UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

BRUNO ALVES FAGUNDES
JEAN LUCA BEZ

Relatório de Pesquisa 372

Avaliação da Pilha de E/S do Supercomputador Santos Dumont

Orientador
Prof. Dr. Philippe Olivier Alexandre Navaux

Porto Alegre, 4 de agosto de 2017.

RESUMO

Neste trabalho buscamos investigar a pilha de E/S do supercomputador brasileiro Santos Dumont através da aplicação mais utilizada nesta plataforma. Identificamos as que consumiam maior tempo de processamento e a implementação MPI por elas utilizadas. Utilizando o Darshan, um *profiler* de I/O, foi possível caracterizar o comportamento do GROMACS. Esta aplicação era a que possuía mais jobs na fila do sistema. Buscamos também coletar informações nos clientes do sistema de arquivos paralelo Lustre. Foi possível observar que a aplicação escreve a maior parte dos dados em um único servidor além de centralizar em um único processo as operações de leitura e escrita. Notou-se também que esta aplicação, apesar de ser uma das mais utilizadas, não passa grande parte do seu tempo em operações de I/O. Isto pode estar relacionado com os casos de testes e cenários os quais a mesma é empregada pelos pesquisadores que a utilizam no Santos Dumont.

1 INTRODUÇÃO

Aplicações científicas como simulações atmosféricas, de fluxos ou de terremotos demandam cada vez mais requisitos computacionais para simular esses fenômenos complexos. Durante suas execuções tais aplicações necessitam acessar dados, sejam arquivos de entrada da simulação, *checkpoints* ou arquivos de saída. Por executarem em ambientes de High-Performance Computing (HPC), como clusters ou supercomputadores, estas aplicações normalmente realizam suas operações de entrada e saída (E/S) em um sistema de arquivos paralelo que é compartilhado por todas elas.

Analisando a história da evolução dos computadores podemos observar que, mesmo não sendo totalmente precisa, a Lei de Moore determinou o ritmo de desenvolvimento do processadores nas últimas décadas, como resultado temos chips mais rápidos e eficientes em intervalos de 18 ou 24 meses. Entretanto quando analisamos a evolução dos dispositivos de armazenamento não observamos a mesma velocidade de desenvolvimento. Mesmo dispondo de várias tecnologias para melhorar o desempenho de acesso aos dados, as operações de leitura e escrita ainda são um gargalo para a várias aplicações que precisam acessar um grandes volumes de dados.

Muitas vezes é dada uma atenção especial ao processamento fazendo com que as aplicações consigam utilizar alguns milhares de cores, todavia dependendo da estratégia adotada para ler e escrever os dados todo esse esforço pode ser comprometido. Conforme descrito em [3], observamos três estudos de caso que demonstram como o planejamento do acesso aos dados podem interferir no desempenho de uma aplicação. São eles: durante a execução de uma aplicação eram feitos *checkpoints* excessivos que resultaram em aproximadamente 80% do tempo com operações de E/S; em outro, uma aplicação que utilizava 50.000 cores gastava 15% do seu tempo de execução para ler apenas 20 arquivos pequenos; no terceiro caso, a escolha de um sistema de arquivos não otimizado estava onerando o tempo total de execução em quase 33%. Para todos os caso após a identificação do comportamento de leitura e escrita das aplicações foi possível definir uma nova estratégia para acessar as informações de forma a melhorar o desempenho. Diante disso o investimento na caracterização do comportamento de E/S de uma aplicação é uma etapa que pode contribuir diretamente no sucesso da otimização empregada.

2 MOTIVAÇÃO E OBJETIVOS

Conforme descrito no site oficial [1], o supercomputador Santos Dumont (SDumont) atua como nó central (*Tier-0*) do Sistema Nacional de Processamento de Alto Desempenho - SINAPAD. Possui capacidade instalada de processamento na ordem de 1,1 Petaflops ($1{,}1 \times 10^5$ *float-point operations per second*) com um total de 18.144 núcleos de CPU distribuídos em 756 nós computacionais. Possui também 396 aceleradoras GPU K40, 108 coprocessadores Xeon Phi 7120 e um sistema de arquivos paralelo Lustre com capacidade bruta de armazenamento na ordem de 1,7 PBytes. Essa configuração faz do SDumont o maior supercomputador da América Latina registrado na lista de Junho de 2015 do TOP500 [2], alcançando as posições 146º, 178º e 208º para GPU, Xeon Phi e CPU respectivamente. Apenas esse supercomputador possui mais que o dobro da capacidade de processamento de todos os Centros Nacionais de Processamento de Alto Desempenho (CENAPADs) do Brasil juntos.

Diante do observado em [4], [5] e [6] a caracterização do padrão de leitura e escrita das aplicações é uma ferramenta muito importante para os desenvolvedores e usuários do SDumont conseguirem obter o melhor desempenho de suas aplicações.

Este trabalho tem como objetivo avaliar a pilha de E/S do supercomputador brasileiro Santos Dumont, buscando investigar problemas e perdas de desempenho relacionados às operações de E/S. Pretende-se correlacionar o comportamento do sistema de arquivos com a carga e as aplicações que nele executam, buscando identificar e propor otimizações que poderão possivelmente melhorar o desempenho das aplicações que dependem deste sistema de arquivos compartilhados para ler e escrever seus dados.

