PUC

INF2056 – Algoritmos Distribuídos
Prof. Markus Endler

Relatório T1

Rafael Pereira de Oliveira
Matrícula 1512356

Departamento de Informática

Maio, 2015.

Sumário

1. Implementação do algoritmo Invitation – Garcia-Molina-82

É apresentado aqui a implementação e testes do algoritmo de eleição *Invitation* [1]. O algoritmo *Invitation* é uma variante do algoritmo de *Bully* para sistemas assíncronos e que trata a possibilidade de ocorrerem partições na rede.

Foi realizada uma implementação para o *framework* Sinalgo [2]. Após a implementação, foram realizados testes para avaliar o desempenho do algoritmo e sua estabilidade quando exposto a falhas de rede, variando a probabilidade de falhas, e a quantidade de partições de rede.

O algoritmo foi implementado utilizando o Sinalgo [2]. Todo o código fonte, assim como as notas do versionamento podem ser baixados/acessados do repositório GIT público no endereço: https://github.com/torraodocerrado/leader_election_sinalgo

Durante o desenvolvimento, foram criados os seguintes modelos:

Modelo	Descrição
LakeAvoidRandomDistribution	Este modelo é responsável por determinar o movimento de nós móveis. O movimento gerado por este modelo permite que se crie mapas durante a simulação com espaços em que os nós não acessam. Apesar de não ter sido utilizado nos testes, ele foi projetado para observar grupos distintos, que esporadicamente trocam componentes de grupos.
LakeAvoid	Responsável pela instanciação dos nós. Este modelo determina uma posição randômica do nó no mapa respeitando as áreas em que nós não podem acessar. Dessa forma, um nó não é instanciado em uma área que ele não poderá se locomover.
Mensagens: • **AYCoord** • **AYC_answer** • **AYThere** • **AYThere_answer** • **Invitation** • **Accept** • **Accept_answer** • **Ready** • **Ready_answer**	Todas as mensagens definidas pelo algoritmo Invitation foram implementadas como extensões da classe “Message” do Sinalgo.
LossyDelivery	Uma extensão do modelo *ReliabilityModel*. Aqui foram implementados os métodos para partição dos nós e para a política de perda de mensagens.
LeaderNode	A classe principal utilizada para a implementação do algoritmo Invitation. É uma extensão da classe Node e contém todos os passos implementados em métodos correspondentes.

2. Avaliação

Este capítulo é dedicado à experimentação da implementação realizada do algoritmo Invitation [1].

2.1. Ambiente onde não há perda de mensagens

Nesta seção, foi descrito os testes do algoritmo Invitation, em um ambiente onde não há perda de mensagens. O intuito é entender melhor, qual o comportamento esperado do algoritmo em um ambiente "ideal", apensar que como explanado em sala de aula, ambientes "ideais" são raros e falhas são inerentes a sistemas distribuídos.

2.1.1. Convergência para diferentes números de nós

Realizamos aqui um teste para avaliar a quantidade de mensagens trocadas, e em quantos passos o algoritmo consegue sair de seu estado inicial (onde todos os nós são coordenadores apenas de si mesmos). A quantidade de nós testadas cresceu geometricamente iniciando-se em 2 até 512 nós.

Quantidade de mensagens trocadas

Durante os testes para avaliar a convergência, foi avaliada a quantidade de mensagens necessárias para a convergência do algoritmo, e a quantidade de passos necessários para alcançar o fim da eleição de líder. O teste mostrou os seguintes resultados:

