Difusão em Grupo Tolerante a Faltas com Ordem Causal Usando Informação Localizada

Válter Santos and Luís Rodrigues

INESC-ID, Instituto Superior Técnico, U. Lisboa
{valter.c.santos,ler}@tecnico.ulisboa.pt

Resumo Neste artigo consideramos um sistema distribuído constituído por N nós que se organizam em vários grupos de comunicação $G_1, \ldots, G_n$, os quais se podem sobrepor. Os nós podem enviar mensagens para qualquer grupo, sendo que estas devem ser recebidas por todos os seus membros, numa ordem que respeita as relações de causa-efeito que derivam do envio e receção de mensagens. Estas propriedades devem ser garantidas mesmo que alguns dos nós do sistema estejam temporariamente incontactáveis. O desafio é atingir este objetivo sem que os nós tenham de manter informação sobre todos os outros nós. Apresentamos um algoritmo em que cada nó necessita de manter unicamente informação sobre um subconjunto com dimensão V de todos os nós, que designamos por vizinhos, em que $V << N$, e que tolera a indisponibilidade temporária de f nós em cada vizinhança.

1 Introdução

O problema da difusão fiável em grupo é um problema clássico dos sistemas distribuídos tolerantes a faltas [4,12,8]. Este problema levanta-se em sistemas de pequena dimensão, em que os grupos são usados para coordenar grupos de réplicas de servidores [4] (tipicamente, com menos de uma dezena de membros) e em sistemas de grande dimensão, como por exemplo sistemas de edição-subscrição, em que os grupos são usados para disseminar informação para centenas de subscritores [6,1]. Independentemente da dimensão do sistema, é desejável oferecer a garantia de entrega das mensagens, mesmo na presença de omissões na rede, ou perante a ocorrência de falhas ou da indisponibilidade temporária de nós. Pretende-se também oferecer garantias de ordenação na entrega da mensagem, geralmente assegurando a ordem FIFO [2,10] ou a ordem causal [4,15,5]. Para oferecer estas garantias de fiabilidade e ordenação, os nós necessitam de manter informação de controlo (por exemplo, números de sequência). O desafio que este artigo aborda é o de concretizar a garantia de entrega de mensagens assegurando a ordem causal, devido a ser o grau de consistência mais forte que se consegue assegurar sem que o sistema bloqueie quando ocorrem falhas [13], sem obrigar que cada nó tenha de manter informação de controlo com tamanho proporcional à dimensão do sistema.

A difusão em grupo tolerante a faltas, com garantias de ordem causal, para sistemas de pequena dimensão está hoje bem estudada, existindo vários sistemas, tanto académicos como comerciais, que oferecem este serviço [4,16,9].

Infelizmente, a informação de controlo que estes sistemas mantêm para oferecer as garantias de fiabilidade e ordenação tem um tamanho considerável. Por exemplo, em [4] os nós necessitam de manter um vetor para cada grupo que existe no sistema, em que cada vetor tem uma entrada para cada membro do grupo (para além disso, os nós precisam de manter os vetores de todos os grupos, mesmo os dos grupos aos quais não pertencem). Este tipo de solução, sendo bastante eficiente em sistemas de pequena dimensão (os vetores permitem capturar as relações de causalidade com grande precisão), não escalam para sistemas com um elevado número de nós.

A única forma de assegurar a escalabilidade de um sistema desta natureza é recorrer a algoritmos que permitam aos nós manter informação localizada, isto é, apenas informação sobre um pequeno número de nós, independentemente do tamanho do sistema como um todo. Isto é tipicamente conseguido organizando os nós do sistema numa rede sobreposta, em que cada nó mantém ligações diretas com um reduzido número de *vizinhos*. Estas relações de vizinhança, que limitam as interações possíveis no sistema, permitem o desenvolvimento de algoritmos em que cada nó só mantém informação de controlo sobre outros nós, na sua vizinhança direta ou indireta, com um horizonte reduzido.

A ideia de organizar os nós numa rede sobreposta para suportar a comunicação em grupo em sistemas de grande escala tem sido aplicada extensivamente, em particular na concretização de sistemas de edição-subscrição [17,2]. No entanto, a grande maioria destes sistemas não recorre a algoritmos completamente localizados. Por exemplo, cada nó no sistema Gryphon [2] necessita de manter estado por cada fonte de informação, de forma a assegurar a ordem FIFO. Para além disso, a grande maioria dos sistemas que usam esta estratégia asseguram a ordem FIFO, mas não garantem a ordem causal. Finalmente, existem trabalhos que exploram a topologia da rede para assegurar a ordem causal mas assumem que a topologia é estática, pelo que não toleram faltas ou obrigam todos os nós da rede a serem replicados. De facto, tanto quanto é do nosso conhecimento, não existem na literatura soluções que simultaneamente: i) usem algoritmos localizados; ii) sejam tolerantes a faltas; e iii) garantam a ordem causal na entrega das mensagens. Neste artigo apresentamos um algoritmo que combina estas características.

