UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS

GUSTAVO VILAÇA LOURENÇO

Análise de algoritmos distribuídos para escalonamento em Data Grids

São Carlos
2012

GUSTAVO VILAÇA LOURENÇO

Análise de algoritmos distribuídos para escalonamento em Data Grids

Dissertação apresentada ao Programa de Pós-Graduação em Física do Instituto de Física de São Carlos da Universidade de São Paulo, para obtenção do título de Mestre em Ciências.

Área de concentração :Física Aplicada
Opção: Física Computacional
Orientador: Prof. Dr. Gonzalo Travieso

Versão Corrigida

(Versão original disponível na Unidade que aloja o Programa)

São Carlos
2012

AUTORIZO A REPRODUÇÃO E DIVULGAÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Ficha catalográfica elaborada pelo Serviço de Biblioteca e Informação do IFSC,
com os dados fornecidos pelo(a) autor(a)

Vilaça Lourenço, Gustavo
Análise de algoritmos distribuídos para
escalonamento em Data Grids / Gustavo Vilaça Lourenço;
orientador Gonzalo Travieso - versão corrigida --
São Carlos, 2012.
108 p.

Dissertação (Mestrado - Programa de Pós-Graduação em
Física Aplicada) -- Instituto de Física de São Carlos,
Universidade de São Paulo, 2012.

1. Data Grid. 2. Rede Lógica. 3. Escalonamento de
processos. 4. Replicação de dados. I. Travieso,
Gonzalo, orient. II. Título.

AGRADECIMENTOS

Ao meu orientador, pelo conhecimento.

Aos meus pais, pela confiança. Aos meus irmãos, pelo incentivo. Aos amigos, pelo entretenimento.

E a todos que, de alguma forma, contribuíram ao longo do caminho.

RESUMO

LOURENÇO, G. V. **Análise de algoritmos distribuídos para escalonamento em Data Grids**. 2012. 108 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2012.

É um resultado conhecido que em Data Grids, onde o processamento envolve grandes quantidades de dados, pode ser mais eficaz escalonar os processos para execução nos sites que já dispõem dos dados do que transferir os dados para um site onde o processo que irá necessitar deles foi escalonado. Os estudos existentes se baseiam em pequenas quantidades de sites, com conhecimento centralizado sobre o estado dos diversos sites. Essa opção não é escalável para Grids com grande número de participantes. Este trabalho analisa versões distribuídas com informação local para os algoritmos de escalonamento de processo e replicação de dados, mostrando o efeito das topologias de interconexão de sites no desempenho desses. É observado que, considerando a existência apenas de informações locais devido às restrições topológicas, resultados diferentes quanto aos melhores algoritmos de escalonamento de processos e replicação de dados são encontrados.

Palavras-chave: Data Grid. Rede lógica. Escalonamento de processos. Replicação de dados.

ABSTRACT

LOURENÇO, G. V. Analysis of distributed algorithms for scheduling in Data Grids. 2012. 108 p. Dissertação (Mestrado em Ciências) - Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2012.

It is a known result that in Data Grids, where the processing involves large amounts of data, can be more effective schedule processes to run on sites that already have the data than transferring data to a site where the process that will require them was installed. The existing studies are based on small numbers of sites, with centralized knowledge about the state of the various sites. This option is not scalable for grids with large numbers of participants. This paper will propose distributed versions with local information for process scheduling algorithms and data replication, showing the effect of interconnect topologies on the performance of these sites. It is observed that, considering the existence of only local information due to topological constraints, different results related to the best scheduling algorithms and data replication processes are found.

Keywords: Data Grid. Logical network. Process scheduling. Data replication.

LISTA DE FIGURAS

Figura 2.1 -	Redes Overlay	26
Figura 3.1 -	Distribuições utilizadas na simulação (na ordem): Aleatória, Geométrica (fator 0,98) e Zipf (fator 0,68).	34
Figura 3.2 -	Topologia Torus em formato plano	39
Figura 3.3 -	Exemplo de topologia Erdos-Rényi.....	40
Figura 3.4	Exemplo de topologia Barabási-Albert	41
Figura 4.1	Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia de Grafo Completo	47
Figura 4.2	Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia Torus.....	49
Figura 4.3	Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia Erdős	50
Figura 4.4	Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia Barabási	51
Figura 4.5	Tamanhos médios de fila de processos para algoritmo Aleatório e Local em Barabási	52
Figura 4.6	Tempos médios de fila de processos para algoritmos de escalonamento Local e Aleatório em topologia Barabási	52
Figura 4.7	Tamanhos médios de fila de processos para algoritmo Aleatório e Local em Erdos	53
Figura 4.8	Tempos médios de fila de processos para algoritmos de escalonamento Local e Aleatório em topologia Erdos	53
Figura 4.9	Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási alterando visibilidade	54
Figura 4.10	Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási usando diferentes distribuições entre arquivos e processos	55
Figura 4.11	Tempos médios de execução de processos para os algoritmos de escalonamento em Erdős com diferentes quantidades de dados	57

Figura 4.12	Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási com menos dados	57
Figura 4.13	Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási com menos dados, na primeira configuração	58
Figura 4.14	Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási com menos arquivos e maior limite de popularidade	59
Figura 4.15	Quantidade total de dados trafegados na rede por replicação passiva para topologias de Grafo Completo, Torus, Erdős e Barabási	61
Figura 4.16	Quantidade total de dados trafegados na rede por replicação passiva para topologia de Grafo Completo com diferentes limites de popularidade	63
Figura 4.17	Quantidade total de dados trafegados na rede por replicação passiva para topologia Barabási com diferentes limites de popularidade	64
Figura 4.18	Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com diferentes níveis de visibilidade para o escalonador	65
Figura 4.19	Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com diferentes números de dados	66
Figura 4.20	Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com distribuições aleatória e geométrica	67
Figura 4.21	Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com distribuição Zipf	67
Figura 4.22	Quantidade total de dados trafegados na rede por replicação ativa para topologias Grafo Completo, Torus, Erdős, Barabási	68
Figura 4.23	Quantidade de dados trafegados na rede por replicação ativa para Barabási com diferentes limites de popularidade	71
Figura 4.24	Quantidade de dados trafegados na rede por replicação ativa para Barabási com limite de popularidade dois	72
Figura 4.25	Quantidade de dados trafegados na rede por replicação ativa para Torus aumentando a visibilidade	73
Figura 4.26	Quantidade de dados trafegados na rede por replicação ativa para Erdős aumentando a visibilidade	73

Figura 4.27	Quantidade de dados trafegados na rede por replicação ativa para Barabási aumentando a visibilidade	73
Figura 4.28	Quantidade de dados trafegados na rede por replicação ativa para Barabási aumentando a visibilidade, com limite de popularidade 1	74
Figura 4.29	Quantidade de dados trafegados na rede por replicação ativa para Barabási com diferentes quantidades de dados	75
Figura 4.30	Quantidade total de dados trafegados na rede por replicação ativa para topologia de Grafo Completo com distribuições aleatória e geométrica	76
Figura 4.31	Quantidade total de dados trafegados na rede por replicação ativa para topologia Torus com distribuições aleatória e geométrica	76
Figura 4.32	Quantidade total de dados trafegados na rede por replicação ativa para topologia Erdős com distribuições aleatória e geométrica	77
Figura 4.33	Quantidade total de dados trafegados na rede por replicação ativa para topologia Barabási com distribuições aleatória e geométrica	77
Figura 4.34	Quantidade total de dados trafegados na rede por replicação ativa para topologia Barabási com distribuição Zipf	78
Figura 4.35	Percentual da quantidade total de dados replicados que resultaram em desperdício para as diferentes topologias	79
Figura 4.36	Percentual ocioso do sistema em diferentes topologias na segunda configuração	81
Figura 4.37	Percentual ocioso do sistema em diferentes topologias na primeira configuração	82
Figura 4.38	Alteração do percentual ocioso do sistema em topologia Barabási para distribuições aleatória e geométrica	83
Figura 4.39	Alteração do percentual ocioso do sistema em topologia Barabási usando distribuição Zipf	84
Figura 4.40	Consequência para o aumento de visibilidade para o percentual ocioso do sistema em topologia Barabási, para segunda configuração	85
Figura 4.41	Alteração do percentual ocioso do sistema em topologia Barabási mudando o número de dados presentes (1000 à esquerda, 10000 à direita)	85
Figura 4.42	Alteração do percentual ocioso do sistema em topologia de Grafo Completo mudando o limite de popularidade para replicação	86

Figura 4.43	Alteração do percentual ocioso do sistema em topologia de Grafo Completo ao desativar a replicação ativa	87
Figura 4.44	Alteração do percentual ocioso do sistema em topologia Barabási mudando o limite de popularidade para replicação	88
Figura 4.45	Alteração do percentual ocioso do sistema em topologia Barabási ao desativar a replicação ativa	89
Figura 4.46	Gráficos de tempos de execução em Grafo Completo para escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente e Local	91
Figura 4.47	Gráficos de tempos de execução em Barabási para escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente e Local	93
Figura 4.48	Valores de replicação ativa para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Barabási	95
Figura 4.49	Valores de replicação passiva para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Barabási	96
Figura 4.50	Valores de replicação passiva para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Menor Fila, em topologia Barabási	96
Figura 4.51	Valores de desperdício de replicação ativa para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Barabási	97
Figura 4.52	Valores de desperdício de replicação ativa para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Grafo Completo	98
Figura 4.53	Gráficos de porcentagem de desperdício de replicação ativa para diferentes algoritmos de replicação em Grafo Completo para escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente, Local	99
Figura 4.54	Gráficos de porcentagem de desperdício de replicação ativa para diferentes algoritmos de replicação em Barabási para escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente, Local	101

SUMÁRIO

1	INTRODUÇÃO.....	19
2	GERENCIAMENTO DE RECURSOS EM DATA GRIDS.....	23
2.1	Grids	23
2.2	O uso de Data Grids	24
2.3	Conexões em Data Grids	24
2.4	Escalonamento de processos e replicação de dados em Data Grids	26
2.4.1	Escalonamento Bag-of-Task.....	27
2.4.2	Algoritmos de escalonamento para aplicações seqüenciais a um grande número de usuários.....	29
2.4.3	Algoritmos de replicação de dados	31
3	DESCRIÇÃO DA SIMULAÇÃO	33
3.1	O Simulador.....	33
3.1.1	Características de processos e dados	33
3.1.2	Requisições por usuários	34
3.2	Elementos da simulação	35
3.2.1	Escalonador.....	35
3.2.2	Replicador.....	36
3.2.3	Gerenciador (escalonador local)	36
3.3	Instrumentação da simulação.....	37
3.4	Topologias investigadas	38
3.4.1	Grafo Completo	38

3.4.2 Topologia Torus	38
3.4.3 Topologia Erdos-Rényi	39
3.4.4 Topologia Barabási-Albert	40
3.5 Funcionamento da simulação	41
3.6 Atributos investigados	42
3.7 Parâmetro de controle	43
3.8 Investigações anteriores	43
4 RESULTADOS E DISCUSSÕES	45
4.1 Configuração do sistema	45
4.2 Estudo dos algoritmos de escalonamento	46
4.2.1 Valores de tempos de execução de processos para diferentes escalonamentos	46
4.2.2 Estatísticas de dados trafegados na rede como consequência do tipo de escalonamento	60
4.2.2.1 Geração de dados pelo uso de replicação passiva	60
4.2.2.2 Geração de dados pelo uso de replicação ativa	68
4.2.2.3 Desperdício de replicação ativa	78
4.2.3 Ociosidade	81
4.3 Algoritmos de replicação de dados	89
4.3.1 Tempos de execução	90
4.3.2 Dados trafegados	94
4.3.2 Replicação Ativa	94
4.3.2 Replicação Passiva	95
4.3.2 Desperdício de Replicação Ativa	97

4.3.3 Ociosidade.....	98
5 CONCLUSÕES	103
REFERÊNCIAS	107

1 INTRODUÇÃO

O meio científico realiza experimentos cada vez mais sofisticados e que para serem concluídos necessitam de grande capacidade de armazenamento de dados, além de bom poder computacional (1).

A quantidade de dados que esses experimentos geram costuma ser grande se comparada à quantidade gerada por computadores comuns, como por exemplo em experimentos de alto throughput em biologia, que geram gigabytes de dados por dia e coletas de dados astronômicos que geram terabytes de dados a cada noite (2), sendo que esse número tende a crescer nas mais variadas áreas e já atinge a ordem de petabytes (3).

Além da quantidade de dados, outros dois quesitos muito importantes caracterizam esse tipo de pesquisa: o custo e a localidade.

Para que esses experimentos sejam viáveis e que seu custo não seja tão exorbitante, é preciso encontrar um ambiente de execução dessas aplicações que não seja caro como o custo para aquisição de novos supercomputadores, mas, ao mesmo tempo, que seja realmente capaz de operar em tal escala e gerar os resultados desejados, ainda com a ideia de que o tempo para tal deva ser aceitável.

Quanto à localidade, muitos problemas científicos e tecnológicos atuais se configuram como grandes trabalhos colaborativos, onde pesquisadores de diversas instituições interagem na coleta e análise dos dados (4). Como as instituições colaboradoras estão geograficamente separadas, a coleta e análise dos dados deve naturalmente ser distribuída.

Assim, os Data Grids (5) se tornam ambientes muito buscados no meio científico e muitas vezes são indispensáveis, pois, sendo constituídos de máquinas já existentes e tendo a soma das capacidades de cada uma, oferecem um vasto repositório para os procedimentos em questão e pode realizar o armazenamento de forma distribuída, tudo isso a um bom custo. O uso de um sistema distribuído para o armazenamento permite eliminar a dependência de um único ponto de acesso, aumentar o desempenho e facilitar a inclusão de novos colaboradores.

Um exemplo é o Grid para o Large Hadron Collider (LHC) (6); mais exemplos de uso de grades de computadores para trabalhos científicos podem ser encontrados em (7).

Os Data Grids são formados por um aglomerado de máquinas que podem ser totalmente distintas e que, por estarem dispersas geograficamente, necessitam de um gerenciamento inteligente, além de uma boa organização e estruturação lógica do Grid que irá tratar os dados, para que o procedimento seja realizado da forma mais eficiente possível.

