Fausto Guzzo da Costa **nUsp: 5230736**
Filipe Del Nero Grillo **nUsp: 5378140**
Vinicíus Augusto Tagliatti Zani **nUsp: 5118935**

Avaliação de Sistemas de Arquivos

(Ext4, ReiserFS e XFS)

São Carlos - SP, Brasil

20 de junho de 2011

Fausto Guzzo da Costa **nUsp: 5230736**
Filipe Del Nero Grillo **nUsp: 5378140**
Vinicíus Augusto Tagliatti Zani **nUsp: 5118935**

Avaliação de Sistemas de Arquivos

(Ext4, ReiserFS e XFS)

Monografia apresentada para conclusão da disciplina de Sistemas Operacionais da pós-graduação do ICMC-USP em 2011

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
UNIVERSIDADE DE SÃO PAULO
ICMC - USP

São Carlos - SP, Brasil

20 de junho de 2011

Resumo

A complexidade crescente presente nos computadores faz com que diversas funcionalidades se tornem obsoletas. Para o domínio de sistemas operacionais, a evolução também se mostra verdadeira. No contexto de utilização de disco, o sistema operacional encapsula a comunicação com periféricos em chamadas de rotina simples e genéricas que permitem uma utilização transparente pela camada de aplicações. A gestão de arquivos lógicos e seu mapeamento em partes físicas é foco de estudo desde há muito tempo no domínio de sistemas operacionais, pois é sabido que o armazenamento secundário é um grande gargalo a ser enfrentado em aplicações que o utilizam. Assim sendo, diversas propostas têm sido feitas para gestão interna de arquivos de sistemas operacionais. No âmbito de sistemas operacionais de código aberto, Ext4, XFS e ReiserFS foram lançados ao público como sistemas de arquivos eficientes para o sistema operacional Linux, e têm tido adoção crescente como consequência das funcionalidades que oferecem para realização de gestão de arquivos. Assim sendo, neste trabalho os sistemas de arquivos Ext4, XFS e ReiserFS foram analisados em comparação uns aos outros. Ao fim da realização dos experimentos foi possível concluir que cada uma das abordagens tem pontos fortes e fracos, que devem ser levados em consideração de acordo com o tipo de uso que se fará do sistema de arquivos.

Sumário

Lista de Figuras

Lista de Tabelas

1 *Introdução*

1.1 Contexto e Motivação

A complexidade crescente presente nos computadores, motivada pelas necessidades cada vez mais apuradas de um mercado tecnológico em plena evolução, faz com que diversas funcionalidades que antigamente atendiam plenamente às necessidades se tornem obsoletas. Tais mudanças podem ser observadas no âmbito de *hardware* e *software*. Para os softwares, mais especificamente dentro do domínio de sistemas operacionais, a evolução também se mostra verdadeira, pois esse conjunto de sistemas é responsável por grande parte da integração que ocorre entre equipamentos de diversos fabricantes, além de prover abstração para que os componentes possam ser utilizados por terceiros, por via de softwares executáveis no contexto dos sistemas operacionais. Portanto, torna-se crescente a necessidade de melhores técnicas de utilização dos componentes periféricos, ao passo que a organização interna dos sistemas operacionais deve ser mantida coesa, segura e eficiente, objetivando uma evolução de longo prazo ao mesmo tempo em que as soluções para os problemas atuais são elaboradas.

No contexto de utilização de disco, o sistema operacional encapsula as diversas marcas de diversos fabricantes em chamadas de rotina simples e genéricas que permitem uma utilização transparente pela camada de aplicações. Dessa forma o desenvolvedor de aplicações para o sistema operacional em questão deve saber somente o conjunto de chamadas genéricas que lhe permite executar as operações de leitura, escrita, abertura e fechamento de arquivos. Cabe assim ao sistema operacional gerir o fluxo de informações e sua organização interna, abstraindo os arquivos lógicos — aqueles visíveis para as aplicações — em arquivos físicos — mapeados em *hardware*. A gestão de arquivos lógicos e seu mapeamento em partes físicas é foco de estudo desde há muito tempo no domínio de sistemas operacionais, pois é sabido que o armazenamento secundário é um grande gargalo a ser enfrentado em aplicações que o utilizam.

Assim sendo, diversas propostas têm sido feitas para gestão interna de arquivos de sistemas operacionais. No contexto de sistemas operacionais de código aberto, podemos encontrar propostas de sistemas de arquivos. Nesse âmbito Ext4, XFS e ReiserFS

foram lançados ao público como sistemas de arquivos eficientes para o sistema operacional Linux, e têm tido adoção crescente como consequência das funcionalidades que oferecem para realização de gestão de arquivos.

1.2 Objetivos

Este trabalho tem como objetivo avaliar o desempenho dos sistemas de arquivos Ext4, XFS e ReiserFS em comparação uns aos outros. Esta avaliação pretende explicitar vantagens e desvantagens destes sistemas de arquivos e esclarecer em que situações cada um possui melhor desempenho.

1.3 Organização do trabalho

O Capítulo 2 descreve os sistemas de arquivos que serão avaliados e apresenta as características necessárias para o correto entendimento da execução dos experimentos desse trabalho. O capítulo 3, as ferramentas utilizadas para execução dos experimentos e o ambiente no qual foram executados são apresentados. No Capítulo 4 descreve-se em detalhes o planejamento e a execução dos experimentos. São definidos os fatores e níveis que serão avaliados, a lista de todos os experimentos e o planejamento adotado. O capítulo 5 traz os resultados obtidos através da coleta realizada nos experimentos e possuis gráficos comparativos dos experimentos bem como análise da influência dos fatores. Por fim, o Capítulo 6 traz as conclusões sobre cada uma das etapas do experimento e considerações finais.

2 *Fundamentação*

2.1 Considerações iniciais

Este trabalho envolve técnicas de Avaliação de Desempenho e os Sistemas de Arquivos XFS, Ext4 e ReiserFS. Este capítulo descreve os conceitos e teorias dos tópicos citados.

2.2 XFS

Nessa seção será primeiramente apresentada a história do desenvolvimento do XFS, e após a arquitetura do mesmo.

2.2.1 História

O sistema de arquivos XFS foi desenvolvido para o sistema operacional IRIX, da Silicon Graphics em 1994. Ele foi planejado para resolver os problemas do sistema de arquivos que era utilizado, o EFS (*Extents File System*). Os problemas apresentados na época eram decorrentes de novas aplicações e novos *hardwares* que estavam surgindo. Aplicações para edição de vídeo descomprimidos, por exemplo, precisavam alocar centenas de *gigabytes* de espaço, e o EFS suportava apenas 8 GB de arquivos, sendo que cada arquivo podia ter no máximo 2 GB. Além disso, o acesso ao disco rígido precisava ser otimizado, pois havia muito *overhead* [3].

O XFS foi distribuído na licença GNU *General Public License* em 2000, e o primeiro suporte à esse sistema de arquivos foi em 2001, no kernel 2.4 do Linux. Hoje ele está disponível para quase todas as distribuições Linux.

2.2.2 Arquitetura

Um sistema de arquivos XFS é dividido em vários Allocation Groups (AGs) de mesmo tamanho. Um AG pode ser pensado como sendo um sistema de arquivos autônomo, podendo ter até um *terabyte* de tamanho, e tem as seguintes características:

(i) Contém um superbloco descrevendo as suas informações;

(ii) Capacidade de gerenciar o seu espaço livre;

(iii) Aloca e busca por *inodes*.

A ideia do desenvolvimento do XFS com AGs, é que conforme o acesso concorrente aumenta no sistema de arquivos, o XFS consegue paralelizar as várias operações sem degradar o desempenho.

Cada AG é dividido em 4 partes como mostra a figura 1. A primeira parte é o superbloco, responsável por guardar todas as informações do AG, como tamanho dos blocos, dos setores e dos *inodes*, versão do XFS, contadores para número de *inodes* alocados e livres, etc. Esse superbloco é replicado dentro do AG, então caso haja uma falha, ele pode ser recuperado.

A segunda parte do AG é o gerenciamento de espaço livre, que é feito através do uso de duas árvores B+. Por motivos de otimização, uma delas é ordenada por número do bloco, e a outra por quantidade de espaço contiguo.

A terceira parte é responsável pelo gerenciamento de *inodes* no sistema de arquivos, sendo responsável por alocar e buscar *inodes*. Os *inodes* são alocados em grupos de 64, e são buscados através de uma árvore B+.

A quarta parte é responsável por reservar espaço para possível crescimento das árvores B+ descritas previamente. Esse espaço não pode ser alocado por nenhum outro tipo de dado.

2.3 Ext4

Nesta seção será apresentada a história do sistema de arquivos Ext até a chegada em sua última versão, a Ext4. Em seguida essa última versão será estudada em mais detalhes.

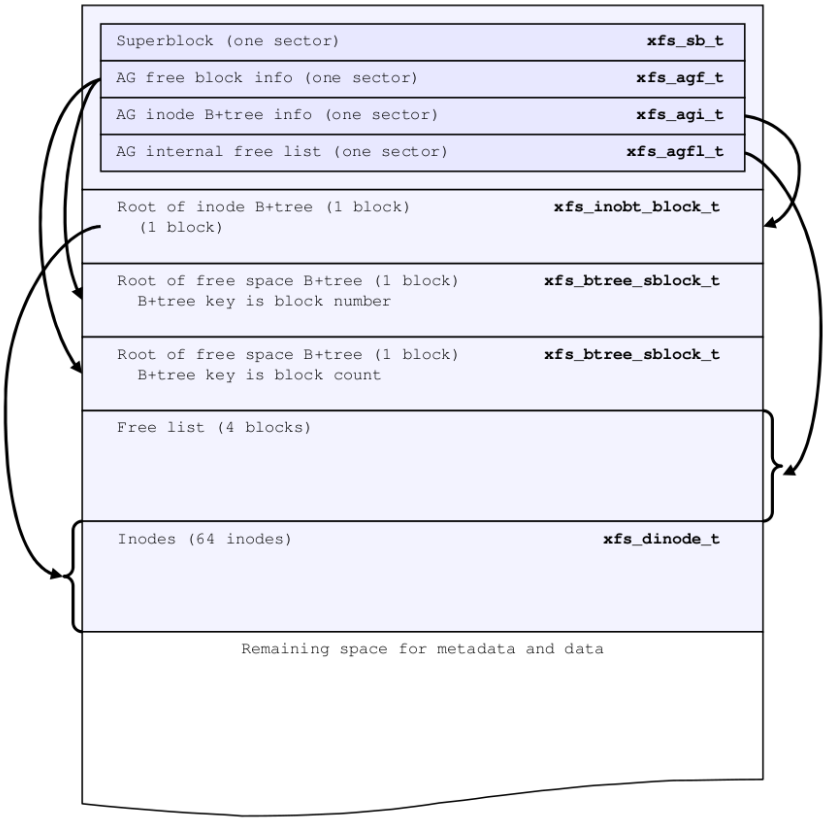

Figura 1: Detalhes de um *Allocation Group* do XFS [1].

2.3.1 História

O sistema de arquivos Ext — *Extended File System* — nasceu através de pesquisas realizadas por Rémy Card et al. em 1992 [4], e foi integrada ao Linux em sua versão 0.96c. Esse novo sistema de arquivos resolveu duas grandes limitações impostas pelo sistema de arquivos Minix [5]:

(i) O limite de 64 MB[1] para tamanho de arquivo: Ext passou a permitir arquivos de até 2 GB[2]; e

(ii) O limite de 14 caracteres para o nome do arquivo: Ext passou a permitir nomes de arquivos de até 255 caracteres.

Entretanto, alguns problemas ainda estavam presentes nessa implementação. Por exemplo, não havia acesso separado, modificação de *inodes* e marcação de datas de modificação[3]. Este sistema de arquivos utilizava listas encadeadas para manter controle dos

[1] *Mega Bytes*: 1024 Bytes.
[2] *Giga Bytes*: 1024 MB.
[3] O termo utilizado em inglês é *timestamp*.

blocos livres, e isso degradava a performance [4]: à medida que o sistema operacional era utilizado, a lista se tornava não-ordenada, e o sitema de arquivos se tornava fragmentado.