O nosso algoritmo, que designámos por LoCaMu (*Local Causal Multicast*), assume que os nós do sistema estão organizados num grafo acíclico e tolera faltas do tipo falha-recuperação, em que um nó pode ficar temporariamente indisponível mas que eventualmente recupera. A indisponibilidade temporária de um nó não impede que os restantes nós corretos continuem a entregar mensagens, tal como se a falta não tivesse ocorrido. O algoritmo requer que cada nó mantenha informação sobre todos os nós que estão contidos na sua vizinhança, a uma distância de $2f + 2$, em que f é o número máximo de nós que podem estar simultaneamente indisponíveis nessa vizinhança. Concretamente, o LoCaMu requer que cada nó mantenha um estado de dimensão V^2 em que V é número de nós que se encontram nesta vizinhança (este número depende da topologia do grafo; se o grafo for uma árvore binária $V = O(2^{2f+3} + 2^{2f+2})$ no pior caso).

2 Trabalho Relacionado

A literatura sobre as diferentes variantes de comunicação em grupo é bastante extensa e aborda cenários com características muito distintas, em termos da dimensão alvo do sistema, garantias de fiabilidade, garantias de ordenação de mensagens, requisitos dos algoritmos no que se refere à quantidade de informação de controlo que necessitam de manter e, em consequência dessas opções, também em termos dos débitos que conseguem suportar.

Num extremo, podemos encontrar sistemas desenhados para suportar o desenvolvimento de sistemas tolerantes a faltas baseados no modelo de máquina de estado replicada e suas variantes [4]. Estes sistemas tipicamente oferecem garantias fortes de fiabilidade, e diferentes políticas de ordenação de mensagens, incluindo a ordem total (que está fora do âmbito do nosso trabalho). No entanto, a maioria destes sistemas assume que os nós estão organizados numa clique, e que mantêm informação de controlo sobre todos os outros nós, executando mecanismos globais de coordenação. São portanto, inerentemente não escaláveis, mesmo quando usam técnicas avançadas para reduzir a informação de controlo trocada [15]. No outro extremo, temos sistemas desenhados para suportar a disseminação de informação para um número muito elevado de participantes, como a difusão IP ou os sistemas para suportar a difusão de fluxos multimédia, que recorrem frequentemente a técnicas de computação entre-pares [17] ou mesmo a protocolos de difusão epidémica [3], que têm grande capacidade de escala mas que oferecem poucas garantias de fiabilidade e de ordenação.

Neste trabalho estamos interessados em sistemas que se encontram entre estes dois extremos, isto é, sistemas com várias centenas de nós, e várias dezenas de grupos, onde manter e trocar informação de controlo de tamanho proporcional ao número total de nós no sistema é inviável, mas para os quais ainda é possível e desejável assegurar garantias de fiabilidade e ordenação. Uma aplicação de relevo e com estas características são os sistemas de edição-subscrição. Apesar de existirem várias estratégias para concretizar este paradigma de interação [6], uma das mais usadas na prática recorre a uma rede sobreposta de nós (tipicamente designados por corretores de eventos) que encaminha eventos num grafo de forma descentralizada. Caso se use um grafo acíclico, com uma topologia estática, e os nós do grafo nunca falharem, é fácil não só assegurar a fiabilidade da comunicação mas também oferecer garantias de ordenação de mensagens, em particular, é fácil mostrar que se os canais entre nós vizinhos forem fiáveis e respeitarem a ordenação FIFO, a comunicação entre qualquer par de nós é também fiável e respeita não só a ordem FIFO mas também a ordem causal [5]. Infelizmente, se os nós do grafo puderem falhar, a reconfiguração do grafo pode facilmente levar a perda de mensagens e também a violações da ordenação de mensagens. No próximo parágrafo descrevemos as principais técnicas que têm sido propostas para abordar este problema.

