O Elo Perdido: Um Modelo de Diagnóstico Distribuído para a Implementação de Detectores de Falhas Não Confiáveis

Elias P. Duarte Jr.[1], **Luiz A. Rodrigues**[2], **Edson T. Camargo**[3], **Rogério Turchetti**[4]

[1]Universidade Federal do Paraná (UFPR)
Depto. Informática, Curitiba E-mail: elias@inf.ufpr.br

[2]Universidade Estadual do Oeste do Paraná
(UNIOESTE) Cascavel, PR E-mail: luiz.rodrigues@unioeste.br

[3]Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Toledo, PR E-mail: edson@utfpr.edu.br

[4]Universidade Federal de Santa Maria (UFSM)
Santa Maria, RS E-mail: turchetti@redes.ufsm.br

Abstract. *Monitoring computer systems for fault identification is essential for building reliable systems. System-level diagnosis was initially proposed in the 1960s as a test-based approach to monitor and identify faulty components. Over the last decades, several diagnosis models and strategies have been proposed, based on different fault models, applied to the most diverse types of computer systems. In the 1990s, unreliable failure detectors emerged as an abstraction to allow the execution of consensus in asynchronous systems subject to crash faults. Since then, failure detectors have become the* de facto *standard for monitoring distributed systems. The purpose of the present work is to close a conceptual gap, presenting a distributed diagnosis model consistent with unreliable failure detectors. Results are presented on the limits of the number of monitoring messages, the latency for event detection, as well as the completeness and accuracy.*

Resumo. *O monitoramento de sistemas computacionais para a identificação de falhas é essencial para a construção de sistemas confiáveis. O diagnóstico em nível de sistema foi proposto inicialmente nos anos 1960 como uma abordagem baseada em testes para o monitoramento e identificação de componentes falhos. Ao longo dos últimas décadas, diversos modelos e estratégias para diagnóstico foram propostos, baseados em diferentes modelos de falha, aplicados nos mais diversos tipos de sistemas computacionais. Nos anos 1990, os detectores de falhas não confiáveis surgiram como uma abstração para, a partir do monitoramento de falhas de processos, permitir a execução do consenso em sistemas assíncronos sujeitos a falhas crash. A partir do modelo original, os detectores de falhas se transformaram no padrão* de facto *para monitoramento de sistemas distribuídos. O presente trabalho visa fechar uma lacuna conceitual, apresentando um modelo de diagnóstico distribuído consistente com os detectores de falhas não confiáveis. São apresentados resultados sobre os limites do número de mensagens de monitoramento, a latência para detecção de eventos, bem como sua completude e precisão.*

1. Introdução

Os sistemas computacionais passaram a fazer parte integral das mais diversas atividades humanas. Na verdade, diversas organizações humanas de grande relevância na sociedade moderna *são* sistemas computacionais (e.g. escolas virtuais ou redes sociais). Outras organizações, ainda que tenham presença física, são fortemente dependentes dos seus sistemas computacionais (e.g. bancos ou supermercados). Neste contexto, a falha de um sistema computacional pode trazer consequências graves, que variam desde a profunda insatisfação de consumidores, até prejuízos de bilhões de dólares decorrente dos danos sofridos [NYT 2021, Codestone 2017].

Desta forma, é essencial construir sistemas computacionais que continuem funcionando mesmo na presença de falhas de seus componentes: sistemas tolerantes a falhas. A tolerância a falhas surge logo em seguida ao desenvolvimento dos primeiros computadores digitais. O próprio Von Neumann investigou, ainda nos anos 1950, a construção de sistemas confiáveis a partir de componentes não confiáveis [Von Neumann 1956]. Hoje a tolerância a falhas é uma área bem estruturada, e o conjunto de propriedades que refletem o grau de confiança que se pode depositar em um sistema é denominado *dependability* [Avizienis et al. 2004].

Existem diversas técnicas para a construção de sistemas tolerantes a falhas [Beyer et al. 2016, Pradhan 1996]. A maioria destas técnicas explora a redundância, tanto explícita como implícita. Na redundância explícita são acrescentadas ao sistema réplicas de componentes com o objetivo de eliminar o ponto único de falha. Por exemplo: ao invés do sistema ter uma única unidade de memória secundária, tem múltiplas, de forma que a falha de uma única unidade não impede seu funcionamento correto. A redundância pode também ser implícita, já estando presente no sistema, como no roteamento tolerante a falhas [Duarte Jr et al. 2004]. No caso dos sistemas distribuídos, a redundância é implícita e intrínseca, pois um sistema distribuído consiste de um conjunto de n processos ($n \geq 2$) que comunicam, colaborando para a realização de alguma tarefa [Greve 2005].

Entre as propriedades de tolerância a falhas, uma das mais importantes é a disponibilidade, que reflete a porcentagem de tempo que se espera que um sistema esteja disponível para uso, mesmo sofrendo falhas e se recuperando. Para aumentar a disponibilidade de um sistema, é essencial ter um mecanismo eficaz de recuperação de falhas, de forma que o tempo que o sistema permanece fora do estado correto seja o menor possível. Apesar de que há sistemas que não necessitam identificar unidades falhas, por exemplo tomando decisões a partir da manifestação de um quorum [Rodrigues et al. 2016], diversos sistemas tolerantes a falhas seguem o modelo clássico de identificação da falha, isolamento da falha e reconfiguração do sistema [Pradhan 1996].