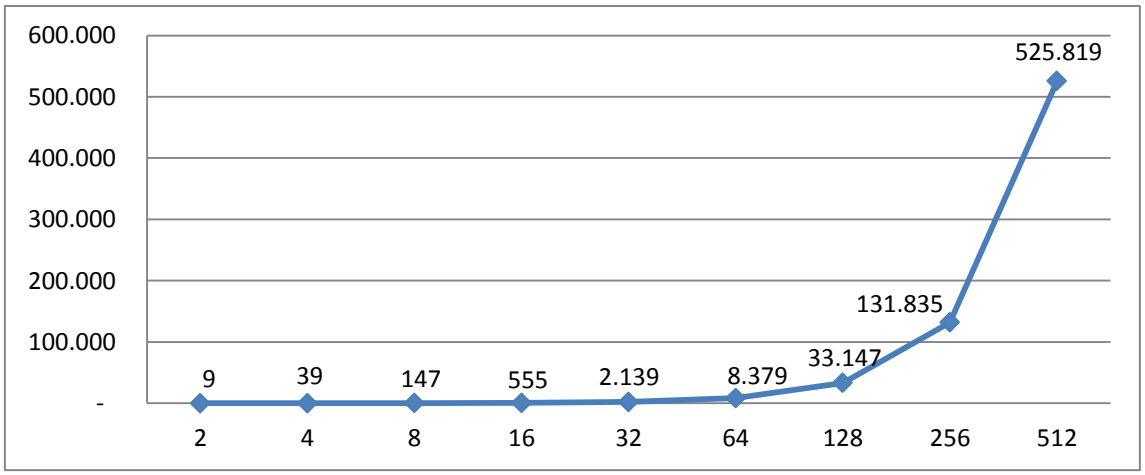


Gráfico 1 - Números de mensagens trocadas durante a fase de eleição inicial

A quantidade de mensagens trocadas para esta implementação e exibidas no Gráfico 1, foram extraídas da execução da ferramenta. Através dela, do algoritmo e da implementação, pode-se calcular uma função que prevê a quantidade exata de mensagens trocadas para cada fase do algoritmo, onde N é a quantidade de nós em que todos podem se comunicar com todos (grafo completo):

Fase	Tipo de Mensagem	Quantidade de mensagens	Descrição
Check Members	AYCoord	$O(N^2-N)$	Considerando o estado inicial como pior caso, onde todos os nós são coordenadores de si mesmos e não coordenam nenhum outro nó.
Check Members	AYC_answer	$O(N^2-N)$	Também considerando o pior caso, todas as mensagens AYCoord serão respondidas com uma AYC_answer.
Merge	Invitation	O(N-1)	O coordenador que tiver maior prioridade, irá iniciar o convite para "*merge*" e irá enviar um convite para todos os outros coordenadores, que no pior caso, será todos os outros nós.
Merge	Accept	O(N-1)	Para cada mensagem Invitation, será enviada uma mensagem de Accept para o coordenador que iniciou o *merge*.
Merge	Accept_answer	O(N-1)	Para cada mensagem Accept, o coordenador que iniciou o merge irá responder um *Accept_answer.*
Reorganize	Ready	O(N-1)	Para iniciar a fase de Reorganize, o coordenador candidato irá enviar uma mensagem de *Ready* para todos os demais que irá coordenar.
Reorganize	Ready_answer	O(N-1)	Todos os nós que receberem a mensagem de Ready irão responder com uma mensagem de Ready_answer

Tabela 1 - Cálculo de mensagens trocadas durante a primeira eleição do algoritmo Invitation

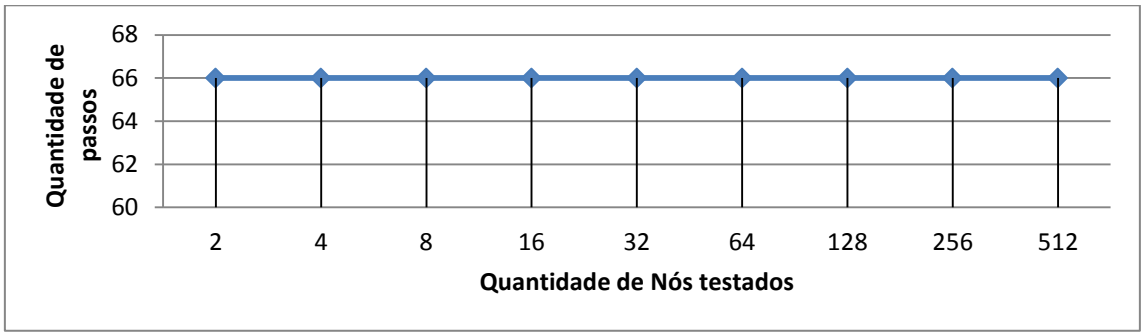
Dessa forma, durante a primeira eleição, considerando o pior caso, serão trocadas:

$$(N^2-N) + (N^2-N) + (N-1) + (N-1) + (N-1) + (N-1) + (N-1) = \mathbf{2N^2 + 3N - 5}$$

