uma constante);
FT-CUP Uniforme. O objetivo é resolver a versão *uniforme* de FT-CUP.

2.1. Condições de Sincronia e de Conectividade para Concordar Apesar das Falhas

2.1.1. Detectores de Falhas: Abstração para Sincronia

O consenso não tem solução num sistema assíncrono, mesmo em presença de uma única falha [Fischer et al. 1985]. Para contornar tal impossibilidade deve-se estender o modelo assíncrono com algum grau de sincronia. Com este intuito, um dos avanços mais significativos é a proposta de uso dos *detectores de falhas* [Chandra and Toueg 1996] (denotado por FD). Formalmente, ele é definido por duas propriedades: a completude e a exatidão. A *completude* assegura que processos faltosos terminarão por ser suspeitos. A *exatidão* restringe os equívocos que podem ser cometidos pelo detector. A combinação das duas propriedades origina algumas classes de detectores. Neste trabalho, considera-se:

FD Forte Após um Tempo ($\diamondsuit\mathcal{S}$). Esta classe garante que todo processo falho será finalmente suspeito por todos os processos corretos (*completude forte*); além disso, existirá um instante a partir do qual algum processo correto não será considerado suspeito por nenhum outro processo correto (*exatidão fraca após um tempo*).

FD Perfeito ($\mathcal{P}$). Esta classe garante que o detector nunca comete erros. Ou seja,

ele satisfaz a propriedade *exatidão forte definitiva*, que assegura que um processo nunca será suspeito antes que falhe, além da *completude forte*.

2.1.2. Detectores de Participação: Abstração para Conectividade por Conhecimento

Com as notáveis exceções de [Cavin et al. 2004, Cavin et al. 2005], os trabalhos sobre consenso presentes na literatura consideram que o conjunto de processos no sistema é conhecido por todos os demais processos. Ocorre que em sistemas móveis, auto-organizáveis e altamente dinâmicos, esta hipótese é irrealista. De fato, não existe uma entidade única que alimente os processos com informação sobre a topologia da rede. Entretanto, para realizar qualquer computação distribuída, os processos precisam descobrir-se de alguma maneira. Por exemplo, através da troca de mensagens do tipo "hello", onde a identidade do nó é passada, eles poderão conhecer ao menos aqueles nós da sua vizinhança. Os *detectores de participação* (denotados por PD) foram propostos com o intuito de representar esse conhecimento sobre os processos do sistema [Cavin et al. 2004]. Semelhante aos detectores de falhas, tais detectores são considerados oráculos distribuídos que fornecem dicas aos processos sobre quais processos são participantes da computação. Seja $i.\mathsf{PD}$ o detector de participação de p_i. Quando consultado por p_i, $i.\mathsf{PD}$ retorna um subconjunto de processos de Π com os quais ele pode colaborar. A informação fornecida por $i.\mathsf{PD}$ pode evoluir ao longo das consultas. Seja $i.\mathsf{PD}(t)$ o resultado de uma consulta de p_i no tempo t. Esta consulta pode ser incompleta, porém ela é sempre crescente e precisa. Assim, deve satisfazer a duas propriedades :

• Inclusão da Informação. A informação retornada por PD é não decrescente ao longo do tempo: $p_i \in \Pi, t' \geq t : i.\mathsf{PD}(t) \in i.\mathsf{PD}(t')$.

• Exatidão da Informação. PD não comete erros: $\forall p_i \in \Pi, \forall t : i.\mathsf{PD}(t) \in \Pi$.

A informação retornada por PD enriquece o sistema com um grafo de conectividade por conhecimento. Este grafo é orientado, pois a relação de conhecimento não é necessariamente bidirecional. Assim, se $p_j \in i.\mathsf{PD}$, então não necessariamente $p_i \in j.\mathsf{PD}$.

