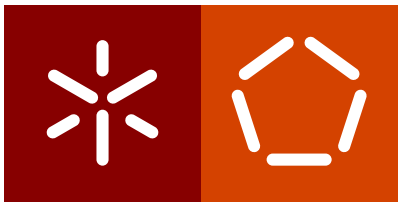
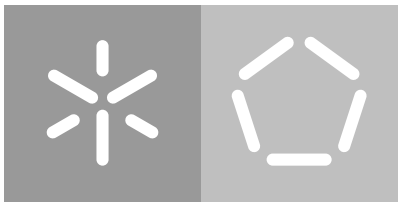


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Joaquim Manuel Gonçalves Oliveira

Análise e Optimização de Protocolos de Acordo Distribuído Aproximado

November 2021



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Joaquim Manuel Gonçalves Oliveira

Análise e Optimização de Protocolos de Acordo Distribuído Aproximado

Master dissertation
Master's in Informatics Engineering

Dissertation supervised by
José Orlando Pereira
Ana Nunes Alonso

November 2021

COPYRIGHT AND TERMS OF USE FOR THIRD PARTY WORK

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity.

I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

RESUMO

Esta dissertação aborda o acordo distribuído aproximado, no qual é pretendido que um grupo de processos decida um valor dentro de um intervalo com uma amplitude limitada. Na primeira fase procede-se ao levantamento dos algoritmos existentes na literatura, onde se incluem algoritmos que propõem a resolução no modelo assíncrono, no qual são consideradas faltas bizantinas. Através de uma análise comparativa, são evidenciadas as principais diferenças entre os algoritmos. Posteriormente, após a seleção criteriosa de um dos algoritmos, é feita a sua análise detalhada sobre os fatores de escalabilidade em resultado da sua implementação. Com base nos resultados, são propostas alterações que promovem a performance do algoritmo face ao aumento do número de processos no sistema.

Palavras chave: Acordo Distribuído Aproximado, Escalabilidade, Sistemas Distribuídos

ABSTRACT

This dissertation addresses the approximate distributed agreement, in which it is intended that a group of processes decides a value within a range with a limited amplitude. In the first phase, a study of existing algorithms in the literature is carried out, including algorithms that propose a resolution in the asynchronous model, where byzantine faults are considered. Then, through a comparative analysis, the main differences between the algorithms are highlighted. In the next phase, after a careful selection of one of the algorithms, a detailed analysis of the scalability factors resulting from its implementation is carried out. Based on the results, changes are proposed to promote the performance of the algorithm given the increase in the number of processes in the system.

Keywords: Approximate distribution agreement, Distributed system, Scalability

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

1

INTRODUÇÃO

Com o aumento de carga de trabalho sobre um dado serviço ou perante a necessidade de garantir a sua alta disponibilidade surge a tendência para abandonar a arquitetura centralizada para uma arquitetura distribuída. A descentralização pode levar a melhorias de desempenho face à disponibilidade e à resiliência do serviço, no entanto poderá também trazer problemas de coerência de dados, o que poderá levar à degradação do sistema. De forma a evitar estes problemas, existem diversos algoritmos que, através de canais de comunicação, oferecem garantias de coerência que permitem aos serviços persistirem ao longo do tempo. Neste contexto, surgem diversos protocolos que visam resolver o problema do acordo distribuído tendo como base diferentes pressupostos. Resolver o acordo distribuído pode ser definido, de forma geral, como conseguir que um grupo de processos decida, de forma coerente, um único valor. No entanto, existem generalizações que exploram condições de validade diferentes em que, por exemplo, o objetivo é alcançar um acordo aproximado em vez de exato. Esse acordo aproximado é alcançado quando todos os processos chegam a valores com uma dispersão limitada, ou seja, os valores finais de todos os processos pertencem a um intervalo com amplitude inferior a ϵ, tendo ϵ como uma constante do problema. Este problema pode ser surgir, como por exemplo, na sincronização de relógios entre maquinas ligadas entre si através de canais de comunicação [10, 19, 12].

1.1 PROBLEMA

Existem vários algoritmos que apresentam propostas de solução para este problema com base em diferentes pressupostos [3, 1, 4] entre os quais diferenciam no tipo de primitivas de comunicação utilizadas ou o modelo de faltas considerado. No entanto, estas soluções revelam, na generalidade dos casos, um baixo desempenho face ao aumento do número de processos. Isto deve-se ao facto de os algoritmos implementarem soluções que implicam a troca sistemática de mensagens entre todos os processos.

1.2 OBJETIVOS

Tendo em conta que a escalabilidade é um fator fulcral para muitos cenários em que estas soluções são aplicadas, este trabalho tem como objetivo fazer uma análise aos protocolos existentes, no sentido de identificar os seus gargalos e fraquezas, e então, após a seleção de um dos algoritmos estudados, tentar melhor o seu desempenho face ao aumento de processos.

Para tal, nesta dissertação é feito um levantamento dos protocolos existentes na literatura que se propõem resolver o problema do acordo distribuído aproximado. Para esse conjunto de algoritmos, é feita uma caracterização de forma a facilitar a compreensão das diferenças entre as diversas propostas. Desses algoritmos, é selecionado um onde é feito um estudo mais detalhado sobre a sua escalabilidade baseado nas funções matemáticas que modelam o seu comportamento. Com base neste estudo, é feita uma proposta de alteração que é implementada num ambiente de simulação através do qual são observados as diferenças entre as duas implementações face ao aumento dos seus principais fatores de escalabilidade.

1.3 ESTRUTURA DO DOCUMENTO

No segundo capítulo desta dissertação é apresentado o contexto onde surge o problema estudado onde serão introduzidos alguns conceitos fundamentais para os seguintes capítulos.

No terceiro capítulo é apresentado o estado da arte onde serão descritos os principais trabalhos desenvolvidos neste contexto.

No quarto capítulo é sustentada a escolha de um dos algoritmos apresentados anteriormente e onde são detalhadas as alterações propostas.

No quinto capítulo são analisados os detalhes de implementação do algoritmo. Ainda neste capítulo é descrita a metodologia para a encontrar a parametrização correta para a primitiva proposta.

No sexto capítulo é feita a análise da escalabilidade do algoritmo, onde é feita a comparação direta do algoritmo inicial e dos resultados das alterações propostas.

Por fim, no sexto capítulo é feita uma análise global do trabalho desenvolvido assim como de trabalho futuro.

2

ENQUADRAMENTO

Para a resolução de um dado problema, podemos encontrar várias solução. No entanto, de forma a encontrar a solução que melhor se adapta ao problema estudado, é necessário caracterizar o sistema onde ele se insere. Um sistema pode ser caracterizado por um conjunto de atributos que interagem entre si segundo um dado conjunto de regras. A este conjunto, assim como às respetivas regras, dá-se o nome de modelo [18].

Neste capítulo vamos explorar algumas definições fundamentais para a modelação de um sistema. Sustentado por essas definições, é apresentado de seguida o contexto onde surge o problema de acordo distribuído e de que forma está associado aos problemas de sincronização entre processos. Ainda neste capítulo, são exploradas definições matemáticas que sustentam os algoritmos estudados ao longo desta dissertação. Por fim, é caracterizada a classe de algoritmos, nomeadamente a classe de algoritmos MSR, que nos ajudará a catalogar os algoritmos.

2.1 MODELAÇÃO DE UM SISTEMA DISTRIBUÍDO

Um sistema diz-se assíncrono se não são considerados limites temporais para a realização de uma dada tarefa [3]. Assim, num sistema assíncrono não se pode assumir uma duração máxima, nem mínima, para o envio e receção de uma mensagem ou para processamento da mesma. De igual forma, não se podem assumir limites entre a diferença entre relógios dos diversos processos. Caso contrário, ou seja, caso exista algum limite temporal para a realização de uma tarefa, o sistema diz-se síncrono.

Para além da sincronia, pode-se caracterizar um sistema quanto ao tipo de falhas que nele podem existir. Um processo diz-se faltoso quando deixa de seguir o algoritmo definido. Desta forma, os processos faltosos podem, de forma arbitrária, alterar o seu estado interno, enviar um número finito de mensagens ou terminar a sua execução antes do estipulado pelo algoritmo. Os processos faltosos podem exibir o seu comportamento de várias formas e causar diferentes impactos no sistema. Deste modo, serão utilizadas as seguintes categorias de falhas de processos:

- **Falhas *crash-stop*** - Os processos podem, num instante arbitrário, falhar entrando num estado de terminação. Quando entram num estado de terminação não o podem recuperar e continuar a sua execução. [6].
- **Falhas por omissão** - Os processos podem falhar no envio ou na receção de um subconjunto de mensagens, o até, num instante arbitrário, entrar num estado de terminação. Tal como nas falhas *crash-stop*, quando entram num estado de terminação não podem recuperar o seu estado [16].
- **Falhas bizantinas** - Os processos podem exibir um comportamento arbitrário [11].

Facilmente se conclui que existe uma hierarquia entre as categorias, pois todas as falhas por *crash-stop* podem ser consideradas falhas por omissão que, por sua vez, podem ser consideradas falhas bizantinas. O contrário já não se verifica. Este resultado leva a que um algoritmo que resolva um dado problema num modelo de falhas bizantinas resolve também o mesmo problema num modelo de falhas omissivas e consequentemente num modelo de falhas *crash-stop*.

Dada a importância da comunicação entre processos num sistema distribuído, torna-se indispensável analisar as principais primitivas de comunicação assim como alguns algoritmos de dispersão de mensagens. Por omissão, vai ser considerados que os canais de transmissão de mensagens são canais *Reliable*, ou seja, canais que após o envio de uma mensagem oferecem a garantia que a mensagem será inevitavelmente entregue ao seu remetente. No entanto, esta primitiva não garante que a ordem de envio das mensagens seja a mesma que a ordem de chegada. Existem outras primitivas de comunicação que oferecem a garantia que a ordem da receção das mensagens é igual para todos os processos ou de que esta receção é atómica.

Para além destas primitivas, existem também protocolos de dispersão de mensagens tal como o algoritmo de disseminação epidémica que oferece uma solução escalável para a dispersão de mensagens na rede [13].

2.2 Problema do acordo aproximado distribuído

Existem várias metodologias para a sincronização entre processos. Estas baseiam-se, geralmente, em algoritmos que utilizam primitivas de comunicação em grupo [20]. No entanto, estas primitivas podem ser reduzidas ao problema de acordo distribuído [8, 2]. O acordo distribuído, ou também conhecido na literatura como problema de consensos, consiste num conjunto de processos fazerem uma votação, através de troca de mensagens, com o fim de alcançar um estado final coerente. Um algoritmo resolve o acordo distribuído se garantir as seguintes condições [14]:

- **Terminação** - Inevitavelmente, todos os processos corretos decidem um valor

- **Integridade** - O valor decidido pelos processos corretos têm que corresponder ao valor do estado inicial de, pelo menos, um processo.
- **Acordo** - Todos os processos corretos decidem um só valor final.

Este problema tem sido muito estudado surgindo alguns resultados interessantes. Mais concretamente, foi demonstrada a impossibilidade de resolver o acordo distribuído num sistema completamente assíncrono onde, pelo menos, um processo falhe segundo o modelo de *crash stop* [7]. No campo das falhas bizantinas foi demonstrando também que num sistema parcialmente síncrono onde podem ocorrer falhas bizantinas, o acordo pode ser alcançado se o número total de processos for três vezes superior ao número de processos faltosos admitidos [11].