Uma solução possível consiste em replicar cada nó do grafo, recorrendo a um algoritmo como o Paxos [11]. Esta solução é utilizada em sistemas como o Saturn [5]. No entanto, isto só é exequível para sistemas com poucos nós, uma vez que triplica o custo da instalação, o que pode ser incomportável em

	Sem Replicação	Entrega sem Lacunas	Localidade	Paralelismo	Causalidade	Escalável
[4]	✓	✓	✗	✓	✓	✗
[15]	✓	✓	✗	✓	✓	✗
[2]	✗	✓	✗	✓	✗	✗
[10]	✓	✗	✓	✗	✗	✓
LoCaMu	✓	✓	✓	✓	✓	✓

Tabela 1. LoCaMu vs Trabalho Relacionado

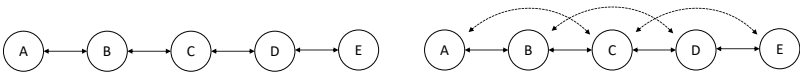
sistemas de grande dimensão. Para além disso, a execução do Paxos sempre que se encaminham eventos penaliza o desempenho do sistema. O Gryphon [2] também usa replicação dos nós mas reduz o custo de coordenação entre réplicas, não obrigando a executar um algoritmo de consenso a cada mensagem. Isto tem a vantagem de oferecer uma menor latência, mas obriga cada nó a ter de manter estado por cada emissor que existe no sistema para tolerar perda de mensagens.

Outra alternativa para atingir a tolerância a faltas consiste em aumentar o grafo com arestas adicionais que permitem que um nó envie dados diretamente para os vizinhos dos seus vizinhos (os quais estariam a dois saltos de distância sem estas arestas adicionais), de forma a assegurar o encaminhamento mesmo quando esse vizinho fica indisponível. Esta solução evita a replicação dos nós, mas obriga a manter mais informação de controlo para detetar e resolver a entrega de mensagens duplicadas e fora de ordem, que resultam da utilização de caminhos redundantes. O Delta-Neighbourhood [10] é um algoritmo que usa informação localizada para este efeito, embora só garanta ordem FIFO. Este último algoritmo tem também a grande desvantagem de só permitir ter uma única mensagem em trânsito, para cada emissor, em cada instante de tempo.

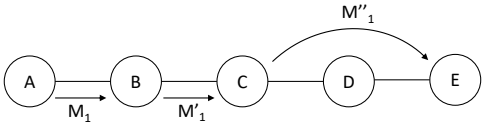
A Tabela 1 apresenta de forma sintética uma comparação das principais características dos sistemas mencionados anteriormente. Como se pode verificar, os trabalhos anteriores oferecem apenas um solução parcial para o problema que abordamos no nosso trabalho. O algoritmo que apresentamos de seguida, que designámos por LoCaMu, consegue oferecer tolerância a faltas, garantir ordem causal, ter várias mensagens em trânsito em paralelo para a mesma fonte e, finalmente, recorre apenas a informação localizada. Desta forma combina garantias de qualidade de serviço com a capacidade de escala.

3 LoCaMu

Nesta secção descrevemos o LoCaMu. Começamos por introduzir a noção de grafo base e grafo de comunicação estendido assim como o modelo de faltas e o modelo de endereçamento considerado. De seguida, introduzimos as estruturas de dados usados pelo algoritmo e, posteriormente, descrevemos como se processa o envio, receção, reencaminhamento e retransmissão de mensagens.



(a) Grafo Base e Grafo Estendido



(b) Reencaminhamento de uma mensagem

Figura 1. Grafo base, grafo estendido e reencaminhamento de uma mensagem

3.1 Grafo Base e Grafo Estendido

Assumimos que os nós estão organizados num grafo não direcionado $G = (v, a)$, em que os vértices v representam os nós do sistema e as arestas a os canais de comunicação entre estes vértices. No resto do texto, em função do contexto, tanto usamos a expressão nó como a expressão vértice, para referir um participante no sistema. Este grafo, que designamos por *grafo base*, é acíclico e completamente ligado. Seja v_i um vértice do grafo. Designamos por $\mathcal{V}(G, k, v_i)$ o conjunto de vizinhos do vértice v_i a, no máximo, k saltos de distância. Ou seja, se $v_j \in \mathcal{V}(G, k, v_i)$, então o caminho mais curto no grafo base entre v_i e v_j tem no máximo k arestas. Os vértices em $\mathcal{V}(G, 1, v_i)$ designam-se por vizinhos diretos de v_i (no grafo base). O grafo base não possui arestas redundantes e a falha de qualquer nó particiona o grafo.

De forma a tolerar f falhas numa dada vizinhança, o grafo base G é aumentado com arestas adicionais, de forma a criar um *grafo estendido* G^f. O grafo estendido G^f é construído a partir do grafo base, acrescentando uma aresta entre cada vértice v_i e todos os seus vizinhos a $f + 1$ saltos, isto é, entre v_i e $v_j \in \mathcal{V}(G, f + 1, v_i)$. A Figura 1a ilustra um grafo base e o grafo estendido correspondente para $f = 1$. Para tolerar falhas, quando um nó v_i precisa de encaminhar uma mensagem para um nó vizinho que está falhado, v_i encaminha a mensagem diretamente para os nós subsequentes a este.

