Uma Solução Autonômica para a Criação de Quóruns Majoritários Baseada no VCube

Luiz A. Rodrigues[1,2], Elias P. Duarte Jr.[2] e Luciana Arantes[3]

[1] Colegiado de Ciência da Computação – Universidade Estadual do Oeste do Paraná
Caixa Postal 801 – 85819-110 – Cascavel – PR – Brasil

[2]Departamento de Informática – Universidade Federal do Paraná
Caixa Postal 19.081 – 81531-980 – Curitiba – PR – Brasil

[3]Laboratoire d'Informatique - Université Pierre et Marie Currie
CNRS/INRIA/REGAL – Place Jussieu, 4 – 75005 – Paris, France

luiz.rodrigues@unioeste.br, elias@inf.ufpr.br, luciana.arantes@lip6.fr

Abstract. *This paper presents an autonomic solution for building and maintaining quorums in distributed systems. The processes are organized in a virtual hypercube topology called VCube. The topology is constructed dynamically based on the information obtained from a system monitoring. Faulty process are removed from the system and a new configuration is created using only the correct processes. Processes can fail by crash and a failure is permanent. The quorum size and load generated are well distributed and the reconfiguration is automatic, tolerating up to $n-1$ faulty processes. The proposed system was compared with a solution using a binary tree. Experimental results confirm the resilience and stability of quorums in the VCube.*

Resumo. *Este trabalho apresenta uma solução autonômica para a construção e manutenção de quóruns em sistemas distribuídos. Os processos são organizados em uma topologia virtual de hipercubo chamada VCube. A topologia é construída dinamicamente com base nas informações de falhas obtidas de um sistema de monitoramento. Processos falhos são eliminados do sistema e uma nova configuração é criada utilizando apenas os processos corretos. Os processos podem falhar por crash e uma falha é permanente. Os quóruns gerados possuem tamanho e carga bem distribuídos e a reconfiguração é automática, tolerando até $n-1$ processos falhos. O sistema proposto foi comparado com uma solução em árvore binária. Resultados experimentais confirmam a resiliência e estabilidade dos quóruns no VCube.*

1. Introdução

Sistemas de quóruns foram introduzidos por Thomas (1979) como uma solução para coordenar ações e garantir consistência em um sistema de banco de dados replicado. Considere um sistema distribuído como um conjunto finito P de $n > 1$ processos independentes $\{p_0, ..., p_{n-1}\}$ que se comunicam usando troca de mensagens. Um *sistema de quóruns* em P é um conjunto de subconjuntos de P, chamados *quóruns*, no qual cada par de subconjuntos tem uma intersecção não-vazia [Merideth e Reiter 2010]. Garcia-Molina e Barbara

(1985) estenderam esta definição introduzindo o conceito de *coteries*, isto é, grupos exclusivos. Este modelo inclui a propriedade de minimalidade, na qual nenhum conjunto contém outro conjunto do sistema, o que o torna mais eficiente.

O algoritmo proposto por Thomas implementa um mecanismo tradicional de votação no qual o vencedor é o valor com a maioria dos votos. Cada atualização em um item de dados é marcada com um *timestamp* e executada em um quórum composto pela maioria dos servidores. Quando um cliente precisa ler um valor do banco, ele acessa novamente um quórum com a maioria das réplicas e escolhe o item com o *timestamp* mais atual. Em um sistema com três servidores, por exemplo, se a escrita for replicada em quaisquer dois servidores, toda leitura posterior em no mínimo dois servidores retornará o valor mais atual, isto é, a intersecção dos quóruns garante que, no mínimo, uma replica terá o item de dado mais recente. O trabalho de Thomas considera que cada processo contribui de forma igualitária na votação. Gifford (1979) estendeu este modelo atribuindo pesos aos processos. Além disso, os quóruns são divididos em duas classes, leitura e escrita, e somente quóruns de classes diferentes precisam se intersectar.

Além da replicação de dados [Liu et al. 2011, Abawajy e Mat Deris 2013], sistemas de quórum têm sido utilizados por diversas outras aplicações, como exclusão mútua [Maekawa 1985, Fujita 1998, Atreya et al. 2007, Naimi e Thiare 2013], comunicação em grupo [Tanaka et al. 1999], controle de acesso seguro [Naor e Wool 1996], e sistemas tolerantes a falhas bizantinas [Liskov e Rodrigues 2006, Merideth e Reiter 2007, Merideth e Reiter 2008]. Por exemplo, em uma solução de exclusão mútua baseada em pedidos de permissão sem a utilização de quóruns, se um processo deseja obter acesso a um recurso compartilhado, ele deve enviar uma mensagem de requisição para *todos* os outros processos do sistema e aguardar pelas mensagens de permissão [Ricart e Agrawala 1981]. Neste caso, um sistema de quórum pode reduzir o total de mensagens, visto que este mesmo processo precisa enviar a requisição apenas para um subconjunto dos processos que forma um quórum. Os processos que respondem ao pedido de permissão ficam bloqueados para novos pedidos até que o recurso seja liberado. A intersecção entre os quóruns garante a integridade da exclusão mútua.

Em geral, as soluções de quóruns são avaliadas em termos de tamanho, carga e disponibilidade [Vukolić 2010]. O tamanho é o número de processos em cada quórum e a carga indica em quantos quóruns cada processo está contido. A disponibilidade está ligada ao impacto das falhas no conjunto de quóruns do sistema. O modelo de Maekawa (1985), por exemplo, gera quóruns de tamanho próximo a $\sqrt{n}$, sendo n o número de processos no sistema. No entanto, a solução não pode ser aplicada para qualquer valor de n. Agrawal e El-Abbadi (1991) utilizam árvores binárias para criar quóruns de tamanho $\log_2 n$, mas que podem chegar a $\lceil (n+1)/2 \rceil$ em cenários com falhas. Além disso, a carga é altamente concentrada na raiz e proporcionalmente nos nodos próximos a ela.

