XIX Simpósio em Sistemas Computacionais de Alto Desempenho
De 01 a 03 de Outubro de 2018
São Paulo – SP

Minicursos do WSCAD 2018

Organizadores
Denise Stringhini
Edson Norberto Cáceres

Sociedade Brasileira de Computação – SBC

ISBN: 978-85-7669-453-3

STRINGHINI, Denise; CÁCERES, Edson Norberto (Org.)
Minicursos do WSCAD 2018. / Denise Stringhini,
Edson Norberto Cáceres (Org.). - São Paulo: Sociedade
Brasileira de Computação, 2018.

84 folhas : il., fig., tab.

ISBN: 978-85-7669-453-3

1. Computação de alto desempenho . 2. Computação em nuvem.
3. I/O paralela. 4. Laboratório de supercomputação I. Título.

Apresentação

Neste livro estão compilados os três minicursos apresentados durante o XIX Simpósio em Sistemas Computacionais de Alto Desempenho, realizado entre os dias 01 e 03 de outubro de 2018 em São Paulo, SP.

Em todos os minicursos destacamos o viés prático que incentiva os participantes e os leitores a utilizarem alto desempenho de maneira efetiva. O minicurso *Introdução à Sistemas de E/S e Armazenamento Paralelos* apresenta uma introdução a sistemas de E/S e armazenamento paralelos voltados para ambientes de computação de alto desempenho. Os autores mostram que, cada vez mais, é preciso se preocupar com a forma como grandes conjuntos de dados são recuperados e armazenados por aplicações distribuídas de larga escala, visto que o acesso a estes dados pode ocupar um tempo significativo da execução da aplicação.

O minicurso *Boas Práticas para a Implementação e Gerência de um Centro de Supercomputação Desassistido* propõe compartilhar algumas das melhores práticas adotadas na implantação de um supercomputador. A principal finalidade do minicurso é trazer informações sobre integração de diversos serviços para realizar este gerenciamento, tais como a implantação de um Portal de Usuários o uso de internet das coisas em ambientes de supercomputação. Neste sentido, também são abordadas algumas tecnologias que podem ser utilizadas para aumentar a segurança do sistema.

Finalmente, o minicurso *Introdução à Computação de Alto Desempenho na Nuvem Computacional* apresenta, além de uma introdução aos conceitos de computação de alto desempenho, uma visão geral dos serviços oferecidos na Nuvem Computacional e como estes serviços podem ser utilizados para realizar computação de alto desempenho. Além disso, o minicurso apresenta dois casos de uso no provedor de serviços *Amazon Web Services* (AWS) caracterizando o aspecto prático do minicurso.

Com este livro esperamos contemplar os usuários e pesquisadores da área de Alto Desempenho que não puderam estar presentes no evento. Desejamos uma boa leitura.

Denise Stringhini e Edson Norberto Cáceres
Coordenadores dos Minicursos - WSCAD 2018

Sumário

Capítulo 1

Introdução a Sistemas de E/S e Armazenamento Paralelo

Eduardo C. Inacio - *PPGCC/UFSC - eduardo.camilo@posgrad.ufsc.br*
Mario A. R. Dantas - *DCC/UFJF - mario.dantas@ice.ufjf.br*

Resumo

A proposta deste minicurso é oferecer uma introdução a sistemas de entrada e saída (E/S) e armazenamento paralelos voltados para ambientes de computação de alto desempenho (CAD). Por meio de uma abordagem expositiva, utilizando exemplos referentes a problemas reais, pretende-se demonstrar como o desempenho de E/S pode ser consideravelmente melhorado em aplicações distribuídas com modificações tanto na programação da aplicação quanto na configuração e utilização do sistema de armazenamento. Espera-se que, ao final deste minicurso, o participante tenha um visão geral da infraestrutura básica de armazenamento e da pilha de software de E/S encontrada nos ambientes de CAD de larga escala modernos, assim como consiga identificar e utilizar as funções básicas dos principais middlewares, bibliotecas e ferramentas para otimização de E/S.

1.1. Introdução

Tópicos relacionados a processamento continuam sendo o centro das atenções da área de computação de alto desempenho (CAD). Novas técnicas e tecnologias vêm sendo propostas nas últimas décadas visando aumentar o poder de processamento de infraestruturas computacionais de larga escala e, com isso, reduzir o tempo necessário para resolução de problemas científicos e de engenharia que dependem de sistemas de simulação e visualização. Ao mesmo tempo, a disponibilidade de infraestruturas com maior poder computacional tem incentivado a pesquisa de problemas cada vez maiores e mais complexos.

À medida que tais problemas crescem em complexidade e tamanho, têm-se observado que o volume de dados produzido e consumido também vem crescendo significativamente. Poderosos instrumentos empregados em pesquisa experimental e observacional, como aceleradores de partículas, telescópios, satélites e sequenciadores genéticos podem produzir terabytes de dados por segundo, dos quais, petabytes de dados chegam a ser armazenados anualmente [Bell et al. 2009, CERN 2016, Guzman et al. 2016]. Aplicações

de simulação computacional em áreas como cosmologia, física de altas energias, geociência e exploração e produção de petróleo podem gerar terabytes de dados de saída em uma única execução [Chen et al. 2009, Baker et al. 2014, Roten et al. 2016]. Usualmente, estes enormes conjuntos de dados são processados posteriormente por aplicações de visualização e análise para que informações relevantes para a pesquisa em questão possam ser extraídas [Mitchell et al. 2011, Nonaka et al. 2014, Dorier et al. 2016]. Vale ressaltar também a crescente sinergia entre ambientes e aplicações de CAD, Big Data e Internet das Coisas, cujos prognósticos são de demandas de dados ainda maiores, ultrapassando a escala dos yottabytes [Reed and Dongarra 2015, Radha et al. 2015, Zhao et al. 2016].

Neste contexto, o acesso, seja para leitura ou escrita, a estes grandes conjuntos de dados se apresenta como um dos principais gargalos de desempenho para estas aplicações, uma vez que a latência dos dispositivos que compõem a infraestrutura de armazenamento secundário chega a ser até seis ordens de magnitude maior do que a latência de acesso a memória principal, por exemplo [Lüttgau et al. 2018]. Consequentemente, observa-se em aplicações distribuídas que fazem uso intensivo de dados, principalmente naquelas que utilizam operações de entrada e saída (E/S) síncronas, ou seja, o processamento somente é retomado após a conclusão da operação, que o tempo utilizado para o armazenamento e recuperação de dados corresponde a uma fração considerável do tempo total de execução, podendo chegar até mesmo a superar o tempo utilizado para o processamento [Nonaka et al. 2018]. Visando atender as demandas de escalabilidade, tanto de desempenho quanto de capacidade de armazenamento, impostas por estas aplicações, infraestruturas com múltiplas camadas e sistemas de armazenamento altamente distribuídos são tradicionalmente empregados em ambientes de CAD. A Figura 1.1 apresenta um modelo de infraestrutura física de armazenamento e uma pilha de software de E/S típicos de ambientes de CAD.

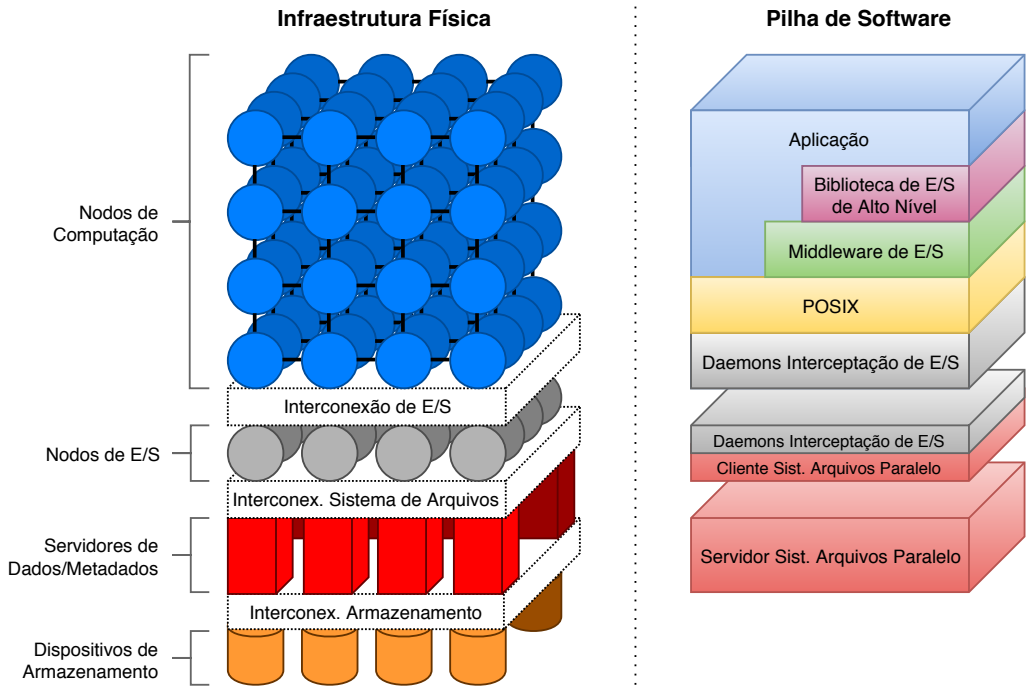


Figura 1.1. Visão geral de um modelo de infraestrutura física e uma pilha de software de E/S típicas de ambientes de CAD.

Nestes ambientes, como pode ser observado na figura, elementos de processamento e de armazenamento são bem diferenciados em termos de infraestrutura. As aplicações executam em milhares de nodos especializados, com alto poder de processamento, que se comunicam entre si utilizando redes de interconexão de alta velocidade, muitas vezes de tecnologia proprietária. Estes *nodos de computação*, em muitos casos, não possuem dispositivos de armazenamento local, tendo seu armazenamento secundário localizado inteiramente em servidores remotos. Desta forma, todos os dados necessários para a execução da aplicação, assim como toda a saída gerada por ela, precisa ser transportada entre os nodos de computação e o armazenamento secundário remoto. A aplicação pode se utilizar de diferentes bibliotecas, disponibilizadas nos nodos de computação, para realizar o acesso aos seus arquivos. As alternativas vão desde chamadas de sistema POSIX, funções coletivas de middlewares de E/S distribuídas como o MPI-IO [MPIForum 2018], até bibliotecas de alto nível como o HDF5 [The HDF Group 1997] e PNETCDF [Li et al. 2003]. Mais detalhes sobre métodos utilizados por aplicações distribuídas para armazenamento e recuperação de dados em arquivos serão apresentados na Seção 1.3.

Em grande parte dos ambientes de CAD modernos, o armazenamento secundário é formado por múltiplos discos rígidos magnéticos (hard-disk drive (HDD)), uma tecnologia madura que oferece alta durabilidade e baixo custo por byte, porém com alta latência e moderada taxa de transmissão. Novas tecnologias de armazenamento baseadas em memórias não voláteis, como solid state drive (SSD) e non-volatile random-access memory (NVRAM), vêm ganhando espaço nestes ambientes, oferecendo uma redução de até três ordens de magnitude na latência de acesso, para uma vazão até dezesseis vezes superior a dos HDDs [Lüttgau et al. 2018]. Contudo, questões de custo e características de durabilidade destes novos dispositivos ainda inviabilizam sua ampla utilização em substituição aos tradicionais HDDs no armazenamento secundário de grande porte. Uma abordagem que vem sendo utilizada em alguns ambientes e tem mostrado bons resultados consiste na utilização de SSDs e NVRAMs para a composição de uma camada intermediária entre os nodos de computação e a infraestrutura de armazenamento secundário, atuando como *nodos de E/S*. Uma das principais finalidades destes nodos é absorver as rajadas de transferência de dados dos nodos de computação, amortizando assim a latência do armazenamento secundário baseado em HDDs [Liu et al. 2012]. Assim que os dados são transferidos para os nodos de E/S, os processos da aplicação podem ser liberados para prosseguir com o processamento, enquanto os nodos de E/S transferem os dados para o armazenamento secundário propriamente dito.

Nestes ambientes de larga escala, o armazenamento secundário precisa ser altamente distribuído e escalável, de forma a suportar um grande número de requisições de E/S concorrentes, sejam elas oriundas dos processos das aplicações executando nos nodos de computação, ou dos processos coordenados pelos nodos de E/S [Prabhat and Koziol 2014]. Tradicionalmente, esse armazenamento secundário é implementado por um sistema de arquivos paralelo (SAP), como o LUSTRE [OpenSFS 2018] e o ORANGEFS [Omnibond 2018], por exemplo. Nos SAPs, como será visto com mais detalhes na Seção 1.2, os arquivos são particionados e distribuídos entre múltiplos servidores. Grandes requisições de E/S têm desempenho diferenciado nesses ambientes, uma vez que o acesso aos dados pode ser realizado de forma paralela. Adicionalmente, visando o aumento de escalabilidade, estes sistemas de arquivos, em geral, apresentam um abordagem típica de

armazenamento definido por software (software-defined storage (SDS)), separando o gerenciamento dos dados e dos metadados dos arquivos entre serviços distintos: o *servidor de dados* e o *servidor de metadados*.