Definição 1 (Grafo de Conectividade por Conhecimento) *Seja $G_{di}(V, E)$ um grafo orientado representando a relação de conhecimento estabelecida por um oráculo* PD. *Então, $V = \Pi$ e $(p_i, p_j) \in E$ se e somente se $p_j \in i.\mathsf{PD}$, i.e., p_i conhece p_j.*

Definição 2 (Grafo Não-Orientado de Conectividade por Conhecimento) *Seja $G(V, E)$ um grafo não orientado representando a relação de conhecimento determinada pelo oráculo* PD. *Então, $V = \Pi$ e $(p_i, p_j) \in E$ se e somente se $p_j \in i.\mathsf{PD}$ ou $p_i \in j.\mathsf{PD}$.*

A partir das propriedades apresentadas pelo grafo de conhecimento, algumas classes de detectores de participação foram propostas por [Cavin et al. 2004] para resolução de CUP. Neste trabalho, com o intuito de resolver o consenso em presença de falhas (FT-CUP), introduzimos duas novas classes de detectores de participação, estas são:

PD k-Fortemente Conexo (k-SCO). O grafo orientado G_{di}, que representa a relação de conhecimento induzida pelo oráculo PD é k-fortemente-conexo[2].

PD k-Redutível a Único Poço (k-OSR). O grafo orientado G_{di}, que representa a relação de conhecimento induzida pelo oráculo PD, satisfaz as seguintes condições:

[2] G_{di} é *k-fortemente conexo* se para quaisquer par de vértices (v_i, v_j), v_i pode alcançar v_j através de k caminhos distintos nos vértices.

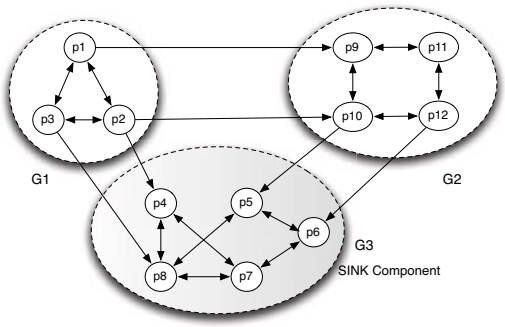

Figura 1. Grafo de Conhecimento Induzido pelo Detector de Participação 2-OSR

1. O grafo de conectividade por conhecimento G, obtido de G_{di}, é conexo;

2. O grafo direcionado acíclico, obtido pela redução de G_{di} às suas componentes k-fortemente conexas, tem uma e somente uma componente poço[3].

3. Considere quaisquer duas componentes k-fortemente conexas G_1 e G_2, se existe um caminho de G_1 para G_2, então existem k caminhos disjuntos nos nós de G_1 para G_2.

A Figura 1 ilustra um grafo G_{di} induzido por um $\mathsf{PD} \in k$-OSR, considerando-se $k = 2$. Observe que existe uma única componente poço (G_3) e que cada componente G_i é 2-fortemente-conexa.

3. Condições de Conectividade e Sincronia para Resolução de FT-CUP

Nesta seção, mostramos como o FT-CUP pode ser resolvido com base nos novos detectores de participação k-SCO e k-OSR e dado que o sistema satisfaz as condições de sincronia minimais para a resolução do consenso em redes conhecidas, *i.e.*, $\diamondsuit\mathcal{S}$. Através de uma consulta inicial aos detectores de participação, os nós terão uma visão parcial dos participantes do sistema. Esta visão é materializada na forma do grafo de conectividade por conhecimento. Se esse grafo satisfaz determinadas propriedades (definidas pelas classes de detectores), é possível resolver FT-CUP. Por exemplo, intuitivamente, para que o consenso possa ser resolvido, é necessário que o grafo seja conexo. Considerando a falha de no máximo f nós ($f < k$), apresentamos proposições e algoritmos que mostram que *(i)* k-OSR é necessário para resolver FT-CUP (Proposição 1); *(ii)* k-SCO é suficiente para resolver FT-CUP Uniforme (Proposição 2) e *(iii)* k-OSR é suficiente para resolver FT-CUP Uniforme (Proposição 3). Os algoritmos sugeridos e o argumento principal de algumas provas serão apresentados nas próximas seções. O leitor interessado, encontrará no relatório [Greve and Tixeuil 2006] as provas completas das proposições enunciadas.