Assim, e de forma a contornar estas impossibilidades, surgiram algoritmos que solucionam este problema num modelo mais forte, ou seja, com um conjunto de pressupostos do sistema mais fortes. No mesmo contexto surgiram também variantes do problema inicial relaxando as condições de verificação do problema. Neste contexto surge a definição do acordo distribuído aproximado. O problema do acordo distribuído aproximado consiste num conjunto de processos que iniciam o seu estado com um número real e que, através da troca de mensagens, alcançam um valor final cuja dispersão é limitada, isto é, que o conjunto formado pelos valores finais de todos os processos formam um intervalo real com amplitude menor que ϵ, sendo ϵ uma constante do problema. Um algoritmo resolve o acordo distribuído aproximado se satisfizer as seguintes condições [3]:

- **Terminação** - Inevitavelmente, todos os processos corretos decidem um valor
- **Integridade** - O valor acordado pelos processos corretos têm que estar dentro do intervalo formado pelo conjunto dos estados iniciais dos processos não faltosos.
- **Acordo** - Todos os processos corretos têm que decidir um valor dentro de um intervalo cuja amplitude é igual ou menor a ϵ.

Os algoritmos que resolvem este problema podem ser utilizados, por exemplo, na sincronização de relógios entre maquinas ligadas entre si através de canais de comunicação [10, 19, 12].

2.3 Definições e notações

Ao longo desta dissertação é abordada a forma como os processos colecionam os valores recebidos e a forma como fazem a transformação para obter o valor final. Por este motivo, nesta secção são abordados conceitos matemáticos fundamentais para perceber a forma como essas transformações ocorrem.

Consideremos um *multiset* como uma coleção de valores onde podem existir valores repetidos. Dado um *multiset* V, a multiplicidade de um dado valor r em V corresponde

ao número de ocorrências desse valor em V. Assim sendo, a cardinalidade de V se define como uma função $\mathbb{R} \longmapsto \mathbb{N}$ que para cada valor de r associa a sua multiplicidade onde $|V| = \sum_{r \in \mathbb{R}} V(r)$.

Tendo como base estas definições, sejam U e V *multisets* com $|V| = n$. Então:

- Define-se $min(V)$ como o menor valor de V, ou seja, o menor valor r onde $V(r) > 0$. De forma análoga define-se $max(V)$ como o maior valor de V.
- Define-se o intervalo $\rho(V) = [min(V), max(V)]$ e a respetiva amplitude como $\delta(V) = max(V) - min(V)$.
- Define-se o *multiset* $W = V - \mathrm{U}$ como um *multiset* formado pelos elementos de V que não pertencem a U. Desta forma, W pode ser definido como $W(r) = max(V(r) - \mathrm{U}(r),\ 0)\ \forall r \in \mathbb{R}$.
- Seja α um inteiro positivo onde $n > \alpha$. Define-se $high_\alpha(V)$ como os $n - \alpha$ maiores valores de V. De forma análoga, definimos $low_\alpha(V)$ como os $n - \alpha$ menores valores de V.
- Seja α um inteiro positivo onde $|V| > 2\alpha$. Define-se $reduce^\alpha(V) = high_\alpha(low_\alpha(V))$ um *multiset* após remover os α maiores e os α menores de V. Assim $|reduce^\alpha(V)| = |V| - 2\alpha$. Esta função é chamada função de redução.
- Define-se $mean(V)$ como um valor real resultante de calcular a média aritmética dos valores de V. Esta função é chamada função de média.

2.4 CLASSES DE ALGORITMOS

No âmbito dos algoritmos de votação convergente existem diversas classes de algoritmos. A classe a que um algoritmo pertence deve ser tida em consideração quando se comparam dois algoritmos na medida em que cada classe descreve um comportamento distinto do qual se podem extrair diferentes propriedades. Nesta secção são apresentadas as principais classes de algoritmos abordados ao longo desta dissertação.

2.4.1 *Algoritmos ronda a ronda*

Os algoritmos desta classe [1] são caracterizados por executarem um determinado conjunto de rondas onde em cada ronda é feita a troca de mensagens e, depois de colecionar um determinado número de mensagens, é-lhe aplicada a função de aproximação resultando um novo valor a transitar para a ronda seguinte. Este ciclo de rondas é finito e o valor resultante da última ronda é o valor final do algoritmo.

As rondas destes algoritmos têm a particularidade de serem independentes umas das outras, isto é, não existe passagem de informação entre as sucessivas rondas e não são considerados eventos passados.

2.4.2 *Algoritmos de votação MSR*

A classe de algoritmos *Mean-Subsequence-Reduced* (MSR) [9, 3] é um caso particular da classe de algoritmos ronda a ronda. Os algoritmos desta classe são caracterizados pela função de aproximação f ter um conjunto de características que lhe permite fazer o papel da função de média e que, mesmo perante a presença de falhas, garante a convergência dos valores obtidos pelos processos.

A função f pode ser definida como:

$$f(V) = mean[select_{\sigma}(reduce^{\alpha}(V)]$$

estando a sua aplicação representada na figura 1. O parâmetro t da função de redução define o total de valores removidos dos extremos e o parâmetro σ da função de seleção define o número de elementos do conjunto do $reduce^{\alpha}(V)$ que são selecionados para posterior aplicação da função *mean*. Desta forma, os algoritmos pertencentes a esta classe diferem entre si nos parâmetros σ e α e na definição da função *select*.

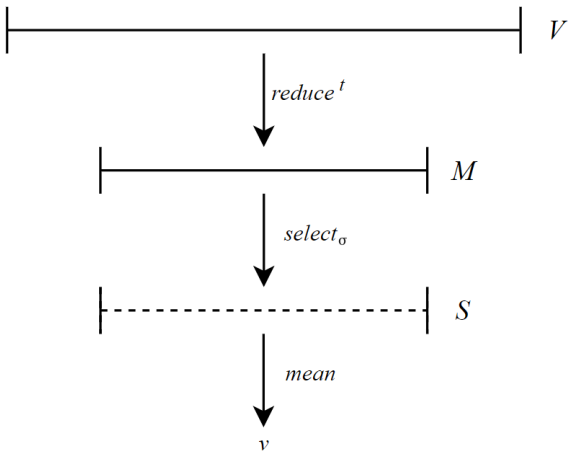


Figura 1: Processo de aproximação

Para comparar os algoritmos pertencentes a esta classe é apresentada uma medida genérica C [9] que representa a taxa de convergência da função de aproximação. Essa a medida representa a diferença máxima obtida pela aplicação da função entre dois processos. Este valor depende do número de processos no sistema, do número máximo de processos faltosos

tolerados e da amplitude do conjunto formado pelos valores comuns pelos dois processos. Tem-se então que

$$|f(N_i) - f(N_j)| \leq C \cdot \delta(N_i \cap N_j), \forall_{i,j} \in [1, n]$$

onde f é a função de aproximação, N_i e N_j representam os conjuntos iniciais numa dada ronda dos processos i e j, respetivamente, e $\delta(N_i \cap N_j)$ corresponde à amplitude dos valores comuns colecionados por i e j. Note-se que quanto menor for o valor de C, maior será a convergência do algoritmo. Nesta dissertação esta medida é generalizada a todos os algoritmos que executam uma sequência de rondas de forma a possibilitar a comparação da convergência dos mesmos.

2.4.3 *Algoritmo S-Round*

Esta classe de algoritmos [4] engloba todos os algoritmos que executam um determinado conjunto de rondas onde existe a partilha e acumulação de informação que é considerada quando se aplica a função de aproximação.

Para estes algoritmos a taxa de convergência também pode ser analisada mas para um determinado número de rondas, já que pode não fazer sentido analisar para cada ronda individualmente dada a natureza do algoritmo.

3

ESTADO DA ARTE

Atualmente existem várias propostas para resolver o acordo distribuído aproximado em diversos modelos. Neste capítulo vamos analisar algumas destas propostas. Esta análise é concluída com uma perspetiva global de todos os algoritmos apresentados.

Consideremos ao longo deste capítulo n como o número de processos do sistema e t o número máximo de processos faltosos tolerados.

3.1 MODELO SÍNCRONO

3.1.1 *Classe de algoritmos MSR*

Modelo de faltas bizantinas

Para este modelo é proposto um algoritmo [3] para resolver o problema onde é assumido a existência de uma primitiva de comunicação *reliable* através da qual todos os processos conseguem comunicar entre si assim como identificar o emissor de uma dada mensagem.

O algoritmo é constituído por uma sequência de rondas onde em cada uma delas os processos não faltosos enviam para todos o seu atual valor, incluindo para si mesmo, e colecionam os valores recebidos num *multiset* V. Caso uma mensagem de um dado processo não chegue dentro do tempo estimado, e porque estamos num sistema síncrono, o processo é considerado faltoso e é-lhe atribuído um valor por omissão. Depois de colecionar n valores em V, é aplicada a função de aproximação f ao *multiset* V através da qual é obtido o valor a transitar para a ronda seguinte.

A função de aproximação utilizada neste algoritmo segue o padrão apresentado na classe de algoritmos a que este pertence. Sendo assim, pode definir-se f como sendo

$$f(V) = mean[select_{c(n-2t,t)}(reduce^t(V)]$$

Nesta definição, a função de redução remove t valores de cada extremo, sendo $R = reduce^t(V)$ e $|R| = n - 2t$. Esta remoção deve-se à possibilidade de existirem até t va-

lores faltosos em V e esses valores estarem nos extremos do *multiset* afetando de forma significativa a função de aproximação. Ao *multiset* R resultante, é aplicada a função de seleção. Para definir esta função considere o *multiset* R como uma lista ordenada, por ordem crescente, $< r_0, r_1, \ldots, r_m >$ onde $m = n - 2t$. Define-se a função de seleção que devolve um *multiset* S formado por $< r_0, r_k, r_{2k}, \ldots, r_{jk} >$ onde e $|S| = c(m,k) = \lfloor \frac{m-1}{k} \rfloor$. Tendo em conta a possibilidade de ainda existirem t valores faltosos em R, tem-se que $k = t$ resultando $S = select_{c(n-2t,t)}(R)$. Por fim é aplicada a função aritmética de média da qual resulta o valor final da função de aproximação.

Para um sistema onde $n \geq 3t + 1$, seja p um processo correto numa dada ronda h. Sejam V e V' os conjuntos de valores obtidos por p na ronda h e $h + 1$ respetivamente. Nestas condições, é demonstrado que:

1. $\rho(V') \subseteq \rho(V)$, isto é, que a amplitude é decrescente de ronda para ronda
2. $\delta(V') \leq \frac{\delta(V)}{c(n-2t,t)}$, isto é, que a amplitude decresce a uma taxa mínima de $\frac{1}{c(n-2t,t)}$

Desta forma é demonstrado que a função utilizada neste algoritmo é garantidamente convergente com uma taxa de convergência $C = \frac{1}{c(n-2t,t)}$ [9] quando o número de participantes é três vezes superior ao número de falhas toleradas. É demonstrado ainda que para o modelo de faltas e a classe de algoritmos considerados, os resultados obtidos são os ideais quanto à resiliência e à taxa de convergência.

Embora esteja demonstrada a convergência do algoritmo ainda não foi garantida a correção do algoritmo tendo em conta que não foi definida a condição de terminação. Deste modo, é acrescentada uma ronda zero onde o processo executa os mesmos passos descritos para a generalidade das rondas mas que acrescenta o cálculo do número de rondas a executar. Este número de rondas é calculado através da função $\lceil \log_{c(n-2t,t)}(\frac{\delta(V)}{\epsilon}) \rceil$ que nos dá a garantia, através de 1) e 2), que após a execução desse número de rondas a amplitude dos valores iniciais dos processos não faltosos formam um conjunto cuja amplitude é inferior a ϵ e que a amplitude desse conjunto será decrescente nas sucessivas rondas. Tendo isto, os processos após executar esse número de rondas podem assinalar a sua terminação e abandonar a sua execução, tendo a garantia que as condições de verificação irão se manter nas sucessivas rondas. Os restantes processos ao receberem a mensagem de terminação assumem que para as restantes rondas o valor desse processo irá manter-se.