Os HDDs predominam como *dispositivos de armazenamento* nos SAPs dos ambientes de CAD. Porém, diferentes formas de organização podem ser encontradas. Estes podem estar conectados diretamente aos servidores (direct-attached storage (DAS)), conectados por uma rede de interconexão de alta velocidade (storage area network (SAN)), ou na forma de um servidor de arquivos (network-attached storage (NAS)) [Troppens et al. 2009]. Tanto SAN quanto NAS permitem que vários servidores compartilhem os dispositivos de armazenamento. Uma outra forma bastante comum de organização dos dispositivos de armazenamento em ambientes de CAD é o conjunto redundante de discos independentes (redundant array of independent disks (RAID)) [Chen et al. 1994]. Essa abordagem oferece paralelismo e redundância no nível de blocos e é complementar as abordagens DAS, NAS e SAN.

Apesar da sofisticação das infraestruturas de armazenamento, tanto em termos de hardware quanto de software, encontradas nos ambientes de CAD modernos, o desempenho de E/S observado pelas aplicações pode variar consideravelmente. Esta variação se deve a um grande número de variáveis que compõem estes ambiente complexos, incluindo os padrões de acesso e a carga de trabalho da aplicação, topologia e arquitetura do sistema, configurações nas múltiplas camadas da pilha de software de E/S, propriedades dos dispositivos físicos, interferências e ruídos, entre outros [Carns et al. 2009, Lofstead et al. 2010, Inacio et al. 2015a, Inacio et al. 2015b, Herbein et al. 2016, Inacio et al. 2017a]. Uma abordagem muito empregada em pesquisas focadas no desempenho de sistemas de E/S e armazenamento paralelos é a caracterização do impacto de diferentes variáveis no desempenho de E/S observado pelas aplicações [Inacio and Dantas 2014, Boito et al. 2018]. Tais trabalhos de pesquisa se utilizam de ferramentas especializadas para a avaliação de desempenho de sistemas de E/S e armazenamento paralelos, como MPI-TILE-IO [ANL 2002], INTERLEAVED-OR-RANDOM (IOR) [Shan et al. 2008] e IOR-EXTENDED (IORE) [Inacio and Dantas 2018b]. Mais detalhes sobre essas ferramentas serão apresentados na Seção 1.4.

1.2. Sistemas de Arquivos Paralelos

Sistemas de arquivos paralelos (SAPs) como o LUSTRE [Braam and Schwan 2002, OpenSFS 2018], ORANGEFS [Moore et al. 2011, Omnibond 2018], SPECTRUM SCALE [IBM 2018], CEPH [Weil et al. 2006], e PANASAS FILE SYSTEM (PANFS) [Nagle et al. 2004], são sistemas de arquivos distribuídos especializados, focados em alto desempenho de E/S e escalabilidade horizontal. Embora variações de projeto e implementação possam ser observadas entre os diferentes sistemas, em geral, os SAPs são compostos por três componentes principais: módulo cliente, servidor de dados e servidor de metadados. A Figura 1.2 ilustra a organização tradicional destes componentes.

O *módulo cliente* consiste basicamente de uma biblioteca ou um módulo de software executando nos nodos de computação ou de E/S, que implementa o protocolo de comunicação com o SAP. Em SAPs compatíveis com POSIX, por exemplo, o módulo cliente intercepta as requisições de E/S das aplicações e realiza as operações necessárias,

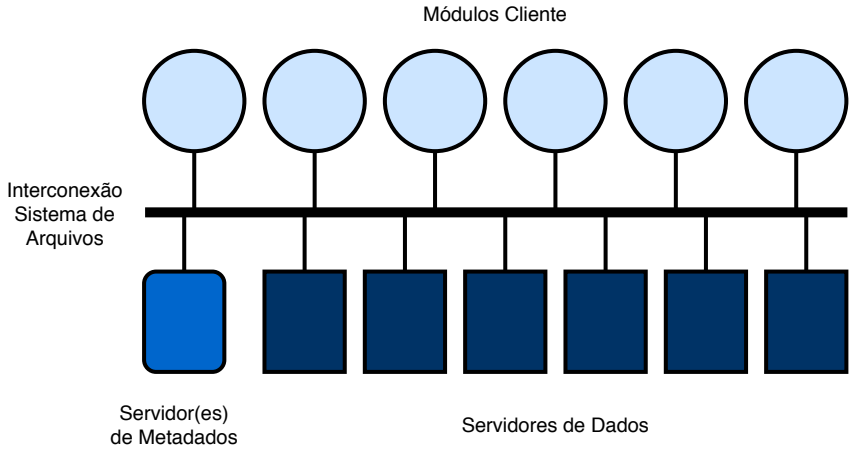


Figura 1.2. Principais componentes de um SAP genérico.

utilizando funções específicas do SAP, para o atendimento das requisições. O *servidor de metadados*, como o nome sugere, gerencia a estrutura de diretórios e mantém atualizadas as informações sobre os arquivos armazenados no sistema, como permissões, proprietário, datas de criação e atualização, localização, entre outras. Em algumas implementações de SAPs, o gerenciamento dos metadados é centralizado, com um único servidor ativo por sistema de arquivos, enquanto em outros, o gerenciamento dos metadados também é distribuído. Por fim, o conteúdo propriamente dito dos arquivos é mantido pelos *servidores de dados*. Cada servidor de dados, em geral, recebe apenas alguns fragmentos de cada arquivo, de forma que, para acessar todo o conteúdo de um arquivo, vários servidores de dados são acessados.

O processo de *particionamento e distribuição de arquivos* é uma das principais características de um SAP, que o diferencia de outros sistemas de arquivos distribuídos. A Figura 1.3 apresenta um exemplo do processo de particionamento e distribuição de um arquivo de 400 KiB em um SAP com cinco servidores de dados. É possível observar na figura que o arquivo estende-se ao longo de uma *faixa* de servidores de dados. Basicamente, dois parâmetros controlam o processo de particionamento e distribuição de arquivos em um SAP: a largura da faixa e o tamanho dos fragmentos da faixa. A *largura da faixa* (*stripe width*) define quantos servidores de dados serão utilizados para se armazenar o conteúdo de um arquivo. O *tamanho dos fragmentos da faixa* (*stripe size*) estabelece o tamanho do bloco contínuo de dados do arquivo que será armazenado em cada servidor.

No exemplo da Figura 1.3, uma largura de faixa igual a quatro e um tamanho de fragmento igual a 64 KiB foram utilizados. Primeiramente, observa-se que o arquivo é fragmentado em partes de tamanhos iguais, conforme o tamanho de fragmento estabelecido, com exceção do fragmento final, que apresenta os últimos 16 KiB do arquivo. O primeiro fragmento é armazenado no primeiro servidor, o segundo fragmento no segundo servidor, e assim por diante até que o quarto fragmento é armazenado no quarto servidor, atingindo o limite da largura de faixa. O quinto fragmento, então, é armazenado no primeiro servidor novamente, e o processo continua até que todos os fragmentos do arquivo

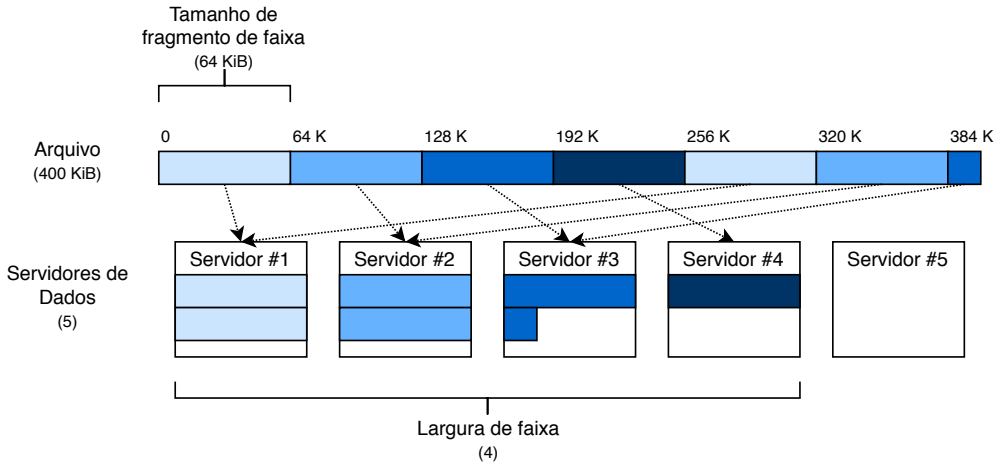


Figura 1.3. Exemplo de particionamento e distribuição de um arquivo de 400 KiB em um SAP com 5 servidores de dados, tamanho de fragmento de faixa de 64 KiB e largura de faixa igual a 4.

sejam armazenados. No caso deste arquivo ser estendido posteriormente para além dos 400 KiB, os próximos bytes seriam armazenados no *Servidor #3*, neste exemplo, até que os 48 KiB restantes do último fragmento fossem completados.

Apesar de a descrição sugerir uma sequencialidade de eventos, este mecanismo empregado pelos SAPs permite que operações de leitura e escrita possam ser realizadas de forma paralela, aumentando a vazão do sistema. Ainda sobre o exemplo da Figura 1.3, supondo uma requisição de leitura para 400 KiB do arquivo, cada servidor de dados enviaria, em paralelo, todos os fragmentos contidos nele. Em teoria, o tempo de resposta para tal requisição de leitura seria quatro vezes inferior ao tempo de resposta obtido com o arquivo armazenado em um único servidor de dados. Tal observação leva naturalmente à conclusão de que quanto maior a largura da faixa (mais servidores de dados), melhor o desempenho das requisições de E/S. Contudo, na prática, custos de comunicação com múltiplos servidores combinados ao overhead do processo de particionamento e distribuição fazem com que o desempenho estabilize a partir de uma determinada largura de faixa, fazendo com que a utilização de mais servidores de dados não resulte em incrementos de desempenho [Inacio et al. 2015a].

Dada esta característica prática relacionando a distribuição do arquivo com o desempenho das operações de acesso aos dados, layouts de distribuição costumam ser explorados em diferentes situações. Estes layouts de distribuição podem ser classificados basicamente em três grupos: horizontal, vertical e bidirecional [Song et al. 2012]. A Figura 1.4 ilustra a distribuição de quatro arquivos, com quatro fragmentos cada, segundo cada um destes três layouts de distribuição, em um SAP com quatro servidores de dados.

No *layout horizontal*, o arquivo é distribuído entre todos os servidores de dados disponíveis. Este layout de distribuição explora o paralelismo máximo oferecido pelo sistema. Em contrapartida, no *layout vertical*, cada arquivo é armazenado integralmente em um único servidor de dados. Neste layout, operações sobre um mesmo arquivo não são paralelizáveis. Contudo, se uma distribuição uniforme dos arquivos for realizada,

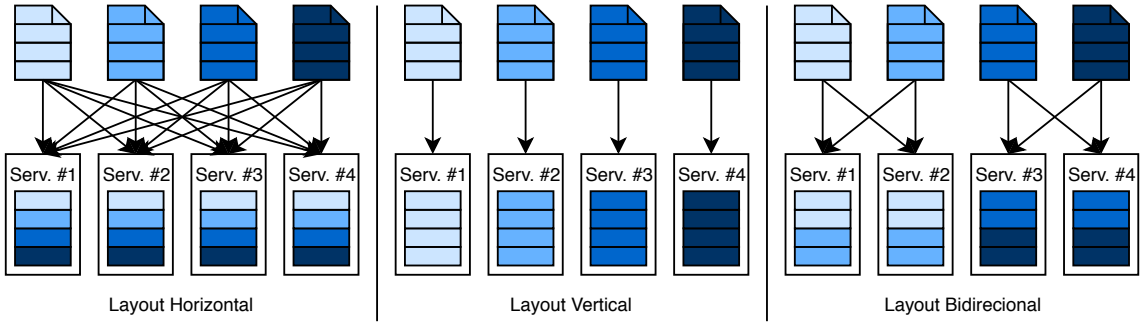


Figura 1.4. Layouts de distribuição de arquivos em SAPs (Adaptado de [Song et al. 2012]).

como no exemplo, o layout vertical pode reduzir a contenção provocada por diferentes processos acessando um mesmo servidor de dados. O *layout bidirecional* apresenta-se como um compromisso entre os layouts horizontal e vertical. No layout bidirecional, o arquivo é distribuído ao longo de uma fração dos servidores de dados, experimentando assim algum nível de paralelismo, ao mesmo tempo que a contenção entre processos em um mesmo servidor de dados é reduzida.

1.3. E/S em Aplicações Distribuídas

Quase que invariavelmente, uma aplicação distribuída de larga escala irá, em algum momento da sua execução, ler ou escrever dados em arquivos do sistema de armazenamento secundário. Isso significa que a aplicação deverá realizar operações, pelo menos, para *criar* ou *abrir* um arquivo, *ler* ou *escrever* uma quantidade de dados neste arquivo, e, finalmente, *fechar* o arquivo. Estas são as operações mais básicas, e comumente encontradas nas aplicações, com relação a E/S de dados em arquivos. Nesta seção serão discutidas diferentes maneiras de se realizar estas operações, considerando algumas das principais APIs utilizadas na área.