3 ANÁLISE

Para compreender melhor como as aplicações que executam no SDumont acessam e utilizam o sistema de arquivos compartilhado Luste foram utilizados três componentes básicos: o Darshan [7] para caracterização das operações de entrada e saída, o collectl [8] para obter os dados sobre o sistema de arquivos e os comandos do gerenciador de recursos SLURM v14.11.11-Bull.1.0 [11] para relacionar as informações com as aplicações dos usuários. Estes aplicativos foram executados através de scripts ou diretamente na linha de comando e as suas respectivas saídas foram armazenadas em arquivos de log para obtenção dos resultados.

Conforme já mencionado anteriormente o SDumont possui 756 nós computacionais que são distribuídos em três grupos: o primeiro é composto por lâminas B710 com dois nós computacionais com 2 CPUs Intel Xeon E5-2695v2 Ivy Bridge, 64GB de memória RAM DDR3 e um disco SSD de 128 GB em cada; o segundo é formado por lâminas do tipo B715 que contém um nó computacional com 2 CPUs Intel Xeon E5-2695v2 Ivy Bridge, 64GB de RAM DDR3, duas placas aceleradoras podendo ser ou NVidia K40 ou Intel Xeon Phi 7120 e um disco SSD de 128 GB; e o terceiro grupo é formado por um nó do tipo S6130 (mesca2) que contém 16 CPUs E7-2870v2 totalizando 240 núcleos e 6,0 TB de memória RAM. Cada nó possui uma interface infiniband FDR 56Gb/s configuradas em uma topologia *Fat Tree* para comunicação MPI e acesso ao sistema de arquivos Lustre.

O Lustre é um sistema de arquivos paralelo, baseado em objetos, open source, projetado para ser escalável e com suporte a dezenas de milhares de clientes. A arquitetura de um sistema Lustre é composta por: Servidor de Gerenciamento (MGS - Management Server); Servidor de Metadados (MDS - Metadata Server); Servidor de armazenamento dos objetos de metadados (MDT - Metadata Target); Servidor de objetos (OSS - Object Storage Server) e Servidor de armazenamento dos dados dos objetos (OST - Object Storage Target). O sistema Lustre disponível no supercomputador é composto por 2 servidores de administração, dois nós com função MDT e MGS que funcionam em modo ativo-ativo (caso um nó falhe o par assume as suas funções dentro do sistema) e 10 nós como OSS em modo ativo-ativo.

A apresentação deste trabalho está dividida em 4 partes: definição dos cenários de execução, coleta das informações de I/O utilizando o Darshan e coleta dos dados do sistema de arquivos.

3.1 Definição dos Cenários

O supercomputador SDumont atende a diversos projetos de pesquisa com necessidades distintas, isso permite que o ambiente configurado para uma tarefa seja completamente diferente do ambiente de outra. Testes realizados em outro projeto de pesquisa que visava otimizar o desempenho de E/S de uma aplicação de modelagem numérica atmosférica identificaram que o mesmo software utilizando as diferentes implementações do MPI disponíveis no Santos Dumont apresentou tempos de execução bem diferentes. Em virtude dessas informações é importante saber quais implementações MPI são utilizadas por cada *job*.

O ambiente do Santos Dumont possui o gerenciador de recursos SLURM, que é o software responsável por organizar, priorizar e iniciar todos as tarefas que são criadas pelos usuários, diante disso qualquer tarefa que é executada no supercomputador passa primeiro pelo gerenciador de recursos onde são verificados se os recursos solicitados estão disponíveis.

O software *module* é o responsável pelo gerenciamento e configuração do ambiente de execução. Partindo do princípio que os usuários precisam executar o comando *module* para carregar as configurações do ambiente MPI desejado foi desenvolvido um *script* que executa de forma periódica, em intervalos de 5 minutos, registrando qual ambiente foi configurado para as tarefas submetidas.

Os cenários foram classificados de acordo com a implementação MPI.

- **Cenário 1 (BullxMPI):** Aplicações que utilizam a implementação do MPI mantida pela empresa BULL, versão 1.2.8.4.

- **Cenário 2 (Intel MPI):** Aplicações que utilizam a implementação do MPI da Intel, tanto para versão 5.1.3 Build 20160120 quanto para versão 2017 Update 1 Build 20161016.

- **Cenário 3 (OpenMP):** Aplicações que utilizam a implementação do OpenMP versões 1.10, 2.0.1 e 2.1.

- **Cenário 4 (MPI não identificado):** Nesse cenário estão contidos os *jobs* que não utilizam uma implementação MPI ou não carregaram explicitamente um módulo de configuração do ambiente utilizando o comando *module*.

A Figura 1 apresenta o total de *jobs* submetidos considerando cada um dos cenários descritos, no período de 15/06/2017 até 22/07/2017 .