Para que um processo possa determinar corretamente o número de rondas a executar é necessário ter a garantia que os valores determinados por todos os processos para a ronda seguinte está dentro da amplitude dos valores que foram consideramos para determinar o número de rondas a executar. Caso contrário, o processo poderá determinar um número de rondas insuficientes para alcançar o acordo. Esta garantia é extraída das propriedades da função de aproximação f sendo que esta função é utilizada para todas as rondas do algoritmo.

Ronda 0 (Ronda inicial):

Input v

$V \leftarrow$ *trocaSincrona(v)*

$v \leftarrow f_{t,t}(V)$

$H \leftarrow \lceil log_c(\delta(V)/\epsilon) \rceil$, onde $c = c(n-2t, t)$

Ronda h ($1 \leq h < H$) (Rondas de aproximação):

$V \leftarrow$ *trocaSincrona(v);*

$v \leftarrow f_{t,t}(V)$

Ronda H (Ronda final):

broadcast(<v, halted>)

output v

trocaSincrona(v):

broadcast(v)

coleciona n valores em V

- adiciona os valores dos processos que já terminaram
- adiciona valores por defeito, se necessário

retorna V

Figura 2: Pseudo-código do algoritmo

3.1.2 *Classe de algoritmos S-Round*

Para esta classe de algoritmos, o número de faltas toleradas e o número de faltas que efetivamente ocorrem durante a execução do algoritmo afeta a progressão do algoritmo, em contraste com os restantes algoritmos que apenas são afetados pelo número total de processos faltosos tolerados. Para além disso, estes algoritmos toleram t processos faltosos ao longo de toda a execução do algoritmo, em contraste dos restantes algoritmos que toleram t processos faltosos por ronda.

Assim sendo, consideremos no cálculo do fator de convergência que as t faltas toleradas pelo algoritmo ocorrem durante a execução do mesmo.

Modelo de faltas crash-stop

O algoritmo [4] proposto tira partido do modelo de faltas onde se insere para obter uma resiliência de $n > t$ e um coeficiente C duas vezes inferior, fazendo com que a amplitude dos valores seja reduzida duas vezes mais rápido e se obtenha a convergência em menos rondas.

Durante a execução de uma dada ronda, os processos faltosos são representados com um símbolo em vez de lhes ser atribuído um valor por defeito. Assim, na redução do conjunto de forma a obter o valor a transitar para a ronda seguinte, estes símbolos são os

primeiros valores a serem descartados. Como estamos num modelo síncrono, os processos são considerados faltosos se a mensagem desse processo demorar mais do que o tempo estimado.

Modelo de faltas omissivas e bizantinas

Para esta classe de algoritmos foi proposto um algoritmo [4] que se propõem resolver o problema no modelo de faltas bizantinas ou no modelo de faltas omissivas. Este algoritmo pode ser ajustado através de alguns parâmetros para se adaptar ao modelo de faltas onde está a ser aplicado. No entanto, para ambos os modelos a lógica do algoritmo é a mesma.

Este algoritmo, em contraste à estrutura dos algoritmos ronda a ronda, é constituído com duas fases distintas. A primeira fase é caracterizada por uma elevada troca de mensagens onde o objetivo é detetar os processos faltosos através da retransmissão de mensagens. Na segunda fase são executados cálculos considerando os dados provenientes da primeira fase de forma a determinar o valor final do algoritmo.

Assim, durante a execução da primeira fase são executadas S rondas, sendo S um parâmetro do algoritmo. As mensagens trocadas entre os processos neste algoritmo, para além de conterem o valor proposto pelo processo, contêm uma lista de processos por onde essa mensagem passou. Assim, na receção de uma mensagem, o processo acrescenta à lista dos processos retransmissores daquela mensagem o seu identificador único, mantendo assim um historial de onde a mensagem passou.

No início de cada ronda, cada processo, envia uma mensagem para todos os participantes contendo todos os valores recebidos na ronda anterior. Este comportamento faz com que o tamanho das mensagens cresça exponencialmente de ronda para ronda. Depois de terminar o envio das mensagens, o processo coleciona as mensagens enviadas por todos os processos. Os processos que demoram mais do que o do tempo estimado, e porque estamos num sistema síncrono, são considerados faltosos e o seu valor é definido como $\perp$. Depois de colecionar n mensagens, o processo executa uma função de deteção de faltas que denota os processos faltosos de igual forma com $\perp$. Após esta execução, o processo avança para a ronda seguinte repetindo todo o processo.

Note-se que durante esta fase os valores propostos pelos processos matem-se inalterados durante as várias rondas. Nesta fase apenas é feita a retransmissão das mensagens e assinalados os processos faltosos.

Depois de executar as S rondas o processo entra na segunda fase. Durante esta fase não existe troca de mensagens entre os processos, sendo esta fase caracterizada por uma forte componente computacional. Durante esta fase os dados provenientes da primeira fase são reduzidos de uma forma iterativa até resultar um único valor. Este valor corresponde ao resultado final de executar o algoritmo.

O algoritmo ajusta-se ao modelo de faltas através de parâmetros que são aplicados na função de deteção de faltas e nas funções que reduzem os dados na segunda fase.

Com este algoritmo é obtida uma resiliência $n > 2t$ para o modelo de faltas omissivas e $n > 4t$ para o modelo de faltas bizantinas.

3.2 MODELO ASSÍNCRONO

3.2.1 *Classe de algoritmos MSR*

Modelo de faltas bizantinas

O algoritmo proposto para este modelo [3] é uma adaptação ao modelo assíncrono do algoritmo proposto para a classe de algoritmos MSR considerando a existência de faltas bizantinas.

Neste algoritmo, cada processo coleciona apenas $n - t$ valores por ronda, contrariamente ao algoritmo inicial onde eram colecionados n. Isto deve-se ao facto de no modelo assíncrono não existirem limites temporais para considerar um processo faltoso e atribuir-lhe um valor por defeito.

Os parâmetros da função de aproximação é afetada tendo em conta que o *multiset* inicial apenas contem $n - t$ elementos. Desta forma, da aplicação a função redução que remove t elementos de cada extremos, resulta $R = reduce^t(V)$ e $|R| = (n-t) - 2t = n - 3t$. Aplicando a mesma definição da função de seleção, temos que $S = select_{c(n-3t,t)}(R)$. Por fim, é aplicada a função aritmética de média da qual resulta o valor final da função de aproximação. Assim sendo, a função de aproximação pode ser vista como

$$f(V) = mean[select_{c(n-3t,t)}(reduce^t(V)]$$

Para um sistema onde $n \geq 5t + 1$, seja p um processo correto numa dada ronda h. Sejam V e V' os conjuntos de valores obtidos por p na ronda h e $h + 1$ respetivamente. Nestas condições, é demonstrado que:

1. $\rho(V') \subseteq \rho(V)$, isto é, a amplitude decresce entre rondas
2. $\delta(V') \leq \frac{\delta(V)}{c(n-3t,t)}$, isto é, a amplitude decresce a uma taxa mínima de $\frac{1}{c(n-3t,t)}$

Desta forma é demonstrado que a função utilizada neste algoritmo é garantidamente convergente com uma taxa de convergência $C = \frac{1}{c(n-3t,t)}$ quando o número de participantes é cinco vezes superior ao número de falhas toleradas. É demonstrado ainda que para o modelo de faltas e a classe de algoritmos considerados, os resultados obtidos são os ideais quanto à resiliência e à taxa de convergência.

Da mesma forma que no algoritmo síncrono inicial, a função de aproximação utilizada na ronda inicial deve dar a garantia que os valores determinados por todos os processos para a ronda seguinte está dentro da amplitude dos valores que foram consideramos para determinar o número de rondas a executar. Para este algoritmo, a função de aproximação utilizada nas rondas de aproximação não oferece esta garantia de forma a que esta não possa ser utilizada para a ronda inicial. Assim, é utilizada na ronda zero a função $mean[reduce_{2t}(V)]$. Esta função é apenas utilizada na ronda zero apresenta uma taxa de convergência menos vantajosa face à função de aproximação utilizada.

Modelo de faltas omissivas

Este problema foi anteriormente abordado para o mesmo modelo de faltas num sistema síncrono[4]. Mais tarde, e tendo este primeiro trabalho como base, foi demonstrado [5] que para o modelo assíncrono um algoritmo ronda a ronda seria mais vantajoso do que um *S-Round* na medida em que no modelo assíncrono não há informação conclusiva sobre os processos faltosos, dada a natureza assíncrona do sistema. Desta forma, é apresentado um algoritmo da classe de algoritmos MSR onde a função de aproximação utilizada é

$$f(V) = mean[select_{c(n-t,t)}(V)]$$

segunda a definição de c anteriormente apresentada. O algoritmo tem uma resiliência de $n > t$ e uma taxa de convergência de $C = (\lceil \frac{n-t}{t} \rceil)^{-S}$

Ainda neste artigo é demonstrado que o desempenho do algoritmo depende do número máximo de faltas toleradas, não havendo variações de desempenho consoante o número de faltas que decorrem numa dada execução. Assim, podemos concluir que o valor de C obtido mantém-se no pior e no melhor caso, ao contrário do resultado obtido em algoritmos anteriores [4].

3.2.2 *Classe de algoritmos Ronda a Ronda*

Modelo de faltas omissivas

Este algoritmo [17] tem por base o trabalho desenvolvido anteriormente apresentado [5] e apresenta um algoritmo para o modelo assíncrono considerando a presença de faltas omissivas. O algoritmo tem uma estrutura muito semelhante aos algoritmos da classe MSR, no entanto este não pertence à classe porque a função de aproximação tem uma estrutura diferente.

Assim, durante a execução do algoritmo cada processo, em cada ronda, envia para os restante processos uma mensagem com o seu valor proposto e coleciona as mensagens recebidas num *multiset* com $n - t$ valores. A este *multiset* é aplicada função

$$f(V) = \frac{mean[select_t(low_{n-t}(V)] + mean[select_t(hight_{n-t}(V)]}{2}$$

Com esta função de aproximação é demonstrado que o algoritmo tem uma resiliência de $n > t$ e tem um coeficiente $C = (\lceil \frac{n-t}{t} \rceil \times 2)^{-1}$

Modelo de faltas bizantinas

Por fim, é proposto um algoritmo [1] que endereça o mesmo problema num sistema assíncrono considerando faltas bizantinas. No modelo é considerada uma primitiva de comunicação *reliable-broadcast* que oferece as seguintes garantias:

- Se um processo p envia uma mensagem m associada a uma ronda r, inevitavelmente todos os processos não faltosos recebem m de p associada a r
- Se um processo não faltoso p não envia uma mensagem m associada a uma ronda r, então nenhum processo não faltoso recebe m associada a r de p.
- Se um processo recebe uma mensagem m de p associada a uma ronda r, então todos os processos não faltosos vão receber uma mensagem m' de p relativa à ronda r e $m = m'$

Com esta primitiva de comunicação o modelo de faltas, embora bizantino, fica enfraquecido na medida em que um processo faltoso não pode enviar mensagens diferentes relativas à mesma ronda para processos diferentes.

O algoritmo apresentado é baseado no conceito de testemunha. Um processo q diz-se testemunha de p se os primeiros $n - t$ valores recebidos q estão contidos nos valores recebidos por p. O algoritmo é constituído por uma série de rondas onde, cada processo, envia o seu valor atual para todos os processos. Depois de enviar, coleciona os valores recebidos relativos à atual ronda em V. Para além disso, sempre que recebe uma mensagem, envia um tipo especial de mensagem para todos os processos a sinalizar esse evento. Dessa forma, todos os processos sabem que valores foram recebidos por todos, e dessa forma conseguem calcular o número de testemunhas que cada um tem. Quando o número de testemunhas for igual a $n - t$, ou seja, existem $n - t + 1$ processos com, pelo menos, $n - t$ valores em comum, o processo aplica-lhe a função de aproximação f e transita para a ronda seguinte. A função f é definida como:

$$f(V) = \frac{max(reduce_t(V)) + min(reduce_t(V))}{2}$$

Tal como nos algoritmos apresentados, o número de rondas necessárias é determinado numa ronda zero com base na taxa de convergência do algoritmo e nos valores colecionados.

