SmartGossip: Difusão Probabilística Inteligente de Mensagens em Redes de Topologia Arbitrária

Wellington Souza, Aurora Pozo, Elias P. Duarte Jr.

Departamento de Informática – Universidade Federal do Paraná (UFPR)
Caixa Postal 19018 – 81531-980 – Curitiba – PR

{wgvs16,aurora,elias}@inf.ufpr.br

Resumo. *A difusão de mensagens é uma das abstrações mais poderosas de sistemas distribuídos, permitindo que uma mensagem seja transmitida de um para todos os processos do sistema, sendo necessária para a execução de diversos tipos de tarefas e aplicações. Esse trabalho apresenta o algoritmo SmartGossip, para a difusão probabilística e inteligente de mensagens em redes de topologia arbitrária, inspirada em uma estratégia de otimização baseada em colônias de formigas, mais especificamente pelo conceito de comunicação por estigmergia. O algoritmo SmartGossip é especificado e avaliado através de simulação no OMNeT++. Foram executados experimentos medindo o número de mensagens utilizadas e o número de rodadas necessárias para completar a difusão em redes de até 1024 processos com conectividades diversas. O SmartGossip é comparado com algoritmos baseados em inundação (Flooding) e em difusão probabilística (Gossip tradicional). Os resultados mostram que o número médio de mensagens utilizadas pelo SmartGossip é sempre menor que dos demais algoritmos, além de apresentarem baixa dispersão na medida em que o tamanho do sistema e conectividade variam.*

1. Introdução

Um sistema distribuído é um conjunto de processos que se comunicam e colaboram para realizar uma tarefa [Cachin et al. 2011]. Muitas destas tarefas necessitam que um processo transmita uma mensagem para todos os processos do sistema. Algoritmos de difusão (*broadcast*) permitem justamente a comunicação de um-para-todos, sendo imprescindíveis para diversos sistemas e aplicações distribuídas, inclusive da Internet. Neste contexto, especialmente para sistemas distribuídos de grande escala, é desejável que o tempo para completar a transmissão de mensagens (*i.e.* a latência) seja baixa, bem como o número de mensagens utilizadas.

A primitiva clássica de sistemas distribuídos para realizar a difusão de mensagens de forma confiável é o *broadcast* [Rodrigues et al. 2014]. O tipo de difusão considerado aqui é o de melhor esforço, que deve garantir a propriedade da validade. Se um processo faz a difusão de uma mensagem e permanece correto, então todos os processos corretos entregam aquela mensagem. Além disso, nenhum processo entrega uma determinada mensagem mais de uma vez, não há mensagens espúrias e a difusão termina depois de um intervalo finito de tempo.

Uma forma simples e direta para garantir a difusão confiável é apresentada pelo algoritmo de inundação (Flooding) [Duarte Jr and Mattos 2000]. O algoritmo consiste em garantir que cada processo do sistema transmite a mensagem para todos seus vizinhos

conhecidos ao recebê-la, garantindo que todo processo não-falho do sistema distribuído receberá a mensagem. Como ponto negativo, o algoritmo utiliza um grande número de mensagens redundantes, já que um processo não tem conhecimento do recebimento da mensagem pelos outros processos do sistema.

Algoritmos de difusão probabilística representam uma alternativa que gera menor número de mensagens. Apesar de manterem uma boa probabilidade de que todos os processos entreguem uma mensagem m, algoritmos probabilísticos não garantem que 100% dos processos sempre entregam m. O algoritmo clássico de difusão probabilística é também chamado de Gossip [Eugster et al. 2004]. No Gossip, ao receber uma mensagem m, um processo transmite a mensagem m apenas para um subconjunto aleatório de processos da sua vizinhança, de tamanho definido por um parâmetro conhecido como *fanout*. Esse procedimento se repete por um número predeterminado de rodadas. Tanto o número de mensagens transmitidas como a probabilidade de que todos os processos entreguem a mensagem m são afetados diretamente pelo parâmetro *fanout* e pelo número máximo de rodadas.

No presente trabalho, um algoritmo inteligente para a disseminação probabilística de mensagens é proposto, denominado SmartGossip. O algoritmo é inspirado na estratégia de otimização conhecida como colônia de formigas [Dorigo et al. 2006], utilizando o conceito de comunicação por estigmergia [Dorigo et al. 2000]. Aplicações desta estratégia foram anteriormente propostas no contexto de redes de computadores, como por exemplo para balanceamento de carga [Muteeh et al. 2021], descoberta de topologia de redes dinâmicas [Nassu et al. 2007], a disseminação de eventos [Banzi et al. 2011] e o roteamento [Ramamoorthy and Thangavelu 2021]. Por outro lado, acreditamos que o algoritmo SmartGossip é o primeiro esforço de uso de estimergia para difusão probabilística de mensagens. O SmartGossip deve atender à propriedade de validade probabilística: se um processo faz a difusão de uma mensagem e permanece correto, então todos os processos corretos entregam aquela mensagem com uma determinada probabilidade. Além disso, não há mensagens espúrias nem entregas duplicadas.

A estratégia utilizada pelo algoritmo SmartGossip consiste em utilizar "depósitos de feromônios" nos enlaces de um sistema distribuído, armazenados localmente por cada processo, como um critério de decisão para os próximos destinos de uma mensagem. Quando um processo recebe uma mensagem por meio de um enlace, uma quantidade fixa de feromônios é depositada naquele enlace. A escolha do próximo destino da mensagem é probabilística, sendo influenciada pela concentração de feromônios em cada enlace do processo para sua vizinhança. Enlaces com uma concentração menor de feromônios possuem maior probabilidade de serem escolhidos, fazendo com que a transmissão da mensagem sempre priorize caminhos ainda não percorridos, aumentando a chance de que processos que ainda não entregaram a mensagem a recebam.

