Análise de desempenho do sistema de arquivos Lustre sob padrões de acesso a dados exclusivos

Francieli Zanon Boito Rodrigo Virote Kassick
Philippe O. A. Navaux
Universidade Federal do Rio Grande do Sul
Instituto de Informática
Porto Alegre, RS, Brasil
{francieli.zanon, rvkassick, navaux}@inf.ufrgs.br

Resumo

Aplicações que executam em ambientes de cluster frequentemente geram grandes quantidades de dados, que podem precisar ser acessados de forma eficiente por todos os nós envolvidos na computação. Sistemas de arquivos distribuídos (SADs) constituem uma solução natural nesse caso. Dependendo das suas opções de projeto, esses sistemas podem apresentar variados comportamentos sob diferentes padrões de acesso. Assim, estudar como se comporta o desempenho de um SAD sob padrões de acesso verificados na prática é uma tarefa importante, pois fornece ferramentas para que as aplicações possam adequar as suas operações de E/S para tirar melhor proveito do sistema utilizado. Um desses padrões comuns em aplicações científicas é o acesso pelos nós a dados exclusivos. Duas formas de fazê-lo são empregar um arquivo por processo, ou segmentos de um arquivo compartilhado. Esse artigo apresenta um estudo sobre o comportamento do sistema de arquivos Lustre nessas situações, a fim de determinar a que deve ser preferida em cada caso. Para realizar o estudo, foram realizados testes que simulam os padrões de acesso. Os resultados permitiram concluir que, para operações de escrita com números grandes de clientes, a abordagem de arquivos múltiplos é a melhor. No entanto, para outras situações, ela se mostra equivalente e até pior que a de arquivo compartilhado.

1. Introdução

Com a disparidade no crescimento da velocidade de processadores e dispositivos de entrada e saída, e com aplicações de alto desempenho gerando grandes quantidades de dados, E/S se tornou um ponto crucial no desempenho das aplicações. Nessas aplicações, os dados podem precisar ser acessados em todos os nós envolvidos na computação. Para permitir isso, uma solução bastante empregada é o uso de Sistemas de Arquivos Distribuídos (SADs).

Dependendo de suas opções de projeto, sistemas de arquivos podem apresentar comportamentos distintos para padrões de acesso diferentes. Assim, um *benchmark* simples pode não ser o bastante para decidir entre dois sistemas, já que ele simula apenas uma situação. É importante estudar o comportamento do desempenho de sistemas sob diferentes formas de acesso para ter-se ferramentas para uma melhor comparação.

Além de comparar sistemas, esse estudo é útil para que as aplicações, conhecendo as características do SAD a ser utilizado, se adaptem para tirar melhor proveito das otimizações providas por ele. Da mesma forma, sabendo as conseqüências no desempenho causadas pelas diferentes opções de projeto tomadas, os projetistas de sistemas de arquivos distribuídos podem guiar o seu trabalho para beneficiar aplicações alvo.

O objetivo desse artigo é avaliar o desempenho do sistema de arquivos Lustre [4, 16, 2] sob uma situação bastante comum em aplicações científicas: o acesso pelos clientes a dados exclusivos. Nessa situação, cada nodo envolvido na computação escreverá ou lerá os seus próprios dados em um determinado momento. Para fazê-lo, cada um deles pode ter o seu próprio arquivo, ou podem ser delegados segmentos de um arquivo compartilhado.

Quando utilizando um arquivo compartilhado, há diversas formas de distribui-lo pelos processos. Pode ser atribuida apenas uma porção contígua a cada nodo, ou diversas porções esparsas dentro do arquivo. Para determinar o comportamento do Lustre sob todas essas formas de acesso, foram conduzidos testes que as simulam. A comparação entre os resultados permite a caracterização dos padrões que devem ser preferidos quando utilizando o Lustre.

O resto do artigo está organizado como segue: a Seção 2 apresenta alguns trabalhos relacionados. A Seção 3 fala sobre o sistema de arquivos Lustre. A Seção 4 descreve como os padrões de acesso foram simulados através de testes, e os resultados são trazidos na Seção 5. A seguir, conclusões e trabalhos futuros são apresentados.

2 Trabalhos Relacionados

Operações de entrada e saída têm sido um importante limitante do desempenho das aplicações por bastante tempo. Desde a década de 90, muitos trabalhos exploraram a caracterização dos padrões de acesso de aplicações paralelas para adaptação dos sistemas de armazenamento com o objetivo de prover melhor desempenho. [12, 15, 17]

Em Kotz e Ellis [12], para avaliar o desempenho de técnicas de cache para sistemas de arquivos paralelos, é apresentada uma organização de padrões de acesso a arquivos em classes. Essa classificação inspirou a utilizada para guiar esse trabalho. A classificação é apresentada em detalhes em [10] e [3].