A identificação de falhas é problema também antigo: o primeiro modelo de sistemas para *diagnóstico* de falhas foi proposto ainda nos anos 1960: o modelo PMC, nome vindo das iniciais dos seus autores [Preparata et al. 1967]. O modelo PMC considera que as unidades do sistema realizam testes umas nas outras. Com base no conjunto de resultados de testes, é possível determinar quais unidades estão falhas. O modelo PMC assume que as unidades corretas são capazes de realizar testes perfeitos, i.e. determinam e reportam corretamente o estado de cada unidade que testam. A partir do modelo PMC, um número extraordinário de resultados foi produzido na área, considerando diferentes modelos e premissas, permitindo compreender os limites

do diagnóstico e definindo as mais diversas estratégias para a identificação de falhas [Masson et al. 1996, Duarte Jr et al. 2011].

No contexto de sistemas distribuídos, em 1985 é publicada a impossibilidade FLP, nome também vindo das iniciais de seus autores [Fischer et al. 1985]. Trata-se da impossibilidade de garantir a correta execução do consenso em sistemas distribuídos assíncronos em que podem ocorrer falhas por parada (*crash*). Nos sistemas assíncronos não há garantias temporais: nem para a execução de tarefas por um processo, nem para a transmissão de mensagens entre processos. Na raiz da impossibilidade FLP está justamente a dificuldade de distinguir um processo falho de um processo lento. Tendo em vista a importância do consenso – por muitos considerado o problema central de sistemas distribuídos – este resultado teve um impacto muito significativo na área como um todo.

Nos anos 1990, Chandra e Toueg [Chandra and Toueg 1996] investigaram o consenso sob um novo ponto de vista: como a impossibilidade FLP é afetada, caso os processos tenham informações sobre as falhas que ocorreram no sistema? Desta forma, definiram os detectores de falhas como oráculos, que informam o estado dos processos do sistema. O estado de um processo é reportado como ou correto ou suspeito de ter falhado. Os detectores de falhas são ditos não confiáveis, isto é, o detector pode informar um estado que não corresponde à realidade. Foram definidas duas propriedades para os detectores: completude e precisão. Informalmente, a completude reflete a capacidade do detector de identificar processos que efetivamente falharam. A precisão, por outro lado, reflete a capacidade do detector de não suspeitar incorretamente que processos corretos estão falhos.

A estratégia utilizada pela maioria dos detectores de falhas para monitorar processos consiste do envio periódico de mensagens de *heartbeat* que são recebidas por todos os processos do sistema [Bertier et al. 2002, Turchetti et al. 2016]. Se o sistema executa em um único segmento de rede, é possível realizar implementações eficientes desta estratégia, por exemplo utilizando multicast em hardware. Em outros ambientes a estratégia não escala, necessitando da transmissão periódica de n^2 mensagens. Os esforços para o desenvolvimento de detectores escaláveis em geral envolvem a difusão probabilística [Gupta et al. 2001]. Por outro lado, os algoritmos de diagnóstico distribuído têm sido propostos justamente com o objetivo de reduzir o número de mensagens necessárias para o monitoramento do sistema, bem como a latência para a detecção de novas mudanças de estado de processos. Este trabalho traz um esforço no sentido de unificar o diagnóstico e a detecção de falhas. É proposto um novo modelo de diagnóstico que permite a especificação de detectores de falhas escaláveis. O modelo é investigado para a futura implementação de algoritmos de monitoramento de sistemas distribuídos, sendo apresentados resultados sobre os limites do número de mensagens de monitoramento, a latência para detecção de eventos, bem como sua completude e precisão.

O restante deste trabalho está organizado da seguinte maneira. A próxima seção traz uma visão geral do diagnóstico em nível de sistema. Em seguida, a Seção 3 define e traz uma visão geral dos detectores de falhas. Na Seção 4 apresenta o modelo para diagnóstico distribuído como um detector de falhas, apresentando resultados para o número de testes e latência da detecção de falhas. A Seção 5 mostra resultados para a completude e precisão do diagnóstico/detecção de falhas. Por fim, a Seção 6 conclui o trabalho.

2. Diagnóstico em Nível de Sistema: Uma Visão Geral

Em 1967, Preparata, Metze and Chien publicaram o primeiro modelo de sistemas "diagnosticáveis" [Preparata et al. 1967], chamado modelo PMC a partir das iniciais dos nomes dos autores. As abordagens anteriores para identificação de falhas em sistemas computacionais tinham sido em nível de componentes individuais. No modelo PMC, um sistema é composto de unidades que são capazes de se testarem umas às outras. Um teste consiste de um procedimento completo o suficiente para permitir a classificação da unidade testada em *falha* ou *sem-falha*. O resultado de todos os testes executados é denominado síndrome, e é coletado por uma unidade central, externa ao sistema. A síndrome é processada para classificar as unidades como falhas ou sem-falhas.

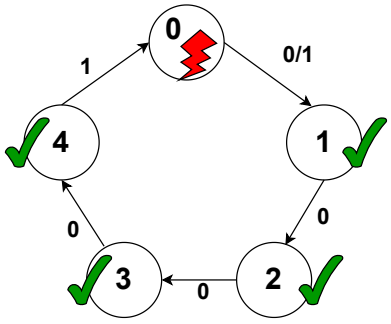

Figura 1. Exemplo clássico do modelo PMC.

