Modelo Adaptativo de Interconexões de Data Centers
Projeto Orientado em Computação II
Relatório Final

Otávio Augusto de Oliveira Souza[1], **Olga Nikolaevna Goussevskaia**[1], **Bruna Peres**[1]

[1]Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
31270-901 - Belo Horizonte - MG - Brasil

{oaugusto, olga}@dcc.ufmg.br

Abstract. *Today's data center networks are still optimized toward static metrics, such as the diameter or the longest route. This approach has a fundamental short-coming: it must be decided in advance how much capacity to allocate to each link, making the interconnect either too expensive or imposing limits on application performance when the demand exceeds the capacity allocated to some link. To solve this problem, exciting new technologies have been emerging to enable reconfigurable interconnects in data centers. This trend, in turn, creates the necessity of new routing protocols, which are able to optimize the network's topology according to the communication pattern between end hosts in real time. SplayNets is a generalization of the classic splay tree data structures: given a set of communication requests and a network comprised of n nodes, the goal is to dynamically find and adjust a (locally routable) tree topology, which optimizes the routing cost for the given communication pattern and minimizes the topological reconfiguration costs. In this work, we present the first distributed and concurrent implementation of such self-adjusting SplayNets. We implemented the algorithm and plan to perform an in-depth simulation study of system-oriented aspects of SplayNets using real data center workloads*

1. Introdução

A computação em nuvem é uma área que vem evoluindo e se estabelecendo para mudar vários aspectos na tecnologia da informação. A cada ano o número de processamento e armazenamento de dados em nuvem cresce a altas taxas. Empresas como Google, Facebook, Amazon e outras gigantes da tecnologia possuem centenas de máquinas poderosas agrupadas, denominados data centers, que demandam altos custos de operação e manutenção.

A quantidade de tráfego de usuário para máquina é enorme e está pleno crescimento, entretanto esse tipo de tráfego é apenas a ponta do iceberg. O fluxo de dados interno dos data centers, tráfego entre a máquina, é substancialmente maior em ordens de magnitude que o fluxo para a Internet, o que mostra as pesquisas realizadas [Andreyev 2014] e ilustrado na figura 1. Dependendo da capacidade provida, as interconexões são muitas vezes caras (devido à grande quantidade necessária de bandwidth) ou limitam o desempenho das aplicações quando as demandas entre as máquinas excedem a capacidade. Para atender as exigências do mercado, operadores de serviços estão

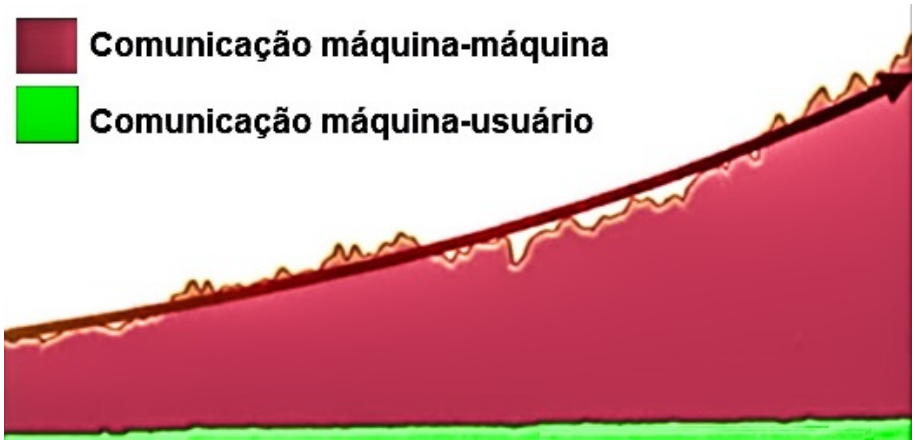


Figura 1. Comunicação em servidores

otimizando às estratégias em data center investido massivamente em infraestrutura para dar suporte ao crescimento das aplicações.

Infelizmente, as infraestruturas de data center são tecnologias recentes para compartilhar suas requisições entre diversas aplicações, tornando um grande desafio planejar a estrutura interna de um data center para uma aplicação específica. Muitos pesquisadores têm se atentado para esse fato e proposto modelos de infraestruturas reconfiguráveis para as interconexões entre os servidores, fazendo uso de tecnologias capazes de alterar sua capacidade de suporte de comunicação entre as máquinas em tempo real.

Neste trabalho propomos a implementação de um modelo de data center de interconexões ajustáveis, visando a utilização de tecnologias já existentes e algoritmos adaptativos de configuração de redes. Esperamos dessa forma alcançar reduções no custo de manutenção e projeto interno de data centers. Para alcançar esse objetivo, o trabalho foi dividido entre os dois semestres de forma a pesquisar e implementar a solução. Para o primeiro semestre desenvolvemos a parte de pesquisa e implementação do modelo em um simulador. Para a parte final do trabalho obtemos uma implementação funcional e realizamos experimentações.

2. Trabalhos Relacionados

Aplicações tradicionais (ex., aplicações map-reduce, aplicações streaming, scale-out databases, etc.) são configuradas e escaladas sobre uma estrutura fixa, que é oculta a carga de trabalho da rede. Tecnologias recentes, entretanto, têm sugerido outro paradigma: as redes têm se tornado incrivelmente flexíveis, permitindo otimizar ou ajustar a infraestrutura sobre as demandas e aplicações. Redes reconfiguráveis estão sendo propostas e avaliadas dominantemente para datacenters [Ghobadi et al. 2016, Liu et al. 2014, Farrington et al. 2010, Hamedazimi et al. 2014, Zhou et al. 2012]. Entretanto, redes ópticas reconfiguráveis são uma abordagem em crescimento na área [Jia et al. 2017], permitindo aos provedores de serviços realizar Programação da topologia (Topology Programming) em adição a engenharia de tráfego(TE), e produzindo um resultado ainda melhor da utilização dos recursos [Oda et al. 2016].

Enquanto os trabalhos apresentados empregam técnicas de reconfiguração do canal somente por comunicação direta [Ghobadi et al. 2016], estudos recentes propostos

suportam o roteamento multi-hop sobre redes reconfiguráveis e sobre diferentes camadas de IP e optica [Jia et al. 2017]. Nosso trabalho é próximo de (self-adjusting data structures) onde é avaliado sobre estrutura de dados auto-ajustáveis, em particular emphsplay trees [Sleator and Tarjan 1985]: Splay trees são árvores de busca binárias otimizadas onde os nós mais frequentes ou recentemente requisitados são movidos para próximo da raiz para reduzir o custo médio de acesso. Splay trees e suas variantes (ex., Tango trees [Demaine et al. 2004] or multi-splay trees [Wang et al. 2006]) tem sido tema intense de pesquisa por muitos anos(e.g. [Allen and Munro 1978, Sleator and Tarjan 1985, Chalermsook et al. 2016, Iacono and Langerman 2016]), e a conjectura sobre otimalidade dinâmica continua em aberto: A conjectura afirma que splay tree executam tão bem quanto qualquer outro algoritmo em árvore binária de busca [Demaine et al. 2004, Wang et al. 2006, Iacono and Langerman 2016]. Em contraste com as estruturas splay tree clássicas, in DISPLAYNET, requisições não originam unicamente da raiz da estrutura, mas as comunicações acontecem entre todos os pares de nós da rede.