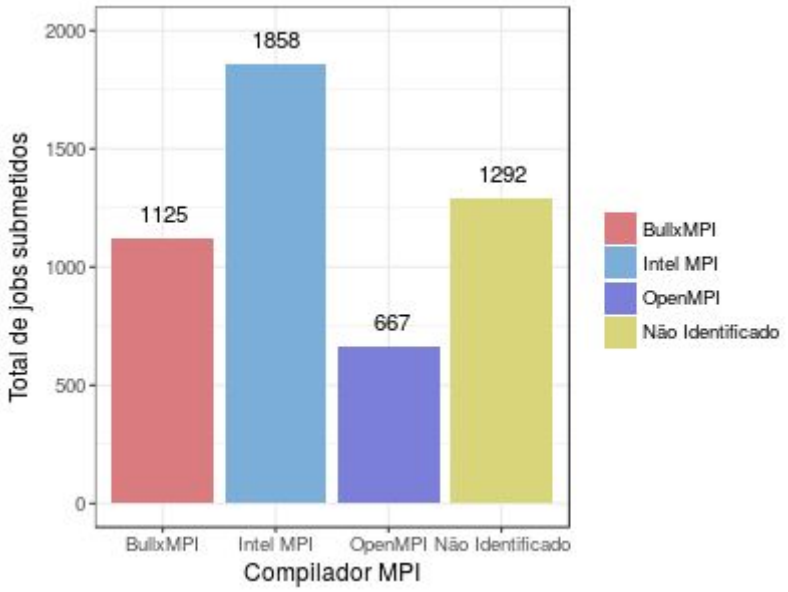

Figura 1 - *Jobs* por ambiente MPI

Em seguida foram relacionados os ambientes MPI e o tempo médio de execução dos *jobs*, para esse cálculo não incluímos os *jobs* submetidos para as filas de desenvolvimento. Nesse contexto temos 3915 *jobs* agrupados conforme o tempo total de execução. Conforme ilustrado pela Figura 2 a distribuição do tempo de execução dos *jobs* é semelhante para as diferentes implementações do MPI. Isso indica que não há uma tendência de jobs mais curtos ou mais longos em utilizarem alguma implementação específica.

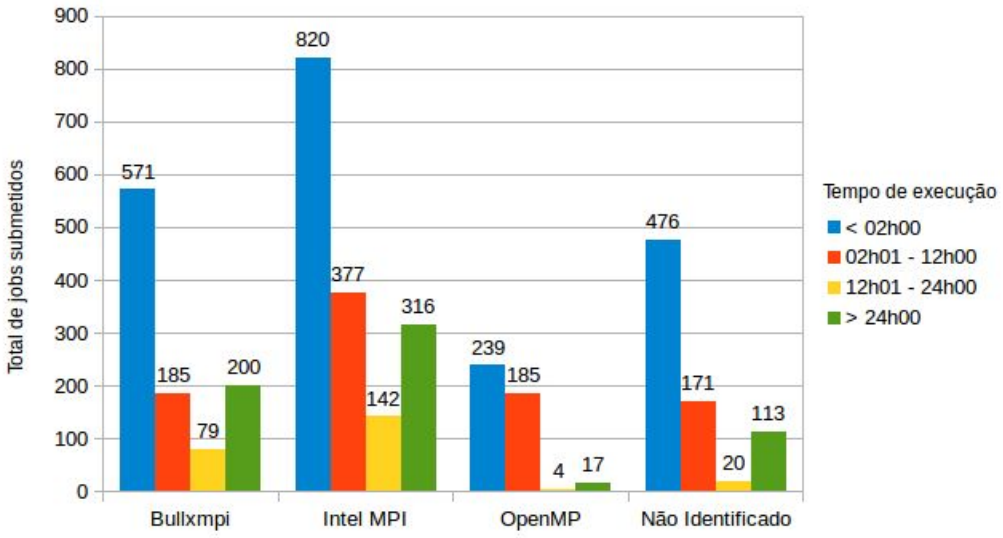

Figura 2 - Tempo de execução por ambiente MPI

O próximo passo foi identificar quais *jobs* utilizaram os softwares compilados pela equipe de suporte do supercomputador. Observou-se que existiam muitas submissões que apresentaram falha ou foram canceladas, diante disso foram descartados os *jobs* cancelados pelos usuários ou que apresentaram o status *FAILED*, nesse contexto obteve-se o total de 2560 *jobs*. Assim como na identificação dos ambientes MPI, utilizamos o carregamento explícito do módulo de ambiente para identificar os softwares que foram utilizados na execução dos *jobs.* A Tabela 1 relaciona os três softwares mais utilizados durante o período considerado neste trabalho.

Aplicação	***Jobs* submetidos**	**Total de horas**
Gromacs [11]	309	13312
Quantum Espresso [12]	80	3895
Plumed [13]	9	346

Tabela 1 – Total de execuções das aplicações disponíveis no SDumont.

Conforme pode ser observado o software GROMACS foi o mais utilizado pelos usuários. Buscando compreender o uso desta aplicação no SDumont, a Figura 3 apresenta a relação entre os *jobs* submetidos, o tempo total de execução e os ambientes MPI configurados.

