Saturnus: Um Simulador Discreto Baseado em Eventos para Sistemas de Arquivos Paralelos

Lucas P. Bordignon[1], **Eduardo C. Inacio**[1],
Marcos A. Rodrigues[2] , **Mario A. R. Dantas**[1]

[1]Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC)
Florianópolis – SC – Brasil

[2]Sheffield Hallam University (SHU)
Sheffield, U.K.
lucas.bordignon@grad.ufsc.br, eduardo.camilo@posgrad.ufsc.br
M.Rodrigues@shu.ac.uk, mario.dantas@ufsc.br

Resumo. *O foco desse trabalho é estudar sistemas de arquivos paralelos, seus elementos e apresentar o Saturnus, um simulador em desenvolvimento para sistemas como esses. A proposta consiste em utilizar o mesmo para extrair dados e gerar informações, com foco no balanceamento de carga através do ambiente estudado, mostrando experimentos e possíveis usos para a ferramenta.*

1. Introdução

Sistemas de arquivos paralelos (SAPs) vem sendo adotados em diversas empresas e laboratórios de pesquisa ao redor do mundo como uma solução para o armazenamento de grande volume de dados. Sua principal vantagem é o acesso paralelo a várias partes de um mesmo arquivo em diversos nodos de armazenamento. Projetos como [Siddeq e Rodrigues 2016], onde o tempo gasto com o armazenamento, transmissão e, principalmente, processamento (compressão e descompressão) de dados é crucial e até alvo de pesquisas recentes, necessitam de diversas ferramentas que sirvam como base para obter resultados cada vez melhores. A utilização de sistemas de arquivos como esses tendem a auxiliar no desempenho dessas aplicações, principalmente em chamadas de escrita e leitura de arquivos.

Embora várias opções desses sistemas existam, como OrangeFS e Lustre, muitos projetos de pesquisa ainda tem como objetivo aprimorar ou criar novos SAPs. Entretanto, durante essa fase, utilizar, testar e prototipar são tarefas muito custosas em questão de dinheiro e tempo, o que leva muitos grupos de pesquisa a recorrerem ao uso de simuladores. Uma das principais dificuldades é compreender com clareza como cada elemento do sistema contribui para o desempenho das aplicações [Inacio et al. 2015].

Com foco nisso, este artigo apresenta o desenvolvimento de um simulador para SAP, produzido no Laboratório de Pesquisa em Sistemas Distribuídos (LAPESD) da UFSC e desenvolvido sobre o framework DESMO-J[Göbel et al. 2013], utilizando uma abordagem de simulação discreta e baseada em eventos. Com esse tipo de simulação, quando ocorrem requisições de execução de determinada tarefa, sinais de mudança de estado são enviados para o sistema, sendo esses chamados de eventos, eliminada a necessidade de executar o ambiente inteiro durante todo o período de simulação. Consequentemente, o software possui a vantagem de ser executado em máquinas mais modestas e distintas, com pouco poder de processamento, reduzindo assim os custos de seu uso.

2. Fundamentação Teórica

Embora vários trabalhos sobre simulação de sistemas de arquivos paralelos existam, como [Yonggang et al. 2013, Molina-Estolano et al. 2009], a idéia principal de cada um deles não é a mesma proposta nesse artigo. [Yonggang et al. 2013] tem uma abordagem voltada para algoritmos de agendamento de operações de entrada e saída, enquanto [Molina-Estolano et al. 2009] foca em aspectos de sincronismo de dados, como *locking*, localidade de dados e estratégias de replicação de dados.

[Settlemyer 2009] introduz o HECIOS, que modela de uma forma muito mais complexa os elementos de cache do sistema, o que reduz a flexibilidade do software para testes mais rápidos e com foco em outros aspectos, como o assunto principal desta pesquisa que, através do desenvolvimento do *Saturnus*, tem a finalidade de avaliar o balanceamento de carga em diferentes ambientes.

3. Saturnus

Visando reduzir a complexidade de uso do simulador, ele foi desenvolvido sobre uma abstração de um SAP, com algumas influências do sistema OrangeFS, abrangendo propriedades como particionamento de arquivo, comunicação cliente-servidor, entre outros. Além disso, até o presente momento, alguns elementos, como a camada de rede, não foram modelados ou implementados. O código fonte se encontra disponível na internet(https://github.com/lapesd/saturnus).