Soluções para reconfiguração *networks* em que a comunicação mais frequente de nós são migradas topologicamente para regiões mais próximas sobre o tempo são emergentes. Por exemplo, *dynamic skip graphs* [Huq and Ghosh 2017] minimizam o custo médio de roteamento entre pares arbitrários de nós executando adaptações na topologia conforme o padrão de comunicação, e Flattening [Reiter et al. 2008] otimiza o custo de requisições de comunicação ponto-a-ponto sobre k-ary tree, por transformações locais de acordo com o padrão de comunicação. Entretanto, existe soluções que não levam em consideração a concorrência ou análise, e também não provê uma estrutura de roteamento local: sem a propriedade de busca local, manter tabelas de roteamento tem um custo caro em ambientes dinâmicos.

Um trabalho mais próximo ao nosso é *SplayNet* [Schmid et al. 2015]. Entretanto, SplayNets são baseadas em algoritmos centralizados (requerem um controle global e escalonamento), e são puramente sequenciais. Nós neste trabalho apresentamos um algoritmo distribuído, descentralizado e implementação concorrente de SplayNets. Essa extensão não-trivia, ambos em termos de resultado e técnicas requeridas. Essa configuração de distribuição traz mudanças fundamentais nas noções básicas como o conjunto de trabalho (em uma dada distribuição, mantendo o conjunto de trabalho de nós próximos da raiz é insuficiente) e torna impossível utilizar a análise amortizada empregando a técnica de "soma telescópica"abordada em [Sleator and Tarjan 1985, Schmid et al. 2015]. E também aparentemente parece não produzir uma abordagem geométrica fácil como em [Demaine et al. 2009] para analisar árvores binárias de busca.

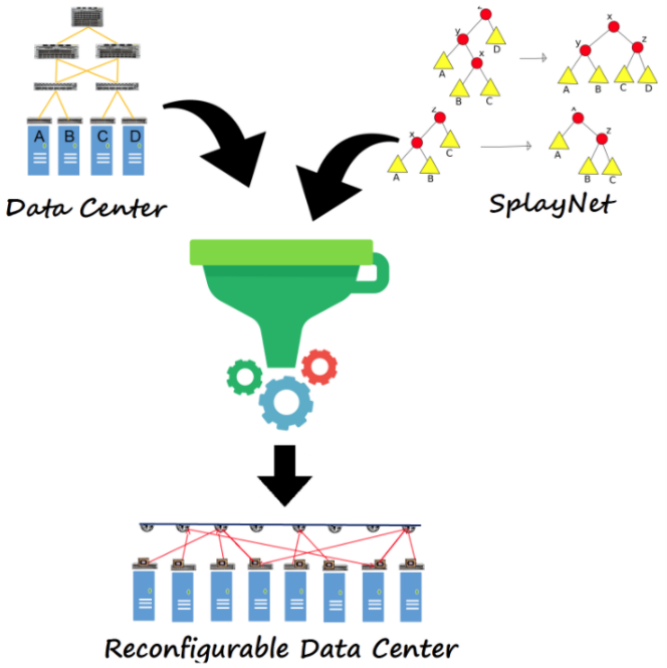
De forma interessante, como nosso modelo generaliza splay trees, nosso trabalho pode de certa forma prover novas ideias para a estrutura clássica. Em particular, para o nosso conhecimento, existe somente uma implementação de árvore binária de busca concorrente e auto ajustável até o momento, em *CBTree* [Afek et al. 2012]: ao contrário de usar os passos clássicos de "splays"(zig, zig-zig e zig-zag), CBTrees divide as rotações em simples e duplas. CBTree aumenta a performance por focar as operações nos nós mais populares mais rápido que aos nós mais frequentemente acessados. Entretanto, enquanto nosso algoritmo é distribuído, sua análise não é: os autores somente apresentam uma análise sequencial.

3. Nossa Contribuição

Neste trabalho, nós apresentamos uma primeira implementação distribuída e concorrente do SplayNets. Até agora somente o algoritmo centralizado é conhecido. Ir do algoritmo centralizado para o algoritmo distribuído e concorrente é a tarefa desafiadora, devido à complexidade adicional para manter a rede em estado consistente. Em contrapartida isso também traz benefícios, por simplificar o custo das tarefas, como o escalonamento global de requisições e a expansão da rede, em particular, enquanto a rede continua a manter um tráfego [Schlinker et al. 2015].

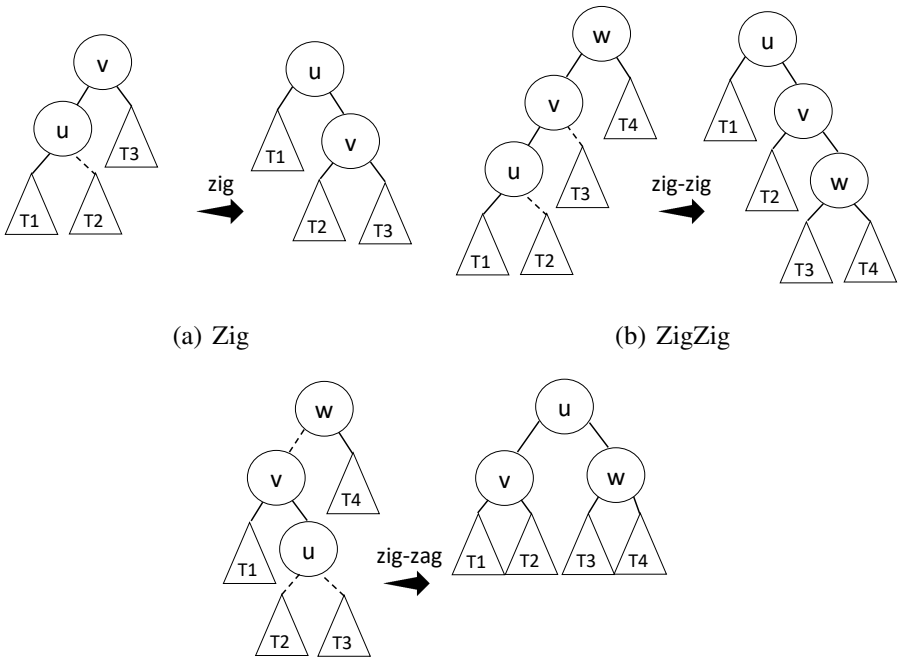
Nós analisamos a complexidade do algoritmo distribuído e concorrente. Em particular, mostramos que o algoritmo é livre de loop e deadlock, e a *amortized average cost* de uma requisição de comunicação é $O(\log n)$, onde n é o número de nós, e o tempo de execução é $O(\log n \log m)$, onde m é o número de requisições de comunicação concorrentes, que cresce somente por um fator logaritmo, onde comparado ao centralizado, no cenário não concorrente, analisado em [Schmid et al. 2015]. Note que esse resultado é uma contribuição teórica, independente do domínio da aplicação.

No intuito de validar o algoritmo, implementamos um protótipo na plataforma de simulação *Sinalgo*. A implementação leva em consideração todo modelo desenvolvido e simula um padrão de requisição entre pares de nós. A partir de simulações esperamos validar a análise teórica e descobrir novas variações viáveis a aplicações em data centers.



4. Modelo

Um SplayNet $\mathcal{T}$ é formado por um conjunto de n nós com distintos identificadores, interagindo de acordo com um dado padrão de comunicação $\mathcal{F}$. Diferentemente de [Schmid et al. 2015], onde o padrão de comunicação foi modelado como uma sequência de requisições consecutivas, nós consideramos um conjunto de m requisições concorrentes de comunicação. O objetivo é encontrar dinamicamente uma topologia de roteamento



(a) Zig (b) ZigZig

(c) ZigZag

Figura 2. Operações de rotação

em árvore binária de busca (BST), que conecta todos os nós e otimiza o custo de roteamento para $\mathcal{F}$, fazendo transformações locais na topologia, chamadas de rotação.