Como resposta a estes problemas, dois novos sistemas de arquivos foram desenvolvidos em versão alfa, em Janeiro de 1993: o sistema de arquivos Xia e o Ext2 — *Second Extended File System*. Enquanto o Xia representava algumas melhorias para o sistema Minix, o Ext2 era baseado no código do Ext e trazia melhorias como [4]:

(i) Suporte a padrões de arquivos Unix padrão;

(ii) Capacidade de gerir sistemas de arquivos criados em partições de até 4 TeraBytes, eliminando a necessidade de várias partições para utilização de discos grandes;

(iii) Capacidade de nomes de arquivos de até 1012 caracteres; e

(iv) Reserva de blocos para super usuário (*root*), em geral 5%. Isso permite com que administradores de sistema recuperem o sistema em situações onde o disco fica cheio.

Em novembro de 2001 foi introduzida uma nova versão do sistema de arquivos Ext, a Ext3. Com muitas novidades, essa versão foi rapidamente integrada às versões mais populares do Linux [2]. Entre as inovações apresentadas por essa versão está a introdução de *journaling* como ferramenta para aumento da confiabilidade de sistema, em casos onde há interrupção repentina do funcionamento do sistema operacional. A função do *journaling* é salvar em um *log* as operações a serem executadas. Esse *log* é utilizado na inicialização do sistema para restaurá-lo a um estado válido. Com o passar do tempo, o sistema de arquivos Ext3 foi tornando-se limitado para as necessidades de utilização, tanto pessoal quanto na indústria, especialmente em relação ao seu limite de tamanho de arquivos. Assim uma atualização desse sistema de arquivos foi feita em 2006 [2]. Em Dezembro de 2008 surgiu o Ext4. Esse sistema será detalhado na sub-seção 2.3.2.

Por fim, vale observar que as versões mais recentes do Ext sempre suportam as versões mais antigas, o que justifica a facilidade de adoção pelo mercado: não há esforço em migração de dados para novos formatos, pois o sistema pode operar ao mesmo tempo com partições de versões anteriores.

2.3.2 Ext4

O sistema de arquivos Ext4 vem sendo utilizado no Linux desde a sua versão 2.6.19. Seus grandes objetivos são de resolver problemas de escalabilidade e de gargalo do seu

antecessor — o Ext3, detalhado na sub-seção 2.3.1. São avanços notados nesse sitema de arquivos [2]:

1. Utilização de 48 bits para mapeamento de blocos, frente aos 32 bits utilizados pelo Ext3. Isso permite com que sistemas de arquivos de até 256 TB sejam mapeados;

2. Utilização de *extents* para gravação de arquivos. Um *extent* é uma posição contígua no disco, representada por início e fim em um descritor contido em um *inode*, num limite de quatro *extents* por *inode*. Assim, arquivos fragmentados exigem mais *extents* que um arquivo pequeno ou contíguo em disco. Um *extent* pode ter até 2^{15} blocos, ou 128 MB quando os blocos são de 4 KB. Sua utilização otimiza tanto leitura quanto escrita em disco para arquivos grandes. A solução implementada no Ext3 — mapeamento indireto de bloco — era boa para arquivos pequenos e esparsos, mas ineficiente para casos onde arquivos grandes eram utilizados. Esse mapeamento de *extents* é observado na Figura 2;

3. Utilização de bits reservados de *inode* para mapear blocos de arquivos, elevando a limitação de tamanhos de arquivos de 2 TB (Ext3)) para 16 TB (Ext4). Há um planejamento para que o limite seja elevado nas melhorias futuras desse sistema de arquivos;

4. Eliminação de restrição de número de subdiretórios contidos em um único diretório, e indexação de diretórios usando uma árvore H[4] de profundidade fixa. O sistema de arquivos Ext3 possuía um limite de 32000 arquivos;

5. Pré-alocação persistente, onde os blocos são reservados fisicamente no disco para arquivos que ainda não os ocuparam. Essa característica é útil para minimizar fragmentação de arquivos;

6. Alocação atrasada de blocos múltiplos, onde as operações de escrita são mantidas em *cache* até que ocorra um *flush* de página, e ele seja de fato aplicado em disco. Essa característica melhora utilização de disco e minimiza fragmentação, fazendo melhor uso dos *extents*;

7. Melhorias na forma como a integridade do sistema de arquivos é verificada (usando a ferramenta *e2fsck*). O ganho de performance é visível quando comparado à versão Ext3; e

8. Utilização de *checksum* CRC32 nos metadados utilizados para recuperação do sistema de arquivos — o *journal*. Assim o sistema não confia cegamente nos dados de reparação, e pode evitar danos de recuperação indevida.

[4]Uma árvore H é uma implementação de árvores B usando *hashes* de 32 bits.

As características acima descrita permitem obter uma melhor visão do sistema de arquivos Ext4, utilizado na avialiação de desempenho conduzida como foco desse trabalho.

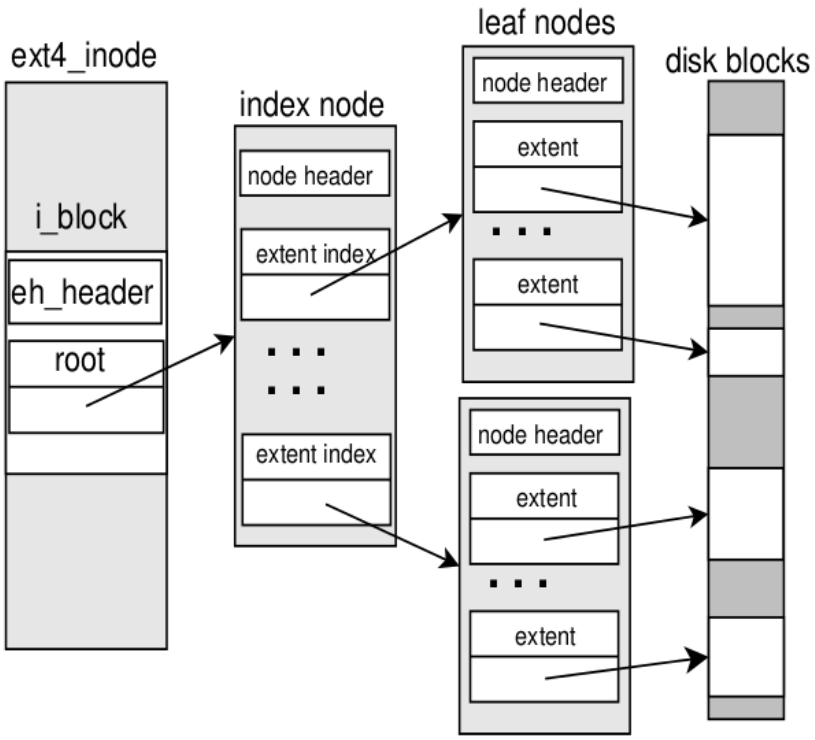

Figura 2: Uso de *extents* no sistema de arquivos Ext4 [2].

2.4 ReiserFS

O ReiserFS foi criado em 1996 por Hans Thomas Reiser e se tornou muito famoso, entre outras coisas, por ser o primeiro sistema de arquivos a utilizar a técnica de *journaling* para melhorar a confiabilidade do sistema. No caso do ReiserFS, o *journaling* embora sejam guardadas informações sobre todo o sistema de arquivos, ele tem a intenção de garantir a integridade dos metadados dos arquivos, ou seja, ele não é capaz de reconstruir todo o sistema de arquivos a partir do *journal*.

Passou a ser suportado oficialmente pelo kernel do linux a partir de sua versão 2.4 e chegou a ser adotado como sistema de arquivos padrão da distribuição comercial da Novell, o SUSE Linux Enterprise. Sua adoção foi baseada no fato do Reiser ser o único sistema de arquivos da época que possuia *journaling* e também em acordos de suporte com a empresa de de seu criador a Namesys.

Este sistema utiliza árvores B* para melhorar o desempenho em operações de busca. De acordo com a documentação encontrada sobre o sistema[6] ele pode ser utilizado com quatro opções de tamanho de bloco: 4096 (padrão), 512, 1024 e 8192 KBytes. No entanto, durante testes preliminares com o sistema Kubuntu 10.10 não foi possível criar um sistema de arquivos Reiser com tamanho de bloco diferente de 4096 KBytes. Em função disso, nas etapas 2 e 3 da avaliação o fator B - Tamanho do bloco não foi utilizado.

Á estrutura de árvore utilizada possui dois tipos de nós: Nós internos e nós folhas e cada nó é um bloco do disco. O primeiro tipo é utilizado para manter a estrutura de árvore, apontando para outros blocos. Os nós folha são os itens propriamente ditos. No ReiserFS os itens podem representar arquivos, diretórios ou *stat item*. Os arquivos podem ser de dois tipos, itens diretos ou indiretos, dependendo do tamanho do arquivo.

Cada diretório ou arquivo é sempre precedido por um *stat item* que contém metadados sobre o arquivo ou diretório que o sucede.

A Figura 3 mostra um exemplo da estrutura de árvore utilizada pelo ReiserFS para organizar os arquivos no disco.

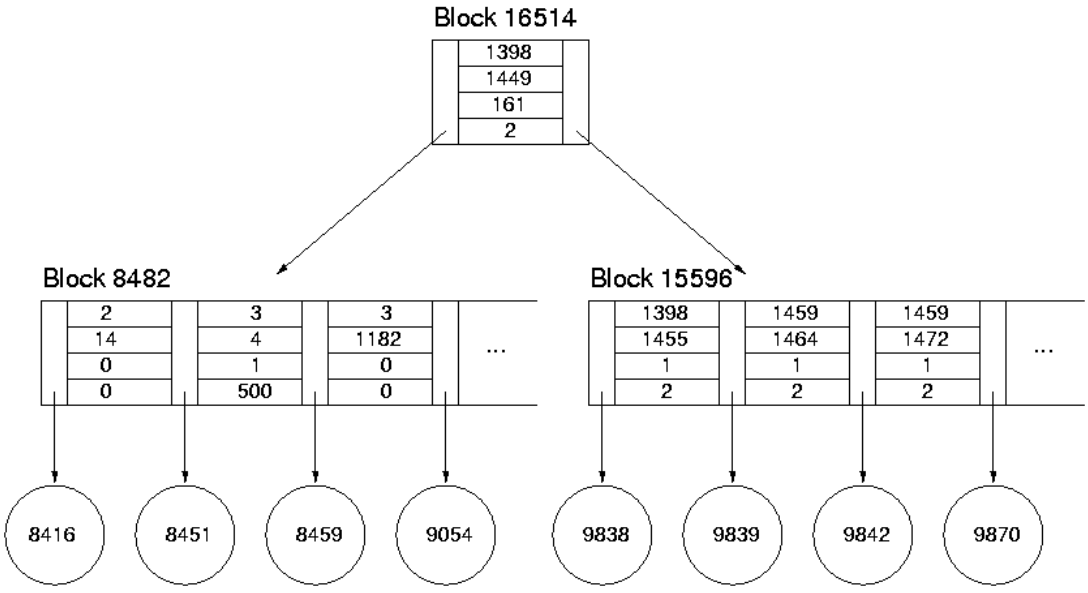

Figura 3: Modelo de árvore utilizado pelo ReiserFS com blocos internos representados pelos blocos retangulares e blocos folha representados por esferas.

3 *Métodos e Ferramentas*

A seguir encontra-se uma descrição detalhada da ferramenta utilizada e o ambiente no qual os experimentos foram executados.

3.1 IOzone

A ferramenta IOzone é uma ferramenta de avaliação de desempenho de sistemas de arquivo [7]. Esta ferramenta foi disponibilizada para diversas versões de sistemas operacionais, incluindo Kubuntu Linux, a distribuição Linux utilizada para realização da avaliação de desempenho desse trabalho. Quando a ferramenta é utilizada, ela simula a utilização dos discos de uma máquina, ao passo em que faz medições de tempo, utilização e taxa de entrada/saída para os algoritmos utilizados. Assim, são operações utilizáveis e mensuráveis:

- Read;
- Write;
- Re-read;
- Re-write;
- Read backwards;
- Read strided;
- Fread;
- Fwrite;
- Random read;
- Pread;
- Mmap;
- Aio read; e
- Aio write.

No contexto desta avaliação de desempenho foram utilizadas somente medidas para *Read*, *Random read*, *Write* e *Random write*.