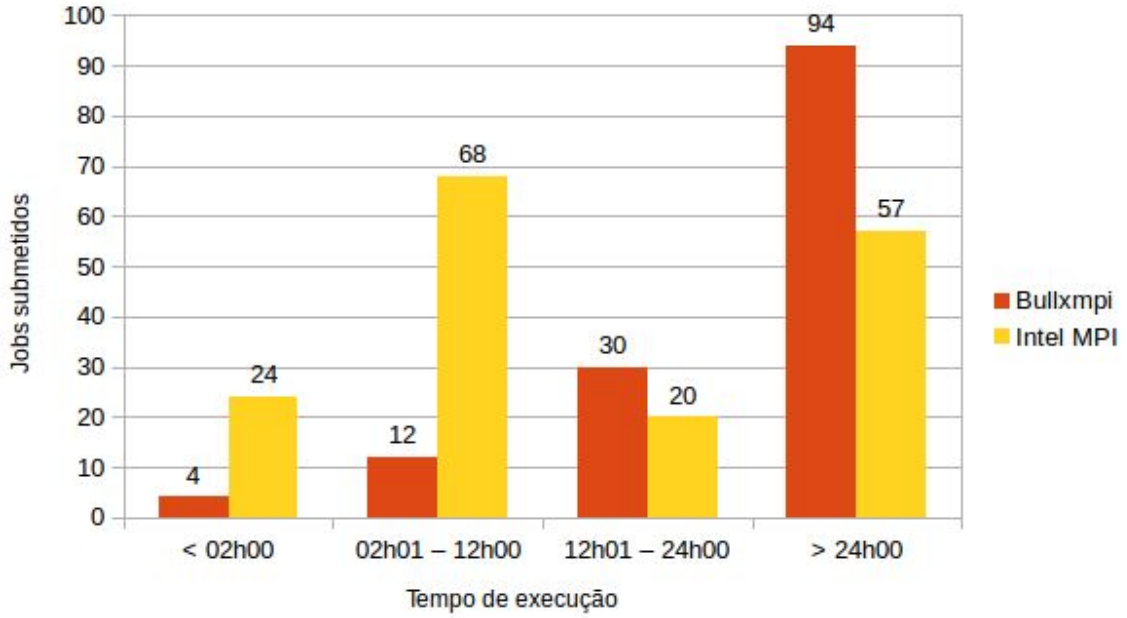

Figura 3 - Tempo de execução dos *jobs* do GROMACS

Com base nos dados coletados observou-se que, das aplicações disponíveis no supercomputador, o GROMACS é a mais utilizada e a divisão entre os ambientes MPI escolhidos está equilibrada, temos 169 *jobs* que rodaram com Intel MPI e 140 com a implementação da Bull. Entretanto, na maioria dos casos, execuções mais rápidas utilizam o Intel MPI e execuções mais longas utilizam o BullxMPI. Isso pode indicar que a implementação da BullxMPI provê um desempenho pior ao executar o GROMACS, porém uma análise mais aprofundada seria necessária para confirmar esta hipótese.

Uma vez que a aplicação mais utilizada pelos usuários foi identificada faz-se necessário obter mais informações sobre como ela acessa os dados no sistema de arquivos paralelo do supercomputador.

4 CARACTERIZAÇÃO DAS APLICAÇÕES COM O DARSHAN

O Darshan [7] foi originalmente desenvolvido para os computadores da série IBM Blue Gene e posteriormente portado para diversas plataformas como Cray e clusters Linux. Conforme demonstrado em [5] a utilização dessa ferramenta permite identificar os padrões de acesso aos arquivos com um sobrecusto mínimo para as aplicações. Sua utilização geralmente está associada a investigação ou ao *tuning* das operações de leitura e escrita.

Existem duas formas de instrumentar as aplicações com o Darshan: uma através de *wrappers* para os compiladores quando precisamos gerar executáveis estaticamente linkados; e outra através do pré-carregamento de bibliotecas quando pretendemos gerar executáveis linkados dinamicamente. O Darshan instrumenta apenas aplicações MPI que fazem uso das funções `MPI_Init()` e `MPI_Finalize()`, entretanto ele captura informações sobre o acesso aos arquivos tanto para MPI-IO quanto para POSIX. Para as aplicações que utilizam HDF5 e PnetCDF são capturadas uma quantidade limitada de informações.

Quando executamos uma aplicação o Darshan intercepta as funções de E/S e coleta as informações que permitem a identificação dos padrões de acesso aos arquivos, compacta os dados e metadados em um buffer de memória e ao final da execução da aplicação grava todas as informações em arquivos de log.

Foram compiladas duas versões do Darshan 3.1.4, uma utilizando o Bullxmpi v1.2.8 e outra utilizando o MPI da Intel v2017 Update 1, ambas com o compilador `gcc` versão 6.2. Cada instalação foi armazenada em um diretório específico e foi utilizada para coletar os dados das respectivas implementações MPI.

Inicialmente o Darshan foi configurado para gerar os arquivos em um diretório no sistema de arquivos Lustre, mas os testes envolvendo o MPI da Intel apresentaram um comportamento inesperado. Para usufruir do suporte nativo ao sistema de arquivos a documentação da ferramenta [9] recomenda a utilização da variável de ambiente `I_MPI_EXTRA_FILESYSTEM_LIST` setada com o valor "`lustre`", durante os testes realizados sempre que essa variável era configurada o Darshan não conseguia escrever seus arquivos fazendo com que o *job* fosse encerrado com erro. A alternativa encontrada foi exportar um sistema de arquivos `ext4` via NFS para todos os nós computacionais e configurar o Darshan para gerar os arquivos de *log* nessa área. Como isso é feito após o fim da aplicação então o desempenho das operações de I/O da mesma não é afetado por isso.