Para otimizar o uso desses Grids, mecanismos têm que ser implementados para que as tarefas sejam executadas de forma eficiente, tanto no quesito tempo quanto no que está relacionado à quantidade de dados que irão ser transportados na rede. Tendo isso como requisito, algoritmos de escalonamento de processos podem ser aliados a de replicação de dados (8), de modo a existir menor ociosidade de processos à espera por dados. Testes para ambientes mais centralizados foram realizados e mostraram que essa combinação gera bons resultados (8).

Algo que muito deve ser considerado também no funcionamento de um Grid é a maneira como os nós da rede se conectam, determinando como será o espalhamento da informação através dessa. O modo como esses nós se conectam constitui uma topologia e é estabelecida na parte lógica da rede. Novas máquinas são inseridas na rede de acordo com um protocolo pré-estabelecido e que informará como essa se conectará com outros nós, definindo como serão as conexões entre todos esses. A topologia das redes formadas tem características de redes complexas (9-11). Este trabalho contribui estudando como características dessas redes influenciam um ambiente Grid.

Para poder verificar o desempenho de aplicações em Data Grids e ver o quanto o método que combina escalonamento de processos com replicação de dados pode ser eficiente em uma rede definida por características topológicas específicas, criamos um simulador que pode considerar variados cenários com parâmetros que definem seu funcionamento, onde o registro de suas atividades nos informa o custo e o ganho de se implementar tal mecanismo, através de informações como o tempo de execução de um conjunto de tarefas, a quantidade de dados trafegados pela rede durante a execução e o quanto o sistema pode se tornar ocioso.

Com tais informações em um cenário pré-determinado no simulador, podemos alcançar nosso objetivo que é avaliar o efeito da topologia no desempenho dos algoritmos de escalonamento e replicação.

O restante desse texto está organizado da seguinte forma:

O capítulo 2 faz uma revisão de Data Grids e dos algoritmos a serem utilizados. As características do funcionamento do simulador são apresentadas no capítulo 3 e os resultados obtidos são expostos e discutidos no capítulo 4.

Por fim, o capítulo 5 apresenta as principais conclusões e algumas propostas para trabalhos futuros.

2 GERENCIAMENTO DE RECURSOS EM DATA GRIDS

2.1 Grids

Grid pode ser definido como um conjunto de máquinas heterogêneas presentes em larga escala em diferentes domínios formando um computador virtual (12).

Podemos definir também como uma infraestrutura física e lógica formada através do compartilhamento de máquinas distribuídas geograficamente (13). Tais máquinas trabalham em conjunto para, assim, fornecer um maior poder computacional ao usuário sem que esse conheça detalhes dos recursos que está utilizando, caracterizando um acesso transparente. Busca também oferecer um funcionamento escalável, de modo a sempre poder ter sua capacidade aumentada, se necessário (14).

Uma das características que justifica a utilização de Grids é permitir esse grande poder computacional através de um baixo custo. Isso é possível através do aproveitamento de recursos já existentes, ou seja, máquinas como clusteres e supercomputadores, que já possuíam suas funções anteriormente, podem ser utilizadas para formar o Grid (não sendo preciso comprar novos equipamentos) (15).

Os Grids podem ser divididas nos seguintes tipos (segundo Skillicorn (16)):

Computational Grids: tem como principal característica tentar oferecer um grande poder de processamento computacional.

Acess Grids: busca permitir interação entre um grande número de usuários (através de áudio e vídeo, por exemplo) e requer um gerenciamento de recursos específicos dentro de um conjunto muito maior de recursos. Características como processamento e armazenamento ficam em segundo plano.

Data Grids: tem como alvo aplicações de uso intensivo de dados e tem como característica mover grande quantidade desses entre os vários recursos distribuídos, necessitando de uma boa organização para que o sistema não se torne ineficiente.

Datacentric Grids: é voltado também a aplicações que necessitam de grande número de dados, mas, nesse cenário os dados se movem o mínimo possível (os processos são os que trafegam mais, nesse caso).

2.2 O uso de Data Grids

Um Data Grid nada mais é do que um aglomerado de recursos dispersos geograficamente que oferece grande capacidade de armazenamento de informação.

Tem como meta lidar com aplicações que envolvam quantidades de dados em larga escala e que necessitem de um grande nível de processamento, exigindo um bom gerenciamento para o seu funcionamento. Por exemplo, escolhas de como será a hierarquia de tomadas de decisões e de como cada nó estará visível e terá acesso aos recursos devem ser bem estabelecidas de modo a não prejudicar nenhuma das características desejáveis ao sistema.

Possui o desafio de lidar com sistemas extremamente heterogêneos, já que qualquer tipo de máquina pode ser incluída no Grid. Por ser um recurso que envolve distância física, há a necessidade de existir um meio de conexão entre os núcleos que compõem o Grid, o qual, na maioria das vezes, é a Internet, o que dificulta o desempenho do sistema, já que pode tornar-se dependente do tráfego de dados e não se torna imune a problemas nesse meio (baixa largura de banda, por exemplo).

Data Grids em estudos científicos tais como European DataGrid e o Nordic Data Grid Facility mostram um resultado atraente para esses tipos de aplicações.

2.3 Conexões em Data Grids

Sabendo-se sobre a grande quantidade de recursos e de aplicações nos Data Grids, destacamos novamente a necessidade de um bom gerenciamento desses.

Vale a pena nesse ponto definir alguns termos que serão bastante usados daqui para a frente. Um desses termos é o chamado nó, elemento mais primitivo de um Grid e que pode ser uma máquina desktop, um cluster ou até um

supercomputador. É ele quem recebe os pedidos dos usuários, que podem estar hospedados ou não nesse nó. Os pedidos feitos pelos usuários são representados em forma de processos, que são aplicações que necessitam de um processador e um conjunto de dados para executar. Dados são informações que servem de entrada para os processos dos usuários.

Os nós podem estar conectados de diversas maneiras, configurando um modelo específico de organização do Grid. Uma forma em que o gerenciamento pode ser feito é a centralizada, onde um nó enxerga todos os outros e toma as decisões. Apesar da solução no modo teórico poder parecer atraente, pois informações sobre todo o sistema são levadas em conta nessa tomada de decisões, na prática a solução não é boa, pois esse nó pode ficar sobrecarregado com a tarefa de receber e escalonar processos (tornando-se um gargalo), além do risco de falha que pode prejudicar todo o sistema envolvido, tornando-se um ponto crítico.

Uma solução é fazer com que todos os elementos tenham o poder de escolher o destino das aplicações, não sobrecarregando um nó somente, havendo uma divisão lógica de elementos do Grid que definirá para quem um nó poderá passar os processos. Com essa divisão existindo, quando uma nova máquina vai participar do Grid, deve-se ter pré-estabelecido um tipo de protocolo que irá informar a essa máquina quais recursos serão visíveis a ela.

Essas conexões que serão estabelecidas podem ser vistas como fazendo parte de uma camada superior à camada física, a camada de aplicação da rede, constituindo uma rede Overlay (17).

Redes Overlay, por definição, são redes lógicas que atuam sobre redes físicas. Redes Overlay são usadas em redes P2P (peer-to-peer), freqüentes em aplicativos de comunicação na Internet; nesse tipo de rede, a decisão de como será a conexão entre os nós independe da conexão física e sim do protocolo citado anteriormente (formando essa “camada superior”).

Figura 2.1 - Redes Overlay.

Esses tipos de rede podem ser descritos segundo o seu grau de centralização ou segundo a sua estrutura (17). A divisão dada segundo a centralização seria:

- Puramente descentralizada: todos os nós possuem as mesmas tarefas, podendo ser clientes ou servidores.
- Parcialmente centralizada: existência de super-nós (contendo informação de localização de dados, por exemplo) escolhidos dinamicamente e substituídos em caso de falhas.
- Descentralizada híbrida: há um nó central facilitando interação entre nós, informando os nós que contêm os dados.

Segundo sua estrutura:

- Não-estruturada: não há relação entre topologia e posicionamento de dados (dados precisam ser constantemente encontrados através de métodos de busca).
- Estruturada: oferece mapeamento entre dados e localização.
- Fracamente Estruturada: não possui, a princípio, a localização dos dados, mas, esta pode ser obtida indiretamente.

2.4 Escalonamento de processos e replicação de dados em Data Grids

Todo sistema responsável por executar aplicações busca obter o melhor desempenho que lhe é possível, levando em conta as limitações de hardware e as características dos processos. Como um exemplo, em CPU's tradicionais, há uma preocupação em escolher os processos de forma que executem de maneira rápida, sem prejudicar outros processos que esperam para executar também, lembrando

que nesse mecanismo de escolha de processos busca-se não deixar o processador ocioso durante a execução (em uma operação de entrada/saída, por exemplo). Essas preocupações são resolvidas no processo de escalonamento, que pondera várias características envolvidas e toma as decisões que julga necessárias segundo diferentes políticas (18).

Em Grids não é diferente, mesmo havendo um poder de processamento muito maior, é preciso um bom gerenciamento para que nenhuma aplicação seja prejudicada, lembrando que, com poderes computacionais maiores, tende-se a ter aplicações maiores. Ao utilizar, mais especificamente, Data Grids, temos uma maior preocupação também na maneira em que os dados serão armazenados.

Uma maneira de se obter um bom resultado para as aplicações nessas Data Grids é aliar métodos de escalonamento de processos com algoritmos de replicação de dados, de modo a obter uma diminuição na latência em busca de dados na execução de processos. Veremos mais à frente alguns tipos de algoritmos de escalonamento, de acordo com o tipo de aplicação utilizada.

2.4.1 Escalonamento Bag-of-Tasks

Um tipo de escalonamento em Grids é voltado para aplicações bag-of-tasks, que não será nosso enfoque por estar relacionado com paralelização de aplicações (consideramos somente processos sequenciais), mas, por caracterizar um segundo tipo de análise, torna-se interessante citar. Nesse tipo de escalonamento se conhecem todas as tarefas a serem escalonadas (geralmente, preocupa-se com várias tarefas de uma única aplicação). Eis alguns algoritmos:

2.4.1.1 Algoritmo Workqueue

Nesse algoritmo, as tarefas são enviadas aos processadores assim que esses ficam livres. Após a tarefa ser completada, o processador envia ao escalonador os

resultados, o qual reenvia outra tarefa. Tem a vantagem de não precisar de informação de performance como muitos algoritmos (19).

2.4.1.2 Algoritmo Workqueue With Replication

Igual ao Workqueue até que não se tenha mais tarefas a escalonar. Quando um processador fica ocioso, recebe réplica de tarefa que está ainda executando (provavelmente, esse processador é mais rápido que os outros); quando a tarefa termina, suas réplicas são encerradas (20).

2.4.1.3 Algoritmo Dynamic Fastest Processor to Largest Task First

Vem de outro algoritmo, o Fastest Processor to Largest Task First (20). É escolhido o processador que fornecerá o melhor tempo de execução para a tarefa através de um cálculo usando, inicialmente, três variáveis: tamanho da tarefa, velocidade do nó e carga local do host. Tarefas vão sendo alocadas em ordem decrescente de tamanho aos nós que oferecem melhor tempo e conforme os hosts vão sendo utilizados, outra variável começa a ser calculada, referindo-se ao tempo em que o host vai ficar disponível; essa tem seu valor aumentado quando hosts estão ocupados, fazendo com que o valor oferecido pelo processador se altere de acordo com o dinamismo na Grid (21).

2.4.1.4 Algoritmo Sufferage

Aqui há uma tentativa de causar o menor prejuízo em tempo de execução para as tarefas. Assim, analisam-se os melhores processadores a serem escolhidos para cada tarefa (aqueles que forneceria os melhores tempos de execução para cada uma); calcula-se qual tarefa seria a mais prejudicada se para ela fosse escolhida o segundo melhor processador ao invés do primeiro; escolhe-se então, o melhor processador para essa (22).

2.4.2 Algoritmos de escalonamento para aplicações sequenciais a um grande número de usuários

Passamos agora a não nos preocupar com aplicações paralelas e sim a um grande número de aplicações sequenciais de um grande número de usuários. Em um Grid envolvendo grande quantidade de usuários, é imprescindível que os processos requisitados por esses sejam executados de maneira otimizada, de modo que o usuário tenha uma boa resposta dos recursos utilizados.

Para que essa otimização possa existir, alguns algoritmos de escalonamento de processos podem ser aplicados, cada um com uma característica diferente para decisão (23), conforme segue:

2.4.2.1 Algoritmo Local

Nenhum tipo de decisão é feita para esse algoritmo, simplesmente o nó de onde vem o pedido (o nó onde está o usuário) é o que vai receber o processo.

2.4.2.2 Algoritmo Aleatório

Um método simples e com custo baixo de decisão é alocar o processo aleatoriamente em qualquer dos nós envolvidos no Grid. A ideia desse algoritmo é apresentar uma boa diversidade de nós escolhidos nas decisões, independentes dos recursos que oferecem, causando estatisticamente, para um número de processos muito grande em relação ao número de máquinas, uma distribuição aproximadamente uniforme de processos.

2.4.2.3 Algoritmo de Menor Fila

Outra ideia de otimização para a execução de processos é uma pré-análise dos nós envolvidos no Grid, para que os processos sejam alocados de acordo com a disponibilidade dos nós. Uma dessas análises é relativa à carga do nó, ou seja, a fila de processos que ainda precisam ser executados. A ideia desse algoritmo é, então, alocar o processo naquele que possui menor carga, ou seja, menor fila. Caso mais de um nó possua tal característica, um deles é escolhido aleatoriamente.

2.4.2.4 Algoritmo Aleatório em Menor Fila

Esse algoritmo é uma junção do Aleatório e de Menor Fila. É escolhido um grupo com um número fixo de nós com as menores filas de processos que ainda estão para executar, e, dentre os nós desse grupo, um é escolhido aleatoriamente para receber o processo.

Tem a vantagem de minimizar ações de vários escalonadores agindo ao mesmo tempo, distribuindo melhor a carga para os nós de menor fila.