O algoritmo SmartGossip foi implementado utilizando o simulador OMNeT++[1]. Foram executados experimentos medindo o número de mensagens utilizadas e o número de rodadas necessárias para completar a difusão em sistemas com o número de processos variando de 64 a 1024 processos. Para cada tamanho de sistema, foram geradas topologias correspondentes a grafos randômicos com conectividades variando de 0,5 a 1,0,

[1]https://omnetpp.org

caso em que o grafo é completo. Foram também implementados para comparação os algoritmos clássicos de difusão de mensagens Flooding e Gossip. Os resultados mostram que o número médio de mensagens utilizadas pelo SmartGossip é sempre menor que dos demais algoritmos, além de apresentarem baixa dispersão na medida em que o tamanho do sistema e conectividade variam.

O restante do trabalho está organizado da seguinte forma. A Seção 2 apresenta os dois algoritmos distribuídos clássicos para difusão de mensagens em redes de topologia arbitrária: Flooding e Gossip. Na Seção 3 o algoritmo SmartGossip é apresentado. A Seção 4 descreve o simulador implementado e apresenta os resultados obtidos nas comparações entre o Flooding, Gossip e SmartGossip. As conclusões seguem na Seção 5.

2. Difusão de Mensagens em Sistemas Distribuídos

A difusão de mensagens (*broadcast*) em um sistema distribuído permite que um processo envie uma mensagem para todos os processos do sistema. A difusão que se considera aqui é a de melhor esforço, que deve atender à propriedade de validade: se dois processos i e j permanecem corretos e i executa *broadcast(msg)* então j entrega *msg* após um tempo. Além disso não são entregues mensagens duplicadas nem espúrias.

Nesta seção são descritos dois algoritmos distribuídos clássicos para difusão de mensagens: inundação, ou Flooding e a difusão probabilística, ou Gossip. Estes algoritmos foram implementados e resultados de comparação com a estratégia inteligente proposta são apresentados na Seção 4.

2.1. Modelo de Sistema

O sistema distribuído aqui considerado possui topologia arbitrária (i.e. sistema *multi-hop*), podendo ser definido como um grafo $G = (V, E)$. Cada vértice $v \in V$ representa um processo, e cada aresta $e \in E$ representa um enlace. Sabendo que $i \in V$ e $j \in V$ são dois processos distintos, e que $e_{ij} \in E$ é um enlace que os conecta, assume-se que os enlaces são perfeitos, garantindo que uma dada mensagem não será entregue mais que uma vez e que as mensagens transmitidas são recebidas em algum momento. Para reforçar, os enlaces não são direcionados, portanto $e_{ij} = e_{ji}$. Um processo $i \in V$ tem conhecimento de sua vizinhança $N_i \subseteq V$. A vizinhança N_i consiste dos processos j tal que $e_{ij} \in E$, isto é, os processos adjacentes a i em G.

Assume-se um sistema parcialmente síncrono GST (*Global Stabilization Time*) [Cachin et al. 2011]. Nesse modelo, o sistema começa instável e, após um certo instante de tempo desconhecido, passa a se comportar como síncrono para sempre. O modelo de falhas utilizado é por parada (*crash*). Um processo falho interrompe sua execução completamente, não produzindo nenhuma saída e perdendo o estado interno.

Além disso, são definidas as seguintes primitivas: ***send(dst, src, msg),*** utilizada pelo processo *src* para transmitir a mensagem *msg* para o processo *dst*; ***broadcast(msg),*** dispara a difusão da mensagem *msg* para todos os processos do sistema; ***receive(src, dst, msg),*** utilizada por um processo *dst* para receber uma mensagem *msg* vinda do processo *src*; ***deliver(msg),*** utilizada por um processo para entregar uma mensagem *msg*. Observação importante: em alguns dos algoritmos abaixo, as primitivas são utilizadas com diferentes parâmetros, definidos caso a caso.

2.2. O Algoritmo de Inundação (Flooding)

O algoritmo de inundação ou Flooding, baseia-se na estratégia de "inundar" um sistema com mensagens: um processo retransmite mensagens para todos os seus vizinhos. Seja $N_i \subseteq V$ a vizinhança de um processo i. No Flooding, o processo i executando a primitiva *broadcast(msg)* transmite a mensagem a todo processo $j \in N_i$. Por sua vez, ao receber uma nova mensagem, um processo $j \in N_i$ entrega a mensagem caso seja a primeira vez que a está recebendo. Além disso, o processo j retransmite a mensagem a todo processo $k \in N_j - \{i\}$. O procedimento se repete até que todos os processos do sistema tenham recebido a mensagem.

O algoritmo de inundação não é um algoritmo eficiente em relação ao número de mensagens utilizadas. Como todos processos enviam cada mensagem para todos seus vizinhos, em geral um grande número de mensagens repetidas serão transmitidas. No pior caso, o número de mensagens transmitidas é $|E|$. Por outro lado, se um caminho existe entre dois processos, o algoritmo de inundação irá garantir que uma mensagem *msg* enviada utilizando *broadcast(msg)* será sempre entregue.

2.3. O Algoritmo de Difusão Probabilística (Gossip)

O algoritmo de difusão probabilística [Eugster et al. 2004], conhecido também como Gossip, se propõe a solucionar o problema de desperdício de mensagens presente no algoritmo de inundação, limitando o número de mensagens enviadas. O Gossip se baseia na forma como epidemias se alastram, com cada processo que recebe a mensagem transmitindo esta para um outro grupo aleatório de processos vizinhos, até que todos os processos do sistema recebam a mensagem.