3.1. Condições Necessárias para Resolução de FT-CUP

Proposição 1 *O detector de participação k-OSR é necessário para resolver FT-CUP, dado que $f < k < n$ nós podem falhar.*

Prova: Considere por contradição que existe um algoritmo $\mathcal{A}$ que resolve FT-CUP através de um detector $\mathsf{PD} \notin k$-OSR. Seja G_{di} o grafo de conhecimento induzido por PD e decomposto em suas componentes k-fortemente conexas. Os seguintes cenários são

[3]Uma componente G_l de G_{di} é considerada *poço* (*sink*) quando não existem caminhos saindo de vértices em G_l para vértices pertencentes a outras componentes de G_{di}, diferentes de G_l.

possíveis: (i) ou existem menos do que k caminhos disjuntos nos nós entre duas componentes de G_{di}; (ii) ou a decomposição de G_{di} origina mais de um poço. No primeiro cenário, a falha de $k-1$ nós pode desconectar o grafo; como a conexão é necessária para resolver o consenso [Cavin et al. 2004], alcançamos uma contradição. No segundo cenário, sejam G_1 e G_2 duas das componentes poços. Considere que todos os nós em G_1 têm valor de proposição igual a v e que todos em G_2 têm valor igual a w, $v \neq w$. Pela propriedade de *terminação* do consenso, os nós em G_1 decidem num tempo $t1$ e em G_2, eles decidem num tempo $t2$. Podemos atrasar a recepção de mensagens provenientes de outras componentes para ambos os nós em G_1 e G_2 para um tempo $t > max\{t1, t2\}$. Como os nós no poço não sabem da existência de outros nós, pela propriedade de *validade* do consenso, os nós em G_1 decidem pelo valor v e os nós em G_2 decidem pelo valor w, violando assim o *acordo*, pois $v \neq w$, e alcançando uma contradição. □

3.2. Condições Suficientes para Resolução de FT-CUP Uniforme

Princípio. Para a resolução de FT-CUP, cada processo p_i deve, inicialmente, realizar uma consulta ao seu detector $i.\mathsf{PD}$. Esta consulta é realizada *exatamente uma vez*. Tal medida contribui para definir um grafo G_{di} de conectividade por conhecimento que seja único e comum para todos os participantes do sistema. Note que $i.\mathsf{PD}$ alimenta p_i com os nós com os quais ele está diretamente relacionado no grafo G_{di}. A fim de conhecer outros nós do sistema, p_i deve efetuar um procedimento de prospecção em G_{di}. Tal pesquisa lhe trará o aprendizado sobre novos processos com os quais ele está indiretamente ligado através de algum caminho em G_{di}. O algoritmo COLLECT (apresentado como algoritmo 1) tem esse objetivo de ampliar o grau de conhecimento dos processos. Ao término da sua execução, qualquer processo correto p_i consegue determinar um conjunto maximal de participantes que ele pode alcançar através do grafo G_{di} e com os quais ele pode eventualmente se comunicar. A depender das características de G_{di}, existem duas possibilidades.

• $\mathsf{PD} \in k\text{-}SCO$, ou seja, se o grafo G_{di} é k-fortemente conexo, garante-se que o conjunto retornado por COLLECT será formado por *todos* os processos corretos participantes do sistema, ou seja, Π. Nesse caso, como todos os processos corretos se conhecem, é trivial resolver FT-CUP Uniforme. Basta, em seguida, utilizar qualquer um dos algoritmos clássicos de consenso uniforme para redes conhecidas [Chandra and Toueg 1996].

• $\mathsf{PD} \in k\text{-}OSR$, ou seja, se o grafo G_{di}, redutível a suas componentes k-fortemente conexas, tem um e somente um poço, garante-se que o conjunto retornado por COLLECT conterá todos os processos da componente conexa G_l de p_i, além de todos os processos das outras componentes alcançáveis a partir de G_l (o que inclui, necessariamente todos os processos da componente poço). No exemplo da Figura 1, COLLECT irá retornar para $p_i \in G_1$, um subconjunto contendo $p_j \in \{G_1 \cup G_2 \cup G_3\}$; para $p_i \in G_2$, um subconjunto contendo $p_j \in \{G_2 \cup G_3\}$; para $p_i \in G_3$, um subconjunto contendo $p_j \in \{G_3\}$. Desta maneira, quando $\mathsf{PD} \in k\text{-}OSR$, os processos são alimentados por COLLECT com uma visão *parcial* do sistema e essa visão ainda não é suficiente para resolver FT-CUP. É importante notar que após a coleta, todos os processos que estão na componente poço conhecem-se apenas a si próprios. Logo, todos os processos do sistema conhecem os processos do poço, mas estes só se conhecem a si mesmos. Nessas circunstâncias, uma maneira natural de resolver FT-CUP seria a seguinte. Inicialmente, determina-se quem faz parte da componente poço. Posteriormente, realiza-se um consenso somente entre os nós do poço. Finalmente, faz-se com que os demais processos do sistema coletem a decisão tomada através de mensagens de requisição aos membros do poço. Seguindo essa