A solução de Thomas faz uso dos chamados quóruns majoritários. Um quórum majoritário Q em P tem $|Q| = \lceil (n+1)/2 \rceil$ elementos. A grande vantagem deste tipo de quórum é a alta disponibilidade. Mesmo em uma solução estática, os quóruns majoritários toleram $f < n/2$ processos falhos [Lipcon 2012]. A tolerância a falhas pode ser ainda aprimorada se o modelo permite a atualização dinâmica do sistema [Amir e Wool 1996]. Além disso, embora outras soluções apresentem modelos teóricos com propriedades otimizadas, as soluções majoritárias continuam sendo as mais utilizadas na prática.

Este trabalho apresenta um sistema de quórum majoritário construído sobre uma topologia de hipercubo virtual chamada VCube. Assume-se um sistema representado por um grafo completo no qual os enlaces são confiáveis. VCube organiza os processos do sistema em clusters progressivamente maiores formando um hipercubo quando não há processos falhos. Processos podem falhar por *crash* e uma falha é permanente. Após uma falha, a topologia do VCube é reestruturada estabelecendo-se novos enlaces entre os processos anteriormente conectados ao processo falho. As propriedades logarítmicas herdadas do hipercubo são mantidas mesmo após a ocorrência de falhas. A avaliação teórica e testes comparativos mostram que os quóruns formados possuem tamanho e carga balanceados e o sistema tolera até $n - 1$ processos falhos.

O restante do texto está organizado nas seguintes seções. A Seção 2 discute os trabalhos correlatos. A Seção 3 apresenta as definições básicas e o modelo do sistema. A Seção 4 descreve o algoritmo de geração de quóruns proposto. Uma avaliação experimental é apresentada na Seção 5. A Seção 6 apresenta a conclusão e os trabalhos futuros.

2. Trabalhos Correlatos

Desde as soluções baseadas em votação propostas por Thomas (1979) e Gifford (1979), muitos outros modelos para a construção de quóruns têm sido apresentados na literatura. Nos modelos estáticos, isto é, aqueles em que a quantidade de processos é fixa e conhecida, o tratamento de falhas normalmente é baseado na substituição de elementos falhos do quórum ou na redefinição dos conjuntos após a falha.

Barbara e Garcia-Molina (1986) propuseram algumas heurísticas para diminuir a vulnerabilidade dos sistemas baseados em votação quando há possibilidade de falhas de processo ou de enlace. Em Barbara et al. (1989) uma solução autonômica de redistribuição de votos é apresentada para o problema da exclusão mútua. Após a ocorrência de falhas, mesmo em caso de particionamento, os votos do processo falho são redistribuídos entre os processos corretos que fazem parte do grupo que contém a maioria.

Maekawa (1985) utilizou planos projetivos finitos para criar quóruns com $c\sqrt{n}$ elementos, $3 \leq c \leq 5$. Planos projetivos são estruturas geométricas derivadas do plano nas quais quaisquer duas linhas se intersectam em apenas um ponto. Entretanto, estes planos existem somente se $n = p^i$, sendo p um número primo e i um inteiro. Para os casos em que não há plano, duas outras soluções são propostas pelo autor, mas com desempenho não balanceado e quóruns de tamanho maior que $\sqrt{n}$. Uma delas organiza o sistema em uma grade de tamanho $l \times l$ com l^2 elementos, na qual cada quórum q_i é composto por todos os elementos da linha e da coluna que cruzam em i. Cada quórum possui $2l - 1$ integrantes e cada par de quóruns possui uma intersecção de, no mínimo, dois elementos. Kumar e Agarwal (2011) apresentaram um generalização para grades multidimensionais que permite a reconfiguração dos quóruns, visando aumentar a disponibilidade do sistema.

Uma solução semelhante às grades organiza os elementos do sistema em um triângulo de tamanho $r \times r$, $r \geq 2$. A quantidade de elementos de cada linha aumenta de 1 até r. Cada quórum é formado por uma linha i completa e por um representante de cada linha $j > i$. Preguiça e Martins (2001) propuseram um sistema de quórum hierárquico com base nesta arquitetura. Cada quórum é formado pela divisão do triângulo em subtriângulos e pela união dos quóruns de cada unidade. Agrawal e El-Abbadi (1991) utilizam uma árvore binária lógica para criar quóruns de tamanho $\lceil \log_2 n \rceil$ a $\lceil (n+1)/2 \rceil$. Cada

quórum é formado por um caminho da raiz até uma folha. Se um nodo neste caminho está falho, o quórum é construído pela junção de um caminho da sub-árvore direita com um caminho da sub-árvore esquerda do nodo falho. No pior caso, um quórum majoritário é formado. Considerando que cada caminho termina em um nodo folha, um quórum não pode ser formado se o caminho termina em um nodo falho. Neste caso o algoritmo retorna uma condição de erro. Uma generalização desta solução foi apresentada por Agrawal e El-Abbadi (1992) para permitir a construção de quóruns com base em árvores arbitrárias.

Uma abordagem híbrida utilizando árvore e grade é apresentada em [Choi e Youn 2012]. Para aumentar a disponibilidade do sistema, a altura da árvore, a quantidade de descendentes e a profundidade da grade são ajustados dinamicamente.

Guerraoui e Vukolić (2010) introduziram a noção de sistemas de quóruns refinados (*refined quorum systems*). Neste modelo, três classes de quóruns são definidas. Os quóruns de primeira classe tem uma grande intersecção com os outros quóruns. Os quóruns de segunda classe tem uma menor intersecção com os quóruns de terceira classe. Estes por sua vez, correspondem aos quóruns tradicionais.

Malkhi et al. (2001) apresentam o conceito de sistema de quórum probabilístico. Em um sistema com n processos, um quórum probabilístico contém $k\sqrt{n}$ processos escolhidos de maneira aleatória e uniforme, sendo k o parâmetro de confiabilidade. De forma semelhante, Abraham (2005) aplicou quóruns probabilísticos em sistemas dinâmicos. De acordo com o autor, quóruns probabilísticos otimizam a carga e a disponibilidade, flexibilizando a propriedade de intersecção, isto é, os quóruns se intersectam com uma probabilidade de acordo com uma determinada distribuição.