Para guiar essa discussão, será utilizado um estudo de caso baseado em um problema real. Considere um conjunto de dados Cartesiano, como o apresentado na Figura 1.5. Este conjunto de dados corresponde a um intervalo de tempo simulado pelo modelo NICAM, utilizado para o estudo de convecção atmosférica global de alta umidade em escala sub-quilométrica [Miyamoto et al. 2013]. As dimensões deste conjunto de dados tridimensional são de 11.520 pontos no eixo x, 5.760 pontos no eixo y e 94 pontos no eixo z. No conjunto de dados real, múltiplas variáveis são computadas para cada ponto. Neste estudo de caso, para fins de simplificação, consideraremos uma única variável de ponto flutuante com precisão simples (*float*), de 4 bytes, por ponto. Como resultado, tem-se um conjunto de dados com um tamanho total de aproximadamente 23 GiB (24.949.555.200 bytes). Adicionalmente, considere neste estudo de caso que o conjunto de dados será armazenado em um único arquivo.

A simulação na qual este conjunto de dados foi baseado, utilizou-se de todos os 82.944 nodos de computação do supercomputador japonês *K computer*, para simular 48 intervalos de tempo. Estes números mostram a necessidade de dividir o domínio do problema entre múltiplos processos para que a computação se torne praticável. Idealmente,

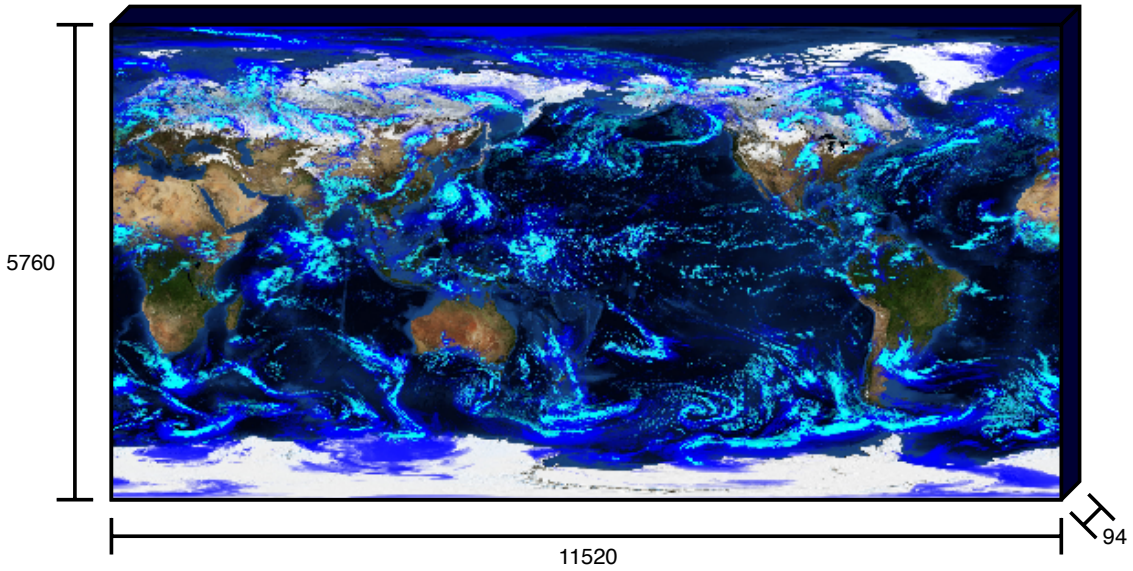


Figura 1.5. Exemplo de um conjunto de dados Cartesiano real extraído da saída do modelo de simulação NICAM [Miyamoto et al. 2013]. Este conjunto de dados apresenta 11.520 pontos no eixo x, 5.760 pontos no eixo y e 94 pontos no eixo z.

o domínio do problema é dividido igualmente entre os processos disponíveis, o que, por consequência, se traduziria em cada processo lendo ou escrevendo o mesmo volume de dados. Mesmo nestas condições, diferentes maneiras de se dividir o conjunto de dados entre os múltiplos processos podem resultar em variações significativas no desempenho de E/S da aplicação [Inacio et al. 2017b]. Neste estudo de caso, em favor da simplicidade, o domínio do problema será dividido entre 40 processos, sendo que cada um será responsável por uma seção cúbica do conjunto de dados com 1.152 pontos no eixo x, 1.440 pontos no eixo y e 94 pontos no eixo z, totalizando aproximadamente 595 MiB (623.738.880 bytes) de dados por processo.

1.3.1. POSIX

A especificação PORTABLE OPERATING SYSTEM INTERFACE (POSIX) [IEEE and The Open Group 2013] define funções padrão para sistemas operacionais, focando, principalmente, na portabilidade de software. Entre as funções definidas pelo padrão POSIX, existe um grande número voltado para permitir o acesso a dados em arquivos. Estas funções podem ser agrupadas de acordo com o mecanismo utilizado para representar a conexão entre a aplicação e o arquivo: descritores de arquivos e *stream*.

Funções baseadas em descritores de arquivos oferecem uma interface mais primitiva e de mais baixo nível para as aplicações. Por outro lado, funções baseadas em *stream* apresentam facilidades diferenciadas, como *buffering*, e são usualmente implementadas sobre funções mais básicas baseadas em descritores de arquivos. Além de especificadas na POSIX, as funções de E/S baseadas em *stream* são características do padrão da linguagem de programação C. Esta seção focará nas chamadas de sistema baseadas em descritores de arquivos da POSIX, enquanto a seção seguinte abordará funções de E/S do

padrão C baseadas em *stream*.

Primeiramente, para ter acesso a um arquivo no sistema de arquivos, a aplicação precisa estabelecer uma conexão com o mesmo. Para isso, utiliza-se a função `open()`. Esta função recebe como argumentos o caminho para o arquivo, que pode ser relativo ou absoluto, e *flags*, que indicam o estado e o modo de acesso do arquivo, como a *flag* `O_CREAT`, que define que, se o arquivo informado não existir no sistema de arquivos, este deve ser criado quando da chamada da função. O retorno da função `open()` é um número inteiro que identifica o descritor do arquivo. No estudo de caso em questão, cada um dos 40 processos precisa estabelecer a conexão com o arquivo antes de acessar sua porção do conjunto de dados.

Para ler e escrever dados no arquivo, a aplicação pode utilizar, respectivamente, as funções `read()` e `write()` da POSIX. Ambas recebem como argumentos o descritor do arquivo, um ponteiro para um endereço de memória contendo os dados para serem escritos ou alocado para receber os dados lidos, e o número de bytes a serem transferidos. Um detalhe particular que surge no contexto de aplicações distribuídas cujos processos acessam dados em um arquivo compartilhado, como no estudo de caso apresentado, é a definição de em qual região do arquivo estão os dados desejados.

Por padrão, a função `open()` define o marcador com a posição atual no arquivo para o início do arquivo. Portanto, se cada processo, por exemplo, escrever sua porção do conjunto de dados no arquivo compartilhado sem antes alterar este marcador, os dados serão sobrescritos a cada operação realizada pelos processos. Para redefinir este marcador, a POSIX oferece a função `lseek()`, que recebe como argumentos o descritor do arquivo, um *offset* (em bytes) e a posição a partir da qual o offset deve ser contabilizado. Vale ressaltar que o offset do arquivo é incrementado automaticamente toda vez que as funções `read()` e `write()` são invocadas, utilizando o número de bytes transferidos. Alternativamente, para evitar a necessidade de invocar as funções `lseek()` antes das leituras e escritas, a POSIX oferece as funções `pread()` e `pwrite()`, que recebem um argumento adicional com o offset a partir do início do arquivo onde a operação deve ser realizada e não movem o marcador do offset do arquivo ao final da execução. Esta facilidade é particularmente interessante quando a aplicação tem por característica realizar acessos a regiões não-contíguas do arquivo.

Independente da utilização das funções `lseek()` e `read()`/`write()` ou das funções `pread()`/`pwrite()`, o desenvolvedor da aplicação precisa garantir que cada processo acessará os offsets corretos no arquivo compartilhado, o que pode ser considerado uma atividade não trivial e propensa a erros. Primeiramente, é preciso considerar como os dados são armazenados no arquivo. Considerando este estudo de caso, por exemplo, onde o conjunto de dados é estruturado como um espaço Cartesiano tridimensional, uma convenção típica em aplicações escritas na linguagem C consiste em ordenar estes dados no arquivo por linhas (*row-major order*). No estudo de caso, isso se traduz em armazenar, para cada ponto do eixo x, todos os pontos do eixo y e, para cada ponto do eixo y, todos os pontos do eixo z, conforme ilustrado na Figura 1.6. Desta forma, observa-se que o conhecimento sobre a estrutura do domínio do problema é fundamental para a coordenação do acesso distribuído e concorrente aos dados. Em posse de um referencial do processo dentro da aplicação distribuída, como o *rank* MPI, por exemplo, é possí-

vel se definir uma fórmula para se calcular os offsets para cada operação de E/S de cada processo.

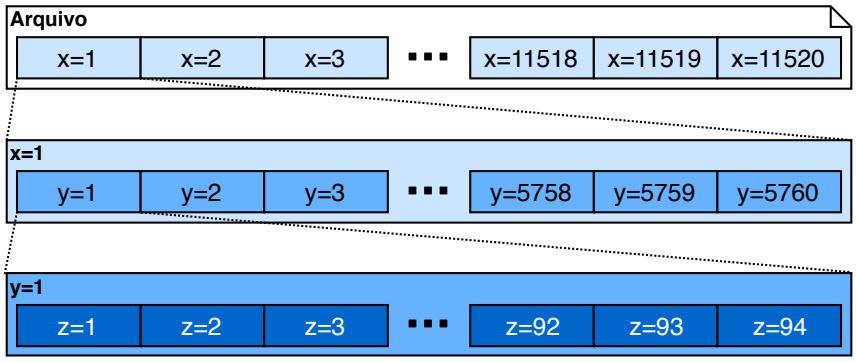


Figura 1.6. Serialização do conjunto de dados Cartesiano do estudo de caso em um arquivo.

Finalmente, uma vez transferidos todos os dados de ou para o arquivo, a aplicação fecha a conexão com o arquivo utilizando a chamada de sistema POSIX `close()`, passando como único argumento o descritor do arquivo. Assim como no estabelecimento da conexão, todos os 40 processos, considerando este estudo de caso, devem fechar a conexão com o arquivo ao final da sua utilização. Esta é uma boa prática que visa evitar diversos problemas, entre eles, o bloqueio de arquivos.

1.3.2. Padrão C (Stream de E/S)

Como mencionado anteriormente, o padrão da linguagem de programação C define algumas funções de alto nível para E/S em arquivos baseadas em *stream* [Free Software Foundation 2018]. Estas funções são usualmente implementadas sobre funções de mais baixo nível, como as chamadas de sistema POSIX baseadas em descritores de arquivos apresentadas na seção anterior. A interface baseada em *stream* apresenta algumas vantagens, principalmente para operações de E/S que realizam transferência de dados.

Além da oferta de funções mais elaboradas, como o suporte para formatação de E/S, por exemplo, as funções baseadas em *stream* utilizam um mecanismo de *buffering* para transferência de caracteres. Durante operações de escrita, os caracteres são acumulados e transferidos assincronamente para o arquivo em blocos, ao invés de aparecer imediatamente após executada a função. De maneira similar, em operações de leitura, os dados do arquivo são recuperados em blocos e não caractere por caractere [Free Software Foundation 2018]. Esta facilidade visa, principalmente, minimizar o impacto do acesso ao sistema de armazenamento secundário, cujo desempenho é consideravelmente inferior ao das memórias principais.

A função `fopen()` estabelece a conexão com um arquivo, cujo caminho é passado como primeiro argumento, e retorna o *stream* associado com esta conexão. O segundo argumento desta função corresponde ao modo de acesso. Dependendo do modo de acesso, que inclui somente leitura, somente escrita, leitura e escrita e acréscimo ao final do arquivo (*append*), o marcador com a posição do arquivo pode variar entre o início e o final do arquivo. Sendo assim, como foi observado na seção anterior, é necessário contro-

lar o marcador da posição do arquivo de forma que os diferentes processos da aplicação distribuída acessem a sua região específica do conjunto de dados adequadamente. Para isso, o padrão C disponibiliza as funções `fseek()` e `rewind()`. A função `fseek()` é equivalente a chamada de sistema POSIX `lseek()`, com a exceção de que o primeiro argumento é um *stream* e não um descritor de arquivo. A função `rewind()` é um forma simplificada da `fseek()` que recebe como argumento apenas o *stream* e redefine o marcador da posição do arquivo para o seu início.