Curiosamente, o modelo de falhas adotado pelo modelo PMC pode ser entendido hoje como o das falhas bizantinas. Apesar de que não estavam previstas ameaças ou componentes maliciosos, a saída produzida por uma unidade falha é arbitrária. Em particular, unidades falhas testam e reportam resultados dos seus testes, que podem ter qualquer valor. Por outro lado, o modelo PMC assume que uma unidade sem-falha é capaz de executar testes e reportar os resultados dos testes executados com precisão. Desta forma, dependendo dos testes executados, a síndrome permite ou não a identificação correta das unidades falhas e sem-falhas. Veja por exemplo a Figura 1. Arcos representam testes com resultados indicados com os rótulos 1 para falhou e 0 sem-falha. Neste sistema com $n = 5$ unidades, se uma única unidade estiver falha é possível determinar corretamente os estados de todas as unidades a partir da síndrome. Por outro lado, com duas unidades falhas o problema já se torna impossível: não há como identificar quem está falho. Para refletir a capacidade de um sistema de realizar o diagnóstico de f falhas foi definida a *diagnosability*; um sistema f-diagnosticável é capaz de identificar corretamente até f unidades falhas. O sistema da Figura 1 é 1-diagnosticável.

O conjunto de testes executados no sistema é denominado originalmente *connection assignment* e, mais tarde passou a ser chamado de *testing assignment*, que em português tem sido traduzido como *assinalamento de testes*. Nos primeiros anos do diagnóstico, boa parte da pesquisa na área se concentrou em determinar assinalamentos de testes com boa *diagnosability*, que permitissem o diagnóstico de forma eficaz e eficiente. Em 1974, Hakimi e Amim provaram que para um sistema de n unidades ser t-diagnosticável é necessário que (i) $n \geq 2t + 1$ e que (ii) cada processo seja testado por pelo menos t outros processos [Hakimi and Amin 1974].

Dez anos depois, em 1984, dois avanços tiveram grande impacto na área de di-

agnóstico em nível de sistema. No primeiro destes, Hakimi e Nakajima propõem que o assinalamento de testes seja adaptativo, isto é, os próximos testes a serem executados são definidos de acordo com os resultados dos testes anteriores [Hakimi and Nakajima 1984]. Esta possibilidade de adaptar os testes executados dá uma dimensão temporal ao diagnóstico até então inédita: primeiro um conjunto de testes são executados, para então seus resultados serem avaliados e serem definidos novos testes para execução em uma nova rodada. No segundo resultado de 1984, Hosseini, Kuhl and Reddy propuseram finalmente a eliminação de qualquer componente centralizado, criando o diagnóstico distribuído [Hosseini et al. 1984]. No diagnóstico distribuído, as unidades não apenas executam os testes, mas também coletam os resultados umas das outras para obter a síndrome. Cada unidade sem-falha pode então processar a síndrome e determinar o conjunto de unidades falhas do sistema.

Foi apenas na década seguinte, em 1992, que Bianchini e Buskens propuseram a união do diagnóstico adaptativo e distribuído [Bianchini Jr and Buskens 1992]. Foi proposto o algoritmo Adaptive-DSD (*Adaptive Distributed System-Level Diagnosis*) que foi implementado e utilizado no monitoramento de uma rede de grande porte em produção. O algoritmo é baseado em uma topologia de anel que não se rompe, apresentando o número ótimo de testes: no máximo n, mas com latência de até n rodadas sequenciais consecutivas de testes. Em 1998, Duarte e Nanya propõem o algoritmo Hi-ADSD (Hierarchical Adaptive Distributed System-Level Diagnosis) que mantém um modelo de diagnóstico adaptativo e distribuído, mas utiliza uma topologia virtual hierárquica para organizar as unidades do sistema que leva a uma latência máxima de $log^2 n$ rodadas de testes, todos os logaritmos neste trabalho são base 2.

O algoritmo Hi-ADSD foi proposto visando a construção de sistemas de gerência de redes eficientes e tolerantes a falhas [Duarte Jr et al. 1998, Duarte Jr et al. 1994], tendo sido implementado utilizando o protocolo de gerência da Internet SNMP (*Simple Network Management Protocol*) [De Bona and Duarte 2004]. O algoritmo Hi-ADSD anos mais tarde foi levemente modificado para garantir que o número de testes executados não passe de $nlogn$ a cada n rodadas de testes [Ruoso 2013]. Na nova versão o algoritmo passou a ser denominado VCube [Duarte et al. 2014] que já foi referenciado como um detector de falhas [Jeanneau et al. 2017].

Os resultados de diagnóstico apresentados acima consideram o modelo PMC e uma topologia representável por grafo completo. Diversos trabalhos já foram realizados no sentido de investigar o diagnóstico em redes de topologia arbitrária [Duarte Jr 1998, Siqueira et al. 2000], as quais são particionáveis por definição. Desta forma o problema de encontrar unidades falhas se transforma no problema de realizar o diagnóstico de alcançabilidade na rede [Duarte et al. 2011]. Outros trabalhos consideram modelos diferentes de diagnóstico. Em [Camargo and Duarte 2018] é considerada a possibilidade de um teste não trazer o resultado correto. O diagnóstico baseado em comparações [Ziwich and Duarte 2016] assume um modelo de falhas distinto, em que resultados de tarefas são comparados para realizar o diagnóstico, por exemplo, de integridade de dados [Ziwich et al. 2005]. Por fim, o diagnóstico probabilístico também permite uma visão alternativa, que leva em conta a dificuldade de realizar a tarefa de forma determinística [Masson et al. 1996].

