PIBITI/CNPq/UFCG-2012

AVALIAÇÃO DE ESCALABILIDADE E DESEMPENHO DE UM SISTEMA DE ARQUIVOS DISTRIBUÍDO OPORTUNISTA PARA REDES LOCAIS

João Pedro Ferreira de Melo Leôncio[1], Lívia Maria Rodrigues Sampaio Campos[2]

RESUMO

Questões de desempenho e escalabilidade são essenciais para sistemas distribuídos, em particular os sistemas de arquivos. Essas questões são consideradas, muitas vezes, como critério de comparação entre as diferentes soluções disponíveis no mercado. Uma avaliação detalhada permite aos desenvolvedores a identificação de gargalos e, melhoramento do sistema por conseguinte. O BeeFS é um sistema de arquivos distribuído oportunista caracterizado como uma solução barata e eficiente. Com o objetivo de investigar melhor a eficiência do sistema, foram realizados experimentos de medição em diferentes cenários, considerando versões do BeeFS para dois sistemas operacionais. Com o resultado desses experimentos foi possível identificar ao longo do projeto algumas falhas e ponderar o que representam os valores muito dísparos que estão presentes para certos cenários. Ao mesmo tempo, foi possível entender melhor o comportamento do sistema e as características que impactam no desempenho e escalabilidade para futuras melhorias.

Palavras-chave: desempenho, escalabilidade, avaliação quantitativa, sistema de arquivo distribuído.

SCALABILITY AND PERFORMANCE EVALUATION OF A DISTRIBUTED FILE SYSTEM FOR LOCAL AREA NETWORKS

ABSTRACT

Performance and scalability issues are essential for distributed systems, in particular file systems. These issues are often considered as a comparison criterion between the different solutions available on the market. A detailed evaluation allows the developers to identify bottlenecks and thus, make improvements. The BeeFS is a distributed file system that harness idle free space and is characterized as a cheap and efficient solution. In order to investigate more the efficiency of the system, experiments were performed measuring into different scenarios, considering versions of the BeeFS for two operating systems. With the results of these experiments, it was possible to identify throughout the project some failures which represent the outliers that are present for certain scenarios. At the same time, it was possible to understand better the behavior of the system and the characteristics that impact the performance and scalability for future improvements.

Keywords: performance, scalability, quantitative evaluation, file system.

[1]Aluno do Curso de Ciência da Computação, Departamento de Sistemas e Computação, UFCG, Campina Grande, PB, e-mail: joao.leoncio@ccc.ufcg.edu.br
[2]Ciência da Computação, Professora Doutora, Departamento de Sistemas e Computação, UFCG, Campina Grande, PB, e-mail: livia@computacao.ufcg.edu.br

INTRODUÇÃO

Sistemas de arquivos distribuídos são comumente utilizados em empresas e ambientes acadêmicos para compartilhamento de arquivos entre usuários espalhados em uma rede de computadores onde estes arquivos estão armazenados. Os sistemas de arquivos podem ter diversas arquiteturas e um exemplo muito popular é o NFS (Network File System) (PAWLOWSKI et al, 1994) que, por sua vez, utiliza a arquitetura cliente-servidor. Essa arquitetura pode muitas vezes ser a razão para problemas muito comuns, como observado na literatura, de escalabilidade e disponibilidade de um sistema de arquivos desse tipo. A escalabilidade diz respeito à capacidade de um sistema suportar um crescimento e disponibilidade se refere à capacidade de oferecer o serviço esperado em um determinado instante de tempo. A abordagem cliente-servidor é constituída de um único servidor para o qual todos dos clientes vão ter que fazer suas requisições todas as vezes que for preciso acessar um arquivo. O problema dessa abordagem é que o servidor pode ficar sobrecarregado em termos de processamento e/ou armazenamento, a depender do número de usuários do sistema. A solução para problemas nesse sentido se mostra bastante custosa podendo ser necessária a aquisição de mais discos e/ou memória, fato que torna o sistema menos eficiente e mais difícil de gerenciar. Ainda nesse sentido, há casos de implantação de mais servidores para balancear a carga sobre um único servidor, mas isto também é custoso e causa dificuldade de gerenciamento. (EDWARD et al, 1991) (SATYANARAYANAN et al, 1990)

O BeeFS (PEREIRA, 2010) é sistema de arquivos distribuído para redes locais que segue as normas POSIX (POSIX, 2008) e tem um modelo de distribuição de dados híbrido. Nesse modelo, há um servidor centralizado apenas para o armazenamento, requisição de metadados e servidores distribuídos para o armazenamento dos arquivos. Outra característica do BeeFS é ser oportunista, isso implica na utilização dos recursos ociosos (disco) ou não dedicados do sistema, evitando o desperdício de recursos (POSIX, 2008) (SUDHARSHAN et al, 2005) (TOLIA, 2003). Tal característica torna a solução BeeFS eficiente, barata e escalável. Note que, a capacidade dos HDs modernos têm aumentado a uma proporção muito maior do que a necessidade de vários usuários, gerando espaço de armazenamento ocioso nas máquinas destes usuários. Por exemplo, seja um laboratório de pesquisa de uma universidade com 50 postos de trabalho que usa o NFS como sistema de arquivos. Nesse caso, os arquivos dos vários usuários associados a este laboratório estão centralizados em um servidor dedicado, ao mesmo tempo, o espaço que deveria estar sendo ocupado por tais arquivos nas 50 máquinas disponíveis neste laboratório fica ocioso e, por conseguinte, subutilizado.

A arquitetura do BeeFS (ver Figura 1) é chamada híbrida porque mescla características de sistemas P2P (par-a-par) e cliente-servidor. A ideia é separar metadados (servidor central) e dados (servidores distribuídos; pares), facilitando a administração do sistema. A distribuição dos dados em diferentes máquinas confere ao sistema uma diminuição na sobrecarga sobre o servidor central, além de promover o crescimento incremental da capacidade de armazenamento do sistema.

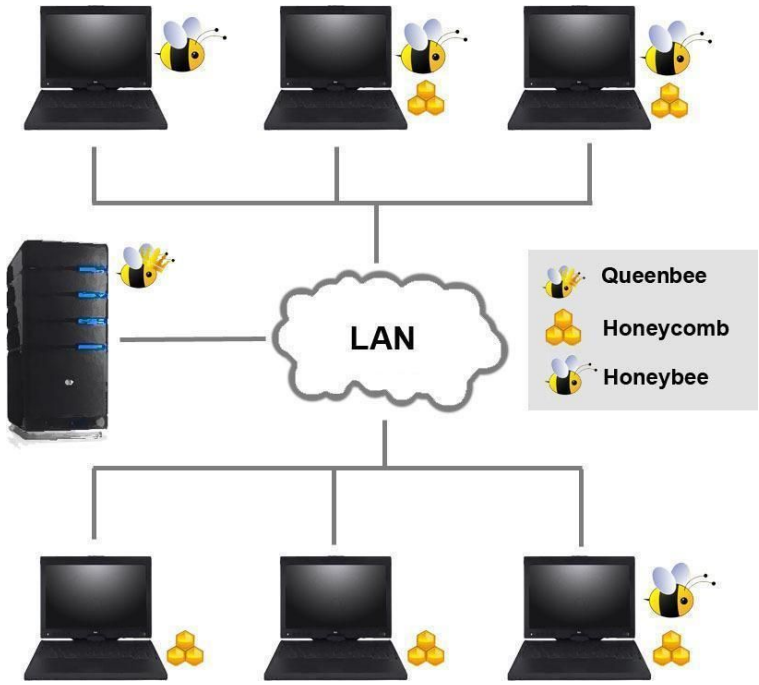

Figura 1: Arquitetura do BeeFS

O BeeFS está sendo desenvolvido no Laboratório de Sistemas Distribuídos, Departamento de Sistemas e Computação do Campus I da Universidade Federal de Campina Grande (LSD/DSC/UFCG-Campus I) com resultados positivos e perspectivas de trabalhos futuros (BRASILEIRO et al, 2011)(SOUZA et al, 2010)(BRASILEIRO et al, 2009)(PEREIRA, 2010)(SILVA, 2010)(CAMPOS et al, 2012). O Laboratório conta com um rede local com dezenas de computadores conectados, suporte técnico dedicado, biblioteca com vasto acervo da área e um auditório onde ocorrem defesas de projetos e palestras. Também com equipe de desenvolvimento constituída atualmente de 8 integrantes que se reúnem semanalmente para definir prazos e acompanhar as atividades de projeto.

Experimentos anteriores realizados no BeeFS demonstram ser uma solução eficiente, barata e escalável, porém o sistema vem evoluindo desde então o que requer a realização de novos experimentos. Além disso, aspectos de escalabilidade não foram considerados nos estudos já realizados. O projeto descrito neste relatório consistiu em um estudo mais detalhado sobre a eficiência (em termos de desempenho) do BeeFS e alguns resultados sobre aspectos de escalabilidade do sistema.

REVISÃO BIBLIOGRÁFICA

A revisão bibliográfica foi definida como primeira fase do trabalho em conjunto com um treinamento no BeeFS que é foco do estudo de desempenho e escalabilidade sendo realizado. O objetivo era se familiarizar com sistemas de arquivos distribuidos em geral e fazer um levantamento dos principais mecanismos para a avaliação de desempenho em sistemas computacionais.