Modelo de Comunicação: os nós comunicam via passagem de mensagens por canais confiáveis e síncronos: nós não falham e a execução ocorre em intervalos de tempo de tamanho fixo.

Menor ancestral comum $\alpha(a, b)$: O menor ancestral comum entre dois nós a e b ($\alpha(a, b)$) em $\mathcal{T}$ é o nó que têm ambos como decentes e está mais próximo tanto de a e b.

Rotação $\beta(u)$: Rotações são transformações locais da rede, executadas atomicamente. Uma rotação move um nó u dependendo da posição relativa de u, com seus pais v e seus avôs w. Existem três tipos diferentes de rotações:

- **zig:** Os avós de u não participam em rotações zig (u pode não ter avós), veja 2(a). Nesse caso, realizamos rotação de u sobre v, fazendo os filhos de u forem nós de v e sub-árvore T_1, mantendo as sub-árvores intactas.
- **zig-zig:** u e seus pais v são ambos filhos da direita ou da esquerda, (veja a figura 2(b)). w é substituído por u, v torna se filho de u e w um filho de v, mantendo a sub-árvore intacta.
- **zig-zag:** $u \oplus v$ é um filho da esquerda e ou um filho da direita, veja Figura 2(c) A rotação substituí w por u e o filho de u torna se v e w, mantendo a subárvore intacta.

Nosso objetivo é produzir um algoritmo distribuído e auto-ajustável para uma dada

rede. Tal rede é definida sobre o conjunto $V = \{v_1, \ldots, v_n\}$ de nós n (e.g., top-of-rack switches or peers). A entrada de nosso problema é uma sequência $\sigma = (\sigma_1, \sigma_2 \ldots \sigma_m)$ de requisições de comunicação ocorrendo ao longo do tempo, tal que σ_i ocorre antes de σ_j se $i < j$. Para cada σ_i descrevendo um par de comunicação, e , $\sigma_i = (s_i, d_i) \in V \times V$, com fonte em s_i e destino d_i.[1] Caso não dito, entretanto, nós assumimos para facilidade de apresentação, que existe ao menos uma requisição por tempo i. A sequência σ pode ser arbitrária: em particular, em nossa análise consideramos o pior cenário, onde σ é escolhido *adversário*, no intuito de maximizar o custo dado o algoritmo distribuído.

Modelo de custo: Neste modelo, nós referimos a reconfigurações locais como **passos(steps)** e assumimos que em cada passo,que envolve um número constante de trocas de links,tem um custo $O(1)$.

Modelo de tempo: Para estudar a concorrência, dividimos o tempo de execução em *rodadas*: em cada rodada, múltiplos (independent) nós podem fazer reconfigurações (passos) concorrentes.

Nosso objetivo é **minimizar** o custo em termos do *trabalho* e o custo em termos do *tempo*. Desde que a execução concorrente é executada em paralelo, tempo e trabalho nem sempre coincidem. Especialmente, nós consideramos a seguinte notação, i.e.,reconfiguration cost to serve a sequence of requests σ::

Em termos de tempo, medimos o quanto é gasto para processar um dado conjunto de requisições de comunicação. Que é, nosso objetivo minimizar o *makespan*.

Nós estamos interessados no pior caso sobre uma sequência arbitrária de operações, e então, conduzimos a análise amortizada [Cormen et al. 2001]. No modelo, a análise de custo pode ser vista como o custo médio por requisição para uma dada sequência σ de comunicação.

5. Projeto

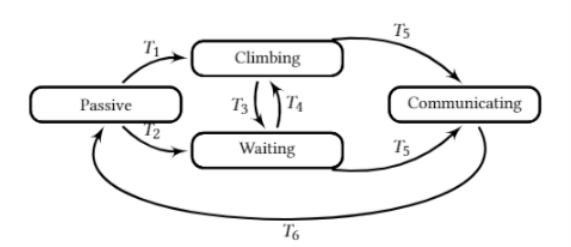
5.1. Reconfiguração Distribuída

O algoritmo distribuído implementado no sistema segue o conceito chave abaixo:

1. *Reconfiguração local:* No intuito de ajustar a topologia localmente sem violar a propriedade de roteamento, nós utilizamos das operações de **zig**, **zig-zig**, e **zig-zag** conhecida de splay trees(see Definition [Tarjan 1985]).
2. *Conjunto independente:* No intuito de facilitar o ajuste concorrente enquanto evitando deadlocks, nós calculamos (de forma distribuída) *clusters* locais: clusters são cordenador por um nó requisitando rotação (resp. a cluster *master*), e podem ser atualizados em paralelo, sem interferência.
3. *Prioridade:* No intuito de não causar inanição, nós damos prioridade à requisições de acordo com seu tempo de origin(b_i).

A seguir iremos elaborar cada um dos componentes em maior detalhe:

[1] Nota se que o nó $v \in V$ pode ser participante em várias requisições em σ, e.g, $\sigma_i(v, d_i)$ e $\sigma_j(s_j, v), i < j$.



5.2. Preservação da ordem nas transformações

No intuito de realizar roteamento local e preservar a ordem de reconfiguração, o algoritmo requer que cada nó u armazena o identificador de seus vizinhos direto na árvore de busca binária, i.e., seu pai ($u.p$), seu filho esquerdo ($u.l$) e seu filho direito ($u.r$), o menor ($u.smallest$) e maior ($u.largest$) identificador corrente na sub-árvore em u.

Sobre uma $\sigma_i = (u, v)$, o nó u e v começa a mover em direção um do outro, realizando rotações locais que preservam a propriedade de busca. DISPLAYNETS implementam as transformações usando das rotações **zig**, **zig-zig**, e **zig-zag** de splay trees. Diferente das splay trees, em DISPLAYNETS, os nós se movem em direção a raiz. Ao contrário, sobre uma dada requisição $\sigma_i = (u, v)$, o nó u e v são rotacionados somente em direção ao *ancestral comum mais próximo* ($LCA(u, v)$).

5.3. Algoritmo

Com esses conceitos em mente podemos apresentar nosso algoritmo 1. Cada nó na rede executa os seguintes passos:

while(*true*) {

    execute *clusterStep()*

}

Em maiores detalhes, DISPLAYNETS pode ser descrito como uma máquina de estados, executando em cada nó de forma concorrente. Cada nó pode estar em um dos quatro estados a seguir:

1. *Passive:* O nó está passivo no tempo t se não for a fonte ou destino de uma requisição em $\sigma_i \in \sigma, b_i \leq t$
2. *Climbing:* O nó s_i (ou d_i) está subindo no tempo t se tem uma *requisição ativa*: $\exists \sigma_i(s_i, d_i) \in \sigma, b_i \leq t$ and $d_t(s_i, d_i) > 1$, e em adição s_i (or d_i) $\neq LCA_t(s_i, d_i)$;
3. *Waiting:* O nó s_i (ou d_i) está em espera no tempo t se tem uma requisição ativa e $s_i = LCA_t(s_i, d_i)$.
4. *Communicating:* Um nó s_i ou d_i está em comunicação no tempo t se $\exists \sigma_i(s_i, d_i) \in \sigma, b_i \leq t$ e $d_t(s_i, d_i) = 1$.

A figura 5.3 mostra os possíveis estados e suas transições.

Para evitar a ocorrência de deadlock e inanição, sequência de splays concorrentes são escolhidos de acordo com a *prioridade* em DISPLAYNETS. Dado duas requisições σ_i e σ_j, nós dizemos que σ_i tem prioridade mais alta que σ_j se $i < j$.