2.4.2.5 Algoritmo Dado presente

Para que um processo possa executar, ele necessita, obrigatoriamente, de alguns dados durante seu processamento. Uma estratégia que pode acelerar sua execução é ter os dados requisitados antes que esse chegue no local onde irá executar, de modo que o processo não fique esperando ociosamente a chegada desses dados.

A ideia desse algoritmo é, então, alocar o processo somente nos nós que possuem os dados necessários, caso não existam tais dados nos nós que o escalonador enxerga, é feito um pedido por esses através do Grid (o que também é

chamado de replicação passiva, descrita mais à frente) e o processo é alocado em um nó aleatório a espera do dado.

Caso vários nós possuam o recurso requisitado, é feita uma escolha do nó com menor fila, tornando, assim, esse algoritmo o mais complexo do conjunto.

2.4.3 Algoritmos de Replicação de Dados

Replicação é a criação de cópias (réplicas) de dados para diferentes nós.

A utilização de replicação de dados é importante para que haja uma menor comunicação entre os nós durante a execução de processos, melhorando o tempo de execução. É importante também no caso de algum nó do sistema parar de funcionar: caso tenham sido feitas réplicas de seus dados, o sistema pode continuar a funcionar sem muitos prejuízos.

Possui a desvantagem de gerar maior tráfego de dados, podendo piorar o desempenho como um todo e também necessitar de um maior poder de armazenamento; sendo assim, torna-se necessária uma boa organização para o uso de cópias.

Os tipos de replicação podem ser divididos em (segundo Theotokis (16)):

- Replicação Passiva: a cópia do dado ocorre quando um nó precisa utilizá-lo, mas, não o possui; assim, é criada uma cópia do dado do nó que o detém e é feita a transferência.

- Replicação baseada em Cache: quando um dado é requisitado de alguma forma por um nó, esse dado muitas vezes terá que passar por nós intermediários até chegar em seu destino. Esse tipo de replicação cria cópias do dado pelos nós que passa no caminho.

- Replicação ativa: utiliza métodos para cópia de dados em outros nós para melhorar a localização e disponibilidade desses, buscando melhorar a performance das tarefas.

No Grid, alguns dados podem acabar se tornando “populares” ou seja, muitos processos precisam utilizá-los, de forma que realizar cópias em vários nós (o que é

feito pela replicação ativa), nesse caso, pode causar um funcionamento mais eficiente.

Alguns algoritmos de replicação ativa:

2.4.3.1 Algoritmo Dado para Aleatório

Um dos nós do Grid é escolhido arbitrariamente para recepção do dado. Assim como no algoritmo de escalonamento apresentado na seção 2.1.1, distribui dados sem nenhuma análise e uniformemente.

2.4.3.2 Algoritmo Dado para Menor Fila

Análogo ao algoritmo de escalonamento de Menor Fila, escolhe o nó com menor fila de processos pendentes para inserir o dado.

2.4.3.3 Algoritmo Dado para Aleatório em Menor Fila

Segue também a ideia do algoritmo de escalonamento 2.1.4, onde o dado será alocado aleatoriamente em um dos nós do grupo de menor fila.

2.4.3.4 Algoritmo Caching

As replicações anteriores realizam o que é chamado de replicação ativa. Nesse algoritmo, nenhuma replicação ativa é realizada, ao invés disso, um dado só será inserido se o processo a ser executado precisar desse e ele não existir no nó (tendo, assim, que fazer uma busca pelo dado).

3 DESCRIÇÃO DA SIMULAÇÃO

3.1 O simulador

O simulador tem por finalidade, através de um modelo simplificado, estudar alguns efeitos de topologia em algoritmos de Grid.

É criada uma rede artificial com vários nós conectados e com elementos lógicos fazendo parte do gerenciamento de recursos. Há vários usuários conectados a ela e cada link da rede tem largura de banda fixa.

Diferentes tipos de distribuições estatísticas são usadas para determinar o comportamento da simulação. Por exemplo, na definição de como os dados são distribuídos pelos nós e de qual intervalo de tempo um usuário leva para submeter um novo processo; essas distribuições serão descritas mais adiante.

É importante mencionar que a diferenciação desse simulador em relação a trabalhos anteriores é que ele utiliza uma topologia lógica para as decisões no Grid.

3.1.1 Características de Processos e Dados

Para simular um Grid em funcionamento, temos usuários fazendo requisições de processos, onde tais processos, para executar, precisam de dados específicos. Inicialmente, para que esses dados necessários estejam presentes no Grid, utiliza-se uma distribuição aleatória desses pelos nós, de modo a não existir uma preferência nos sites. Cada dado possui um tamanho fixo de 1GB (Foster e Ranganathan utilizaram valores de 0,5GB a 2GB em suas simulações (23)). Cada processo precisa de um único dado para processar e o tempo que levará executando é proporcional ao tamanho desse dado.

Cada dado no Grid pode ser requisitado por múltiplos processos; os dados são necessários aos processos segundo diferentes distribuições, que são a distribuição de Zipf (24), a Geométrica (25) ou Aleatória (escolhidas antes do início

da simulação), onde a probabilidade de um dado ter n requisições para Zipf é dada por

$$\alpha \frac{1}{n^s}$$

onde α é um fator de normalização, n é o número de requisições e s é um fator constante (utilizamos aqui fator 0,68), e para geométrica é dada por

$$(1 - s)^{n-1}s$$

sendo que utilizamos um fator 0,98 para essa. Isso implica que muitos processos podem necessitar de um mesmo dado e também pode haver um dado pouco requisitado (vai depender de qual distribuição está sendo utilizada).

Abaixo, exemplos de distribuições, onde o eixo x representa cada dado e o eixo y a quantidade de processos que requisitará cada um desses.

Figura 3.1 – Exemplos de distribuições de freqüência de pedidos utilizadas na simulação (na ordem): Aleatória, Geométrica (fator 0,98) e Zipf (fator 0,68).

3.1.2 Requisições por usuários

Os usuários são distribuídos pelos nós de modo aleatório e os pedidos de processos que esses fazem podem ser realizados de diferentes maneiras; uma das maneiras utilizadas é a que segue a política de que um usuário só pode enviar um único processo por vez, evitando o caso em que o usuário submete vários processos sem se importar com resultados anteriores. Assim, quando o processo enviado termina de executar, o usuário submete outro depois de um certo tempo, que pode

ser determinado aleatoriamente. Outra maneira é deixar o usuário livre para submeter independente do retorno dos processos submetidos, de modo que os usuários possam fazer vários pedidos em tempos próximos ou longos, tempos que também podem ser definidos aleatoriamente. Uma forma comum de se determinar os intervalos aleatórios em ambos os casos é utilizando a distribuição de Poisson (26) (onde o valor médio passado como parâmetro para a equação é que determinará a amplitude desse intervalo).

3.2 Elementos da simulação

Os nós da simulação podem possuir funções além de simplesmente alocar os processos que ali chegam. Podem se tornar os seguintes elementos:

3.2.1 Escalonador

É o elemento responsável por receber os pedidos dos usuários e decidir em qual máquina o processo será alocado. Qualquer nó pode se tornar um escalonador; no modo descentralizado, todos os nós são escalonadores e podem alocar o processo requisitado em um dos nós visíveis a ele, o que é definido segundo a topologia estabelecida. Esse arranjo de nós visíveis é determinado neste trabalho segundo os primeiros vizinhos da topologia (ou seja, os vizinhos diretos) ou pelos vizinhos dos vizinhos, onde tem sua vizinhança ampliada ao se conectar com os vizinhos dos nós vizinhos diretos, o que aumenta a “visão” do escalonador. Caso não encontre onde alocar o processo devido à ausência de dados necessários no arranjo visível (o que pode ocorrer para Dado Presente), entra em ação o replicador, que realiza uma busca por tal dado pelo Grid (ainda assim, o processo não deixa de ser alocado).

3.2.2 Replicador

É o elemento que gerencia o tráfego de dados entre os nós. Pode realizar os dois tipos de replicação, a passiva e a ativa.

A replicação passiva ocorre quando o processo a ser executado necessita de um dado que não está presente em um nó. Pode ser feita em dois momentos distintos, dependendo das decisões do Gerenciador, como veremos mais a frente.

A replicação ativa ocorre segundo os algoritmos vistos no capítulo anterior; nesse caso, o replicador analisa a popularidade dos dados no nó e cria cópias desses em outro quando o valor ultrapassa uma determinada faixa. A replicação só pode ser feita para um dos nós visíveis ao replicador, novamente, determinados pela topologia. Cada nó é também um replicador, trocando mensagens com outros nós.

3.2.3 Gerenciador (escalonador local)

É o elemento responsável por administrar os dados que chegam até ele e por oferecer recursos aos processos que precisam executar.

Gerencia os processos por meio da política FIFO (First In First Out), ou seja, o primeiro elemento que chega no site será o primeiro com direito ao uso do processador. Cada nó possui uma fila única de processos, que vai diminuindo quando o processador se torna livre. Para que o processo possa executar, antes é verificado no site a presença de dados necessários a ele, chamando o replicador conforme necessário. Essa verificação pode ocorrer de três maneiras:

I) Imediata, quando o processo chega no site. Caso não esteja presente, o replicador é acionado e esse, então, verifica se o dado existe em outro nó. Caso exista, o nó detentor do dado envia-o.

II) Na hora de execução, com fila de espera. Nenhuma análise é feita quando o processo está na fila, somente quando é a vez desse executar; caso o dado não

esteja presente, o processo passa para uma fila de espera até tal dado chegar e, então, passa para o início da fila de processos para execução.

III) Na hora de execução, sem fila de espera. Igualmente ao anterior, a verificação ocorre somente na hora de execução, porém, o processo permanece em sua posição na fila, fazendo com que essa não diminua (o processador fica, conseqüentemente, ocioso).

As diferentes maneiras podem alterar o tempo ocioso do processador; a princípio, espera-se que para o primeiro caso, o tempo ocioso seja menor. Em todos os testes, utilizamos a segunda maneira.

3.3 Instrumentação da simulação

Cada processo tem seu tempo de chegada em um nó registrado, assim como seu tempo de espera na fila e seu tempo de execução. O tempo ocioso dos processadores também é registrado, assim como a quantidade de dados trafegados na rede. Uma média desses valores é calculada para os diferentes algoritmos.

A aplicação possui como parâmetros de entrada o número de nós, como esses nós estarão conectados entre si pela definição de uma topologia, o grau de vizinhança que define a visibilidade de um nó durante o escalonamento (ou seja, até qual nó ele conseguirá enxergar naquela topologia), o valor máximo de popularidade que um dado pode ter até gerar uma replicação ativa de dado (chamado aqui de limite de popularidade), o número de dados existente no Grid, o modo como os processos necessitam desses dados definido por uma distribuição e o número de usuários.

Alguns desses parâmetros fazem parte da lógica do sistema e outros são decorrentes de como o sistema se caracteriza em um determinado momento. Os da parte lógica são: topologia, grau de visibilidade, limite de popularidade. Os que representam o sistema são: número de dados e distribuição entre processos e dados.

A simulação fornece, para determinadas topologias e distribuições de dados, resultados que podem ser usados para a análise de como será a eficiência de cada

método de escalonamento e replicação ao longo dela (em relação aos valores registrados).

3.4 Topologias investigadas

No simulador criado, as topologias implementadas não representam as conexões de Grids reais; temos aqui a intenção de analisar a influência de características topológicas e, portanto, o uso de redes complexas sintéticas, definidas através de modelos, é mais apropriado. Com um estudo do efeito dessas características, pode ser possível determinar quais são as ideais para o bom desempenho do sistema. Isso permitirá também fazer o ajuste dos protocolos utilizados para construção dessas redes.

Para o estudo do efeito de topologias nas decisões do grid na simulação, foram utilizados modelos das redes complexas Barabási-Albert, Erdős-Rényi, Torus e Grafo Completo.

3.4.1 Grafo completo

Nesse tipo de topologia, todos os nós se conectam com todos os outros (27).

3.4.2 Topologia Torus

Essa topologia tem a característica de, diferentemente das que serão mostradas à frente, ser regular, pois possui o mesmo número de nós vizinhos para cada nó (todos possuem 4 vizinhos). Pode ser visualizada como a figura abaixo (28):

Figura 3.2 -Topologia Torus em formato plano.

Em uma rede visualizada de modo plano, cada nó central se liga a seus laterais e os nós da extremidade se conectam com a outra extremidade (mantendo o número de ligações individual).

3.4.3 Topologia Erdős-Rényi

A ideia dessa topologia é que cada nó possua a mesma probabilidade de se conectar a outro e que, assim, todos os nós da rede tenham quase o mesmo número de conexões (29).

Para construí-la, temos, inicialmente, uma rede onde não há conexões entre os nós. Cada nó, então, poderá se conectar a outro nó aleatoriamente, ou seja, para cada nó, é feita uma varredura com todos os outros, verificando, para cada um, se haverá conexão. Essa decisão de construir ou não uma conexão é feita com uma probabilidade fixa.

Figura 3.3 – Exemplo de topologia Erdős.

3.4.4 Topologia Barabási-Albert

Para essa topologia, a lei de potência é que define o grau de conectividade dos nós (30). A ideia é que essa distribuição permita que alguns poucos nós tenham muitas conexões e muitos outros tenham um número menor de conexões, definindo assim a característica da rede. Com o aumento da rede, nós com mais conexões tendem a ter o número dessas aumentado, seguindo a regra “rich get richer” (31).

Na prática, a criação dessa topologia é feita inicialmente com uma rede em que cada vértice é ligado com todos os outros.

A partir dessa condição inicial, cada novo vértice a ser inserido terá o mesmo número de conexões, e cada vértice que já existia tem uma probabilidade proporcional a seu grau de conectividade de ser ligado a esse novo vértice (32).

Figura 3.4 – Exemplo de topologia Barabási-Albert.

3.5 Funcionamento da simulação

Os testes foram feitos com base em uma simulação com eventos discretos.

Cada nó possui um processador e para a ligação entre os nós, utilizou-se uma banda fixa de 10 MB/s. Cada tarefa leva 300 segundos para executar. Cada rodada é repetida 15 vezes para cálculo de média de valores investigados (descritos no próximo item).