3. Detectores de Falhas: Uma Visão Geral

A saída produzida por detector de falhas não confiável é exatamente a mesma de um algoritmo distribuído de diagnóstico: a lista de processos considerados falhos. Apesar disso, os detectores foram produzidos em um contexto completamente diferente. A motivação original para o seu desenvolvimento foi a impossibilidade do consenso em sistemas assíncronos sujeitos a falhas *crash*. Chandra e Toueg investigaram os limites da impossibilidade FLP considerando que os processos que executam o consenso tem, individual e localmente, acesso a um detector de falhas, do qual podem obter informações sobre o estado dos demais processos [Chandra and Toueg 1996].

Desta forma, mais que investigar estratégias eficientes para o monitoramento de processos, o objetivo era determinar quais propriedades do detector podem colaborar para a resolução do consenso. Assim, os autores definiram duas propriedades essenciais, que chamaram de *completeness* e *accuracy*, neste trabalho traduzidas como completude e precisão, respectivamente. Informalmente, a completude diz respeito à capacidade do detector de falhas de efetivamente suspeitar de processos falhos. Tendo em vista que o modelo de falhas considerado é o *crash*, um processo falho não produz qualquer resposta a nenhum estímulo, portanto não é difícil garantir que, decorrido um tempo após uma falha, todos os processos corretos levantem suspeitas do que ocorreu.

A precisão, por outro lado, não pode ser jamais garantida em sistemas assíncronos. O problema é clássico: um processo pode estar correto, mas lento. Lento tanto para executar tarefas, como lento na comunicação de mensagens. Assim, é difícil (impossível, na verdade) distinguir um processo falho de um processo lento. Esta é exatamente a raiz da impossibilidade FLP. Entretanto, se os processos corretos têm uma certa capacidade de não suspeitar de processos que não falharam, qual o impacto disso no consenso? Após reclassificar as propriedades dos detectores em forte e fraca para obterem um total de oito classes de detectores de falhas (descritas no próximo parágrafo), em [Chandra et al. 1996] os autores mostram que a classe mais fraca de todas é suficiente para resolver o consenso em sistemas distribuídos sujeitos a falhas *crash*.

Para construir as oito classes de detectores [Chandra and Toueg 1996], a propriedade da completude do detector de falhas pode ser forte (após um tempo, todo processo falho é suspeitado por todos os processos corretos) e fraca (após um tempo, todo processo falho é suspeitado por pelo menos 1 processo correto). Observe que entre a falha e a suspeita há um intervalo de tempo, até que a estratégia de monitoramento determine que justifica a suspeita. Além disso, não é difícil converter a completude fraca em completude forte: basta que o processo correto que detectou a falha faça uma difusão da informação para os demais processos corretos.

A propriedade da precisão também é reclassificada em forte e fraca. A precisão forte corresponde a que não haja nenhuma suspeita de falhas de processos que estão corretos. Na precisão fraca, pelo menos 1 processo correto jamais levanta suspeita de ter falhado. Estas duas propriedades são ainda estendidas em outras classes ditas "*eventual*" (em inglês, não português): depois de um tempo nenhum/um (respectivamente) processo correto levanta suspeita de ter falhado. A classe de detectores com completude fraca e precisão fraca após um tempo – dita $\diamond W$ – já garante que o consenso seja possível. Observe que um sistema assíncrono com detector de falhas já não é mais "puramente" assíncrono, precisando sim atender a propriedades temporais para ser útil.

As implementações clássicas de detectores de falhas são baseadas no envio de mensagens periódicas por cada processo para todos os demais informando que estão "vivos", daí sendo chamadas de "*heartbeats*" [Bertier et al. 2002, Turchetti et al. 2016]. Esta estratégia é eficiente para implementação em uma única rede física baseada em difusão, pois basta uma única mensagem para enviar cada *heartbeat*. Por outro lado, em sistemas que não são baseados em difusão, incluindo aqueles executando em redes distintas da Internet, a estratégia não escala, necessitando da transmissão periódica de n^2 mensagens. Esta é uma das motivações para a definição do modelo de diagnóstico para a construção de detectores baseados em um assinalamento de testes, apresentado na próxima seção.

4. Um Novo Modelo de Diagnóstico para Detecção de Falhas em Sistemas Assíncronos

Nesta seção é apresentado um novo modelo de diagnóstico distribuído em nível de sistema para a implementação de detectores de falhas não confiáveis.

O modelo assume um sistema distribuído Π que consiste de n processos que se comunicam utilizando troca de mensagens. Os processos têm identificadores sequenciais, assim $\Pi = \{p_0, p_1, ..., p_{n-1}\}$. Alternativamente, os identificadores são utilizados diretamente para fazer referência aos processos, $\Pi = \{0, 1, ..., n-1\}$. O sistema é totalmente conectado, representável por um grafo não direcionado completo $K_n = (\Pi, E)$, sendo E o conjunto de enlaces tal que $E = \{\{i, j\} \,|\, 0 \leq i, j < n \text{ e } i \neq j\}$ assim qualquer processo pode se comunicar diretamente com qualquer outro processo, sem a necessidade de intermediários.

O sistema é assíncrono, não havendo limites conhecidos para o tempo máximo de transmissão de uma mensagem entre dois processos quaisquer, nem para a execução de uma tarefa por um processo qualquer. O modelo de falhas é por parada (*crash*), um processo que falha perde seu estado completamente e não produz qualquer saída para nenhum estímulo. Assim, um processo pode estar em um de dois estados: *falho* ou *correto*. Um processo no estado *correto* pode também ser dito *sem-falha*. Formalmente, a função *estado(i)* retorna o verdadeiro estado de um processo i: *estado*$(i) = \{$*falho* $\mid$ *correto*$\}$. Um *evento* é definido como a transição de estado de um processo, de *correto* para *falho*. Apesar da facilidade de incluir a recuperação de processos no modelo, além da ocorrência de múltiplos eventos em um mesmo processo ao longo do tempo, neste artigo, por motivo de espaço, são apenas consideradas falhas de processos sem recuperação.