Exemplo (4 nós): $2(4)^2 + 3(4) - 5 = 39$ mensagens, que corresponde às mensagens trocadas pelo teste com 4 nós (Gráfico 1) pela implementação que acompanha este trabalho. Este cálculo nos mostra, que durante a eleição inicial, considerando que todos os nós estão interligados com todos os outros nós, a quantidade de mensagens trocadas possui complexidade exponencial de $O(N^2)$ para este algoritmo.

Quantidade de passos necessários para a eleição

Como não há perda de mensagens, todos os testes tiveram a mesma quantidade de passos:



Todas as quantidades de nós testadas necessitaram de 66 passos para realizar a eleição inicial.

Segundo o algoritmo de Garcia-Molina, ao ser instanciado cada nó é o coordenador de si mesmo. Também é definido o passo onde cada coordenador verifica se há outros coordenadores acessíveis (*Check_members*()). Essa checagem é realizada através do envio de uma mensagem AYCoord para todos os nós alcançáveis. Essa checagem foi implementada para acontecer a cada 50 passos, logo, no passo 50 todos os coordenadores enviam uma mensagem AYCoord (como podemos ver na linha 1 e 2 do log da Tabela 1). Seguindo isso, ocorrem as trocas de mensagens para a eleição do coordenador entre os passos 51 até 66 (Tabela 1). Logo, como não há perda de mensagens, e todos podem enviar mensagens dentro de um mesmo passo, independente da quantidade de nós, a quantidade de passos para a eleição inicial será de 66 passos para esta implementação. Vale ressaltar, que este número está diretamente ligado ao intervalo de checagem (*Check_members*()) de outros coordenadores.

1	`STEP 51- Node 1: Received MSG AYCoord by 2`
2	`STEP 51- Node 2: Received MSG AYCoord by 1`
3	`STEP 52- Node 1: Received MSG AYC_answer by 2 said your coord is Node(ID=2)`
4	`STEP 52- Node 1: Setup timerToMerge = STEP 72`
5	`STEP 52- Node 2: Received MSG AYC_answer by 1 said your is Node(ID=1)`
6	`STEP 52- Node 2: SetUP timerToMerge 62`
7	`STEP 62- Node 2: Start merge`
8	`STEP 63- Node 1: Received MSG Invitation by 2`
9	`STEP 64- Node 2: Received MSG Accept by 1`
10	`STEP 64- Node 2: Start Reorganizing`
11	`STEP 65- Node 1: Received MSG Accept_answer by 2`
12	`STEP 65- Node 1: Received MSG Ready by 2`
13	`STEP 66- Node 2: Received MSG Ready_answer by 1`

Tabela 2 - Log dos primeiros passos do algoritmo Invitation (Garcia-Molina)

Após a eleição inicial, como não há perda de mensagens neste teste nem nenhum outro distúrbio nos nós, ele ficará indefinidamente no estado ótimo (um coordenador ativo e aceito por todos os outros nós ativos).

2.2. Ambiente com perda de mensagens

Durante a segunda fase dos testes, foram inseridas no sistema falhas na entrega das mensagens. Para isso, foi implementado uma extensão do modelo *ReliabilityModel* que foi chamado de *LossyDelivery.* A implementação foi realizada para criar falhas intermitentes na entrega das mensagens baseadas em uma probabilidade de falha, e também para simular a criação de partições controladas entre os nós.