estratégia, propomos a realização do algoritmo SINK (algoritmo 2) para determinar se um nó pertence ou não à componente poço, seguido da realização do algoritmo CONSENSUS (algoritmo 3) para efetuar o consenso propriamente dito.

Discussão. Na solução apresentada, nós poderão entrar ou sair do sistema, modificando G_{di} da seguinte maneira. A saída de um nó (remoção em G_{di}) só será suportada dentro do limite das f falhas. Na prática, isto significa que o nó pode se mover, mas não poderá, por exemplo, sair da área de cobertura da rede. A entrada de um nó (inserção em G_{di}) deve respeitar as condições k-OSR para G_{di}. Assim, o novo nó poderá detectar a participação de alguns nós de G_{di}, porém, ele não será reconhecido por nenhum dos seus pares (já que, para estes, as relações de conhecimento já foram definidas). Logo, o novo nó apenas terá arestas de saída e nenhuma de entrada. Mudanças na topologia do sistema só serão consideradas na próxima resolução do consenso, pois os nós que já fazem parte de G não irão modificar as suas relações de conhecimento até que haja uma convergência para uma decisão. Assim, a cada novo consenso, deve haver a construção de um novo grafo.

3.2.1. Algoritmo COLLECT: Ampliação do Conhecimento dos Processos no Sistema

Princípio. No início do algoritmo, p_i consulta o seu detector para obter i.PD. Em seguida, através de um procedimento de descoberta do grafo, p_i irá requisitar aos novos processos as suas relações de conhecimento, até que nenhuma nova informação possa ser obtida. O algoritmo executa naturalmente em rodadas. Em cada rodada $r > 0$, p_i contata todos os nós que ele não conhece e que acabou de tomar conhecimento na rodada $r-1$. Na rodada 0, p_i conhece apenas ele mesmo; na rodada 1, ele conhece todos os que são retornados pelo seu PD, ou seja, i.PD; na rodada 2, ele conhece quem os processos em i.PD conhecem e assim sucessivamente, até que não possa mais incrementar o seu conhecimento.

Variáveis. O processo p_i gerencia as seguintes variáveis locais:
- $i.known$: subconjunto de processos conhecido por p_i na rodada corrente;
- $i.responded$: subconjunto de processos através dos quais p_i recebeu uma mensagem;
- $i.previously\_known$: conjunto de processos conhecido por p_i na rodada anterior;
- $i.wait$: número de processos os quais p_i ainda está esperando mensagens.

Descrição. Um processo p_i inicia o algoritmo pela fase INIT (linhas 13-15). Inicialmente, p_i consulta o seu detector, sendo que a lista retornada (i.PD) será armazenada em $i.known$. Em seguida, é feita uma chamada ao procedimento *Inquiry()*. Neste, p_i irá enviar para todos os nós que ele conhece a sua visão parcial do sistema; o que é feito através da emissão da mensagem VIEW(i, $i.known$) a cada p_j conhecido (linhas 9-10). Posteriormente, variáveis locais são atualizadas; em particular, atualiza-se $i.wait$ com a quantidade de nós para os quais espera-se uma resposta, ou seja, $|i.known - f|$, que corresponde ao número de corretos no sistema segundo p_i. Na fase IMPROVEMENT, após o recebimento da mensagem VIEW($m.initiator$, $m.known$) de p_j para p_i, duas situações são possíveis:

- $m.initiator \neq i$: significa que p_i recebeu uma mensagem de um nó remoto p_j solicitando a sua visão da composição. Assim, p_i retorna a p_j seu conjunto i.PD (linha 28).
- $m.initiator = i$: significa que p_i recebeu uma resposta de p_j contendo a sua listas de conhecidos. Neste caso, p_i irá incrementar a sua visão, estendendo $i.known$ com j.PD e atualizando as variáveis $i.responded$ e $i.wait$ (linhas 18-20). Posteriormente, através do teste do predicado ($i.wait = 0$), p_i determina se recebeu mensagens de resposta proveni-

Algorithm 1 COLLECT()

constant:
(1) f: int *// upper bound on the number of crashes*
variables:
(2) $i.previously\_known$: set of nodes
(3) $i.known$: set of nodes
(4) $i.responded$: set of nodes
(5) $i.wait$: int
message:
(6) VIEW message:
(7) *initiator*: node
(8) *known*: set of nodes
procedure:
Inquiry():
(9) **for** j in $i.known \setminus i.previously\_known$ **do**
(10) SEND VIEW $(i, i.known)$ to p_j; **end do**
(11) $i.wait = |i.known \setminus i.responded| - f$;
(12) $i.previously\_known = i.known$;

***** Initiator Only *****
INIT:
(13) $i.known = i.\mathsf{PD}$;
(14) $i.responded = i.previously\_known = \{\}$;
(15) call upon *Inquiry* ();

***** All Nodes *****
IMPROVEMENT:
(16) **upon receipt of** VIEW$(m.initiator, m.known)$ **from** p_j **to** p_i:
(17) **if** $i == m.initiator$ **then**
(18) $i.known = i.known \cup m.known$;
(19) $i.responded = i.responded \cup \{j\}$;
(20) $i.wait = i.wait - 1$;
(21) **if** $i.wait == 0$ **then**
(22) **if** $i.previously\_known == i.known$ **then**
(23) *return* $(i.known)$;
(24) **else**
(25) call upon *Inquiry()*; **end if**
(26) **end if**
(27) **else**
(28) *send* VIEW$(m.initiator, i.\mathsf{PD})$ to p_j;
(29) **end if**

entes de todos os nós corretos (linha 21). Em caso positivo, duas situações são possíveis.

• $i.previously\_known = i.known$: então p_i coletou informações de todos os corretos, pois entre a rodada corrente e a rodada anterior, nenhuma nova informação foi adquirida. Neste caso, o algoritmo termina e p_i retorna seu conjunto $i.known$ (linha 23).

• $i.previously\_known \neq i.known$: então p_i descobriu novos nós. Assim, ele irá iniciar uma nova rodada para incrementar mais ainda o seu conhecimento, a partir das informações trazidas pelos novos nós do universo $\{i.known \setminus i.previously\_known\}$. Desta maneira, inicia-se um nova rodada. Nesse caso, p_i chama novamente ***Inquiry()*** para enviar a mensagem VIEW(i, $i.known$) a cada novo nó descoberto na rodada anterior (linhas 9-10). Posteriormente, algumas variáveis locais são atualizadas; em particular, atualiza-se $i.wait$ com o número de nós para os quais espera-se o recebimento de respostas à nova solicitação de p_i, ou seja, $i.known$, excluindo-se aqueles que já responderam em rodadas anteriores ($i.responded$), diminuída de f (linha 11). Finalmente, $i.previously\_known$ recebe $i.known$, que corresponde ao conhecimento acumulado na rodada anterior (linha 12).

Lema 1 *Seja G_{di} um grafo de conhecimento induzido por um detector de participação. Seja $f < k < n$ o número máximo de nós que podem falhar no sistema. O algoritmo* COLLECT *(1) executado por cada nó do sistema, satisfaz as seguintes propriedades:*

- Terminação*: cada nó p_i termina a execução e retorna uma lista contendo nós conhecidos de p_i;*
- Exatidão*: o algoritmo* COLLECT *retorna o conjunto maximal de nós corretos alcançáveis a partir de p_i no grafo de conhecimento G_{di}.*