Em Wang *et at*. [17], foram utilizados traces para a caracterização das operações de E/S de aplicações científicas. Através de experimentos utilizando o sistema Lustre, foi apontado que a utilização de arquivos próprios aos nós apresenta desempenho cerca de 5 vezes melhor do que a utilização de um arquivo compartilhado.

As situações testadas são semelhantes às simuladas nesse trabalho. No entanto, nos experimentos desse trabalho relacionado não foi utilizado MPI-IO, menos configurações de distribuição dos acessos pelos clientes foram exploradas e o número de servidores que recebem fatias de cada arquivo utilizado foi bastante grande. Apesar dos experimentos realizados, o artigo não foca na avaliação do Lustre, e sim na caracterização das aplicações que o utilizam.

Muitos trabalhos mais recentes estudam o comportamento do desempenho do Lustre, especialmente quando utilizado juntamente com MPI-IO. Em Liao *et al.* [11], é mostrado o dano causado ao desempenho quando as operações de E/S não estão alinhadas com o *stripe* quando utilizando chamadas MPI-IO independentes com o Lustre.

Em Larkin e Fahey [14], é analisado o comportamento do Lustre no Cray XT3/XT4. Baseando-se nessa análise, foram apresentadas algumas características desejáveis às aplicações usuárias do sistema de arquivos nessa plataforma. Entre elas, está o número de processos realizando E/S igual ou maior que o número de servidores, porém não muito grande. São estudados apenas acessos contíguos, ao contrário do aqui apresentado.

Dickens e Logan [7] mostraram que, ao contrário do que é amplamente aceito, acessos a grandes áreas contíguas de arquivos podem não constituir o padrão de melhor desempenho com MPI-IO e Lustre. Isso acontece porque, devido ao stripe, essa área pode estar distribuída entre todos os servidores de dados, dando aos clientes maior sobrecusto de comunicação. Em [8], os mesmos autores propõem uma biblioteca que reorganiza os acessos de forma a fazer cada cliente acessar um número menor de servidores, obtendo melhor desempenho. Nesse último artigo relacionado exposto, estudou-se apenas o número de servidores acessados, ao contrário desse trabalho. Aqui, a ênfase dada é na distribuição dos acessos dos clientes nos arquivos.

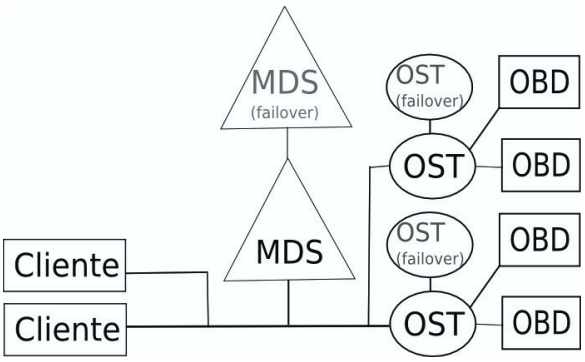

Figura 1. Arquitetura do Lustre File System.

Com esse trabalho, intenciona-se investigar o comportamento do Lustre sob os padrões de acesso em que dados exclusivos são acessados. O objetivo não é investigar as causas desse comportamento para propor melhorias no sistema, mas determinar os padrões mais adequados para diferentes situações, permitindo que as aplicações se adaptem ao Lustre a fim de obter melhor desempenho.

3. Lustre File System

O Lustre [4, 16, 2] é um sistema de arquivos distribuído e paralelo construído com o objetivo de prover altos desempenho e escalabilidade ao servir clusters de dezenas de milhares de nós. A sua arquitetura, ilustrada na Figura 1, é composta por um servidor de metadados centralizado (MDS) e diversos servidores de armazenamento baseados em objetos (OSTs, *Object Storage Targets*).

O metasservidor suporta todas as operações no espaço de nomes do sistema de arquivos, como *lookups* e criações. Os servidores de dados são responsáveis pelas operações sobre os dados propriamente ditos e pela interação com os dispositivos de armazenamento, os *Object-Based Disks* (OBDs).