A criação de partições, por parte do modelo *LossyDelivery*, é criada através de uma função que divide os nós por seu identificador único (ID) em N grupos, de acordo com um parâmetro informado no início da execução. Através da divisão, o modelo consegue avaliar se o remetente e o destinatário de uma mensagem estão dentro do mesmo grupo. Se estiverem, o modelo garante sempre a entrega da mensagem, se não estiverem, ela utiliza a probabilidade de entrega de uma mensagem (também parâmetro) para decidir se haverá ou não um particionamento da rede.

O particionamento entre a quantidade de grupos de nós perdura por uma quantidade determinada de passos do Sinalgo. Para os testes, foram considerados que a falha de rede que durava 500 passos. O valor 500 é um valor escolhido arbitrariamente, que de acordo com a observação do algoritmo, era uma quantidade de passos que permitia as partições executarem todas as fases de eleição de um novo líder e permanecerem estáveis por algum tempo. Após essa quantidade de 500 passos, a comunicação entre todos os nós é reestabelecida, e então os grupos realizam uma nova eleição e o *merge*.

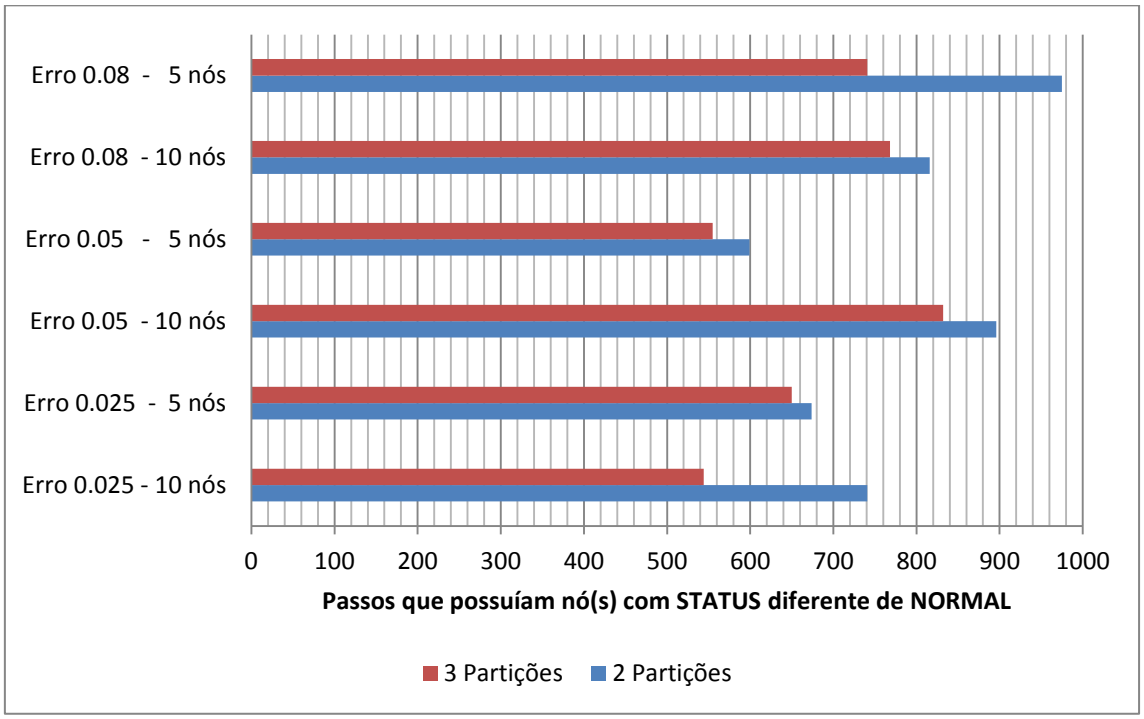


Gráfico 2 - Quantidade de passos que possuíam pelo menos 1 (um) nó com STATUS diferente de NORMAL - (em 10.000 passos executados)

As probabilidades de interrupção da comunicação testadas foram 0,025; 0,05 e 0,08 durante um período de 10.000 passos. O Gráfico 2 um dos testes realizados, onde se contabilizou a quantidade de passos que possuía pelo menos 1 (um) nó que não tinha o status igual a "NORMAL". Considerou-se que o ambiente ótimo fosse todos os nós com status normal, e que qualquer coisa diferente disso, seria uma instabilidade.