3.2 Modelo de Faltas

Assumimos que os nós podem falhar e posteriormente recuperar, sendo mantido o seu estado em memória persistente. O sistema é assíncrono e não são definidos limiares para o tempo em que um nó pode estar indisponível. Finalmente, assumimos que, em qualquer instante de tempo, em qualquer vizinhança de tamanho $2f + 1$ no grafo base, não existem mais do que f nós indisponíveis. Um nó que nunca falha designa-se por correto.

Assumimos que cada nó do sistema tem acesso a um detetor de falhas eventualmente perfeito [7] que lhe indica se os seus vizinhos estão disponíveis ou indisponíveis. Este detetor garante que se um vizinho fica indisponível, o nó é

notificado (para evitar estar a transmitir mensagens em vão) e, de forma semelhante, quando o vizinho recupera o nó também acaba por ser notificado (de forma a poder usar de novo esse vizinho).

O algoritmo LoCaMu é tolerante a faltas no sentido em que uma mensagem m enviada para um conjunto de vértices V é entregue a todos os nós corretos $v_j \in V$ (isto é, a todos os destinatários que não estejam ou que não venham a falhar), independentemente da falha de outros nós ou da eventual recuperação de nós que se encontrem falhados. O LoCaMu assegura também que os nós que estão temporariamente indisponíveis acabam também por receber todas as mensagens; naturalmente, isto só acontece quando estes recuperam.

3.3 Modelo de Endereçamento

Assumimos que os nós se organizam em grupos que são usados no endereçamento de mensagens, sendo que cada mensagem tem como destinatário um único grupo. A comunicação usa um modelo de edição-subscrição, em que a informação sobre os membros do grupo é propagada no grafo estendido através de mensagens de *subscrição*. Isto permite manter em cada nó uma tabela de encaminhamento que, para cada grupo, indica quais os vizinhos que necessitam de participar no reencaminhamento das mensagens. O tamanho desta tabela depende do número de grupos e não do número de nós do sistema.

Dada uma mensagem m, que tem por destinatário um grupo g, um dado nó i pode consultar a sua tabela de encaminhamento para identificar o sub-conjunto dos seus vizinhos no grafo estendido que está envolvido no encaminhamento de m. Chamamos a este sub-conjunto os *alvos* de m em i, que designamos por $\mathcal{A}(m, i)$.

3.4 Estruturas de Dados

No LoCaMu, cada nó mantém informação sobre mensagens que tenham sido enviadas para ou recebidas por nós com os quais comunica diretamente. Como referimos, cada nó i comunica diretamente com os nós que estão na sua vizinhança $f + 1$ no grafo base, ou seja, com nós em $\mathcal{V}(G, f + 1, i)$. Por sua vez, todos os nós que estão em condições de enviar uma mensagem para um nó em $\mathcal{V}(G, f + 1, i)$, estão necessariamente contidos na vizinhança $\mathcal{V}(G, 2f + 2, i)$. Designamos esta última vizinhança a *vizinhança de segurança* de i, que denotamos por $\mathcal{V}^S(i)$. Desta forma, cada nó, só mantém informação sobre nós em $\mathcal{V}^S(i)$.

Cada nó mantém uma matriz de números de sequência, chamada *passado*, que denotamos por P_i, com uma entrada para cada par de nós em $\mathcal{V}^S(i)$. P_i captura o passado causal do nó i. Considere que a posição $P_i[j][k] = s$ ($s \neq 0$). Isto significa que o estado do nó i depende causalmente de uma mensagem enviada do nó j para o nó k com o número de sequência s. Assinala-se que a linha $P_i[i]$ regista a última mensagem enviada por i para cada vizinho j.

Finalmente, cada nó mantém uma matriz de identificadores *recebidos*, R_i, com uma entrada para cada par de nós em $\mathcal{V}^S(i)$, em que cada entrada $R_i[k][l]$

contém o conjunto dos identificadores das mensagens enviadas de k para l e das quais uma cópia foi já processada no nó i

Cada mensagem trocada no sistema carrega também metadados. Mais concretamente, cada mensagem m transporta os seguintes campos de controlo. O primeiro, *identificadores* (I_M), é uma matriz que tem até $2f + 1$ vetores que contém os números de sequência conhecidos que foram atribuídos à mensagem m por nós em $\mathcal{V}^S(i)$. O segundo, *dependências*, de tamanho $|\mathcal{V}^S(i)|^2$, que denotamos por D_m, transporta o passado causal de m conhecido pelo encaminhador.