3.2 Ambiente

Os experimentos foram executados computador desktop com as seguintes configurações:

- Processador: AMD Atlhon 64bits X2 Dual Core Processor 4000+
- Memória: 1GB/2GB (Dois pentes de 1GB) DDR2 800Mhz
- HD: Samsung SP2004C 200GB 7200rpm 8MB de cache
- Sistema operacional: Kubuntu Linux 10.10 32bits, Kernel 2.6.35-28-generic

Para execução das repetições de cada experimento foi criado um *script* bash que roda automaticamente todas as variações de fatores para a configuração de memória RAM da máquina. Então é necessário alterar a configuração de memória manualmente e depois executar o *script* novamente. O *script* pode ser encontrado no Anexo A.

4 *Planejamento e execução*

Neste capítulo são especificados o modelo de avaliação que foi adotado, quais as variáveis de resposta medidas, fatores e níveis considerados e como os experimentos foram divididos em etapas.

4.1 Variáveis de resposta, fatores e níveis

Para a escolha das variáveis de resposta foram escolhidas as operações mais comuns utilizados pelo sistemas operacionais que são escrita e leitura. Portanto, as variáveis de resposta escolhida foram:

- Leitura
- Escrita
- Leitura aleatória
- Escrita aleatória

Para a avaliação foi adotado o planejamento fatorial completo 2^4 na etapa 1 e fatorial completo 2^3 para as etapas 2 e 3 para manter a complexidade dos complexidade dos experimentos em um nível aceitável foram utilizados apenas 4 fatores que acredita-se que poderiam ter maior influência sobre os resultados dos experimentos e cada um deles com 2 níveis de variação.

Para simplificar a execução dos experimentos e ainda assim comparar todos os sistemas uns com os outros, a avaliação foi dividida em 3 etapas. Na etapa 1 foram comparados os sistemas de arquivos Ext4 e o XFS. A Tabela 1 mostra os fatores e níveis escolhidos para a etapa 1.

Na etapa 2 foram comparados os sistemas de arquivos Ext4 e ReiserFS e a Tabela 2 mostra os fatores e níveis adotados para a etapa 2. Como foi dito anteriormente, o

Fator	Nível 1	Nível -1
A - Sistema de arquivo	Ext4	XFS
B - Tamanho do bloco	4KB	1KB
C - Memória RAM	1GB	2GB
D - Tamanho do arquivo	64KB	64MB

Tabela 1: Fatores e níveis escolhidos para a Etapa 1.

ReiserFS não permite a mundança do tamanho do bloco, este fator não foi considerado para as etapas 2 e 3 e o bloco ficou fixado em 4096 bytes para ambos os sistemas comparados nessas etapas.

Fator	Nível 1	Nível -1
A - Sistema de arquivo	Ext4	ReiserFS
C - Memória RAM	1GB	2GB
D - Tamanho do arquivo	64KB	64MB

Tabela 2: Fatores e níveis escolhidos para a Etapa 2.

Na etapa 3 foram comparados os sistemas XFS e ReiserFS. A Tabela 3 mostra os fatores e níveis adotados para esta etapa.

Fator	Nível 1	Nível -1
A - Sistema de arquivo	XFS	ReiserFS
C - Memória RAM	1GB	2GB
D - Tamanho do arquivo	64KB	64MB

Tabela 3: Fatores e níveis escolhidos para a Etapa 3.

4.2 Experimentos

Na etapa 1 da avaliação foram realizados 16 experimentos para cobrir todas as variações possíveis entre os 2 níves de cada um dos 4 fatores definidos anteriormente. Para as etapas 2 e 3 foram realizados 8 experimentos, novamente para cobrir todas as variações entre os 2 níveis e 3 fatores definidos anteriormente. Para cada experimento foram realizadas 10 repetições resultando em um total de 160 execuções na etapa 1, 80 execuções na etapa 2 e 80 execuções na etapa 3 em função de possuírem 1 fator a menos do que a etapa 1.

A tabela 4 mostra os níveis utilizados em cada um dos 16 experimentos realizados na etapa 1 e a Tabela 5 mostra os níveis utilizados em cada um dos 8 experimentos

tanto da etapa 2 quanto da etapa 3. A definição dos fatores A, B, C e D podem ser encontradas nas Tabelas 1, 2 e 3

Experimento	A	B	C	D
1	1	1	1	1
2	1	1	1	-1
3	1	1	-1	1
4	1	1	-1	-1
5	1	-1	1	1
6	1	-1	1	-1
7	1	-1	-1	1
8	1	-1	-1	-1
9	-1	1	1	1
10	-1	1	1	-1
11	-1	1	-1	1
12	-1	1	-1	-1
13	-1	-1	1	1
14	-1	-1	1	-1
15	-1	-1	-1	1
16	-1	-1	-1	-1

Tabela 4: Níveis utilizados em cada um dos experimentos para a Etapa 1

Experimento	A	C	D
1	1	1	1
2	1	1	-1
3	1	-1	1
4	1	-1	-1
5	-1	1	1
6	-1	1	-1
7	-1	-1	1
8	-1	-1	-1

Tabela 5: Níveis utilizados em cada um dos experimentos para as Etapa 2 e 3

Para a execução das repetições foi utilizado o *script* bash que pode ser visto no Anexo A. Os testes executados com o IOzone *Write*, *Read*, *Random Read* e *Random Write* que correspondem, respectivamente, a Escrita, Leitura, Leitura aleatória e Escrita aleatória. Nos casos de Escrita e Leitura, o IOZone realiza as operações de leitura e escrita sequencial dentro de arquivos que ele mesmo cria com o tamanho especificado por um parâmetro pois é um fator desta avaliação. Nas operações de Escrita aleatória e Leitura aleatória, o IOZone realiza operaçõs de leitura e escrita não sequencial dentro dos arquivos criados por ele, ou seja, ele faz *seek* para posições aleatórios dentro do arquivo e então executa a operação.

A alteração para variar a quantidade de memória RAM foi realizada manualmente, retirando-se um dos dois módulos de 1GB de memória e então prosseguir com a execução dos experimentos.

5 *Resultados Obtidos*

Este capítulo apresenta os resultados obtidos nos experimentos nas Etapas 1, 2 e 3. Os gráficos de barra utilizados foram agrupados de modo que cada par de experimentos que possuam de variação entre eles apenas o fator Sistema de arquivo sejam exibidos juntos para facilitar a comparação de um sistema de arquivo em relação ao outro.

5.1 Etapa 1: comparação entre Ext4 e XFS

Na primeira etapa do trabalho os sistemas de arquivos Ext4 e XFS foram analisados em conjunto, a partir dos dados de experimentos de análise de desempenho realizados separadamente, conforme descrito no Capítulo 4.

Como pode ser observado na Figura 4, o sistema de arquivos XFS teve melhor desempenho para todos os experimentos envolvendo tamanho de blocos de 4 KB (experimentos 9 à 12). Entretanto, quando o bloco passa a ser de 1 KB (experimentos 13 à 16), nada pode se afirmar sobre qual dos dois sistemas de arquivos tem melhor desempenho, pois seus intervalos de confiança se sobrepõem. Nota-se também que o melhor desempenho de escrita (MB/s) foi obtido utilizando XFS com blocos de 4 KB e arquivos de tamanho de 64 MB, onde nada pode se afirmar sobre a memória, pois para ambos os valores de 1 GB e 2 GB (experimentos 10 e 12, respectivamente) as taxas de escrita com seus respectivos intervalos de confiança se sobrepõem.

Ao comparar XFS e Ext4 para operação de leitura — exemplo da Figura 5 — podemos notar que em dois experimentos o Ext4 possui melhor desempenho que XFS: na leitura de blocos de 1 KB, com memória de 2 GB e tamanho de arquivos de 64 KB; e em uma situação similar onde os tamanhos de arquivos são de 64 MB (experimentos 7 e 8, respectivamente). Apesar da diferença ser significativa para o primeiro caso, para o segundo não é. Nos demais casos nada se pode afirmar sobre um melhor desempenho, pois os resultados obtidos possuem intervalos de confiança sobrepostos quando comparados.

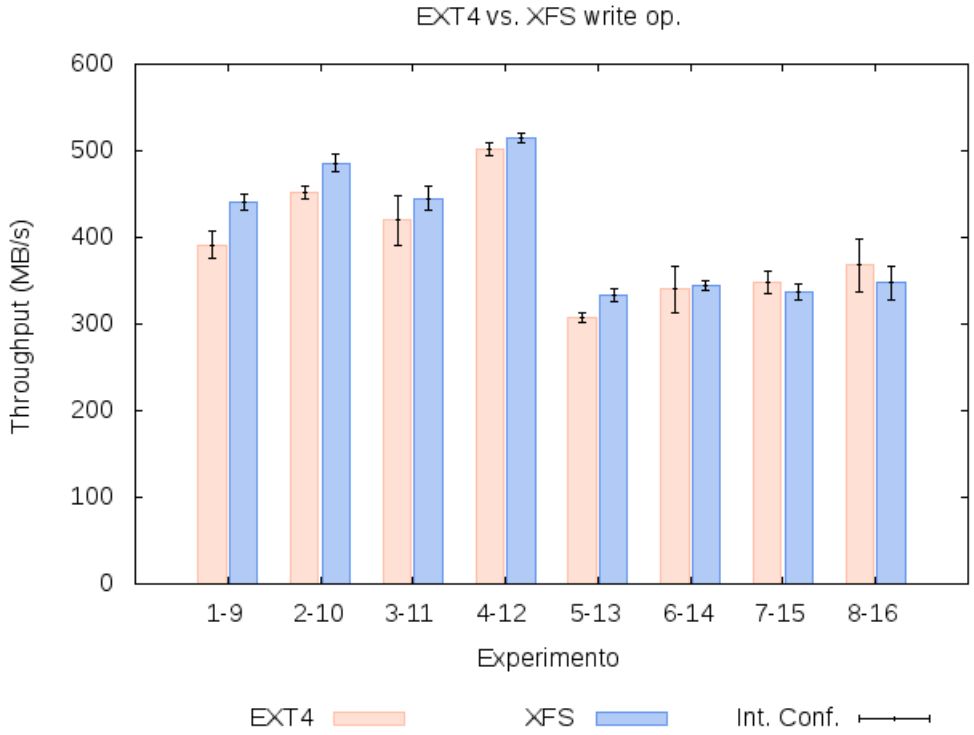

Figura 4: Gráfico comparando os experimentos da Etapa 1 na operação de Escrita.

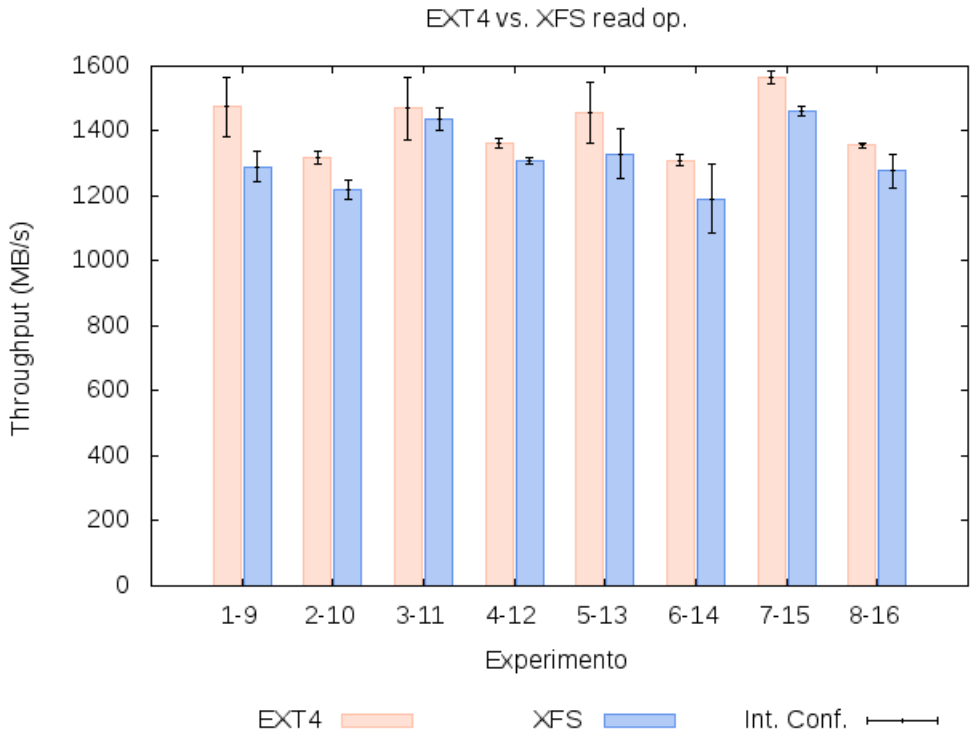

Figura 5: Gráfico comparando os experimentos da Etapa 1 na operação de Leitura.