Os resultados mostraram (Gráfico 2) que, como esperado, quanto maior a probabilidade de falhas, maior a instabilidade dos nós. A menor quantidade de falhas foi encontrada com a probabilidade de erro 0.025. Outra constatação possível de se observar pelo gráfico é que quanto maior a quantidade de partições, mais rápido o algoritmo de recupera de uma falha. Todas as execuções, independente da probabilidade de falha, três partições de nós tiveram resultados melhores que quando os nós eram divididos em apenas duas partições.

O Gráfico 2 abaixo, já mostra a quantidade de mudanças de status para algo diferente de "NORMAL". Tentou-se mostrar aqui, não a quantidade de passos instáveis (Gráfico 2), mas o somatório de nós que mudaram para algum status diferente de "NORMAL".

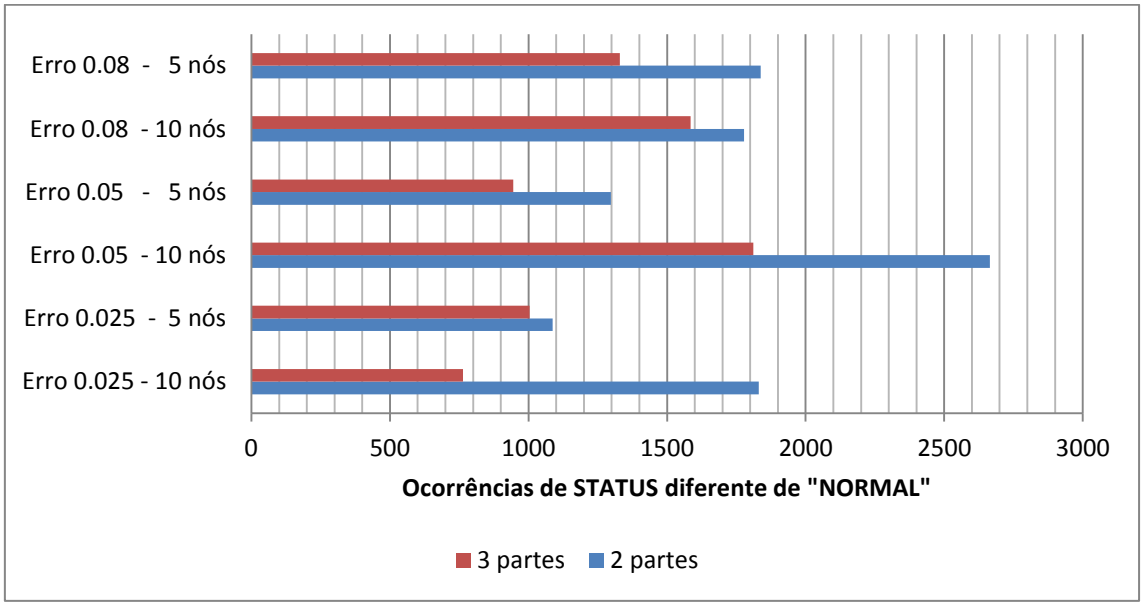


Gráfico 3 - Quantidade de ocorrências de mudanças de status para algo diferente de "NORMAL"

Como podemos observar, o Gráfico 3 também mostrou a mesma situação em que uma maior quantidade de partições diminuiu a instabilidade de nós. Observa-se que quando há três partições de nós, a quantidade de nós que entram em status diferente de "NORMAL" é menor que quando há apenas duas partições.

Referências bibliográficas

[1] Garcia-Molina, "Elections in a Distributed Computing System," *IEEE Transactions on Computers*, vol. C–31, no. 1. pp. 48–59, 1982.

[2] Distributed Computing Group, "Sinalgo," 2015. [Online]. Available: http://disco.ethz.ch/projects/sinalgo/. [Accessed: 25-May-2015].