Uma rodada é iniciada com um usuário submetendo um processo; o nó onde o usuário está hospedado recebe o processo e, agindo como um escalonador, verifica para qual nó irá reenviar o processo. Após a decisão, um nó alvo recebe o processo e analisa sua carga local para hospedar o processo; se nenhum outro processo estiver utilizando seu processador, o que chegou é enviado para executar, senão, entra em uma fila. Quando um processo está para executar, é analisada a existência do dado de entrada para esse. Se esse não estiver presente, o replicador é acionado e essa faz uma busca pela rede por esse (na chamada replicação passiva) e o processo é enviado para uma fila de espera enquanto o dado não chega. Quando o dado chega nesse nó, tal processo é colocado na frente da fila de processos, sendo assim, o próximo a executar.

Se o dado necessário ao processo já existia no nó que o recebeu, a popularidade desse dado é incrementada; senão, a popularidade do dado é

incrementada no nó que irá enviar o dado. O replicador analisa a popularidade do dado no nó e, se essa ultrapassar um determinado limite, o dado é replicado por um dos algoritmos de replicação ativa.

O tempo de processamento é contabilizado desde que o processo chega no nó até o encerramento do seu processamento. A quantidade de dados replicações é registrada separadamente em replicação passiva e ativa.

3.6 Atributos investigados

Para avaliação dos resultados, são utilizadas como métricas as seguintes características do sistema:

a) Tempos de execução:

Analisar os tempos de execução dos processos inseridos no Grid é informar o quão eficiente seu sistema responde ao que lhe é requerido, mostrando o seu desempenho.

b) Ociosidade do sistema

Característica que informa o quanto o poderio do sistema está sendo utilizado; se o sistema está ocioso, seu método de uso deve ser repensado para poder obter maior aproveitamento.

c) Dados trafegados

Em um ambiente como o Grid, a quantidade de dados sobre a rede é um fator importante a ser levado em conta; um método que gera grande quantidade de dados a serem trafegados fica “refém” das características da banda utilizada.

3.7 Parâmetro de controle

Nos testes feitos, o parâmetro de controle utilizado foi o intervalo de tempo que o usuário leva para submeter novos processos. Ter os atributos anteriores “medidos” segundo o tempo entre submissões de processos é querer obter os resultados de acordo com diferentes situações do sistema (submissões com tempo menor deixam o sistema mais sobrecarregado).

3.8 Investigações anteriores

Os algoritmos de escalonamento citados podem ser utilizados em uma forma mais centralizada (um nó toma todas as decisões do sistema). Ranganathan e Foster (23) realizaram um estudo utilizando um cenário um pouco mais próximo a esse e pode ser interessante já mencionar as características que observaram (utilizaremos os nomes em português, como no restante do texto):

a) Tempos de execução

Utilizando algum algoritmo de replicação de dado em conjunto com o escalonamento, o algoritmo que melhor teve resultado para esse quesito foi o Dado Presente, vindo logo em seguida pelo Aleatório em Menor Fila. Demorando aproximadamente o dobro do tempo médio desses vem o Menor Fila, e os algoritmos Aleatório e Local obtêm performances parecidas, cerca de três vezes o tempo do Dado Presente.

Ao deixar de utilizar a replicação, o algoritmo Data Presente que, até então, era considerado o melhor algoritmo, torna-se o que tem pior rendimento, mostrando a importância da utilização de réplicas de dados para esse caso. Todos os outros algoritmos obtêm uma leve flutuação nos valores médios de tempo de execução (o que torna o algoritmo Aleatório em Menor Fila o melhor para escalonamento sem replicação).

b) Tráfego na rede

Nesse quesito, todos os algoritmos são similares quando não há replicação de dados, a não ser o Dado Presente que não causa nenhum tráfego (pois o processo é inserido já no local com o dado necessário e o nó central enxerga todos os outros), sendo assim, o melhor.

Ao se utilizar novamente o recurso de replicação, o algoritmo Dado Presente se mostra mais uma vez superior aos demais, com um valor cerca de nove vezes menor que o do segundo melhor (Aleatório em Menor Fila). Com o terceiro melhor valor vem o Menor Fila, com cerca de 14 vezes a quantidade de dados trafegados do primeiro, e por último vêm o Aleatório e o Local, com valores parecidos (cerca de 16 vezes o Dado Presente).

c) Ociosidade do sistema

Referindo-se à quantidade de tempo em que processadores ficam ociosos, ao não se utilizar replicação, o algoritmo que tem pior performance é, novamente, o Dado Presente, sendo seguido pelo Aleatório e Local (com valores médios de tempo ocioso menores que a metade do valor do Dado Presente). O que possui o menor tempo médio ocioso é o Aleatório em Menor Fila (menos de 6 vezes o tempo do Dado Presente) e em segundo lugar vem o Menor Fila (4 vezes menos).

Ao se utilizar replicação, a tendência se repete: Dado Presente com o melhor valor, seguido bem próximo pelo Aleatório em Menor Fila, depois pelo Menor Fila, em quarto pelo Local e por último pelo Aleatório (se torna realmente inferior aos outros ao se utilizar Dado para Aleatório em conjunto).

4 RESULTADOS E DISCUSSÕES

Esse capítulo começará, primeiramente, explicando a configuração do sistema criado, pois é preciso definir um cenário inicial para fazermos nossas análises; após definida essa configuração, poderemos ver como o sistema se comporta e o que ocorre ao variar alguns parâmetros.

Analisaremos o desempenho dos algoritmos de escalonamento de acordo com os tempos de execução, tráfego de dados na rede e o tempo ocioso do sistema. Depois, veremos a implicação de se alterar os algoritmos de replicação de dados.

4.1 Configuração do sistema

A simulação em estudo utilizou, em todos os casos, uma rede constituída por 300 nós com um processador cada um. O número médio de vizinhos para as topologias Erdős e Barabási era quatro (para equivalência com Torus, que tem um número fixo de vizinhos igual a quatro também).

Para a submissão de processos, havia 1.200 usuários espalhados pelos nós e 1.000 ou 10.000 dados (requisitados por esses processos, segundo as diferentes distribuições explicadas no capítulo 3) iniciando na rede; para a definição do posicionamento dos usuários e dados, foi utilizada uma distribuição aleatória. Cada usuário submetia, em média, 50 processos durante toda a simulação.

Diferentes faixas de limite de popularidade de dado para replicação ativa foram utilizadas, em sua maioria, zero a três requisições. A popularidade de um dado define quando esse será replicado de acordo com o número de requisições para esse (daí a classificação do quão “popular” é o dado).

4.2 Estudo dos algoritmos de escalonamento

Para cada algoritmo de escalonamento que iremos investigar, os tempos de submissão de processos pelos usuários variaram na simulação de 200 a 3000 segundos, com intervalos para análise de 50 segundos (ou seja, depois de executar com tempo médio entre submissões igual a 200s, o próximo tempo médio a ser utilizado é o de 250s, e assim em diante) para cada rodada da simulação.

O algoritmo de replicação de dado para as comparações entre os diferentes algoritmos de escalonamentos foi o algoritmo de Dado para Aleatório.

Iremos nesse estudo mudar os cinco parâmetros descritos no capítulo 3: a topologia, a visibilidade, a distribuição para requisição de dados, o número de dados e os limites de popularidade para replicação. Assim, enxergaremos o sistema em diferentes condições. Começemos agora as análises:

4.2.1 Valores de tempos de execução de processos para diferentes escalonamentos

Para analisar como seriam os tempos médios de execução dos processos para cada algoritmo de escalonamento, o limite de popularidade de dados utilizado foi, inicialmente, de uma requisição (ultrapassando esse valor, o dado é replicado, sendo que depois da replicação, a popularidade é zerada). Nesse cenário, os processos submetidos pelos usuários necessitam de determinados dados segundo uma distribuição aleatória (lembrando que cada processo necessita somente de um dado para processar); 10.000 dados começaram na rede.

a) Tempos de execução para diferentes topologias

O conjunto de nós conhecidos por cada nó escalonador é determinado considerando, de início, os primeiros vizinhos determinados por cada topologia. Os resultados para os algoritmos em cada topologia foram:

- Tempos para topologia em Grafo Completo:

Figura 4.1 - Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia de Grafo Completo.

Podemos ver de imediato, em relação à forma geral da figura, que ao diminuirmos o tempo entre submissões, aumentamos o tempo de processamento. Explicaremos isso detalhando o comportamento da curva.

Observa-se, primeiramente, que os valores de tempo de execução decaem de modo linear com o aumento do intervalo entre submissões até um valor próximo de 1000s para tempo entre submissões para a maioria dos algoritmos.

Isso ocorre porque, se analisarmos um intervalo de tempo fixo grande o suficiente, ao diminuirmos o tempo médio entre submissões, um maior número de processos chega a um nó nesse intervalo (lembrando que cada processo necessita do mesmo tempo para executar nesse cenário da simulação), aumentando, assim, o tamanho das filas e demorando mais para o processo concluir sua execução a partir do momento que chegou no nó (ou seja, a resposta é mais demorada). Portanto, diminuindo o tempo entre submissões, ocorrerá proporcionalmente um acréscimo no tempo de espera em fila desse processo, o que aumenta seu tempo de processamento.

Depois desse decaimento, há uma mudança na curvatura até chegar a um valor aproximadamente constante (que é quando as filas começam a se tornar praticamente inexistentes).

Quanto ao desempenho de cada algoritmo de escalonamento, vemos que o único algoritmo que se destaca em grande escala é o algoritmo Local. É possível observar também que a mudança de curvatura descrita acima ocorre de modo muito mais lento para ele, não sofrendo a mudança mais brusca dos outros algoritmos. Isso indica que, para esse algoritmo, quase sempre haverá fila em alguns nós, prejudicando o desempenho, a não ser que o tempo entre submissões seja realmente grande. Espera-se que seu desempenho não seja bom por não poder enviar um processo a outro nó, desprezando qualquer status daquele que o recebe; veremos o porquê dele ser pior ao diferenciá-lo do Aleatório.

O Aleatório é o segundo pior, mas, com uma grande diferença em relação ao Local. Pode-se, a princípio, acreditar que o algoritmo Aleatório deveria ter um desempenho semelhante ao anterior, pois a distribuição de usuários pelos nós é também aleatória e, se no algoritmo Local os processos são inseridos exatamente onde esses usuários estão, espera-se ter um escalonamento de caráter randômico também. Porém, uma pequena variação para a distribuição de usuários (suponha que um nó recebeu mais usuários que os demais) gerará um maior desbalanceamento de carga para o algoritmo Local, já que os processos obrigatoriamente terão que executar no nó que os “abriga”, gerando maior quantidade de filas, aumentando, assim, o tempo médio de execução dos processos para esse algoritmo, explicando seu maior tempo de execução em relação ao Aleatório.

Os algoritmos de Menor Fila, Aleatório em Menor Fila e Dado presente possuem os melhores tempos, com superioridade do Dado Presente, mostrando que análise de carga e principalmente a não espera pela chegada do dado melhoram os tempos de processamento, seguindo o desempenho esperado de acordo com o artigo de Ranganathan e Foster (23).

- Tempos para topologia Torus:

Figura 4.2 - Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia Torus.

Novamente, o algoritmo que mais se destaca é o Local, mas, pode-se observar que o algoritmo Aleatório e o Aleatório em Menor Fila apresentam comportamento distinto dos outros algoritmos e mudam seu desempenho em relação à topologia anterior.

O algoritmo Aleatório em Menor Fila torna-se prejudicado nessa topologia, pois a vantagem que possuía ao escolher um grupo de nós para não ocorrer de um mesmo nó de menor fila ser escolhido por vários, acaba se tornando sua desvantagem, pois, como a quantidade de vizinhos é pequena nessa rede, o algoritmo passa a funcionar praticamente como o Aleatório, tendo seu desempenho parecido com esse.

Em relação ao Aleatório, o que podemos ver aqui é que ele sofre consequências relacionadas à localidade em suas decisões.

Dissemos anteriormente que se existem mais submissões para um mesmo nó, o desempenho do algoritmo de escalonamento se torna ruim (caso do algoritmo Local). É o que ocorre para o algoritmo Aleatório em Torus, pois, devido ao número limitado de vizinhos para os quais pode distribuir processos (no caso, só há quatro vizinhos), há maior probabilidade de mandá-los ao mesmo nó. Os valores de tempo de execução de processo para os outros algoritmos são bem próximos dos valores obtidos com a topologia anterior.

- Tempos para topologia Erdős:

Figura 4.3 - Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia Erdős.

Nessa topologia, a maioria dos algoritmos continua com um comportamento semelhante e o Aleatório em Menor Fila volta a ter um bom desempenho, pois a irregularidade da rede faz com que perca a característica aleatória dominante, mesmo que fique ainda um pouco atrás do Menor Fila e Dado Presente. Porém, o algoritmo aleatório acaba piorando seu desempenho com esse aumento da heterogeneidade.

Como já mencionado, o algoritmo Aleatório depende de problemas de localidade, o que pode gerar grandes tamanhos de filas. Gráficos com tal aspecto estão na descrição da próxima topologia.

- Tempos para topologia Barabási

Figura 4.4 - Tempos médios de execução de processos para diferentes algoritmos de escalonamento em topologia Barabási.

Confirmando a tendência notada nos gráficos anteriores, o algoritmo Aleatório piora cada vez mais seu desempenho, chegando a ser pior que o algoritmo Local para essa topologia que apresenta maior heterogeneidade na distribuição de graus.

A mudança na curvatura para o algoritmo Aleatório se torna ainda mais suave conforme o aumento da irregularidade da quantidade de vizinhos das topologias, adquirindo forma semelhante ao Local na mais irregular, indicando, assim, uma grande existência de filas para todos os tempos de submissão analisados.

A heterogeneidade mais alta faz com que o desbalanceamento de carga do algoritmo Aleatório seja ainda maior do que o desbalanceamento existente no algoritmo Local (esse devido às suas próprias características, descritas no item de Grafo completo), pois os hubs (nós com mais conexões) ficam sobrecarregados. Os nós periféricos de grau baixo (e, portanto, com poucas opções de para onde enviar os processos) tendem a ser ligados com os hubs no modelo de Barabási. Como cada hub está ligado a muitos desses nós, ele acaba recebendo muitos processos.