3.5 Encaminhamento de Mensagens

No LoCaMu as mensagens podem necessitar de ser reencaminhadas ao longo do grafo para chegarem ao destino. Por exemplo, uma mensagem m é enviada de a para b. Posteriormente, b cria uma cópia m' de m que é enviada para c, e assim sucessivamente. Desta forma, a pode enviar m para grupos com membros fora da sua vizinhança. Para tolerar falha de nós, o LoCaMu pode reencaminhar uma mensagem por mais que um caminho. É pois possível que um nó receba várias cópias de uma mesma mensagem.

Quando um nó i cria uma cópia m' de m com o objetivo de reencaminhar m, não envia m' de imediato para todos os alvos $\mathcal{A}(m, i)$. Como veremos mais à frente, nem sempre é possível reencaminhar a mensagem de imediato, sob pena de violar as propriedades do algoritmo. Desta forma, cada nó mantém uma fila de mensagens a reencaminhar no futuro, que é despejada ao longo do tempo.

3.6 Transmissão de uma Mensagem

Seja m uma mensagem gerada no nó i. Para transmitir m, o nó i faz as seguinte manipulações ao seu estado e aos metadados de m. Primeiro, i consulta a sua tabela de encaminhamento para obter os alvos de m, $\mathcal{A}(m, i)$. Depois, i coloca $D_m = P_i$. Depois, incrementa os valores de $P_i[i][j]$, $\forall j \in \mathcal{A}(m, i)$ e adiciona um vetor novo com as entradas alteradas a I_m. A mensagem m é então colocada na fila de mensagens a reencaminhar. Finalmente, o procedimento para despejar a fila de mensagens é invocado. Este procedimento, descrito adiante, escolhe quais os alvos para onde a mensagem é enviada em primeiro lugar.

3.7 Receção de uma Mensagem

Quando o nó j recebe uma mensagem m, que foi enviada pelo nó i, j executa os passos apresentados no procedimento *rececaoDeMensagem* na Figura 2. Em síntese, o nó espera até que todas as dependências causais da mensagens estejam satisfeitas (isto é, as mensagens no passado já tenham sido entregues) e depois, se a mensagem não for um duplicado, entrega-a e coloca-a na fila de mensagens a reencaminhar. Quando uma cópia de uma mensagem m já anteriormente processada é recebida, atualiza-se a lista de identificadores atribuídos à mensagem fazendo a união dos identificadores recebidos na nova cópia com os identificadores já conhecidos localmente. Esta junção é apenas efetuada nas mensagens que estão por encaminhar.

procedure *transmitirMensagem* (m) no nó i
  call *estampilharEnviar* (m);

procedure *estampilharEnviar* (m) no nó i
  foreach $k, l \in \mathcal{V}^S(i)$ **do** $D_m[k][l] \leftarrow max(D_m[k][l], P_i[k][l])$;
  foreach $k \in \mathcal{A}(m, i)$ **do** { $P_i[i][k] \leftarrow P_i[i][k] + 1$; $I_m[i][k] \leftarrow P_i[i][k]$; }
  alvos $(m) \leftarrow \mathcal{A}(m, i)$;
  pendentes$_i \leftarrow$ pendentes$_i \cup \{m\}$;
  call *reencaminharMensagem* (m);

procedure *receberMensagem* (m) no nó j que foi enviada pelo nó i
  while $\exists k : P_j[k][j] < D_m[k][j]$ **do** *wait*;
  foreach $k, l \in \mathcal{V}^S(i)$ **do** $P_j[k][l] \leftarrow max(P_j[k][l], D_m[k][l], I_m[k][l])$.
  if *mensagemDuplicada* (m) **then**
    $c \leftarrow$ *copiaPendente* (pendentes$_i$, m);
    foreach $k, l \in \mathcal{V}^S(i)$ **do** $I_c[k][l] \leftarrow max(I_c[k][l], I_m[k][l])$;
    call *reencaminharMensagem* (c);
  else
    if $j \in$ *grupoDestinatarios*(m) m **then** entrega m à aplicação
    call *estampilharEnviar* (m);
  foreach $k, l \in \mathcal{V}^S(i)$ **do** $R_j[k][l] \leftarrow R_j[k][l] \cup \{I_m[k][l]\}$;

procedure *rencaminharMensagem* (m) no nó i
  $I_m \leftarrow removerAlvosAntigos(I_m)$
  foreach $k \in$ alvos(m) **do**
    if *reencaminhamentoSeguro* (m, k) **then**
      alvos$(m) \leftarrow$ alvos$(m) \setminus \{k\}$;
      call *send* (m, k);

function *mensagemDuplicada* (m) no nó i
  if $\exists k, l \in \mathcal{V}^S(i) : I_m[k][l] \in R_i[k][l]$ **then return** true **else return** false;