As funções `fread()` e `fwrite()` possibilitam, respectivamente, ler e escrever dados em um arquivo. Ambas as funções recebem como argumentos um endereço de memória apontando para onde estão os dados a serem escritos ou para onde os dados a serem lidos devem ser colocados; o tamanho (em bytes) de cada item a ser lido ou escrito, por exemplo, um byte no caso de um caractere; o número de itens; e o *stream* do arquivo. Ao final da execução, ambas as funções movem o marcador da posição no arquivo de acordo com a quantidade de dados acessada. Não há no padrão C funções equivalentes às chamadas de sistema POSIX `pwrite()` e `pread()`, que permitem passar a posição no arquivo onde a operação deve ser realizada como argumento. Portanto, para estabelecer o padrão de acesso não-contíguo, como o demandado pelo estudo de caso apresentado, é necessário realizar duas operações: uma de posicionamento no arquivo, com `fseek()`, e outra para acessar os dados, com `fread()` ou `fwrite()`.

Assim como discutido anteriormente, ao final do acesso aos dados do arquivo, a aplicação deve encerrar a conexão com o mesmo. A função `fclose()` é responsável por concluir a transferência de todos os dados acumulados no *buffer* para o arquivo e fechar a conexão com o *stream* passado como único argumento. A transferência dos dados acumulados no *buffer* pode ser realizada antes do fechamento do arquivo, utilizando a função `fflush()`, que recebe o *stream* do arquivo como argumento. Esta função força a aplicação a aguardar até que todos os dados em *buffer* sejam transferidos para o armazenamento secundário, o que pode resultar em uma degradação de desempenho.

1.3.3. MPI-IO

MPI-IO é uma especificação de interface para E/S paralela em arquivos definida a partir da segunda versão do padrão MESSAGE PASSING INTERFACE (MPI) [MPI Forum 1997]. Esta especificação define um amplo conjunto de funções e facilidades projetadas para prover maior eficiência nas operações de E/S realizadas por aplicações distribuídas, principalmente, àquelas baseadas em MPI. Localizada normalmente numa camada intermediária, entre o SAP e a aplicação, MPI-IO é frequentemente referido como um *middleware* de E/S [Prabhat and Koziol 2014].

MPI-IO suporta os três métodos fundamentais de se realizar E/S em arquivos em uma aplicação distribuída [Prabhat and Koziol 2014], ilustradas na Figura 1.7. No primeiro deles, conhecido como *arquivo por processo*, cada processo escreve em um arquivo independente. Apesar de ser um método consideravelmente simples e de oferecer alto paralelismo, apresenta como desvantagem o grande número de potencialmente pequenos arquivos. O segundo método visa resolver este problema, *concentrando* todos os dados em um único processo, que os transfere para um único arquivo. Evidentemente, além de não explorar nenhum tipo de paralelismo, esta não é uma abordagem escalável.

Congestionamentos de comunicação podem ter um impacto negativo no desempenho das aplicações devido ao grande número de processos, assim como a capacidade de memória em um único processo poder não ser suficiente para acumular os dados enviados pelos demais. O terceiro método, particularmente explorado no MPI-IO, busca prover uma solução para as questões anteriores. Neste método, todos os processos acessam dados em um *arquivo compartilhado* de maneira coordenada. O restante desta seção se dedicará a prover maiores detalhes sobre este método no MPI-IO.

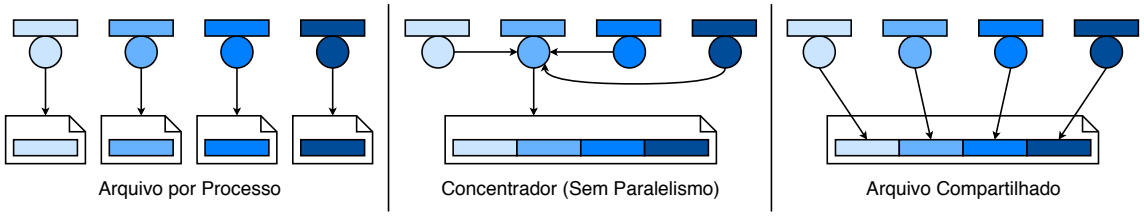


Figura 1.7. Três métodos fundamentais para se realizar E/S em uma aplicação distribuída.

Existem dois grupos de funções de E/S no MPI-IO: independentes e coletivas [MPI Forum 1997]. Apesar de serem muito similares na definição, as funções pertencentes a cada um destes dois grupos têm uma diferença fundamental. As funções de E/S *independentes* podem ser executadas por um processo sem depender dos demais processos da aplicação distribuída, similar ao que ocorre com as chamadas de sistema POSIX e com a funções de *stream* de E/S do padrão C. Por outro lado, as funções de E/S *coletivas* requerem que todos os processos vinculados a um comunicador MPI, executem a mesma função para que a aplicação possa prosseguir.

1.3.3.1. E/S Independente

Como mencionado anteriormente, a E/S independente no MPI-IO se assemelha na estrutura e aparência à E/S POSIX e do padrão C. Para mover o marcador de posição no arquivo, utiliza-se a função `MPI_File_seek()`, que recebe como argumentos a referência para o arquivo aberto, um offset (em bytes) e a posição a partir da qual o offset deve ser contabilizado. Vale ressaltar que esta função move apenas o marcador na referência do arquivo para o processo invocador, sem afetar os demais processos. As funções `MPI_File_read()` e `MPI_File_write()` realizam, respectivamente, a leitura e escrita de dados em um arquivo. Os argumentos para ambas as funções são: a referência para o arquivo aberto, um endereço de memória apontando para ou de onde os dados devem ser transferidos, o quantidade de elementos na memória, o tipo do elemento (caractere, inteiro, tipo customizado, etc.) e um objeto `MPI_Status` (que pode ser ignorado passando `MPI_STATUS_IGNORE`), que armazena dados referentes a comunicação realizada durante a execução da função.

MPI-IO oferece alternativas para funções de leitura e escrita em que o offset do arquivo onde a operação deve ser executada é passado como argumento adicional: `MPI_File_read_at()` e `MPI_File_write_at()`. Estas funções evitam a realização de duas chamadas de funções para se realizar uma leitura e escrita, uma para mover

a posição no arquivo e outra para acessar os dados. Porém, vale lembrar que o marcador da posição do arquivo não é atualizado ao final da execução destas funções.

As funções para abertura (criação) e fechamento de arquivos no MPI-IO, obrigatórias antes e depois, respectivamente, do acesso aos dados de um arquivo, são *coletivas* sobre o comunicador MPI. Isto significa que todos os processos vinculados ao comunicador MPI passado como argumento para as funções de abertura (criação) e fechamento de um arquivo, precisam necessariamente invocar a referida função para que a aplicação possa prosseguir. Por esta razão, estas funções serão discutidas na próxima seção, junto com as demais funções de E/S coletivas do MPI-IO.

1.3.3.2. E/S Coletiva

Para estabelecer uma conexão com um arquivo, o MPI-IO disponibiliza a função coletiva `MPI_File_open()`. Esta função recebe como argumentos um comunicador MPI, que estabelece os processos participantes da operação coletiva; o caminho do arquivo; o modo de abertura, que permite definir, entre outras opções, que um arquivo seja criado caso este não exista; um objeto `MPI_Info`, que permite passar informações adicionais para o MPI-IO, possibilitando otimizações mais específicas; e um endereço de memória para um objeto `MPI_File`, onde será armazenada a referência do arquivo aberto. O objeto `MPI_File` é utilizado pelas demais funções de E/S do MPI-IO, sejam elas coletivas ou independentes, de maneira similar ao descritor de arquivo no POSIX ou o *stream* nas funções de E/S do padrão C.

As funções de leitura e escrita coletivas do MPI-IO são idênticas em termos da lista de argumentos às suas respectivas funções independentes. Os nomes destas funções coletivas apresentam uma pequena diferença, com o acréscimo do sufixo `_all`: `MPI_File_read_all()`,`MPI_File_write_all()`,`MPI_File_read_at_all()` e `MPI_File_write_at_all()`. Estas funções, quando invocadas dentro da aplicação distribuída, precisam ser feitas por todos os processos vinculados ao comunicador MPI da abertura do arquivo para que a operação seja realizada e a aplicação possa prosseguir sua execução.

Mesmo tratando-se de funções coletivas, vale notar que cada processo pode acessar uma região diferente do arquivo. Basta utilizar a função `MPI_File_seek()` para mover o marcador da posição no arquivo antes das funções `MPI_File_read_all()` e `MPI_File_write_all()`, ou utilizando diretamente as funções que recebem o offset como argumento `MPI_File_read_at_all()` e `MPI_File_write_at_all()`. Esta é uma característica de particular interesse, por exemplo, para o estudo de caso apresentado, onde cada um dos 40 processos acessa determinados pontos dentro do conjunto de dados armazenados em offsets específicos do arquivo compartilhado.

A função `MPI_File_close()`, que recebe como argumento único a referência para o arquivo (objeto `MPI_File`), fecha a conexão com o arquivo. Esta função, assim como as demais funções coletivas mencionadas anteriormente, precisa ser chamada por todos os processos vinculados ao comunicador MPI utilizado no momento da abertura do arquivo compartilhado. Percebe-se que, independente da complexidade e das facilidades da API utilizada, o fluxo para acesso aos dados é o mesmo. O arquivo deve ser aberto

antes de ser acessado, seja para leitura ou escrita, e fechado ao final do acesso.

As funções de leitura e escrita do MPI-IO, sejam elas independentes ou coletivas, apresentadas até este ponto foram discutidas em termos de offsets específicos. Entre outras palavras, considerando múltiplos processos acessando um arquivo compartilhado, como no estudo de caso apresentado, cada processo precisa mover o marcador para a posição adequada no arquivo antes de fazer a leitura ou escrita, ou utilizar uma função que aceite o offset no arquivo como argumento. Esta abordagem de relativo baixo nível, suportada por outras APIs já discutidas como POSIX e do padrão C, é bastante flexível, uma vez que permite controle total sobre a região do arquivo a ser acessada. Por outro lado, em cenários como o do estudo de caso apresentado, em que um conjunto de dados Cartesiano tridimensional é dividido em subconjuntos, também tridimensionais, para múltiplos processos, que precisam acessá-lo em um arquivo compartilhado, a representação das regiões visíveis para cada processo através de um conjunto de offsets não é natural e pode ser tornar uma tarefa desafiadora e propensa a erros. Visando endereçar esta questão, o MPI-IO oferece um mecanismo para possibilitar uma representação mais natural de acessos não-contíguos em um arquivo.

1.3.3.3. Acesso Não-contíguo

Duas facilidades particulares do MPI e do MPI-IO favorecem a programação de acesso a dados não-contíguos em arquivos: tipos de dados customizados e visão de arquivo. Todas as funções de leitura e escrita do MPI-IO descritas nas seções anteriores realizam a transferência de dados baseando-se no tipo do elemento (dado) e na quantidade de elementos. O MPI-IO oferece um conjunto de tipos de dados básicos, como caracteres, inteiros, número de ponto flutuante, entre outros, para descrever estes elementos. Porém, em algumas situações, o elemento pode ser de um tipo mais complexo, composto por vários tipos básicos, como um `struct` do C, por exemplo. Para possibilitar a descrição deste tipo de elemento, o MPI-IO oferece um conjunto de funções para definição de *tipos de dados customizados*. A função `MPI_Type_create_struct()`, por exemplo, permite a definição de um tipo de dado customizado para representar uma `struct` que a aplicação pretende armazenar ou recuperar de um arquivo utilizando uma única operação de escrita ou leitura. Um detalhe importante é que, uma vez criado o tipo, este precisa ser registrado com a função `MPI_Type_commit()` antes de poder ser utilizado pela aplicação.

A *visão de arquivo* é uma facilidade que visa possibilitar a definição, de maneira mais natural, de quais regiões do arquivo compartilhado são visíveis para cada processo. Isto inclui a possibilidade de definir várias regiões não-contíguas no arquivo. A visão de um arquivo é definida após a sua abertura, utilizando a função `MPI_File_set_view()`. Esta função recebe como argumentos a referência do arquivo aberto, um deslocamento (em bytes), contado a partir do início do arquivo; o tipo de dados elementar, correspondente ao menor elemento que compõe a região, que pode ser tanto um tipo básico quando um tipo customizado do MPI; um tipo de dados representando o arquivo, que estabelece quais regiões do arquivo são visíveis para o processo e que deve ser o mesmo tipo de dados elementar ou derivado deste; o tipo de representação de dados; e um objeto `MPI_Info` com informações adicionais para auxiliar em otimizações mais específicas.

Uma vez definida a visão do arquivo para o processo, este poderia acessar, utilizando uma única chamada para uma função de leitura e escrita, toda a sua parte do conjunto de dados, mesmo que esta seja composta por regiões não-contíguas.