Os OSTs são também responsáveis por gerenciar travamentos (*lockings*) dos dados que armazena para manter a coerência em acessos concorrentes. A forma com que os arquivos são separados em objetos e a distribuição deles são configuráveis pelo usuário via linha de comando, ou durante a criação do arquivo quando da utilização de APIs como

MPI-IO.

Apesar de serem chamados *Object-Based Disks*, os dispositivos de armazenamento não precisam ser discos. Isso ocorre porque a interface deles com o OST é feita através de um *device driver* que mascara a identidade do OBD sendo utilizado. É permitido, então, que novas tecnologias de armazenamento sejam aplicadas, como, por exemplo, discos externos multiporta e sistemas de arquivos jornalizados para Linux (ext3, RaiserFS, etc).

Todos os servidores podem ter cópias sincronizadas que tomarão os seus lugares em caso de falha, sem interrupção no serviço oferecido. O metasservidor redundante substitui o original sem prejuízos, pois ambos mantém um registro transacional sincronizado de todas as alterações sobre os metadados. Um serviço chamado MGS é responsável por prover informações sobre servidores substitutos, além de auxiliar na configuração e manutenção do SAD.

Os OSTs não possuem cache. No entanto, ela pode estar presente no OBD empregado. O cliente possui cache e faz *read-ahead*. Isso significa que mais dados do que a aplicação está acessando num determinado momento são efetivamente requisitados ao servidor e armazenados localmente.

O sistema oferece a sua interface aos clientes em Linux através de uma camada de sistema de arquivos virtual. Essa camada é responsável por encaminhar as operações sobre arquivos remotos ao Lustre, e sobre arquivos locais ao sistema local.

A comunicação no Lustre é feita através da *Lustre Networking* (LNET), que provê suporte para diferentes infraestruturas de rede através de drivers conectáveis, chamados *Lustre Network Drivers* (LNDs). Isso torna o Lustre bastante adaptável a diferentes tecnologias e arquiteturas.

O Lustre foi escolhido para esse trabalho devido a sua grande visibilidade: 15 dos top-30 supercomputadores utilizam esse sistema [2]. Sendo alvo de diversas pesquisas, ele não foi baseado em nenhum sistema anterior. Assim, o Lustre não herda deficiências de sistemas que não foram construídos objetivando o alto desempenho.

4 Simulando padrões de acesso

Entre as aplicações científicas, são comuns situações onde os nodos envolvidos na computação escrevem ou lêem dados exclusivos. Esses dados podem, por exemplo, ser resultados de cálculos efetuados, que ficarão disponíveis para uma próxima fase de processamento. Esse padrão também é comum na geração de *checkpoints*, onde cada nó gravará o seu estado em um determinado momento para que a execução possa ser retomada em caso de falha.

Duas maneiras de os nós terem dados exclusivos no SAD são armazenar esses dados em arquivos próprios ou em segmentos de um arquivo compartilhado. Um arquivo compartilhado dá ao programador o trabalho de gerenciar a distribuição de segmentos. Por outro lado, uma grande quantidade de arquivos independentes pode ser penosa numa próxima fase de processamento sobre esses dados. Nesse trabalho, essas duas situações são representadas pelas classes de testes SFSA (*Single File, Segmented Access*) e MFWA (*Multiple Files, Whole Access*).

A classe MFWA é exemplificada por aplicações como sPPM [1] (algoritmo hidrodinâmico), ESCAT [18] (simulação de colisões entre moléculas e elétrons), MESSKIT e NWChem [15] (cálculo de densidade de elétrons). Em todos esses exemplos, os nós guardam resultados em arquivos próprios que serão acessados em uma próxima fase.

Utilizando o método de arquivo único (SFSA), pode-se atribuir diferentes números de segmentos aos processos. Assim, os acessos podem ser em áreas contíguas ou em porções esparsas dentro do arquivo. Por exemplo, considerando uma aplicação em que cada processo opera sobre algumas linhas (não em sequência) de uma matriz, armazenar a matriz ordenadamente num arquivo geraria acessos esparsos a ele. Alternativamente, cada processo poderia armazenar todas as suas linhas correspondentes em uma porção pré-determinada do arquivo. Essa abordagem traria custos a acessos posteriores a essa matriz caso ela precisasse ser reordenada.

A aplicação mpiBLAST [6] (busca de sequências genômicas) é um exemplo da classe SFSA onde cada escravo acessa um segmento próprio (delegado pelo mestre) de um grande arquivo. Já na aplicação Flash [9] (algoritmo da Astrofísica), em intervalos periódicos de tempo, todos os nós escrevem os valores de suas variáveis em um arquivo de *checkpoint*. Essas escritas são feitas de forma bastante granular, pois cada segmento do arquivo armazena os valores de apenas uma variável em todos os processos.