Em algumas situações os *jobs* em execução no supercomputador são mais longos do que a janela de execução definida nas políticas de utilização dos recursos (48 horas) e nesses casos o SLURM encerra a aplicação enviando um sinal SIGTERM para a aplicação em seguida envia um SIGKILL e a execução é encerrada. Se a aplicação que receber o sinal SIGTERM não executar a chamada de sistema `MPI_Finalize()` o Darshan não irá criar os arquivos de log e as informações coletadas durante a execução serão descartadas. Outra situação que pode ocorrer é quando o usuário solicita os recursos computacionais ao gerenciador de recursos e dentro da mesma submissão executa diversas aplicações, neste caso um único *job* poderá gerar vários arquivos de estatísticas, um para cada vez que uma aplicação foi finalizada corretamente.

Após avaliação preliminar dos *jobs* que foram submetidos ao gerenciador de recursos, optou-se por iniciar a caracterização das operações de entrada e saída com o GROMACS uma vez que era o software que possuía mais *jobs* na fila, conforme demonstrado na Seção 3.1. GROMACS é uma aplicação paralela para simulação de dinâmica molecular criada inicialmente para simulação de moléculas bioquímicas como proteínas, lipídios, e ácidos nucléicos [10], mas também empregada em outros cenários que utilizam *non-bonded interactions* [14]. O GROMACS utiliza uma abordagem master-slave para escrever os resultados no disco. A cada timestep onde dados são escritos, o mestre centraliza os dados de forma síncrona, atraveś de troca de mensagens MPI antes de escrever em um único arquivo. Durante esta etapa a simulação é bloqueada, o que pode afetar o desempenho da aplicação e forçar cientistas a reduzir a frequência com que os dados são escritos [15].

Optou-se por utilizar o Darshan no modo não instrumentado, ou seja, configurando a variável de ambiente `LD_PRELOAD` tendo em vista que os testes utilizando os *wrappers* para os compiladores MPI apresentaram erros durante a compilação. Todas as versões do GROMACS disponíveis no SDumont foram configuradas para gerar os arquivos de log.

Ao final foram gerados 168 arquivos do Darshan com as informações das execuções do GROMACS, entretanto ao excluirmos os *jobs* que foram cancelados pelos usuários temos um total de 91 arquivos divididos em faixas de tempo conforme indicado na Figura 4.

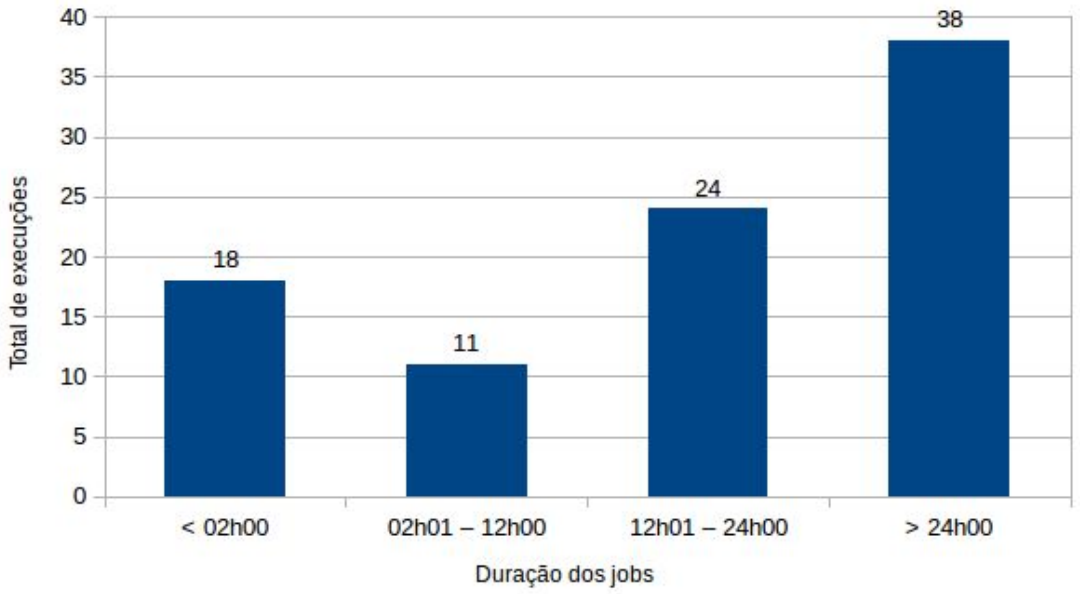

Figura 4 - Execuções do GROMACS

Os resultados das análises do Darshan mostraram que o tempo gasto com operações de entrada e saída são muito baixos, em praticamente todos os experimentos o percentual do tempo de execução gasto com operações de E/S foi inferior a 1%. Todos os *jobs* mapeados utilizam a API STDIO para leitura e escrita.

Execuções	**% do *Walltime* para E/S**
85	< 1%
3	> 1% e < 2%
3	> 2%

Tabela 2 – Tempo de operações de leitura e escrita

A Tabela 3 apresenta a análise dos 6 jobs que tiveram mais de 1% do *walltime* gasto com operações de leitura e escrita. É possível identificar que 5 deles executaram em até 2 segundos e utilizaram apenas uma CPU, com base no comportamento das outras execuções do GROMACS podemos concluir que essas 5 execuções não possuem representatividade na análise. Isto pode estar relacionado ao custo de fazer operações de escrita no GROMACS. Como ocorre com outras aplicações científicas (principalmente as de simulações que geram um grande quantidade de dados a cada *checkpoint*) os cientistas podem acabar optando por

reduzir a frequência com que os dados são escritos para que a aplicação execute mais rapidamente. Outra possibilidade é de que nos casos utilizados pelos pesquisadores que a executam no SDumont não existe grande necessidade de escrita de dados durante a simulação. Porém, essa análise é sugerida como trabalho futuro, já que extrapola o escopo deste trabalho de pesquisa, buscando melhor compreender a necessidade da aplicação e dos pesquisadores que a utilizam neste supercomputador para então sugerir possíveis otimizações que permitam com que o GROMACS possa escrever mais informações.