A visão do arquivo provê uma interpretação global do tipo de acesso que os processos de uma aplicação distribuída podem realizar em um arquivo compartilhado. Esta interpretação, combinada à coordenação oferecida pelas funções coletivas do MPI-IO, permite que diferentes otimizações sejam empregadas de forma que operações de leitura e escrita sejam realizadas de maneira mais eficiente e, por consequência, em um menor intervalo de tempo. Entre as otimizações mais populares estão a *data sieving* e a *two-phase I/O*. A técnica de *data sieving* [Thakur et al. 2002] consiste de acessar uma região do arquivo maior do que a requisitada pelo processo e, em memória, extrair os dados de fato solicitados. Esta técnica visa evitar a realização de um grande número de acessos a pequenas regiões de um arquivo, um padrão de acesso reconhecidamente ruim do ponto de vista de desempenho de E/S em SAPs. Na técnica de *two-phase I/O* [Thakur et al. 1999], operações coletivas de leitura ou escrita são divididas em duas fases, uma de agregação e outra de acesso ao arquivo, conforme ilustrado na Figura 1.8 para o estudo de caso apresentado.

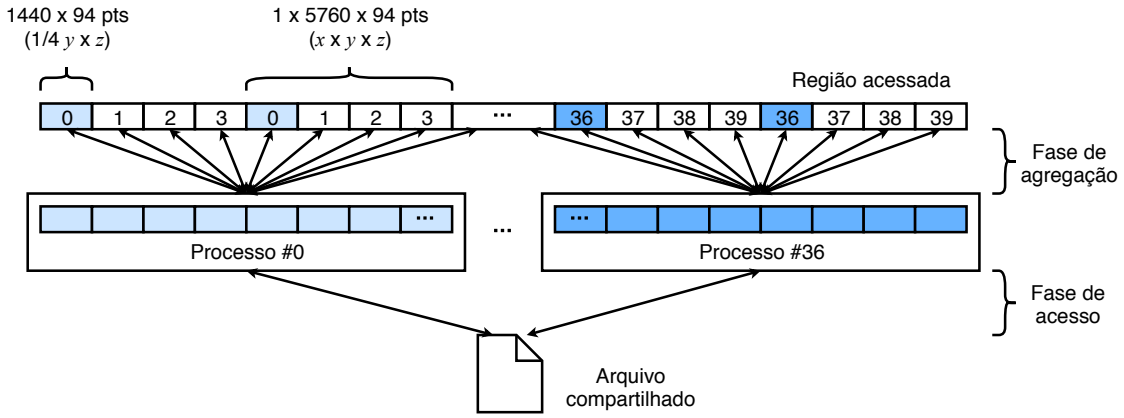


Figura 1.8. Exemplo da técnica de otimização *two-phase I/O* aplicada a uma operação de E/S coletiva do MPI-IO no conjunto de dados do estudo de caso.

Considere, por exemplo, uma operação de escrita coletiva, onde uma visão do arquivo é definida envolvendo os 40 processos do estudo de caso, cada um enviando para um arquivo compartilhado a sua porção tridimensional do conjunto de dados. Na *fase de agregação*, um subconjunto dos processos participantes atuam como agregadores. No exemplo da Figura 1.8, um a cada quatro processos são agregadores (ex.: processos 0 e 36). Cada agregador recebe de outros processos dados pertencentes a outras regiões adjacentes ou próximas a própria região acessada pelo agregador. Uma vez tendo agregado os dados de outros processos, formando uma região contígua maior que a região que seria acessada individualmente por cada processo, inicia-se a *fase de acesso*, em que os agregadores realizam a transferência dos dados agregados para o arquivo compartilhado. Numa leitura coletiva, o processo basicamente se inverte, ocorrendo primeiro a fase de acesso e depois a fase de distribuição (em oposição a fase de agregação), em que os dados seriam

encaminhados para cada processo solicitante de acordo com a região acessada por estes.

1.4. Ferramentas de Avaliação de Desempenho

Embora diferentes APIs e funções de E/S, das mais simples às mais sofisticadas, estejam a disposição, encontrar a alternativa mais eficiente em termos de desempenho de acesso a grandes conjuntos de dados continua sendo um desafio. Isto porque o desempenho de E/S em aplicações distribuídas de larga escala que fazem uso intensivo de dados depende de um grande número de fatores, incluindo aspectos tanto da infraestrutura de armazenamento quanto da pilha de software de E/S [Carns et al. 2009, Lofstead et al. 2010, Inacio et al. 2015a, Inacio et al. 2015b, Herbein et al. 2016, Inacio et al. 2017a]. Muitas vezes, diferentes cenários, construídos a partir da combinação de diferentes métodos de E/S e parâmetros do sistema de armazenamento, são avaliados através de experimentos para se poder identificar qual destes se mostra mais eficiente [Boito et al. 2018]. Para auxiliar neste processo, diversas ferramentas especializadas na reprodução de padrões de acesso a dados em arquivos e avaliação de desempenho de E/S foram desenvolvidas. Nesta seção, serão apresentadas algumas das ferramentas mais populares na comunidade de pesquisa em E/S paralela.

1.4.1. mpi-tile-io

MPI-TILE-IO [ANL 2002] é uma ferramenta para avaliação de desempenho de E/S paralela em acessos não-contíguos utilizando funções de E/S do MPI-IO. O MPI-TILE-IO reproduz um conjunto específico de dados: uma matriz densa bidimensional. Cada processo participante é responsável por uma parte, denominada *tile*, desta matriz e a escreve em um arquivo compartilhado por meio de funções independentes ou coletivas do MPI-IO. Este tipo de carga de trabalho é bastante comum entre aplicações de visualização e de simulação científicas e de engenharia.

Embora o MPI-TILE-IO gere dados apenas segundo um layout de matriz bidimensional, uma lista de parâmetros é oferecida para customização do conjunto de dados gerado, como demonstrado na Figura 1.9:

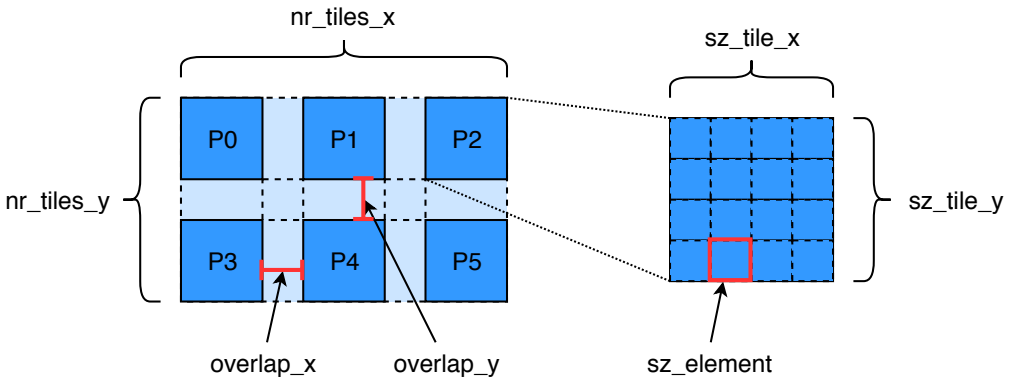


Figura 1.9. Exemplo de um conjunto de dados gerado pelo MPI-TILE-IO, destacando seus parâmetros de execução.

- *nr_tiles_x*: número de *tiles* no eixo *x* da matriz;

- *nr_tiles_y*: número de *tiles* no eixo *y* da matriz;
- *sz_tile_x*: número de elementos no eixo *x* de cada *tile*;
- *sz_tile_y*: número de elementos no eixo *y* de cada *tile*;
- *sz_element*: tamanho de um elemento em bytes;
- *overlap_x*: número de elementos compartilhados entre dois *tiles* adjacentes no sentido do eixo *x*;
- *overlap_y*: número de elementos compartilhados entre dois *tiles* adjacentes no sentido do eixo *y*.

A simplicidade do projeto e desenvolvimento do MPI-TILE-IO, combinada a representatividade da carga de trabalho gerada, contribuíram para sua utilização na comunidade de pesquisa em E/S paralela. Contudo, a sua especificidade limita sua utilização em estudos mais amplos, para investigação de diferentes cargas de trabalho e métodos de E/S. Por esta razão, muitos trabalhos de pesquisa se utilizam de outros benchmarks de E/S em conjunto com o MPI-TILE-IO. Visando concentrar esta demanda de flexibilidade, outras ferramentas foram propostas.

1.4.2. Interleaved-or-Random (IOR)

O INTERLEAVED-OR-RANDOM (IOR) [LLNL 2013, Shan et al. 2008] é um benchmark de E/S paralela que permite reproduzir um amplo e variado conjunto de cargas de trabalho. Por meio de parâmetros relativamente simples, o IOR permite avaliar o desempenho de diferentes APIs de E/S, como POSIX, MPI-IO, HDF5 e PNETCDF. Estes parâmetros permitem também controlar características das cargas de trabalho geradas, como acessos a dados contíguos e não-contíguos, em arquivos compartilhados ou individuais, quantidade de dados e tamanho de requisições, entre outros. Esta flexibilidade oferecida pelo IOR a transformou em um padrão "de facto" na comunidade de pesquisa em E/S paralela, sendo uma das ferramentas mais utilizadas para avaliação de desempenho de E/S paralela na área de CAD [Boito et al. 2018].

Na versão 3.0.1 do IOR, cerca de 50 parâmetros são disponibilizados. A Figura 1.10 apresenta três dos principais parâmetros, utilizados para a definição da carga de trabalho gerada por cada processo do IOR. O parâmetro *transferSize* estabelece o tamanho (em bytes) de cada operação de E/S realizada, seja ela leitura ou escrita. O parâmetro *blockSize* define o tamanho (em bytes) de um bloco de dados a ser acessado. Na prática, este parâmetro estabelece o número de operações de E/S que serão realizadas, uma vez que seu valor deve ser um múltiplo do parâmetro *transferSize*. Em um cenário de acesso sequencial, como o do exemplo da Figura 1.10, cada processo acessa uma região contígua de tamanho *blockSize*, realizando operações de E/S de tamanho *transferSize*. A região contígua que compreende os blocos acessados por todos os processos é denominada de *segmento*. O parâmetro *segmentCount* define quantos segmentos existem no arquivo. No caso do exemplo da Figura 1.10, há dois segmentos. Os segmentos permitem representar padrões de acesso não-contíguos, onde cada processo acessa blocos de dados separados por intervalos regulares.

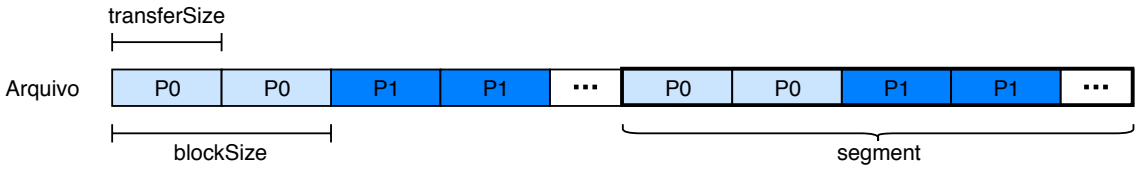


Figura 1.10. Estrutura de um arquivo compartilhado por múltiplos processos gerado pelo IOR, destacando alguns dos parâmetros oferecidos para definição da carga de trabalho.

Apesar das inúmeras facilidades providas pelo IOR, este apresenta algumas limitações. Como mencionado, as cargas de trabalho geradas pelo IOR são obrigatoriamente homogêneas, no sentido que todos os processos acessam a mesma quantidade de dados, utilizando operações também de mesmo tamanho. Além disso, a facilidade de se definir segmentos, oferecida pelo IOR, não é suficiente para se representar precisamente algumas cargas de trabalho usualmente encontradas em aplicações de CAD, como, por exemplo, algumas matrizes bidimensionais, como as geradas pelo MPI-TILE-IO. Estas e outras limitações, aliadas a descontinuidade no desenvolvimento do IOR, motivaram a proposta de uma ferramenta alternativa.

1.4.3. IOR-Extended (IORE)

O IOR-EXTENDED (IORE) [Inacio and Dantas 2018b, Inacio and Dantas 2018a] foi originalmente projetado como uma extensão do benchmark IOR. O foco inicial era endereçar algumas limitações particulares do IOR, como a homogeneidade da carga de trabalho, por exemplo. Porém, mais do que um gerador de carga de trabalho, o IORE evoluiu para uma ferramenta diferenciada para avaliação de desempenho de E/S paralela e de sistemas de armazenamento distribuído. Um conjunto de novas funcionalidades foi incluído no projeto do IORE visando não apenas ampliar e aprimorar aspectos relacionados a geração de carga de trabalho, mas também agilizar a realização de experimentos e análises de desempenho, e aumentar a reprodutibilidade dos resultados.

Uma das principais funcionalidades introduzidas no IORE é a *execução orientada por experimento*. Basicamente, toda a execução do IORE consiste de um *experimento*, cuja estrutura é apresentada na Figura 1.11. Cada experimento consiste de uma ou mais *rodadas*, que, por sua vez, consiste de um conjunto de parâmetros de configuração que define as características de um teste. Rodadas podem ser *repetidas* consecutivamente, assim como experimentos podem ser *replicados* como um todo, mantendo a ordem original de rodadas ou executando as rodadas em ordem aleatória. Esta estrutura provê flexibilidade para o usuário, que pode utilizar o IORE tanto para avaliações de desempenho rápidas, quanto para estudos mais complexos, envolvendo a análise de múltiplas variáveis.