4.1 Metodologia

Para realizar os testes, foi utilizada a ferramenta *MPI-IO Test* (versão 21) [13], que emprega chamadas da API MPI-IO [5]. O teste é descrito através de parâmetros passados à ferramenta, que determinam, entre outras coisas:

- a operação a ser realizada: escrita, leitura ou ambas;
- o número de arquivos envolvidos: arquivo único ou um por processo;
- barreiras antes ou depois de determinadas operações;
- como deve ser a distribuição do arquivo entre os processos: *strided* ou *non-strided*;
- o número de segmentos e o tamanho de cada um deles acessados em cada cliente.

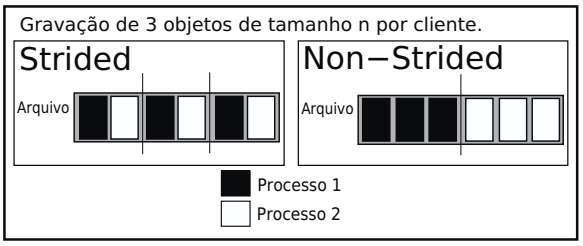

Figura 2. Distribuição de um arquivo em dois processos que acessam 3 objetos de tamanho n.

A Figura 2 ilustra os dois modos de distribuição do arquivo compartilhado entre os nós. Com a opção *non-strided*, cada cliente possui uma área contígua para realizar suas operações. Já com a opção *strided*, todos os nós acessam uma porção do arquivo, e então passam para a próxima porção. Assim, para os testes, dividiu-se a classe SFSA (*Single File, Segmented Access*) em duas subclasses de testes: SFSA *strided* e SFSA *non-strided*.

Os testes realizados possuem duas fases - leitura e escrita - executadas separadamente. Entre elas, os clientes são reiniciados para evitar o efeito da cache. Foram determinadas barreiras após as operações de abertura de arquivo e antes do fechamento dos mesmos.

A configuração do Lustre utilizada foi 4 servidores de dados, com *striping* em blocos de 64KB circular começando em servidor aleatório. Nos OSTs, o meio de armazenamento é o sistema de arquivos local ext3. A quantidade de dados acessada por processo é 2GB. Não foram utilizados servidores *failover*, não havendo, portanto, replicação dos dados.

Os testes foram executados no cluster Helios (Grid 5000). Ele possui 56 nodos biprocessados com AMD Opteron 2.2GHz, 4GB de memória RAM e rede Gigabit Ethernet. Os resultados possuem uma confiança de 90% e erro máximo tolerado de 10%. Para as duas classes (SFSA e MFWA), os testes principais são:

- Variando o número de clientes: o número de segmentos e seu tamanho são fixados (32 objetos de 64MB), enquanto o número de clientes acessando concorrentemente o SAD cresce. A quantidade de dados acessada por cada cliente é fixa, ou seja, o volume total de dados cresce junto com o número de clientes.

- Variando número de clientes com segmentos pequenos: idêntico ao teste anterior, porém utilizando 32K objetos de tamanho 64KB (mesmo tamanho do *stripe* utilizado). Intenciona-se verificar o comportamento do desempenho em uma situação de acessos menores.

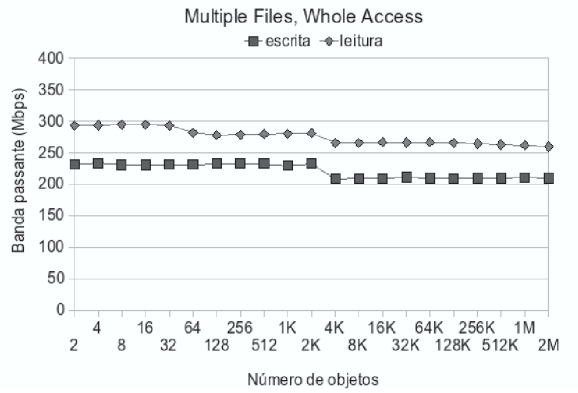

Figura 3. Teste da classe MFWA variando número de segmentos acessados por cada um dos 40 clientes.

- Variando o número de objetos: o número de clientes é fixado em 40, enquanto o número de segmentos cresce. O tamanho de cada objeto é adaptado para manter a quantidade de dados acessada de 2GB por processo.