Para definir a taxa na qual as mensagens são disseminadas de forma epidêmica, o Gossip utiliza o parâmetro *fanout*. Seja f o valor do *fanout* para uma determinada execução de *broadcast(msg)* e seja $N_i \in V$ a vizinhança do processo i. Ao executar a primitiva *broadcast(msg)* o processo i envia a mensagem para um subconjunto aleatório $S \subseteq N_i$ de no máximo f processos em sua vizinhança, tal que $|S| = min\{f, |N_i|\}$. Ao receber uma nova mensagem, um processo j a retransmite para outros $min\{f, |N_j - \{i\}|\}$ vizinhos, até que todos os processos do sistema tenham recebido a mensagem disseminada ou até que o número máximo de rodadas - definido pelo parâmetro *maxRounds* - seja atingido. O pseudocódigo do Gossip pode ser visto no Algoritmo 1.

Nesse algoritmo, as primitivas são utilizadas com os parâmetros descritos a seguir: ***send(dst, src, msg, round),*** utilizada por um processo *src* para transmitir uma mensagem *msg* para o processo *dst* na rodada *round*; ***receive(src, dst, msg, round),*** utilizada por um processo *dst* para receber uma mensagem *msg* vinda do processo *src* na rodada *round*.

Todo processo i do sistema distribuído executa o Algoritmo 1. Assim como no Algoritmo 1, ao iniciar a execução, o processo i inicializa o conjunto de mensagens entregues *delivered*. Ao executar a primitiva *broadcast(msg)*, o processo i escolhe um subconjunto aleatório S de $min\{f, |N_i|\}$ processos de sua vizinhança N_i e transmite a mensagem *msg* para todo $j \in S$, incluindo a mensagem, o identificador de origem e o identificador da rodada. A rodada inicial é dada por $maxRounds - 1$, onde $maxRounds > 0$ é um parâmetro definido antes da execução do algoritmo para limitar o número de mensagens enviadas. Ao executar a primitiva *receive(src, dst, msg, round)*, o processo i verifica se a mensagem recebida de *src* já foi entregue e a armazena caso ainda não tenha sido. Em

Algoritmo 1 Difusão Probabilística de Mensagens (executado pelo processo i)

1: ***Init:***
2: $delivered \leftarrow \emptyset$

3: ***Upon broadcast(msg):***
4: $S \leftarrow$ random subset of f processes of N_i
5: **for all** $j \in S$ **do**
6: *send*$(j, i, msg, maxRounds - 1)$
7: **end for**

8: ***Upon receive(src, i, msg, round):***
9: **if** $msg \notin delivered$ **then**
10: $delivered \leftarrow delivered \cup \{msg\}$
11: *deliver(msg)*
12: **end if**
13: **if** $round > 0$ **then**
14: $S \leftarrow$ random subset of f processes of N_i
15: **for all** $j \in S$ **do**
16: *send*$(j, i, msg, round - 1)$
17: **end for**
18: **end if**

seguida, se o número máximo de rodadas ainda não foi atingido, o processo i continua a execução, escolhendo um subconjunto S de $min\{f, |N_i - \{src\}|\}$ processos de $N_i - \{src\}$ e retransmitindo a mensagem *msg* para todo $j \in S$.

O número de mensagens enviadas é diretamente influenciado pelo parâmetro f (*fanout*). Quando temos $f = |V| - 1$, o número de mensagens enviadas será igual ao do algoritmo de inundação apresentado na seção anterior. Ao escolhermos um número menor para o parâmetro f, os processos escolhidos para o conjunto S serão aleatórios, e portanto, para garantir que a mensagem será entregue para todos os processos, um valor adequado de número máximo de rodadas (*maxRounds*) é necessário. É importante reforçar que o algoritmo é probabilístico, ou seja há uma boa probabilidade que todos os processos recebam a mensagem, não uma garantia.

3. O Algoritmo SmartGossip

O algoritmo SmartGossip para difusão probabilística de mensagens é inspirado em inteligência de enxames, mais especificamente colônias de formigas [Dorigo et al. 2006]. A abordagem utilizada pelo algoritmo é baseada no conceito de estigmergia [Dorigo et al. 2000] como estratégia de comunicação para otimizar o desempenho da difusão de informações entre nodos de um sistema distribuído. A estigmergia permite que formigas se comuniquem indiretamente, através de feromônios cuja concentração dita as ações executadas pela colônia coletivamente. Os feromônios são substâncias químicas produzidas pelos indivíduos que, ao serem disseminadas, disparam reações específicas entre outros indivíduos da mesma espécie. Esta seção inicia descrevendo a estratégia baseada em inteligência de enxames do SmartGossip para, em seguida, apresentar o algoritmo.

Um processo executando o algoritmo SmartGossip define os destinatários de cada mensagem em difusão utilizando uma abordagem que leva em conta feromônios que influenciam a escolha probabilística de destinos. O SmartGossip utiliza feromônios da forma contrária à tradicional, na qual os feromônios atraem indivíduos. Um processo executando o algoritmo SmartGossip associa a cada um de seus enlaces um depósito de feromônios. Inicialmente, é nula a concentração de feromônios de cada enlace $e_{ij} \in E$ conectando o processo i com o processo j, chamados aqui de "vizinhos". Ao transmitir uma mensagem para o processo j, o processo i atualiza a concentração de feromônios C_{ij} associada ao enlace e_{ij}, como mostra a expressão 1.

$$C_{ij} \leftarrow C_{ij} + 1 \tag{1}$$

Os feromônios de cada enlace "evaporam" de rodada para rodada, conforme uma taxa de evaporação $0 \leq \rho < 1$. Uma nova rodada de disseminação ocorre a cada vez que um processo tem que encaminhar uma mensagem da difusão em curso. A concentração de feromônios em uma rodada r é calculada tendo em vista o número de rodadas que se passou desde que o processo recebeu a mensagem a última vez, como mostra a expressão 2.

$$C_{ij} = C_{ij} \cdot (1 - \rho) \tag{2}$$

O próximo destino de uma mensagem em uma rodada é escolhido probabilisticamente, com influência da concentração de cada depósito de feromônios. Seja $\alpha > 0$ a intensidade dos feromônios depositados, $|N_i|$ o número de vizinhos de i, seja k um vizinho de i, a probabilidade P_{ij} do vizinho j ser escolhido para encaminhar uma mensagem é dada pela expressão 3. O processo i calcula as probabilidades $\forall j \in N_i$.