CPU	***Walltime* (s)**	**Total E/S (s)**	**% do *Walltime* para E/S**
1	1	0,0295	2,95
1	1	0,0318	3,18
1	2	0,0245	1,22
1	2	0,0294	1,47
1	2	0,5080	25,40
960	33476	457,2994	1,36

Tabela 3 – Tempo de operações de leitura e escrita

5 COLETA DAS INFORMAÇÕES DOS SISTEMA DE ARQUIVOS LUSTRE

Coletar os dados referentes ao sistema de arquivos na arquitetura do supercomputador SDumont se mostrou um grande desafio uma vez que o Lustre v2.1 é implementado através do software ClusterStor v1.5.0 da Xyratex/Seagate e existe pouca informação disponível sobre esta versão. A escolha do collectl para obter as informações do sistema de arquivos foi motivada pelo fato de conseguirmos, através dele, obter os dados a partir dos clientes Lustre sem a necessidade de modificar o sistema operacional dos componentes do sistema, dessa forma mantendo a configuração original do ambiente. O ClusterStor possui uma ferramenta web que apresenta algumas informações sobre a situação atual do sistema de arquivos, como por exemplo: quantidade de operações read, write, open, close, throughput entre outras. Entretanto não foi possível, até o momento, obter mais detalhes de como esses dados são coletados ou se existe uma API disponível para consulta. Sendo assim a estratégia adotada para obter informações foi utilizar o aplicativo collectl instalado nos clientes Lustre.

O sistema de arquivos Lustre é montado nos nós de processamento com a opção `flock` que habilita o travamento dos arquivos no contexto de todos os clientes Lustre garantindo a integridade para acessos simultâneos aos arquivos.

Foi desenvolvido um *script* utilizando a versão 4.1.3 do collectl que, de 2 em 2 segundos, captura as informações referentes às operações de leitura e escrita agrupadas por OST e agrupadas por cliente. As Tabelas 3 e 4 apresentam a descrição dos dados que são armazenados para cada caso.

OST	Nome da OST
Reads	Quantidade de operações de leitura
ReadKB	KBs lidos/segundo
Writes	Quantidade de operações de escrita
WriteKB	KBs escritos/segundo

Tabela 4 – Dados armazenados por OST.

Reads	Quantidade de operações de leitura
ReadKB	KBs lidos/segundo
Writes	Quantidade de operações de escrita
WriteKB	KBs escritos/segundo
Open	Quantidade de operações *open*
Close	Quantidade de operações *close*
GAttr	Get *attributes*/segundo
SAttr	Set *attributes*/segundo
Seek	Quantidade de operações *seek*
Fsync	*FSyncs*
DrtHit	*Dirty Hits*
DrtMis	*Dirty Misses*

Tabela 5 – Dados armazenados por cliente Lustre.

Durante a realização deste trabalho foram monitoradas 198 máquinas do tipo B715. Espera-se que o *script* de coleta não influencie no desempenho dos nós computacionais. Durante sua execução o processo do collectl consumiu, em média, 1,5% da capacidade de processamento de um core e 21 MB de memória RAM de cada máquina. Os arquivos de log foram gerados no diretório `/tmp` local e para evitar que os arquivos ocupassem muito espaço no sistema de arquivos. Uma vez por dia, às 00h00, o processo de coleta era interrompido e os arquivos de log eram compactados e movidos para o diretório de armazenamento fora do nó computacional, ao final desse procedimento um novo processo de coleta era iniciado e os dados eram armazenados em um novo arquivo vazio.

Após a coleta das informações dos clientes Lustre foram descartadas as linhas que possuíam valor 0 para todas colunas descritas nas Tabelas 4 e 5 o que indica que o cliente Lustre não realizou nenhuma operação no sistema de arquivos. Em seguida, os dados correspondentes ao período de execução de cada *job* foram separados para identificar a quantidade total de KB lidos e escritos por cada cliente em cada OST do Lustre.

A Figura 5 apresenta uma relação entre o tamanho total de dados lidos e a duração dos *jobs* observados, considerando apenas aqueles que tiveram duração de até 24 horas. Na Figura, o eixo *x* indica a duração dos *jobs* em horas, e o eixo *y* o tamanho total de dados lidos pelo *job* em GB. É possível ver que alguns *jobs* de menor duração (até 4 horas) leram uma

grande quantidade de dados (aproximadamente 90GB), enquanto que os com duração entre 14 e 24 horas não leram mais do que 30GB de dados do sistema de arquivos Lustre durante sua execução.