Os processos executam testes uns nos outros. O objetivo de um teste é permitir que o testador classifique o estado do processo testado como *correto* ou *suspeito* de ter falhado. Um teste corresponde ao envio de estímulos do processo testador ao processo testado, para os quais são esperadas respostas. Ao receber a resposta esperada, o testador considera que o processo testado está correto. Considerando dois processos $i, j \in \Pi$, um teste executado pelo testador j no processo testado i é formalmente definido como a função *teste*$_j(i)$ = $\{$*correto* $\mid$ *suspeito*$\}$. Um mecanismo de *timeout* adequado deve ser definido para limitar o tempo máximo de espera de uma resposta [Turchetti et al. 2016]. O procedimento de testes deve ser completo o suficiente, em particular a decisão de classificar o processo testado como *suspeito* deve ser tomada com mais de um único *timeout*. O Corolário 4.1 faz a relação entre os estados dos processos testados e sua classificação pelo detector de falhas.

Corolário 4.1 *Um processo testado como correto está realmente neste estado, assim* $\forall i, j \in V$*, se teste*$_j(i)$ *= correto, então estado(i)= correto. Por outro lado, um processo classificado como suspeito pode estar tanto no estado falho como sem-falha quando foi testado, caso em que o resultado do teste é causado por lentidão, i.e. um timeout prematuro. Desta forma: se teste*$_j(i)$ = $\{$*suspeito*$\}$*, então estado(i)=*$\{$*correto* $\mid$ *falho*$\}$.

O conjunto de todos os testes executados no sistema é denominado de *assinalamento de testes* e é representado por um grafo direcionado $A = (\Pi, T)$. O conjunto de arestas direcionadas T representa os testes e contém arcos (i, j) indicando que o processo i é testador do processo j (testado). Os testes são executados periodicamente, em intervalos de testes definidos usando o relógio local do testador. Os processos não têm relógios sincronizados, e seus intervalos de testes podem diferir aleatoriamente entre si; a única premissa é que os relógios locais crescem assintoticamente. Uma *rodada de testes* ocorre após todos os processos terem executado seus testes assinalados, ou seja todos os testes em T são efetivamente realizados. As rodadas de testes podem ser enumeradas, $r_1, r_2, ...$, a primeira rodada corresponde à primeira vez que os testes são executados, e assim por diante.

O objetivo do diagnóstico é que todos os processos corretos tenham, após um número finito de rodadas de testes, uma classificação dos estados de todos os processos do sistema. Assim, $\forall i, j \in Pi$, o processo j classifica o processo i é formalmente definido como a função *estado*$_j(i)$ = $\{$*correto* $\mid$ *suspeito*$\}$. Diferentes processos podem ter classificações diferentes para cada processo do sistema, que depende do momento em que houve o teste que levou a uma modificação do estado em que um processo é classificado, bem como da propagação desta informação entre os processos corretos.

A latência (L) da detecção de falhas corresponde ao número de rodadas necessárias para que uma nova classificação *estado*$_j(i)$ do estado processo i pelo processo j ser recebida por todos os processos corretos do sistema. Assim, em $r_1, r_2, ..., r_L$ rodadas, $\forall k \mid$ *estado*(k) = *correto*, *estado*$_k(i)$ = *estado*$_j(i)$. A latência é proporcional ao diâmetro do grafo que representa o assinalamento de testes, A.

Assumimos que se o processo sofre um evento de falha, então em uma rodada de testes todos os seus testadores são capazes de classificá-lo como *suspeito*. Esta premissa é trivial de ser garantida, já que o modelo de falhas é *crash* e, portanto, um processo falho não produz qualquer resposta para qualquer teste realizado sobre ele. O Corolário 4.2 explicita que um teste executado sobre um processo falho resulta em uma suspeita.

Corolário 4.2 *Considere cada processo* i *tal que* $estado(i)$ *= falho. Então para todo processo correto* j *que testa* i*, estado*$_j(i)$ *= suspeito.*

O Corolário 4.3 a seguir determina que todo processo deve ser testado por algum processo correto periodicamente. Observe que se o processo não é testado periodicamente, é impossível detectar um evento que ocorre neste processo. Como forma de garantir que todos os processos são adequadamente monitorados, no presente modelo todo processo é testado por pelo menos um processo correto em cada rodada de testes. Assim, em cada rodada de testes r, $\forall i, \exists j$ tal que *estado(j)=correto* e $(j, i) \in A$.

Corolário 4.3 *Todo processo* i *é testado periodicamente por pelo menos um processo correto, caso contrário não é possível detectar evento ocorrido em* i*.*

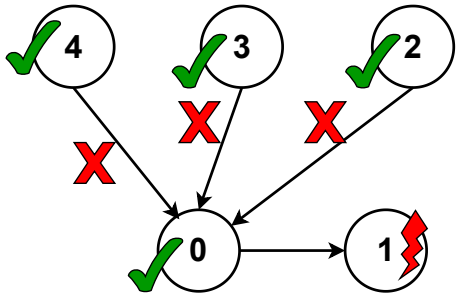

Figura 2. Neste exemplo o processo 1 está falho e é testado suspeito pelo processo correto 0. Entretanto, os processos corretos 2, 3 e 4 todos suspeitam de que o processo 1 está falho.