Para demonstrar a sobrecarga desses hubs, mostraremos um gráfico com a fila existente no nó com maior número de nós conectados e outro com a fila em um vizinho desse contendo o menor número de conexões:

Figura 4.5 - Tamanhos médios de fila de processos para algoritmo Aleatório e Local em Barabási. À esquerda, filas no nó “hub”; à direita, no vizinho com menos conexões do hub.

No gráfico da esquerda, temos os tamanhos de fila para o hub; é possível ver que o Aleatório possui muito mais processos em fila do que o hub segundo algoritmo Local. Novamente, essa sobrecarga acontece porque, tendo muitos outros nós conectados a ele, esse nó pode muitas vezes ser escolhido para receber processo, enquanto que no Local, esses nós vizinhos não poderiam repassar processos ao hub, não havendo o mesmo nível de desbalanceamento. No Aleatório, o hub pode passar seus processos a outros nós, mas, não há compensação em relação ao número de processos que estará recebendo. No gráfico da direita, vemos que as filas para o vizinho do hub com menos conexões são só um pouco maiores para o Local. Como resultado, um gráfico do tempo de fila médio para esses dois algoritmos fica:

Figura 4.6 - Tempos médios de fila de processos para algoritmos de escalonamento Local e Aleatório em topologia Barabási.

Com tempos de fila maiores, o algoritmo Aleatório terá piores tempos de execução.

Em Erdős, a diferença de tamanho de fila entre o hub e o nó conectado a esse com menor número de conexões é mostrado na figura abaixo.

Figura 4.7 - Tamanhos médios de fila de processos para algoritmo Aleatório e Local em Erdős. À esquerda, filas no nó “hub”; à direita, no vizinho com menos conexões do hub.

E como resultado para tempos médios de fila, temos:

Figura 4.8 - Tempos médios de fila de processos para algoritmos de escalonamento Local e Aleatório em topologia Erdős.

Podemos ver que a diferença entre os tamanhos de fila para os hubs e para o nó menos conectado não é tão grande quanto a de Barabási, sendo assim, não há o mesmo desbalanceamento, fazendo com que o algoritmo Aleatório seja ainda melhor que o Local nesse caso.

Para esse tipo de ambiente, os algoritmos de Dado Presente, Menor Fila e Aleatório em Menor Fila se mostraram com desempenho bem próximo, com leve

superioridade do Dado Presente, estando de acordo com resultados de estudos como os de Ranganathan e Foster (23).

b) Consequência do aumento de visibilidade do escalonador

Podemos ampliar a visibilidade de um nó escalonador para vizinhos dos vizinhos de acordo com sua topologia.

Os valores para tempos de execução se tornam levemente menores para vizinhos dos vizinhos utilizando Dado Presente (em torno de 10% menores), com um declive um pouco menos acentuado. Ter mais nós visíveis faz com que o algoritmo encontre mais facilmente os dados necessários para escalonar, ocorrendo menos tempo de fila e menor tempo de execução.

Os algoritmos que consideram fila fazem com que o fator visibilidade não seja tão importante quanto é para os outros, pois os de fila conseguem balancear bem a carga em ambos os casos. O algoritmo Local continua escalonando os processos nos mesmos nós, tendo sua performance inalterada.

A diferença maior ocorre para o algoritmo Aleatório na topologia Barabási, com uma melhora nos seus tempos:

Figura 4.9 - Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási alterando visibilidade; à esquerda, escalonador com visibilidade de primeiros vizinhos, à direita, de vizinhos dos vizinhos.

Isso mostra, mais uma vez, a sensibilidade do algoritmo aleatório em relação à quantidade de nós visíveis, algo observado no item de topologias.

Vemos que os nós com poucos vizinhos (ligados aos hubs) vão agora enxergar os vizinhos dos hubs, que são em grande quantidade, portanto, não vai haver muita sobrecarga em nenhum nó.

O fato de enxergar mais nós pode fazer com que o escalonador perca mais tempo em suas decisões (dependendo da complexidade dessas), mas, poderá ser compensador para algoritmos aleatórios.

c) Desempenho para diferentes distribuições entre dados e processos

Vejam os que ocorrem ao alterarmos a relação entre os processos e os dados que esses necessitam utilizando para essas relações as distribuições Aleatória, Geométrica e Zipf em Barabási:

Figura 4.10 - Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási usando diferentes distribuições entre dados e processos. Na ordem (sentido horário): aleatória, geométrica (fator 0,98) e zipf (fator 0,68).

Os resultados para a distribuição aleatória e geométrica são muito similares, porém, são um pouco diferentes para Zipf.

Temos aqui um resultado diferente quanto a desempenho analisado em estudos anteriores; em uma disposição do Grid como essa, onde poucos nós são visíveis e os processos dependem dos dados segundo essa distribuição, o algoritmo de Dado Presente, o melhor para uma visão mais centralizada, perde lugar para algoritmos que consideram apenas fila.

Isso ocorre porque a distribuição Zipf com esse fator gera significativamente mais pedidos para uns poucos dados e a replicação ativa para um pequeno número de nós não "alivia" da mesma maneira a sobrecarga; se temos visão centralizada, existe maior quantidade de nós que podem receber replicação, já com poucos vizinhos, a probabilidade de replicar para onde o dado já existe é maior, reduzindo a eficácia da replicação.

d) Desempenho para diferentes quantidades de dados presentes

Ao diminuir o número de dados no Grid, esperamos obter um maior número de operações envolvendo dados.

Utilizamos aqui uma segunda configuração, que será também bastante utilizada durante o texto; ela possui ainda primeiros vizinhos, mas, agora uma distribuição geométrica (até então, com resultados similares aos da aleatória) para dados e, para termos uma maior geração de dados, limite de popularidade igual a zero (de modo a gerar mais replicações) e um número de dados 10 vezes menor (no caso, 1.000 dados).

Com menos dados, o tempo de execução se torna um pouco maior (cerca de 5%), algo esperado pois, com uma menor disponibilidade de dados, um maior número de buscas terá que ser feito, aumentando, assim, o tempo ocioso dos processos.

Figura 4.11 - Tempos médios de execução de processos para os algoritmos de escalonamento em com diferentes quantidades de dados, em topologia Erdős. À esquerda, Grid com 1.000 dados, à direita, 10.000 dados.

A variação dos tempos possui o mesmo padrão para a maioria das topologias; para Barabási, porém, obtemos um resultado com padrão distinto:

Figura 4.12 - Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási com menos dados.

Para um número não tão grande de dados no Grid, o algoritmo de Dado Presente também parece não ser tão vantajoso, mostrando-se menos eficiente até que o algoritmo Local. O mesmo tipo de comportamento se observa para a primeira configuração, com uma piora não tão grande:

Figura 4.13 - Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási com menos dados, na primeira configuração.

A disparidade é um pouco menor pois há menos requisições a um mesmo dado, afetando menos o Dado Presente.

O porquê desse comportamento diferente será explicado na parte de algoritmos de replicação, seção 4.3.1.

e) Efeito da alteração dos limites de popularidade para replicação

Testes foram realizados com valores de limite de popularidade diferentes (0 a 3 requisições), mas, aparentemente, são fatores que não alteram o comportamento e desempenho dos algoritmos em relação ao tempo de execução para a configuração inicial do sistema.

Porém, analisando o caso anterior, onde o número de dados é menor, ao aumentarmos o limite de popularidade para replicação, o algoritmo de Dado Presente obtém resultados melhores:

Figura 4.14 - Tempos médios de execução de processos para os algoritmos de escalonamento em Barabási com menos dados e maior limite de popularidade.

O teste realizado aqui foi para popularidade igual a três (os outros itens são os mesmos: distribuição geométrica, 1.000 dados e primeiros vizinhos). É possível ver que a replicação altera bastante o resultado para o tempo de processamento, nesse caso, prejudicando, o que mostra um alerta para o uso da combinação de algoritmo de escalonamento com algoritmo de replicação de dado: nem sempre fazer uma combinação de algoritmo de escalonamento e replicação gera resultados melhores do que usar um algoritmo de escalonamento somente.

Lembrando, explicaremos as características dos itens (d) e (e) ao falarmos mais sobre a replicação de dados, item 4.3.1.

f) Principais conclusões sobre desempenho dos algoritmos em relação aos tempos de execução

Em relação às análises feitas, em especial à do item (c), vimos uma situação onde o algoritmo de Dado Presente não mais apresenta o melhor desempenho, alterando a regra de constante superioridade obtida nos estudos de Foster e Ranganathan (23). Quando temos uma distribuição na Grid onde um dado é altamente requisitado e a visibilidade para os escalonadores é muito irregular, não há uma maneira de reverter a sobrecarga ocorrida com o uso desse algoritmo, sendo assim, torna-se mais indicado utilizar algoritmos que analisam carga ao invés para escalonar.

Em relação aos itens (d) e (e), vimos que o algoritmo de replicação prejudica o desempenho do algoritmo de escalonamento Dado Presente quando há poucos dados. Em casos assim, o algoritmo de escalonamento de Dado Presente fica totalmente dependente de uma melhor redistribuição de dados pelo replicador para aumentar as opções de escalonamento desse algoritmo e, por fim, seu desempenho, fato que não ocorreu com o uso do algoritmo de replicação Dado para Aleatório. Assim, concluímos que não mais qualquer algoritmo de replicação ativa pode melhorar o resultado para Dado Presente, fato também que sempre ocorria em (23). Fica agora a suposição de que um algoritmo de replicação diferente desse poderia melhorar o desempenho nesses dois casos (o que será analisado a partir da seção 4.3) para as topologias investigadas.

4.2.2 Estatísticas de dados trafegados na rede como consequência do tipo de escalonamento

Nesse quesito, veremos o quanto de dados são replicados segundo replicação passiva, replicação ativa e replicação ativa “redundante”. A replicação passiva pode mostrar o quanto o sistema está falho na disposição de recursos e a ativa pode indicar o quanto de dados podem estar sobrecarregando a rede. As chamadas replicações redundantes mostram o quanto de tráfego é feito desnecessariamente.

4.2.2.1 Geração de dados pelo uso de replicação passiva

Utilizaremos aqui também o segundo cenário (de 1.000 dados), exigindo maior tráfego de dados. Reforçando, utilizamos conexões considerando os primeiros vizinhos, limite de popularidade 0 e, ainda, distribuição geométrica. Veremos nos gráficos abaixo as formas das diferentes curvas referentes aos algoritmos de escalonamento, onde temos a quantidade de dados trafegados em GB devido à replicação passiva no eixo vertical e os tempos de submissão em segundos no eixo

horizontal. Vejamos como esses dados são transportados para as diferentes topologias no item abaixo:

a) Geração de replicações passivas em diferentes topologias

Figura 4.15 - Quantidade total de dados trafegados na rede por replicação passiva para topologias de Grafo Completo, Torus, Erdős e Barabási (nessa ordem).

Fazendo uma comparação entre as curvas dos algoritmos, o algoritmo de Dado Presente faz sempre menos replicação passiva que os outros (não há curva em Grafo Completo para esse pois não realiza replicação passiva). Se o dado existe em um dos nós que são visíveis ao escalonador, o Dado Presente escalonará o processo nesse, já os outros podem escalonar em lugares onde não há o dado; assim, fazem mais replicação passiva.

Observamos para todas as topologias que quanto menor o tempo de submissão, maior a quantidade de replicação passiva necessária para o algoritmo de Dado Presente. Isso ocorre pois, com esse tempo menor, quando os novos processos chegam, eles provavelmente terão que entrar na fila ao invés de começarem a executar, não fazendo replicação ativa imediatamente (só ocorre replicação ativa quando o processo está para executar); assim, menor quantidade de dados existirá nos nós em um pequeno intervalo de tempo (se comparado ao tempo de execução de um processo), pois os processos que ainda estão na fila não “geraram” novos dados para os próximos processos que chegarão. Sendo assim, um maior número de replicações passivas deverá ocorrer.

Outro comportamento que é possível ver para esse algoritmo é que a quantidade de replicação diminui quando se aumenta a irregularidade da topologia. Quando temos uma topologia onde as conexões ocorrem só com nós próximos, temos que a informação demora para se espalhar, ou seja, um dado demora para ser propagado por toda a rede, exigindo, de modo global, uma maior quantidade de replicações passivas. Dizemos que em Torus as distâncias são grandes entre os nós. Já para Barabási, um hub facilmente dispersa um dado para um nó que pode estar distante do nó que possuía o dado originalmente, ou seja, após esse hub fazer uma replicação passiva e possuir o dado, muitos outros conectados a ele enxergarão esse dado também, além de poder recebê-lo por replicação ativa; dizemos, para Barabási, que as distâncias são pequenas. Em Erdős, também temos essa característica de distâncias pequenas, mas, em uma escala menor que em Barabási, justificando o resultado.

Outro algoritmo que segue o comportamento de diminuir replicações passivas conforme as topologias vão ficando mais irregulares é o Aleatório. Isso porque, nas mais irregulares, os hubs são frequentemente escolhidos, resultando em que esses acabam já tendo os dados depois de um tempo.

O algoritmo de menor fila possui também uma queda considerável na quantidade de dados replicados com o aumento do tempo entre submissões para a topologia Barabási. É possível que, com um tempo maior para chegada dos processos, haja mais tempo para os dados (por replicação ativa) chegarem, fazendo com que o desperdício de replicação ativa seja menor (essas replicações ativas “naturais”, ou seja, que não são consequência de uma replicação passiva, surtem efeito). Com menos replicações passivas ocorrendo, menos replicações ativas

também serão geradas por essas (não ocorrem as replicações ativas "não naturais" e sim decorrentes somente do uso no nó e não decorrente de uma replicação passiva). Veremos que a quantidade de replicação ativa também diminui, assim como o desperdício dessa.

b) Efeito da alteração dos limites de popularidade para replicação (0, 1 e 2):

Ao mudarmos os limites de popularidade para replicação, teremos os seguintes efeitos:

- Efeito em Grafo Completo:

Figura 4.16 - Quantidade total de dados trafegados na rede por replicação passiva para topologia de Grafo Completo com diferentes limites de popularidade (na ordem: 0, 1 e 2).