Este algoritmo apresenta uma resiliência de $n > 3t$ e uma taxa de convergência $C = \frac{1}{2}$.

3.3 VISÃO GERAL DOS ALGORITMOS APRESENTADOS

Ao longo deste capitulo foram apresentados vários algoritmos para resolver o problema do acordo distribuído aproximado. Embora os algoritmos se proponham a resolver o mesmo problema, estes partem de pressupostos diferentes chegando a soluções distintas. Nesta secção vão ser analisados alguns fatores que nos permitem comparar os algoritmos estudados.

Os primeiros fatores a serem considerados devem ser os fatores relativos ao modelo onde o algoritmo se aplica. Neste âmbito, deve ser considerada a sincronia do sistema assim como o modelo de faltas onde o algoritmo é aplicado. Devem ser considerados também todos outros pressupostos do sistema, como por exemplo, o conjunto de propriedades da primitiva de comunicação.

Dentro do mesmo modelo, os algoritmos podem ser comparados segundo a sua resiliência. Este fator é analisado através da relação entre o número total de processos e o número máximo de processos faltosos que são tolerados pelo algoritmo.

Ainda dentro do mesmo modelo, os algoritmos podem ser comparados face à escalabilidade do número total de processos no sistema. Este fator pode ser analisado em várias vertentes, entra elas:

- número de rondas executadas
- número de mensagens trocadas
- tamanho das mensagens
- carga computacional

O número de rondas a executar é influenciada pela razão entre a amplitude dos valores iniciais e o valor de ϵ. O coeficiente de convergência C também influencia o número de rondas executas dado que este valor indica o quão rápido os valores dos processos convergem ao longo do algoritmo. O número de mensagens trocadas durante a execução do algoritmo é naturalmente afetado pelo número de rondas a executar assim como o total de mensagens trocadas em cada ronda. Note-se que na generalidade dos algoritmos estudados, são enviadas n^2 mensagens por cada ronda. O tamanho das mensagens também pode ser um fator em ter em consideração na medida em que irá representar uma maior carga computacional requerida para o processamento das mensagens ou ate por ser uma restrição do sistema, como por exemplo, em relação à largura de banda disponível. Apenas para

um dos algoritmos estudados [4] este fator representa uma preocupação na medida em que o tamanho das mensagens cresce ao longo da execução. Para os restantes algoritmos o tamanho das mensagens é constante. Por fim, o tempo de execução do algoritmo é uma vertente que é afetada por todas as vertentes anteriores mas também pela arquitetura do algoritmo.

Na tabela 1 é apresentada uma visão geral dos algoritmos estudados onde estão representadas as principais características dos algoritmos.

Na figura 3 pode-se observar a convergência obtida por cada algoritmo considerando a execução de 4 rondas face ao aumento do número de processo. Para cada valor do número total de processos é considerado o número máximo de processos faltosos compatível por todos os algoritmos. Neste gráfico, pode observar que a convergência não é afetada pelo aumento do número processos, excepto para os algoritmos da classe *S-Round*. Isto sucede devido às funções de convergência destes algoritmos refletirem o facto deles apenas tolerarem t processos faltosos durante toda a sua execução. Isto provoca um aumento linear do numerador destas funções face ao aumento do denominador que cresce exponencialmente.

Nas figura 4 pode-se observar a convergência obtida pelos algoritmos face ao aumento de número de rondas executadas. Aqui são considerados 120 processos onde são tolerados 23 faltosos. Neste gráfico também se pode observar que o maior declive dos algoritmos *S-Round* face aos restantes processos.

Por fim, na figura 5 pode-se observar a convergência obtida pelos algoritmos face ao aumento de número de processos faltosos. Aqui são considerados 120 processos onde são executadas 4 rondas. Tal como foi visto nos gráficos anteriores, neste gráfico também se pode observar que os algoritmos *S-Round* obtêm um valores muito inferiores face aos restantes processos. O algoritmo ar1 apresenta um valor constante derivado da função que descreve a sua taxa de convergência ser uma constante.

1 O algoritmo utiliza uma primitiva de comunicação que diminui o poder dos processos bizantinos

Sincronia	Classe	Modelo de faltas	Resiliencia	C	Legenda
Síncrono	MSR	Bizantinas [3]	n >3t	$(\lfloor \frac{n-2t-1}{t} \rfloor + 1)^{-1}$	sm
	S-Round	Crash-stop [4]	n >t	$\frac{t}{(2n-2t)^S}$	ss1
		Bizantinas [4]	n >4t	$\frac{t}{(n-2t)(n-4t)^{S-1}}$	ss2
		Omissivas [4]	n >2t	$\frac{t}{(2n-4t)^{S-1}(2n-2t)}$	ss3
Assíncrono	MSR	Bizantinas [3]	n >5t	$(\lfloor \frac{n-3t-1}{2t} \rfloor + 1)^{-1}$	am1
		Omissivas [5]	n >t	$(\lceil \frac{n-t}{t} \rceil)^{-1}$	am2
	Ronda ronda	Bizantinas [1] [1]	n >3t	$\frac{1}{2}$	ar1
		Omissivas [17]	n >t	$(\lceil \frac{n-t}{t} \rceil \times 2)^{-1}$	ar2

Tabela 1: Sumário dos algoritmos apresentados

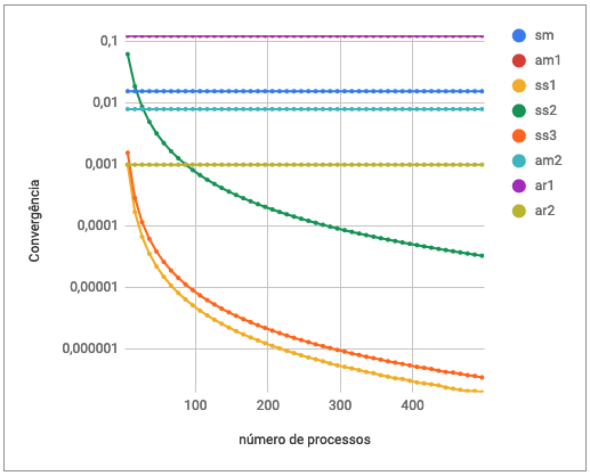


Figura 3: Taxa de convergência obtida por cada algoritmo face ao aumento do número de processos

[2]

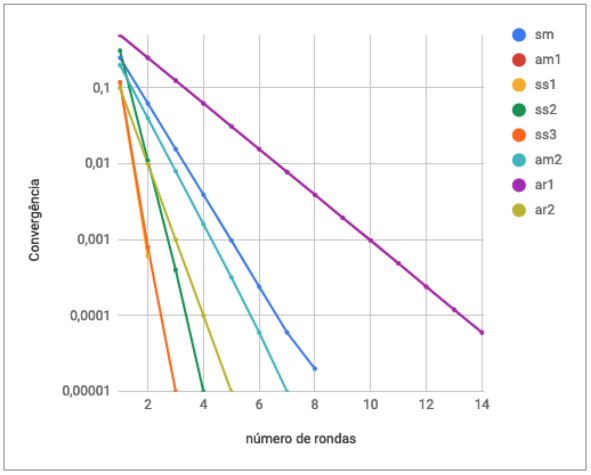


Figura 4: Taxa de convergência obtida face ao aumento do número de rondas a executar [2]

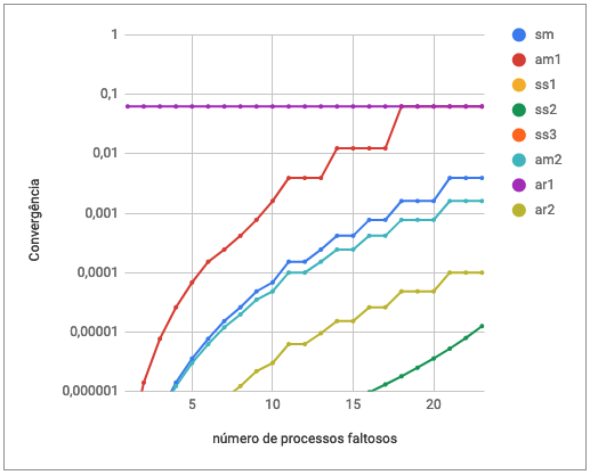


Figura 5: Taxa de convergência obtida face ao aumento do número de processos faltosos [3]

2 No gráfico, os valores do algoritmo am1 estão sobrepostos com o valor do algoritmo ar1
3 No gráfico, os valores do algoritmo ss1 e ss3 estão ocultos por se encontrarem muito perto de valores de zero

4

EXPLORAÇÃO DO ALGORITMO E ALTERAÇÕES PROPOSTAS

No capítulo anterior foram detalhadas algumas propostas para resolver o problema em estudo. Neste capítulo é sustentada a escolha de um dos algoritmos apresentados para ser estudado com mais detalhe. Para este algoritmo é apresentada uma proposta de alteração que cujos resultados são explorados nos capítulos subsequentes.

4.1 EXPLORAÇÃO DO ALGORITMO EM ESTUDO

O algoritmo estudado nesta secção é o algoritmo da classe MSR apresentado para o modelo assíncrono considerando a presença de faltas bizantinas [3]. Esta escolha é sustentada pelo facto de o algoritmo ter a sua correção formalmente demonstrada assim como a demonstração que, para o modelo considerado e para a classe de algoritmos onde se insere, o algoritmo apresenta a melhor resiliência assim como taxa de convergência possível. O modelo de faltas também é um fator determinante tendo em conta que o modelo assume poucas restrições do sistema podendo ser generalizado à maior parte dos casos de uso.

Nesta secção é analisada a função que determina o número de rondas a executar. Estes resultados são importantes para, em capítulos seguintes, sustentar os resultados obtidos pelo algoritmo face ao aumento dos fatores de escalabilidade. Estes resultados são obtidos através da exploração das funções matemáticas que modelam o algoritmo. Para alem dessa análise, é ainda detalhada a modelação do comportamento dos processos bizantinos e a forma como o algoritmo faz o tratamento das mensagens.

4.1.1 *Número de rondas a executar*

Um dos principais fatores que afeta a performance do algoritmo é o número de rondas executadas pelos processos. Este valor é determinado na primeira ronda de cada processo onde é utilizada a amplitude dos valores recebidos $\delta(V)$ juntamente com o coeficiente de convergência do algoritmo c, tal como pode ser observado na sua definição:

$$\left\lceil log_c(\frac{\delta(V)}{\epsilon}) \right\rceil, c = c(n-3t, 2t)$$

Nesta secção é feita uma análise desta função segundo o aumento do número total de processos, do número de processos faltosos tolerados e da razão entre $\delta(V)$ e ϵ. Por omissão, são considerados um total de 151 processos onde são tolerados 30 faltosos e a razão entre o valor de $\delta(V)$ e ϵ é igual a 1000000.

No no gráfico 6 pode ser observado a variação do número total de rondas a executar face ao aumento do número de processos faltosos tolerados. Neste gráfico podemos salientar o rápido crescimento do número de rondas a executar no terço superior do número de processos faltosos. Este comportamento pode ser observado para outras configurações onde o número total de processos, a amplitude ou o valor de ϵ variam.

No gráfico 7 pode-se observar o impacto da variação do número total de processos na determinação do número de rondas a executar. Para cada ponto no gráfico onde é considerado um dado número total de processos n são considerados $(n-1)/5$ processos faltosos tolerados. Assim, podemos observar que o número de rondas a executar tende a estabilizar face ao crescimento de n. Na parte inicial do gráfico pode ser observada uma pequena variação causada pela definição da função c.