Em uma determinada rodada, um processo que testa outro processo correto pode obter informações de diagnóstico do processo testado. Em outras palavras: processos mutuamente classificados como corretos podem trocar suas informações de diagnóstico. Estas informações de diagnóstico mantidas por um processo são obtidas a partir dos resultados dos seus próprios testes executados, ou são informações obtidas de outros processos testados corretos. Entretanto: se o processo correto i testou o processo j na última rodada, então não deve obter informações sobre a classificação do estado de j por outros processos.

Além disso, de acordo com o Corolário 4.4, é necessário garantir que a cada rodada um processo obtenha novas informações de diagnóstico sobre todos os processos do sistema, como especificado do Corolário 4.4. Assim, um processo correto obtém informações sobre outros processos executando testes e obtendo informações de diagnóstico a partir dos processos testados corretos. O assinalamento de testes A pode ser construído de forma a garantir que todos os processos sempre obtenham as informações de diagnóstico de forma eficiente a cada rodada de testes.

Corolário 4.4 *A cada nova rodada de testes um processo obtém novas informações de diagnóstico sobre todos os demais processos do sistema, ou testando aqueles processos ou obtendo informações de diagnóstico a partir de processos testados corretos.*

Executando o diagnóstico em um sistema assíncrono é possível chegar a uma situação extrema em que todos os processos corretos suspeitam de todos os demais processos corretos. Neste caso, todos os processos vão executar testes em todos os demais processos. O Teorema 4.1 explicita este fato.

Teorema 4.1 *Considerando que todos os processos corretos executam seus testes assinalados uma única vez por rodada de testes, um algoritmo de diagnóstico especificado de acordo com o modelo de proposto gera, no pior caso,* $n^2 + n$ *testes por rodada.*

Prova: Se em uma rodada de testes todos os n processos estão corretos e suspeitam de todos os demais $n - 1$ processos serão executados $n^2 + n$ testes por rodada.

Na verdade, não há qualquer premissa sobre a velocidade de execução dos processos do sistema. Assim, um processo pode ser muito mais rápido que outro e executar seus testes assinalados múltiplas vezes, enquanto outro processo mais lento executa uma única rodada. Desta forma, se não há a garantia que os processos corretos executam seus testes

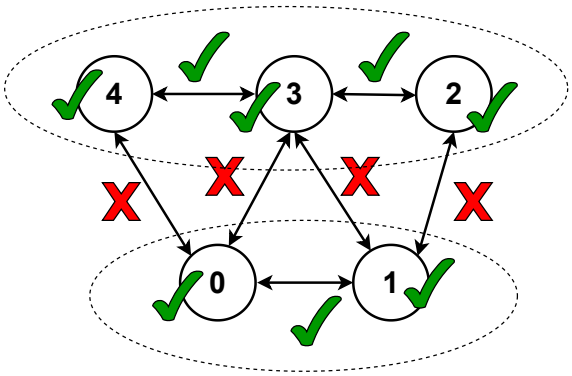

Figura 3. A precisão só pode ser garantida se os processos corretos estiverem fortemente conectados entre si em A.

assinalados uma única vez por rodada, o número de testes executados pode ser maior que $n^2 + n$.

Para concluir, o modelo proposto é $n - 1$-diagnosticável, permitindo que se todos menos um processo falha este ainda é capaz de monitorar o sistema e reportar os estados dos demais processos.

5. Completude & Precisão: Propriedades do Modelo DRCT

As duas propriedades dos detectores de falhas – completude e precisão – são definidas para o modelo de diagnóstico como segue. Informalmente, a completude se refere ao fato dos processos que efetivamente falharam serem considerados suspeitos pelos processos corretos. A precisão é informalmente definida como a capacidade dos processos corretos serem efetivamente capazes de se detectarem como tal, ou seja não cometerem enganos, suspeitando de processos corretos. A seguir, as duas propriedades são formalmente definidas e classificadas como fraca e forte, seguindo a proposta original [Chandra and Toueg 1996].

O diagnóstico tem completude forte se todos os processos corretos classificam processos que falharam como suspeitos em um número finito de rodadas após a ocorrência do evento de falha. Assim, para $i \in V$ se o estado de i muda de *correto* para *falho*, então após um número finito de rodadas $\forall j \in V$ tal que j está correto, j classifica i como *suspeito*. O diagnóstico tem completude fraca se pelo menos um processo correto suspeita de cada processo que efetivamente falhou. Desta forma para $i \in V$, se i sofre uma *falha*, então após um número finito de rodadas $\exists j \in V \mid$ *estado(j)=correto*, $estado_j(i) =$ *suspeito*.

No modelo de diagnóstico proposto não é possível converter a completude fraca em completude forte como é feito em [Chandra and Toueg 1996]. O motivo é que o testador correto j que suspeitou de i pode *ser suspeitado* (incorreta e indefinidamente) por todos os demais processos corretos do sistema. A Figura 2 ilustra este fato: o processo 0 correto testa o processo 1 falho e levanta corretamente a suspeita. Entretanto, todos os demais processos suspeitam de 0, assim não é possível seguir a estratégia do único processo correto comunicar sua suspeita aos demais processos, como é feito em [Chandra and Toueg 1996].