Para o estudo voltado à familiarização com sistemas de arquivos distribuídos foi feito a leitura do livro TANENBAUM (2007), especificamente os capitulos introdutórios sobre sistemas de arquivos distribuídos, sobre a sincronização e conceitos de design desses sistemas.

Para o levantamento dos principais mecanismos para avaliação de desemepenho foi feita leitura do artigo de TRAEGER (2008) que faz considerações sobre os principais mecanismos de benchmarking no mercado. Ainda para reforçar o estudo sobre avaliação de desempenho, foi necessário a leitura do livro (JAIN, 1991) que permitiu conhecer os principais termos e considerações a serem levadas em conta em uma análise de desempenho, quanto a técnicas e ferramentas. Um conhecimento complementar sobre análise de dados e a linguagem R foi adquirido no decorrer do projeto a partir das aulas do curso “Statistics: Making Sense of Data” da University of Toronto, no site Coursera[3].

[3] www.coursera.org

MATERIAIS E MÉTODOS

Este projeto segue uma metodologia experimental para avaliação quantitativa do sistema BeeFS em termos de desempenho e escalabilidade. O objetivo foi avaliar o BeeFS em cenários reais, ou próximos do real, através de experimentos de medição em ambiente de produção. A execução destes experimentos foi precedida de uma fase de planejamento onde foram definidos com cuidado os cenários, métricas e parâmetros de configuração a fim de que o processo de avaliação estivesse bem fundamentado. Esse planejamento requer o estudo de trabalhos relacionados e embasamento teórico sobre avaliação de desempenho e escalabilidade baseados em medição, como também, entender o funcionamento do sistema objeto de estudo; tal embasamento consistiu em atividade inicial do projeto. Após a execução dos experimentos foi conduzida a análise dos dados a partir de métodos estatísticos bem conhecidos. Seguindo essa metodologia, a realização desse projeto compreendeu as seguintes etapas:

1. Embasamento teórico.
2. Treinamento no BeeFS para conhecer, particularmente, seu funcionamento.
3. Planejamento dos experimentos para o estudo do sistema.
4. Execução dos experimentos.
5. Análise dos resultados dos experimentos.
6. Planejamento e implementação de melhorias no BeeFS que impactem em desempenho e escalabilidade.
7. Documentação e divulgação dos resultados

É importante salientar que as etapas 3 a 6 ocorreram em ciclos pois os processos de avaliação de desempenho e escalabilidade envolveram refinamentos.

RESULTADOS E DISCUSSÕES

Para todos os experimentos realizados neste projeto, foram escolhidas cargas baseadas em experimentos anteriores, deste modo foi possível definir o impacto das alterações no BeeFS ao longo de sua evolução.

Ambiente de Execução

INFRAESTRUTURA. Os experimentos de avaliação de desempenho foram conduzidos em quatro das máquinas da rede local do Laboratório de Sistemas Distribuidos (LSD), tendo por nomes **tubarao[4], abelhinha[5], mulato[6] e gupi[7]** e com as configurações descritas na Tabela 1.

Máquina	**Processador**	**Memória**	**Disco**	**Sistema operacional**
mulato	Intel® Pentium® 4 CPU @ 3.00GHz	2GB	250GB	Ubuntu 10.04 LTS
abelhinha	Intel® Core™ 2 Duo CPU E6550 @ 2.33GHz	2GB	160GB	Ubuntu 10.04 LTS
tubarao	Intel® Core™ 2 Duo CPU E6550 @ 2.33GHz	2GB	160GB	Ubuntu 12.04 LTS
gupi	Intel® Pentium® 4 CPU @ 3.00GHz	2GB	80GB	Windows XP Professional SP2

[4] tubarao.lsd.ufcg.edu.br
[5] abelhinha.lsd.ufcg.edu.br
[6] mulato.lsd.ufcg.edu.br
[7] gupi.lsd.ufcg.edu.br

Tabela 1: configuração das máquinas usadas nos experimentos de avaliação de desempenho no BeeFS

CARGA DE TRABALHO. As cargas de trabalho descrevem o conteúdo do sistema de arquivos, ou seja, a quantidade e tamanho dos arquivos. Nesse caso, tem-se cargas de trabalho altas e baixas. As cargas baixas foram: Carga 1, que consiste em 100 arquivos de 103 Kilobytes, totalizando 100 Megabytes; a Carga 2 com 3739 arquivos de 103 Kilobytes, totalizando 378 Megabytes; e a Carga 4 com 7900 arquivos de 49 Kilobytes, totalizando 378 Megabytes. A carga alta, denominada de Carga 3, engloba 1 arquivo de 2 Gigabytes.

APLICAÇÃO. Os experimentos consistiram em realizar operações sobre o sistema de arquivos e observar algumas métricas. Para tal, foi utilizado um micro-benchmark desenvolvido pela equipe do BeeFS que realiza operações de escrita no ponto de montagem do sistema.

CENÁRIOS. O BeeFS foi configurado para dois modos de execução diferentes: sincronização e composição. A sincronização determina a maneira com que o Honeycomb irá trabalhar sobre a escrita de arquivos, podendo ser de dois tipos: síncrono (sync) ou assíncrono (async). A composição diz respeito ao uso da JVM na execução do sistema. Ocorre composição quando o Honeybee e o Honeycomb são executados na mesma JVM, pelo componente Combee; por outro lado, se o Honeybee e Honeycomb são executados em JVMs diferentes não ocorre composição. Esses dois modos de configuração criaram 4 cenários diferentes para cada uma das 4 cargas de trabalho, totalizando 16 cenários diferentes.

Experimentos de Desempenho

Os experimentos de desempenho consistiram em comparar o BeeFS com o NFS a fim de entender melhor o desempenho do BeeFS em determinados cenários. No caso, a métrica de desempenho foi o tempo de execução do experimento (makespan). A execução dos experimentos permitiu fazermos a comparação direta para todos os cenários do BeeFS com o NFS, assim como comparar as versões do BeeFS no Linux e no Windows. Utilizou-se sempre como referencial o NFS por ser um sistema de arquivos muito popular. A seguir serão detalhados os resultados dos experimentos realizados.

Para realizar os experimentos de desempenho nos 16 cenários já citados, o micro-benchmark utilizado pela equipe foi melhorado no sentido de que as operações de escrita na partição montada pelo BeeFS e a captura do tempo decorrido por cada uma das operações fosse salva em um arquivo contendo todas as informações sobre o teste específico (logging). Antes e depois de cada operação, o micro-benchmark realiza a limpeza do ambiente de execução, eliminando os metadados e dados envolvidos para permitir que vários experimentos sejam feitos em sequência sem intervenção manual.

Ainda para tornar os experimentos mais fáceis de serem conduzidos e repetidos no futuro foi gerada uma documentação detalhada para descrição do ambiente de teste.

A primeira rodada de experimentos foi realizada apenas na plataforma Linux e foram recolhidas 10 amostras de cada um dos 16 cenários de execução. O Queenbee foi executado na máquina **abelhinha** e os outros componentes na máquina **mulato**. Esses experimentos permitiram a detecção de uma falha na sincronização do BeeFS que gerou resultados equivalentes para a sincronização ligada e desligada. Esse resultado implicou na procura pelo que havia ocorrido no BeeFS. A busca se deu exatamente no ponto do código fonte onde é verificado, a partir da leitura do arquivo de configuração, o modo pelo qual o BeeFS deve funcionar. Foi, então, que se percebeu o fato do BeeFS estar gerando resultados apenas para o modo de sincronização desligada. Identificada a falha e corrigida, os experimentos foram reexecutados para que se pudesse verificar os valores reais de execução do BeeFS de modo síncrono. Essa segunda rodada de experimentos foi realizada apenas na plataforma Linux, seguindo as configuração já descritas e foram recolhidas 10 amostras de cada um dos cenários de execução. Os resultados são ilustrados nos Gráficos 1, 2, 3 e 4. Esses gráficos estão em escala logarítmica porque os valores de tempo de execução (makespan) apresentaram disparidades grandes em alguns cenários, ficando impossível o entendimento do gráfico em escala normal.

Os Gráficos 1 e 2, em forma de boxplot[8], descrevem os resultados dos experimentos para os modos assíncrono e síncrono, respectivamente. considerando o impacto da composição, ou seja, o fato dos componentes do BeeFS estarem ou não executando na mesma JVM, com um nível de confiança de 95%.

[8] Boxplot: é uma forma robusta de demonstrar resultados bastante utilizada, descrevendo os resultados em quartis.

Esses gráficos demonstram um melhoramento no sistema BeeFS, quando se tem a união dos componentes Honeybee e Honeycomb em um só componente, denominado de Combee. O resultado mostra-se ainda mais acentuado na Carga 3 para ambos modos de sincronização, demonstrando que com cargas mais altas o impacto de usar o Combee é ainda maior em termos de desempenho. O fato de ser uma carga alta indica que a comunicação entre Honeybee e Honeycomb será mais intensa e, desse modo, para a utilização do combee que faz com que a comunicação entre Honeybee e Honeycomb, estando em mesma máquina, seja mais curta, o resultado, como esperado, é melhor.

O Gráfico 3 descreve os resultados considerando o impacto da sincronização utilizando o componente combee, em forma de boxplot. Esse gráfico demonstra que a alteração no modo de sincronização do BeeFS é impactante no sistema operacional Linux. Para tentar identificar a possível fonte do problema foram realizados experimentos simples de escrita síncrona e assíncrona nas funções utilizadas para escrita desses modos na linguagem Java, utilizada pelo BeeFS. O resultados apontam que há realmente uma diferença para os modos de sincronização advinda do Java.