3. Definições, Modelo do Sistema e a Topologia Virtual VCube

3.1. Definições

Um sistema distribuído consiste de um conjunto finito P com $n > 1$ processos $\{p_0, .., p_{n-1}\}$. Cada processo executa uma tarefa e está alocado em um nodo distinto. Assim, os termos *nodo* e *processo* são utilizados com o mesmo sentido.

Definição 1 (Quorum)**.** *Seja $P = \{p_0, .., p_{n-1}\}$ um sistema distribuído com $n \geq 1$ processos e $Q = \{q_1, .., q_m\}$ um conjunto de grupos de processos de P tal que $Q \neq \emptyset$, e $Q \subseteq P$. Q é um sistema de quórum se, e somente se, $\forall q_i, q_j \in Q$, $q_i \cap q_j \neq \emptyset$ (propriedade de intersecção).*

Definição 2 (*Coterie*)**.** *Um quórum Q é uma coterie em P se, e somente se, $\forall i, j$ não existe $q_i, q_j \in Q$ tal que $q_i \subset q_j$ (propriedade de minimalidade).*

O modelo de *coteries* garante que existem apenas conjuntos mínimos no sistema, isto é, o tamanho de qualquer conjunto é mínimo porque não existe nenhum conjunto menor contido nele.

O **tamanho** do quórum é a quantidade de processos contida nele. A **carga** está relacionada ao número de quóruns em que cada processo está presente. Um sistema de quóruns é dito s-uniforme se cada quórum possui exatamente s elementos. A carga está relacionada ainda à qualidade do sistema de quórum e pode ser associada tanto aos processos individualmente quanto ao sistema como um todo.

Em termos de **disponibilidade**, duas medidas são empregadas aos quóruns: resiliência e probabilidade de falhas. Um quórum é t-resiliente se, mesmo após a falha

de t processos quaisquer, existe ao menos um quórum $q \in Q$ tal que nenhum processo pertencente a q falhou. Neste caso, ao menos um quórum sempre poderá ser ativado. A resiliência máxima é $\lfloor (n-1)/2 \rfloor$, proveniente dos quóruns majoritários. A probabilidade de falhas é a probabilidade de que ao menos um processo em cada quórum esteja falho, isto é, nenhum quórum possui um processo sem falha [Naor e Wool 1998]. De acordo com a propriedade de intersecção, se todos os processos em um dado quórum falham, ao menos um processo falho estará presente em todos os demais quóruns. No entanto, estes limites são especificados para quóruns estáticos, isto é, aqueles que não se adaptam após a ocorrência de falhas.

3.2. Modelo do Sistema

Comunicação. Os processos se comunicam através do envio e recebimento de mensagens. A rede é representada por um grafo completo, isto é, cada par de processos está conectado diretamente por enlaces ponto-a-ponto bidirecionais. No entanto, processos são organizados em uma topologia de hipercubo virtual, chamada VCube. Em um hipercubo de d dimensões (d-cubo) cada processo é identificado por um endereço binário i_{d-1}, i_{d-2}, i_0. Dois processos estão conectados se seus endereços diferem em apenas um bit. Se não existem processos falhos, VCube é um hipercubo completo. Após uma falha, a topologia do VCube é modificada dinamicamente. O VCube é descrito com mais detalhes na Seção 3.3 abaixo. As operações de envio e recebimento são atômicas. Enlaces são confiáveis, garantindo que as mensagens trocadas entre dois processos nunca são perdidas, corrompidas ou duplicadas. Falhas de particionamento da rede não são tratadas.

Modelo de Falhas. O sistema admite falhas do tipo *crash* permanente. Um processo que nunca falha é considerado *correto* ou *sem-falha*. Caso contrário ele é considerado *falho*. Falhas são detectadas por um serviço de detecção de falhas perfeito, isto é, nenhum processo é detectado como falho antes do início da falha e todo processo falho é detectado por todos os processos corretos em um tempo finito [Freiling et al. 2011].

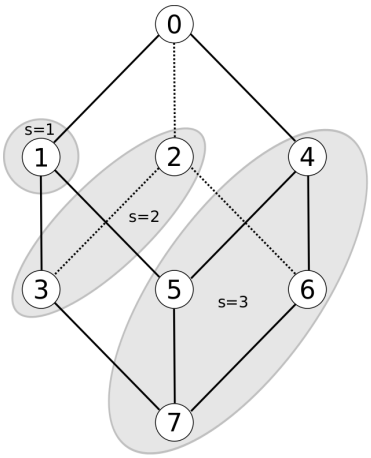
3.3. A Topologia Virtual VCube

O hipercubo virtual utilizado neste trabalho, denominada VCube, é criado e mantido com base nas informações de diagnóstico obtidas por meio de um sistema de monitoramento de processos descrito em Ruoso (2013). Cada processo que executa o VCube é capaz de testar outros processos no sistema para verificar se estão corretos ou falhos. Para isso, o processo executa um procedimento de teste e aguarda por uma resposta. Um processo é considerado correto ou sem-falha se a resposta ao teste for recebida corretamente dentro do intervalo de tempo esperado. Os processos são organizados em clusters progressivamente maiores. Cada cluster $s = 1, .., \log_2 n$ possui 2^s elementos, sendo n o total de processos no sistema. Os testes são executados em rodadas. Para cada rodada um processo i testa o primeiro processo sem-falha j na lista de processos de cada cluster s e obtém informação sobre os processos naquele cluster.