Para a operação de leitura randômica, XFS e Ext4 obtiveram desempenhos similares, com diferenças tênues para grande parte dos casos, como visto na Figura 6. Um dos resultados com grande diferença — mas também grandes intervalos de confiança — foi observado quando blocos de 4 KB foram utilizados com 1 GB de memória com arquivos de tamanho de 64 KB (experimento 9). Nesse experimento, o Ext4 teve um desempenho *médio* superior ao XFS. Entretanto, os intervalos de confiança se sobrepõem nos limites, fazendo com que não seja possível afirmar com a certeza desejada qual se comporta da melhor forma sob tais circunstâncias.

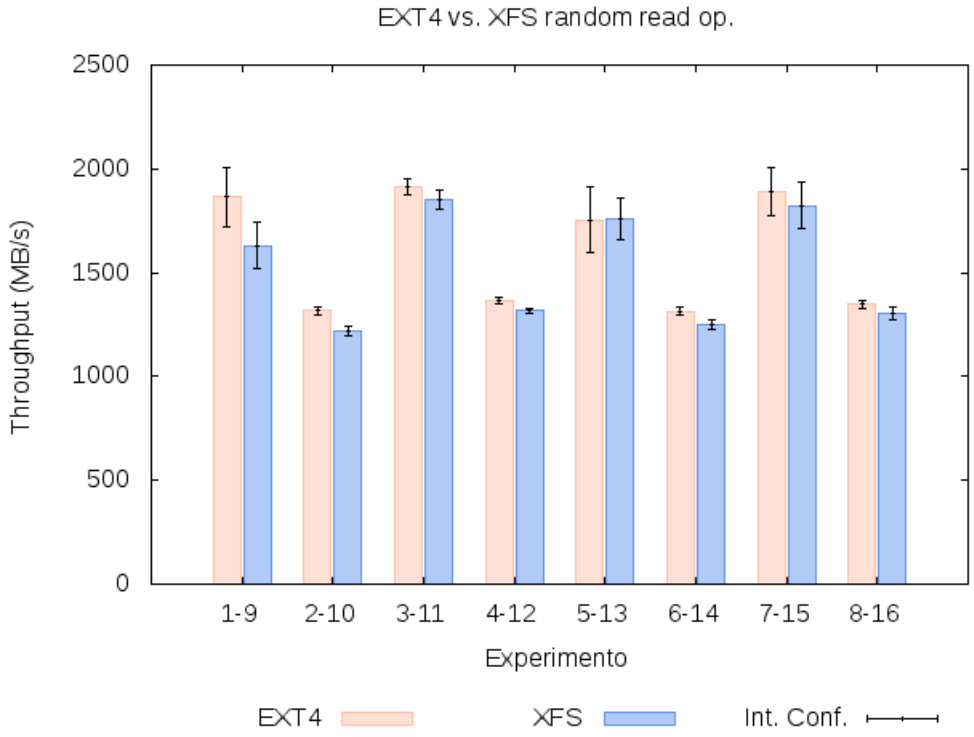

Figura 6: Gráfico comparando os experimentos da Etapa 1 na operação de Leitura aleatória.

Por fim, na comparação utilizando operações de escrita randômica (vista na Figura 7) o Ext4 teve um desempenho visilmente superior ao XFS quando os blocos eram de 4 KB, a memória de 1 GB e arquivos de 64 KB (experimento 1). Uma diferença um pouco menor, com *throughput* similar, pode ser notada com os mesmos fatores acima, porém tamanho de arquivo de 64 MB (experimento 2). De forma geral nesse teste os sistemas de arquivos se comportam de maneira similar, à parte exceções detalhadas previamente.

Por fim, ao utilizar os dados das medições, foi possível calcular as influências dos fatores, apresentados na Tabela 5.1.

Nos gráficos gerados a partir dessa tabela (vide Figura 5.1), é possível notar que

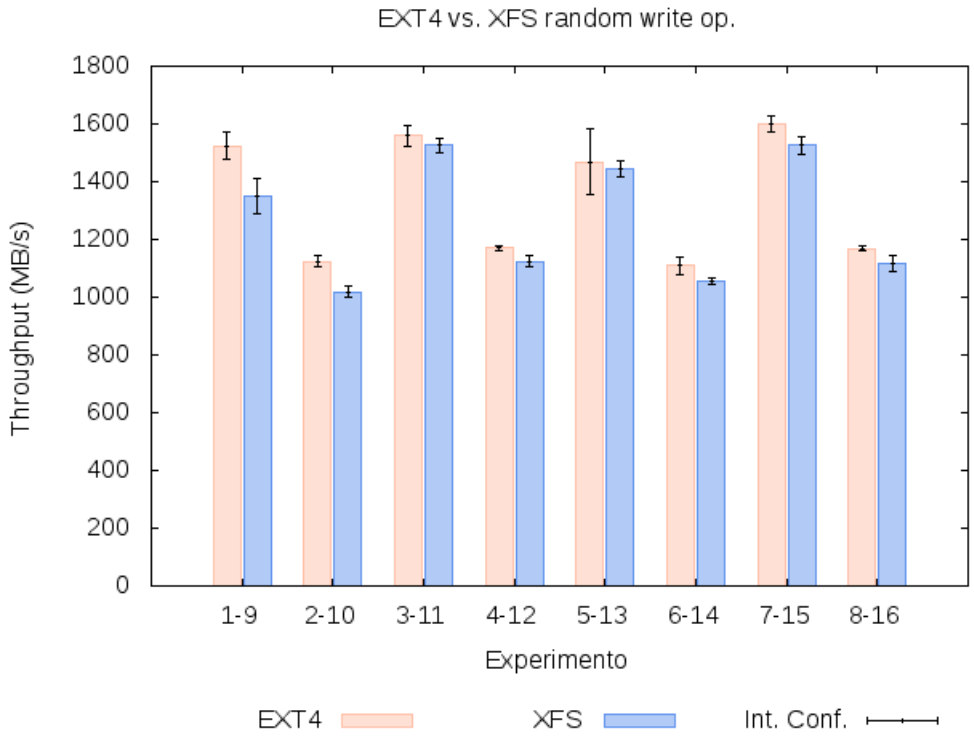

Figura 7: Gráfico comparando os experimentos da Etapa 1 na operação de Escrita aleatória.

o tamanho de blocos é um fator que possui grande influência quando comparamos o desempenho de Ext4 e XFS para a operação de escrita. Esse fator faz com que o desempenho ambos os sistemas de arquivos seja visivelmente degradado quando blocos pequenos são utilizados. Também podemos notar que o tamanho de arquivos é um fator de grande influência quando comparamos Ext4 e XFS para as operações de leitura, leitura randômica e escrita randômica. Arquivos maiores (64 MB) degradaram o desempenho de ambos os sistemas de arquivos.

Fator	Write	Read	Random Read	Random Write
iA	1.32	24.59	32.63	2.99
iB	78.78	0.16	0.00	0.08
iC	3.27	16.64	2.29	4.49
iD	10.12	50.71	62.44	90.90
iAB	1.44	0.14	0.37	0.23
iAC	1.03	2.55	0.11	0.20
iAD	0.29	0.40	1.44	0.03
iBC	0.12	0.37	0.02	0.01
iBD	3.13	1.78	0.01	0.04
iCD	0.10	0.56	0.20	0.23
iABC	0.02	0.71	0.23	0.57
iABD	0.00	0.04	0.14	0.04
iACD	0.03	0.32	0.00	0.01
iBCD	0.34	0.39	0.01	0.01
iABCD	0.01	0.63	0.11	0.16

Tabela 6: Tabela com os valores calculados de influência dos fatores para a Etapa 1 da avaliação de desempenho separados para o tipo de operação.

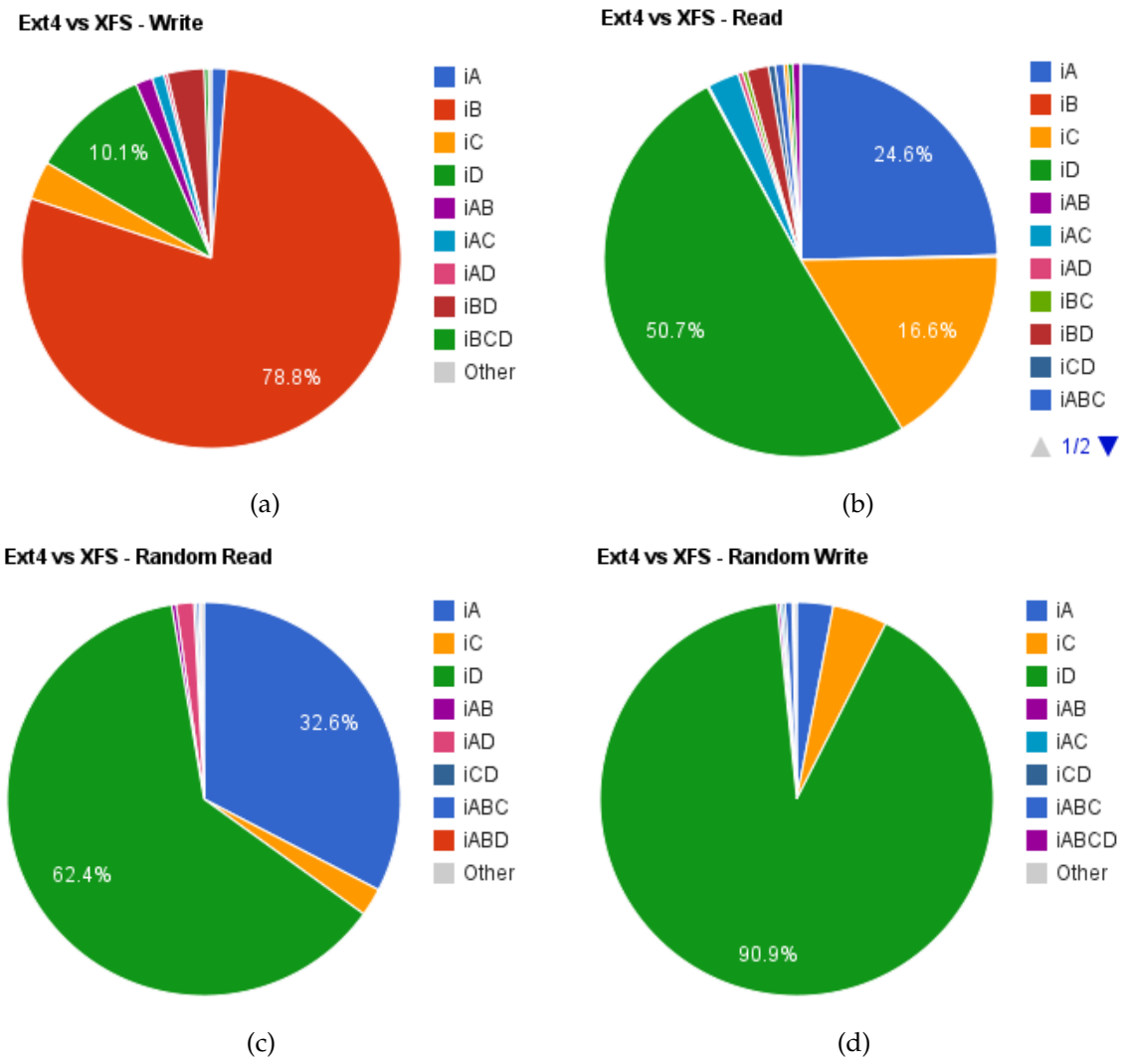

Figura 8: Gráficos com as influências dos fatores calculadas para as operações avaliadas na Etapa 1: (a) - Escrita, (b) - Leitura, (c) - Leitura aleatória e (d) - Escrita aleatória

5.2 Etapa 2: comparação entre Ext4 e ReiserFS

Na segunda etapa do trabalho os sistemas de arquivos Ext4 e ReiserFS foram analisados em conjunto, a partir dos dados de experimentos de análise de desempenho realizados separadamente, conforme descrito no Capítulo 4. Vale ressaltar que o ReiserFS não disponibiliza opção de utilização de blocos de tamanho de 1 KB. Portanto, o Fator B, conforme apresentado na Tabela 2, não foi utilizado.