O Gráfico 4 descreve os melhores resultados do BeeFS gerados pelo uso do Combee, comparados com resultados do NFS nos mesmos cenários. Os resultados se apresentam em pontos negativos do gráfico devido a utilização da escala logarítmica[9]. Nesse caso, os experimentos no NFS foram realizados também na plataforma Linux e foram coletadas 10 amostras de cada um dos 16 cenários de execução considerados. O servidor NFS foi executado na máquina **tubarao** e o cliente na máquina **mulato.**Os resultados demonstram que o BeeFS apresenta um desempenho inferior ao NFS tanto para os cenarios síncrono quanto assíncrono. Observou-se também que a diferença entre os modos assíncrono e síncrono do NFS são menores, possivelmente, pelo fato do NFS ser escrito na linguagem C que é mais eficiente do que Java em operações sobre arquivos.

[9] O ln(0) tende ao infinito negativo, enquanto o ln(1) é 0. As médias dos resultados estão entre 0 e 1.

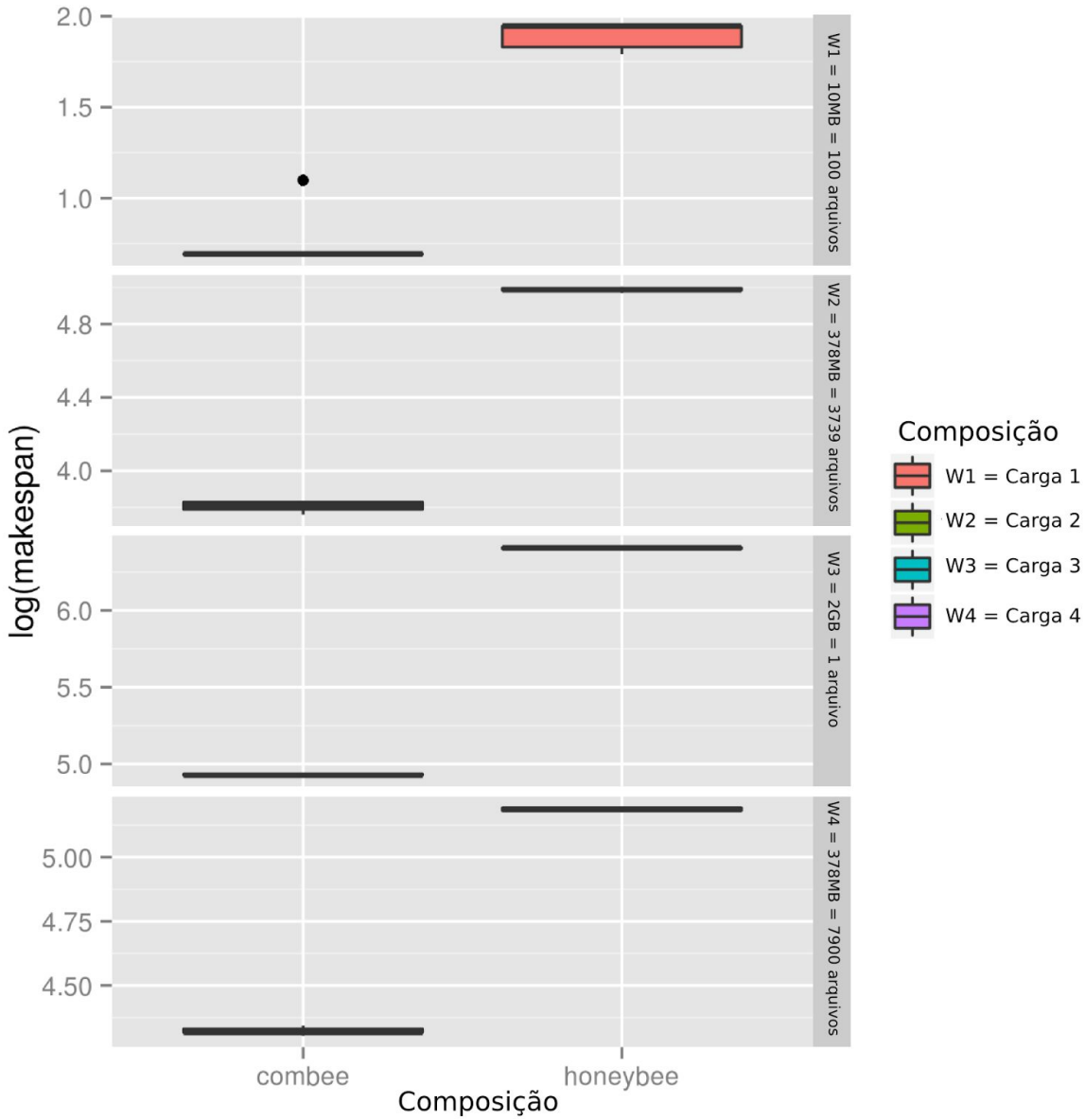

Gráfico 1: Boxplot das operações de escrita realizadas com BeeFS configurado para maneira assíncrona (async) no sistema operacional Linux. No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

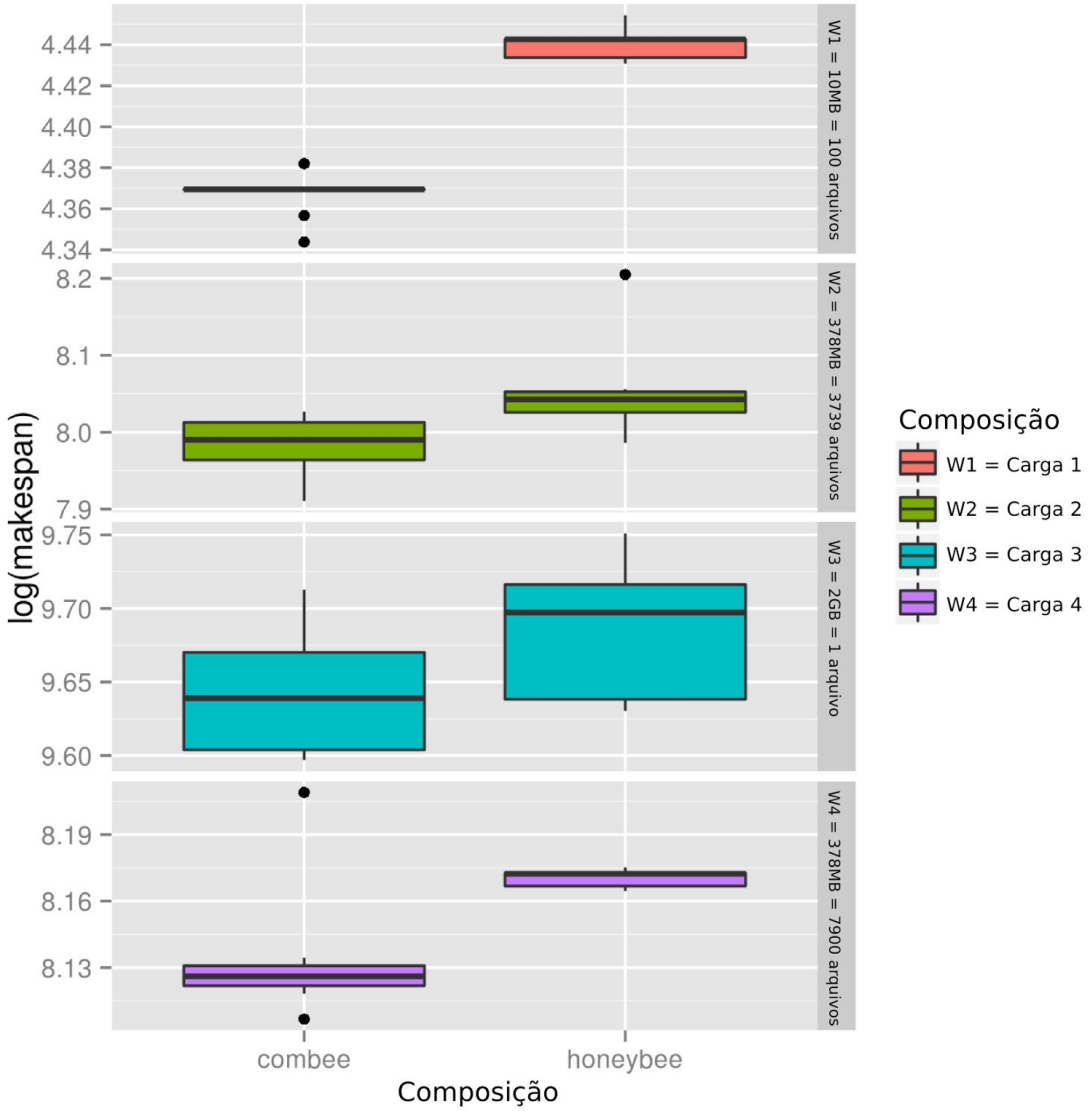

Gráfico 2: Boxplot das operações de escrita realizadas com BeeFS configurado para maneira síncrona (sync) no sistema operacional Linux. No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