Por fim, no gráfico 8 o número total de rondas a executar é analisada segundo a razão entre $\delta(V)$ e ϵ. Pode-se observar um comportamento logarítmico da curva.

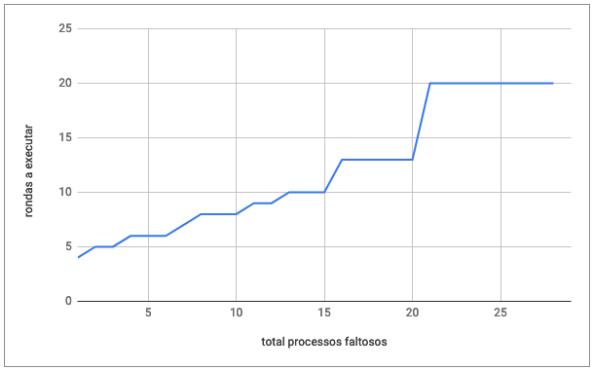


Figura 6: Total de rondas a executar face ao aumento do número de processos faltosos

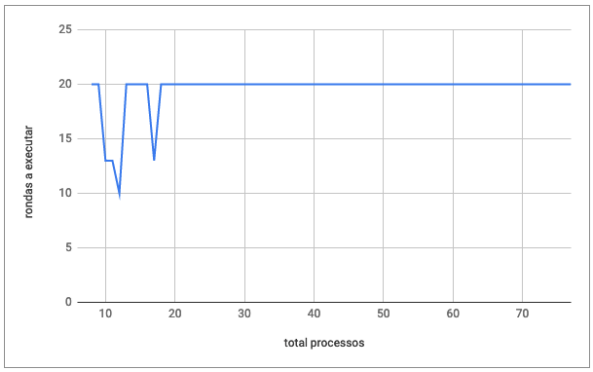


Figura 7: Total de rondas a executar face ao aumento do número total de processos

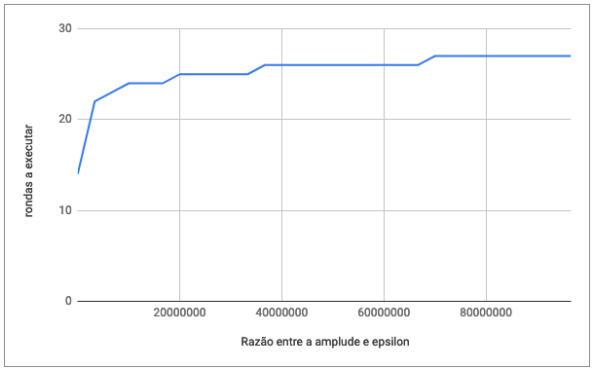


Figura 8: Total de rondas a executar face ao aumento da amplitude dos valores iniciais dos processos

4.1.2 *Processos faltosos*

O comportamento dos processos faltosos pode ser expresso de várias formas tendo em conta o sua natureza bizantina. Este comportamento pode afetar uma única ronda ou um determinado conjunto de rondas. Por exemplo, se um processo faltoso enviar uma mensagem com o valor proposto adulterado, essa falta bizantina apenas irá afetar a ronda à qual a mensagem diz respeito. No entanto, se um processo faltoso assinalar a sua terminação de forma precoce, esta falta irá afetar a ronda onde o processo enviou a mensagem faltosos assim como as seguintes rondas. Embora estes dois tipos de faltas requerem uma gestão mais complexa do número de processos faltosos em cada ronda, a diferença entre elas não afeta a progressão do algoritmo desde que haja a garantia de que em cada momento existem no máximo t faltas.

Desta forma, o comportamento dos processos bizantinos foi modelado de forma e simplificar a gestão de faltas por cada ronda e de forma a maximizar o impacto destes processos no sistema. Assim sendo, um processo é considerado bizantino numa dada ronda se adulterar o

valor proposto nas mensagens. Quando faltosos, os processos enviam diferentes mensagens para diferentes processos referentes à mesma ronda. Com este comportamento, os processos bizantinos não omitem envios de mensagens de forma a promover a diferença entre os valores colecionados por cada processo.

O comportamento dos processos bizantinos descrito não introduz nenhuma restrição à sua modelação, apenas irá maximizar o seu impacto no algoritmo.

4.1.3 *Mensagens*

Tal como foi visto anteriormente, a troca de mensagens é o mecanismo pelo qual os processos comunicam. Um processo ao receber uma mensagem pode fazer o processamento da mesma de formas distintas consoante o ronda da mensagem e o seu estado interno.

Assim, consideremos um processo p que se encontra a executar a ronda r_p. Num dado instante, esse processo recebe uma mensagem m relativa a uma ronda r_m.

- Caso r_p seja igual a r_m, p guarda a mensagem m e esta será utilizada para determinar o valor a transitar para a ronda seguinte. Esta mensagem, para p, é considerada uma mensagem útil.
- Caso r_p seja superior a r_m, isto é, a mensagem é relativa a uma ronda passada de p, o processo não a considera em qualquer tipo de cálculos sendo a mensagem considerada ignorada para p.
- Caso r_p seja inferior a r_m, p irá guardar a mensagem para a processar quando alcançar a ronda à qual a mensagem é relativa. Neste caso, a mensagem pode ser considerada útil ou ignorada, consoante o instante em que a mensagem for processada no futuro.

Nesta secção foi introduzido o conceito de mensagem ignorada e de mensagem útil. Estes conceitos serão úteis ao longo desta dissertação não só para entender a lógica do algoritmo mas também para analisar os resultados das execuções.

4.2 Alterações propostas

As alterações propostas visam minimizar o número de mensagens enviadas entre os processos de forma a que o algoritmo possa obter melhores resultados face à escalabilidade do número de processos envolvidos no acordo. O tempo de execução dos processos é também uma métrica importante dado que esta é afetada pela redução de mensagens a processar mas também é afetada pelas alterações propostas ao algoritmo que requerem mais poder computacional.

Como foi visto anteriormente, no algoritmo estudado parte das mensagens recebidas são ignoradas. Estas mensagens ignoradas devem ser minimizadas na medida em que vão

requerer recursos da rede de comunicação e poder computacional dos processos sendo que estas não contribuem para a progressão do algoritmo. No entanto, os processos emissores não têm informação para determinar se as mensagens que estão a enviar vão ser ignoradas pelos processos recetores. Desta forma, eles não podem enviar apenas um subconjunto dessas mensagens sem pôr em causa a correção do algoritmo.

Assim sendo, as alterações propostas visam minimizar estes custos dando aos processos um mecanismo para descartar o envio de mensagens sem colocar em causa a correção do algoritmo.

Para tal, foi desenvolvida uma primitiva de comunicação que retarda parte dos envios das mensagens. Esta primitiva é denominada por primitiva modificada. Quando o processo utiliza esta primitiva para fazer a dispersão de uma mensagem para os n processos, a primitiva seleciona de forma aleatória $fanout$ processos para os quais as mensagens são enviadas de imediato. Os restantes $n - fanout$ processos são agrupados em conjuntos de tamanho $fanout$ e ordenados numa lista de forma a que o envio das respetivas mensagens é agendado para $(i + 1) * delay$ milissegundos mais tarde, onde i representa a posição do conjunto na lista. Os valores de $fanout$ e $delay$ são configuráveis e numa secção futura vão ser analisados com mais detalhe.

A primitiva descrita apenas retarda o envio das mensagens de forma a que a correção do algoritmo é assegurada, tal como foi demonstrado em trabalhos anteriores [15].

Esta primitiva ajuda a detetar mensagens que muito possivelmente serão ignoradas mas, tal como no algoritmo inicial, os processos ainda não têm ferramentas para o determinar. De forma a complementar esta primitiva, foi acrescentada uma lógica na receção das mensagens que guarda a ronda mais alta das mensagens recebidas por cada processo. Com esta informação, os processos podem descartar o envio de uma mensagem caso a ronda da mensagem seja inferior à ronda registada na estrutura de dados. Os atrasos configurados vão aumentar a probabilidade de haver mais mensagens a serem descartadas. No entanto, estes atrasos poderão aumentar o tempo de execução dos processos.

5

IMPLEMENTAÇÃO

Neste capítulo são descritas os características da implementação do algoritmo assim como as tecnologias utilizadas. Também é descrita a ferramenta utilizada para monitorizar os processos e de que forma podem são gerados os gráficos analisados nos capítulos anteriores.

Na ultima secção deste capítulo é descrita a metodologia utilizada para encontrar a parametrização correta para a primitiva de comunicação modificada.

5.1 AMBIENTE DE SIMULAÇÃO

O algoritmo estudado necessita durante a sua execução de uma rede de comunicação que assegure a entrega das mensagens enviadas pelos processos. Como o algoritmo é executado localmente, é necessária uma ferramenta que ofereça uma simulação da rede de comunicação de forma a ser possível modelar as características de um sistema real. Neste contexto, surge a ferramenta *minha4*. Esta ferramenta disponibiliza uma rede comunicação configurável assim como um ambiente de simulação onde os processos são executados. Estas características permitem fixar variáveis de ambiente de forma a focar o estudo nas variáveis do algoritmo.

A rede de comunicação é configurada de forma a que os processos reconheçam o emissor das mensagens de forma equivoca, mesmo na presença de processos bizantinos. A latência entre os processos é gerada através de uma distribuição uniforme que varia entre um a dois milissegundos. Os canais de comunicação são confiáveis, isto é, existe a garantia que uma mensagem enviada é inevitavelmente recebida pelo destinatário. Esta configuração dos canais de comunicação apenas surge para facilitar a implementação, pois irá facilitar a gestão dos processos faltosos, no entanto, esta característica não é uma restrição do algoritmo. Relativamente ao poder de processamento, cada processo é configurado de forma a não existirem diferenças no tempo de processamento de cada tarefa entre os vários processos do sistema.

5.2 IMPLEMENTAÇÃO DO ALGORITMO

O algoritmo está desenvolvido na linguagem Java e consiste num executável que é passado para o *minha4* que inicia e faz a gestão da sua execução. Cada executável representa um participante no decorrer do algoritmo.

Durante a execução, os processos necessitam de conhecer as constantes do problema em estudo. Para tal, os processos são inicializados com uma estrutura que contém essas constantes. Dentro essa estrutura, podem-se encontrar as constantes relativas ao número total de processos, ao número máximo de processos faltosos tolerados e ao valor de ϵ que assume por defeito o valor de 0.001.

Tendo acesso às constantes do problema os processos podem iniciar a sua execução. Para iniciar a votação, cada processo precisa que lhe seja atribuído um valor inicial. Este valor inicial é gerado segundo uma distribuição normal cuja média e desvio padrão estão também definidos na configuração inicial da execução. Por defeito, estas variáveis estão configuradas com o valor 100. Após determinar o seu valor inicial, cada processo envia-o para todos os participantes dando assim inicio ao algoritmo.

Para facilitar o desenvolvimento e abstrair a rede de comunicação da lógica do algoritmo, os processos utilizam uma primitiva de comunicação que lhes oferece uma interface que lhes permitem dispersar mensagens para todos os processos no sistema. Esta interface também permite aos processos bizantinos enviarem mensagens distintas para todos os processos.

Os processos durante a sua execução mantêm em memória estruturas de dados que representam o estado do processo. Nessas estruturas são colecionadas todas as mensagens recebidas relativas às rondas iguais ou superiores à atual ronda onde o processo se encontra. Também são colecionadas todas as mensagens de terminação recebidas pelo processo. Estas estruturas são essenciais para determinar o momento em que o processo progride para a ronda seguinte e para determinar o valor a propor para essa nova ronda.