Uma requisição mais antiga na rede tem maior prioridade que a mais recente requisição. Nota se que, o nó s nó estado de *espera* pode ser removido do LCA por um splay com prioridade mais alta. Se isso acontecer, s retorna para o estado de *subindo* e volta a requisitar rotações. Finalmente, quando s e d se encontram, eles se comunicam.

Para sincronizar o processo entre os nós, o algoritmo distribuído executa em *rodadas*. Cada rodada é composta de cinco fases, convertidas no algoritmo 1:

1. Requisição de cluster
2. Top-down Acks
3. Bottom-up Acks
4. Atualização dos links
5. Atualização do estado

Cada nó u mantém um buffer local, contendo uma fila de requisições, gerada por si mesmo, seu filho direito ou esquerdo, um de seus quatro bisnetos ou um dos seu oito próximos descendentes. Em cada rodada, cada requisição $\mathcal{C}_u$ é enviada para cima até atingir seu *avô* (2 hops para cima em caso de zig e 3 hops em caso de zig-zig or zig-zag). Uma vez que todas as requisições foram recebidas, a de mais alta prioridade é confirmada para o nó que a requeriu. Se sua requisição é de mais alta prioridade que recebeu na fase 1, sobre uma confirmação, quem requisitou envia uma confirmação para o seu avô. Nós dizemos que nós em um cluster são vizinhos $\mathcal{C}_u$ se todos os que receberam uma confirmação top-down ou bottom-up para rotação($\mathcal{C}_u$).

5.4. Implementação do Algoritmo

Para implementar o algoritmo de forma a prover legibilidade e facilitar futuras modificações, seguimos o diagrama de classe mostrado na figura 3. Nessa arquitetura a classe genérica *BinaryTreeNode* encapsula todas as operações da estrutura em árvore binária, simplificando a implementação e extensão do modelo. A classe SplayNetNode é responsável por implementar todas as operações de *splay*, filas de prioridade e *buffer* de mensagens ligadas ao algoritmo.

A implementação do algoritmo foi realizada na plataforma sinalgo utilizando da linguagem de programação Java 1.8. O simulador sinalgo permite validar algoritmos distribuídos em redes sem-fio e para o nosso trabalho utilizamos da plataforma a fim de avaliar a corretude e confirmar a análise estabelecida para nosso algoritmo. Na figura 4 mostramos a simulação de uma requisição entre dois nós da rede. A partir de rotações locais ambos os nós realizam operações sucessivas até atingir seus objetivos como mostra a figura 5.

6. Análise

Nesta seção apresentamos nossas análises sobre o algoritmo e destacamos as principais propriedades encontradas.

6.1. Primeiro observação

Nós primeiramente notamos que diferentes clusters não interferem e formam um ”conjunto independent”.

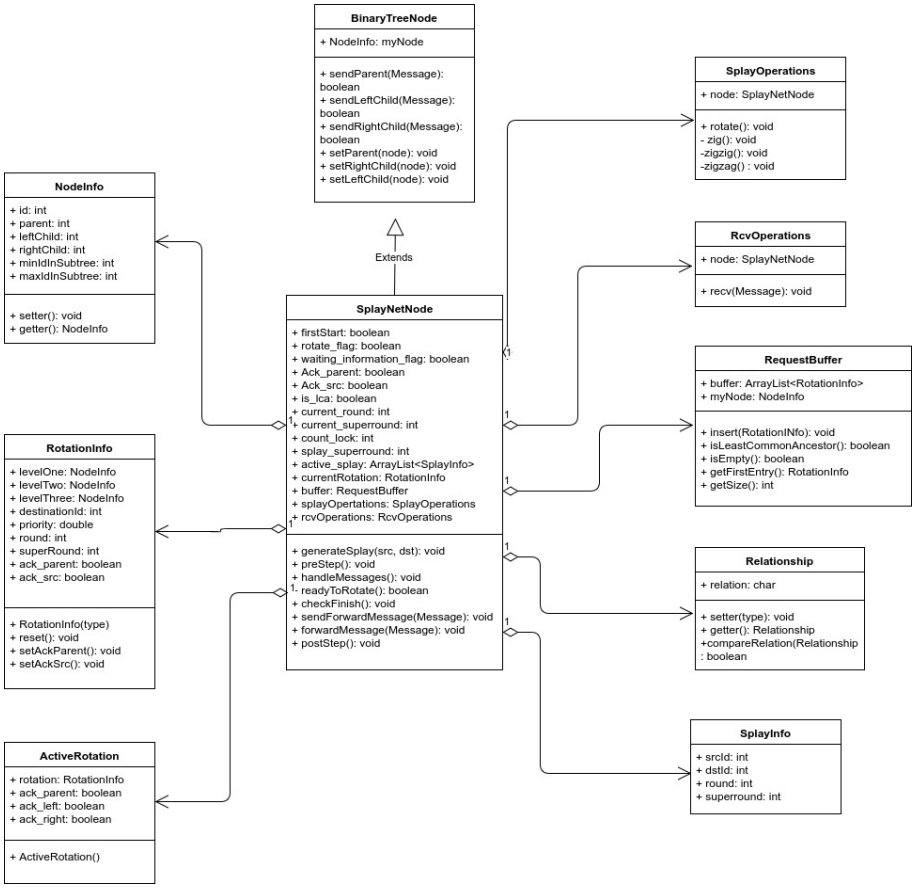


Figura 3. Diagrama de classe

6.2. Livre de deadlock

Uma propriedade essencial do sistema é que ele nunca entra em deadlock:

6.3. Análise de desempenho

Em função de calcular o pior caso sobre uma sequência arbitrária, nós conduzimos uma análise amortizada da performance do sistema.

Seja o tamanho $s_i(u)$ denotar o número de nós na sub-árvore de u incluindo u na rodada t_i. Nós definimos o **rank** $r_i(u)$ do nó u como o logaritmo em base 2 do número de nós na sub-árvore de u. , i.e., $r_i(u) = \log_2(s_i(u))$. Nós definimos o total DISPLAYNET rank $r(T_i)$ como a soma de todos os ranks de todos os nós em T_i. Nota se que o tamanho máximo e o rank máximo de um nó é n e $\log_2 n$, respectivamente. O potencial de uma dada árvore T_i no tempo t_i é então a soma do rank de todos os nós na árvore: $\phi(T_i) = \sum_{j=1}^{n} r_i(j)$. No método potencial, o custo amortizado $\hat{c}_i$ de uma operação (step) i é o custo atual c_i mais o incremento no potencial δ_i devido a operação i, onde $\delta_i = \phi(T_i) - \phi(T_{i-1})$. Isto nos dá:

$$\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1}) \tag{1}$$

Isso significa que o custo amortizado e a função potencial deve ser definida de forma a sempre manter certa equivalência. Essa equivalência implica que se o custo atual