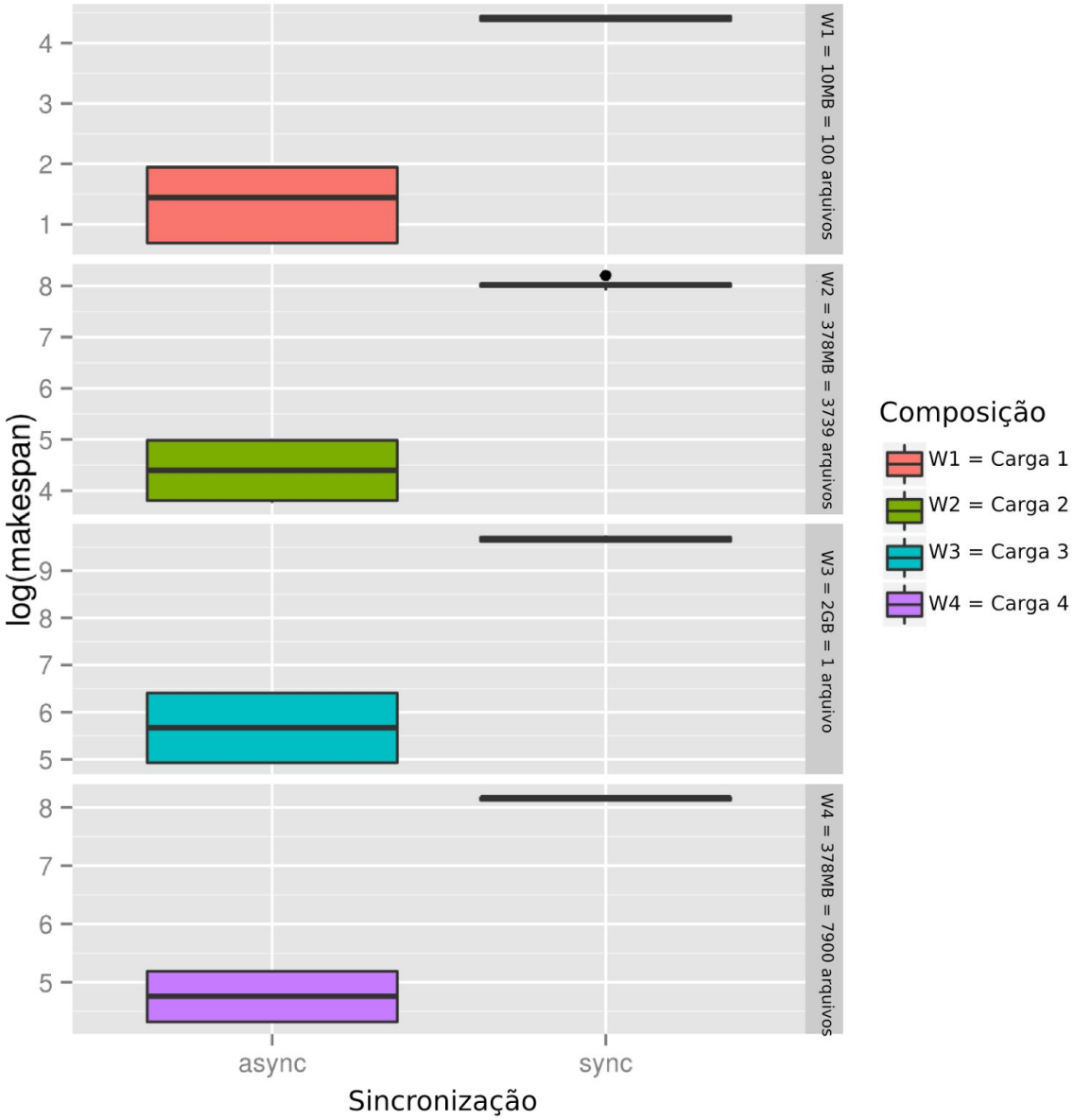

Gráfico 3: Desempenho do BeeFS em operações de escrita realizadas com os componentes do BeeFS de maneira síncrona ou assíncrona no sistema operacional Linux No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

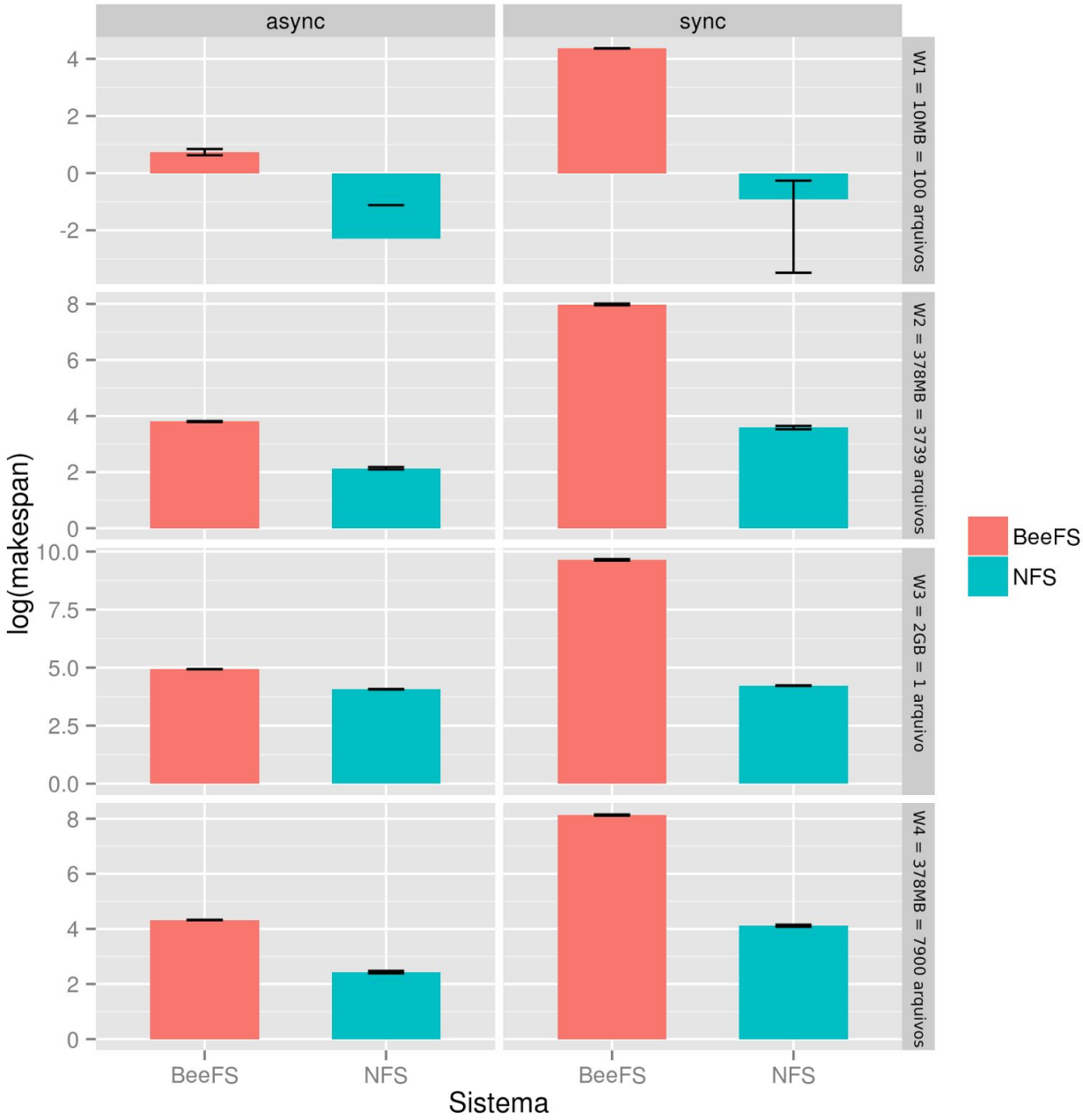

Gráfico 4: Desempenho do BeeFS em operações de escrita realizadas com os componentes do BeeFS em uma mesma JVM e o NFS de maneira síncrona ou assíncrona. No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

Como parte do estudo do comportamento do BeeFS, foi necessário avaliar também o desempenho no outro sistema operacional para o qual o BeeFS oferece suporte, o Windows®. Para os experimentos no Windows foram coletadas 10 amostras de cada um dos cenários de execução. O Queenbee foi executado na máquina **tubarao** e o Honeybee e o Honeycomb, ou combee, na máquina **gupi**. Os Gráficos 5 e 6 demonstram os resultados de composição, para o modo síncrono e modo assíncrono, respectivamente. Esses resultados demonstram um melhoramento do sistema BeeFS também presente no sistema operacional Windows quando utiliza-se o componente Combee. Esse resultado é mais ressaltado na **Carga 3** em ambos os gráficos. O fato de ser uma carga alta indica que a comunicação entre Honeybee e Honeycomb será mais intensa e, desse modo, para a utilização do combee que faz com que a comunicação entre Honeybee e Honeycomb, estando em mesma máquina, seja mais curta, o resultado, como esperado, é melhor. Por último, o Gráfico 7 indica uma grande semelhança na execução do BeeFS no Windows configurado para modo de sincronização ligado ou desligado. O que pode indicar uma falha no BeeFS ou que a escrita de Java no

sistema operacional Windows acontece mais rapidamente que no Linux.