Quando um processo alcança uma nova ronda e após determinar o seu novo valor, este é enviado para todos os participantes. Depois de fazer esse envio, o processo prioriza o processamento das mensagens que tem guardadas nas suas estruturas, nomeadamente as mensagens recebidas relativas à nova ronda alcançada assim como as mensagens de terminação relativas a uma ronda igual ou inferior à atual ronda do processo.

Quando o processo alcança a ronda final, cujo valor é determinado na primeira ronda, o processo assinala a sua terminação através de um tipo de mensagem reservada para o efeito e entra num estado de terminação.

5.3 PRIMITIVA DE COMUNICAÇÃO

A primitiva de comunicação oferece à implementação do algoritmo uma abstração da rede de comunicação que é utilizada. Para além disso, esta estrutura também é importante na medida em que é nela onde são aplicadas as alterações propostas.

Desta forma, são desenvolvidas duas primitivas de comunicação. A primeira primitiva, mais rudimentar, apenas dispersa as mensagens na rede de comunicação sem qualquer processamento das mesmas. Esta primitiva é denominada por primitiva original. Na figura 9 está representado o envio de uma mensagem através desta primitiva.

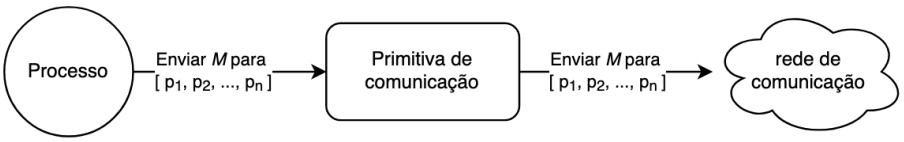


Figura 9: Representação do envio de uma mensagem através da primitiva oiginal.

A segunda primitiva desenvolvida aplica os atrasos anteriormente descritos. O envio de uma mensagem através desta primitiva está representado na figura 10 onde se pode observar a forma como os parâmetros *delay* e *fanout* afetam o funcionamento desta primitiva. Ainda nesta figura pode se observar o momento onde o envido das mensagens é cancelado e as mensagens descartadas.

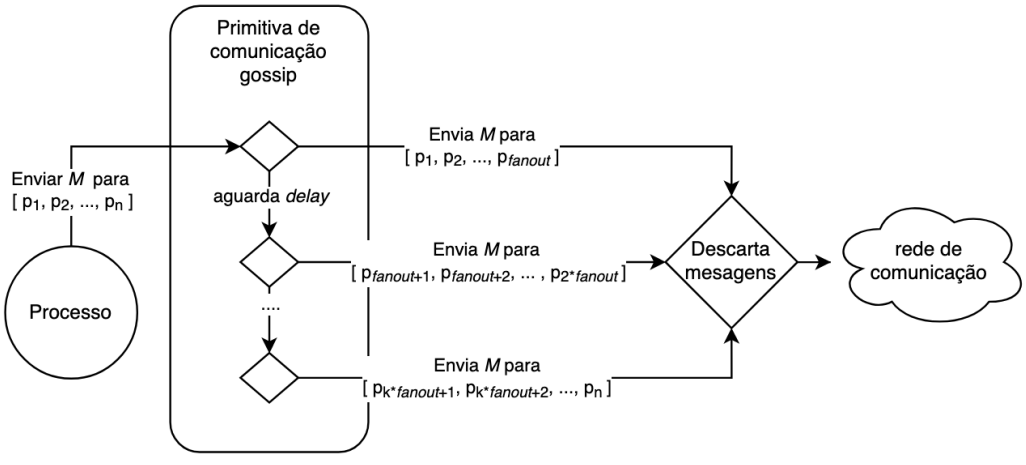


Figura 10: Representação do envio de uma mensagem através da primitiva modificada

5.4 MONITORIZAÇÃO

A ferramenta de monitorização é uma peça fundamental para possibilitar a analise do algoritmo. Através desta ferramenta são recolhidos eventos durante a execução dos processos para posteriormente serem analisados. Esta ferramenta é utilizada pelo algoritmo e pelas primitivas de comunicação que utilizaram a sua interface para registarem os seus principais eventos, tais como o alcance de uma nova ronda ou a entrada no estado de terminação. Após todos os processos assinalarem a sua terminação a ferramenta persiste os dados para disco.

Após a execução das simulações os dados podem ser analisados por uma ferramenta complementar à ferramenta de monitorização que carrega os dados e sumariza os resultados através de gráficos ou em ficheiros *csv*.

Para além de fazer a monitorização esta ferramenta tem mecanismos internos que garantem a integridade da execução garantindo assim, como, por exemplo, que em cada ronda existe um determinado número de processos faltosos ou que os valores iniciais obedecem a uma distribuição normal.

5.5 PARAMETRIZAÇÃO DA PRIMITIVA MODIFICADA

Com a implementação da primitiva modificada surgem dois parâmetros configuráveis. O primeiro trata-se do parâmetro $fanout$ que, como foi visto anteriormente, define o número total de processos aos quais as mensagens são enviadas de imediato, sendo para os restantes $n - fanout$ processos o seu envio agendado em conjuntos de tamanho $fanout$. O segundo parâmetro, denominado *delay*, define a diferença temporal entre cada agendamento.

Nesta secção é analisado o desempenho da primitiva modificada face à variação destes parâmetros. Esta análise é inicialmente feita para cada um dos parâmetros sendo concluída com uma análise global onde é apresentada a metodologia para encontrar a configuração mais apropriada para um sistema. Os resultados obtidos são analisados apenas segundo o número de mensagens ignoradas e o tempo médio de execução dos processos. Os restantes parâmetros não são considerados relevantes nesta análise na medida em que vão-se manter constantes ao longo das várias execuções.

Os resultados analisados ao longo desta dissertação são obtidos através da execução de simulações cuja implementação é descrita nas secções 5.1, 5.2 e 5.3.

5.5.1 *Configuração do sistema*

Nas simulações executadas ao longo deste capítulo é utilizada a mesma configuração do sistema de forma a tornar percetível o impacto das variáveis em estudo. Para esta análise,

é importante salientar que na configuração inicial estão definidos 71 processos onde são tolerados 14 processos faltosos. Os processos comunicam entre si através de uma rede de comunicação confiável onde nenhuma mensagem é perdida e em que os processos têm uma latência entre si gerada por uma distribuição linear que varia de 1 ate 2 milissegundos.

5.5.2 *Parâmetro delay*

Para analisar a variação do *delay*, o parâmetro *fanout* foi configurado com o valor de 10 unidades. Com esta configuração, pode-se observar no gráfico 11 que para valores superiores aos 1,5 milissegundos o tempo de processamento cresce proporcionalmente ao crescimento do valor do *delay*. O número de mensagens ignoradas mantém-se inalterado para valores do parâmetro inferiores a 1. Entre os valores 1 e 3 milissegundos o número de mensagens ignoradas sofre uma descida acentuada, estabilizando para valores superiores a 3 milissegundos nas 16000 mensagens.

Com isto, pode-se concluir que para valores muito baixos do parâmetro *delay* as otimizações podem não se manifestar. No entanto, se este valor for muito elevado o tempo de processamento dos processos é muito afetado não havendo melhorias significativas no número de mensagens ignoradas.

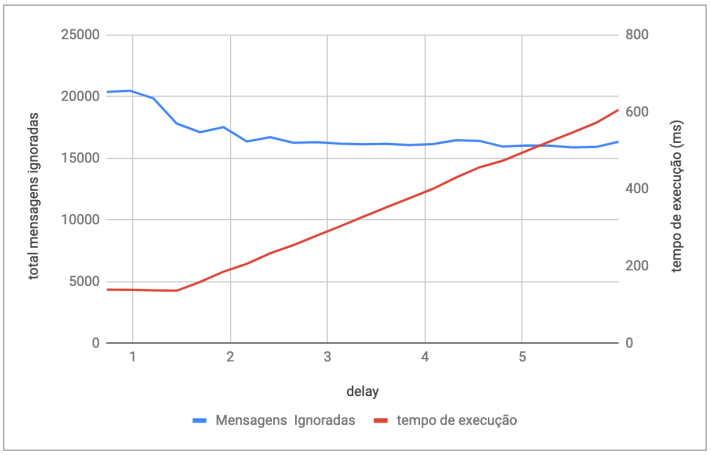


Figura 11: Variação do parâmetro *delay* com o parâmetro *fanout* igual a 10

5.5.3 *Parâmetro fanout*

Para analisar a variação do *fanout*, o parâmetro *delay* foi configurado com o valor de 2 milissegundos. Com esta configuração, pode-se observar no gráfico 12 que o tempo de execução é muito elevando quando o parâmetro é inferior a 10 unidades. Para valores superiores a 20 unidades o tempo de execução estabiliza em valores próximos dos 137

milissegundos. O número de mensagens ignoradas ronda as 16500 para valores inferiores a 10 unidades. Para valores entre as 10 e as 20 unidades o número de mensagens ignoradas sofre um aumento estabilizado nas 20000 a partir das 20 unidades.

Inversamente ao comportamento do parâmetro *delay*, para este parâmetro para valores muito altos as otimizações podem não se manifestar. No entanto, se este valor for muito baixo o tempo de processamento dos processos é muito afetado não havendo melhorias significativas no número de mensagens ignoradas.

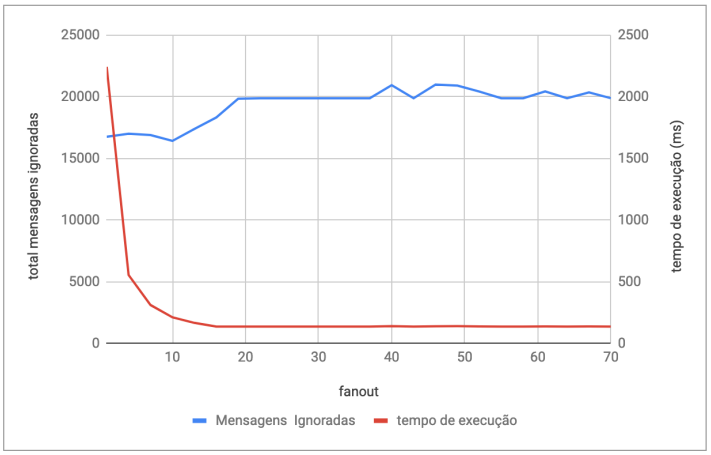


Figura 12: Variação do parâmetro $fanout$ com o parâmetro *delay* igual a 2 ms

5.5.4 *Analise dos resultados*

De forma a concluir este capitulo é feita uma análise aos parâmetros estudados de forma a encontrar a parametrização mais indicada para a configuração descrita.

Assim sendo, foi considerado um determinado conjunto de valores para o parâmetro *delay* que varia entre 0.1 e 120 milissegundos. Para cada valor deste conjunto, são executadas um conjunto de simulações onde o parâmetro $fanout$ varia de 1 até o número máximo de processos. Aos resultados destas simulações são selecionados dois valores para o parâmetro $fanout$. O primeiro corresponde ao valor para o parâmetro $fanout$ onde o número de mensagens ignoradas é minimizado para o parâmetro *delay* considerado. De forma similar, o segundo valor corresponde ao valor do parâmetro $fanout$ onde o tempo de execução médio dos processos é minimizado.

De forma a comparar os resultados obtidos com o algoritmo inicial, foram adicionados aos gráficos da figura 13 e 14 a representação dos resultados obtidos pelo algoritmo com primitiva original. Estes valores, representados nos gráficos com uma reta horizontal a tracejado, atingem cerca de 20000 mensagens ignoradas e um tempo de execução médio de 137 milissegundos.