Prova: A demonstração desse Lema passa pela constatação de que a cada nova rodada r, o conjunto $i.known$ é acrescido de novos processos cujo *distância* em relação a p_i no grafo G_{di} é igual a r. De fato, o algoritmo COLLECT executa uma espécie de *busca em largura* no grafo. Como o grafo é finito, o algoritmo termina. Além disso, cada novo nível alcançado na busca incorpora a $i.known$ vértices v, tal que $distancia(p_i, v) = r$. Desta maneira, a prova pode ser feita por indução no número de rodadas, iniciando-se por $r = 1$, onde fica evidente que $i.known$ contempla todo $w \in i.\mathsf{PD}$, logo $distancia(p_i, w) = 1$. A prova completa deste Lema, encontra-se em [Greve and Tixeuil 2006]. □

Proposição 2 *O detector de participação k-SCO é suficiente para resolver o FT-CUP uniforme, apesar de $f < k < n$ falhas de nós e dado que o sistema é enriquecido com o detector de falhas $\Diamond\mathcal{S}$.*

Prova: Se $\mathsf{PD} \in k$-SCO, o grafo de conhecimento tem apenas uma componente k-fortemente conexa. Assim, ao término da execução de COLLECT, pelo Lema 1, cada nó p_i será alimentado com o conjunto de todos os nós corretos do sistema (Π), apesar da falha de $f < k$ nós. Com base nesse conhecimento, algoritmos indulgentes para a resolução do consenso em redes conhecidas podem ser utilizados [Chandra and Toueg 1996]. Em particular, se $f < n/2$ e $f < k < n$, é possível resolver FT-CUP, assim como FT-CUP uniforme, num sistema enriquecido com ambos os detectores: k-SCO e $\Diamond\mathcal{S}$. □

Como apresentado, se o detector de participação pertence à classe k-OSR, as informações fornecidas pelo COLLECT não serão suficientes para resolver o FT-CUP. Para resolver o problema, inicialmente, determina-se quais nós pertencem à componente poço (execução de SINK), para em seguida, efetuar o consenso (execução de CONSENSUS).

3.2.2. Algoritmo SINK: Determinação dos Processos na Componente Poço

Com base na visão parcial do sistema retornada pelo COLLECT, tem-se que os nós no poço terão a mesma visão, enquanto que os nós nas demais componentes terão necessariamente mais conhecimento do que aqueles que estão no poço. A partir desta constatação, o algoritmo SINK determina se o processo p_i pertence ou não ao poço da seguinte forma.

Algorithm 2 SINK ()

constants:
(1) f: upper bound on the number of crashes
variables:
(2) $i.known$: set of nodes
(3) $i.in\_the\_sink$: boolean
(4) $i.responded$: set of nodes
messages:
(5) REQUEST message:
(6) *known*: set of nodes
(7) RESPONSE message:
(8) *ack/nack*: boolean

***** All Nodes *****
INIT:
(9) $i.known = \mathsf{COLLECT}()$;
(10) $i.responded = \{\}$;
(11) **for** each j in $i.known$ **do**
(12) send REQUEST $(i.known)$ to p_j; **endfor**

VERIFICATION:
(13) **upon receipt of** REQUEST **(**$m.known$**) from** p_j :
(14) **if** $m.known == i.known$ **then**
(15) send RESPONSE (ack) to p_j;
(16) **else**
(17) send RESPONSE $(nack)$ to p_j; **endif**

(18) **upon receipt of** RESPONSE (m) **from** p_j :
(19) **if** $m.ack$ **then**
(20) $i.responded = i.responded \cup \{j\}$;
(21) **if** $|i.responded| >= |i.known| - f$ **then**
(22) $i.in\_the\_sink = \text{true}$;
(23) *return* $(i.in\_the\_sink, i.known)$; **endif**
(24) **else**
(25) $i.in\_the\_sink = \text{false}$;
(26) *return* $(i.in\_the\_sink, i.known)$;
(27) **endif**

Descrição. Um nó p_i inicia o algoritmo pela execução de INIT (linhas 9-12). Nesta fase, o COLLECT é executado e retorna a lista parcial de nós conhecidos por p_i. Esta lista é armazenada em $i.known$ (linha 9). Em seguida, p_i envia uma mensagem de REQUEST($i.known$) para cada nó p_j conhecido (linhas 11-12).