- Efeito em Barabási:

Figura 4.17 - Quantidade total de dados trafegados na rede por replicação passiva para topologia Barabási com diferentes limites de popularidade (na ordem: 0, 1 e 2).

Novamente, não foi inserida a curva lilás para Grafo Completo, pois Dado Presente não exige replicação passiva para essa topologia (ou seja, possui valor igual a zero).

Sobre o comportamento das curvas, vê-se que quanto maior o limite de popularidade para replicação de dado, uma maior quantidade de replicação passiva é necessária, já que demorará mais para ocorrer uma replicação ativa e, sendo assim, menos dados existirão no Grid.

c) Alteração da quantidade de réplicas passivas com aumento de visibilidade do escalonador:

O que ocorre ao aumentarmos a visibilidade do escalonador é visto na sequência: do lado esquerdo, primeiros vizinhos; do direito, vizinhos dos vizinhos.

- Barabási:

Figura 4.18 - Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com diferentes níveis de visibilidade para o escalonador; na ordem: primeiros vizinhos e vizinhos dos vizinhos.

É possível ver que o algoritmo que mais sofre mudança é o de Dado Presente; quanto menos vizinhos o escalonador enxerga, mais difícil é encontrar o dado necessário ao processo. Enxergando muitos vizinhos, o algoritmo de Dado Presente quase não necessita de replicação passiva, pois insere o processo no nó que tem o dado; ao limitarmos a visão do escalonador, o algoritmo vai obtendo características mais similares às dos outros, pois terá que fazer replicação passiva em mais casos.

d) Quantidade de réplicas passivas para diferentes quantidades de dados presentes:

Figura 4.19 - Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com diferentes números de dados; 1.000 à esquerda, 10.000 à direita.

O aumento de replicação passiva é também uma decorrência direta da quantidade de replicação ativa: com um maior número de dados e, conseqüentemente, uma menor requisição de cada dado por vários processos (pois o número total de processos não muda), um menor número de replicações ativas serão geradas, o que oferecerá um menor número de dados disponíveis, necessitando, assim, de mais replicação passiva.

e) Quantidade de réplicas passivas em diferentes distribuições entre dados e processos:

Começaremos aqui com a primeira configuração (para ver também o comportamento dessa) e depois iremos alterando a distribuição. Primeiramente, teremos distribuição aleatória à esquerda e geométrica à direita (adianta-se aqui que as mesmas tendências se observam com a outra configuração).

Figura 4.20 - Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com distribuições aleatória e geométrica.

Esse resultado é também uma decorrência da quantidade de replicações ativas. Na distribuição geométrica, um maior número de processos necessitam de um mesmo dado; assim, esse dado gerará um grande número de réplicas pois se tornará popular ao longo da execução. Tendo mais dados espalhados pelo Grid, uma configuração que segue uma distribuição geométrica necessita de menos replicação passiva.

Para distribuição Zipf, temos o seguinte gráfico:

Figura 4.21 - Quantidade total de dados replicados na rede por replicação passiva para topologia Barabási com distribuição Zipf.

O mesmo raciocínio se aplica à distribuição Zipf, porém, a diminuição ocorre em uma escala muito maior (quantidades ainda menores de replicação passiva), pois existem dados bem mais populares que geram quantidade maior de replicação ativa.

4.2.2.2 Geração de dados pelo uso de replicação ativa

a) Quantidade de réplicas ativas para diferentes topologias

Figura 4.22 - Quantidade total de dados trafegados na rede por replicação ativa para topologias (na ordem): Grafo Completo, Torus, Erdős, Barabási.

Para começar a explicar esse item, vale ressaltar uma característica consequente da configuração utilizada (que é a segunda configuração, com distribuição geométrica, 1.000 dados e limite de popularidade 0); se cada replicação ativa ocorre quando um processo executa (para o limite de popularidade atual, equivalente a zero), é coerente esperar ter uma quantidade de replicações ativas equivalente ao número de processos (valor próximo a 60000).

Quando uma replicação passiva ocorre, a popularidade do dado é incrementada no nó que contém o dado (mas, não no nó que recebe), gerando somente uma replicação ativa, mantendo ainda a relação de uma replicação ativa para uma execução. Assim, seria esperado ter uma constante ao invés de curvas, porém, isso não ocorre para topologias diferentes de Grafo Completo, principalmente em Torus, que é onde ocorre a maior quantidade de replicação ativa.

Essa propriedade ocorre porque, com menores possibilidades de escalonamento (o que é mais grave em Torus que só possui um pequeno grupo de nós à volta de um desses), há uma probabilidade relativamente alta de ocorrer uma replicação ativa desnecessária (veremos isso no item 4.2.2.3). Na configuração presente, uma replicação passiva gera uma ativa (pois o limite de popularidade é zero), só que essa replicação gera um dado que deverá ser tratado diferentemente, não gerando uma nova ativa no nó destino quando for recebido (já que foi “chamado” por replicação passiva). Porém, se outro nó enviar o dado a esse nó por uma replicação ativa normal (sem ser consequência de uma passiva), o nó destino poderá então gerar uma nova ativa, já que a origem do dado não foi de um pedido por esse. Ocorrendo isso, quebra-se a regra de que um processo gera somente uma replicação ativa para essa configuração e, como Torus é o que tem maior probabilidade de enviar o mesmo dado para um mesmo nó, é onde mais ocorrerá esse tipo de situação.

Quanto ao comportamento das curvas, os algoritmos, fora o Dado Presente, possuem um comportamento tendencioso até a região do “cotovelo”. Até nesse ponto, a quantidade de replicações ativas desses aumenta. O evento descrito que mais ocorre em Torus acontece quando os processos estão na chamada fila de espera (ou seja, estão prontos para executar, mas, esperando por dado para começar). Em tempos entre submissões menores, as filas totais são maiores. Quando as filas são maiores, pode ser que quando o dado chegue por replicação ativa, muitos processos estejam na frente desse e ele permaneça por mais tempo na

fila (não na fila de espera), não ocorrendo tal situação. Quando as filas vão diminuindo, os processos passarão maior tempo na fila de espera, havendo maior probabilidade de acontecer o caso mencionado. Com tempos muito grandes, haverá mais tempo do dado dispersar e menor chance de ocorrer as replicações simultâneas, de acordo com o que mencionamos em 4.2.2.1, item a.

Já para Dado Presente, em todas as topologias exceto Grafo Completo, é possível ver que a variação da quantidade de dados também não é tão grande ao mudarmos o tempo entre submissões (cerca de 1,6% da quantidade total); mas, é possível enxergar uma tendência: quanto maior esse tempo entre submissões, menor a quantidade de replicação ativa. Ocorre aqui o que já mencionamos, as replicações desnecessárias. Como um número menor de nós são visíveis ao utilizar uma topologia diferente de Grafo Completo, o algoritmo de Dado Presente muitas vezes pode ser obrigado a inserir o processo em um nó que não contém o dado; quando chega o momento de execução desse processo, ele é encaminhado para a fila de espera e uma replicação passiva ocorre para que esse possa executar futuramente. Enquanto esse processo espera pela chegada do dado, pode ser que esse chegue pelo mecanismo de replicação ativa realizado em outro nó (um nó diferente do que recebeu o pedido da replicação passiva), sendo assim, mais de uma replicação ativa acontecerá para um mesmo processo, quebrando a já mencionada regra de uma replicação para uma execução. Quanto maior a fila de espera (o que para Dado Presente acontece com menores tempos entre submissões), maior a chance disso ocorrer (como isso raramente acontece, a alteração é pequena, justificando a variação de 1,6%).

b) Efeito da alteração dos limites de popularidade para replicação

Aumentando os limites de popularidade, indo de zero para um e depois dois:

Figura 4.23 - Quantidade de dados trafegados na rede por replicação ativa para Barabási com diferentes limites de popularidade, na sequência: zero, um e dois.

Vemos que, quanto maior o limite de popularidade, menos replicação ativa, algo esperado pois, com um limite alto, menor a frequência de replicações ativas. Uma ressalva deve ser feita: se observarmos a curva de Dado Presente para limite de popularidade maior que zero, notamos que há também um comportamento curvilíneo para o algoritmo (gráfico abaixo em maior escala). Explicamos no item (a) que a curva se deve muito ao fato de usarmos limite de popularidade zero, o que gera mais a situação das redundâncias que descrevemos.

Figura 4.24 - Quantidade de dados trafegados na rede por replicação ativa para Barabási com limite de popularidade dois.

Para popularidade maior, a curva se deve a outro motivo. Mudando a escala para analisarmos a curva usando limite de popularidade 2, é possível ver uma grande dependência da replicação em relação aos tempos entre submissões.

Com tempos entre submissões menores, as filas serão naturalmente maiores e, assim, existirá maior quantidade de processos em um mesmo nó precisando do mesmo dado, incrementando bastante a popularidade dos dados requisitados.

Quando o limite de popularidade vale dois, é preciso três processos necessitando um mesmo dado para replicar; quando as filas são menores, o que ocorre para tempos entre submissões maiores, pode ser que essas não cheguem a ter uma quantidade suficiente de processos requisitando um mesmo dado para alcançar o limite de popularidade estabelecido, não ocorrendo replicação.

Com probabilidade maior de existência de processos no mesmo nó requisitando um mesmo dado para tempos menores entre submissões, mais replicações ativas ocorrerão nessa situação.

c) Alteração da quantidade de réplicas ativas com aumento de visibilidade do escalonador

Vejam agora o que ocorre ao ampliarmos a visibilidade para vizinhos dos vizinhos (à direita):

Aleatório —+—
 Menor Fila —x—
 Aleatório em Menor Fila —*—
 Dado Presente —□—
 Local —■—

- Torus

Figura 4.25 - Quantidade de dados trafegados por replicação ativa em Torus aumentando visibilidade.

- Erdős

Figura 4.26 - Quantidade de dados trafegados por replicação ativa em Erdős aumentando visibilidade.

- Barabási

Figura 4.27 - Quantidade de replicação ativa em Barabási aumentando visibilidade.

Partindo do ponto que, aumentando a visibilidade, menos replicações redundantes ocorrem, temos uma menor quantidade de replicação ativa. Com isso, o comportamento das curvas fica cada vez mais próximo daquele observado em Grafo Completo.

Ao mudarmos, porém, o limite de popularidade para 1, teríamos resultados diferentes ao ampliar a visibilidade. Isso porque não teríamos o mesmo caso com replicações redundantes. Vejamos abaixo:

Figura 4.28 - Quantidade de dados trafegados na rede por replicação ativa para Barabási aumentando a visibilidade, com limite de popularidade 1.

Para Dado Presente: com distribuição geométrica e um maior número de nós visíveis, mais acessos o dado de um nó poderá receber (pois a distribuição gera mais requisições para um mesmo dado), gerando mais replicações ativas.

Para algoritmos que não buscam dado para escalonar, quanto maior o número de nós visíveis, a dificuldade de colocar um processo em um nó já com o dado pode ser maior (com menos nós, a probabilidade de colocar em um nó que já recebeu o dado é maior); com isso, a replicação ativa será levemente menor. Além disso, menos replicações redundantes ocorreriam (o que gera menos replicação ativa).

d) Quantidade de réplicas ativas para diferentes quantidades de dados presentes

À esquerda, 1.000 dados e á direita, 10.000 dados.

Figura 4.29 - Quantidade de dados trafegados na rede por replicação ativa para Barabási com diferentes quantidades de dados.

Com mil dados e o mesmo número de processos executando, há mais referências a um mesmo dado do que se tivéssemos 10 mil dados, o que fará com que os dados tenham uma maior popularidade e, assim, uma maior quantidade de replicação ativa seja realizada.

e) Quantidade de réplicas ativas em diferentes distribuições entre dados e processos

Para compararmos as três distribuições, partiremos novamente da primeira configuração e e aí mudaremos o tipo de distribuição. São primeiros vizinhos, 10 mil dados, popularidade 1. Começaremos com aleatória (à esquerda) e geométrica (à direita).

Aleatório —+—
 Menor Fila —x—
 Aleatório em Menor Fila —*—
 Dado Presente —□—
 Local —■—

- Grafo Completo:

Figura 4.30 - Quantidade total de dados trafegados na rede por replicação ativa para topologia de Grafo Completo com distribuições aleatória e geométrica.

-Torus:

Figura 4.31 - Quantidade total de dados replicados na rede por replicação ativa para topologia Torus com distribuições aleatória e geométrica.

- Erdős:

Figura 4.32 - Quantidade total de dados replicados na rede por replicação ativa para topologia Erdős com distribuições aleatória e geométrica.

- Barabási:

Figura 4.33 - Quantidade total de dados replicados na rede por replicação ativa para topologia Barabási com distribuições aleatória e geométrica.

Para Zipf:

Figura 4.34 - Quantidade total de dados replicados na rede por replicação ativa para topologia Barabási com distribuição Zipf.

Como visto no trecho de replicação passiva, comparando a distribuição aleatória com a geométrica, há mais replicação ativa para distribuição geométrica pelas maiores popularidades dos dados devido à característica da distribuição, o que gera um maior número de réplicas.

Em Zipf isso também ocorre, o número de replicações ativa é maior pois há mais requisições para um mesmo dado.

4.2.2.3 Desperdício de replicação ativa

Nem todos os dados replicados pelos algoritmos de replicação ativa precisam ser inseridos nos nós de destino, pois, muitas vezes ele pode já existir naquele nó (por uma replicação passiva ou por uma replicação ativa anterior), podendo ser uma operação desnecessária. Vejamos quando a replicação ativa é realmente necessária nos nós, em gráficos de “desperdício” de replicação ativa:

Figura 4.35 - Percentual da quantidade total de dados replicados que resultaram em desperdício para as diferentes topologias (na ordem): Grafo Completo, Torus, Erdős e Barabási.