$$P_{ij} = \frac{(C_{ij} + 1)^{-\alpha}}{\sum_{\forall k \in N_i} (C_{ik} + 1)^{-\alpha}} \tag{3}$$

Para garantir que a quantidade de mensagens seja adequada tanto para acelerar a difusão, como para diminuir o número de mensagens redundantes, é feito um controle populacional sobre as mensagens. As concentrações de feromônio também são utilizadas para este propósito, com um limite superior L_{max} definido. Cada processo calcula a concentração local total de feromônios, consistindo na soma dos depósitos de todos os seus enlaces, de acordo com a expressão 4. Quando o valor supera L_{max}, novas mensagens que o processo recebe para encaminhar na difusão em curso são descartadas.

$$C_i = \sum_{\forall k \in N_i} C_{ik} \tag{4}$$

Como um número maior de mensagens tenderá a passar por processos com maior número de enlaces, o limite superior L_{max} varia com o grau. Dadas as constantes de peso da vizinhança $\delta \geq 0$ e limite superior máximo $\gamma_{max} > 0$, a concentração C_i de feromônios é comparada com os valores de limite ajustados para o processo i com base no tamanho de sua vizinhança, definidos na expressão 5.

$$L_{max}(t) = \gamma_{max} \cdot |N_i|^{\delta} \tag{5}$$

O Algoritmo 3 apresenta o pseudocódigo do SmartGossip. As primitivas utilizadas são aquelas utilizadas para o algoritmo Gossip da Seção 2.

Algoritmo 2 O Algoritmo SmartGossip (executado pelo processo i)

1: ***Init:***
2: $delivered \leftarrow \emptyset$
3: $N_i \leftarrow$ set of neighbors
4: **for all** $j \in N_i$ **do**
5: $pheromones[j] \leftarrow 0$
6: **end for**

7: ***Upon broadcast(msg):***
8: $S \leftarrow$ random subset of f processes of N_i
9: **for all** $j \in S$ **do**
10: *send*$(j, i, msg, maxRounds - 1)$
11: $pheromones[j] \leftarrow pheromones[j] + 1$
12: **end for**

13: ***Upon receive(src, i, msg, round):***
14: **for all** $j \in N_i$ **do**
15: $pheromones[j] \leftarrow pheromones[j] \cdot (1 - \rho)^{|rounds\ elapsed\ since\ last\ seen|}$
16: **end for**
17: $pheromones[src] \leftarrow pheromones[src] + 1$
18: **if** $msg \notin delivered$ **then**
19: $delivered \leftarrow delivered \cup \{msg\}$
20: *deliver(msg)*
21: **end if**
22: $C_i \leftarrow \sum_{\forall k \in N_i} C_{ik}$
23: **if** $round > 0$ **and** $C_i < L_{max}$ **then**
24: $S \leftarrow$ random subset of f processes of N_i
25: **for all** $j \in S$ **do**
26: *send*$(j, i, msg, round - 1)$
27: $pheromones[j] \leftarrow pheromones[j] + 1$
28: **end for**
29: **end if**

Todo processo i do sistema distribuído executa o Algoritmo 3. Ao iniciar a execução, o processo i inicializa o conjunto de mensagens entregues (*delivered*). Em seguida, inicializa o conjunto de depósitos de feromônio (*pheromones*) de cada enlace existente entre i e j, $\forall j \in N_i$. Ao executar a primitiva *broadcast(msg)*, o processo i escolhe um subconjunto aleatório S de $min\{f, |N_i|\}$ processos de sua vizinhança N_i e transmite a mensagem *msg* para todo $j \in S$, incluindo o *payload*, o identificador de origem, o identificador de destino e o identificador da rodada. A rodada inicial aqui também é dada por $maxRounds - 1$, com $maxRounds > 0$ sendo um parâmetro definido antes da execução do algoritmo. Após transmitir as mensagens, o processo i incrementa seu contador de feromônios local para cada um dos enlaces sobre o qual a mensagem foi

transmitida.

Ao executar a primitiva *receive(src, dst, msg, round)*, o processo i atualiza o contador dos feromônios, aplicando a evaporação conforme descrita na Equação 2. Em seguida, atualiza também o valor do depósito de feromônio para o enlace utilizado pela mensagem recebida. O processo então verifica se a mensagem recebida de *src* já foi entregue e a armazena caso ainda não tenha sido. Para decidir se a mensagem recebida será retransmitida, o processo i verifica se o número máximo de rodadas ainda não foi atingido e se o valor da concentração total de feromônios C_i (Equação 4) não ultrapassou o limite máximo L_{max}. Caso o valor de C_i esteja acima do limite superior, a mensagem é destruída. Em caso negativo, a execução prossegue, com mais $min\{f, |N_i|\}$ processos sendo escolhidos como destino da mensagem a ser retransmitida. Após retransmitir a mensagem, o processo i incrementa o seu contador de feromônios local para cada um dos enlaces sobre o qual a mensagem foi retransmitida.

Ambas as métricas de latência e de número de mensagens enviadas são influenciadas diretamente por diversos dos parâmetros, e portanto estes devem ser ajustados apropriadamente. Primeiramente, o valor da constante γ_{max} utilizada na expressão 5 influencia no controle populacional, isto é, o número de mensagens enviadas por cada processo. Caso o valor de γ_{max} seja muito pequeno, as mensagens serão destruídas com maior frequência, reduzindo o número de mensagens transmitidas, gerando impacto na latência. Inversamente, se γ_{max} é muito elevada, o sistema pode ser inundado com mensagens redundantes desnecessárias. A constante δ também influencia diretamente no controle populacional, visto que δ é dado como o peso da vizinhança de i no cálculo do limite L_{max}, justamente utilizado para controle populacional. O valor da taxa de evaporação ρ também influencia no desempenho do algoritmo, tanto no controle populacional - com a concentração de feromônios ultrapassando os limites mais frequentemente - como na escolha dos próximos destinos, afetando o número de mensagens enviadas e a latência. O algoritmo SmartGossip é validado e avaliado experimentalmente na próxima sessão.