Os membros de cada cluster s e a ordem na qual eles são testados por um processo i são dados pela lista $c_{i,s}$, definida a seguir. O símbolo $\oplus$ representa a operação binária de OU exclusivo (XOR):

$$c_{i,s} = (i \oplus 2^{s-1}, c_{i \oplus 2^{s-1},1}, ..., c_{i \oplus 2^{s-1},s-1}) \quad (1)$$

A Figura 1 exemplifica a organização hierárquica dos processos em um hipercubo de três dimensões com $n = 8$ elementos. A tabela da direita apresenta os elementos de cada cluster $c_{i,s}$. Como exemplo, na primeira rodada o processo p_0 testa o primeiro processo no cluster $c_{0,1} = (1)$ e obtém informações sobre o estado do processo p_1. Em seguida, p_0 testa o processo p_2, que é primeiro processo no cluster $c_{0,2} = (2, 3)$, e obtém informações sobre p_2 e p_3. Por fim, p_0 executa testes no processo p_4 do cluster $c_{0,3} = (4, 5, 6, 7)$ e obtém informações sobre os processos p_4, p_5, p_6 e p_7.



s	$c_{0,s}$	$c_{1,s}$	$c_{2,s}$	$c_{3,s}$	$c_{4,s}$	$c_{5,s}$	$c_{6,s}$	$c_{7,s}$
1	1	0	3	2	5	4	7	6
2	2 3	3 2	0 1	1 0	6 7	7 6	4 5	5 4
3	4 5 6 7	5 4 7 6	6 7 4 5	7 6 5 4	0 1 2 3	1 0 3 2	2 3 0 1	3 2 1 0

Figura 1. Organização Hierárquica do VCube.

4. O Sistema de Quórum Proposto

Esta seção apresenta a abordagem para construção dos clusters hierárquicos baseados na topologia virtual em hipercubo do VCube. Inicialmente são definidas algumas funções e, em seguida, o algoritmo proposto.

4.1. Definição das Funções

Com base na organização virtual dos processos utilizada pelo VCube e na função $c_{i,s}$ apresentados na Seção 3.3, foram definidas algumas funções para auxiliar a implementação do algoritmo de quóruns proposto neste trabalho.

Inicialmente, o conjunto $correct_i$ é definido para registrar as informações que o processo i tem sobre o estado dos processos informadas pelo VCube. Estas informações são obtidas através dos testes realizados pelo monitoramento. Sendo assim, em razão da latência do detector, é possível que um processo j falho ainda seja considerado como correto pelo processo i. No entanto, tão logo o processo i seja informado sobre a falha, j é removido de $correct_i$ e, após o diagnóstico completo, todos os processos terão a mesma informação sobre processos corretos e falhos.

Seja o resultado da função $c_{i,s}$ uma lista $(a_1, a_2, .., a_m)$, $m = 2^{s-1}$ dos processos do cluster s em relação ao processo i. A lista de m' elementos considerados corretos por i em um cluster $c_{j,s}$ é definida como $FF\_cluster_i(s) = (b_1, .., b_{m'}), b_k \in (c_{i,s} \cap correct_i)$, $k = 1..m'$, $m' \leq m$. Desta forma, a função $FF\_mid_i(s) = (b_1, .., b_{\lceil m'/2 \rceil})$ gera um conjunto que contém a metade absoluta dos processos considerados corretos pelo processo i em um cluster $c_{i,s}$. Se não existem processos corretos no cluster s, $FF\_cluster_i(s) = \perp$ e, por conseguinte, $FF\_mid_i(s) = \perp$. Note que esta função é dependente do conhecimento

atual que um processo tem a respeito das falhas no sistema. Devido à latência de detecção, em um mesmo espaço de tempo, é possível que dois processos possuam visões diferentes sobre quais processos estão corretos ou falhos.

4.2. Descrição do Algoritmo

Em termos gerais, cada processo i constrói o seu próprio quórum adicionando a si mesmo e a metade absoluta dos elementos j que ele considera sem-falha ($j \in correct_i$) em cada cluster $c_{i,s}$ do hipercubo virtual definido por VCube. O estado dos processos é informado pelo VCube. O Algoritmo 1 apresenta um pseudo-código desta solução.

Algoritmo 1 Obtenção do quórum de um processo j pelo processo i

```
1: correct_i ← {0, .., n − 1}
2: function GETQUORUM
3:     q_i ← {i}
4:     for s ← 1, .., log_2 n do
5:         q_i ← q_i ∪ FF_mid_i(s)
6:     return q_i
7: upon notifying CRASH(j)
8:     correct_i ← correct_i ∖ {j}
```

Como exemplo, considere o cenário sem falhas representado pelo hipercubo de três dimensões da Figura 2(a), que representa os quóruns dos processos p_0 e p_7. Estes processos foram escolhidos por estarem o mais distante possível um do outro no hipercubo. O quórum q_0 calculado pelo processo p_0 é composto por ele mesmo e pela metade absoluta dos clusters $c_{0,1} = (1)$, $c_{0,2} = (2, 3)$ e $c_{0,3} = (4, 5, 6, 7)$. Neste caso, $FF\_mid_0(1) = (1)$, $FF\_mid_0(2) = (2)$ e $FF\_mid_0(3) = (4, 5)$. Logo, $q_0 = \{0, 1, 2, 4, 5\}$. O quórum q_7 referente ao processo p_7 é formado por $FF\_mid_7(1) = (6)$, $FF\_mid_7(2) = (5)$ e $FF\_mid_7(3) = (3, 2)$. Assim, $q_7 = \{7, 6, 5, 3, 2\}$. Logo, a intersecção, $q_0 \cap q_7 = \{2, 5\}$. A Figura 2(b) ilustra um cenário após a falha dos processos da interseção p_2 e p_5. Neste caso, a topologia virtual é reestruturada e o processo p_6 passa a fazer parte do quórum de p_0 pois, dada a falha de p_5, $FF\_mid_0(3) = (4, 6)$. O processo p_2 é incluído no quórum de p_7 por razões semelhantes. Assim, p_1 e p_6 passam a compor a nova interseção.