Como pode ser observado na Figura 9, o sistema de arquivos Ext4 possui melhor desempenho que o ReiserFS para todos os experimentos realizados com a operação de escrita.

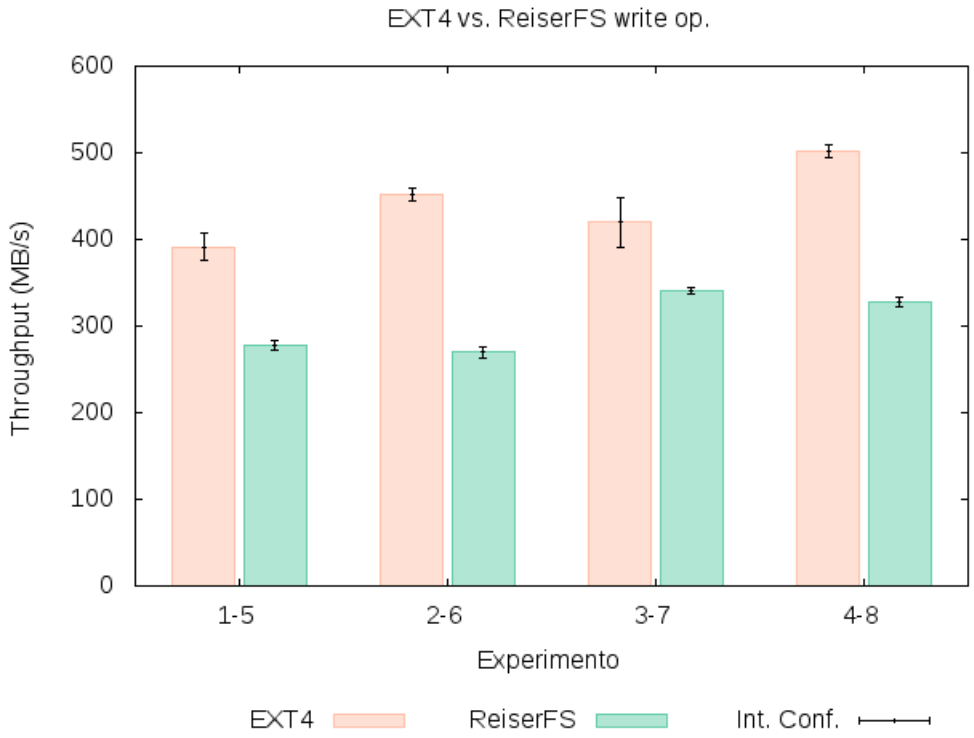

Figura 9: Gráfico comparando os experimentos da Etapa 2 na operação de Escrita.

Na Figura 10 — que representa experimentos realizados para a operação de leitura — houve resultados muito parecidos em quase todos os experimentos, com exceção do experimento 7, onde foi testado com 2 GB de memória e arquivos de 64 KB, e houve um *throughput* bem maior do ReiserFS.

Para a operação de leitura aleatória — representada na Figura 11 — os resultados não são conclusivos de um modo geral: o ReiserFS tem melhor desempenho para arquivos menores (64 KB) e memória de 2 GB. Quando há mais memória disponível — fator importante para *caching* — não se pode concluir sobre qual é o sistema de arqui-

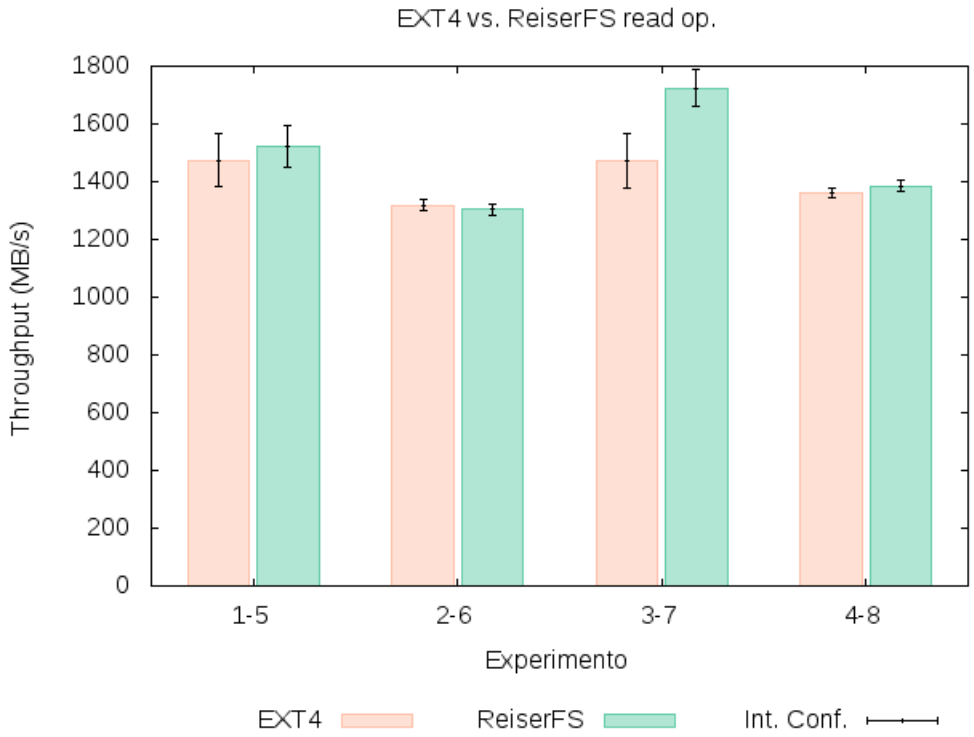

Figura 10: Gráfico comparando os experimentos da Etapa 2 na operação de Leitura.

vos mais eficiente, pois os intervalos de confiança se sobrepõem.

Por fim, para a operação de escrita randômica (vide Figura 12), nada se pode afirmar sobre o desempenho de ambos, pois os intervalos de confiança se sobrepõem.

Por fim, ao utilizar os dados das medições, foi possível calcular as influências dos fatores, apresentados na Tabela 7.

Fator	Write	Read	Random Read	Random Write
iA	78.06	9.11	1.61	4.39
iC	10.37	10.06	1.15	11.51
iD	3.79	64.00	95.71	82.09
iAC	0.46	5.60	0.09	0.02
iAD	7.06	8.04	1.43	0.33
iCD	0.08	0.46	0.00	1.51
iACD	0.18	2.72	0.00	0.15

Tabela 7: Tabela com os valores calculados de influência dos fatores para a Etapa 2 da avaliação de desempenho separados para o tipo de operação.

Nos gráficos gerados a partir dessa tabela (vide Figura 5.2), é possível notar que o tamanho de arquivos é um fator que possui grande influência quando comparamos o desempenho de Ext4 e ReiserFS para as operações de leitura, leitura randômica e

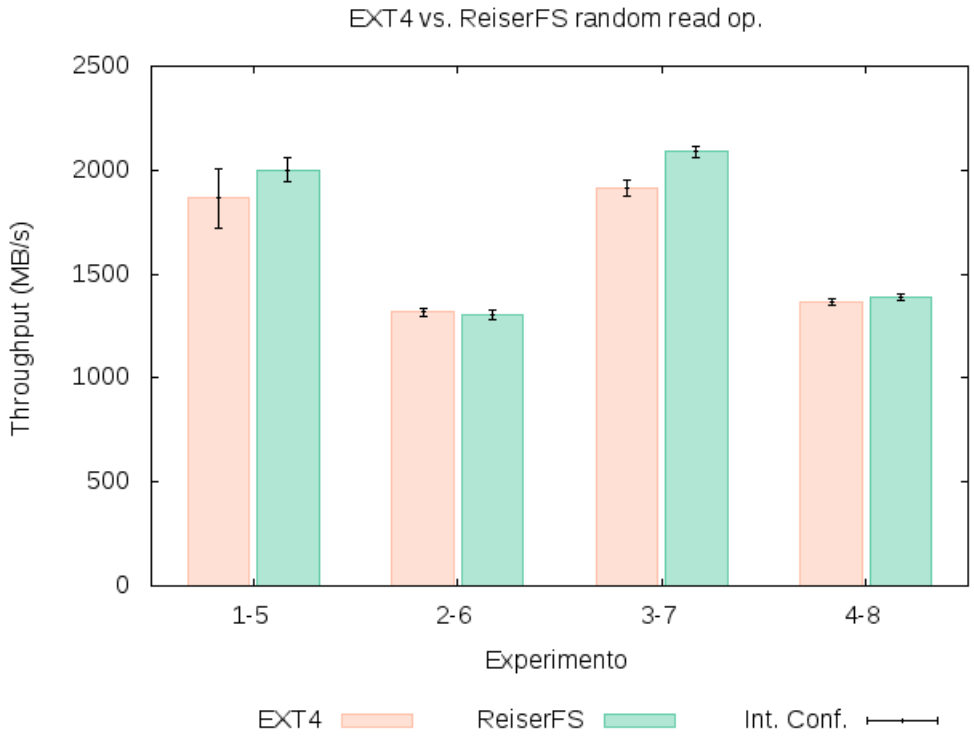

Figura 11: Gráfico comparando os experimentos da Etapa 2 na operação de Leitura aleatória.

escrita randômica. Esse fator faz com que o desempenho do ReiserFS seja visivelmente degradado quando arquivos grandes são utilizados.

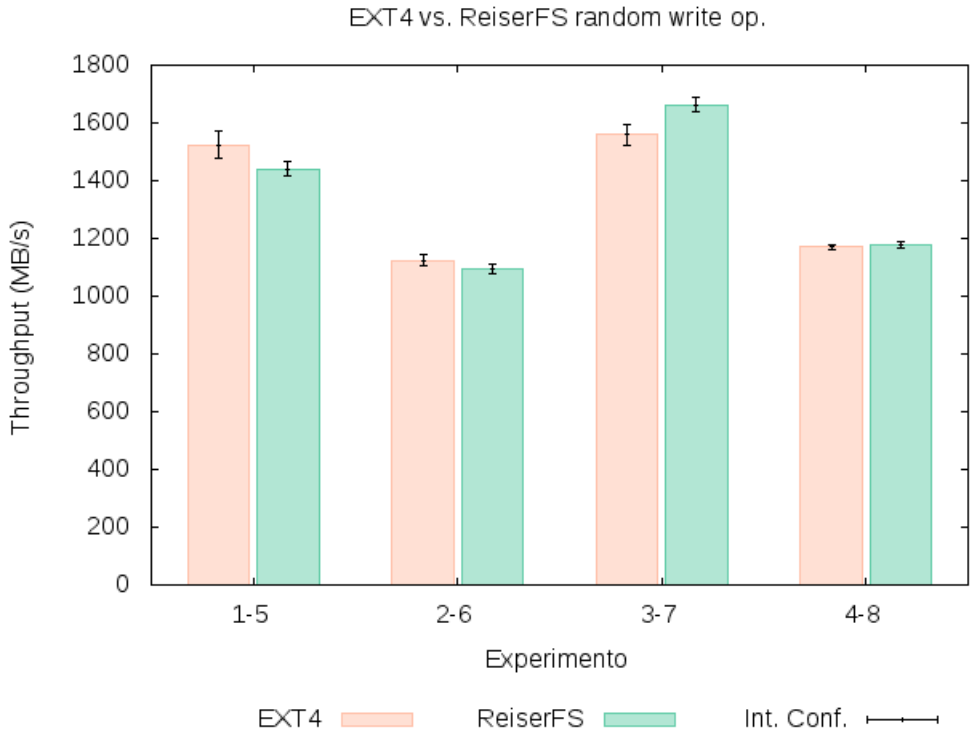

Figura 12: Gráfico comparando os experimentos da Etapa 2 na operação de Escrita aleatória.