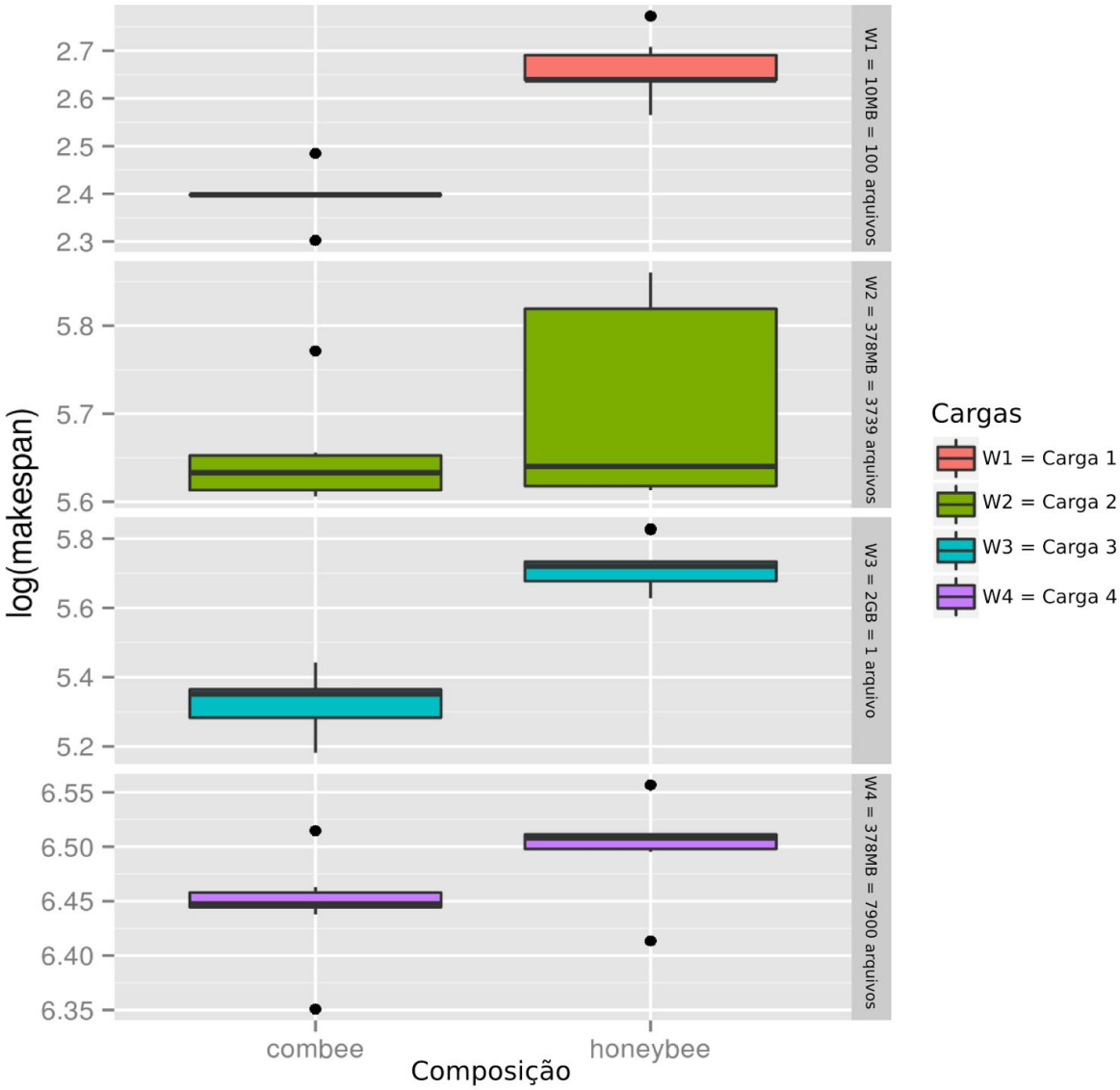

Gráfico 5: Boxplot das operações de escrita realizadas com BeeFS configurado para maneira assíncrona (async) no sistema operacional Windows. No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

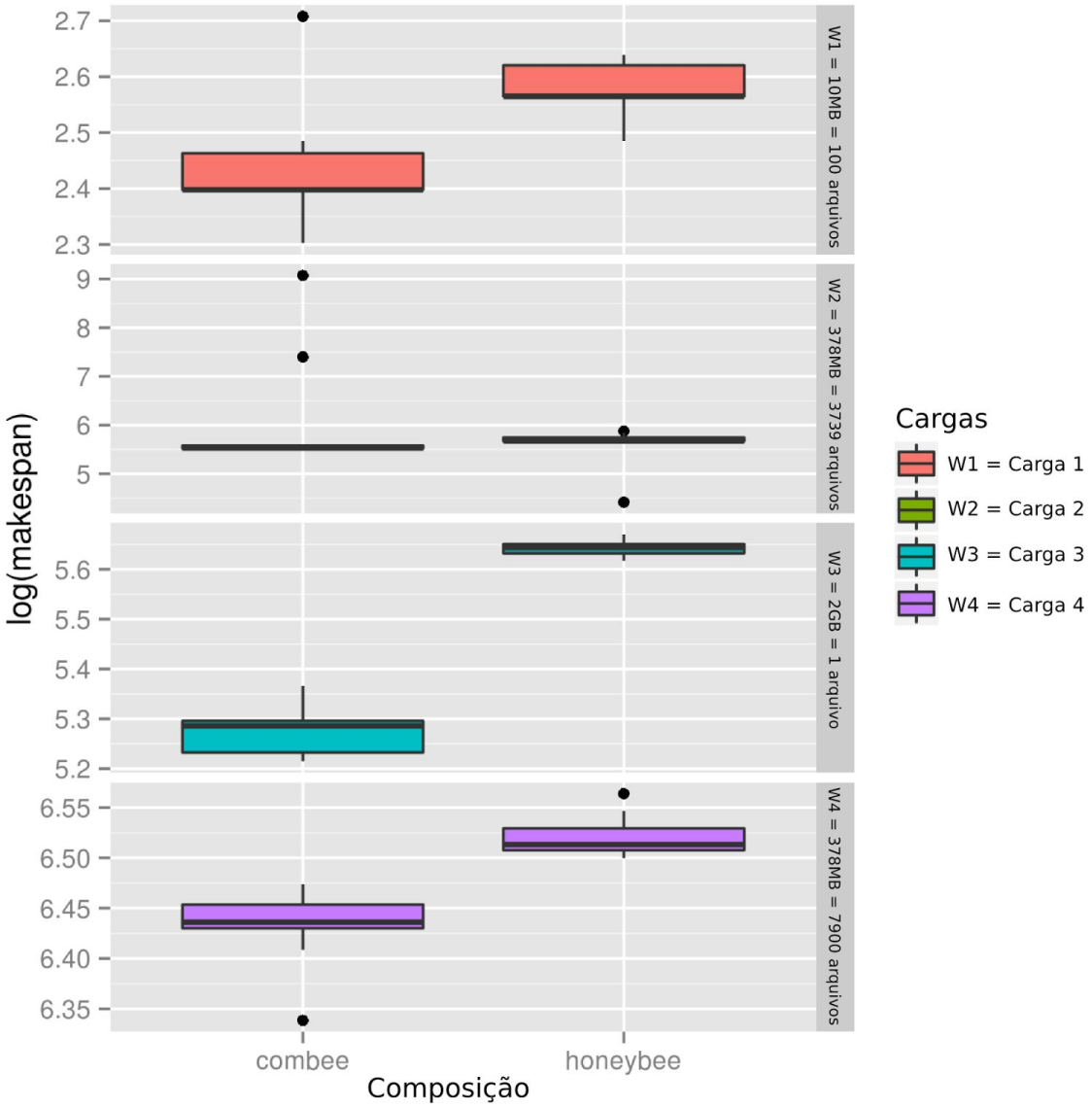

Gráfico 6: Boxplot das operações de escrita realizadas com BeeFS configurado para maneira síncrona (sync) no sistema operacional Windows. No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