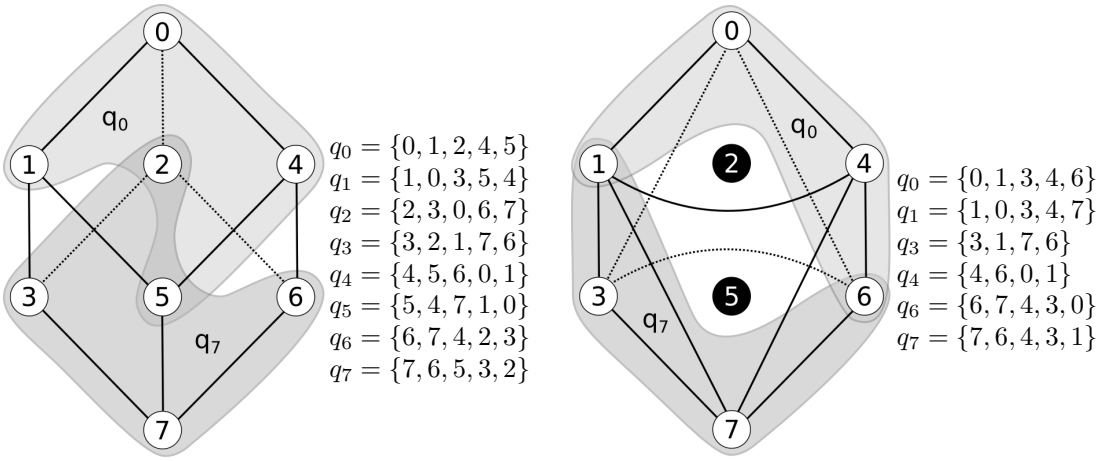
Teorema 1 (Intersecção)**.** *Todo quórum q_i construído por um processo i utilizando o Algoritmo 1 tem uma interseção com cada outro quórum do sistema de, no mínimo, dois elementos.*

Prova. A prova é por indução baseada nos clusters do VCube.

Base: o teorema é valido para um VCube de dimensão 1 com dois processos $\{p_0, p_1\}$. O processo p_0 adiciona a si mesmo e o processo p_1 ao seu quórum, visto que $c_{0,1} = (1)$ e $FF\_mid_0(1) = (1)$. O processo p_1 adiciona p_0 ao seu quórum de forma semelhante. Neste caso, $q_0 \cap q_1 = \{0, 1\}$.

Hipótese: suponha que o teorema é válido para um VCube de d dimensões.

Passo: considere um VCube com $d + 1$ dimensões com $n = 2^{d+1}$ processos. Cada cluster $c_{i,s}$ tem $2^s/2$ elementos. A soma dos elementos de todos os clusters s de um processo i é



(a) Sem falhas (b) Processos p_2 e p_5 falhos

Figura 2. Quóruns do VCube de 3 dimensões com representação gráfica para os processos p_0 e p_7.

dada por $2^1/2 + .. + 2^d/2 + 2^{d+1}/2 = 2^{d+1} - 1 = n - 1$. Portanto, a soma das metades absolutas dos elementos de cada cluster s é dada por:

$$\left\lceil \frac{\frac{2^1}{2} + .. + \frac{2^d}{2} + \frac{2^{d+1}}{2}}{2} \right\rceil = \left\lceil \frac{2^{d+1} - 1}{2} \right\rceil = \left\lceil \frac{n-1}{2} \right\rceil$$

Sendo n uma potência de 2 e, portanto, par, $\left\lceil \frac{n-1}{2} \right\rceil$ equivale a $\frac{n}{2}$.

Sejam q_i e q_j dois quóruns construídos por dois processos quaisquer i e j, respectivamente. Como cada processo i adiciona a si mesmo ao seu quórum q_i, $|q_i| = 1 + \frac{n}{2}$. Seja $P = \{p_0, .., p_{n-1}\}$ o conjunto de processos que representa o VCube em questão. Considere que $q_i = \{p_0, .., p_{n/2-1}\}$ e $q_j = \{p_{n/2}, .., p_{n-1}\}$ contém cada um, a metade distinta dos elementos de P. A adição de qualquer elemento $p_a \in q_i$ a q_j e de $p_b \in q_j$ a p_i garante a maioria para ambos e, por conseguinte, a intersecção em, no mínimo, dois elementos: $\{p_a, p_b\}$.

Assim, por indução, o teorema é válido. □

4.3. Análise do Algoritmo

Nesta seção são discutidas as propriedades do sistema de quóruns proposto. As métricas avaliadas são o tamanho e a carga dos quóruns, e os aspectos de disponibilidade e tolerância a falhas.

Tamanho e Carga dos Quóruns. Em um sistema de quóruns ideal, todos os quóruns são do mesmo tamanho, a interseção entre dois quóruns quaisquer tem o mesmo número de elementos e cada elemento pertence ao mesmo número de quóruns.

O tamanho de um quórum q_i no VCube pode ser dado por:

$$|q_i| = 1 + \sum_{s=1}^{\log_2 n} |FF\_mid_i(s)| \quad (2)$$

Considerando que cada cluster possui 2^s processos, a metade absoluta quando não existem processos falhos no cluster s é $\lceil |c_{i,s}|/2 \rceil$. No total, a metade dos processos é $\sum_{s=1}^{\log_2 n} \lceil \frac{2^s}{2} \rceil$. A inclusão do próprio processo i completa a maioria. Assim, nos cenários sem falha, os quóruns são uniformes e possuem exatamente $(n/2+1)$ elementos. Além disso, nenhum conjunto está contido em outro (*coteries*).

Seja f_s o total de processos falhos em um cluster s. Se $|c_{i,s}| - f_s$ é par, a metade que corresponde aos processos corretos é mantida. Se é ímpar, a quantidade de elementos que fazem parte do quórum do cluster é o resultado da divisão inteira de $\frac{|FF\_cluster_i(s)|}{2}$ acrescido de 1. Assim, no pior caso, o tamanho do quórum será:

$$|q_i| = 1 + \sum_{s=1}^{\log_2 n} \left\lfloor \frac{|c_{i,s}| - f_s}{2} \right\rfloor + c, c = \begin{cases} 0 & \text{se } |c_{i,s}| - f_s \text{ é par} \\ 1 & \text{se } |c_{i,s}| - f_s \text{ é ímpar} \end{cases} \quad (3)$$