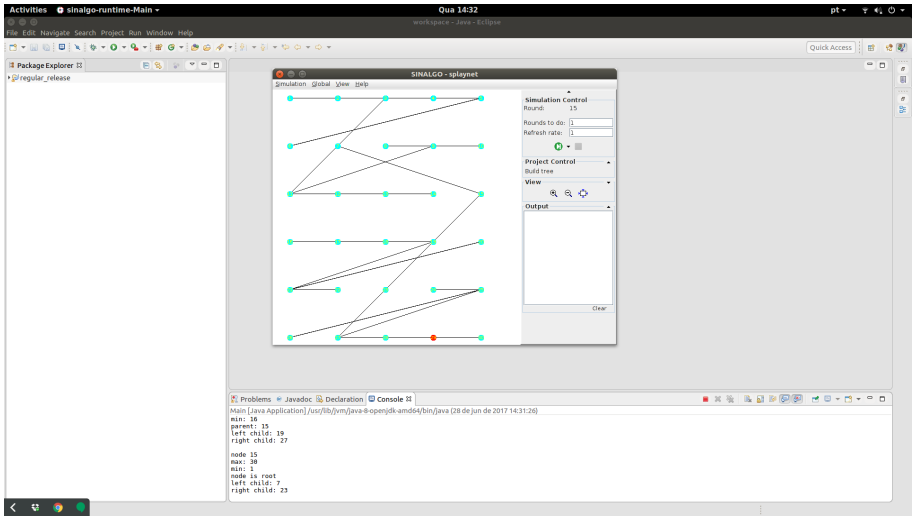


Figura 4. Início de requisição

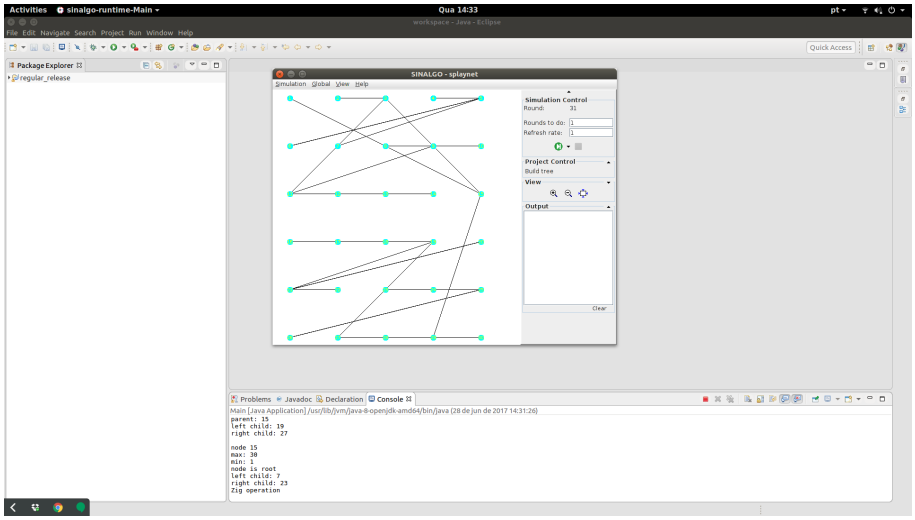


Figura 5. Estado final do splay

de uma operação é menor que o custo amortizado, o potencial é incrementado, e se o custo potencial de uma operação é maior que o custo amortizado, então o potencial é decrementado.

O fundamento da análise amortizada segui o seguinte: O total de potencial alterado em uma rodada que consiste de múltiplos passos, é simplesmente a soma do potencial mudado de um cluster individual.

- $\delta_i(u) \leq 3(r'(u) - r(u)) - 2$, se a operação for um zig-zig ou zig-zag;
- $\delta_i(u) \leq 3(r'(u) - r(u))$, se a operação for um zig.

Uma vez que podemos representar o potencial alterado para cada passo em termos do rank alterado de um nó requisitado, nós temos o custo executado para todos os passos na rodada t_i.

$$\begin{aligned}\Delta(u) &\leq \sum_{i=1}^{2m}\sum_{j=1}^{p_i}\delta_j(u) \leq \sum_{i=1}^{2m}\sum_{j=1}^{p_i}(3(r_j(u)-r_{j-1}(u))-2)+2 \\ &\leq \sum_{i=1}^{2m}(3(r_{p_i}(u)-r_i(u))-2p_i)+2 \\ &\leq 6m(\log n)-2\sum_{i=1}^{2m}p_i+2 \qquad (2)\end{aligned}$$

7. Simulações

Para completar nossa análise do pior caso e para avaliar o desempenho de DISPLAYNETS sobrecarga de trabalho realista, ambos em termos do custo de trabalho e tempo (makespan and throughput) nós conduzimos diversas simulações. Nesta seção, reportamos os principais resultados.

7.1. Configuração e Baseline

Para gerar um fluxo de requisições, nós observamos o dataset publicado coletado pelo **ProjecToR** projeto [pro 2016]. O dataset descreve a probabilidade sobre 8367 pares de comunicação na rede consistindo de 128 nós. Os 128 nós são aleatoriamente selecionados de 2 clusters (executando processamentos, incluindo MapReduce, banco de dados e armazenamento). No intuito de estudar a performance de DISPLAYNET sobre tamanhos $n' = k \times n, n = 128$, nós empregamos k do dado padrão de comunicação. Em particular, nós focamos em redes com 128 (*pequenas*) e 1024 (*grandes*).

Para gerar uma sequência no tempo, nós assumimos uma distribuição de Poisson para a chegada de cada requisição. Os Ids originais dos nós são sorteados a cada nova simulação.

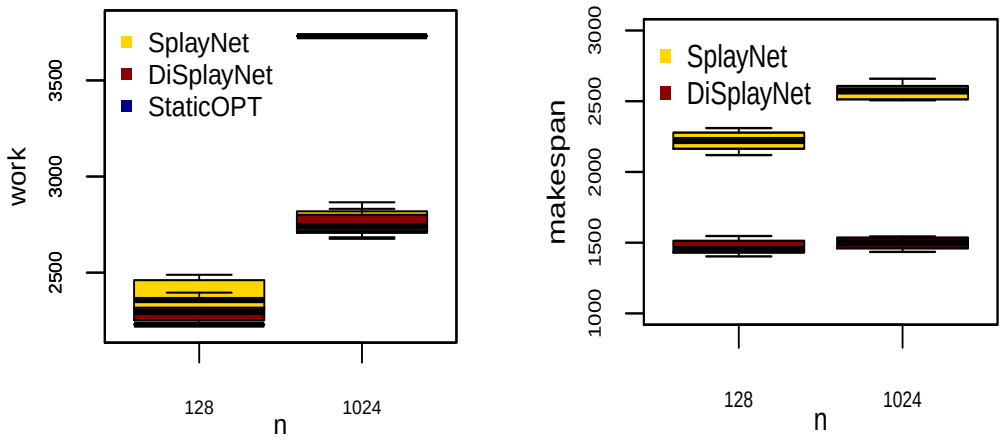
Cenário de amostragem independente DS(1): Requisições são uniformemente distribuídas pela matriz de requisição dada. No geral, a simulação é dirigida por evento e baseada no simulador Sinalgo [Group 2007]. Cada experimento repetido 50 vezes.

Nossos baselines foram algoritmo "statically optimal", usando programação linear [Schmid et al. 2015].

7.2. Trabalho: O preço da descentralização?

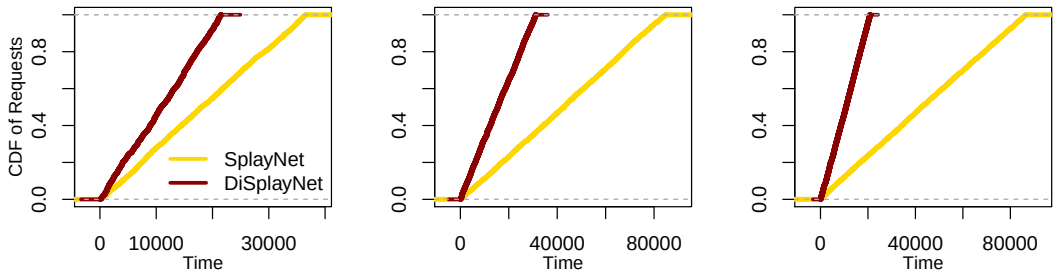
Interestingly, os resultados de nossa simulação sugerem que o overhead em termos do trabalho é negligenciável em comparação ao algoritmo centralizado. Figura 12 mostra o número de rotações locais (para dois tamanhos de rede), sobre DISPLAYNETS e duas *baselines*. Os resultados mostram que a de fato uma pequena diferença no trabalho entre nosso sistema concorrente e o sequencial. Nós também observamos que o trabalho total aumenta com o tamanho da rede, e em redes estática ótima executa em resultados significativamente piores que redes auto-ajustáveis.