Na fase VERIFICATION, no recebimento da mensagem REQUEST($m.known$) proveniente de p_j, p_i testa se seu próprio conjunto $i.known$ é igual ao $m.known$ de p_j. Neste caso, p_i responde *ack* a p_j, indicando que os dois nós têm a mesma visão parcial do sistema (linha 15). Caso contrário, p_i retorna uma resposta negativa *nack* (linha 17). Quando a mensagem RESPONSE(*ack/nack*) é recebida de p_j, p_i determina se pertence ou não ao poço da seguinte maneira. Se p_j responde *nack*, então p_i identificou nós (incluindo o próprio p_j) pertencentes a outras componentes. Logo, p_i deduz que não pertence ao poço (Lema 1) e nesse caso, termina a execução retornando *false* (linhas 25-26). Se p_j responde *ack*, então p_j tem a mesma visão que p_i sobre os nós alcançáveis no sistema. Se, adicionalmente, p_i recebe mensagens *a*ck de cada nó correto na sua visão, p_i pode concluir que ele está contido no poço. Assim, na recepção de um *ack*, p_i atualiza $i.responded$ para contabilizar a resposta de p_j, passando ao teste da condição ($|i.responded| >= |i.known| - f$) para saber se ele recebeu respostas de cada nó correto. Quando esta condição torna-se verdadeira, p_i tem certeza que ele pertence ao poço (Lema 1) e pode terminar a execução, retornando *true* (linhas 20-23).

3.2.3. Algoritmo CONSENSUS: Realização do Acordo

O protocolo CONSENSUS é apresentado no algoritmo 3. Na fase inicial, cada nó executa o algoritmo SINK (2) para obter uma visão parcial da composição do sistema e, além disso, decidir se ele pertence ou não à componente poço k-fortemente conexa (linha 11). A depender deste resultado, dois comportamentos são possíveis. Os nós que pertencem à componente poço irão lançar um protocolo de consenso para obtenção da decisão (execução da fase AGREEMENT). Por construção, todos os nós da componente poço compartilham a mesma visão do sistema e, pelo menos, todos os corretos irão realizar esse acordo (linhas 16-17). Os demais nós (nas componentes restantes) não participam deste consenso. Eles iniciam uma fase de REQUEST para solicitar e coletar o valor de decisão aos membros do poço. Isto é feito através da emissão de mensagens de solicitação para os nós conhecidos (pertencentes ao seu conjunto $i.known$) (linhas 27-28). Como, ao menos um nó do poço está correto ($f < k$), ao menos um nó irá receber e atender a solicitação de requisição do valor de decisão, seja executando as linhas 22-26, seja executando as linhas 19-20.

Proposição 3 *O detector de participação k-OSR é suficiente para resolver o FT-CUP uniforme, apesar de $f < k < n$ falhas de nós e dado que o sistema é enriquecido com o detector de falhas $\Diamond\mathcal{S}$ e possui uma maioria de processos corretos na componente poço.*

Prova: A prova de que o algoritmo 3 resolve FT-CUP uniforme segue trivialmente das propriedades de *terminação*, *validade* e *acordo_uniforme*, asseguradas pelo consenso subjacente utilizado no CONSENSUS (linhas 16-17). Neste caso, se existem ao menos $2f + 1$ processos na componente poço, é possível resolver FT-CUP, assim como FT-CUP uniforme, num sistema enriquecido com ambos os detectores: k-OSR e $\Diamond\mathcal{S}$, utilizando-se qualquer consenso indulgente proposto [Chandra and Toueg 1996]. □

Complexidade dos Algoritmos. O custo computacional dos algoritmos (número de mensagens e latência para decidir) depende do grafo de conhecimento G_{di} formado pelos nós e este depende da topologia da rede. Embora uma análise teórica, no pior caso, possa ser

efetuada, ela só terá significado se acompanhada de uma análise real, baseada numa topologia específica (Manets, P2P, etc.). Assim, para o nó p_i, COLLECT tem latência máxima equivalente à *excentricidade* de p_i (máxima distância de p_i em G_{di}); o número de mensagens é proporcional à qtde de nós alcançáveis ($|i.known|$); como p_i pode conhecer os n nós, este valor será de $O(n)$. Em SINK, esses valores também dependem de $|i.known|$. O CONSENSUS tem a sua complexidade dependente do consenso clássico subjacente.