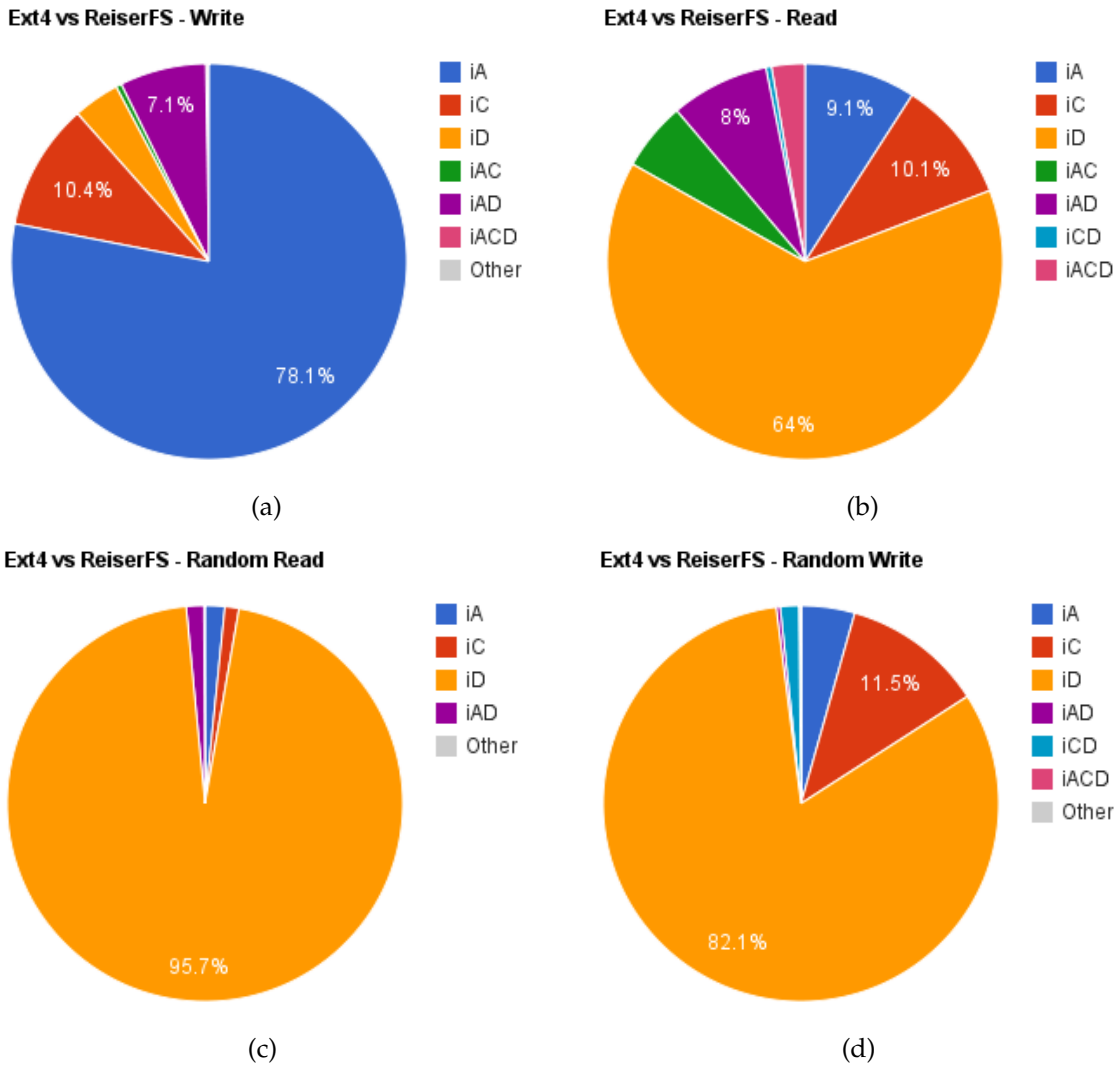

Figura 13: Gráficos com as influências dos fatores calculadas para as operações avaliadas na Etapa 2: (a) - Escrita, (b) - Leitura, (c) - Leitura aleatória e (d) - Escrita aleatória

5.3 Etapa 3: comparação entre XFS e ReiserFS

A terceira etapa do trabalho é a comparação entre os sistemas de arquivo XFS e ReiserFS. Note que nessa etapa não foi considerado a mudança de tamanho de blocos, já que o ReiserFS não possibilitava a sua utilização com 1 KB. As discussões sobre os resultados obtidos nos gráficos seguintes serão, portanto, considerados sempre com tamanho de bloco de 4 KB.

A partir da Figura 14, podemos concluir que o XFS desempenha a operação de escrita melhor que o ReiserFS. Sendo que os melhores resultados aparecem nos experimentos 2 e 4, ou seja, onde estava sendo utilizado arquivos grandes, de 64 MB. Note que a memória RAM não influencia muito esses resultados.

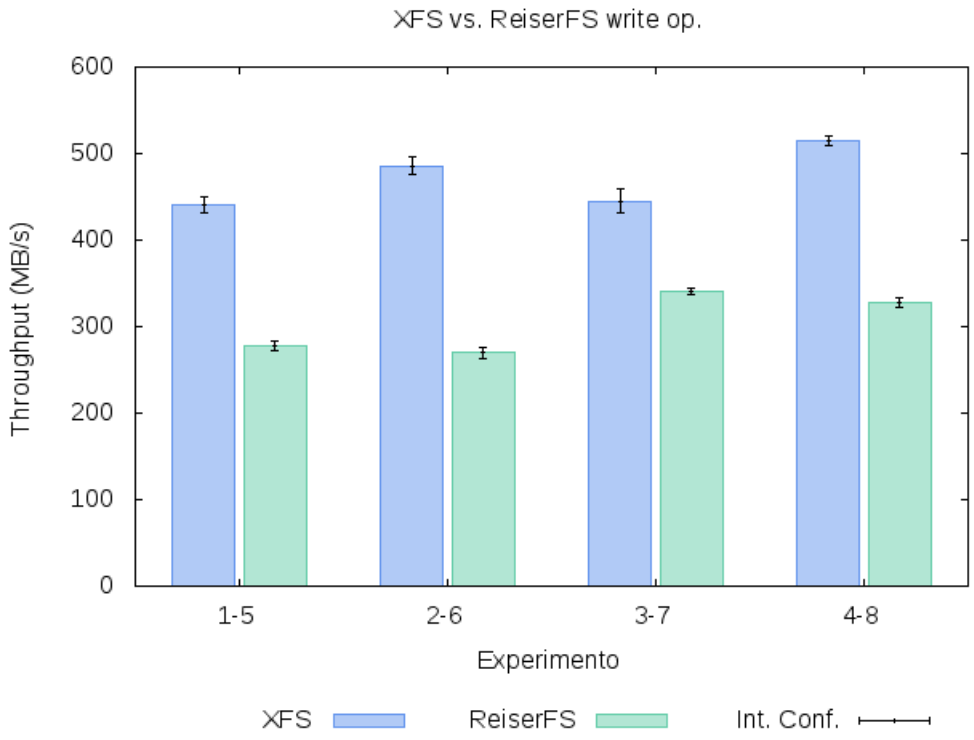

Figura 14: Gráfico comparando os experimentos da Etapa 3 na operação de Escrita.

Na operação de leitura, mostrada na Figura 15, há um ganho em todas as comparações do ReiserFS. Nota-se, principalmente, um ganho nos experimentos 5 e 7, onde foi considerado o teste com tamanho de arquivos pequenos, de 64 KB; e houve aumento do desempenho com o aumento da memória.

As operações de escrita randômica e leitura randômica geraram resultados parecidos, como é visto nas Figuras 17 e 16, respectivamente. Através desses gráficos é possível visualizar que o melhor resultado é obtido nos experimentos 5 e 7 pelo ReiserFS

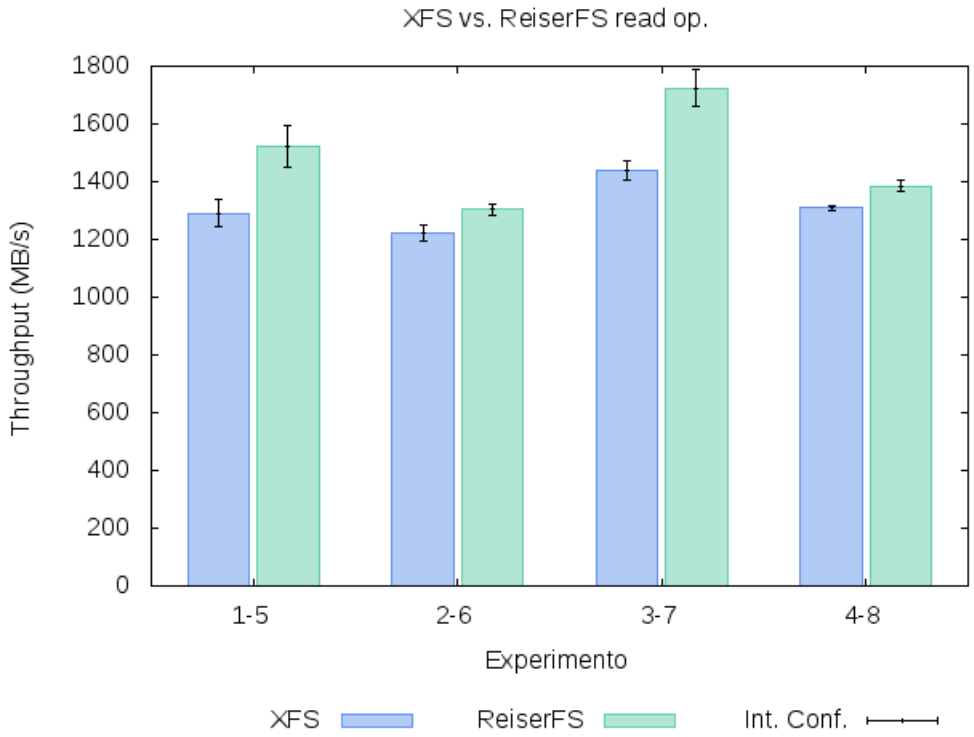

Figura 15: Gráfico comparando os experimentos da Etapa 3 na operação de Leitura.

em ambas operações. Estes experimentos utilizavam tamanhos de arquivos pequenos, de 64 KB e a memória principal variável. Note que a memória influencia muito nesses resultados, aumentando o desempenho.

Para o cálculo da influência dos fatores, seguem a tabela 8 com os dados e os gráficos de pizza na figura 5.3.

Fator	Write	Read	Random Read	Random Write
iA	88.16	31.25	8.89	4.39
iC	4.68	18.53	3.70	11.51
iD	1.70	38.86	83.58	82.09
iAC	1.50	0.16	0.31	0.02
iAD	3.69	8.77	3.05	0.33
iCD	0.09	2.19	0.24	1.51
iACD	0.18	0.25	0.23	0.15

Tabela 8: Tabela com os valores calculados de influência dos fatores para a Etapa 3 da avaliação de desempenho separados para o tipo de operação.

Nos gráficos gerados a partir dessa tabela (vide Figura 5.3), é possível notar que os próprios sistemas de arquivos têm influência quando comparados para a operação de escrita. Também podemos notar que o tamanho de arquivos é um fator de grande

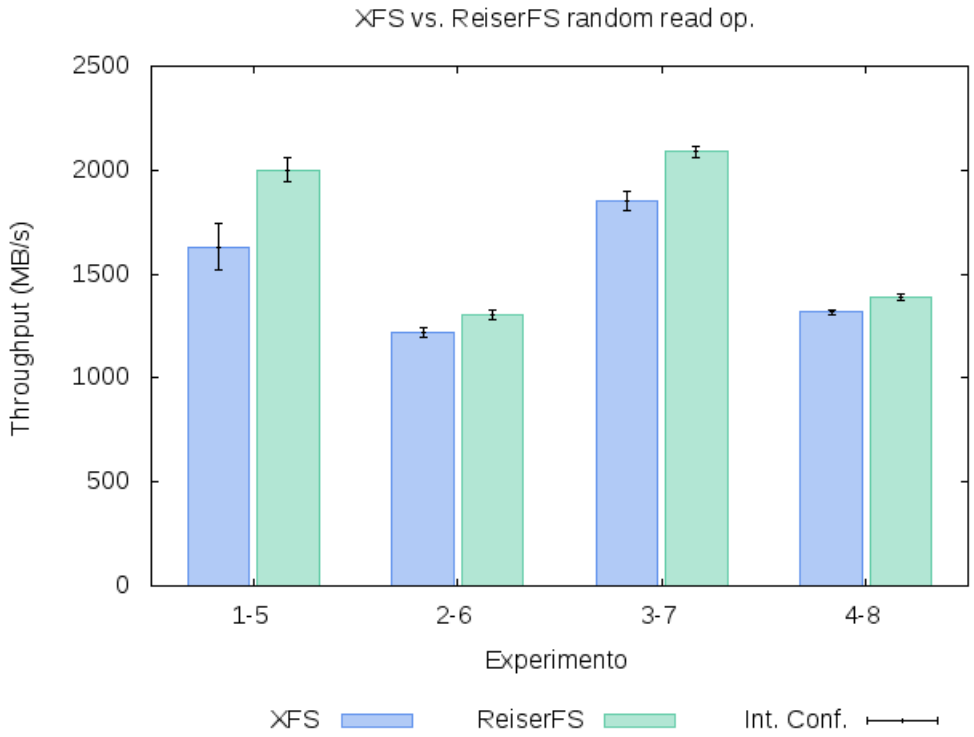

Figura 16: Gráfico comparando os experimentos da Etapa 3 na operação de Leitura aleatória.

influência quando comparamos XFS e ReiserFS para as operações de leitura, leitura randômica e escrita randômica. Arquivos maiores (64 MB) degradaram o desempenho de ambos os sistemas de arquivos. Por fim, a quantidade de memória principal é também um fator de importância para operação de leitura. Nesse caso, com o aumento da memória disponível há um ganho no desempenho de ambos os sistemas de arquivos.