Em conclusão, DISPLAYNETS parece ter um desempenho próximo de SplayNets na prática, sugerindo uma análise assintótica mais ajustada que a apresentada na análise.



(a) (b)

Figura 6. Concorrência reduz o makespan, ambos em pequenas e grandes redes .



(a) dataset1 (b) dataset2 (c) dataset3

Figura 7. Throughput CDF

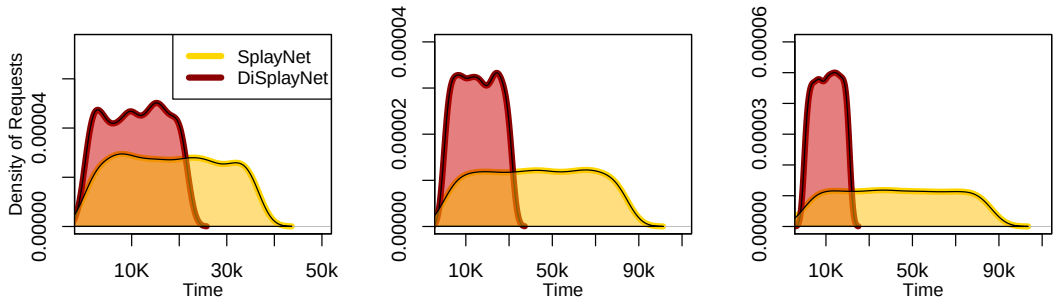
7.3. Make Span e Throughput

Os ajustes concorrentes geram um grande desempenho no tempo de execução e trabalho que pode ser demandado pela rede. Nas figuras 7 e 8 nós podemos ver que DISPLAYNETS melhora o *make span*, o tempo gasto para servir um conjunto de requisições de comunicação, tão bem quanto o *throughput*, o número de requisições completas por unidade de tempo.

Random Walk Cenários DS(2):

Na figura 9 e 10 nós mostramos o trabalho total e o makespan para diferentes combinações de parâmetros n, $|w|$ e $\#w$ no dataset DS(2). Nota se que caminhos longos no random walk $|w|$, menor e a localidade temporal da sequência de requisições. Nós podemos ver que, para uma dada rede, o trabalho total e a makespan gradualmente cresce com $|w| \times \#w$ cresce. Isso é explicado pelo fato que o tamanho do conjunto ativo da sequência de requisições cresce ambos com $|w|$ e $\#w$.

Colocamos atenção no throughput, figuras 7 e 8 representam o número de



(a) dataset1 (b) dataset2 (c) dataset3

Figura 8. Throughput PDF

requisições completas por rodada durante a simulação por inteira para diferentes datasets, ambos como uma CDF (função cumulada de distribuição) e como uma PDF (função de distribuição de probabilidade). Nós podemos ver que DISPLAYNETS explora a concorrência para atingir mais trabalho em menor tempo em todos os datasets.

Tabela 1. DS(2) cenários de simulação (legenda para as figuras 9,10, 11)

scenario#	n	$\|w\|$	$\#w$	scenario#	n	$\|w\|$	$\#w$
1	128	16	4	5	1024	16	4
2	128	16	16	6	1024	16	16
3	128	64	4	7	1024	64	4
4	128	64	16	8	1024	64	16

7.4. Implicações da estrutura concorrente

Nós também estudamos a performance da concorrência do algoritmo para a estrutura clássica *Splay Trees*. Para essa finalidade nós geramos novas sequências de requisições:

Cenário de requisições da raiz DS(3): Neste dataset, nós geramos requisições que são sempre originadas da raiz da árvore e comunica para cada nó como destino de acordo com uma distribuição normal sobre todos os links da rede. Figuras 7 (*right*), 8 (*right*), 9, e 10 mostram que, resultados similares obtidos em DS(1) e DS(2), o overhead causado no trabalho para o cenário sequencial é negligenciável. Ao mesmo tempo, nós podemos novamente beneficiar em termos do makespan e throughput.

Tabela 2. DS(3) cenários de simulação (legenda para a figura ??,??)

scenario#	n	std	scenario#	n	std
1	128	.2	4	1024	.2
2	128	1	5	1024	1
3	128	5	6	1024	5

8. Conclusões e Próximos Passos

ProjecToR [Ghobadi et al. 2016] é uma recente tecnologia que está ainda em protótipo, mas promete muitas vantagens, como vários fluxos de comunicação com baixo tempo.

Essas características são cruciais para a otimização do desempenho, desde que o padrão de tráfego observado depara com 50-99% dos pares de racks não trocando grande quantidade mensagens, enquanto que 0.04-0.4% deles possuem uma quantidade de 80% do tráfego total (i.e., topologias que provém distribuição uniforme da capacidade entre os pares de racks são tanto sobrecarregadas em alguns canais quanto desperdiçada na maioria das conexões). Além disso, é difícil determinar a priori a capacidade demandada por cada par de racks.

As redes atuais ainda são otimizadas levando em vista métricas estáticas, como diâmetro ou rota mais longa [Al-Fares et al. 2008][Goussevskaia et al. 2014]. O surgimento de tecnologias de reconfiguração de interconexões cria a necessidade de novos protocolos de roteamento, que são capazes de otimizar as topologias da rede de acordo com o padrão de comunicação entre o destino e a origem, adaptando dinamicamente e distribuída.

Em particular, SplayNets é uma solução natural para gerenciamento de topologia de redes para ambientes de DC equipados com hardware reconfigurável. Não somente podem otimizar o custo das rotas dinamicamente, de acordo com o tráfego em tempo real, mas também podem permitir uma implementação concorrente e distribuída do sistema.

Nossos próximos passos vão focar na implementação dos aspectos do sistema do algoritmo em modelos reais. Em particular, nosso plano são avaliar outras situações, como a concorrência de várias redes (várias splayNets) e produzir resultados com dados reais em data centers.

Referências

(2016). Projector dataset. `www.microsoft.com/en-us/research/project/projector-agile-reconfigurable-data-center-interconnect`.

Afek, Y., Kaplan, H., Korenfeld, B., Morrison, A., and Tarjan, R. E. (2012). Cbtree: A practical concurrent self-adjusting search tree. In *Proceedings of the 26th International Conference on Distributed Computing*, DISC'12, pages 1–15, Berlin, Heidelberg. Springer-Verlag.

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74.

Allen, B. and Munro, I. (1978). Self-organizing binary search trees. *J. ACM*, 25(4):526–535.

Andreyev, A. (2014). Introducing data center fabric, the next-generation facebook data center network. `https://code.facebook.com/posts/360346274145943/`. Accessed: 2017-07-04.

Chalermsook, P., Goswami, M., Kozma, L., Mehlhorn, K., and Saranurak, T. (2016). The landscape of bounds for binary search trees. *CoRR*, abs/1603.04892.

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

Demaine, E. D., Harmon, D., Iacono, J., Kane, D., and Patraşcu, M. (2009). The geometry of binary search trees. In *Proc. 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 496–505. SIAM.

Demaine, E. D., Harmon, D., Iacono, J., and Patrascu, M. (2004). Dynamic optimality - almost [competitive online binary search tree]. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 484–490.

Farrington, N., Porter, G., Radhakrishnan, S., Bazzaz, H. H., Subramanya, V., Fainman, Y., Papen, G., and Vahdat, A. (2010). Helios: a hybrid electrical/optical switch architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*, 40(4):339–350.