Outro aprimoramento introduzido no IORE é o suporte para definição de *cargas de trabalho baseadas em conjuntos de dados*. Os parâmetros oferecidos pelo IOR, como apresentado na seção anterior, permitem definir a carga de trabalho gerada sob uma perspectiva de volume de dados e offsets. Embora flexível, pode-se observar que determinados conjuntos de dados típicos de aplicações de CAD não são precisamente representadas utilizando-se tais parâmetros. Através de parâmetros simples, porém específicos por tipo

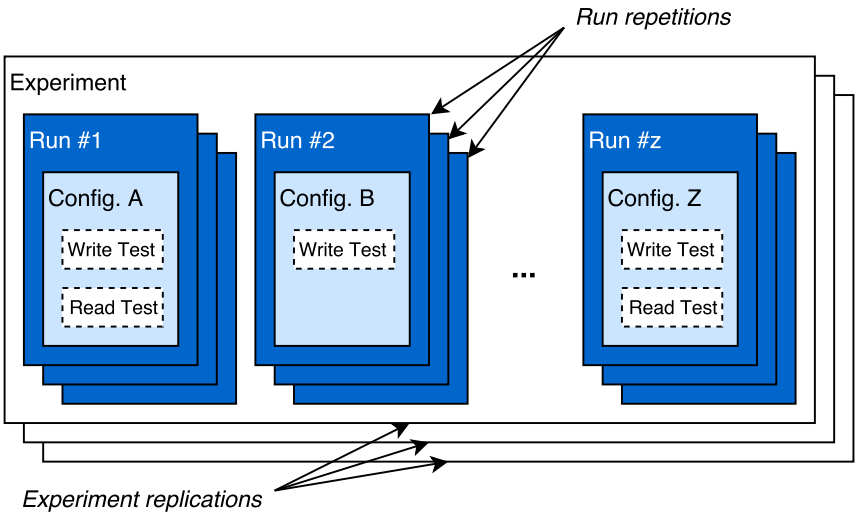
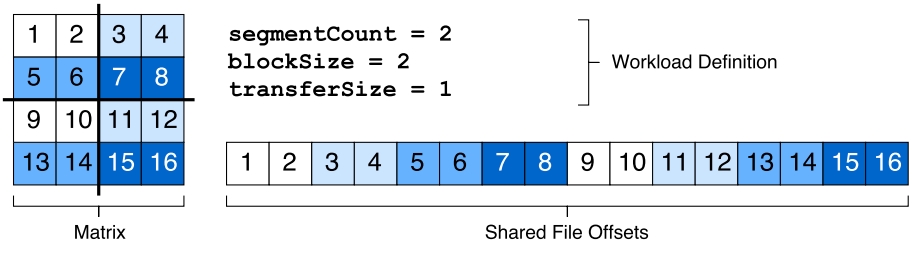
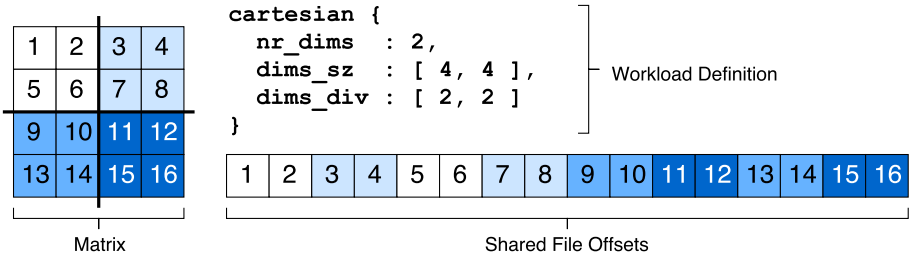


Figura 1.11. Estrutura de um experimento no IORE.

de conjunto de dados, o IORE permite que tais cargas de trabalho possam ser geradas, conforme ilustrado no exemplo da Figura 1.12. Neste exemplo, a proposta é que uma matriz bidimensional de dimensões 4x4 seja dividida entre quatro processos, de forma que cada um atue sobre uma submatriz 2x2. Na Figura 1.12a, observa-se que, utilizando os parâmetros oferecidos pelo IOR, a carga de trabalho não reflete exatamente a proposta. Apesar de cada processo acessar o volume de dados correto, os offsets acessados não são os esperados. O modelo de definição de conjunto de dados Cartesianos oferecido pelo IORE, como demonstrado na Figura 1.12b, permite representar precisamente a carga de trabalho esperada para este exemplo.



(a) IOR



(b) IORE

Figura 1.12. Representação de uma matriz bidimensional usando o IOR e o IORE.

Além de suportar a geração de cargas de trabalho homogêneas, assim como no IOR, o IORE oferece também suporte para cargas de trabalho heterogêneas. Listas de quantidades de dados e tamanho de requisições podem ser passadas como parâmetro, assim como distribuições de probabilidade (ex.: normal, uniforme, geométrica), de forma que cada processo possa gerar uma carga de trabalho diferente dos demais. O IORE oferece ainda parâmetros para manipulação de opções de configuração do sistema de arquivos paralelo, como tamanho de faixa, por exemplo, em tempo de execução; e exportação das medidas de desempenho coletadas em arquivo CSV ao final da execução, evitando a necessidade de tratamento de informações enviadas para saída padrão.

O projeto e desenvolvimento do IORE faz parte de um trabalho de pesquisa em andamento. Porém, resultados preliminares observados com sua utilização têm se mostrado promissores [Inacio and Dantas 2018b, Inacio and Dantas 2018a]. Espera-se, com a evolução da pesquisa, que o IORE possa auxiliar trabalhos na área de E/S e armazenamento paralelo de larga escala, não apenas no domínio de aplicações de CAD, mas também em aplicações de Big Data e Internet of Things. O código fonte da ferramenta está disponível gratuitamente no repositório do Laboratório de Pesquisa em Sistemas Distribuídos (LAPESD) no Github (`http://github.com/lapesd/iore`).

1.5. Considerações Finais

A proposta deste minicurso foi apresentar uma introdução a sistemas de E/S e armazenamento paralelos voltadas para ambientes de computação de alto desempenho (CAD). Mostrou-se que, cada vez mais, é preciso se preocupar com a forma como grandes conjuntos de dados são recuperados e armazenados por aplicações distribuídas de larga escala, visto que o acesso a estes dados pode ocupar um tempo significativo da execução da aplicação. Um visão geral da infraestrutura física e da pilha de software de E/S paralela tipicamente encontrada em ambientes de CAD modernos foi apresentada, identificando as principais funções de cada elemento e camada.

Em seguida, as principais características de sistemas de arquivos paralelos (SAPs) foram apresentadas. Estes sistemas, responsáveis por prover o sistema de armazenamento secundário em ambientes de grande porte, como clusters de larga escala e supercomputadores, são projetados para oferecer alta escalabilidade, em termos tanto de capacidade quanto de desempenho, e suportar acessos altamente concorrentes. A E/S paralela, da perspectiva das aplicações, também foi explorada neste minicurso. O fluxo básico para acesso a dados em um arquivo compartilhado por múltiplos processos de uma aplicação distribuída foi discutido considerando-se três APIs amplamente utilizadas: chamadas de sistema POSIX, funções do padrão C e funções independentes e coletivas do MPI-IO. Por fim, algumas das mais populares ferramentas para geração de carga de trabalho e avaliação de desempenho de E/S foram apresentadas. Estas ferramentas visam, em geral, auxiliar pesquisadores, usuários, desenvolvedores e administradores de sistemas a otimizar o desempenho da E/S em suas aplicações e ambientes.

Referências

[ANL 2002] ANL (2002). Parallel I/O Benchmarking Consortium. `http://www.mcs.anl.gov/research/projects/pio-benchmark/`.

[Baker et al. 2014] Baker, A. H., Xu, H., Dennis, J. M., Levy, M. N., Nychka, D., and Mickelson, S. A. (2014). A methodology for evaluating the impact of data compression on climate simulation data. In *HPDC '14 Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 203–214. ACM Press.

[Bell et al. 2009] Bell, G., Hey, T., and Szalay, A. (2009). Beyond the Data Deluge. *Science*, 323(5919):1297–1298.

[Boito et al. 2018] Boito, F. Z., Inacio, E. C., Bez, J. L., Navaux, P. O. A., Dantas, M. A. R., and Denneulin, Y. (2018). A Checkpoint of Research on Parallel I/O for High-Performance Computing. *ACM Computing Surveys*, 51(2):1–35.

[Braam and Schwan 2002] Braam, P. J. and Schwan, P. (2002). Lustre: The intergalactic file system. In *OLS '02 Proceedings of the Ottawa Linux Symposium*, pages 50–54.

[Carns et al. 2009] Carns, P., Latham, R., Ross, R., Iskra, K., Lang, S., and Riley, K. (2009). 24/7 Characterization of petascale I/O workloads. In *CLUSTER '09 Proceedings of the IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE.

[CERN 2016] CERN (2016). Processing: What to record? `http://home.cern/about/computing/processing-what-record`.

[Chen et al. 2009] Chen, J. H., Choudhary, A., de Supinski, B., DeVries, M., Hawkes, E. R., Klasky, S., Liao, W. K., Ma, K. L., Mellor-Crummey, J., Podhorszki, N., Sankaran, R., Shende, S., and Yoo, C. S. (2009). Terascale direct numerical simulations of turbulent combustion using S3D. *Computational Science & Discovery*, 2(1):1–31.

[Chen et al. 1994] Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. (1994). RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185.

[Dorier et al. 2016] Dorier, M., Sisneros, R., Gomez, L. B., Peterka, T., Orf, L., Rahmani, L., Antoniu, G., and Bougé, L. (2016). Adaptive Performance-Constrained In Situ Visualization of Atmospheric Simulations. In *CLUSTER '16 Proceedings of the IEEE International Conference on Cluster Computing*, pages 269–278. IEEE.

[Free Software Foundation 2018] Free Software Foundation (2018). The GNU C Library Manual. Technical report.

[Guzman et al. 2016] Guzman, J. C., Chapman, J., Marquarding, M., and Whiting, M. (2016). Status report of the end-to-end ASKAP software system: towards early science operations. In Chiozzi, G. and Guzman, J. C., editors, *Software and Cyberinfrastructure for Astronomy III*, volume 9913 of *SPIE Proceedings*. International Society for Optics and Photonics.

[Herbein et al. 2016] Herbein, S., Ahn, D. H., Lipari, D., Scogland, T. R., Stearman, M., Grondona, M., Garlick, J., Springmeyer, B., and Taufer, M. (2016). Scalable I/O-Aware Job Scheduling for Burst Buffer Enabled HPC Clusters. In *HPDC '16*

Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing, pages 69–80. ACM Press.

[IBM 2018] IBM (2018). Overview of IBM Spectrum Scale. `https://www.ibm.com/support/knowledgecenter/en/STXKQY{\_}4.2.3/com.ibm.spectrum.scale.v4r23.doc/bl1ins{\_}intro.htm`.

[IEEE and The Open Group 2013] IEEE and The Open Group (2013). IEEE Std 1003.1-2013 - Standard for Information Technology - Portable Operating System Interface (POSIX). Technical report, IEEE.

[Inacio et al. 2017a] Inacio, E. C., Barbetta, P. A., and Dantas, M. A. R. (2017a). A Statistical Analysis of the Performance Variability of Read/Write Operations on Parallel File Systems. *Procedia Computer Science - Special Issue: International Conference on Computational Science, ICCS 2017*, 108:2393–2397.

[Inacio and Dantas 2014] Inacio, E. C. and Dantas, M. A. R. (2014). A Survey into Performance and Energy Efficiency in HPC, Cloud and Big Data Environments. *International Journal of Networking and Virtual Organisations*, 14(4):299–318.

[Inacio and Dantas 2018a] Inacio, E. C. and Dantas, M. A. R. (2018a). An I/O Performance Evaluation Tool for Distributed Data-Intensive Scientific Applications. In *LADaS '18 - Proceedings of the Latin America Data Science Workshop*, pages 9–16. CEUR-WS.

[Inacio and Dantas 2018b] Inacio, E. C. and Dantas, M. A. R. (2018b). IORE : A Flexible and Distributed I/O Performance Evaluation Tool for Hyperscale Storage Systems. In *ISCC '18 Proceedings of the IEEE Symposium on Computers and Communication*, page (to appear). IEEE.

[Inacio et al. 2015a] Inacio, E. C., Dantas, M. A. R., and de Macedo, D. D. J. (2015a). Towards a performance characterization of a parallel file system over virtualized environments. In *ISCC '15 Proceedings of the 20th IEEE Symposium on Computers and Communications*, pages 595–600. IEEE.

[Inacio et al. 2017b] Inacio, E. C., Nonaka, J., Ono, K., and Dantas, M. A. R. (2017b). Analyzing the I/O Performance of Post-Hoc Visualization of Huge Simulation Datasets on the K Computer. In *WSCAD '17 - Anais do XVIII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 148–159. SBC.