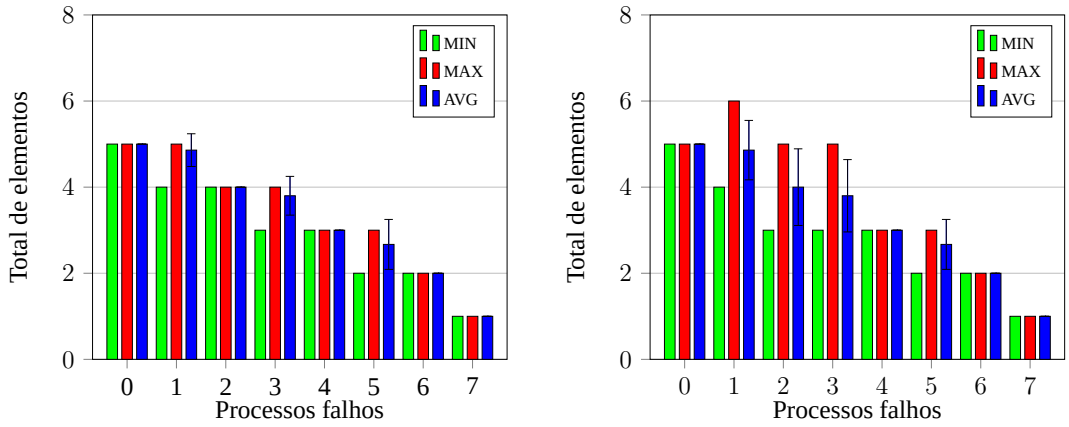
Em relação à carga, quando não existem processos falhos, a característica simétrica da $c_{i,s}$ garante a igual distribuição dos processos nos clusters e, portanto, o equilíbrio na inclusão dos elementos em cada quórum. Em caso de falha de um processo j, se $j \in c_{i,s}$, os processos corretos k que pertencem ao cluster s do processo i terão uma maior probabilidade de pertencer a um novo quórum. Se após a falha $c_{i,s} - f_s$ é ímpar, ao menos um processo k será incluído no quórum do processo i, possivelmente aumentando a carga de k, exceto se k pertencia ao quórum do processo j que falhou.

Como exemplo, considere o sistema P com $n = 8$ processos. O gráfico da Figura 3(a) mostra o tamanho dos quóruns quando não existem processos falhos e após a detecção de falhas. Inicialmente, quando não existem processos falhos cada quórum possui exatamente $(n/2+1) = 5$ elementos. Após a ocorrência de uma falha, restam 7 processos corretos e os quóruns passam a ter entre 4 (a maioria absoluta de 7) e 5 elementos (a maioria mais um). Quando existem mais processos falhos o comportamento é semelhante. A carga está representada na Figura 3(b). Quando não há falhas o tamanho e a carga são idênticos, o que mostra a uniformidade dos quóruns. Após a ocorrência de uma falha, a carga varia entre 4 e 6, mas a média se mantém em 5. Para 4 processos falhos o tamanho e a carga são iguais novamente e cada quórum possui a maioria dos processos corretos, isto é, $4/2 + 1 = 3$ elementos.

Disponibilidade. A disponibilidade é a capacidade do sistema de quóruns em tolerar falhas. Na solução com VCube, após o diagnóstico completo do nodo falho, todos os processos atualizam seus quóruns automaticamente. Assim, mesmo após $n-1$ falhas é possível reconstruir o sistema de quóruns, ou seja, o sistema proposto é $(n-1)$-resiliente.

5. Avaliação Experimental

Nesta seção são apresentados os resultados dos experimentos comparativos realizados com a solução de quóruns majoritários proposta e com o algoritmo de quóruns em árvore de Agrawal e El-Abbadi (1991), denominada TREE. Foi utilizada uma árvore binária balanceada com raiz em p_0, na qual os demais processos são adicionados sequencialmente seguindo uma distribuição em largura. Primeiro são apresentados os resultados para cenários sem processos falhos e, em seguida, para os cenários com falhas.



(a) Tamanho (b) Carga

Figura 3. Tamanho e carga dos quóruns em um sistema VCube com 8 processos.

5.1. Cenários sem falhas

A Tabela 1 mostra as propriedades dos quóruns TREE e VCube para cenários com número de processos n variando de 8 a 1024 em potência de dois. A quantidade de quóruns em cada algoritmo está representada por $|Q|$. Para o tamanho dos quóruns e a carga de um processo, isto é, o total de elementos em cada quórum e a quantidade de quóruns a qual cada elemento pertence, são mostrados o menor ($<$), o maior ($>$), a média ($\overline{q_i}$) e o desvio padrão (σ).

Quando não há falhas, o algoritmo TREE gera $|Q| = n/2$ quóruns com tamanho médio $\overline{q_i}$ muito próximo a $\log_2 n$. Neste exemplo, em função do valor de n, a árvore gerada é completa e balanceada, exceto pelo último elemento folha que compõe o último nível da árvore. Este elemento faz parte do único quórum formado pelo ramo esquerdo mais longo da árvore e que possui tamanho $\log_2 n + 1$. A solução VCube, por outro lado, gera n quóruns, um para cada processo, e cada quórum possui $n/2 + 1$ elementos. O tamanho e a carga são idênticos, visto que, em função da simetria do cálculo de $c_{i,s}$, cada processo está contido em $n/2 + 1$ quóruns.

Tabela 1. Resultados dos testes em cenários sem falhas.