Ghobadi, M., Mahajan, R., Phanishayee, A., Devanur, N., Kulkarni, J., Ranade, G., Blanche, P. A., Rastegarfar, H., Glick, M., and Kilper, D. (2016). Projector: Agile reconfigurable data center interconnect. In *SIGCOMM*, pages 216–229.

Goussevskaia, O., Halldórsson, M. M., and Wattenhofer, R. (2014). Algorithms for wireless capacity. *IEEE/ACM Transactions on Networking*, 22(3):745–755.

Group, D. C. (2007). Sinalgo - simulator for network algorithms. `http://disco.ethz.ch/projects/sinalgo/index.html`. Accessed 10-May-2017.

Hamedazimi, N., Qazi, Z., Gupta, H., Sekar, V., Das, S. R., Longtin, J. P., Shah, H., and Tanwer, A. (2014). Firefly: A reconfigurable wireless data center fabric using free-space optics. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 319–330. ACM.

Huq, S. and Ghosh, S. (2017). Locally self-adjusting skip graphs. arXiv:1704.00830.

Iacono, J. and Langerman, S. (2016). Weighted dynamic finger in binary search trees. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 672–691, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Jia, S., Jin, X., Ghasemiesfeh, G., Ding, J., and Gao, J. (2017). Competitive analysis for online scheduling in software-defined optical wan. In *Proc. IEEE INFOCOM*.

Liu, H., Lu, F., Forencich, A., Kapoor, R., Tewari, M., Voelker, G. M., Papen, G., Snoeren, A. C., and Porter, G. (2014). Circuit switching under the radar with reactor. In *NSDI*, volume 14, pages 1–15.

Oda, S., Miyabe, M., Yoshida, S., Katagiri, T., Aoki, Y., Rasmussen, J. C., Birk, M., and Tse, K. (2016). Demonstration of an autonomous, software controlled living optical network that eliminates the need for pre-planning. In *Optical Fiber Communications Conference and Exhibition (OFC), 2016*, pages 1–3.

Reiter, M. K., Samar, A., and Wang, C. (2008). Self-optimizing distributed trees. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12.

Schlinker, B., Mysore, R. N., Smith, S., Mogul, J. C., Vahdat, A., Yu, M., Katz-Bassett, E., and Rubin, M. (2015). Condor: Better topologies through declarative design. In *SIGCOMM*.

Schmid, S., Avin, C., Scheideler, C., Borokhovich, M., Haeupler, B., and Lotker, Z. (2015). Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking*, PP(99):1–13.

Sleator, D. D. and Tarjan, R. E. (1985). Self-adjusting binary search trees. *J. ACM*, 32(3):652–686.

Tarjan, R. (1985). Amortized computational complexity. *SIAM Journal on Algebraic and Discrete Methods*, 6(2):306–318.

Wang, C. C., Derryberry, J., and Sleator, D. D. (2006). O(log log n)-competitive dynamic binary search trees. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 374–383, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

Zhou, X., Zhang, Z., Zhu, Y., Li, Y., Kumar, S., Vahdat, A., Zhao, B. Y., and Zheng, H. (2012). Mirror mirror on the ceiling: Flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review*, 42(4):443–454.

Algorithm 1 *ClusterStep()* (one round)

1: Cluster Requests (3 time-slots)

if Climbing for some $\sigma_i(s, d)$ **then**
send $request(\mathcal{C}_u)$ upward;
insert $request(\mathcal{C}_u)$ into buffer;

upon receiving $request(\mathcal{C}_w)$**:**
insert $request(\mathcal{C}_w)$ into buffer;
forward $request(\mathcal{C}_w)$ upward;

2: Top-down Acks (3 time-slots)

get highest priority $request(\mathcal{C}_x)$ in buffer;
if Master($request(\mathcal{C}_x)$) **then**
send Ack($request(\mathcal{C}_x)$) downward;

upon receiving top-down ack $request(\mathcal{C}_w)$**:**
if $w = x$ **then**
forward Ack($request(\mathcal{C}_w)$) toward requester;

3: Bottom-up Acks (3 time-slots)

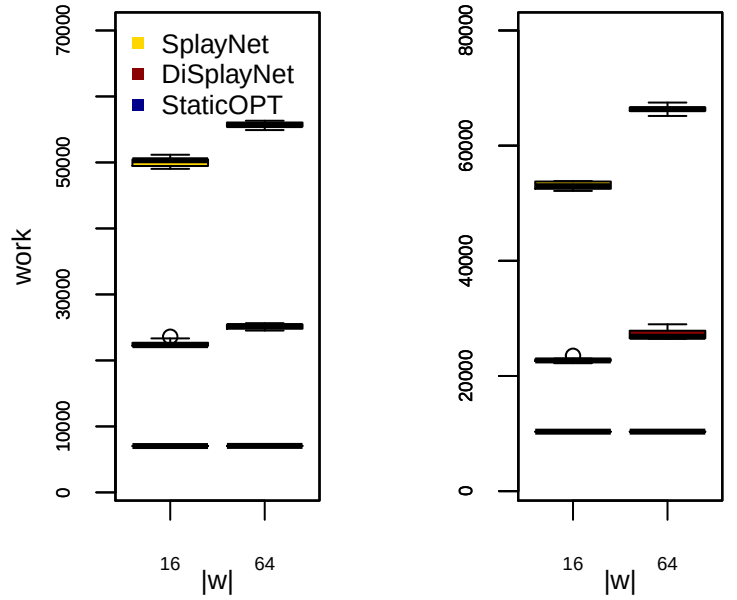
upon receiving top-down ack($request(\mathcal{C}_u)$**):**
if Requester($request(\mathcal{C}_u)$) and $u = x$ **then**
send Ack($request(\mathcal{C}_u)$) up toward master;
create $\mathcal{C}_u$;
join $\mathcal{C}_u$;
else
forward Ack($request(\mathcal{C}_u)$) toward master;
join $\mathcal{C}_u$;

4: Link Updates (3 time-slot)

if in($\mathcal{C}_u$) **then**
update links according to $\mathcal{C}_u$;

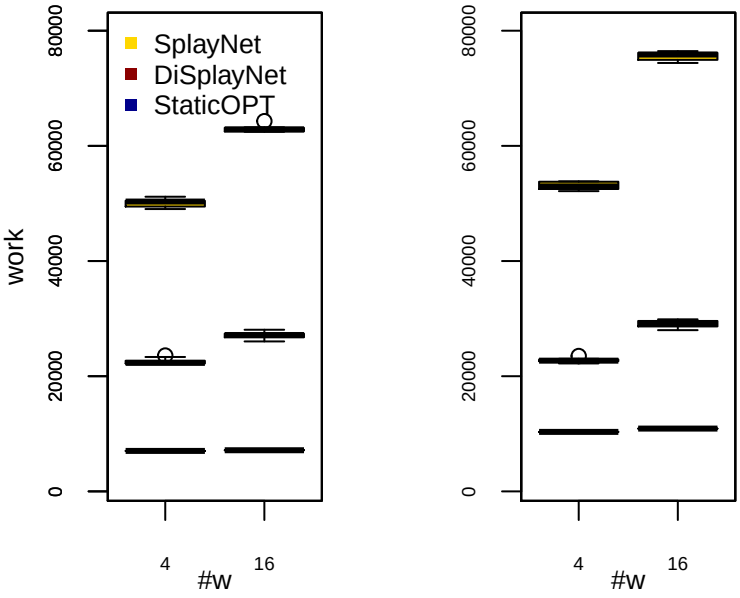
5: State Updates (1 time-slot)

update state; ▷ Figure 5.3
clear buffer;
leave cluster;