Observamos aqui que o algoritmo de Dado Presente ainda segue uma pequena dependência com os tempos entre submissões. Vimos anteriormente que com valores baixos desses tempos, um maior número de replicações passivas acontece; assim, após vários dados serem inseridos nos nós dessa maneira, quando ocorrer replicação ativa, os dados têm uma probabilidade maior de estarem presentes e a replicação ativa terá sido desnecessária.

O algoritmo de Menor Fila tem também seu desperdício diminuído de acordo com esses tempos entre submissões em Barabási, de acordo com o que foi afirmado em 4.2.2.1, item a.

Para todos os casos, o algoritmo de Dado Presente é o que possui maior eficácia na replicação dos dados, ou seja, acaba enviando mais dados para nós que

ainda não o continham. Por enviar um processo em um nó que possui um determinado dado, a probabilidade de replicar esse dado para outro nó que também contém o mesmo é menor do que para os outros algoritmos. Os outros algoritmos, por terem que fazer uma maior quantidade de replicação passiva, possuem maior probabilidade de ocorrer uma replicação ativa para um nó que já contém o dado (a replicação passiva pode gerar uma ativa para o nó com o dado, enquanto que para Dado Presente, o processo já iria ser escalonado no nó com esse dado, esse é, então, eliminado da possibilidade de desperdício).

Em Torus, porém, o algoritmo de Dado Presente possui desperdício igual ao dos outros. Por ter uma quantidade pequena de vizinhos e estarmos utilizando uma distribuição geométrica (onde há mais processos requisitando um mesmo dado do que em relação à aleatória), logo os vizinhos vão sendo preenchidos com o dado, tendo que haver desperdício após esse “enchimento” (além da probabilidade de se enviar ao mesmo nó ser grande). Os outros algoritmos também possuem desperdício alto, pois também estão restritos em relação a quais nós podem enviar.

Em Barabási, temos um alto desperdício também. Novamente, isso se deve à existência de hubs, já que esses são frequentemente escolhidos para receber os dados.

Em Erdős, não temos os problemas de hubs e nem a mesma restrição quanto aos nós conectados, diminuindo um pouco a quantidade de desperdício de replicação ativa.

4.2.2.4 Conclusão sobre dados trafegados com o uso de replicação

Temos aqui, resultados que concordam com os estudos feitos anteriormente (ver (23)). Novamente, o algoritmo de Dado Presente é o que gera menos replicação passiva e, ainda, o que possui menor desperdício de replicação ativa. Dependendo da disponibilidade de banda que se pode ter, é algo que deve ser considerado.

4.2.3 Ociosidade

Estudaremos agora o quanto o sistema fica em estado ocioso durante sua operação. Faremos as análises variando os mesmos itens de antes.

a) Ociosidade para diferentes topologias em uso:

Para a segunda configuração (distribuição geométrica, 1.000 dados, primeiros vizinhos e limite de popularidade 0):

Figura 4.36 - Percentual ocioso do sistema em diferentes topologias na segunda configuração. Na ordem: Grafo Completo, Torus, Erdős, Barabási.

Agora, para a primeira configuração (distribuição aleatória, 10 mil dados, limite de popularidade 1):

Figura 4.37 - Percentual ocioso do sistema em diferentes topologias na primeira configuração. Na ordem: Grafo Completo, Torus, Erdős, Barabási.

Em todas as situações, o algoritmo de escalonamento de Menor Fila é o que gera menos ociosidade, devido a sua capacidade de balanceamento de carga.

O Aleatório em Menor Fila perde lugar em Torus e Erdős. Como mencionado no começo do texto (seção 4.1, item a), esse algoritmo tem um desempenho ruim para essa topologia, e é possível observar que sua ociosidade fica muito semelhante à do algoritmo Aleatório, pois ficam com o mesmo mecanismo de decisão nesse caso (não consideram a carga nos nós). Em Erdős, o algoritmo ainda perde um pouco sua eficácia, pois temos também casos onde a quantidade de vizinhos é

pequena e assim ele passa a desconsiderar a fila como requisito de escalonamento (não chegando a ter a mesma piora que ocorre em Torus).

As topologias em que o Aleatório em Menor Fila piora são as mesmas em que Dado Presente assume, juntamente com o de Menor Fila, o primeiro lugar. Quando o algoritmo de Dado Presente não encontra o dado requerido (o que muito ocorre quando a quantidade de nós vizinhos é limitada), ele funciona como o de Menor Fila e a replicação envolvida não altera muito o seu comportamento por serem poucas as opções para replicação (vimos na seção 4.2.1, item e, que a replicação prejudica o desempenho desse, não chegando a prejudicar tanto nesses casos), havendo, em pouco tempo, replicação para todos os outros nós também.

Como se esperava, o Local tem seu comportamento intacto, já que seu escalonamento independe da rede lógica. O Aleatório tem a ociosidade aumentada novamente de acordo com a irregularidade da topologia (com sobrecarga para alguns nós somente - os hubs -, muitos ficam ociosos).

b) Variação de ociosidade para diferentes distribuições entre dados e processos

Para analisar a ociosidade entre diferentes distribuições, partiremos da primeira configuração (primeiros vizinhos, 10 mil dados, limite 1) e depois mudaremos as distribuições. começando com aleatória e geométrica:

Figura 4.38 - Alteração do percentual ocioso do sistema em topologia Barabási para distribuições aleatória (esquerda) e geométrica (direita).

Vemos aqui que somente o algoritmo de Dado Presente sofre mudanças de acordo com a distribuição, os outros algoritmos não são afetados. Ele possui uma maior ociosidade utilizando a distribuição geométrica pois nós com dados mais populares receberão um maior número de processos, gerando maior sobrecarga e deixando outros desocupados e, portanto, ociosos.

Para Zipf, temos o seguinte comportamento:

Figura 4.39 - Alteração do percentual ocioso do sistema em topologia Barabási usando distribuição Zipf.

Para Zipf, como a maioria dos processos vão precisar dos mesmos dados, quando há muitas submissões em pouco tempo, as submissões acabam indo para os mesmos nós e ficando na fila, fazendo com que demore para replicar para outros nós, deixando esses outros nós ociosos. A medida que se dá mais tempo entre as submissões, tem-se mais chance de achar cópias dos dados em outros nós, diminuindo a ociosidade.

c) Alteração da ociosidade com o aumento da visibilidade:

A topologia abaixo é Barabási, com a segunda configuração à esquerda; à direita, utilizam-se vizinhos dos vizinhos.

Figura 4.40 - Consequência para o aumento de visibilidade para o percentual ocioso do sistema em topologia Barabási, para segunda configuração.

Ou seja, o fato de ter mais nós visíveis alivia um pouco o prejuízo que o Dado Presente tem ao ser utilizado junto com o algoritmo de replicação, o que indica que esse problema está relacionado com localidade e de algum modo, pode gerar sobrecarga (o que, então, deve ser corrigido por outros algoritmos de replicação de dado considerando carga, o que verificaremos na comparação entre esses).

d) Ociosidade para diferentes quantidades de dados presentes:

Figura 4.41 - Alteração do percentual ocioso do sistema em topologia Barabási mudando o número de dados presentes (1.000 à esquerda, 10.000 à direita).

Com mil dados, voltamos a ver o resultado ineficiente do Dado Presente. Veremos novamente esse comportamento no próximo item (onde utilizamos a segunda configuração) e a sua dependência direta com a utilização de algoritmo de replicação.

e) Efeito da alteração dos limites de popularidade para replicação:

Voltaremos agora à segunda configuração, onde tivemos o resultado diferente do padrão no trecho de tempos de execução (seção 4.2.1, item d), com distribuição geométrica e primeiros vizinhos. Os valores de limite serão 0, 1, 2 e 3, na sequência evidenciada. Primeiramente, testando para Grafo Completo:

Figura 4.42 - Alteração do percentual ocioso do sistema em topologia de Grafo Completo mudando o limite de popularidade para replicação; na ordem: zero, um, dois e três.

Desativando a replicação ativa, temos:

Figura 4.43 - Alteração do percentual ocioso do sistema em topologia de Grafo Completo ao desativar a replicação ativa.

Vemos que na topologia de Grafo Completo a existência da replicação de dados é importante nesse quesito, pois, quanto mais tardia ela ocorre, mais ocioso se torna o escalonamento por Dado Presente.

Para Torus e Erdős, a variação do limite de popularidade pouco influencia na ociosidade dos algoritmos. Vejamos agora para Barabási:

Figura 4.44 - Alteração do percentual ocioso do sistema em topologia Barabási mudando o limite de popularidade para replicação; na ordem: zero, um, dois e três.

Ao desativar a replicação ativa, temos:

Figura 4.45 - Alteração do percentual ocioso do sistema em topologia Barabási ao desativar a replicação ativa.

É possível ver que a replicação, nesse caso, prejudica o desempenho do escalonamento de Dado Presente pois, quanto menos ocorre, menor a ociosidade. Explicaremos agora porque isso ocorre, vendo o que acontece ao mudarmos o algoritmo de replicação de dados no próximo item, diferenciando cada um desses algoritmos de replicação.

f) Principais conclusões sobre ociosidade para algoritmos de escalonamento

Após ver os últimos itens de ociosidade, em especial o item (e), fica evidente aqui a grande dependência do algoritmo de Dado Presente em relação à replicação ativa e a necessidade de escolha de uma combinação correta de algoritmos de escalonamento e replicação.

4.3 Algoritmos de replicação de dados

Analisaremos aqui o desempenho obtido ao mudarmos os algoritmos de replicação de dados executando em conjunto com os algoritmos de escalonamento (agora, não será somente o Dado para Aleatório).

Para isso, utilizaremos a segunda configuração (1000 dados, primeiros vizinhos e distribuição geométrica, “intermediária” entre aleatória e zipf, no quesito irregularidade).

4.3.1 Tempos de Execução

Veremos aqui os valores de tempo de execução obtidos variando-se o tipo de replicação. Analisando os algoritmos isoladamente para cada topologia, temos os seguintes desempenhos, a começar pelo Grafo Completo:

Figura 4.46 - Gráficos de tempos de execução em Grafo Completo na ordem: escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente e Local.

Para tempos entre submissões não muito pequenos, ou seja, que não geram grande quantidade de fila, vemos novamente que o algoritmo de Dado Presente

possui valores menores de tempos de execução (melhor na região de mudança de curvatura da figura 4.2), algo mencionado na seção 4.1. Fica evidente aqui a dependência do algoritmo com a existência de replicação ativa, pois seu desempenho piora muito com o algoritmo de replicação Caching. Sem muitos dados disponíveis no Grid, o Dado Presente fica sem opções para escalonar, gerando sobrecarga por ter que enviar muitas vezes um processo ao mesmo nó.

Os dados para Torus e Erdős não são apresentados pois pouco se nota a diferença entre os algoritmos de replicação de dados.

Continuando agora para Barabási:

Figura 4.47 - Gráficos de tempos de execução em Barabási na ordem: escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente e Local.

Outra diferença significativa entre algoritmos de replicação de dado ocorre para a topologia Barabási, onde o algoritmo de escalonamento de Dado Presente possui tempos de execução maiores utilizando a replicação aleatória.

Esses tempos são maiores pois após replicar para um nó qualquer, o algoritmo de escalonamento terá como opção somente esse nó para enviar o processo sem analisar se há sobrecarga. Como só esse nó terá o dado, um grande número de processos será alojado nesse, demorando para uma nova replicação ativa ser gerada pois estará sobrecarregado (fazendo com que a sobrecarga aumente).

Usar algoritmos de replicação que se baseiam em fila é como guiar o algoritmo de Dado Presente para os nós de menor fila também, gerando menor ociosidade e, por consequência, menores tempos de execução. Ao enviar para os nós de menor fila, não demorará para existir uma nova replicação, fazendo com que exista mais possibilidades para escalonamento e, assim, diminuindo a sobrecarga (diferente do Dado para Aleatório que pode enviar para um nó com grande e fila e demorar para haver a dispersão do dado).

Isso explica porque obtivemos resultados ruins nos itens (d) e (e) de 4.2.1: um algoritmo de replicação desse tipo não é uma boa aplicação nessas circunstâncias.

4.3.2 Dados trafegados:

Veremos aqui a quantidade de dado que é trafegada para cada algoritmo de replicação. Veremos primeiro a quantidade de replicação passiva, depois ativa e, por último, o desperdício de replicação ativa.

4.3.2.1 Replicação ativa

Os valores de dados replicados por cada algoritmo de escalonamento é dado pelas figuras da seção 4.2.2.2. A proporção entre cada algoritmo de replicação de dado tem a seguinte forma:

Figura 4.48 – Valores de replicação ativa para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Barabási.

Para o algoritmo Dado para Menor Fila, em um determinado momento (principalmente quando os tempos entre submissões são pequenos), o nó de menor fila se torna alvo de envio de dado por muitos outros nós. Assim, temos novamente o caso de um mesmo nó receber replicações ativas de vários outros em tempos próximos, fazendo com que seja possível que dados que foram requisitados por processos acabem chegando através dessas replicações ativas, gerando ainda mais replicação, tornando esse algoritmo aquele com maiores valores de replicação ativa.

Isso não ocorre para o Dado para Aleatório em Menor Fila, pois, ao escolher um nó aleatório dentro de um conjunto de nós com menor fila, evita-se de escolher obrigatoriamente o mesmo nó para receber o dado, não gerando, assim, a mesma quantidade de replicação ativa.

4.3.2.2 Replicação passiva

Para os valores de replicação passiva, temos, para quase todos os casos, o seguinte padrão:

Figura 4.49 – Valores de replicação passiva para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Barabási.

O uso do algoritmo de replicação Dado para Aleatório em Menor Fila mostra exigir, como consequência da replicação ativa, uma maior necessidade de replicações passivas em relação aos outros algoritmos de replicação ativa. O algoritmo Caching, por não envolver replicação ativa, naturalmente necessitará de mais replicação passiva.

Um comportamento diferente, porém, é observado para o algoritmo de Menor Fila:

Figura 4.50 – Valores de replicação passiva para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Menor Fila, em topologia Barabási.

Nessa ocasião, o que ocorre é que o mesmo nó que é escolhido para receber o dado por replicação ativa, é também aquele escolhido para receber o processo, fazendo com a replicação passiva não seja mais necessária.