Figura 2. Algoritmo LoCaMu

3.8 Reencaminhamento de uma Mensagem

Infelizmente, o uso de informação localizada acarreta risco de se perder informação sobre o modo de como as mensagens estão relacionadas quando se mantém e propaga apenas informação parcial. Usando um exemplo, ilustra-se um cenário onde uma mensagem duplicada não é detetada como tal. Considerando o cenário da Figura 3 que tem $f = 1$, o nó a publica m e encaminha para b e c. Devido a problemas na rede, b apenas reencaminha m' para d e c apenas para e, com d encaminhando m''' para e. Devido às mensagens apenas conterem identificadores dos últimos $2f + 1$ nós quando chega a um dado nó, m'' contém apenas o identificador de c e m''' contém os de b e d. Assim são vistas como sendo mensagens diferentes.

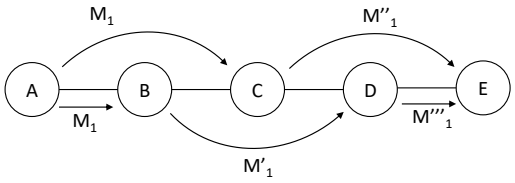


Figura 3. Deteção de duplicados

Este problema é superado ao não permitir uma mensagem ser reencaminhada a não ser que tenha f ou mais identificadores no sub-caminho dos últimos $2f+1$ nós até ao destino. Assim é garantido que há sempre um nó capaz de detetar mensagens duplicadas em qualquer sub-caminho. Esta condição é capturada pelo predicado *reencaminhamentoSeguro* (m, k), que é verdadeiro se em I_m existirem f ou mais identificadores de nós distintos quando esta mensagem já foi reencaminhada numa dada distância de tamanho $2f + 1$. Visto que o grafo é estático e cada nó tem noção da sua vizinhança, é possível inferir quais são os identificadores que estão em falta.

O procedimento *reencaminharMensagem* verifica quais os alvos da mensagem m são seguros e envia a mensagem para esses alvos. Este procedimento é re-executado sempre que uma cópia da mensagem é recebida, uma vez que a receção de uma cópia permite atualizar a lista de identificadores conhecidos tornando seguros alvos que anteriormente não o eram. O procedimento *removerAlvosAntigos* pega na matriz $identificadores$ e remove os vetores que foram adicionados por nós a mais de $2f + 1$ nós de distância.

3.9 Recuperação de Falhas nos Canais

Quando o nó a comunica com b mas este falhou, a comunica diretamente com c mas é possível que este não tenha recebido todas as mensagens que a enviou. Neste caso c, ao perceber que faltam mensagens, envia um vetor de dependências a a que contém as últimas mensagens recebidas direcionadas a c, para que a possa retransmitir as mensagens em falta e/ou as que causam estas.

3.10 Otimizações

Para simplificar a exposição, na versão acima descrita do algoritmo, os conjunto mantidos na matriz R_i nunca são expurgados de valores antigos. Isto obrigaria a manter memória infinita. Na prática, uma sequência contínua de identificadores de 1 a n pode ser representada pelo último identificador n. Note-se, no entanto, que os identificadores podem não ser conhecidos pela ordem pelo que foram enviados (dado que as mensagens podem percorrer diferentes caminhos concorrentemente), pelo que nem sempre o conjunto pode ser reduzido a um único identificador.

O Algoritmo apresentado na Figura 2 tem cada nó a reencaminhar cada mensagem para todos os alvos. Isto é desnecessário, pois se não houver falhas, é

suficiente transmitir a mensagem para os vizinhos diretos para que ela chegue a todos os destinatários, só sendo preciso enviar a mensagem aos vizinhos destes se este estiver em baixo. Para fazer esta otimização é necessário fazer algumas alterações ao algoritmo, mas por razões de espaço não a elaboramos.

4 Avaliação

Nesta secção avaliamos o desempenho do LoCaMu recorrendo a simulações. Comparamos também o seu desempenho com os dois sistemas do trabalho relacionado que mais se aproximam do nosso, nomeadamente o sistema baseado em Barreiras Causais [15] (BC) que é um dos poucos que assegura ordem causal (apesar de não ser localizado) e o Vizinhança-Delta [10] (VD) que é o único que usa informação localizada (apesar de não garantir a ordem causal). Avaliamos como estes sistemas se comportam quando variamos diferentes características do sistema tal como a largura de banda disponível nos canais de comunicação, o número total de nós da rede, e o número de emissores enviando mensagens concorrentes. Apresentamos também a quantidade máxima de metadados presentes em cada mensagem, para os vários algoritmos.