O funcionamento do *Saturnus* se baseia em 9 parâmetros de entrada: número de clientes, número de segmentos, número de servidores de dados, número de faixas, tamanho de bloco, tamanho de requisição, tamanho de faixa, padrão de acesso e tipo de arquivo. As sub-requisições, os eventos do sistema, são enviadas aos nodos de armazenamento com base em um algoritmo *round-robin*, de acordo com o número de faixas, sendo o primeiro nodo escolhido aleatoriamente em cada máquina cliente. Além disso, é utilizada uma distribuição normal com média baseada no tamanho de faixa e desvio padrão de 0.1% para computar seus devidos tempos de execução.

A Figura 1 mostra um exemplo onde máquinas clientes estão gerando requisições, que são divididas em sub-requisições, de acordo com os tamanhos de bloco, de requisição e de faixa, e enviadas ao sistema de arquivos. O seguinte conjunto de parâmetros foi utilizado: arquivo compartilhado, acesso sequencial, número de faixas = 3, número de clientes = 2, número de segmentos = 1, tamanho de bloco = 1024, tamanho de requisição = 512, tamanho de faixa = 256 e número de servidores = 5. Desse modo, é possível analisar quais são, por exemplo, os melhores tamanhos de bloco, de requisição e de faixa para determinadas situações.

4. Resultados Preliminares

A Figura 2 mostra o que ocorre com a variação do número de faixas em relação ao tempo necessário para executar as requisições, resultante da simulação. Ou seja, qual a influência de se adicionar mais servidores de dados ao sistema. O experimento apresenta uma média dos tempos de execução para cada número de faixa, onde cerca de 110 execuções foram registradas. O seguinte conjunto de parâmetros é utilizado: número de clientes = 3, número de segmentos = 5, tamanho de bloco = 2048, tamanho de requisição = 1024,

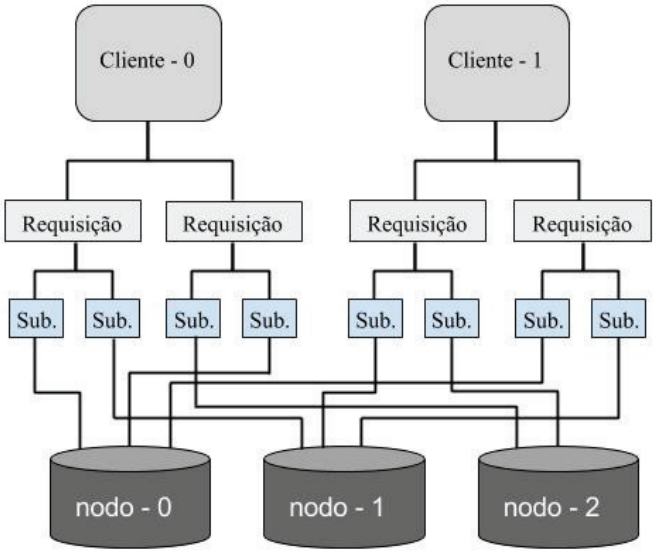

Figura 1. Exemplo básico de funcionamento do simulador.

tamanho de faixa = 512, tipo de arquivo = ”Compartilhado”, número de servidores = 11 e padrão de acesso = ”Sequencial”. Com essas informações, conseguimos perceber que, embora o número de servidores de dados recebendo requisições aumente, o tempo não reduz no mesmo ritmo, em virtude de sincronizações necessárias entre requisições, principalmente por causa do padrão de acesso sequencial utilizado.

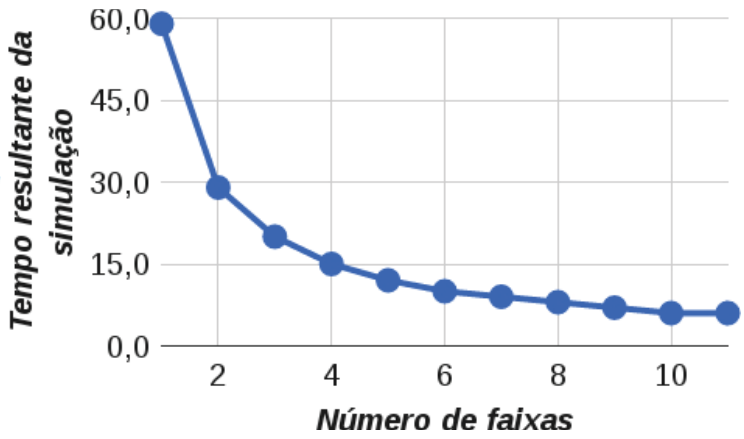

Figura 2. Evolução do tempo de execução conforme número de faixas.