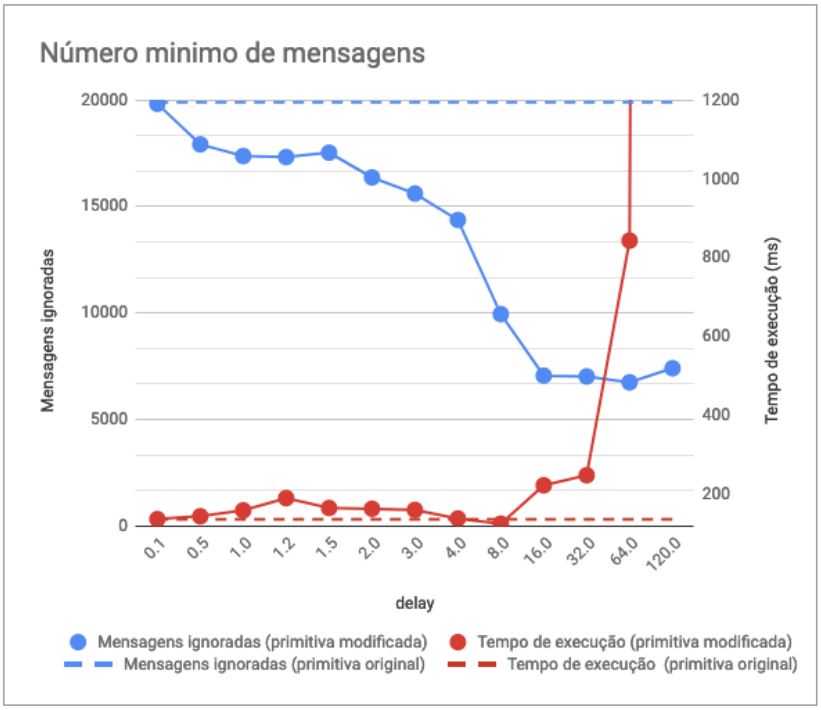


Figura 13: Resultados do algoritmo quando o número de mensagens ignoradas é minimizado face ao aumento do valor de *delay*

Na figura 13 onde estão representados os resultados obtidos para o valor *fanout* onde o número de mensagens ignoradas, pode ser observado que com o valor 8 milissegundos para o parâmetro *delay* é obtida uma redução de aproximadamente 50% do número de mensagens ignoradas face ao algoritmo com a a primitiva original e tendo ainda reduzido o tempo médio de execução em cerca de 9%. Estes valores são obtidos quando o parâmetro *fanout* está configurado a 55 unidades.

No figura 14 onde estão representados os resultados obtidos onde o tempo de execução é minimizado, pode ser observado que o número de mensagens ignoradas sofre uma diminuição com valores superiores a 0.5 milissegundos, no entanto, os melhores resultados são obtidos em valores do *delay* superiores a 8 milissegundos. Com 16 milissegundos configurados para o *delay* e 55 unidades para o parâmetro *fanout* a primitiva modificada reduz o número de mensagens ignoradas em aproximadamente 64%, no entanto, o tempo de processamento aumenta cerca de 62%.

Os dados que alimentam os gráficos apresentados podem ser analisados com mais detalhe na tabela 2.

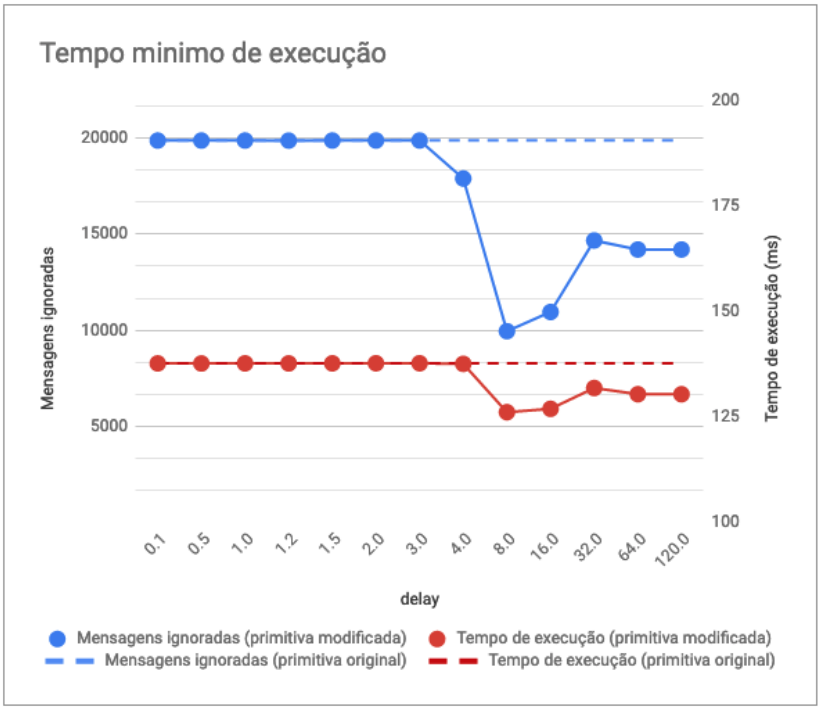


Figura 14: Resultados do algoritmo quando o tempo de execução dos processos é minimizado face ao aumento do valor de *delay*

	Tempo minimo de execução			Minimo de mensagens ignoradas		
Delay	Fanout	M. Ignoradas	Tempo de Exc.	Fanout	M. Ignoradas	Tempo de Exc.
0.1	10	19880	137,58	1	19813	137,86
0.5	7	19880	137,57	1	17912	145
1.0	22	19880	137,59	7	17363	160
1.2	13	19876	137,58	7	17314	191
1.5	19	19880	137,59	10	17522	166
2.0	28	19880	137,61	13	16358	163,94
3.0	40	19880	137,61	19	15604	161,11
4.0	31	17900	137,47	28	14366	139,42
8.0	55	9945	126,02	55	9945	126,02
16.0	64	10946	126,82	55	7061	223,47
32.0	67	14671	131,75	61	7023	248,94
64.0	67	14200	130,28	55	6747	843,37
120.0	67	14200	130,27	1	7407	24850,71

Tabela 2: Dados resultantes das simulações executadas para os dois cenários em estudo

6

ESCALABILIDADE

Neste capítulo são analisados os principais fatores de escalabilidade do algoritmo, nomeadamente a amplitude dos valores iniciais, o número total de processos e o número de processos faltosos tolerados. Esta análise irá incidir sobre o número total de mensagens no sistema, no número máximo de rondas executadas por todos os processos e no tempo médio de execução de cada processo. Estes fatores são analisados para o algoritmo sem as alterações propostas assim como o algoritmo com a primitiva modificada implementada de forma a analisar de uma forma mais generalizada as melhorias obtidas com estas alterações.

Para configurar a primitiva modificada foi utilizado o método descrito no capítulo anterior onde foi extraída a configuração onde o tempo de execução é minimizado. Desta forma, são obtidas melhorias ao nível do número total de mensagens ignoradas assim como para tempo de execução dos processos.

6.1 AMPLITUDE DOS VALORES INICIAS

Nesta secção é estudada o impacto da variação da amplitude dos valores iniciais dos processos. Tal como seria esperado, este aumento provoca um aumento do número de rondas a executar. No entanto, pode-se observar nos gráficos 15 que este a função descreve um crescimento logarítmico. Como seria esperado, este resultado é obtido para as duas implementações estudadas dado que as alterações propostas não afetam a determinação do número total de ronda.

Com o aumento do número de rondas observado na figura 16, o número total de mensagens enviadas e o tempo de execução sofrem um aumento na mesma proporção. Nestes indicadores já são observadas diferenças entre as implementações já que as alterações propostas promovem a diminuição do número de mensagens enviadas causando assim uma diminuição do tempo médio de processamento dos processos.

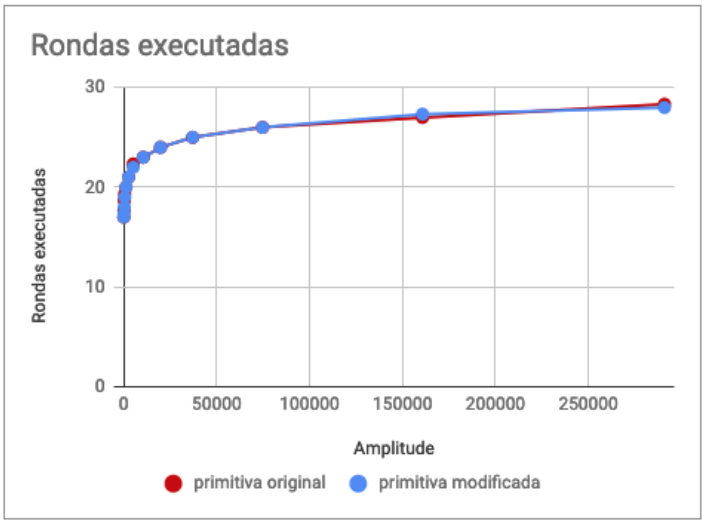


Figura 15: Número de rondas executadas face ao aumento da amplitude dos valores iniciais dos processos

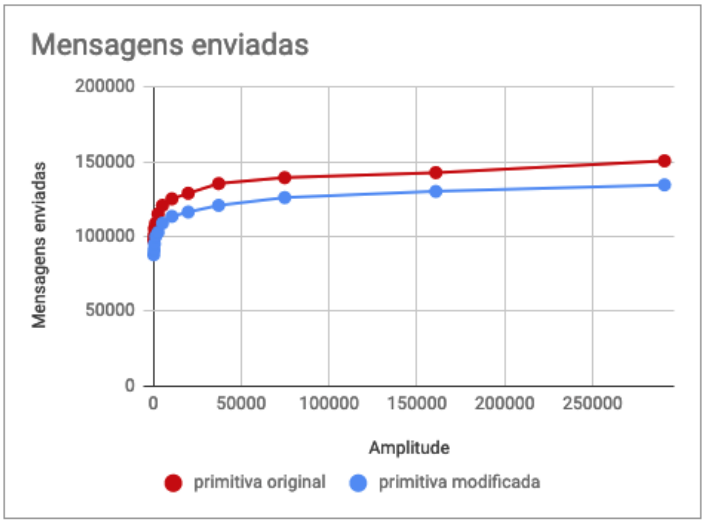


Figura 16: Número total de mensagens enviadas face ao aumento da amplitude dos valores iniciais dos processos

6.1.1 *Número de processos faltosos*

Nesta secção é estudado o aumento do número de processos faltosos onde o número total de processos é constante. Nos gráficos 17 e 18 podem ser observados os resultados desta variação num sistema com o total de 71 processos. No gráfico 17 pode ser observada uma tendência de crescimento linear do número de rondas executadas face ao crescimento

do número de processos faltosos. Este crescimento sofre um aumento significativo no último terço do gráfico. Este comportamento pode ser observado para outras configurações. Estes resultados são reflexo do comportamento da função de aproximação analisada em capítulos anteriores.

De igual forma à secção anterior, o número de rondas executadas é igual para as duas implementações. Ao nível de número de mensagens enviadas e tempo de execução podem ser observadas melhorias com a utilização da primitiva modificada.