Para realizar as simulações usamos o Peersim [14] com uma extensão que simula a latência na rede, sendo adicionado uma latência mínima de 1ms entre cada nó, e largura de banda, que é dividida igualmente por cada ligação que um nó tem. Os canais entre os nós garantem ordem FIFO no envio e receção de mensagens. De forma a fazer uma comparação justa foram executadas simulações sem falhas e no caso do LoCaMu este apenas envia as mensagens para os nós diretamente vizinhos, de forma a poupar largura de banda. Tendo em atenção que se pretende analisar o custo inerente ao tamanho das mensagens e o débito máximo, o tempo de processamento é ignorado, o que beneficia o BC e prejudica o VD. A topologia usada na rede sobreposta que liga os nós é uma árvore binária. Finalmente, apenas é usado um grupo.

A Figura 4a) ilustra o débito útil (*goodput*) atingido pelos vários algoritmos quando existe apenas um emissor e se varia a largura de banda. Como seria de esperar, quer o LoCaMu quer o BC aumentam o débito quando a largura de banda aumenta (note-se que neste gráfico, o débito aparece em escala logarítmica). Já o VD, devido ao facto de só enviar uma mensagem depois da anterior ter sido confirmada, não consegue beneficiar do aumento da largura de banda nos canais.

A Figura 4b) ilustra o débito atingido pelos vários algoritmos quando se varia o tamanho da rede (de novo, apenas com um único emissor). Neste caso, o débito do LoCaMu mantém-se constante, pois os seus metadados não dependem do tamanho da rede. O débito do VD diminui com o tamanho da rede, pois quando o diâmetro da rede aumenta, o tempo necessário para receber a confirmação de receção da última mensagem também aumenta. O algoritmo mais afetado é o BC, pois o tamanho dos metadados cresce de forma quadrática com o tamanho da rede.

A Figura 4c) ilustra o débito atingido pelos vários sistemas quando se aumenta o número de emissores enviando mensagens de forma concorrente. O Lo-

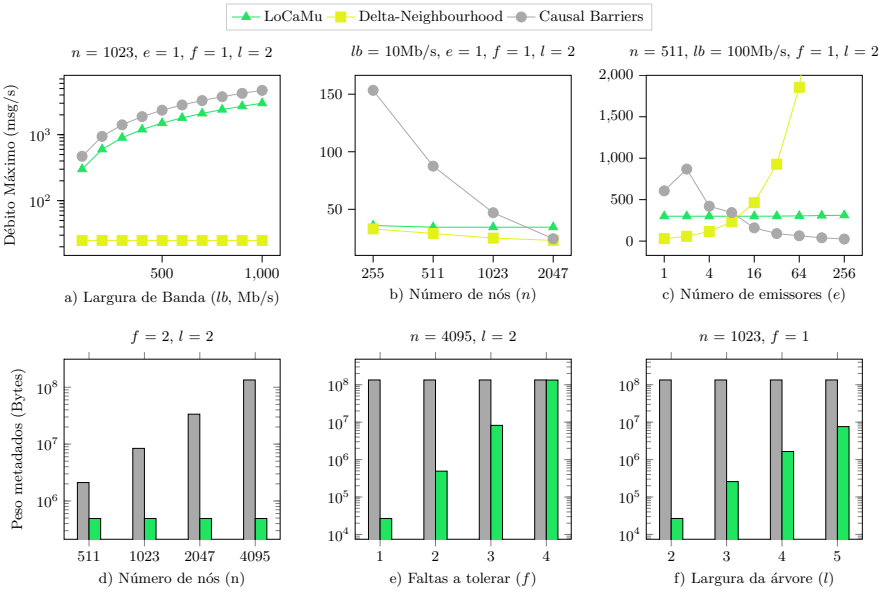


Figura 4. Desempenho do LoCaMu vs. o desempenho dos algoritmos BC [15] e VD [10]

CaMu não é afetado por esta variação, pois está sempre limitado pelo fluxo máximo da rede sobreposta (no caso destas experiências, pelos canais do nó raiz da árvore), independentemente do número de emissores. Já o VD, devido às limitações anteriormente referidas, não consegue atingir o fluxo máximo da rede a não ser quando se usam muitos emissores (pois cada emissor tem um débito limitado pelo diâmetro da rede); neste caso, como usa menos metadados, consegue um maior débito útil. Por sua vez, o BC tem um decréscimo ao se aumentarem os emissores, pois os metadados crescem com o grau de concorrência no sistema.