A Figura 3 mostra o tamanho da fila de sub-requisições de cada servidor de dados a cada intervalo de uma unidade de tempo. Os parâmetros usados foram os mesmos do experimento anterior, porém com contagem de faixa fixada em 4. Para esses testes, o software selecionou aleatoriamente os nodos de armazenamento 0, 1, 5 e 7. É possível perceber que, embora tenhamos 4 servidores, muitas vezes, como há a necessidade de manter sincronia entre requisições, os eventos não são distribuídos de maneira uniforme.

5. Conclusão e Trabalhos Futuros

O presente trabalho trouxe uma breve introdução sobre o funcionamento do *Saturnus*, sua estrutura interna e alguns resultados já obtidos. Conforme informações extraídas dos experimentos, podemos perceber que, embora ainda seja um modelo simples, ele já consegue modelar o recebimento de requisições em servidores de armazenamento e

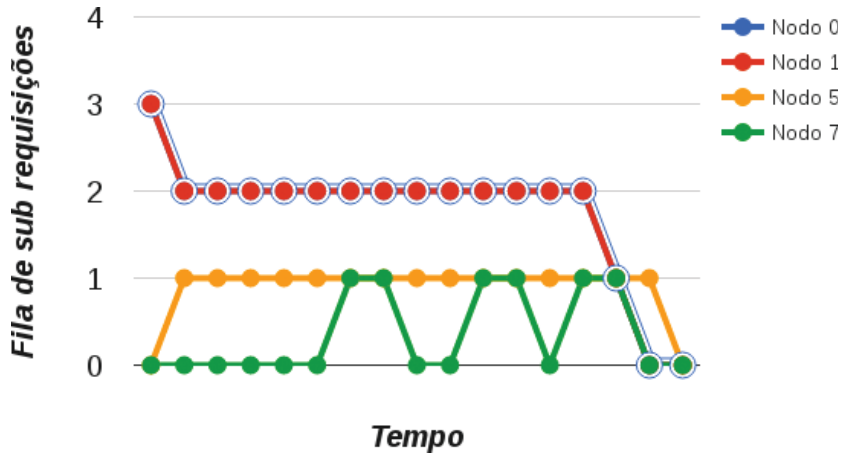

Figura 3. Fila de sub-requisições durante uma execução.

representá-las de maneira adequada. Além disso, mecanismos que mantém sincronia entre requisições, existentes em SAPs reais, também já estão presentes no mesmo.

Projetos que tenham como objetivo estudar ou utilizar SAPs de alguma maneira são beneficiados com o desenvolvimento deste software, já que o mesmo reduz o tempo gasto com testes em ambientes reais e permite aos usuários inferirem as melhores configurações para seus ambientes de uma maneira muito rápida. Para futuros trabalhos, a modelagem de outros elementos, como servidores de metadados e camadas mais complexas de rede, são os alvos. Além disso, a realização de mais testes para aprimorar a acurácia do modelo proposto também é um dos focos.

Referências

Göbel, J., Joschko, P., Koors, A., and Page, B. (2013). The discrete event simulation framework desmo-j: Review, comparision to other frameworks and latest development. *Proc. - 27th European Conf. on Modelling and Simulation, ECMS 2013*, pages 100–109.

Inacio, E. C., Pilla, L. L., and Dantas, M. A. R. (2015). Understanding the effect of multiple factors on a parallel file system's performance. *WETICE '15 Proc. 24th IEEE Intl. Conf. on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 90–92.

Molina-Estolano, E., Maltzahn, C., Bent, J., and Brandt, S. A. (2009). Building a parallel file system simulator. *J. Physics: Conf. Series*, 180(1), 1-7.

Settlemyer, B. W. (2009). A Study of Client-based Caching for Parallel I/O. *PhD Dissertation, Clemson University, Dep. Electrical Engineering*.

Siddeq, M. M. and Rodrigues, M. A. (2016). 3d point cloud data and triangle face compression by a novel geometry minimization algorithm and comparison with other 3d formats. *Proc. Intl. Conf. on Computational Methods, Berkeley, California*, pages 379–394.

Yonggang, L., Renato, F., Yiqi, X., and Ming, Z. (2013). On the design and implementation of a simulator for parallel file system research. *IEEE Symp. on Mass Storage Systems and Technologies*.