	TREE	Tamanho				Carga				VCube	Tamanho/
n	$\|Q\|$	$<$	$>$	$\overline{q_i}$	σ	$<$	$>$	$\overline{q_i}$	σ	$\|Q\|$	Carga
8	4	3	4	3,25	0,50	1	4	1,63	1,06	8	5
16	8	4	5	4,13	0,35	1	8	2,06	1,88	16	9
32	16	5	6	5,06	0,25	1	16	2,53	3,07	32	17
64	32	6	7	6,03	0,18	1	32	3,02	4,77	64	33
128	64	7	8	7,02	0,13	1	64	3,51	7,18	128	65
256	128	8	9	8,01	0,09	1	128	4,00	10,58	256	129
512	256	9	10	9,00	0,06	1	256	4,50	15,35	512	257
1024	512	10	11	10,00	0,04	1	512	5,00	22,07	1024	513

Em relação à carga, verifica-se que à medida que o VCube a distribui igualitariamente entre os processos, TREE sobrecarrega a raiz, bem como proporcionalmente os nodos nos níveis mais próximos a ela. Considerando uma árvore binária, a raiz está contida em todos os quóruns, os elementos do nível seguinte em $|Q|/2$ conjuntos e assim

sucessivamente até as folhas, que pertencem a apenas um quórum. Nas medidas de carga da Tabela 1, a maior carga ($>$) é da raiz é a menor ($<$) representa as folhas. O desvio padrão confirma esta dispersão. Além disso, em um sistema que utiliza TREE, se todos os processos precisam acessar um quórum a carga pode ser ainda maior, visto que $|Q|$ representa metade do número de processos.

5.2. Cenários com falhas

Inicialmente foi testado o impacto da falha de um único processo nas duas soluções. A Tabela 2 mostra os valores de tamanho e carga dos quóruns no VCube quando um único processo está falho. Nota-se que o tamanho dos quóruns se mantém estável e a carga, embora não mais simétrica, é ainda bem distribuída.

Tabela 2. Resultados dos testes com VCube para a falha de um único processo.

VCube		Tamanho				Carga			
n	$\|Q\|$	$<$	$>$	$\overline{q_i}$	σ	$<$	$>$	$\overline{q_i}$	σ
8	7	4	5	4,86	0,38	4	6	4,86	0,69
16	15	8	9	8,93	0,26	8	10	8,93	0,70
32	31	16	17	16,97	0,18	16	18	16,97	0,71
64	63	32	33	32,98	0,13	32	34	32,98	0,71
128	127	64	65	64,99	0,09	64	66	64,99	0,71
256	255	128	129	129,00	0,06	128	130	129,00	0,71
512	511	256	257	257,00	0,04	256	258	257,00	0,71
1024	1023	512	513	513,00	0,03	512	514	513,00	0,71

A posição do nodo falho no VCube influencia os quóruns independentemente, não tem impacto no desempenho geral do sistema. Em TREE, por outro lado, cada falha em um nível da árvore altera o sistema como um todo, exceto pelos elementos folha, que invalidam o quórum nos caminhos da árvore que os contém e nos quais o impacto é maior quando combinados com múltiplas falhas. A Tabela 3 mostra os resultados para um único processo falho no segundo nível da árvore, isto é, um filho da raiz. O tamanho dos quóruns aumenta em relação ao cenário sem falhas, especialmente para os sistemas com maior número de processos. Isto é esperado em função do maior número de combinações entre as sub-árvores esquerda e direita do nodo falho. A carga também é aumentada, mas a carga mínima ainda é 1, presente nos quóruns da sub-árvore esquerda da raiz.

Tabela 3. Resultados dos testes com TREE para um processo falho não-raiz (p_1).

TREE		Tamanho				Carga			
n	$\|Q\|$	$<$	$>$	$\overline{q_i}$	σ	$<$	$>$	$\overline{q_i}$	σ
8	3	3	4	3,33	0,58	1	3	1,43	0,79
16	8	4	6	4,75	0,89	1	8	2,53	1,85
32	24	5	8	6,50	1,14	1	24	5,03	5,24
64	80	6	10	8,50	1,29	1	80	10,79	15,54
128	288	7	12	10,61	1,30	1	288	24,06	45,91
256	1088	8	14	12,74	1,20	1	1088	54,34	134,12
512	4224	9	16	14,83	1,04	1	4224	122,61	388,01
1024	16640	10	18	16,90	0,87	1	16640	274,89	1114,43

A Tabela 4, também com resultados obtidos para TREE, apresenta os dados para o caso em que a raiz (processo p_0) está falha. Em função das combinações entre os elementos da árvore esquerda e direita da raiz, a quantidade de quóruns disponíveis representada

na coluna $|Q|$ aumenta consideravelmente. O tamanho dos quóruns praticamente dobrou em relação ao cenário sem falhas. As cargas mínima e máxima também aumentam, mas continuam muito desproporcionais. Como cada quórum é formado pela combinação de um caminho da sub-árvore esquerda com um caminho da sub-árvore direita, a carga mínima aumenta proporcionalmente ao número de possíveis combinações.

Tabela 4. Resultados dos testes com TREE para a falha da raiz (p_0).

TREE		Tamanho				Carga			
n	$\|Q\|$	$<$	$>$	$\overline{q_i}$	σ	$<$	$>$	$\overline{q_i}$	σ
8	4	4	5	4,50	0,58	2	4	2,57	0,98
16	16	6	7	6,25	0,45	4	16	6,67	4,19
32	64	8	9	8,13	0,33	8	64	16,77	14,95
64	256	10	11	10,06	0,24	16	256	40,89	49,00
128	1024	12	13	12,03	0,17	32	1024	97,01	152,61
256	4096	14	15	14,02	0,12	64	4096	225,13	449,65
512	16384	16	17	16,01	0,09	128	16384	513,25	1353,97
1024	65536	18	19	18,00	0,06	256	65536	1153,48	3930,10

Em resumo, embora o tamanho dos quóruns de VCube seja maior que TREE, a carga de VCube é uniformemente distribuída e o sistema é mais estável em cenários com falhas.

6. Conclusão

Este trabalho apresentou uma solução distribuída e autonômica para a construção de quóruns majoritários em sistemas distribuídos sujeitos a falhas de *crash*. Os quóruns são construídos e mantidos sobre uma topologia de hipercubo virtual denominada VCube. O VCube organiza os processos em clusters progressivamente maiores e os conecta através de enlaces confiáveis de forma que, se não há falhas, um hipercubo é formado. Cada processo constrói seu próprio quórum adicionado a si mesmo e a metade absoluta dos processos sem falha em cada cluster.