[Inacio et al. 2015b] Inacio, E. C., Pilla, L. L. L., and Dantas, M. A. R. (2015b). Understanding the Effect of Multiple Factors on a Parallel File System's Performance. In *WETICE '15 Proceedings of the 24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 90–92. IEEE.

[Li et al. 2003] Li, J., Liao, W.-k., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. (2003). Parallel netCDF: A High-Performance Scientific I/O Interface. In *SC '03 Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, New York, New York, USA. ACM Press.

[Liu et al. 2012] Liu, N., Cope, J., Carns, P., Carothers, C., Ross, R., Grider, G., Crume, A., and Maltzahn, C. (2012). On the role of burst buffers in leadership-class storage systems. In *MSST '12 Proceedings of the IEEE 28th Symposium on Mass Storage Systems and Technologies*, pages 1–11. IEEE.

[LLNL 2013] LLNL (2013). IOR: Parallel filesystem I/O benchmark. `https://github.com/llnl/ior`.

[Lofstead et al. 2010] Lofstead, J., Zheng, F., Liu, Q., Klasky, S., Oldfield, R., Kordenbrock, T., Schwan, K., and Wolf, M. (2010). Managing Variability in the IO Performance of Petascale Storage Systems. In *SC '10 Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE.

[Lüttgau et al. 2018] Lüttgau, J., Kuhn, M., Duwe, K., Alforov, Y., Betke, E., Kunkel, J., and Ludwig, T. (2018). Survey of Storage Systems for High-Performance Computing. *Supercomputing Frontiers and Innovations*, 5(1):31–58.

[Mitchell et al. 2011] Mitchell, C., Ahrens, J., and Wang, J. (2011). VisIO: Enabling Interactive Visualization of Ultra-Scale, Time Series Data via High-Bandwidth Distributed I/O Systems. In *IPDPS '11 Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, pages 68–79. IEEE.

[Miyamoto et al. 2013] Miyamoto, Y., Kajikawa, Y., Yoshida, R., Yamaura, T., Yashiro, H., and Tomita, H. (2013). Deep moist atmospheric convection in a subkilometer global simulation. *Geophysical Research Letters*, 40(18):4922–4926.

[Moore et al. 2011] Moore, M., Bonnie, D., Ligon, W., Mills, N., Yang, S., Ligon, B., Marshall, M., Quarles, E., Sampson, S., and Wilson, B. (2011). OrangeFS: Advancing PVFS. In *FAST '11 Proceedings of the 9th USENIX conference on File and Storage Technologies*, pages 1–2, San Jose, CA, USA. USENIX Association.

[MPI Forum 1997] MPI Forum (1997). MPI-2: Extensions to the Message-Passing Interface. Technical report, Message Passing Interface Forum.

[MPIForum 2018] MPIForum (2018). Message Passing Interface Forum. `http://www.mpi-forum.org/`.

[Nagle et al. 2004] Nagle, D., Serenyi, D., and Matthews, A. (2004). The Panasas ActiveScale Storage Cluster - Delivering Scalable High Bandwidth Storage. In *SC '04 Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE.

[Nonaka et al. 2018] Nonaka, J., Inacio, E. C., Ono, K., Dantas, M. A. R., Kawashima, Y., Kawanabe, T., and Shoji, F. (2018). Data I/O management approach for the post-hoc visualization of big simulation data results. *International Journal of Modeling, Simulation, and Scientific Computing*.

[Nonaka et al. 2014] Nonaka, J., Ono, K., and Fujita, M. (2014). Multi-step image compositing for massively parallel rendering. In *HPCS '14 Proceedings of the International Conference on High Performance Computing & Simulation*, pages 627–634. IEEE.

[Omnibond 2018] Omnibond (2018). The OrangeFS Project. `http://www.orangefs.org`.

[OpenSFS 2018] OpenSFS (2018). Lustre. `http://lustre.org/`.

[Prabhat and Koziol 2014] Prabhat and Koziol, Q. (2014). *High Performance Parallel I/O*. Chapman & Hall/CRC, 1 edition.

[Radha et al. 2015] Radha, K., Rao, B. T., Babu, S. M., Rao, K. T., Reddy, V. K., and Saikiran, P. (2015). Service Level Agreements in Cloud Computing and Big Data. *International Journal of Electrical and Computer Engineering*, 5(1):158–165.

[Reed and Dongarra 2015] Reed, D. A. and Dongarra, J. (2015). Exascale computing and big data. *Communications of the ACM*, 58(7):56–68.

[Roten et al. 2016] Roten, D., Cui, Y., Olsen, K. B., Day, S. M., Withers, K., Savran, W. H., Wang, P., and Mu, D. (2016). High-Frequency Nonlinear Earthquake Simulations on Petascale Heterogeneous Supercomputers. In *SC '16 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE.

[Shan et al. 2008] Shan, H., Antypas, K., and Shalf, J. (2008). Characterizing and predicting the I/O performance of HPC applications using a parameterized synthetic benchmark. In *SC '08 Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE.

[Song et al. 2012] Song, H., Jin, H., He, J., Sun, X.-H., and Thakur, R. (2012). A Server-Level Adaptive Data Layout Strategy for Parallel File Systems. In *IPDPSW '12 Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pages 2095–2103. IEEE.

[Thakur et al. 1999] Thakur, R., Gropp, W., and Lusk, E. (1999). On implementing MPI-IO portably and with high performance. In *IOPADS '99 Proceedings of the sixth workshop on I/O in parallel and distributed systems*, pages 23–32, New York, New York, USA. ACM Press.

[Thakur et al. 2002] Thakur, R., Gropp, W., and Lusk, E. (2002). Optimizing noncontiguous accesses in MPI–IO. *Parallel Computing*, 28(1):83–105.

[The HDF Group 1997] The HDF Group (1997). Hierarchical Data Format, version 5. `https://support.hdfgroup.org/HDF5/`.

[Troppens et al. 2009] Troppens, U., Erkens, R., Mueller-Friedt, W., Wolafka, R., and Haustein, N. (2009). *Storage Networks Explained: Basics and Application of Fibre Channel SAN, NAS, iSCSI, InfiniBand and FCoE*. Wiley Publishing, 2 edition.

[Weil et al. 2006] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., and Maltzahn, C. (2006). Ceph: a scalable, high-performance distributed file system. In *OSDI '06 Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, Berkeley, CA, USA. USENIX Association.

[Zhao et al. 2016] Zhao, D., Liu, N., Kimpe, D., Ross, R., Sun, X.-H., and Raicu, I. (2016). Towards Exploring Data-Intensive Scientific Applications at Extreme Scales through Systems and Simulations. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1824–1837.

Capítulo 2

Boas Práticas para a Implementação e Gerência de um Centro de Supercomputação Desassistido

Albino A. Aveleda, Ricardo P. Pareto, Alvaro L.G.A. Coutinho

Núcleo Avançado de Computação de Alto Desempenho (NACAD), COPPE/UFRJ
{albino, padilha, alvaro}@nacad.ufrj.br

Abstract

The High Performance Computing Center (NACAD) of COPPE / UFRJ is a laboratory specialized in the application of high performance computing to problems of engineering and science in general. NACAD also has extensive experience in the administration, management and tools development to support the Supercomputing Center and to develop and implement innovations in the machine management environment. The present mini-course proposes to share some of these best practices adopted by NACAD-COPPE/UFRJ made in the implementation of the Lobo Carneiro supercomputer.

Resumo

O Núcleo Avançado de Computação de Alto Desempenho (NACAD) da COPPE/UFRJ é um laboratório especializado na aplicação de computação de alto desempenho a problemas de engenharia e ciências em geral. O NACAD também possui grande experiência na administração, gerência e ferramentas de apoio ao Centro de Supercomputação e de desenvolver e implementar inovações no ambiente de administração e gerência da máquina. O presente minicurso propõe compartilhar algumas dessas melhores práticas adotadas pelo NACAD-COPPE/UFRJ feitas na implantação do supercomputador Lobo Carneiro.

Palavras-chave*: segurança, consumo de energia, portal de usuários, Internet das Coisas, indústria 4.0, aprendizado de máquina*

2.1. Introdução

Este trabalho tem a proposta de compartilhar algumas das melhores práticas para o gerenciamento e desenvolvimento de ferramentas de apoio ao Centro de Supercomputação desassistido que hospeda o supercomputador Lobo Carneiro. O foco do trabalho é mostrar o desenvolvimento e implementação das inovações no ambiente de administração e gerência do supercomputador.

Em um ambiente de computação de alto desempenho, normalmente a maior quantidade de ativos são os computadores. Eles foram e são uma das mais importantes ferramentas utilizadas que impulsionam o atual desenvolvimento tecnológico. Então, por que não usar esse potencial e desenvolver um ambiente desassistido que possa atuar de forma mais inteligente e automatizada? Dessa maneira torna-se possível que o sistema atue em caso de alguma falha no processo e se adapte para mitigar seus efeitos. A fim de viabilizar esse tipo de arquitetura foi desenvolvido um ambiente que integra várias tecnologias emergentes, tais como: internet das coisas, computação em nuvem, aplicativos (celulares e *tablets*) etc. A integração de várias dessas tecnologias também é conhecida como indústria 4.0.

O termo indústria 4.0 se originou a partir de um projeto de estratégias do governo alemão voltadas à tecnologia. "*Estamos a bordo de uma revolução tecnológica que transformará fundamentalmente a forma como vivemos, trabalhamos e nos relacionamos. Em sua escala, alcance e complexidade, a transformação será diferente de qualquer coisa que o ser humano tenha experimentado antes*", diz Klaus Schwab [Schwab]. Seu fundamento básico implica que conectando máquinas, sistemas e ativos, torna-se possível criar redes inteligentes ao longo de toda a cadeia de valor que podem controlar os módulos da produção de forma autônoma. Ou seja, as fábricas inteligentes terão a capacidade e autonomia para agendar manutenções, prever falhas nos processos e se adaptar aos requisitos e mudanças não planejadas na produção.

No nosso caso especifico, o uso dessa tecnologia é para preservar o investimento fazendo com que o supercomputador opere com segurança e baixo consumo de energia, dentro da faixa definida pelo fornecedor. Caso ocorra algum problema durante a operação desassistida, o sistema de controle irá interferir de forma autônoma para preservar a máquina. Podem-se citar alguns exemplos de problemas e ações de controle, ou seja:

- Falta de fornecimento de energia elétrica: apesar do nobreak manter o supercomputador funcionando, é necessário monitorar a temperatura e a carga das baterias do nobreak, a fim de permitir um desligamento correto das máquinas, principalmente no que se refere a manutenção da integridade dos arquivos contidos na área de armazenamento;
- Problemas na refrigeração: é necessário o monitoramento da temperatura do ambiente, independente da falta de energia, para evitar que caso ocorra algum problema na refrigeração o supercomputador não ultrapasse a temperatura máxima permitida de operação. A perda da garantia do fabricante do computador e de sistemas auxiliares pode ser uma consequência danosa desse tipo de falha;
- Problemas com o nobreak: é necessário o monitoramento das baterias e da carga do nobreak para verificar se o funcionamento está dentro dos limites de operação, a fim de evitar o desligamento precoce na falta de fornecimento de energia elétrica.

Sendo assim, de forma a conduzir esta exposição da forma mais clara possível, este texto está organizado da seguinte forma:

- A seção 2.2 apresenta as premissas do desenho da solução do Centro de Supercomputação;
- A seção 2.3 aborda um item cada vez mais importante nos centros de supercomputação, que é o alto consumo de energia elétrica;
- A seção 2.4 mostra o uma visão geral do desenho, desenvolvimento e implementação do Portal de Usuários e da Wiki de Suporte.
- A seção 2.5 discute algumas das soluções de segurança, tanto física como lógica, aplicadas ao supercomputador;
- A seção 2.6 introduz o uso de Internet das Coisas aplicadas ao ambiente do supercomputador;
- A seção 2.7 faz as considerações finais.

2.2. Desenho da Solução do Centro de Supercomputação

O Núcleo Avançado de Computação de Alto Desempenho (NACAD) da COPPE/UFRJ possui grande experiência na administração, gerência e desenvolvimento de ferramentas de apoio ao Centro de Supercomputação. Aproveitando-se da experiência adquirida durante a implementação do cluster Galileu, que fez parte da lista do TOP500 [TOP500] entre 11/2009 a 06/2012, e em 2010 foi a maior supercomputador da América Latina, foi feito o desenho da solução no supercomputador atual do NACAD, o Lobo Carneiro.

O supercomputador Lobo Carneiro possui a seguinte especificação: total de nós de processamento: 252; 504 CPUs Intel Xeon E5-2670v3 (Haswell): 6048 Cores; cores/nó processamento: 24; threads/nó processamento: 48; memória por nó de processamento: 64 GBytes; total de memória RAM: 16 TBytes (distribuída); sistema de arquivo paralelo: Intel Lustre (500 TBytes); armazenamento em disco: 60 TBytes; rede: Infiniband FDR - 56 Gbs topologia hipercubo. O desempenho de pico do Lobo Carneiro no HPL [Dongarra] é de 191TFlop/s.