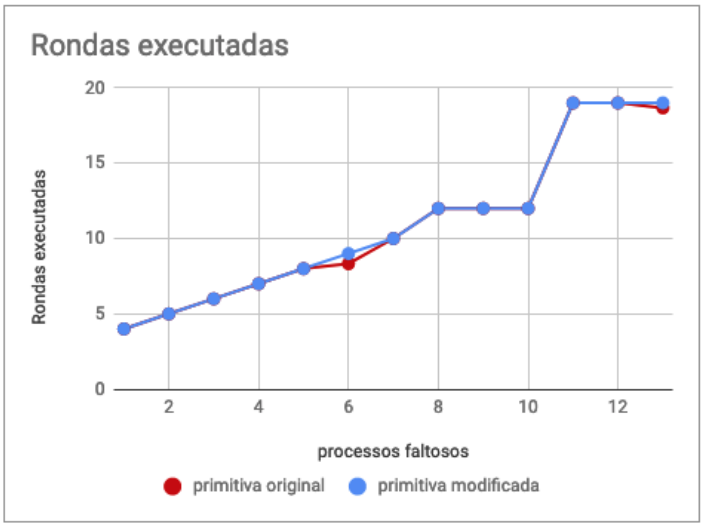


Figura 17: Número de rondas executadas face ao aumento do número de processos faltosos tolerados

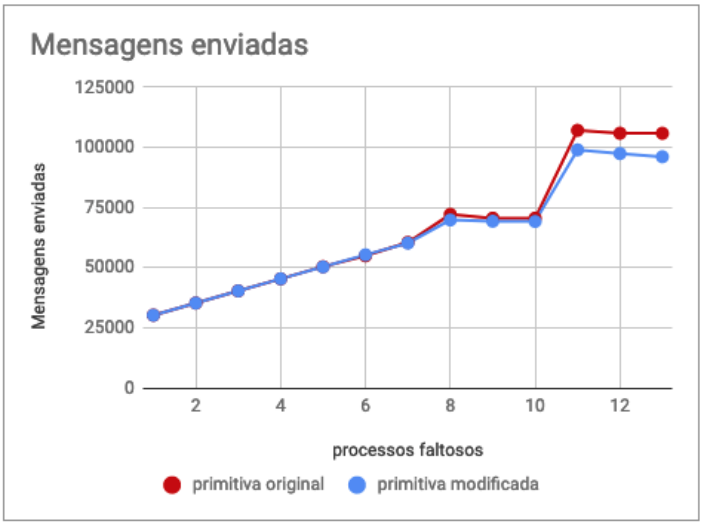


Figura 18: Número total de mensagens enviadas face ao aumento do número de processos faltosos tolerados

6.1.2 *Número de participantes*

Por fim, nesta secção é analisado o crescimento do número de participantes no sistema. Neste estudo o número de participantes n varia de 71 até 500 processos onde o número de processos faltosos tolerados f é maximizado para cada valor de n, sendo configurado de forma a que $n = 5f + 1$.

Nas simulações executadas, verifica-se que o número de rondas executadas não sofre variações significativas. Este resultado deve-se ao facto de que a amplitude dos valores iniciais e a razão entre o número total de participantes e o número de processos faltosos se manterem constantes ao longo das várias execuções.

Nos gráficos 19, 20 e 21 pode observar-se que para o algoritmo com a primitiva original o tempo de execução e o número de mensagens enviadas têm um comportamento exponencial em relação ao crescimento linear do número total de processos. No entanto, quando são aplicadas as alterações propostas, pode-se verificar que o número de mensagens enviadas descreve uma curva com um declive muito inferior ao algoritmo com a primitiva original. O tempo de execução dos processos com cresce também de forma exponencial, mas com uma taxa de crescimento inferior.

No gráfico 19 podem ser observadas uma diferença de 72% do número total de mensagens ignoradas quando são considerados um total de *301* processos. Este valor poderia ser melhorado, no entanto, essa melhoria iria trazer aumentos significativos no tempo médio de processamento tal como foi visto no capítulo anterior. Esta redução tem um impacto no número total de mensagens enviadas e no tempo médio de execução em cerca de 15%.

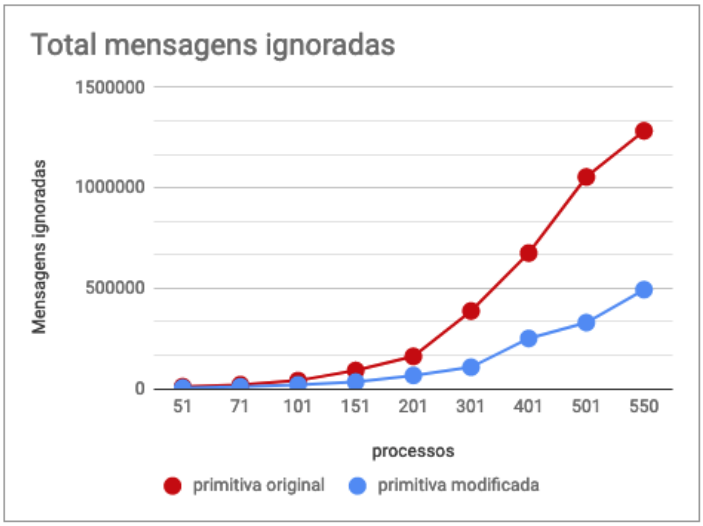


Figura 19: Número total de mensagens ignoradas face ao aumento do número total de processos

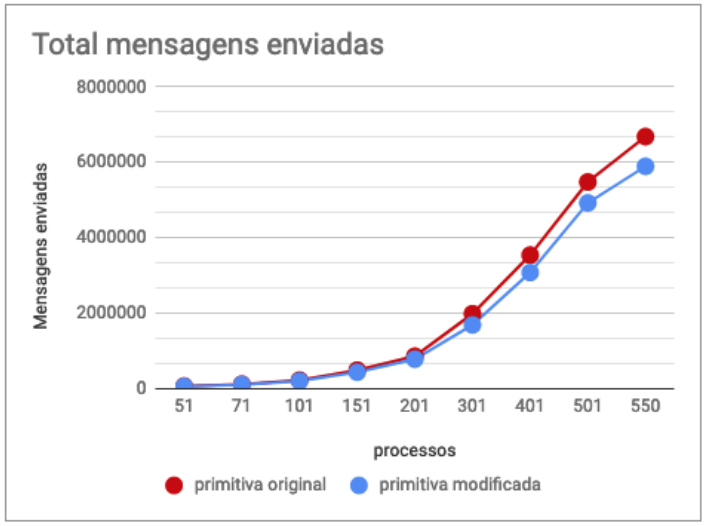


Figura 20: Número total de mensagens enviadas face ao aumento do número total de processos

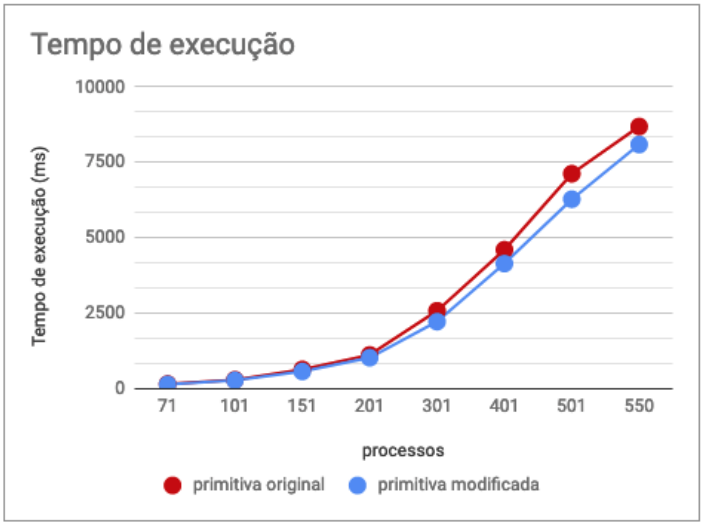


Figura 21: Tempo de execução médio dos processos face ao aumento do número total de processos

7

CONCLUSÃO

De forma a concluir esta dissertação, neste capítulo é feita uma análise ao trabalho realizado fazendo uma visão geral dos resultados obtidos. Após esta reflexão é feita uma proposta para continuação do trabalho desenvolvido.

Após o problema em estudo ser bem definido e feita a contextualização dos conceitos essenciais para os capítulo subsequentes, foi feito um levantamento dos algoritmos que se propõem a resolver o problema do acordo distribuído aproximado nos diversos modelos apresentados. Este estudo foi finalizado com uma comparação da modelo performance de cada um através dos fatores de escalabilidade enumerados.

Tendo em consideração as características dos algoritmos estudados, foi selecionado um algoritmo da classe MSR como ponto de partida para obter um algoritmo mais escalável, sendo detalhada a análise de escalabilidade do mesmo, bem como o conjunto de alterações propostas.

De seguida, foram descritos os detalhes da implementação do algoritmo e das duas primitivas. No mesmo capítulo foi apresentada a metodologia para determinar a melhor parametrização da primitiva modificada. Esta metodologia foi finalizada com um caso de estudo concreto onde podem ser observadas reduções até aproximadamente 77% do número total de mensagens ignoradas e de até cerca de 10% do tempo de processamento médio dos processos.

Por último, a metodologia descrita foi utilizada para analisar a performance do algoritmo face ao aumento dos seus fatores de escalabilidade. Nesta análise foi feita uma comparação direta entre as duas implementações onde podem ser observadas reduções até 15% no número total de mensagens enviadas causada por uma redução na ordem dos 72% do número de mensagens ignoradas. Ainda nestes resultados foram observadas reduções na ordem dos 15% do tempo médio de execução dos processos.

Como este estudo foi realizado localmente recorrendo a uma ferramenta que simula um ambiente de execução assim como a rede de comunicação, numa próxima etapa desta dissertação poderia-se aplicar o algoritmo num ambiente real. Com a arquitetura da primitiva de comunicação desenvolvida torna-se fácil adaptar o algoritmo desenvolvido para um sistema real. Ainda no espaço de melhorias ao trabalho desenvolvido, poderá ser

explorada também uma outra forma de obter reduções no número de mensagens, através de, por exemplo, alterações da lógica no envio de mensagens de forma a que não ofereçam garantias tão fortes, mas que se obtenha reduções mais significativas.

BIBLIOGRAFIA

[1] Ittai Abraham, Yonatan Amit, and Danny Dolev. Optimal resilience asynchronous approximate agreement. In *International Conference On Principles Of Distributed Systems*, pages 229–239. Springer, 2004.

[2] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, 43(2):225–267, 1996.

[3] Danny Dolev, Nancy A Lynch, Shlomit S Pinter, Eugene W Stark, and William E Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM (JACM)*, 33(3):499–516, 1986.

[4] Alan David Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4(1):9–29, 1990.

[5] Alan David Fekete. Asynchronous approximate agreement. *Information and Computation*, 115(1):95–124, 1994.

[6] Michael J Fischer. The consensus problem in unreliable distributed systems (a brief survey). In *International conference on fundamentals of computation theory*, pages 127–140. Springer, 1983.

[7] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[8] Rachid Guerraoui and André Schiper. Consensus service: a modular approach for building agreement protocols in distributed systems. In *Proceedings of Annual Symposium on Fault Tolerant Computing*, pages 168–177. IEEE, 1996.

[9] Roger M. Kieckhafer and Mohammad H. Azadmanesh. Reaching approximate agreement with mixed-mode faults. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):53–63, 1994.

[10] Leslie Lamport and P Michael Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM (JACM)*, 32(1):52–78, 1985.

[11] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport*, pages 203–226. 2019.

[12] Jennifer Lundelius and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 75–88, 1984.

[13] Romualdo Pastor-Satorras and Alessandro Vespignani. Epidemic spreading in scale-free networks. *Physical review letters*, 86(14):3200, 2001.

[14] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[15] José Pereira and Rui Oliveira. The mutable consensus protocol. In *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, pages 218–227. IEEE, 2004.

[16] Kenneth J Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, (3):477–482, 1986.

[17] Richard Plunkett and Alan Fekete. Optimal approximate agreement with omission faults. In *International Symposium on Algorithms and Computation*, pages 468–475. Springer, 1998.

[18] Fred B Schneider. What good are models and what models are good. *Distributed systems*, 2:17–26, 1993.

[19] Jennifer Lundelius Welch and Nancy Lynch. A new fault-tolerant algorithm for clock synchronization. *Information and computation*, 77(1):1–36, 1988.

[20] Matthias Wiesmann, Fernando Pedone, André Schiper, Bettina Kemme, and Gustavo Alonso. Understanding replication in databases and distributed systems. In *Proceedings 20th IEEE International Conference on Distributed Computing Systems*, pages 464–474. IEEE, 2000.