Os sistema de quóruns com VCube foi comparado com um modelo em árvore binária denominado TREE. Os experimentos em cenários sem falhas mostram que os quóruns gerados no VCube são uniformes e têm tamanho $\lceil (n+1)/2 \rceil$. Nos cenários com falha, o tamanho e a carga dos quóruns pode aumentar, mas ainda se mantém equilibrados. Além disso, ao contrário de TREE, a posição do processo falho não tem impacto no desempenho geral do sistema. Em relação à disponibilidade, o VCube é capaz de reconstruir os quóruns com até $n-1$ processos falhos.

Como trabalhos futuros, uma aplicação será implementada para o problema da exclusão mútua distribuída baseada em quóruns. Um algoritmo de *multicast* para o quórum no VCube está em desenvolvimento.

Agradecimentos

Estre trabalho teve apoio da Fundação Araucária/SETI (convênio 341/10, proj. 19.836) e do CNPq, proj. 309143/2012-8.

Referências

Abawajy, J. e Mat Deris, M. (2013). Data replication approach with data consistency guarantee for data grid. *IEEE Trans. Comput.*, PP(99):1–1.

Abraham, I. e Malkhi, D. (2005). Probabilistic quorums for dynamic systems. *Distrib. Comput.*, 18(2):113–124.

Agrawal, D. e El Abbadi, A. (1991). An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. Comput. Syst.*, 9(1):1–20.

Agrawal, D. e El Abbadi, A. (1992). The generalized tree quorum protocol: an efficient approach for managing replicated data. *ACM Trans. Database Syst.*, 17(4):689–717.

Amir, Y. e Wool, A. (1996). Evaluating quorum systems over the internet. In *FTCS*, pages 26–, Washington, DC, USA. IEEE Computer Society.

Atreya, R., Mittal, N. e Peri, S. (2007). A quorum-based group mutual exclusion algorithm for a distributed system with dynamic group set. *IEEE Trans. Parallel Distrib. Syst.*, 18(10):1345–1360.

Barbara, D. e Garcia-Molina, H. (1986). The vulnerability of vote assignments. *ACM Trans. Comput. Syst.*, 4(3):187–213.

Barbara, D., Garcia-Molina, H. e Spauster, A. (1989). Increasing availability under mutual exclusion constraints with dynamic vote reassignment. *ACM Trans. Comput. Syst.*, 7(4):394–426.

Choi, S. C. e Youn, H. Y. (2012). Dynamic hybrid replication effectively combining tree and grid topology. *J. Supercomput.*, 59(3):1289–1311.

Freiling, F. C., Guerraoui, R. e Kuznetsov, P. (2011). The failure detector abstraction. *ACM Comput. Surv.*, 43:9:1–9:40.

Fujita, S. (1998). A quorum based k-mutual exclusion by weighted k-quorum systems. *Inf. Process. Lett.*, 67(4):191–197.

Garcia-Molina, H. e Barbara, D. (1985). How to assign votes in a distributed system. *J. ACM*, 32(4):841–860.

Gifford, D. K. (1979). Weighted voting for replicated data. In *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*, SOSP '79, pages 150–162, New York, NY, USA. ACM.

Guerraoui, R. e Vukolić, M. (2010). Refined quorum systems. *Distributed Computing*, 23(1):1–42.

Kumar, V. e Agarwal, A. (2011). Generalized grid quorum consensus for replica control protocol. In *Computational Intelligence and Communication Networks (CICN), 2011 International Conference on*, pages 395–400.

Lipcon, T. (2012). Quorum-based journaling in CDH4.1. Disponível em: `http://blog.cloudera.com/blog/2012/10/quorum-based-journaling-in-cdh4-1/`. Acessado em: 12/02/2014.

Liskov, B. e Rodrigues, R. (2006). Tolerating byzantine faulty clients in a quorum system. In *ICDCS '06*, pages 34–, Washington, DC, USA. IEEE Computer Society.

Liu, T.-J., Wang, W.-C. e Tseng, C.-M. (2011). Organize metadata servers by using quorum system. In *IEEE/SICE Int'l Symp. Syst. Integration*, pages 1125–1130.

Maekawa, M. (1985). A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.*, 3(2):145–159.

Malkhi, D., Reiter, M. K., Wool, A. e Wright, R. N. (2001). Probabilistic quorum systems. *Information and Computation*, 170(2):184–206.

Merideth, M. e Reiter, M. (2008). Write markers for probabilistic quorum systems. In *Principles of Distributed Systems*, volume 5401 of *LNCS*, pages 5–21. Springer Berlin Heidelberg.

Merideth, M. G. e Reiter, M. K. (2007). Probabilistic opaque quorum systems. In *DISC'07*, pages 403–419.

Merideth, M. G. e Reiter, M. K. (2010). Selected results from the latest decade of quorum systems research. In *Replication*, volume 5959 of *LNCS*, pages 185–206. Springer-Verlag, Berlin, Heidelberg.

Naimi, M. e Thiare, O. (2013). A distributed deadlock free quorum based algorithm for mutual exclusion. *IJCSIS*, 11(8):7–13.

Naor, M. e Wool, A. (1996). Access control and signatures via quorum secret sharing. In *ACM Conference on Computer and Communications Security*, pages 157–168. ACM.

Naor, M. e Wool, A. (1998). The load, capacity, and availability of quorum systems. *SIAM J. Comput.*, 27(2):423–447.

Preguica, N. e Martins, J. L. (2001). Revisiting hierarchical quorum systems. In *Proc. Int'l Conf. Distr. Comp. Syst. (ICDCS-21)*.

Ricart, G. e Agrawala, A. K. (1981). An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM*, 24:9–17.

Ruoso, V. K. (2013). Uma estratégia de testes logarítmica para o algoritmo Hi-ADSD. Master's thesis, Universidade Federal do Paraná.

Tanaka, K., Higaki, H. e Takizawa, M. (1999). Quorum-based protocol for group communication. In *Proc. Int'l Workshops on Parallel Processing*, pages 24–29.

Thomas, R. H. (1979). A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209.

Vukolić, M. (2010). The origin of quorum systems. *Bulletin of the EATCS*, 101:125–147.