5 Resultados

Essa seção apresenta os resultados obtidos para os testes descritos na seção anterior, separados pelo tipo de teste (variando a granularidade do acesso ou o número de clientes). Além dos resultados, são apresentadas algumas discussões sobre eles. A próxima seção traz as conclusões obtidas.

5.1 Granularidade do acesso

Os resultados apresentados nessa subseção correspondem ao teste em que o número de clientes é mantido em 40, e o número de segmentos acessados por cliente é variado. O tamanho de cada segmento é adaptado para manter-se a quantidade de dados acessada em 2GB por cliente.

A Figura 3 mostra os resultados obtidos para esse teste com múltiplos arquivos (classe MFWA). As variações do desempenho da escrita estão dentro do intervalo do erro tolerado, que para esse teste foi de 5%. A pequena queda observada nessa linha não é, então, significativa. As operações de leitura apresentam uma discreta queda de desempenho (cerca de 10%) com o aumento da granularidade do acesso.

Comportamento semelhante é observado no gráfico da Figura 4, onde são mostrados os resultados do teste utilizando-se um arquivo compartilhado (classe SFSA) e delegando apenas uma porção contígua do mesmo para cada processo (opção *non-strided*). Esse teste também possui

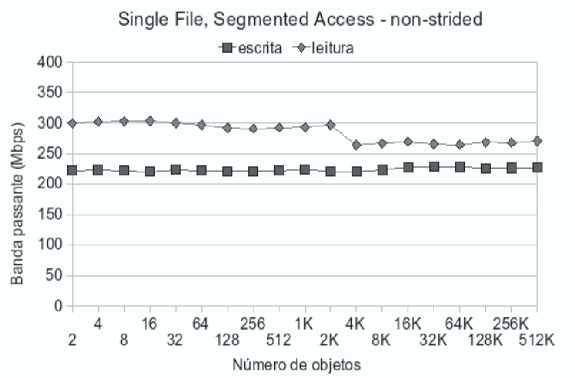

Figura 4. Teste da classe SFSA variando número de segmentos acessados por cada um dos 40 clientes com a opção *non-strided*.

erro máximo de 5%. A pequena queda observada no desempenho da leitura foi de aproximadamente 9%.

A Figura 5 mostra os resultados dos testes variando o número de segmentos para a classe SFSA com as duas opções (*non-strided* e *strided*). Visivelmente, dedicar apenas uma porção contígua do arquivo compartilhado possui melhor desempenho do que mais porções esparsas. Quanto mais porções, pior o desempenho. A leitura sofre mais com a granularização, já que o *read-ahead* realizado nos clientes deixa de ser vantajoso.

Os resultados mostrados indicam que, para acessos a áreas contíguas de dados, a granularidade desses acessos possui um pequeno impacto, e apenas nas operações de leitura. No entanto, banda significativamente menor é obtida quando os clientes realizam acessos a porções esparsas de um arquivo compartilhado.

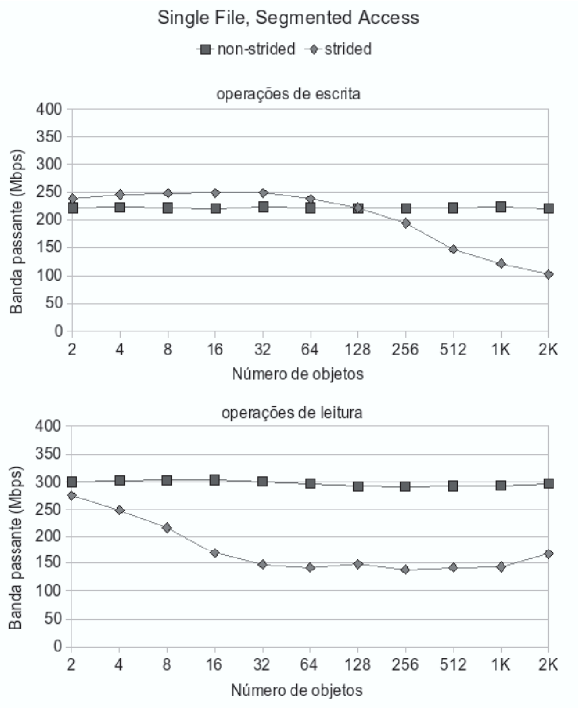

Figura 5. Testes da classe SFSA variando número de segmentos acessados por cada um dos 40 clientes.

5.2 Número de clientes envolvidos

Nos testes apresentados a seguir, o número de clientes acessando concorrentemente o sistema de arquivos é variado. Cada cliente acessa 2GB, então quanto mais clientes, mais dados. Para cada padrão de acesso, foram realizados testes com uma configuração que usa grandes segmentos para os acessos (32 objetos de 64MB cada) e uma com segmentos menores (32K objetos de 64KB).