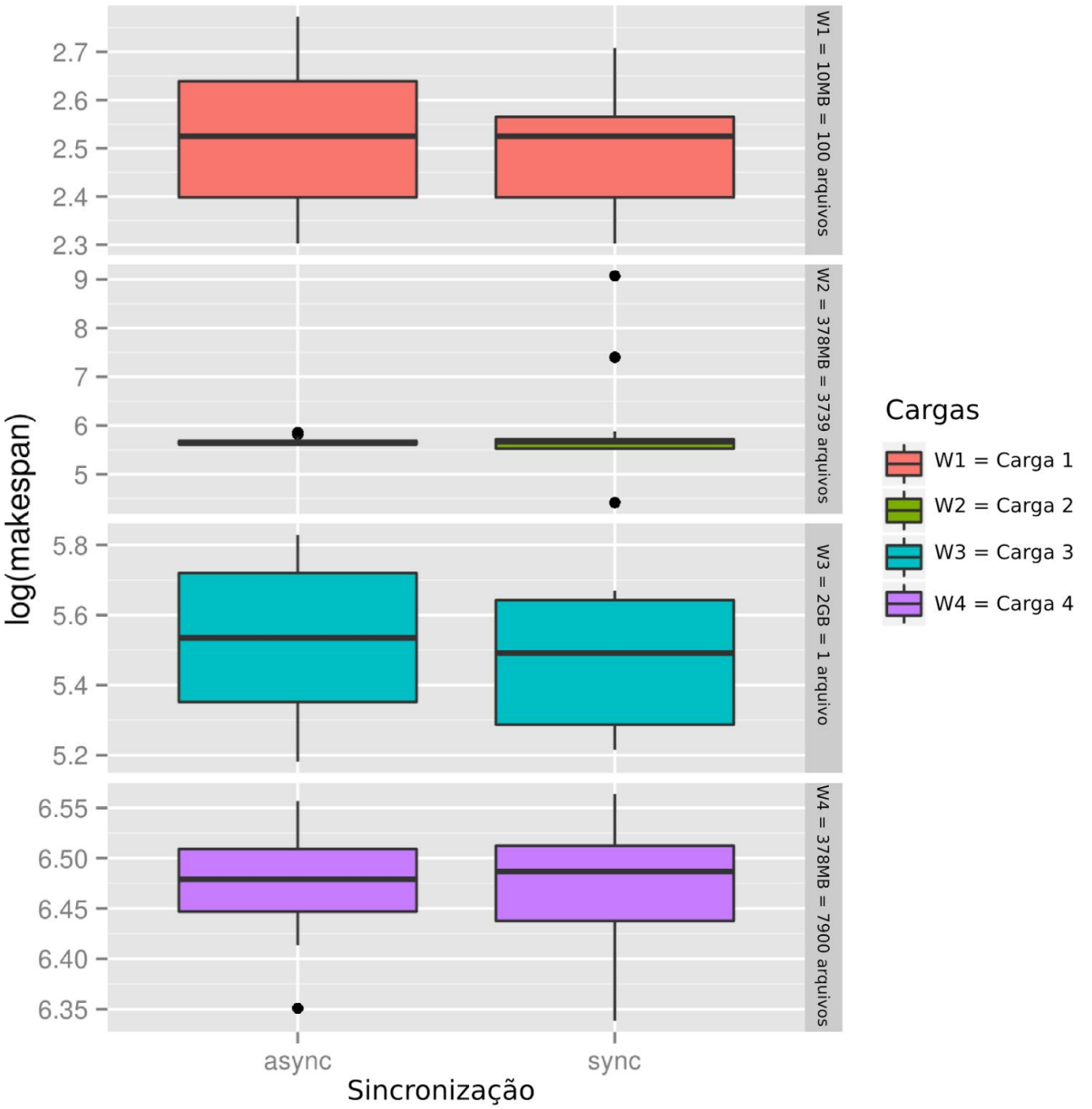

Gráfico 7: Desempenho do BeeFS em operações de escrita realizadas com os componentes do BeeFS de maneira síncrona ou assíncrona no sistema operacional Windows. No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

Comparando o desempenho do BeeFS nos 16 cenários de execução considerados e nos dois sistemas operacionais Linux e Windows, é possível perceber que o BeeFS no Windows, embora ainda tenha um desempenho pior (no modo assíncrono), tende a se aproximar do desempenho no Linux quando reduzida a quantidade de arquivos envolvidos (ver Gráfico 8). A quantidade de arquivos implica diretamente na quantidade de requisições feita pelo Honeybee ao queenbee por metadados. Por outro lado, os resultados no Windows foram melhores para o modo síncrono.

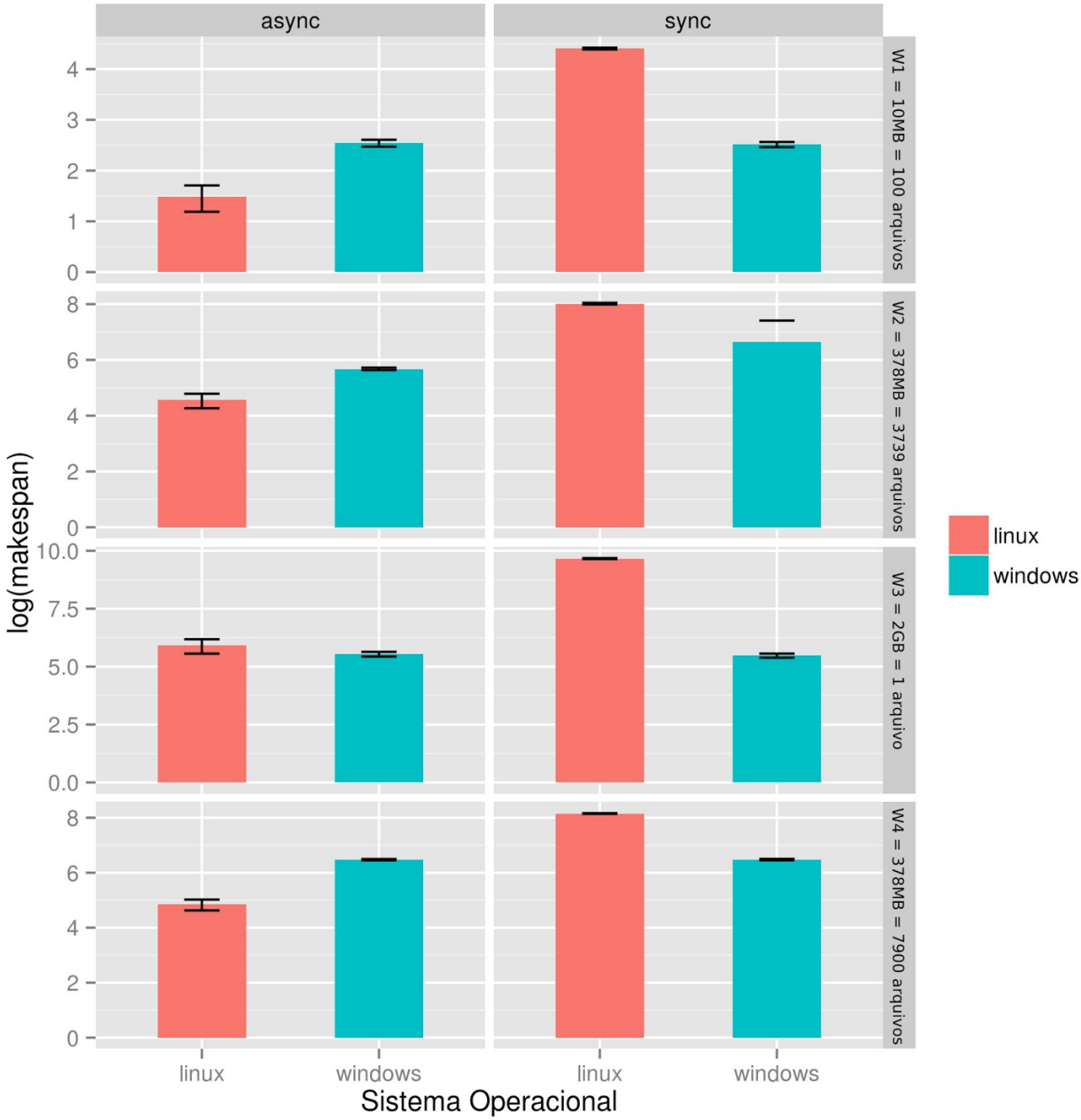

Gráfico 8: Desempenho do BeeFS em operações de escrita realizadas com os componentes do BeeFS em uma mesma JVM e de maneira síncrona ou assíncrona no sistema operacional Windows e Linux. No eixo y, o tempo decorrido de execução (em segundos) separado pelas cargas de trabalho.

Experimentos de escalabilidade

Já os experimentos de escalabilidade foram realizados usando uma infraestrutura virtualizada disponibilizada pela cloud privada do lsd, denominada Jit-Cloud-LSD. A partir dessa cloud privada é possível criar máquinas virtuais (VMs) à medida que se precisa sem se preocupar com o gerenciamento físico dessas VMs. Desse modo, temos garantia de que mesmo estando na mesma máquina, as VMs são independentes e não afetam o trabalho uma da outra e, por isso, podemos considerá-las como ambiente ideal para experimentos de escrita concorrente, como é o nosso caso. Além disso, tem-se a possibilidade de ter um ambiente com maior quantidade de máquinas, como exigido para avaliar a escalabilidade do BeeFS.

Os experimentos de escalabilidade objetivaram caracterizar o comportamento do BeeFS mediante o aumento na escala do sistema de arquivos, isto é, aumentando a quantidade de arquivos para um mesmo cenário, depois verificar também o comportamento para o aumento no número de clientes. Nesse caso, o Queenbee será submetido a uma carga maior de trabalho e deseja-se observar o desempenho do sistema (qual o tempo de execução dos experimentos) no cenário indicado.

Nesse caso, a carga de trabalho utilizado foi a Carga 1, descrita anteriormente, com duas variações: carga 1A de 10MB com 100 arquivos de 100K cada, e carga 1B de 10MB com 1 único arquivo e carga 1C de 10MB e 50 arquivos de 200K cada. O ambiente foi preparado usando uma ferramenta desenvolvida pela equipe do projeto BeeFS para a escrita concorrente, i.e., quando existem múltiplos clientes escrevendo no sistema de arquivos . A ferramenta garante que todas as máquinas virtuais têm seus relógios sincronizados para que a escrita seja realmente concorrente e que todos os clientes terminem em tempo semelhante. O experimento foi desenvolvido em niveis de concorrência o que significa o número de clientes simultâneos sobre o sistema de arquivos, então, nível de concorrência 4 significa 4 clientes no sistema. Inicialmente, foi considerada uma execução sem concorrência, para que fosse possível verificar o desempenho geral do BeeFS no ambiente virtualizado. Em seguida, considerou-se os níveis de concorrência 4, 6, 7, 8, 9 e 10. Desse modo, foi possível observar que o desempenho do BeeFS diminui à medida que o nível de concorrência cresce, considerando a carga 1 onde tem-se uma maior quantidade de arquivos (ver Gráficos 9 e 11). Isso ocorre por dois fatores: o aumento na quantidade de arquivos faz com que junto com a quantidade de clientes, mais requisições de criação de arquivos sejam feitas ao Queenbee, desse modo, sobrecarregando o queenbee, que enquanto responde a uma quantidade de requisições, deixa outras a espera.Por outro lado, no cenário de carga com 1 único arquivo tem-se menos requisições de criação de arquivos à quais são concentradas no início do experimento e não impactam tanto no tempo de execução do experimento (ver Gráfico 10). Nesse caso, o Queenbee só é requisitado na criação do arquivo, quando retorna informação sobre a máquina que guardará tal arquivo. A versão do BeeFS utilizada para os experimentos de escalabilidade foi a de melhores resultados nos experimentos de desempenho, isto é, versão em que os componentes Honeybee e Honeycomb estão co-alocados, chamando-se Combee.