4.3.2.3 Desperdício de replicação ativa

Para Erdős e Barabási, temos a seguinte configuração:

Figura 4.51 – Valores de desperdício de replicação ativa para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Barabási.

É possível ver que o algoritmo de replicação Dado para Aleatório em Menor Fila gera o menor desperdício de replicações ativas, enquanto que o Dado para Menor Fila é o que mais gera. Novamente, para o Dado para Menor Fila, muitos nós escalonadores podem escolher o mesmo nó para enviar o dado, aquele que tem a menor fila de processos no momento, gerando, assim, o maior desperdício. Com maiores tempos entre submissões, o desperdício diminui, pois há mais tempo suficiente para que mude o nó contendo menor fila. Como também já mencionado, o Dado para Aleatório em Menor Fila evita esse tipo de situação não escolhendo sempre o mesmo nó; não tendo vários nós escolhendo um mesmo para enviar o dado, acaba gerando menos desperdício; vendo que o conjunto de nós considerado pelo algoritmo não muda com a mesma velocidade que o nó com menor fila, o mesmo decaimento não é observado.

Para Grafo Completo, porém, a configuração passa a ser diferente:

Figura 4.52 – Valores de desperdício de replicação ativa para os diferentes algoritmos de replicação de dado. Algoritmo de escalonamento Dado Presente, em topologia Grafo Completo.

Nesse caso, temos que um mesmo conjunto de nós com menor fila observado pelo Dado para Aleatório em Menor Fila pode permanecer por um bom tempo como alvo de vários nós, o que faz com um mesmo nó possa ser escolhido várias vezes para receber um dado (lembrando que aqui existe uma quantidade bem maior de nós querendo enviar dado para esse conjunto), gerando, assim, maior desperdício. Isso já não ocorre mais para o Dado para Menor Fila, pois, em Grafo Completo, é alterado mais rapidamente o nó que possui a menor fila, não sendo escolhido muitas vezes em um curto período de tempo.

4.3.3 Ociosidade:

Por fim, veremos o quanto do sistema fica ocioso ao testarmos os novos algoritmos de replicação de dados. Variando as topologias, teremos:

- Grafo Completo:

Figura 4.53 - Gráficos de porcentagem de desperdício de replicação ativa para diferentes algoritmos de replicação em Grafo Completo na ordem: escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente, Local.

Repetindo o observado em 4.3.1, o Dado Presente necessita de um algoritmo de replicação ativa, principalmente com tempos entre submissões menores.

Poucas diferenças entre os algoritmos no quesito ociosidade para Torus e Erdős.

- Barabasi:

Figura 4.54 - Gráficos de porcentagem de desperdício de replicação ativa para diferentes algoritmos de replicação em Barabási na ordem: escalonamento Aleatório, Menor Fila, Aleatório em Menor Fila, Dado Presente, Local.

Vemos novamente o efeito negativo de se utilizar o algoritmo Dado para Aleatório com Dado Presente para Barabási.

É possível ver que os algoritmos de replicação não se tornam tão essenciais observando as diferenças de tempo de execução e ociosidade; pode existir uma melhora pequena nesses valores, mas, essa deve ser balanceada com o custo de existir maior tráfego de dados.

4.3.4 Principais conclusões sobre algoritmos de replicação

Analisando os resultados com vários algoritmos de replicação, vimos que o uso da replicação aleatória não se torna vantajoso para ambientes irregulares (ver figura 4.50). Já as replicações que consideram carga corrigem o desempenho ruim obtido pelo Dado Presente (ver mesma figura).

Algoritmos de replicação ativa podem gerar grande tráfego e, ainda assim, não fazer tanta diferença, sendo necessária, na maioria dos casos, balancear a relação custo/benefício do uso desses, contando que utilizá-los poderá custar grande uso da rede.

5 CONCLUSÕES

De estudos anteriores, tínhamos algumas ideias de como os algoritmos de escalonamento de processos e replicação de dados se comportavam em ambientes que não envolviam topologias.

Dentro de nossos objetivos, pudemos verificar como seriam esses algoritmos sobre influências topológicas.

Podemos, agora, ter algumas conclusões para esses tipos de ambiente:

A - Desempenho dos algoritmos de escalonamento

Para topologias com alto grau de heterogeneidade, o algoritmo de Dado Presente deixa de ser, como era para os casos mais centralizados, a opção mais indicada. Vimos no primeiro item de análise de algoritmos de escalonamento (sobre tempos de execução, 4.2.1) que o algoritmo de Dado Presente possui grandes restrições para o seu funcionamento; muitas configurações tendem a deixá-lo com um desempenho ruim, com resultados onde nenhuma combinação com algoritmo de replicação de dado seria suficiente para contornar o mau desempenho.

Em ambientes onde dados não se tornam muito populares, o resultado pode ser interessante, mas, em ambientes onde um mesmo dado somente é altamente requisitado e, a princípio, não existem muitas cópias desse, algoritmos que consideram como prioridade a carga dos nós se mostraram mais apropriados. Isso se torna mais agravante em redes muito heterogêneas.

B - Uso de algoritmos de replicação

Após várias combinações realizadas em ambientes diferentes, vimos que os algoritmos de replicação podem gerar um grande nível de tráfego de dados, mas, que em muitos casos não geram a melhoria que ocorria para estudos mais centralizados; é preciso, então, ver se o custo por esse será válido para sua realização, dependendo, assim, se o ambiente não terá seu funcionamento prejudicado com seu alto tráfego (ou seja, se a largura de banda é boa o suficiente para poder executá-los).

Como já mencionado, algoritmos de replicação ativa que analisam carga corrigem o desempenho ruim obtido pelo Dado Presente em ambientes irregulares, mas, não dão superioridade a esse quando o número de acessos a um dado é realmente grande (caso de Zipf, como na figura 4.11) em um ambiente muito irregular (como Barabási).

C - Implementação de uma topologia heterogênea

Essa é uma questão mais “aplicada”. Sabemos que implementar um topologia de Grafo Completo não deve ser interessante na prática, pois, o número de análises a serem feitas por um nó somente tende a ser algo muito grande e que pode aumentar, sobrecarregando em demasiado cada nó. Porém, criar um protocolo que faça conexões (se viável também implementar um protocolo desse) entre os nós segundo uma topologia de rede complexa pode não ser também a melhor ideia.

Um estudo avaliando o quanto de tempo os nós perderiam para suas decisões talvez precisasse ser feito, mas, é possível ver que uma topologia de Barabási geraria alguns cuidados para o bom desempenho dos algoritmos, como a utilização de um algoritmo de escalonamento que tenha como prioridade a consideração de carga, além da utilização de um algoritmo de replicação de dados adequado.

D- Estudos futuros

Além do cálculo de tempo para tomada de decisão pelo escalonador mencionado no item anterior, pode ser inserida também uma checagem da existência de um dado no nó que irá receber uma réplica; quando existirem réplicas muito grandes, a quantidade total de dados trafegados pode diminuir bastante com a pré-análise (mesmo necessitando de mais troca de informação na rede para ser realizado).

Mais análises sobre o efeito de características topológicas individuais podem ser feitas, como a mudança de grau médio, coeficiente de aglomeração expoente da distribuição da lei de potência, por exemplo. Também podem ser feitas mudanças na estrutura da rede, como consideração de nós heterogêneos e alteração da largura de banda.

Outros algoritmos de escalonamento e replicação, com possível maior complexidade, podem ser testados.

REFERÊNCIAS

- 1 EMMOTT, S. Towards 2020 Science. **Science in Parliament**, v. 65, n. 4, p. 31-33, 2008.
- 2 MUGGLETON, S. H. Exceeding human limits. **Nature**, v. 440, n. 7083, p. 409-410, 2006.
- 3 SZALAY, A. Extreme data-intensive scientific computing. **Computing in Science and Engineering**, v. 13, n. 6, p. 34-41, 2011.
- 4 SZALAY, A.; GRAY, J. Science in an exponential world. **Nature**, v. 440, n. 7083, p. 413-414, 2006.
- 5 CHERVENAK, A.; FOSTER, I. ; KESSELMAN, C. ; SALISBURY, C.; TUECKE, S. The Data Grid: towards an architecture for the distributed management and analysis of large scale scientific datasets. **Journal of Network and Computer Applications**, v. 23, n. 3, p. 187-200, 2000.
- 6 THE LARGE hadron collider computing grid project. Disponível em: <<http://lcg.web.cern.ch/LCG/>>. Acesso em: jan.2012.
- 7 BOGHOSIAN, B.; COVENEY, P. Guest editors' introduction: scientific applications of grid computing. **Computing in Science and Engineering**, v. 7, n. 6, p. 10-13, 2005.
- 8 STOCKINGER, H.; SAMAR, A.; HOLTMAN, K.; ALLCOCK, B.; FOSTER, I.; TIERNEY, B. File and object replication in data grids. **Cluster Computing**, v. 5, n. 3, p. 305-314, 2002.
- 9 BARABÁSI, A; ALBERT, T. Statistical mechanics of complex networks. **Review of Modern Physics**, v. 74, n. 1, p. 47-97, 2002.
- 10 NEWMAN, M. E. J. The structure and function of complex networks. **SIAM Review**, v. 45, n. 2, p. 167-256, 2003.

11 BOCCALETIA, S.; LATORA, V.; MORENO, Y.; CHAVEZ, M.; HWANG, D.-U. Complex networks: structure and dynamics. **Physics Reports**, v. 424, n. 4-5, p. 175-308, 2006.

12 KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Software, Practice & Experience**, v. 32, n. 2, p.135-164, 2002.

13 FOSTER, I.; KESSELMAN, C. Concepts and architecture .In:_____ **The grid: blueprint for a new computing infrastructure**. San Francisco: Morgan Kaufmann,1998. cap.4. p. 37-41.

14 FOSTER, I.; IAMNITCHI, A. On death, taxes, and the convergence of peer-to-peer and grid computing. **Lecture Notes in Computer Science**, v. 2735, p. 118-128, DOI: 10.1007/978-3-540-45172-3_11, 2003.

15 IRVING, M.; TAYLOR, G.; HOBSON, P. Plug in to grid computing. IEEE **Power and Energy Magazine**, v. 2, n. 2, p. 40-44, 2004.

16 SKILLICORN, D. B. Motivating computational grids. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 2, (CCGRID2002), 2002, BERLIM, ALEMANHA. **Proceedings...** Berlim, Alemanha: IEEE Computer Society, 2002. p. 401-402, DOI: 10.1109/CCGRID.2002.1017168.

17 THEOTOKIS, S. A. A survey of peer-to-peer content distribution technologies. ACM **Computing Surveys**, v. 36, n. 4, p. 335-371, 2004.

18 TANNEMBAUM, A.; WOODHULL, A. Processos. In:_____ **Sistemas operacionais: projeto e implementação**. Porto Alegre: Bookman, 1999. cap. 2. p. 69-75.

19 CIRNE, W.; PARANHOS, D.; COSTA, L.; SANTOS-NETO, E.; BRASILEIRO, F.; SAUVÉ, J.; OSHTOFF, C.; SILVA, F.; SILVEIRA, C. Running bag-of-tasks applications on computational grids: the mygrid approach. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING, 32, (ICPP 2003), 2003, KAOHSIUNG, TAIWAN. **Proceedings...** Kaohsiung, Taiwan: IEEE Computer Society, 2003. p. 407, DOI: 10.1109/ICPP.2003.1240605.

20 MENASCÉ, D.; SAHA, D. ; PORTO, S.; ALMEIDA, V. A. F.; TRIPATHI, S. K. Static and dynamic processor scheduling disciplines in heterogeneous parallel

architectures. **Journal of Parallel and Distributed Computing**, v. 28, n. 1, p. 1-18, 1995.

21 PARANHOS, D.; CIRNE, W.; BRASILEIRO F. Trading cycles for information: using replication to schedule bag-of-tasks applications on computational grids. In: INTERNATIONAL EURO-PAR CONFERENCE, 9, (LNCS 2790), 2003, KLAGENFURT, AUSTRIA. **Proceedings...** Klagenfurt, Austria: Springer, 2003, p.169-180, DOI: 10.1007/978-3-540-45209-6_26.

22 IBARRA, O. H. ; KIM C. H. Heuristic algorithms for scheduling independent tasks on nonidentical processors. **Journal of ACM**, v. 24, n. 2, p. 280-289, 1977.

23 RANGANATHAN K.; FOSTER, I. Simulation studies of computation and data scheduling algorithms for data grids. **Journal of Grid Computing**, v. 1, n. 1, p. 53-62, 2003.

24 ADAMIC, L. A.; HUBERMAN, B. A. Zipf's law and the Internet. **Glottometrics**, v. 3, n. 1, p. 143-150, 2002.

25 WEISSTEIN, E. W. Geometric distribution. Disponível em: <<http://mathworld.wolfram.com/GeometricDistribution.html>>, 2008. Acesso em: mai. 2009.

26 STIGLER, S. M. Poisson on the poisson distribution. **Statistics & Probability Letters**, v. 1, n. 1, p. 33-35, 1982.

27 CONWAY, J. H.; GORDON, C. M. Knots and links in spatial graphs. **Journal of Graph Theory**, v. 7, n. 4, p. 445-453, 1983.

28 FOSTER, I. **Designing and building parallel programs**.1995. Disponível em: <<http://www.mcs.anl.gov/~itf/dbpp/text/node33.html>>. Acesso em: maio 2009.

29 ERDŐS, P.; RÉNYI, A. On random graphs I. **Publicationes Mathematicae**, v. 6, n.1, p. 290-297, 1959.

30 BARABÁSI, A; ALBERT, T. Emergence of scaling in random networks. **Science**, v. 286, n. 5439, p. 509-512, 1999.

31 BARABÁSI, A; ALBERT, T. **Rich get richer**. Disponível em: <http://www.barabasilab.com/LinkedBook/chapters/7Ch_TheRichGetRicher.pdf>, 1999. Acesso em: jan. 2012.

32 NEWMAN, M. E. J. Power laws, pareto distributions and zipf's law. **Contemporary Physics**, v. 46, n. 5, p. 323-351, 2005.