Os resultados para a abordagem de arquivos múltiplos podem ser vistos no gráfico da Figura 6, onde são comparados os resultados com os dois tamanhos de segmentos. Nenhuma das diferenças entre as bandas para os dois tamanhos é significativa, e ambos os testes não apresentaram degradação do desempenho com o aumento da carga.

O gráfico da Figura 7 mostra os resultados obtidos para a classe SFSA com a opção *non-strided*. Novamente, ne-

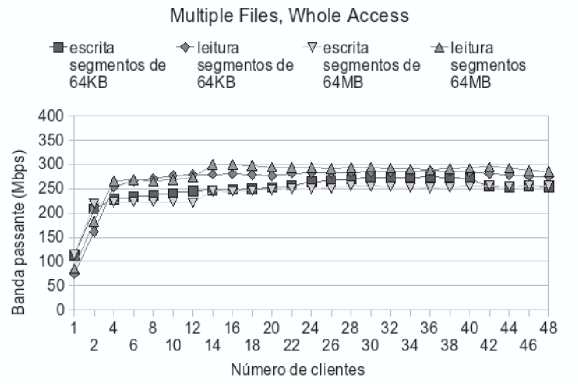

Figura 6. Testes da classe MFWA variando número de clientes.

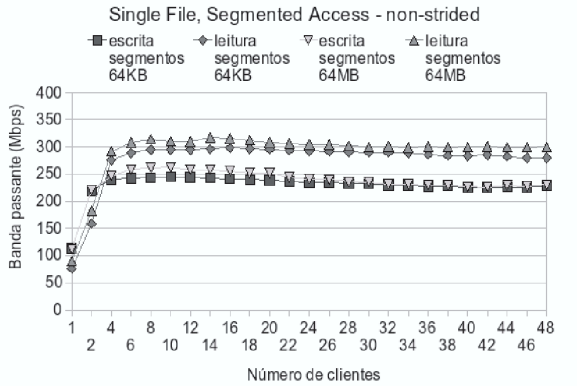

Figura 7. Testes da classe SFSA variando número de clientes com a opção *non-strided*.

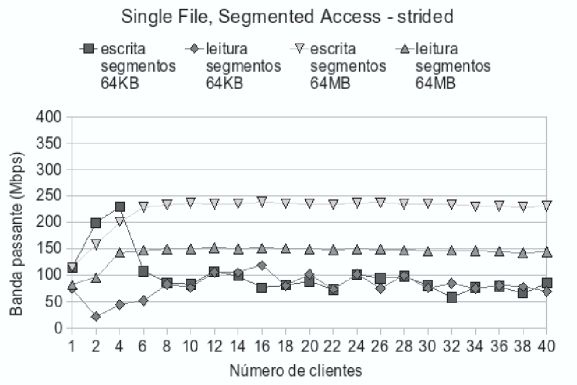

Figura 8. Testes da classe SFSA variando número de clientes com a opção *strided*.

nhuma das diferenças entre as bandas para os dois tamanhos é significativa. Os testes mostraram uma degradação no desempenho das operações de leitura com o aumento do número de clientes (11% no caso dos segmentos de 64MB), indicando que a abordagem não possui boa escalabilidade.

Um comportamento bastante diferente é observado nos resultados do teste da classe SFSA com a opção *strided*, representados no gráfico da Figura 8. Com grandes segmentos, o desempenho estabiliza rapidamente e não sofre degradação com o aumento do número de clientes. Utilizando segmentos de 64KB, a banda obtida é consideravelmente menor.

Com os segmentos de 64KB, até 4 clientes, como o tamanho do segmento é o mesmo tamanho do *stripe* realizado, cada cliente é atendido exclusivamente por um servidor. Como nem todos os OSTs chegam a ser utilizados, o

Tabela 1. Banda passante (Mbps) das operações de escrita com segmentos de 64MB.

Clientes	SFSA non-strided	MFWA	diferença
4	246,83	221,95	$\approx 10\%$
6	258,66	222,88	$\approx 13\%$
8	260,87	222,61	$\approx 14\%$
10	262,61	223,04	$\approx 15\%$
12	258,03	221,3	$\approx 14\%$
38	229,19	253,08	$\approx 10\%$
40	227,09	254,79	$\approx 12\%$
42	226,3	256,08	$\approx 13\%$
44	229,74	254,42	$\approx 10\%$
46	227,3	255,59	$\approx 12\%$
48	230,35	255,41	$\approx 10\%$

Tabela 2. Banda passante (Mbps) das operações de leitura com segmentos de 64MB.