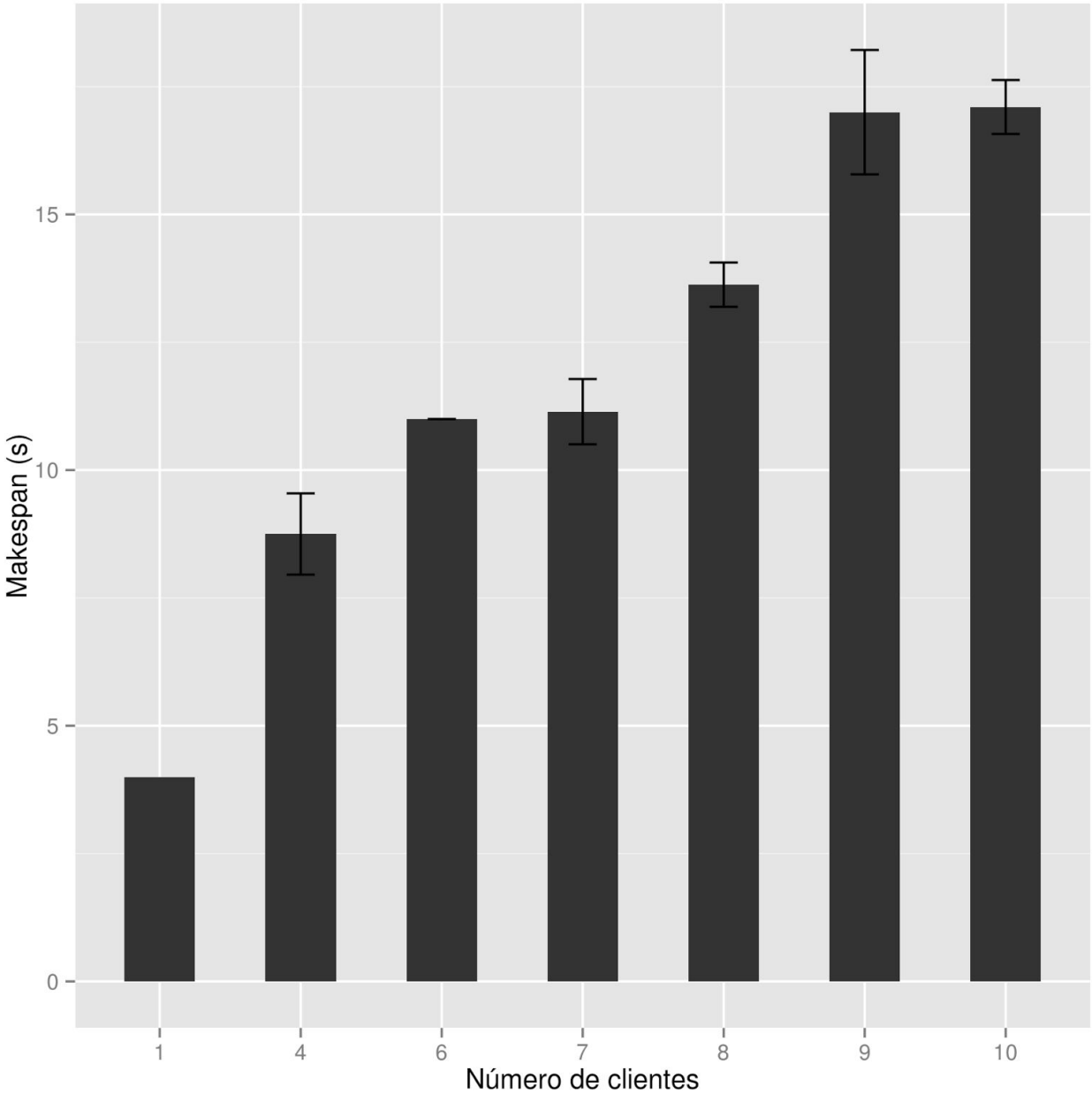

Gráfico 9: Desempenho do BeeFS em operações de escrita com um workload de 10Mb e 100 arquivos, realizadas com os componentes do BeeFS em uma mesma JVM aumentando ao passo que se aumenta a concorrência. No eixo y, o tempo decorrido de execução (em segundos).

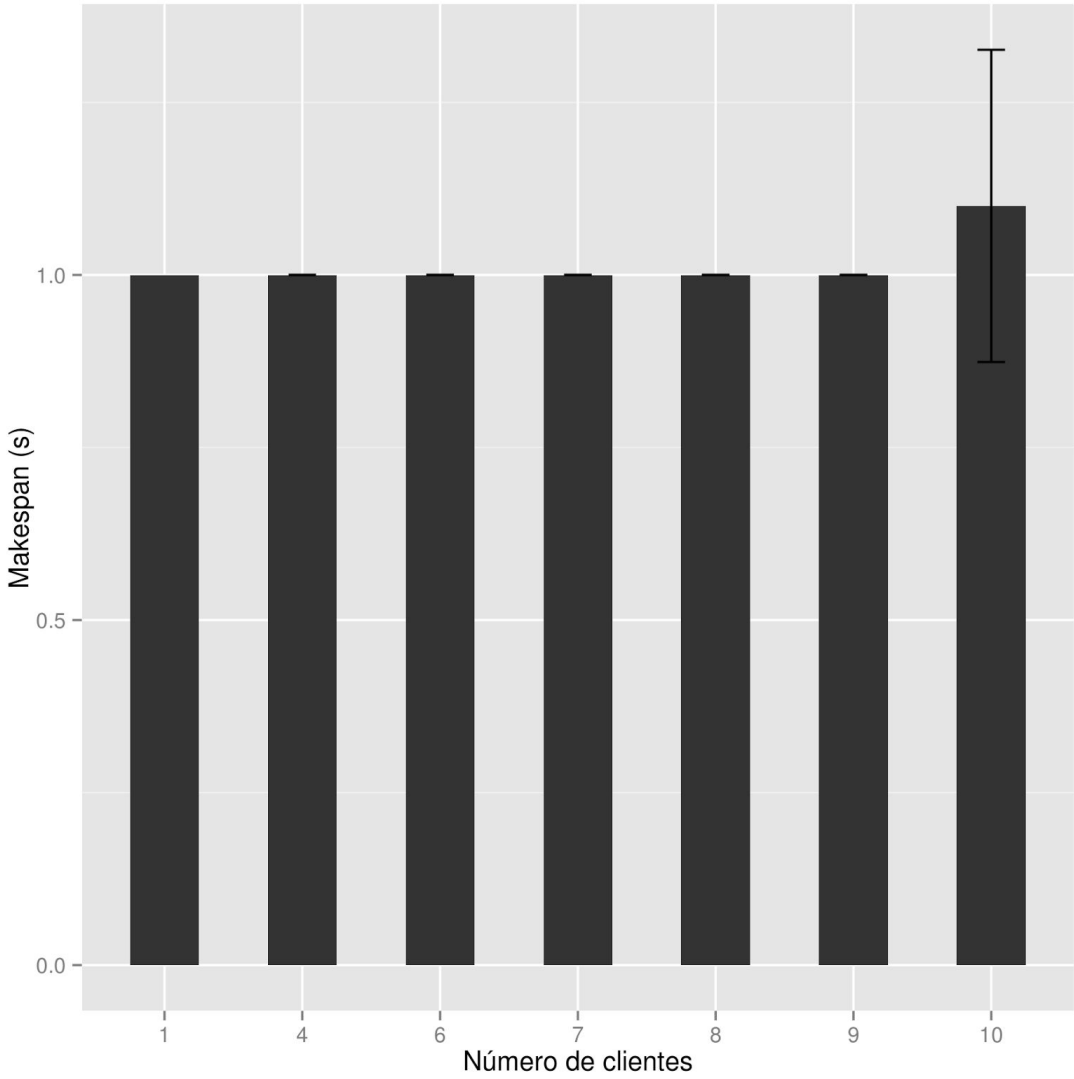

Gráfico 10: Desempenho do BeeFS em operações de escrita com um workload de 10Mb em 1 único arquivo, realizadas com os componentes do BeeFS em uma mesma JVM aumentando ao passo que se aumenta a concorrência. No eixo y, o tempo decorrido de execução (em segundos)