4. Considerações

Neste artigo, mostramos que para resolver o consenso numa rede auto-organizável, existe uma solução de compromisso entre o grau de conhecimento exigido sobre os participantes do sistema e o grau de sincronia a ser incorporado ao sistema. Se a conectividade de conhecimento estabelecida pelos nós está em k-OSR, então é possível resolver FT-CUP, assim como FT-CUP uniforme, exigindo condições de sincronia minimais, representadas por $\diamondsuit\mathcal{S}$. De fato, k-OSR também representa o nível de conectividade minimal exigida sobre o grafo para resolver FT-CUP. O modelo adotado para a resolução do FT-CUP é flexível o suficiente para permitir mobilidade dos nós, porém não forte o suficiente para aceitar entradas e saídas aleatórias. De fato, a identificação de um modelo que leve em conta alto dinamismo e que satisfaça requisitos fortes de consistência, resta um problema em aberto. Note que, o que se busca, não é um consenso que possa envolver todos os participantes do sistema, até porque estes são desconhecidos. O objetivo é identificar condições favoráveis que permitam a convergência do consenso por um subconjunto de nós, em determinando momento da execução do sistema.

Referências

Cavin, D., Sasson, Y., and Schiper, A. (2004). Consensus with unknown participants or fundamental self-organization. In *Proc. of the 3rd Int. Conf. on AD-NOC Networks & Wireless (ADHOC-NOW'04)*, pages 135–148, Vancouver. Springer-Verlag.

Cavin, D., Sasson, Y., and Schiper, A. (2005). Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Research Report IC/2005/026, EPFL.

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

Chandra, T. D., Hadzilacos, V., and Toueg, S. (1996). The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722.

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

Greve, F. and Tixeuil, S. (2006). Knowledge connectivity *vs.* synchrony requirements for fault-tolerant agreement in unknown networks. Research Report 2006-12, DCC-UFBA, em https://twiki.im.ufba.br/bin/view/Gaudi/Publicacoes.

Guerraoui, R. (2000). Indulgent algorithms. In *Proc. of the* 19^{th} *ACM Symp. on Principles of Distributed Computing (PODC'00)*, pages 289–298, Portland, USA.

Souissi, S., Défago, X., and Yamashita, M. (2006). Gathering asynchronous mobile robots with inaccurate compasses. In *Proc. 10th Intl. Conf. on Principles of Distributed Systems (OPODIS 2006)*, LNCS, Bordeaux, France. Springer.

Algorithm 3 CONSENSUS

constant:
(1) f: upper bound on the number of crashes
input:
(2) $i.initial$: value
variable:
(3) $i.in\_the\_sink$: boolean
(4) $i.known$: set of nodes;
(5) $i.decision$: value
(6) $i.asked$: set of nodes
message:
(7) REQUEST message.
(8) RESPONSE message:
(9) *decision*: value

*** All Nodes ****
task T1: { *Main Decision Task* }
(10) $i.asked = \{\}$; $i.decision = \perp$;
(11) $(i.in\_the\_sink, i.known) = \mathsf{SINK}()$;
(12) **if** $i.in\_the\_sink$ **then**
(13) fork AGREEMENT
(14) **else**
(15) fork REQUEST **end if**

*** Node In Sink ****
AGREEMENT: { *make use of classical consensus* }
(16) Consensus.propose($i.initial$)
(17) **upon** Consensus.decide(v):
(18) $i.decision = v$;
(19) **for** every j in $i.asked$ **do**
(20) *send* RESPONSE $(i.decision)$ to p_j; **end for**
(21) *return* $(i.decision)$;

task T2: { *Decision Dissemination Task* }
(22) **upon receipt of** REQUEST() **from** p_j:
(23) **if** $i.decision \neq \perp$ **then**
(24) *send* RESPONSE $(i.decision)$ to p_j;
(25) **else**
(26) $i.asked = i.asked \cup \{j\}$; **end if**

*** Node Not In Sink ****
REQUEST:
(27) **for** every j in $i.known$ **do**
(28) *send* REQUEST () to j
(29) **upon receipt of** RESPONSE (v) **from** j:
(30) **if** $i.decision = \perp$ **then**
(31) $i.decision = v$;
(32) *return* $(i.decision)$; **end if**