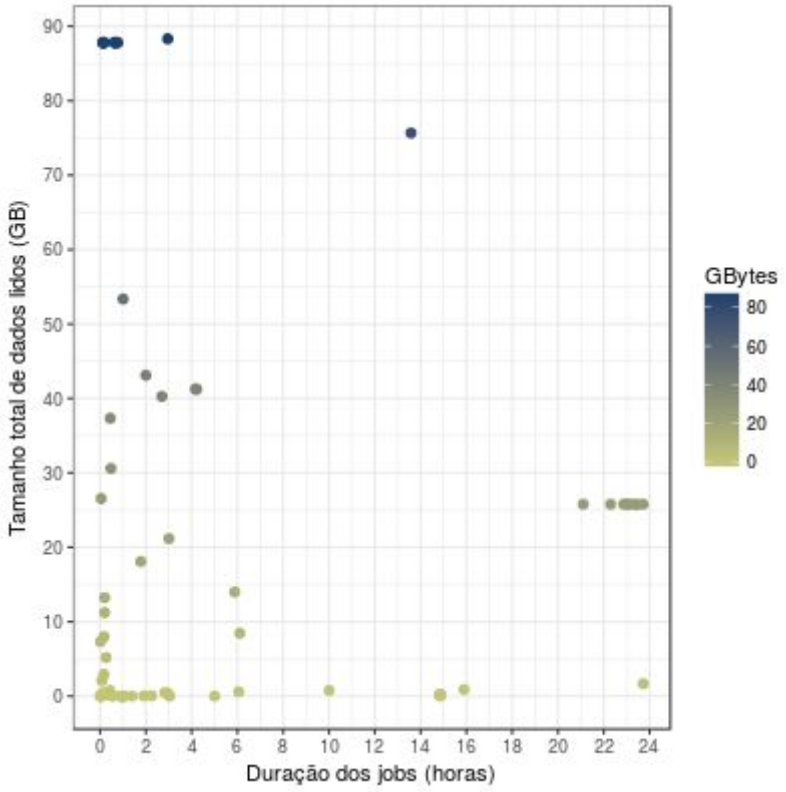

Figura 5 - Tamanho total lido por *job* com duração inferior a 24h

Com as informações coletadas nos clientes Lustre, e agrupadas por *job*, foi possível também analisar o tamanho total de dados escritos por execução da aplicação. A Figura 6 ilustra os *jobs* com até 24 horas de duração (eixo *x*) e o total de dados escritos em GB (eixo *y*). É possível notar que as execuções observadas durante este período não fazem uma grande quantidade de escritas.

Por fim, a Figura 7 ilustra a razão observada entre as operações *seek* e o total de operações reportadas pelo *script* (open, seek, read, write e close). Um grande número de *seeks* pode indicar vários acessos não-contíguos, que geralmente estão associados com um custo maior da operação de leitura ou escrita devido a forma como o acesso é feito aos discos.