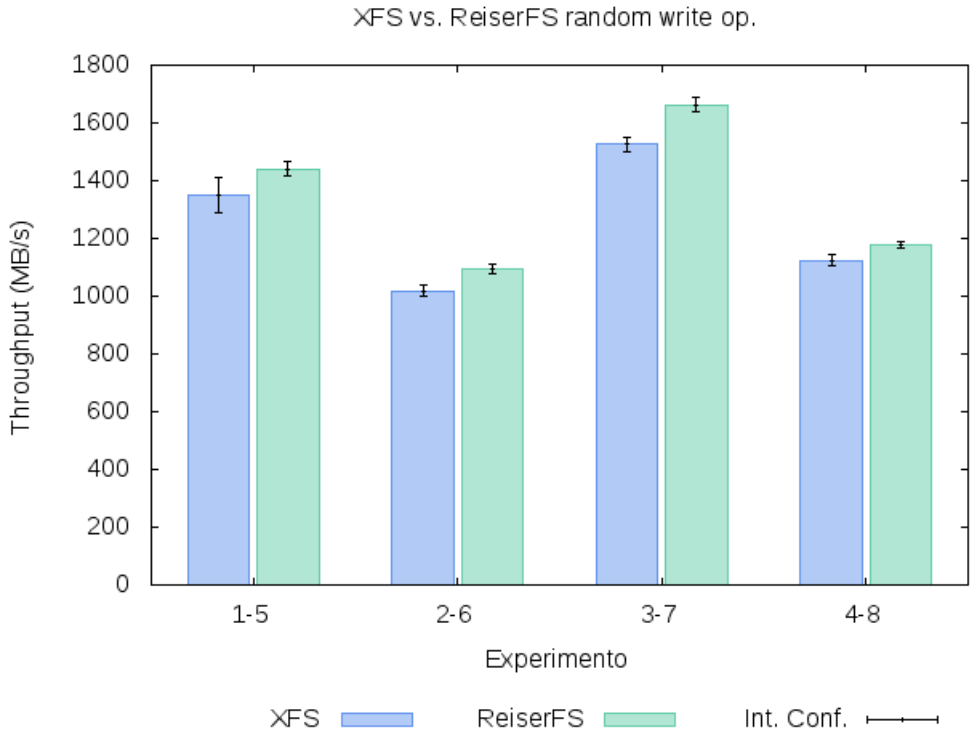

Figura 17: Gráfico comparando os experimentos da Etapa 3 na operação de Escrita aleatória.

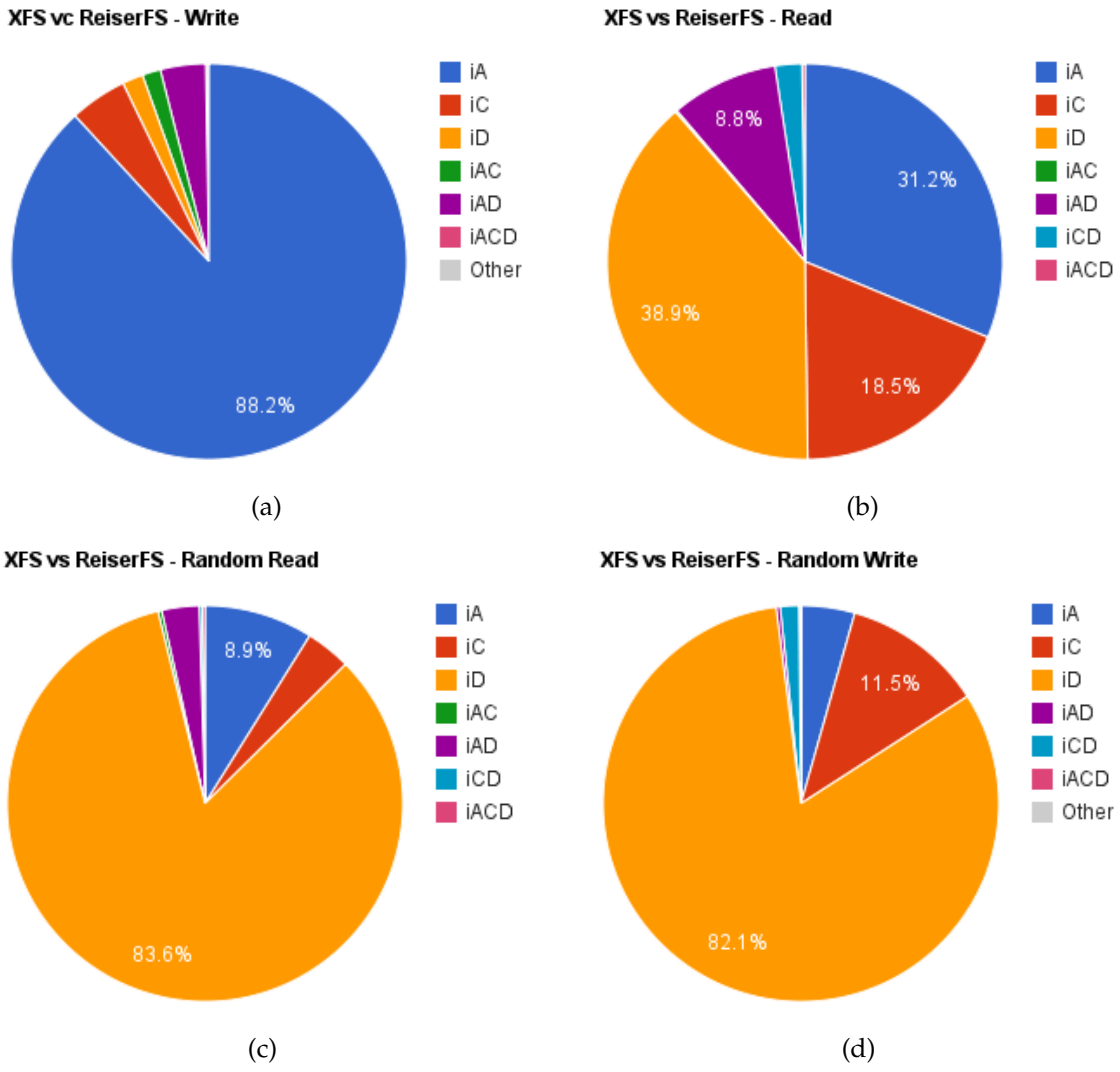

Figura 18: Gráficos com as influências dos fatores calculadas para as operações avaliadas na Etapa 3: (a) - Escrita, (b) - Leitura, (c) - Leitura aleatória e (d) - Escrita aleatória

6 Conclusões

Neste capítulo serão apresetadas as conclusões para as observações feitas a partir dos experimentos realizados e apresentados nos capítulos anteriores. Será feita uma análise para cada etapa de experimentos, seguida de uma conclusão final.

6.1 Etapa 1: comparação entre Ext4 e XFS

Na primeira etapa do trabalho os sistemas de arquivos Ext4 e XFS foram analisados em conjunto. A partir dessa análise, algumas concluões são possíveis:

- O sistema de arquivos XFS teve melhor ou igual desempenho para todos os experimentos realizados envolvendo blocos de tamanho 4 KB;
- Para demais operações XFS e Ext4 obtiveram desempenhos similares, com diferenças tênues para grande parte dos casos;
- O tamanho de blocos é um fator que possui grande influência quando comparamos o desempenho de Ext4 e XFS para a operação de escrita; e
- O tamanho de arquivos é um fator de grande influência quando comparamos Ext4 e XFS para as operações de leitura, leitura randômica e escrita randômica - há uma queda no desempenho quando utilizado arquivos grandes.

6.2 Etapa 2: comparação entre Ext4 e ReiserFS

Na segunda etapa do trabalho os sistemas de arquivos Ext4 e ReiserFS foram analisados em conjunto. A partir dessa análise, algumas conclusões são possíveis:

- O sistema de arquivos Ext4 possui melhor desempenho que o ReiserFS para todos os experimentos realizados com a operação de escrita;

- O sistema de arquivos ReiserFS apresentou em geral um melhor desempenho que o Ext4 em experimentos envolvendo a operação de leitura para arquivos menores (64 KB);
- Para as operações de leitura randômica e escrita randômica, o Ext4 e o ReiserFS obtiveram desempenhos similares, com diferenças tênues para grande parte dos casos;
- O tamanho de arquivos é um fator que possui grande influência quando comparamos o desempenho de Ext4 e ReiserFS para as operações de leitura, leitura randômica e escrita randômica — há queda de desempenho para arquivos maiores (64 MB);

6.3 Etapa 3

Na segunda etapa do trabalho os sistemas de arquivos XFS e ReiserFS foram analisados em conjunto. A partir dessa análise, algumas conclusões são possíveis:

- O sistema de arquivos XFS obteve melhor resultado na operação de escrita em todos os casos;
- O sistema de arquivos ReiserFS teve um desempenho melhor em geral nos casos da operação de leitura, principalmente utilizando arquivos pequenos, de 64 KB;
- As operações de escrita randômica e leitura randômica tiveram *throughput* parecidos em todos os casos, onde o ReiserFS tendo vantagem principalmente com arquivos de 64 KB;
- O tamanho de arquivo foi o fator que mais influenciou as operações de leitura, escrita randômica e leitura randômica.

6.4 Considerações finais

Podemos concluir com essa avaliação de desempenho:

- Os sistemas de arquivos tem um desempenho melhor utilizando blocos de 4 KB do que 1 KB;
- O XFS mostrou ser o melhor sistema de arquivos em escrita, principalmente com 2 GB de memória e arquivos de 64 MB;

- O ReiserFS realizou os maiores valores de leitura, principalmente de arquivos pequenos;
- O Ext4 fica em segundo em todas as operações, sendo um sistema de arquivos que realiza todos os tipos de operação com bom *throughput*.

Referências

[1] GRAPHICS, S. *XFS Filesystem Structure*. 2006. Disponível em: <http://xfs.org/docs/xfsdocs-xml-dev/XFS_Filesystem_Structure/tmp/en-US/html/index.html>.

[2] MATHUR, A. et al. The new ext4 filesystem: current status and future plans. In: *Proceedings of the Linux Symposium*. [s.n.], 2007. Disponível em: <http://kernel.org/doc/ols/2007/ols2007v2-pages-21-34.pdf>.

[3] SWEENEY, A. *Scalability in the XFS File System*. 2006. Disponível em: <http://oss.sgi.com/projects/xfs/papers/xfs_usenix/index.html>.

[4] CARD, R.; TS'O, T.; TWEEDIE, S. Design and implementation of the second extended filesystem. In: *Proceedings of the First Dutch International Symposium on Linux*. [s.n.], 1994. Disponível em: <http://e2fsprogs.sourceforge.net/ext2intro.html>.

[5] SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating System Concepts*. 8th. ed. [S.l.]: Wiley Publishing, 2008. ISBN 0470128720.

[6] BUCHHOLZ, F. *The structure of the Reiser file system*. , http://homes.cerias.purdue.edu/ florian/reiser/reiserfs.php Último acesso em 12/06/2011.

[7] IOZONE Filesystem Benchmark. 2006. Disponível em: <http://www.iozone.org/>.