Nas figuras 4d), 4e) e 4f) apresenta-se a quantidade de metadados existentes numa mensagem no pior caso, para diferentes tamanhos da rede, valores do número de faltas a tolerar e larguras da árvore. Para o BC o pior caso acontece quando todos nós emitem eventos simultaneamente; no caso do LoCaMu o pior caso depende do tamanho da vizinhança de segurança dos nós, que por sua vez depende do número de faltas que se pretende tolerar; finalmente o VD usa metadados constantes. Como se pode ver, para redes de grande dimensão e para valores de f baixos (≤ 2), o LoCaMu usa substancialmente menos metadados do que o BC. Para além disso, os metadados do LoCaMu não variam com o tamanho da rede, enquanto que os metadados do BC aumentam de forma quadrática com o tamanho da rede. No entanto, como a vizinhança de segurança do LoCaMu cresce exponencialmente com f, para valores de $f > 2$ os metadados de LoCaMu aproximam-se dos metadados do BC. Por fim, em 4f), que é o único cenário onde a árvore não é binária, observa-se o mesmo para diferentes larguras de árvore

5 Conclusão

Neste artigo apresentámos o LoCaMu, um algoritmo que garante a entrega de mensagens respeitando a ordem causal num sistema de edição-subscrição construído sobre uma rede sobrepostas de corretores de mensagens. Tanto quanto é do nosso conhecimento, o LoCaMu é o primeiro algoritmo que garante causalidade e tolerância a faltas usando informação localizada, i.e, cada nó tem de manter metadados referentes apenas aos nós na sua vizinhança e não referentes a todos os nós do sistema. Uma avaliação experimental do LoCaMu, recorrendo a simulações, mostra que este algoritmo consegue suportar débitos mais elevados do que outros algoritmos anteriores, que usam muito mais metadados ou que limitam o número de mensagens em trânsito.

Agradecimentos: Este trabalho foi suportado pela FCT – Fundação para a Ciência e a Tecnologia, através dos projectos UID/CEC/50021/2019 e COSMOS (financiado pelo OE com a ref. PTDC/EEI-COM/29271/2017 e pelo Programa Operacional Regional de Lisboa na sua componente FEDER com a ref. Lisboa-01-0145-FEDER-029271).

Referências

1. Banavar, G., et. al: An efficient multicast protocol for content-based publish-subscribe systems. In: ICDCS. pp. 262–272. Austin (TX), USA (Jun 1999)
2. Bhola, S., et. al: Exactly-once delivery in a content-based publish-subscribe system. In: DSN. pp. 7–16. Bethesda (MD), USA (Jun 2002)
3. Birman, K., et. al: Bimodal multicast. TOCS **17**(2), 41–88 (May 1999)
4. Birman, K., Schiper, A., Stephenson, P.: Lightweight causal and atomic group multicast. TOCS **9**(3), 272–314 (Aug 1991)
5. Bravo, M., Rodrigues, L., van Roy, P.: Saturn: a distributed metadata service for causal consistency. In: EuroSys. pp. 111–126. Belgrade, Serbia (Apr 2017)
6. Eugster, P., Felber, P., Guerraoui, R., Kermarrec, A.M.: The many faces of publish/subscribe. CSUR **35**(2), 114–131 (Jun 2003)
7. Guerraoui, R., Rodrigues, L.: Introduction to Reliable Distributed Programming. Springer (2006)
8. Gupta, I., van Renesse, R., Birman, K.: Scalable fault-tolerant aggregation in large process groups. In: DSN. pp. 433–442. Göteborg, Sweden (Jul 2001)
9. JGroups: http://www.jgroups.org/index.html, accessed: 2019-06-29
10. Kazemzadeh, R., Jacobsen, H.: Partition-tolerant distributed publish/subscribe systems. In: SRDS. pp. 101–110. Madrid, Spain (Oct 2011)
11. Lamport, L.: The part-time parliament. TOCS **16**(2), 133–169 (May 1998)
12. Lin, J., Paul, S.: RMTP: a reliable multicast transport protocol. In: INFOCOM. pp. 1414–1424 vol.3. San Francisco (CA), USA (Mar 1996)
13. Mahajan, P., Alvisi, L., Dahlin, M., et al.: Consistency, availability, and convergence. University of Texas at Austin Tech Report (2011)
14. Montresor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: P2P. pp. 99–100. Seattle (WA), USA (Sep 2009)
15. Prakash, R., Raynal, M., Singhal, M.: An efficient causal ordering algorithm for mobile computing environments. In: ICDCS. pp. 744–751. Hong Kong (May 1996)
16. Renesse, R.V., et. al: Horus: A flexible group communications system. Tech. rep., Cornell University, Ithaca (NY), USA (Mar 1995)
17. Rowstron, A., et. al: Scribe: The design of a large-scale event notification infrastructure. In: NGC. pp. 30–43. London, UK (Jul 2001)