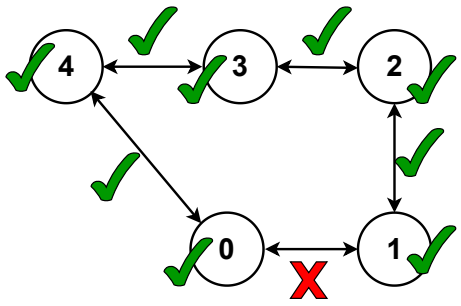

Figura 4. Neste exemplo apesar de que os processos estão fortemente conectados entre si no assinalamento de testes A, os processos 0 e 1 se testam e se suspeitam incorretamente, portanto a precisão forte não está garantida.

No modelo proposto, para garantir a completude forte é necessário que um processo correto, digamos k, ou execute um teste em i, ou obtenha a informação da suspeita a partir de outro processo correto. Veja que tendo em vista o Corolário 4.4, todo algoritmo de diagnóstico especificado de acordo com o modelo proposto garante que isso vai acontecer. Assim, todos os processos corretos vão suspeitar do processo falho em 1 rodada. O Teorema 5.1 prova que qualquer algoritmo construído com o modelo proposto tem completude forte.

Teorema 5.1 *Um algoritmo de diagnóstico especificado de acordo com o modelo proposto tem completude forte.*

Prova: A prova é feita por indução no diâmetro do assinalamento de testes A. Considere o processo falho tal que *estado(i)* = *falho*. Base da indução: se o diâmetro é 1, então todos os processos testam todos os demais. De acordo com o Corolário 4.2, $\forall j \in V, j \neq i$ e *estado(j)* = *correto*: $estado_j(i) = suspeito$. Hipótese da indução: se o diâmetro do assinalamento de testes $A = d$ a completude é forte, ou seja todos os processos corretos com distância até d com relação ao processo i falho suspeitam de i. Passo da indução: incrementamos o diâmetro para $d+1$, incluindo novos processos, que a cada rodada testam aqueles processos que estavam a uma distância d de i, estes processos obtém informação de diagnóstico sobre i a partir dos processos testados.

Para garantir a precisão forte no modelo de diagnóstico proposto é preciso mais, como mostra o Teorema 5.2 a seguir.

Teorema 5.2 *Para uma execução de um algoritmo de diagnóstico especificado de acordo com o modelo de proposto ter precisão forte, os processos corretos não devem suspeitar de outros processos corretos* ***e*** *todos os processos corretos devem formar um componente fortemente conectado no assinalamento de testes A.*

A **prova** pode ser feita de forma similar à do Teorema 5.1: se nenhum processo correto suspeita de outro processo correto e eles estão fortemente conectados entre si em A, então após no máximo d rodadas de testes, todos os processos corretos terão informações sobre todos os demais. Isto é, não existe $k \in V$ tal que $estado(k) = correto$ e $estado_k(i) = suspeito$ indefinidamente.

É fácil ver que os processos corretos precisam estar fortemente conectados em A para garantir a precisão. Na Figura 3 os processos 0 e 1 corretos se testam e não suspeitam um do outro, mas todos os demais processos corretos suspeitam de ambos.

Entretanto, como especificado no teorema, os processos corretos estarem fortemente conectados entre si não é suficiente para garantir a precisão forte. Não podem haver suspeitas incorretas, como há na Figura 4, em que há testes contraditórios sobre os processos 0 e 1. Seja i um processo correto. Sejam j e k dois processos corretos que testam i. Considere que j determina corretamente o estado de i, portanto j não suspeita de i. Por outro lado, k suspeita incorretamente de i. Alguns processos corretos podem obter informação sobre o estado de i a partir de j e outros a partir de k, assim o processo correto i vai ser indevidamente suspeitado, quebrando a precisão do diagnóstico.

6. Considerações Finais

Tanto o diagnóstico distribuído como os detectores de falhas representam abstrações para o monitoramento de processos de sistemas distribuídos, produzindo como saída a lista dos processos falhos. O presente trabalho tem por objetivo iniciar o estabelecimento de um modelo que abarca as duas abstrações. Foi especificado um modelo de diagnóstico que é totalmente compatível com os detectores de falhas. Além da especificação do modelo, foram investigadas sua latência, número de testes, completude e precisão. Entre outros resultados obtidos, mostramos que em um detector de falhas baseado em testes (*pull-based*) a completude fraca só pode se transformar em forte se todos os processos corretos estiverem fortemente conectados no assinalamento de testes A.

Trabalhos futuros incluem a especificação de algoritmos eficientes para detecção de falhas em sistemas assíncronos, dentro do novo modelo proposto. Tradicionalmente os detectores de falhas são implementados utilizando mecanismos de monitoramento baseados em mensagens que são comunicadas de cada processo para *todos* os demais. Acreditamos que o novo modelo facilita o desenvolvimento de estratégias que empregam um número menor de mensagens em média. A utilização prática destes algoritmos de detecção de falhas na Internet pode ser realizada por exemplo através da sua implementação dentro de um sistema de gerência de redes [Moraes and Duarte Jr 2011], utilizando tecnologias de virtualização [Turchetti and Duarte 2015, Turchetti and Duarte Jr 2017].

Referências

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Bertier, M., Marin, O., and Sens, P. (2002). Implementation and performance evaluation of an adaptable failure detector. In *The 32nd International Conference on Dependable Systems and Networks (DSN)*, pages 354–363. IEEE.

Beyer, B., Jones, C., Petoff, J., and Murphy, N. R. (2016). *Site reliability engineering: How Google runs production systems*. O'Reilly.

Bianchini Jr, R. P. and Buskens, R. W. (1992). Implementation of online distributed system-level diagnosis theory. *IEEE Transactions on Computers*, 41(05):616–626.