(a) ds3

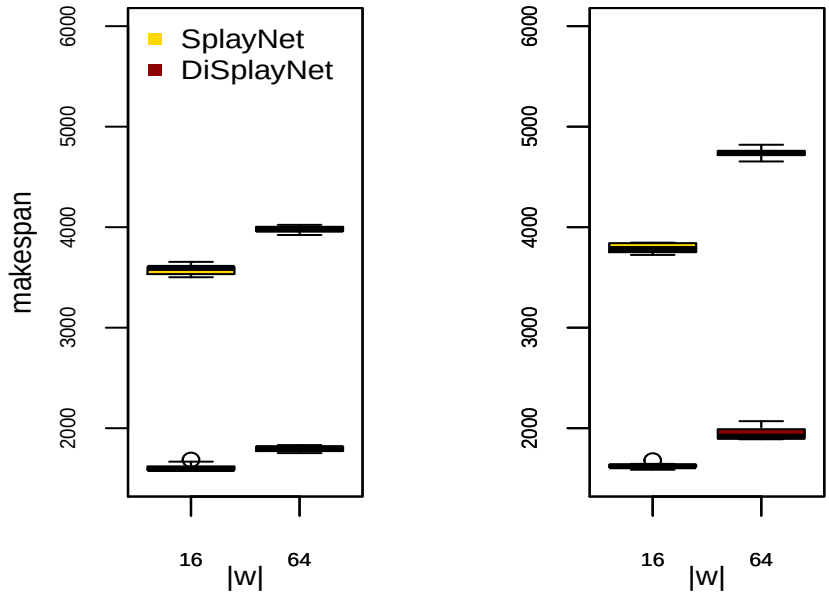
(b) ds3



(c) ds3

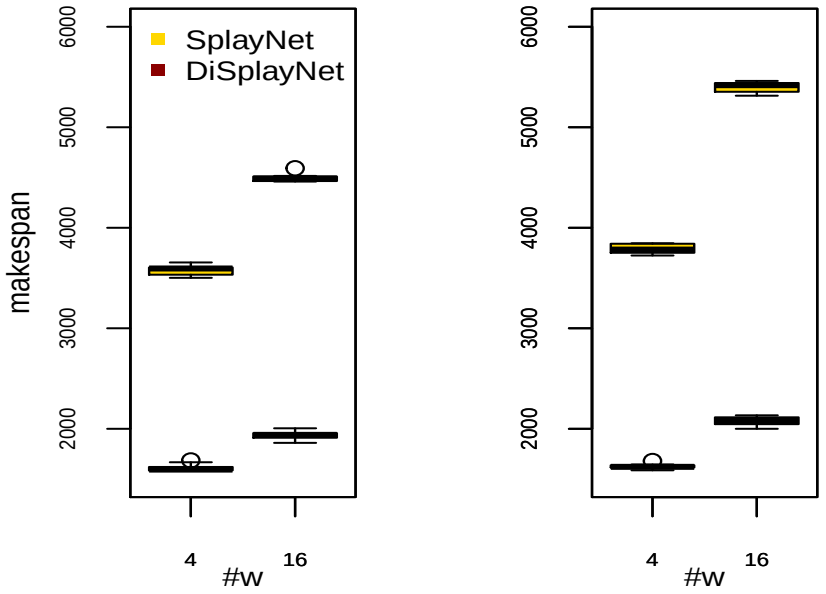
(d) ds3

Figura 9. DS(2) makespan.



]

(a) ds3 (b) ds3



(c) ds3 (d) ds3

Figura 10. DS(2) makespan.

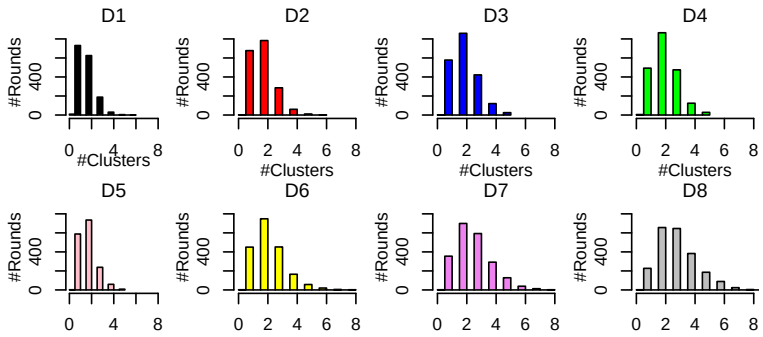
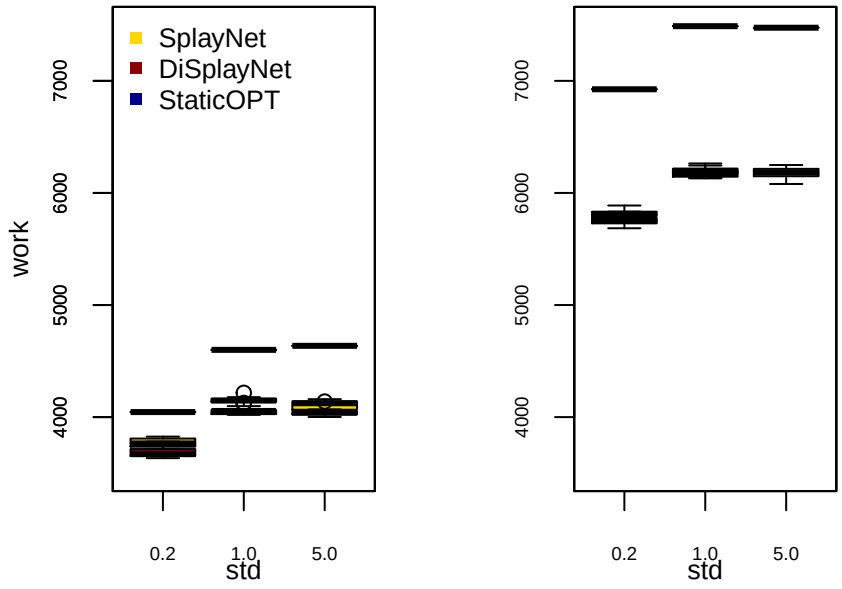
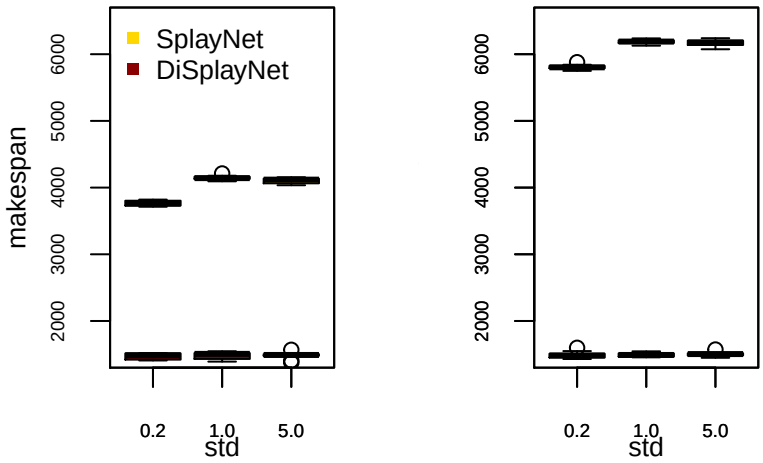


Figura 11. DS(2) clusters ativos (legenda: D=DISPLAYNETS, cenário# listado na tabela 1)



(a) dataset 128 (b) dataset 1024

Figura 12. Dataset DS(3) trabalho



(a) makespan 128 (b) makespan 1024

Figura 13. Dataset DS(3) makespan