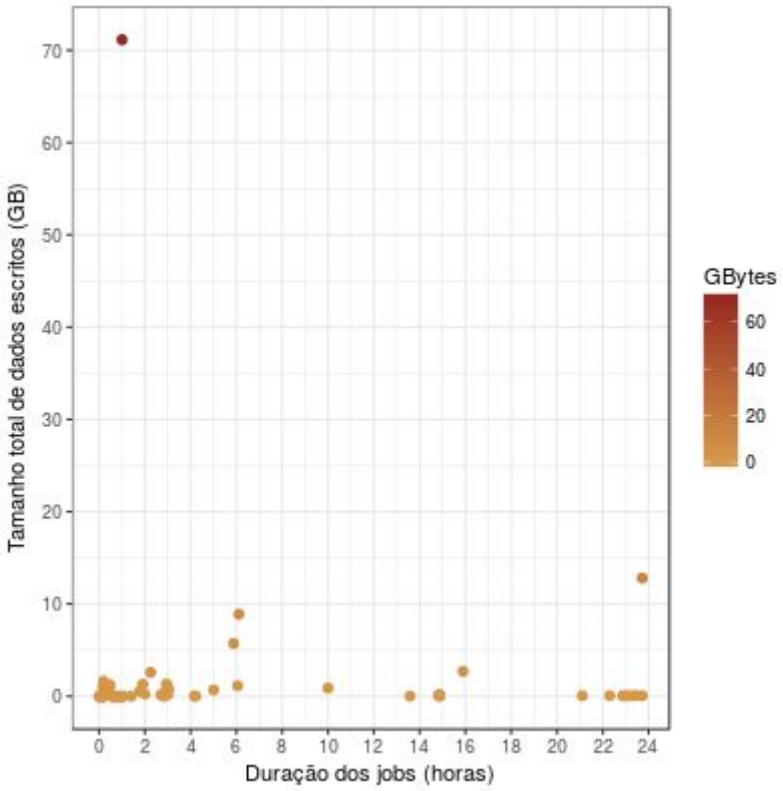

Figura 6 - Tamanho total escrito por *job* com duração menor que 24h

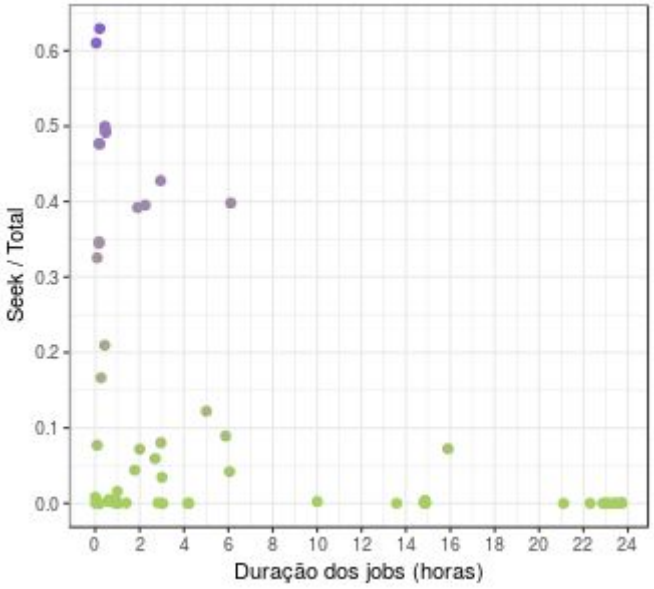

Figura 7 - Razão entre o total de chamadas *seek* com o total de chamadas por *job* (considerando apenas os com duração menor que 24h)

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho buscou-se avaliar a pilha de E/S do supercomputador brasileiro Santos Dumont, buscando investigar problemas e perdas de desempenho relacionados às operações de E/S. Para relacionar o comportamento do sistema de arquivos com a carga e as aplicações que executam no supercomputador foi necessário identificar quais softwares consumiam maior tempo de processamento do cluster e qual implementação MPI cada aplicação utiliza. Assim foi possível configurar uma ferramenta que permitisse obter as informações sobre as operações de leitura e escrita sem gerar prejuízos para as tarefas dos usuários e obter informações sobre como cada nó de processamento está acessando os dados do sistema de arquivos.

Pela observação das informações apresentadas foi possível compreender como o GROMACS se comportou em relação às operações de leitura e escrita para os usuários do supercomputador. Conforme indicado nos resultados do Darshan, o tempo médio gasto com as operações de entrada e saída não tiveram representatividade no tempo total das execuções. Através dos resultados coletados nos clientes do Lustre foi possível observar que a aplicação escreve a maior parte dos dados em uma única OST isso porque ela centraliza em um processo as operações de leitura e escrita no sistema de arquivos.

Para os trabalhos futuros pretende-se realizar novas análises com o GROMACS buscando levantar os requisitos de I/O da aplicação e as necessidades dos pesquisadores que a utilizam no SDumont. Desta forma será possível investigar e sugerir otimizações que permitam, por exemplo, que a aplicação execute operações de escrita com uma maior frequência (caso isso seja apontado como um problema). Pretende-se também ampliar o universo dos testes, permitindo que outras aplicações possam ser caracterizadas. Conforme pode ser observado na Tabela 1 o software Quantum Espresso é o segundo software configurável através de módulos de ambiente mais executado no SDumont e pode ser utilizado como ponto de partida para um novo mapeamento do comportamento do acesso ao sistema de arquivos. Outra frente é investir mais esforços na geração dos *wrappers* do Darshan para que as aplicações compiladas diretamente pelos usuários também possam ser caracterizadas.

BIBLIOGRAFIA

[1] Santos Dumont: http://sdumont.lncc.br. Acessado em abril de 2017.

[2] TOP 500, lista de novembro de 2015: https://www.top500.org/list/2015/06/. Acessado em abril de 2017.

[3] PRABHAT, Quincey Koziol. High Performance Parallel I/O. In: ANTYPAS, Katie; YAO, Yushu. Overview of I/O Benchmarking. California, USA, Quincey Koziol, 2014, pp. 279-288.

[4] CARNS, Philip; LATHAM, Robert; ROSS, Robert; ISKRA, Kamil; LANG, Samuel. "24/7 Characterization of petascale I/O workloads," In *Proceedings of IEEE International Conference on Cluster Computing and Workshops*. New Orleans, LA, 2009, pp. 1-10.

[5] SNYDER, Shane; CARNS, Philip; HARMS, Kevin; ROSS, Robert. "Modular HPC I/O characterization with Darshan". In *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools* (ESPT '16). IEEE Press, Piscataway, NJ, USA, pp. 9-17.

[6] CARNS, Philip; YAO, Yushu; HARMS, Kevin; LATHAM, Robert; ROSS, Robert; ANTYPAS, Katie. "Production I/O Characterization on the Cray XE6", In *Proceedings of Cray User Group* (CRUG'13). Napa Valley, CA, USA, pp. 1-10.

[7] Darshan: http://www.mcs.anl.gov/research/projects/darshan/. Acessado em abril de 2017.

[8] http://collectl.sourceforge.net/. Acessado em maio de 2017.

[9] https://software.intel.com/en-us/node/528846. Acessado em maio de 2017.

[10] http://www.gromacs.org/. Acessado em abril de 2017.

[11] https://slurm.schedmd.com/. Acessado em abril de 2017.

[12] http://www.quantum-espresso.org/. Acessado em maio de 2017.

[13] http://www.plumed.org/. Acessado em maio de 2017.

[14] PAVLOV, Valentin and PETKOV, Peicho; "Data I/O Optimization in GROMACS Using the Global Arrays Toolkit". Disponível em: http://www.prace-ri.eu/IMG/pdf/Data_IO_Optimization_in_GROMACS_Using_the_Global_Arrays_Toolkit-2.pdf.

[15] M. Dreher and B. Raffin, "A Flexible Framework for Asynchronous in Situ and in Transit Analytics for Scientific Simulations," 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Chicago, IL, 2014, pp. 277-286.