4. Simulação e Resultados

O desempenho do algoritmo proposto foi avaliado através da execução de um simulador desenvolvido através do framework OMNeT++, na linguagem C++. O simulador foi executado em uma máquina baseada em um processador Intel(R) Core(TM) i5-8300H CPU @ 2,30GHz com 4 cores, tendo 16GB RAM e 2TB de memória secundária SSD. Além do algoritmo SmartGossip, também foram simulados os algoritmos Flooding e Gossip, que servem como base de comparação para os resultados obtidos pelo algoritmo proposto. Nesta seção são apresentados alguns dos resultados representativos obtidos.

Inicialmente, o simulador implementado gera uma topologia aleatória de com $N > 0$ vértices, que representam os processos do sistema distribuído. O número de enlaces é influenciado por um parâmetro de conectividade $0 < C \leq 1$. O parâmetro C reflete a conectividade da topologia gerada da seguinte maneira. Quando, por exemplo, $C = 1,0$, a conectividade é 100%, ou seja há uma aresta conectando cada um dos processos a todos os demais. De forma análoga, quando $C = 0,5$ a probabilidade de haver uma aresta entre dois processos quaisquer é de 50%. Em outras palavras: C reflete a probabilidade de dois processos quaisquer serem adjacentes. No artigo são apresentados resultados para simulações executadas N igual a 64, 512 e 1024. Para cada valor de N,

são mostrados resultados para topologias com conectividades $C = 0,5, 0,7$ e $1,0$. Cada combinação de configurações de número de processos e nível de conectividade foi executada 30 vezes. Todos os algoritmos (Flooding, Gossip e SmartGossip) foram executados por tantas rodadas quantas foram necessárias para que todos os processos entreguem a mensagem.

O algoritmo Gossip foi simulado utilizando *fanout* $= log_{10}(N)$, valor que tem sido tradicionalmente apontado na literatura como adequado [Eugster et al. 2004]. No trabalho também foi realizada uma avaliação experimental em que foram utilizados diversos outros valores/funções para o *fanout* e $log_{10}(N)$ efetivamente se mostrou o mais apropriado.

Os valores para os parâmetros do algoritmo SmartGossip foram definidos experimentalmente, a partir de diversas execuções das simulações, para obtenção dos valores mais adequados. Em particular, foi utilizada a potência dos feromônios (α) = 8, taxa de evaporação (ρ) = 0,1, peso da vizinhança (δ) = 0,5 e o mesmo *fanout* do Gossip, isto é *fanout* $= log_{10}(N)$. Valores diferentes para o parâmetro de limite superior γ_{max} foram definidos para cada tamanho de sistema N. Para $N = 64$ $\gamma_{max} = 1,3$; para N = 512, $\gamma_{max} = 0,8$ e, finalmente, para N = 1024, $\gamma_{max} = 0,5$.

As métricas utilizadas foram o número de mensagens utilizadas e o número de rodadas executadas até a difusão acabar (latência). Os resultados são apresentados a seguir.

Flooding N=64								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	1277,80	1027	1511	141,32	3,37	3	4	0,49
0,7	2005,60	1881	2095	53,94	2,17	2	3	0,38
1,0	3969,00	3969	3969	0,00	1,00	1	1	0,00
Gossip N=64								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	458,80	254	510	104,15	7,80	7	8	0,41
0,7	424,67	254	510	122,74	7,67	7	8	0,48
1,0	450,27	254	510	110,13	7,60	7	8	0,50
SmartGossip N=64								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	310,20	228	376	51,93	7,77	7	9	0,57
0,7	313,93	234	426	74,78	7,57	7	9	0,63
1,0	323,60	234	502	90,92	7,47	7	9	0,57

Tabela 1. Número de mensagens e rodadas para N=64.

A Tabela 1 mostra os resultados obtidos para a execução dos três algoritmos quando o sistema consiste de 64 processos, com conectividade $C = 0,5, 0,7$ e 1. O Flooding, para $C = 0,5$ gerou em média 1277,80 mensagens trocadas entre os processos, sendo no melhor caso 1027 mensagens e no pior caso 1511 mensagens. Na medida em que a conectividade C aumenta, podemos observar que o número de mensagens também

aumenta. Assim, para $C = 1$ o número de mensagens chega ao pico de 3969. O desvio padrão calculado para $C = 0,5$ foi de 141,32 mensagens, depois ora aumenta ora diminui, até chegar a zero quando $C = 1$, conforme esperado, pois a mesma topologia (grafo completo) é gerada em todos os casos. O número de rodadas foi diretamente influenciado pela conectividade. Para $C = 0,5$, a média foi de 3,37 rodadas, mas conforme a conectividade aumenta, este valor diminui, até que para $C = 1,0$ apenas uma rodada única é necessária, visto que todos processos conseguem comunicar-se com os demais.

Na execução do Gossip, para $C = 0,5$, a média do número de mensagens foi de 458,80, valor que cai para 424,67 para $C = 0,7$, depois subindo um pouco para 450,27 tanto para $C = 1,0$. Destaca-se que para valores maiores de N não houve resultados idênticos para conectividades diferentes, como ocorreu neste caso. O número de rodadas se manteve praticamente constante entre 7 e 8 na média.