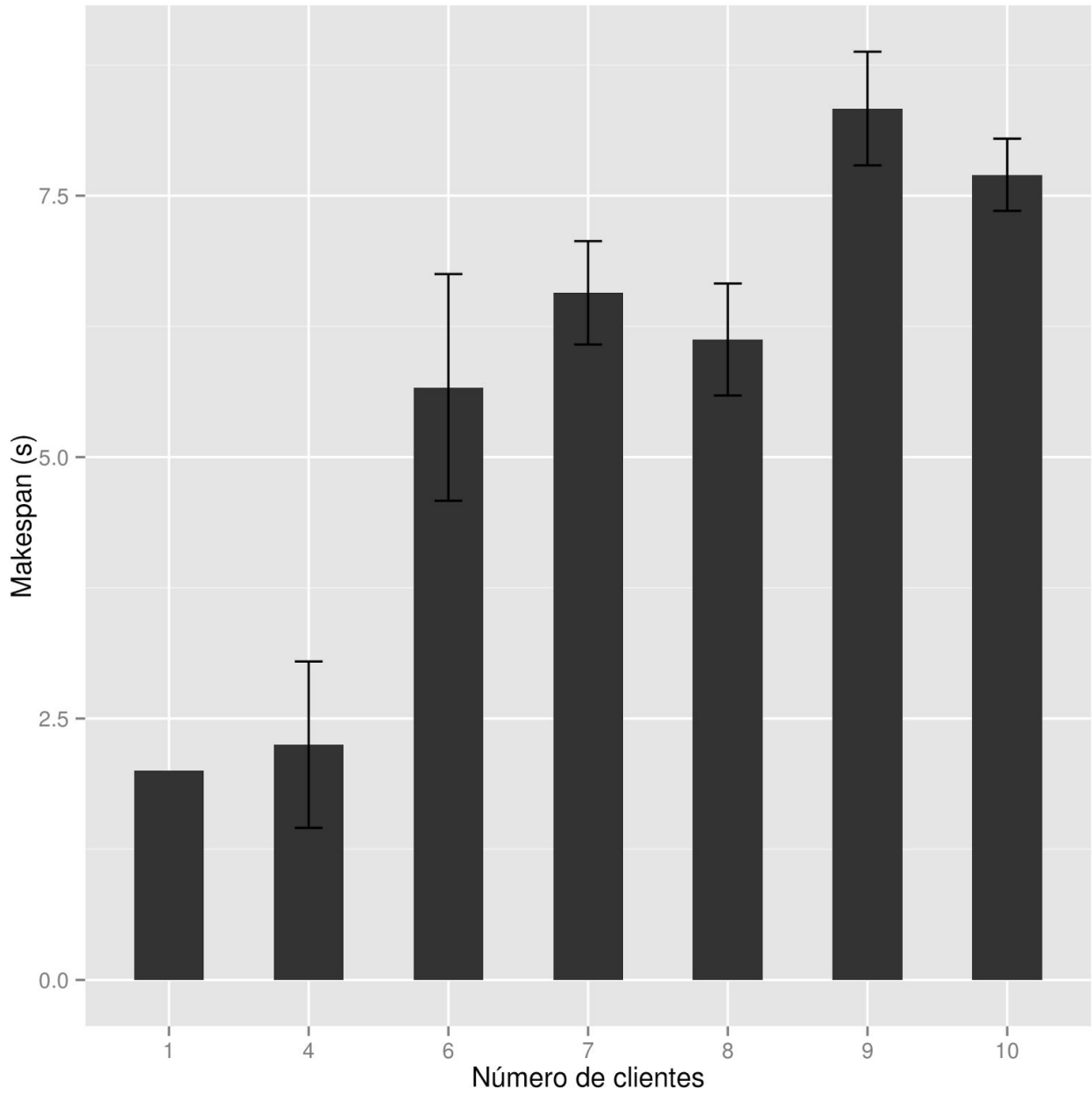

Gráfico 11: Desempenho do BeeFS em operações de escrita com um workload de 10Mb e 50 arquivos, realizadas com os componentes do BeeFS em uma mesma JVM aumentando ao passo que se aumenta a concorrência. No eixo y, o tempo decorrido de execução (em segundos)

CONCLUSÃO

O BeeFS é um sistema de arquivos com um modelo de arquitetura eficiente, barato e escalável, mas que precisou passar por adaptações necessárias para torná-lo adequado a certas normas, como o padrão POSIX. As adaptações trouxeram ao longo da evolução do BeeFS um custo de desempenho que comprometem um pouco sua eficiência quando comparado com o NFS, por exemplo, considerado um sistema de arquivos distribuído muito popular. Este projeto buscou por meio da experimentação caracterizar o desempenho do BeeFS e identificar possíveis gargalos. Um dos gargalos identificados, que na verdade estava relacionado com um erro no código do sistema, foi identificado e corrigido (os resultados mostrados no relatório já consideram a versão sem esse erro), enquanto outros gargalos poderão ser encontradas, a partir de comparativos entre resultados atuais e posteriores realizados pela equipe. Os resultados de escalabilidade puderam confirmar que o desempenho do BeeFS piora a medida que se aumenta o número de requisições no servidor de metadados (Queenbee), como acontece em qualquer outro sistema de arquivos. Embora, no caso do BeeFS, em vantagem, as requisições sejam mais simples e rapidamente resolvidas, por causa de sua arquitetura.

AGRADECIMENTOS

Ao CNPq pela manutenção das bolsas, PIBITI e produtividade em pesquisa;
À Professora Lívia pela orientação e todo o apoio.
Aos colegas do Laboratório de Sistemas Distribuídos do Departamento de Sistemas e Computação, onde este trabalho foi desenvolvido, por todo o apoio e contribuição.

REFERÊNCIAS BIBLIOGRÁFICAS

1. B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS version 3 design and implementation. In Proceedings of the Summer USENIX Conference, pages 137–152, 1994.
2. Z. Edward and R. Zayas. Afs-3 programmer's reference: Architectural overview, 1991.
3. M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steere. Coda: a highly available file system for a distributed workstation environment. IEEE Transactions on Computers, 39(4):447–459, 1990.
4. Carla A. Souza, Ana Clara Lacerda, Jonhnny W. Silva, Thiago Emmanuel Pereira, Alexandro Soares, Francisco Brasileiro. BeeFS: Um Sistema de Arquivos Distribuído POSIX Barato e Eficiente para Redes Locais. In Anais do XXVIII Simpósio Brasileiro de Redes de Computadores - SBRC'2010 - Salão de Ferramentas, pages 1033-1040, Maio 2010.
5. POSIX.1-2008, 2008. Disponível em <http://pubs.opengroup.org/onlinepubs/9699919799/toc.htm >. Acesso em fevereiro de 2013.
6. Sudharshan S. Vazhkudai, Xiaosong Ma, Vincent W. Freeh, Jonathan W. Strickland, Nandan Tammineedi, and Stephen L. Scott. Freeloader: Scavenging desktop storage resources for scientific data. In Proceedings of the 2005 ACM/IEEE conference on Supercomputing, pages 56. IEEE Computer Society, 2005.
7. Niraj Tolia, Michael Kozuch, Mahadev Satyanarayanan, Brad Karp, Thomas C. Bressoud, and Adrian Perrig. Opportunistic use of content addressable storage for distributed file systems. In USENIX Annual Technical Conference, General Track'03, pages 127–140, 2003.
8. Heshan Lin, Xiaosong Ma, Jeremy Archuleta, Wu-chun Feng, Mark Gardner, and Zhe Zhang. Moon: Mapreduce on opportunistic environments. In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing, pages 95–106. ACM, 2010.
9. M. Satyanarayanan, A Survey of Distributed File Systems. Department of Computer Science, Carnegie Mellon University, February 1989.
10. J. W. Silva, , T. Pereira, C. A. Souza, and F. Brasileiro. Computacao intensiva em dados com mapreduce em ambientes oportunistas. In Anais do SBRC'2011 - Salão de Ferramentas, 2011.
11. C. A. Souza, A. C. Lacerda, J. W. Silva, A. S. Soares T. Pereira, and F. Brasileiro. Beefs: Um sistema de arquivos distribuído POSIX barato e eficiente para redes locais. In Anais do SBRC'2010 - Salão de

Ferramentas, 2010.

12. A. S. Soares, T. Pereira, J. W. Silva, and F. Brasileiro. Um modelo de armazenamento de metadatados tolerante a falhas para o DDGfs (in portuguese) . In WSCAD-SSC 2009: Proceedings of the 10th Computational Systems Symposium, October 2009.

13. T. Pereira. Políıticas de alocação e migração de arquivos em sistemas de arquivos em sistemas distribuídos para redes locais. Dissertação de mestrado, COPIN - Universidade Federal da Paraíba, Campina Grande, 2010.

14. J. W. Silva. Processamento paralelo de grandes quantidades de dados sobre um sistema de arquivos distribuído POSIX. Dissertação de mestrado, COPIN - Universidade Federal da Paraíba, Campina Grande, Maio 2010.

15. José Nathaniel Lacerda de Abrante, Lívia Maria Rodrigues Sampaio Campos. Suporte à evolução e testes no BeeFS. Technical Report, Universidade Federal de Campina Grande, 2012.

16. Andrew S. Tanenbaum, Maarten Van Steen. Distributed File Systems, Principles and Paradigms, 2nd edition. Pearson Prentice Hall, 2007.

17. Avishay Traeger, Erez Zadok, Nikolai Joukov, Charles P. Wright. A nine year study of file system and storage benchmarking, ACM Transactions on Storage, 2008.

18. R. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley- Interscience, 1991