Camargo, E. T. d. and Duarte, E. P. (2018). Running resilient mpi applications on a dynamic group of recommended processes. *Journal Braz. Comp. Soc.*, 24(1):1–16.

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

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

Codestone (2017). The True Impact of IT Failures. `https://www.codestone.net/our-thoughts/true-impact-of-it-failures`.

De Bona, L. C. E. and Duarte, E. P. (2004). A flexible approach for defining distributed dependable tests in snmp-based network management systems. *Journal of electronic testing*, 20(4):447–454.

Duarte, E. P., Bona, L. C., and Ruoso, V. K. (2014). Vcube: A provably scalable distributed diagnosis algorithm. In *2014 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, pages 17–22. IEEE.

Duarte, E. P., Weber, A., and Fonseca, K. (2011). Distributed diagnosis of dynamic events in partitionable arbitrary topology networks. *IEEE transactions on parallel and distributed systems*, 23(8):1415–1426.

Duarte Jr, E. (1998). Um algoritmo para diagnóstico de redes de topologia arbitrária. In *Proc. 1st SBC Workshop on Test and Fault Tolerance, SBCWTF*, volume 1, pages 50–55.

Duarte Jr, E. P., Mansfield, G., Nanya, T., and Noguchi, S. (1998). Improving the dependability of network management systems. *Int. Journal Net Mgmt*, 8(4):244–253.

Duarte Jr, E. P., Mansfield, G., Noguchi, S., and Miyazaki, M. (1994). Fault-tolerant network management. *The 2nd International Symposium on Applied Corporate Computing (ISACC)*, pages 109–116.

Duarte Jr, E. P., Santini, R., and Cohen, J. (2004). Delivering packets during the routing convergence latency interval through highly connected detours. In *The 34th International Conference on Dependable Systems and Networks (DSN)*, pages 495–504. IEEE.

Duarte Jr, E. P., Ziwich, R. P., and Albini, L. C. (2011). A survey of comparison-based system-level diagnosis. *ACM Computing Surveys (CSUR)*, 43(3):1–56.

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.

Greve, F. (2005). Protocolos fundamentais para o desenvolvimento de aplicações robustas. *Minicurso do SBRC'2005*, pages 330–398.

Gupta, I., Chandra, T. D., and Goldszmidt, G. S. (2001). On scalable and efficient distributed failure detectors. In *The 20th ACM PODC*, pages 170–179.

Hakimi, S. L. and Amin, A. T. (1974). Characterization of connection assignment of diagnosable systems. *IEEE Transactions on Computers*, 100(1):86–88.

Hakimi, S. L. and Nakajima, K. (1984). On adaptive system diagnosis. *IEEE Transactions on Computers*, 33(3):234–240.

Hosseini, S. H., Kuhl, J. G., and Reddy, S. M. (1984). A diagnosis algorithm for distributed computing systems with dynamic failure and repair. *IEEE Transactions on Computers*, 33(03):223–233.

Jeanneau, D., Rodrigues, L. A., Arantes, L., and Duarte Jr, E. P. (2017). An autonomic hierarchical reliable broadcast protocol for asynchronous distributed systems with failure detection. *Journal of the Brazilian Computer Society*, 23(1):1–14.

Masson, G. M., Blough, D. M., and Sullivan, G. F. (1996). System diagnosis. In *Fault-Tolerant Computer System Design*, pages 478–536. Prentice-Hall.

Moraes, D. M. and Duarte Jr, E. P. (2011). A failure detection service for internet-based multi-as distributed systems. In *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pages 260–267. IEEE.

NYT, N. Y. T. (2021). Gone in Minutes, Out for Hours: Outage Shakes Facebook. `https://www.nytimes.com/2021/10/04/technology/facebook-down.html`.

Pradhan, D. K. (1996). *Fault-Tolerant Computer System Design*. Prentice-Hall.

Preparata, F. P., Metze, G., and Chien, R. T. (1967). On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, 16(6):848–854.

Rodrigues, L. A., Arantes, L., and Duarte, E. P. (2016). An autonomic majority quorum system. In *The 30th International Conference on Advanced Information Networking and Applications (AINA)*, pages 524–531. IEEE.

Ruoso, V. K. (2013). Uma estratégia de testes logarítmica para o algoritmo hi-adsd. Dissertação de mestrado, UFPR.

Siqueira, J., Fabris, E., and Duarte Jr, E. (2000). A token based testing strategy for non-broadcast network diagnosis. In *1st IEEE Latin American Test Workshop*, pages 166–171.

Turchetti, R. C. and Duarte, E. P. (2015). Implementation of failure detector based on network function virtualization. In *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, pages 19–25. IEEE.

Turchetti, R. C., Duarte, E. P., Arantes, L., and Sens, P. (2016). A qos-configurable failure detection service for internet applications. *Journal of Internet Services and Applications*, 7(1):1–14.

Turchetti, R. C. and Duarte Jr, E. P. (2017). Nfv-fd: Implementation of a failure detector using network virtualization technology. *International Journal of Network Management*, 27(6):e1988.

Von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata Studies*, 34(34):43–98.

Ziwich, R. P., Duarte, E., and Albini, L. C. P. (2005). Distributed integrity checking for systems with replicated data. In *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, volume 1, pages 363–369. IEEE.

Ziwich, R. P. and Duarte, E. P. (2016). A nearly optimal comparison-based diagnosis algorithm for systems of arbitrary topology. *IEEE Transactions on Parallel and Distributed Systems*, 27(11):3131–3143.