Chegando ao SmartGossip, em todos os casos os números de mensagens foram menores que os correspondentes para o Gossip, que por sua vez, já foram menores que os valores obtidos para o Flooding. Para $C = 0,5$, a redução do número de mensagens do SmartGossip para o Gossip é de 32,4%, enquanto que a redução do SmartGossip para o Flooding é de 75,7%. Para $C = 1,0$, a redução foi de 28,1% e 91,8%, respectivamente. Por outro lado, os números de rodadas do SmartGossip são semelhantes aos do Gossip, e maiores que os do Flooding, que completa a difusão em 1 única rodada quando $C = 1,0$.

Flooding N=512								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	81449,40	79593	83707	1182,74	2,63	2	3	0,49
0,7	128620,47	127127	129947	919,44	2,10	2	3	0,31
1,0	261121,00	261121	261121	0,00	1,00	1	1	0,00
Gossip N=512								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	97207	32766	262142	67472,91	10,27	9	12	0,64
0,7	84648,67	16382	524286	96025,58	9,93	9	12	0,69
1,0	4640,13	4094	8190	1416,18	7,43	7	8	0,50
SmartGossip N=512								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	4289	3688	5354	676,73	11,43	11	13	0,57
0,7	4135,33	3848	5722	628,34	11,13	11	12	0,35
1,0	4423,07	3946	6950	960,56	11,20	11	13	0,48

Tabela 2. Número de mensagens e rodadas para N=512.

A Tabela 2 mostra os resultados obtidos para $N = 512$. No caso do Flooding é possível observar o mesmo padrão de aumento de número de mensagens influenciado pela conectividade. Para $C = 0,5$, temos que a média do número de mensagens é de 81449,40. Para $C = 1,0$, o número de mensagens foi 261121. O número de rodadas também se manteve no mesmo nível dos tamanhos de sistema anteriores. No caso do Gossip, nos níveis mais baixos de conectividade, o Gossip continua utilizando em média um número maior de mensagens quando comparado ao Flooding para completar o difusão. Para $C =$

$0,5$, em média 97207 mensagens foram utilizadas. Para $C = 1,0$, o número médio de mensagens foi 4640,13. O SmartGossip, para $C = 0,5$, apresenta redução do número de mensagens utilizadas quando comparado com o Gossip de 95,5%, enquanto que a redução do SmartGossip para o Flooding é de 94,5%. Para $C = 1,0$, a redução foi de 4,67% e 98,3%, respectivamente.

Flooding N=1024								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	324801,93	321103	329511	2270,88	2,73	2	3	0,45
0,7	510177,07	504443	515235	2495,76	2,10	2	3	0,31
1,0	1046529,00	1046529	1046529	0,00	1,00	1	1	0,00
Gossip N=1024								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	667243,60	88572	2391438	418509,25	9,17	8	10	0,53
0,7	460579,87	88572	2391483	444635,09	9,17	8	10	0,46
1,0	10496,10	9840	29523	3593,61	7,00	7	7	0,00
SmartGossip N=1024								
	Número de mensagens				Número de rodadas			
Conectividade	Média	Menor	Maior	σ	Média	Menor	Maior	σ
0,5	9160,40	8667	12438	1095,49	8,10	8	9	0,31
0,7	9938,80	9126	14556	1803,76	8,13	8	9	0,35
1,0	10208,30	9450	16425	2097,82	8,10	8	9	0,31

Tabela 3. Número de mensagens e rodadas para N=1024.

Finalmente, a Tabela 3 mostra os resultados obtidos $N = 1024$. No caso do Flooding, quando aumenta a conectividade C, aumenta o número de mensagens, com média igual a 324801,93 para $C = 0,5$, subindo para 1046529 – mais de 1 milhão de mensagens, portanto – quando $C = 1,0$. O número de rodadas continua muito reduzido, de 1 até 3, no máximo, considerando todos os valores da conectividade. O Gossip com $N = 1024$ e $C = 0,5$ gera número de mensagens na média (667243,6) maior que o correspondente do Flooding (324801,93). Para os outros valores da conectividade C o Gossip precisou de menos mensagens, com uma redução de 1046529 (Flooding) para 10496,1 (Gossip) quando a conectividade é $C = 1,0$. O desvio padrão do Gossip continua muito elevado neste caso. O número de rodadas necessárias para completar a difusão varia de 7 a 10, com baixa dispersão. O SmartGossip apresenta número de mensagens na média expressivamente menor que o do Gossip para todos os valores da conectividade, com os desvios padrão obtidos bem pequenos no caso do SmartGossip. Para $C = 0,5$, a redução do número de mensagens do SmartGossip para o Gossip é de 98,6%, enquanto que a redução do SmartGossip para o Flooding é de 97,2%. Para $C = 1,0$, a redução foi de 2,7% e 99,2%, respectivamente. Por outro lado, os números de rodadas do SmartGossip (8 a 9) são semelhantes aos do Gossip (7 a 10), que já são maiores que os do Flooding, que completa a difusão em 1 única rodada quando $C = 1,0$ e em 2 ou 3 rodadas no máximo para outros valores de conectividade.

A Figura 1 apresenta uma comparação entre o número médio de mensagens utilizadas por cada algoritmo para cada tamanho de sistema e conectividade. No gráfico, cada tamanho de marcador utilizado representa um valor diferente de N, com cada cor representando um algoritmo, conforme a legenda. Como é possível observar, para toda