ANEXO A – Scripts utilizados

A.1 Execução dos experimentos

```
#!/bin/bash

TEMP_DEVICE=/dev/sda9
TEMP_MOUNT=/tmp/test_fs
TEMP_FILE=/tmp/test_fs/temp_file
LOG_DIR=/home/fgrillo/Documents/sisop11/logs
IOZONE=/usr/bin/iozone
IOZONE_PARAMS="-R -r 4k -i 0 -i 1 -i 2 -i 8 -+u -f $TEMP_FILE"
NUM_EXPERIMENTS=10

function log {
    echo [ `date '+%d/%m/%Y %X'` ] $1
}

if [ ! -d $TEMP_MOUNT ]; then
    mkdir $TEMP_MOUNT
fi

if [ ! -d $LOG_DIR ]; then
    exit 1
fi

for fs in ext4 reiserfs xfs; do
    for bs in 4096 1024; do
        if [ $fs = "xfs" ]; then
            MKFS_PARAM="-f -b size=$bs"
        fi
        if [ $fs = "ext4" ]; then
            MKFS_PARAM="-b $bs"
        fi
        if [ $fs = "reiserfs" ]; then
            if [ $bs = 1024 ]; then
                continue
            fi
            MKFS_PARAM="-f -b $bs"
        fi

        log "Formatting $TEMP_DEVICE with fs:$fs bs:$bs"
        mkfs.$fs $MKFS_PARAM $TEMP_DEVICE &> /tmp/trace
        if [ $? != 0 ]; then
```

```
41              exit 1
42          fi
43
44          log "Mounting $TEMP_MOUNT"
45          mount $TEMP_DEVICE $TEMP_MOUNT &> /tmp/trace
46          if [ $? != 0 ]; then
47              exit 1
48          fi
49
50          for size in 64k 64m; do
51              for((i=0; i<$NUM_EXPERIMENTS; i++)); do
52                  log "Executing test for fs:$fs bs:$bs fsize:$size exp:$i"
53
54                  $IOZONE $IOZONE_PARAMS -s $size > $LOG_DIR/experiment_${fs}_${bs}_${size}_${↩
                        i}.txt
55                  if [ $? != 0 ]; then
56                      exit 1
57                  fi
58              done
59          done
60
61          log "Umounting $TEMP_MOUNT"
62          umount $TEMP_MOUNT &> /tmp/trace
63          if [ $? != 0 ]; then
64              exit 1
65          fi
66      done
67  done
```

Listagem A.1: Script bash utilizado para execução dos experimentos. Foi executado uma vez com a máquina com 2GB de memória RAM e outra com 1GB de memória RAM.

A.2 Extração dos dados do log

```
1  #!/bin/bash
2
3  LOG_DIR=/home/fgrillo/Documents/sisop11/logs
4
5  mp=$1
6  fs=$2
7  bs=$3
8  fsize=$4
9
10 echo 'KB reclen write rewrite read reread read write'
11 for file in `ls $LOG_DIR/${mp}GB/experiment_${fs}_${bs}_${fsize}_*`; do
12     grep -A1 -E '^\s*KB' $file | tail -1 | awk '{print $1,$2,$3,$4,$5,$6,$7 }'
13 done
```

Listagem A.2: Script bash utilizado para extrair os dados dos log dos experimentos para um formato separado por vírgulas que possibilitava a importação para uma

planilha do OpenOffice.

A.3 Geração dos gráficos de barra

```
set boxwidth 0.3
set xrange [0:9]
set yrange [0:]
set xlabel "Experimento"
set ylabel "Throughput (MB/s)"
set xtics ("1-9" 1, "2-10" 2, "3-11" 3, "4-12" 4, "5-13" 5, "6-14" 6, "7-15" 7, "8-16" 8)
set key below
set terminal png

set style fill solid 0.5


# EXT4 vs. XFS

set title "EXT4 vs. XFS write op."
set output "ext4_xfs/ext4_xfs_write.png"
plot "ext4.dat" using ($1-0.17):($5/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
     "xfs.dat" using ($1+0.17):($5/1024) title "XFS" with boxes lc rgb "#6495ED",\
     "ext4.dat" using ($1-0.17):($5/1024):($6/1024) title "Int. Conf." with yerrorbars lc rgb "black" pt 1 ps 0.3,\
     "xfs.dat" using ($1+0.17):($5/1024):($6/1024) notitle with yerrorbars lc rgb "black" pt 1 ps 0.3


set title "EXT4 vs. XFS read op."
set output "ext4_xfs/ext4_xfs_read.png"
plot "ext4.dat" using ($1-0.17):($7/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
     "xfs.dat" using ($1+0.17):($7/1024) title "XFS" with boxes lc rgb "#6495ED",\
     "ext4.dat" using ($1-0.17):($7/1024):($8/1024) title "Int. Conf." with yerrorbars lc rgb "black" pt 1 ps 0.3,\
     "xfs.dat" using ($1+0.17):($7/1024):($8/1024) notitle with yerrorbars lc rgb "black" pt 1 ps 0.3

set title "EXT4 vs. XFS random read op."
set output "ext4_xfs/ext4_xfs_randread.png"
plot "ext4.dat" using ($1-0.17):($9/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
     "xfs.dat" using ($1+0.17):($9/1024) title "XFS" with boxes lc rgb "#6495ED",\
     "ext4.dat" using ($1-0.17):($9/1024):($10/1024) title "Int. Conf." with yerrorbars lc rgb "black" pt 1 ps 0.3,\
     "xfs.dat" using ($1+0.17):($9/1024):($10/1024) notitle with yerrorbars lc rgb "black" pt 1 ps 0.3

set title "EXT4 vs. XFS random write op."
set output "ext4_xfs/ext4_xfs_randwrite.png"
plot "ext4.dat" using ($1-0.17):($11/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
     "xfs.dat" using ($1+0.17):($11/1024) title "XFS" with boxes lc rgb "#6495ED",\
     "ext4.dat" using ($1-0.17):($11/1024):($12/1024) title "Int. Conf." with yerrorbars lc rgb "black" pt 1 ps 0.3,\
     "xfs.dat" using ($1+0.17):($11/1024):($12/1024) notitle with yerrorbars lc rgb "black" pt 1 ps 0.3
```

```
43
44
45 set xtics ("1-5" 1, "2-6" 2, "3-7" 3, "4-8" 4)
46 # EXT4 vs. ReiserFS
47
48 set title "EXT4 vs. ReiserFS write op."
49 set output "ext4_reiser/ext4_reiserfs_write.png"
50 plot [0.5:4.5]\
51      "ext4.dat" using ($1-0.17):($5/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
52      "reiserfs.dat" using ($1+0.17):($5/1024) title "ReiserFS" with boxes lc rgb "#66CDAA",\
53      "ext4.dat" using ($1-0.17):($5/1024):($6/1024) title "Int. Conf." with yerrorbars lc ↩
         rgb "black" pt 1 ps 0.3,\
54      "reiserfs.dat" using ($1+0.17):($5/1024):($6/1024) notitle with yerrorbars lc rgb "↩
         black" pt 1 ps 0.3
55
56
57 set title "EXT4 vs. ReiserFS read op."
58 set output "ext4_reiser/ext4_reiserfs_read.png"
59 plot [0.5:4.5]\
60      "ext4.dat" using ($1-0.17):($7/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
61      "reiserfs.dat" using ($1+0.17):($7/1024) title "ReiserFS" with boxes lc rgb "#66CDAA",\
62      "ext4.dat" using ($1-0.17):($7/1024):($8/1024) title "Int. Conf." with yerrorbars lc ↩
         rgb "black" pt 1 ps 0.3,\
63      "reiserfs.dat" using ($1+0.17):($7/1024):($8/1024) notitle with yerrorbars lc rgb "↩
         black" pt 1 ps 0.3
64
65 set title "EXT4 vs. ReiserFS random read op."
66 set output "ext4_reiser/ext4_reiserfs_randread.png"
67 plot [0.5:4.5]\
68      "ext4.dat" using ($1-0.17):($9/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
69      "reiserfs.dat" using ($1+0.17):($9/1024) title "ReiserFS" with boxes lc rgb "#66CDAA",\
70      "ext4.dat" using ($1-0.17):($9/1024):($10/1024) title "Int. Conf." with yerrorbars lc ↩
         rgb "black" pt 1 ps 0.3,\
71      "reiserfs.dat" using ($1+0.17):($9/1024):($10/1024) notitle with yerrorbars lc rgb "↩
         black" pt 1 ps 0.3
72
73 set title "EXT4 vs. ReiserFS random write op."
74 set output "ext4_reiser/ext4_reiserfs_randwrite.png"
75 plot [0.5:4.5]\
76      "ext4.dat" using ($1-0.17):($11/1024) title "EXT4" with boxes lc rgb "#FFC1AA",\
77      "reiserfs.dat" using ($1+0.17):($11/1024) title "ReiserFS" with boxes lc rgb "#66CDAA"↩
         ,\
78      "ext4.dat" using ($1-0.17):($11/1024):($12/1024) title "Int. Conf." with yerrorbars lc ↩
         rgb "black" pt 1 ps 0.3,\
79      "reiserfs.dat" using ($1+0.17):($11/1024):($12/1024) notitle with yerrorbars lc rgb "↩
         black" pt 1 ps 0.3
80
81 # XFS vs. ReiserFS
82
83 set title "XFS vs. ReiserFS write op."
84 set output "xfs_reiser/xfs_reiserfs_write.png"
85 plot [0.5:4.5]\
86      "xfs.dat" using ($1-0.17):($5/1024) title "XFS" with boxes lc rgb "#6495ED",\
87      "reiserfs.dat" using ($1+0.17):($5/1024) title "ReiserFS" with boxes lc rgb "#66CDAA",\
88      "xfs.dat" using ($1-0.17):($5/1024):($6/1024) title "Int. Conf." with yerrorbars lc rgb↩
          "black" pt 1 ps 0.3,\
89      "reiserfs.dat" using ($1+0.17):($5/1024):($6/1024) notitle with yerrorbars lc rgb "↩
         black" pt 1 ps 0.3
90
```

```
91  set title "XFS vs. ReiserFS read op."
92  set output "xfs_reiser/xfs_reiserfs_read.png"
93  plot [0.5:4.5]\
94       "xfs.dat" using ($1-0.17):($7/1024) title "XFS" with boxes lc rgb "#6495ED",\
95       "reiserfs.dat" using ($1+0.17):($7/1024) title "ReiserFS" with boxes lc rgb "#66CDAA",\
96       "xfs.dat" using ($1-0.17):($7/1024):($8/1024) title "Int. Conf." with yerrorbars lc rgb↩
             "black" pt 1 ps 0.3,\
97       "reiserfs.dat" using ($1+0.17):($7/1024):($8/1024) notitle with yerrorbars lc rgb "↩
             black" pt 1 ps 0.3
98
99  set title "XFS vs. ReiserFS random read op."
100 set output "xfs_reiser/xfs_reiserfs_randread.png"
101 plot [0.5:4.5]\
102      "xfs.dat" using ($1-0.17):($9/1024) title "XFS" with boxes lc rgb "#6495ED",\
103      "reiserfs.dat" using ($1+0.17):($9/1024) title "ReiserFS" with boxes lc rgb "#66CDAA",\
104      "xfs.dat" using ($1-0.17):($9/1024):($10/1024) title "Int. Conf." with yerrorbars lc ↩
             rgb "black" pt 1 ps 0.3,\
105      "reiserfs.dat" using ($1+0.17):($9/1024):($10/1024) notitle with yerrorbars lc rgb "↩
             black" pt 1 ps 0.3
106
107 set title "XFS vs. ReiserFS random write op."
108 set output "xfs_reiser/xfs_reiserfs_randwrite.png"
109 plot [0.5:4.5]\
110      "xfs.dat" using ($1-0.17):($11/1024) title "XFS" with boxes lc rgb "#6495ED",\
111      "reiserfs.dat" using ($1+0.17):($11/1024) title "ReiserFS" with boxes lc rgb "#66CDAA"↩
             ,\
112      "xfs.dat" using ($1-0.17):($11/1024):($12/1024) title "Int. Conf." with yerrorbars lc ↩
             rgb "black" pt 1 ps 0.3,\
113      "reiserfs.dat" using ($1+0.17):($11/1024):($12/1024) notitle with yerrorbars lc rgb "↩
             black" pt 1 ps 0.3
```

Listagem A.3: Script do Gnuplot utilizado gerar os gráficos de barra para comparação dos experimentos.