Clientes	SFSA non-strided	MFWA	diferença
6	308,12	268,02	$\approx 13\%$
8	314,09	265,52	$\approx 15\%$
10	310,49	268,55	$\approx 14\%$
12	310,12	272,88	$\approx 12\%$

desempenho da leitura é pior do que para mais clientes, pois não aproveita-se o paralelismo oferecido. Após estabilizarem, os desempenhos de ambas as fases (leitura e escrita) apresentam picos em números de processos múltiplos de 4. Isso ocorre porque, com números múltiplos de 4, cada nodo realiza seus acessos sempre ao mesmo servidor. Com os outros números, os acessos de cada cliente serão alternados entre dois servidores. Conforme estudado no trabalho relacionado [7], acessos a menos servidores possuem desempenho melhor. No entanto, conforme o número de clientes cresce, esse efeito perde a sua importância, e os picos diluem-se.

As Tabelas 1, 2 e 3 comparam alguns dos resultados das classes SFSA (com a opção *non-strided*) e MFWA variando o número de clientes com os dois tamanhos de segmentos testados. As tabelas mostram apenas os resultados para os números de clientes onde a diferença entre as duas classes foi superior a 10% (erro permitido no teste). Portanto, os resultados para as operações de leitura com segmentos de 64KB não são comparados porque não possuem diferenças consideradas significativas.

Tabela 3. Banda passante (Mbps) das operações de escrita com segmentos de 64KB.

Clientes	SFSA non-strided	MFWA	diferença
24	234,22	264,12	$\approx 13\%$
26	234,71	265,84	$\approx 13\%$
28	232,46	269,81	$\approx 16\%$
30	232,39	272,75	$\approx 17\%$
32	229,44	272,31	$\approx 19\%$
34	229,48	273	$\approx 19\%$
36	227,17	273,44	$\approx 20\%$
38	228,12	273,13	$\approx 20\%$
40	225,31	270,74	$\approx 20\%$
42	225,61	254,23	$\approx 13\%$
44	226,53	253,19	$\approx 12\%$
46	225,62	254,65	$\approx 13\%$
48	229,41	253,56	$\approx 11\%$

As Tabelas 1 e 2 mostram que, quando os acessos são realizados utilizando grandes segmentos (64MB), a abordagem de arquivo único apresenta desempenho em média 13% melhor para números pequenos de clientes (até 12, ou seja, 3 vezes o número de servidores de dados), para leitura e escrita. Para as operações de leitura (Tabela 2), além dessa diferença inicial para segmentos grandes, não há diferenças significativas entre os dois métodos.

Para as operações de escrita (Tabelas 1 e 3) envolvendo números maiores de clientes, a abordagem de múltiplos arquivos possui desempenho em média 14% melhor do que a de porções contíguas de um arquivo compartilhado. Para segmentos de 64KB, essa diferença começou a ser significativa para um número menor de clientes (24, contra 38 do teste com segmentos de 64MB).

6 Conclusões e trabalhos futuros

O desempenho de um sistema de arquivos distribuído é fortemente dependente do padrão de acesso sendo utilizado pela aplicação. Assim, estudar o comportamento do desempenho de um sistema sob diferentes padrões é uma tarefa importante, pois provê ferramentas para melhores comparações e escolhas entre sistemas. Além disso, aplicações podem utilizar essa informação para realizar a adaptação de suas operações de E/S para que tomem proveito das otimizações providas pelo SAD a ser utilizado.

Um padrão de acesso comum em aplicações científicas é o acesso pelos clientes a dados exclusivos. Duas formas de implementar esse padrão são através de um arquivo para cada nodo, ou um arquivo compartilhado por todos, onde segmentos são atribuídos a processos. Essas duas estratégias foram representadas pelas classes de testes MFWA (*Multiple Files, Whole Access*) e SFSA (*Single File, Segmented Access*). A classe SFSA foi dividida em duas (SFSA *strided* e SFSA *non-strided*), considerando que podem ser atribuídas diversas porções do arquivo a cada processo, ou apenas uma contígua.