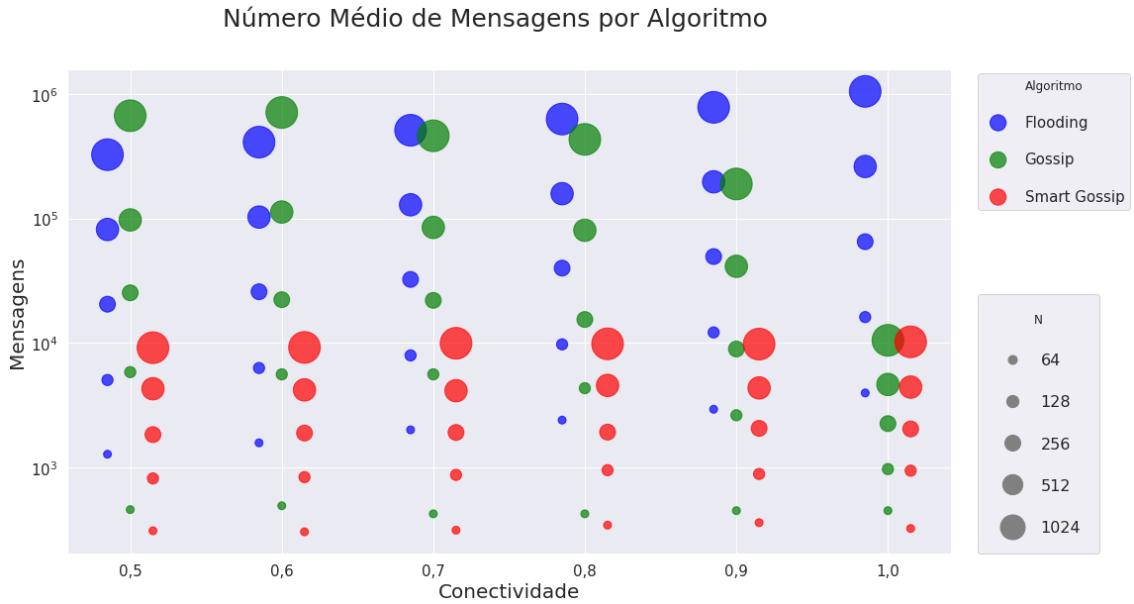

Figura 1. Número médio de mensagens utilizadas por cada algoritmo para cada combinação de tamanho de sistema N e conectividade C.

combinação de valores para N e C, o número médio de mensagens utilizadas pelo algoritmo SmartGossip foi inferior ao número médio de mensagens utilizadas pelos dois algoritmos clássicos Flooding e Gossip. Outro ponto importante a ser destacado é que o número médio de mensagens utilizadas pelo algoritmo SmartGossip para cada tamanho N apresenta baixa dispersão para as diferentes conectividades.

A Figura 2 apresenta a mesma comparação para o número médio de rodadas necessárias para completar a difusão. O algoritmo Flooding garante o melhor resultado no quesito de latência com sua estratégia de inundação, dado o número elevado de mensagens. Para este algoritmo na conectividade $C = 1,0$, basta apenas uma rodada. Para o algoritmo SmartGossip, é possível notar que o número médio de rodadas utilizadas é inferior aos níveis apresentados pelo algoritmo Gossip em certos casos, como para $N = 1024$ em todas conectividades $0,5 \leq C < 1,0$. Em outros casos, como para $N = 512$, o algoritmo SmartGossip necessitou de uma ou duas rodadas a mais. De modo geral, os resultados obtidos pelo SmartGossip no quesito de latência são equiparáveis aos obtidos pelo algoritmo Gossip.

4.1. Discussão dos Resultados

Acreditamos que os resultados apresentam com clareza que o algoritmo SmartGossip representa uma contribuição, em comparação com as outras duas estratégias para difusão de mensagens, Flooding e Gossip. Para *todos* os valores de N e *todas* as conectividades C o SmartGossip resultou em número médio de mensagens utilizadas menor que o dos outros dois algoritmos. Além disso, deve ser destacado que, diferente do Gossip, o SmartGossip manteve uma baixa dispersão em todos os casos. Por exemplo, a média do número de mensagens utilizadas pelo SmartGossip com $N = 1024$ variou de 9160,4 ($C = 0,5$) até 10208,30 ($C = 1,0$). Por outro lado, no mesmo caso, o Gossip variou de 10496,10 ($C = 1,0$) até 708583,83 ($C = 0,5$). Para todos os casos o Gossip apresentou desvio padrão do número de mensagens utilizadas muito significativo, enquanto que o

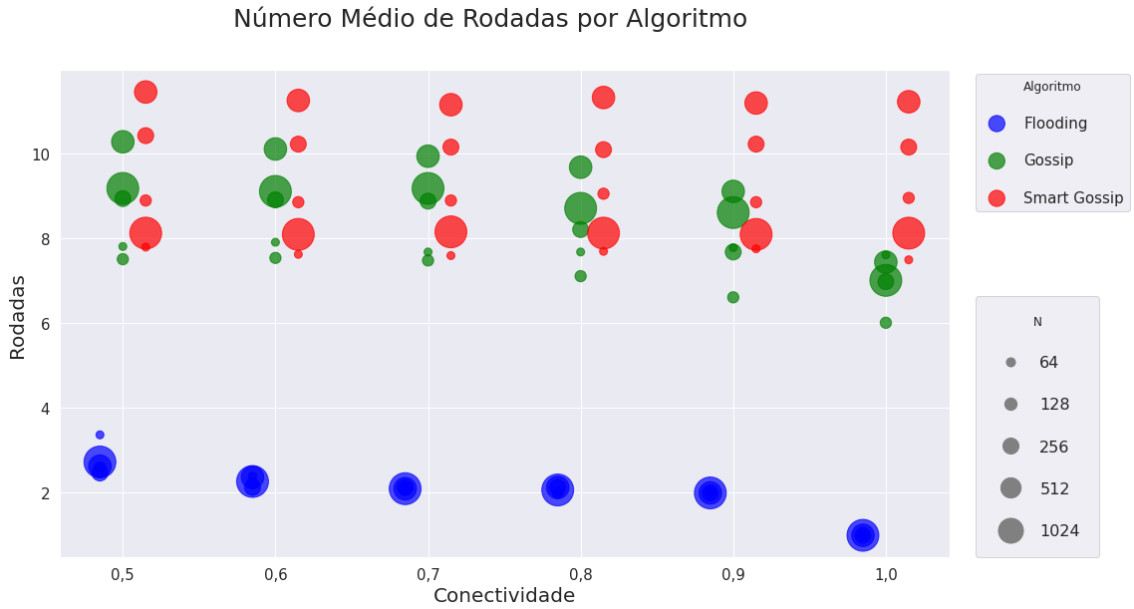

Figura 2. Número médio de rodadas necessárias para finalizar a difusão para cada algoritmo e para cada combinação de tamanho de sistema N e conectividade C.

SmartGossip apresentou desvio padrão pequenino. Por outro lado, apesar de poderem ser considerados equiparáveis em termos do número de rodadas para completar a difusão, o SmartGossip em alguns casos necessitou de 1 ou 2 rodadas a mais que o Gossip. Neste quesito o Flooding é muito melhor, terminando a difusão sempre com menor número de rodadas, mas empregando quantidades significativamente maiores de mensagens.

Outro ponto a ser destacado é o comportamento do SmartGossip quando a conectividade varia: o desvio padrão varia pouco entre os diferentes valores de C, com o número médio de mensagens utilizadas sofrendo pouca influência da conectividade. No caso do Gossip, é possível inferir que a conectividade tem forte impacto no desempenho do algoritmo. Quanto menor a conectividade, maior o número de mensagens e a latência do Gossip para completar a difusão. Este fato fica em particular evidência quando a conectividade passa para 1,0, o que facilita para o Gossip atingir todos os processos. Destaca-se que padrões semelhantes de variação do número médio de mensagens são encontrados também para outros valores de N, o desvio padrão do número de mensagens do Gossip aumenta conforme o tamanho do sistema aumenta.

5. Conclusão

Esse trabalho propôs o algoritmo SmartGossip para difusão probabilística e inteligente de mensagens em sistemas de topologia arbitrária. O SmartGossip é inspirado por técnicas de colônia de formigas, mais especificamente pelo conceito de comunicação por estigmergia. As decisões sobre o encaminhamento de mensagens são feitas levando em consideração os níveis de feromônios dos enlaces para os vizinhos. O algoritmo foi implementado utilizando o simulador OMNeT++ e resultados de comparação com os algoritmos distribuídos clássicos de difusão de mensagens Flooding e Gossip mostram que o SmartGossip apresentou, em média, um número consideravelmente menor de mensagens utilizadas em todos os cenários testados, com número de rodadas equiparável ao Gossip.

Trabalhos futuros incluem investigar o problema da disseminação inteligente utilizando múltiplos enxames em paralelo [De Campos Jr et al. 2019]. Também está previsto executar o algoritmo sobre outros tipos de topologia (e.g. anel, hipercubo, torus, entre outras), assim como sobre topologias dinâmicas, que mudam ao longo do tempo [De Campos Jr et al. 2013]. Algoritmos genéticos, que já foram usados com sucesso no diagnóstico que inclui a disseminação de informações sobre falhas [Nassu et al. 2005, Duarte Jr et al. 2010] também devem ser investigados para a difusão inteligente. Adicionalmente, a validação do algoritmo em uma *testbed* sobre a Internet também está planejada como trabalho futuro.

Referências

Banzi, A. S., Jr., E. P. D., and Pozo, A. R. (2011). An approach based on swarm intelligence for event dissemination in dynamic networks. In *IEEE SRDS*, pages 121–126.

Cachin, C., Guerraoui, R., and Rodrigues, L. (2011). *Introduction to Reliable and Secure Distributed Programming (2nd Ed)*. Springer.

De Campos Jr, A., Pozo, A. T., and Duarte Jr, E. P. (2013). Evaluation of asynchronous multi-swarm particle optimization on several topologies. *Concurrency and Computation: Practice and Experience*, 25(8):1057–1071.

De Campos Jr, A., Pozo, A. T., and Duarte Jr, E. P. (2019). Parallel multi-swarm pso strategies for solving many objective optimization problems. *Journal of Parallel and Distributed Computing*, 126:13–33.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

Dorigo, M., Bonabeau, E., and Theraulaz, G. (2000). Ant algorithms and stigmergy. *Future Generation Computer Systems*, 16(8):851–871.

Duarte Jr, E. and Mattos, G. (2000). Diagnóstico em redes de topologia arbitrária: Um algoritmo baseado em inundação de mensagens. In *II Workshop de Testes e Tolerância a Falhas (WTF)*, pages 82–87. SBC.

Duarte Jr, E. P., Pozo, A. T., and Nassu, B. T. (2010). Fault diagnosis of multiprocessor systems based on genetic and estimation of distribution algorithms: a performance evaluation. *International Journal on Artificial Intelligence Tools*, 19(01):1–18.

Eugster, P., Guerraoui, R., Kermarrec, A.-M., and Massoulié, L. (2004). Epidemic information dissemination in distributed systems. *IEEE Computer*, 37:60–67.

Muteeh, A., Sardaraz, M., and Tahir, M. (2021). Mrlba: multi-resource load balancing algorithm for cloud computing using ant colony optimization. *Cluster Computing*, 24(4):3135–3145.

Nassu, B. T., Duarte Jr, E. P., and Ramirez Pozo, A. T. (2005). A comparison of evolutionary algorithms for system-level diagnosis. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 2053–2060.

Nassu, B. T., Nanya, T., and Duarte, E. P. (2007). Topology discovery in dynamic and decentralized networks with mobile agents and swarm intelligence. In *Int. Conf. Intellinget Sys. Design and Apps. (ISDA)*, pages 685–690.

Ramamoorthy, R. and Thangavelu, M. (2021). An enhanced hybrid ant colony optimization routing protocol for vehicular ad-hoc networks. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–32.

Rodrigues, L., Arantes, L., and Duarte, E. (2014). An autonomic implementation of reliable broadcast based on dynamic spanning trees. In *EDCC*, pages 1–12. IEEE.