Para avaliar o comportamento do *Lustre File System*, testes variando número de clientes envolvidos e granularidade dos acessos foram realizados para as situações descritas. Seus resultados permitem algumas conclusões sobre o comportamento do sistema:

- Se os nodos da aplicação acessarem grandes áreas contíguas de dados, o número de requisições em que esse acesso é realizado não é um fator importante no desempenho final se o acesso for para escrita. Para operações de leitura, a granularidade possui um pequeno impacto (cerca de 10%).
- O pior desempenho foi obtido nos testes SFSA *strided*, em que os processos possuem diversas porções esparsas dentro de um arquivo compartilhado. Se essa for a necessidade da aplicação, pode ser vantajoso estudar uma reorganização dos acessos dentro do arquivo, de forma a agrupar as porções atribuídas a cada processo. Outra alternativa seria escrever esses dados em arquivos independentes, e reorganizá-los numa próxima fase da computação.
- Para números pequenos de clientes (até 3 vezes maior que o número de servidores) e requisições a segmentos grandes, a abordagem de arquivo compartilhado apresenta melhor desempenho do que a de múltiplos arquivos. Para outras situações, ambas se mostram equivalentes para operações de leitura. Para as operações de escrita, a abordagem de arquivos independentes mostrou-se melhor para números grandes de clientes (cerca de 10 vezes maior que o número de servidores).

Como trabalhos futuros, mais situações podem ser simuladas com o Lustre, provendo mais informações sobre o seu comportamento. A mesma metodologia pode ser empregada para testar outros sistemas de arquivos, permitindo uma comparação interessante entre os sistemas. As conclusões podem, ainda, ser utilizadas para a adaptação das operações de E/S de uma aplicação para que acesse o Lustre de forma mais eficiente.

Referências

[1] The asci sppm benchmark code, 1998.

[2] Understanding lustre filesystem internals. National Center for Computational Sciences and Sun Microsystems Inc., April 2009.

[3] F. Z. Boito. Estratégias para avaliação do sistema de arquivos lustre. Trabalho de Conclusão de Curso - Universidade Federal do Rio Grande do Sul, 2009.

[4] Lustre: A scalable, high-performance file system. Cluster File Systems Inc. white paper, version 1.0, November 2002. http://www.lustre.org/docs/whitepaper.pdf.

[5] T. M.-I. Committee. Mpi-io: A parallel file i/o interface for mpi version 0.5, 1996.

[6] A. E. Darling, L. Carey, and W.-C. Feng. The design, implementation, and evaluation of mpiblast. In *In Proceedings of ClusterWorld 2003*, 2003.

[7] P. Dickens and J. Logan. Towards a high performance implementation of mpi-io on the lustre file system. In *OTM '08: Proceedings of the OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008. Part I on On the Move to Meaningful Internet Systems:*, pages 870–885, Berlin, Heidelberg, 2008. Springer-Verlag.

[8] P. M. Dickens and J. Logan. Y-lib: a user level library to increase the performance of mpi-io in a lustre file system environment. In *HPDC '09: Proceedings of the 18th ACM international symposium on High performance distributed computing*, pages 31–38, New York, NY, USA, 2009. ACM.

[9] B. Fryxell et al. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *Astrophys. J. Suppl.*, 131:273–334, 2000.

[10] R. V. Kassick, F. Z. Boito, and P. O. A. Navaux. Testing the performance of parallel file systems. In *Anais do VI Workshop de Processamento Paralelo e Distribuído*, Porto Alegre - RS, 2008.

[11] W. keng Liao, A. Ching, K. Coloma, A. Choudhary, and M. Kandemir. Improving mpi independent write performance using a two-stage write-behind buffering method. *Parallel and Distributed Processing Symposium, International*, 0:295, 2007.

[12] D. Kotz and C. S. Ellis. Practical prefetching techniques for multiprocessor file systems. *Distrib. Parallel Databases*, 1(1):33–51, 1993.

[13] Mpi-io test user's guide. Los Alamos National Laboratory, 2008.

[14] J. Larkin and M. Fahey. Guidelines for efficient parallel i/o on the cray XT3/XT4. Cray Users Group Meeting (CUG), 2007.

[15] E. Smirni and D. A. Reed. Lessons from characterizing input/output bahavior of parallel scientific applications, 1998.

[16] Lustre networking: High-performance features and flexible support for a wide array of networks. Sun Microsystems white paper, version 1.0, November 2008.

[17] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty. File system workload analysis for large scale scientific computing applications. In *In Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, pages 139–152, 2004.

[18] C. Winstead, H. Pritchard, and V. McKoy. Parallel computation of electron-molecule collisions. *IEEE Comput. Sci. Eng.*, 2(3):34–42, 1995.