Para aumentar a eficiência da refrigeração o supercomputador possui um isolamento entre a entrada de ar (corredor frio) e a saída de ar (corredor quente).

O supercomputador Lobo Carneiro está instalado fisicamente a uma distância de aproximadamente 4 (quatro) Km do NACAD, no parque tecnológico da UFRJ, em ambiente especialmente projetado para esse fim e próximo das empresas sediadas neste local. Em função da distância e de uma equipe reduzida, foi necessário desenvolver um ambiente autônomo, pois por exemplo, caso houvesse algum problema de falta no fornecimento de energia, não teríamos acesso a rede para poder interagir com o sistema a fim de monitorá-lo ou desligá-lo.

Desta forma foram definidas as principais premissas de operação do supercomputador Lobo Carneiro:

- Desenvolver uma melhor gerência sobre o supercomputador, levando em consideração restrições financeiras, de forma a automatizar o máximo possível a operação e manutenção do sistema.
- Autonomia de operação 24x7 sem a necessidade de intervenção humana para o caso de algum tipo de falha do sistema. Em uma universidade pública é muito difícil, e as vezes complicado, manter uma estrutura 24x7. Seria necessária uma equipe bem maior para poder montar uma escala de trabalho 24x7, aumentando significativamente o custo de pessoal.
- Monitoramento e gerenciamento das temperaturas do corredor frio, corredor quente, sala do nobreak e sala de operação. Isto permite um melhor controle da temperatura de todos os ambientes e possibilitando que o supercomputador opere dentro das especificações de fábrica, mantendo assim a garantia de todos os equipamentos.
- Maior controle sobre o consumo de energia, que atualmente é um dos fatores críticos no custo total de centros de supercomputação.
- Controle sobre todos os equipamentos ligados/desligados no site. O controle deve permitir acesso remoto para ligar e desligar quaisquer equipamentos do supercomputador, incluindo até os sistemas de armazenamento que não possuem botão de liga e desliga.
- Controle do sistema de UPS que inclui: carga do UPS, carga das baterias, tensões das fases de entrada etc.
- Desenvolvimento de um Portal de Usuários para permitir um ambiente de interação com os usuários do sistema. O portal deve prover:
    - Gerenciamento dos usuários
    - Abertura de chamados de suporte
    - Integração com o supercomputador
        - Automação na abertura de contas
        - Segurança nas comunicações entre o Portal e o supercomputador
- Segurança de acesso tanto físico como lógico.

O perfil de operação do supercomputador deve permitir jobs longos de até 1.000 horas de *walltime* e jobs com múltiplos nós de processamento. Limitamos ao máximo de 40 nós o que equivale a 960 cores reais ou 1.920 threads para que a máquina possa ser usada por mais de um usuário simultaneamente. A Figura 2.1 mostra uma listagem parcial dos jobs, executando no dia 21/08/2018, onde as informações dos usuários foram removidas. Pode-se ver jobs sendo executados a mais de 912 horas, o que equivale 38 dias de execução.

O sistema também deve permitir o controle dos recursos utilizados para evitar que um ou mais grupos/projetos venham a monopolizar os recursos do supercomputador através de dezenas de jobs sendo alocados para processar ao mesmo tempo. Sendo assim, é necessário definir alguns limites para usuários/grupo na máquina, tais como: o número de job simultâneos, o número máximo de nós alocados, etc.

```
user@service1:~> qstat -a
service1:
                                                            Req'd  Req'd   Elap
Job ID          Username Queue  Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- ------ ---------- ------ --- --- ------ ----- - -----
97589.service1  xxxxxx   workq  job31_bh2   43909   2  96    --  999:0 R 918:4
97949.service1  yyyyyy   workq  job_02_1    15840   2  96    --  672:0 R 278:3
97950.service1  yyyyyy   workq  job_02_2    18190   2  96    --  672:0 R 263:5
102580.service1 xxxxxx   workq  job34_bh2   12856   2  96    --  999:0 R 765:5
102581.service1 xxxxxx   workq  job35_bh2   16957   2  96    --  999:0 R 796:3
102989.service1 xxxxxx   workq  job02b_bh1  48539   2  96    --  999:0 R 789:0
102990.service1 xxxxxx   workq  job02_bh12  42434   2  96    --  999:0 R 658:2
103217.service1 zzzzzz   workq  job_run     43872   1  48    --  1000: R 737:3
104076.service1 zzzzzz   workq  job_run      1040   2  96    --  1000: R 427:0
105986.service1 wwwwww   workq  job_F       37483   1  48    --  999:0 R 286:4
. . .
```

Figura 2.1. Listagem parcial dos Jobs em execução no dia 21/08/2018.

2.3. Consumo de Energia

O consumo de energia ainda é um dos grandes obstáculos para atingir a computação exascala e diversas pesquisas estão sendo realizadas com a finalidade de diminuir o consumo por FLOPS, isto é, operações de ponto flutuante por segundo (FLoating-point Operations Per Second).

Atualmente uma das maiores preocupações de um centro de supercomputação é o consumo de energia. Isso se deve a grande capacidade de condensação dos equipamentos por rack, atingindo algumas vezes o consumo maior do que 40KW/rack. O maior consumo de energia pelos computadores gera um maior calor. Com isso, torna-se necessária uma infraestrutura de refrigeração para controlar a temperatura e mantê-la na faixa de operação. Isto contribui para aumentar ainda mais o consumo elétrico e, por conseguinte, a um maior custo financeiro mensal.

Dependendo das condições das tarifas em um determinado país, pode-se gastar em energia o equivalente ao custo do supercomputador em poucos anos [Martin]. Para efetuar este controle de forma integrada são desenvolvidas ações no sistema de filas e na operação do supercomputador. Algumas destas medidas são discutidas em seguida.

2.3.1. Integração com o PBS-Pro

O hardware do supercomputador Lobo Carneiro (SGI ICE-X) permite um maior controle sobre o consumo energético. Este recurso viabilizou a integração com o sistema de fila (PBS-Pro) para permitir uma coleta de informações de consumo de energia durante a execução de um job. Esta coleta é feita por *resources in hooks* escritos em Python que se

integram ao PBS-Pro durante a execução dos jobs. Desta forma é possível definir perfis de consumo energético por job e/ou fila. Além de poder coletar o consumo energético por job, isto é, pelos nós de processamento usados. A Figura 2.2 ilustra a arquitetura de medição do consumo energético. Entretanto, nem todos os equipamentos da SGI possuem esses medidores. A família SGI ICE-X possui racks de computação personalizados. Dentro de cada rack há dois ou quatro subsistemas de energia de 12VDC. Uma interface proprietária da SGI é usada para ler a energia de cada rack que permite coletar as informações dos nós computacionais (lâminas ou *blades* em Inglês).

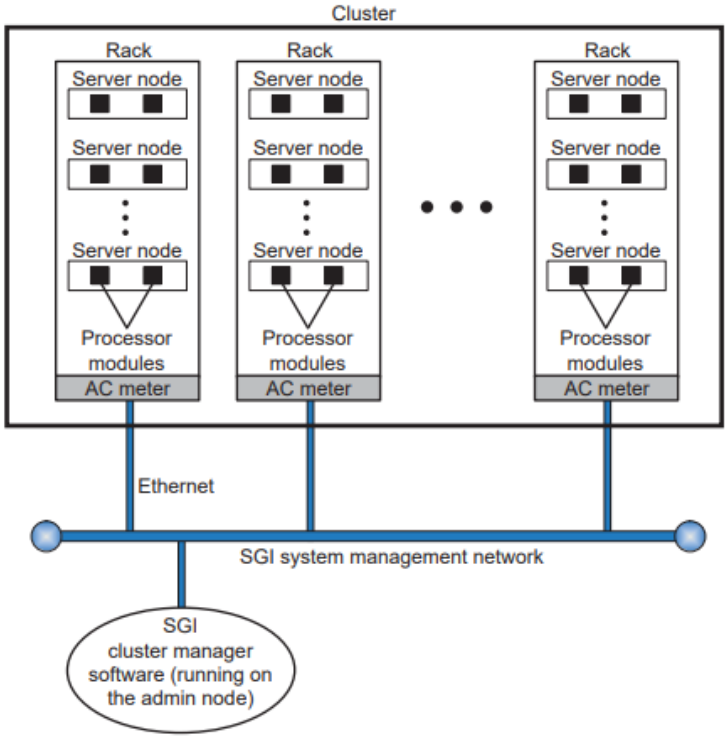


Figura 2.2. Arquitetura do Sistema de Medição de Energia da SGI [PMG].

Apesar do sistema do NACAD-COPPE/UFRJ monitorar todo o consumo energético, não há como calcular o consumo individual das áreas compartilhadas. Entretanto, como o maior consumo se dá nos nós de processamento, essa informação retorna um valor próximo do consumo real do usuário. O usuário pode facilmente obter as informações do PBS-Pro sobre o consumo dos nós de processamento usados durante a execução do seu job. Basta para isso incluir no script do job as informações referentes ao seu e-mail, como mostrado na Figura 2.3.

Esse recurso permitiu com que fosse feito uma comparação [Canesin] entre consumo energético entre nós com CPUs Intel Xeon (SGI ICE-X) e nós com CPUs ARM através do protótipo da máquina do projeto MontBlanc [Rajovic et al] instalado no Barcelona Supercomputing Center. O projeto MontBlanc possui equipamentos de medição externos para monitorar o consumo energético.

```
#!/bin/bash
#PBS -l select=2:ncpus=48:mpiprocs=24
#PBS -l walltime=400:00:00
#PBS -j oe
#PBS -V
#PBS -N mpi-intel
#PBS -m ea
#PBS -M user@domain.br

# load modules
module load intel
# change directory
cd ${PBS_O_WORKDIR}
# run
mpirun ./mpitest
```

Figura 2.3. Exemplo de job com solicitação de e-mail ao término do job.

```
From: root <loboc@nacad.ufrj.br>
Date: 2018-06-05 8:51 GMT-03:00
Subject: PBS JOB 93269.service1
To: user@domain.br

PBS Job Id: 93269.service1
Job Name:   xxxxxxx
Execution terminated
Exit_status=0
resources_used.cpupercent=2700
resources_used.cput=8871:19:46
resources_used.energy=92.9448
resources_used.mem=3403776kb
resources_used.ncpus=48
resources_used.vmem=5038460kb
resources_used.walltime=380:19:14
```

Figura 2.4. Exemplo de e-mail com informação de consumo de energia por job.

A Figura 2.4 ilustra um e-mail com a saída do consumo de energia do job, observado no item "*resources_used.energy*" em KWh.

```
service:~ # qmgr -c "set server power_provisioning=True"
service:~ # qmgr -c "set node @default power_enable=True"

service:~ # pbsnodes -a | grep resources_available.eoe | uniq
   resources_available.eoe = 100W,150W,200W,250W,300W,350W,400W,
   450W,500W,NONE
```

Figura 2.5. Configuração do PBS-Pro.

Depois do script *hook* pronto, a configuração do PBS-Pro se torna muito simples como pode ser visto na Figura 2.5. Basicamente é habilitado o *power_provisioning* no PBS-Pro e depois habilitado o atributo *power_enable* em todos os nós computacionais.

2.3.2. Consumo de energia do supercomputador

No Brasil, durante o "horário de ponta", que corresponde ao intervalo das 17:30hs às 20:30hs nos dias úteis, o preço da energia oscila, dependendo da distribuidora, em torno de 5 a 6 vezes o preço em uma hora normal. Esse é um dos motivos da implantação do horário de verão no Brasil, para tentar diminuir o consumo durante esse período.

Como o supercomputador consome muitos KWh de energia, se for possível diminuir o consumo do mesmo durante o horário de ponta, isso acarretaria numa grande redução do custo de energia. Entretanto, como mostrado na Figura 2.1 alguns dos nossos usuários tem jobs que duram mais de 1 (um) mês sendo executados, o que dificulta esse tipo de controle. Não obstante, o Lobo Carneiro possui algumas ferramentas que podem contribuir para diminuir o consumo de energia.

O controle de energia pode ser feito em diversos níveis: sistema, rack ou nó. A Figura 2.6 mostra como é feito essa coleta. O primeiro exemplo, Figura 2.6(a), corresponde a todo o sistema, o segundo, Figura 2.6(b), de apenas um (1) rack e o último, Figura 2.6(c), o de um nó em modo de execução. O consumo das lâminas (blades) da SGI ICE-X oscilam entre 50-400W e os demais nós oscilam entre 150-600W. Dessa maneira, pode-se limitar o consumo por sistema, rack ou nó de acordo com a necessidade.

Para exemplificar, suponha que se deseje limitar o consumo do nó r1i3n9 para o máximo de 200W, bastaria utilizar o comando mostrado na Figura 2.7. Logo, para o nó em questão o consumo diminuiria em torno de 23%.

Se durante a hora de ponta fosse adotado esse procedimento em todo o supercomputador, o impacto na conta de energia seria grande, pois o custo durante esse período é muito alto. O NACAD ainda está fazendo a análise de desempenho dos jobs para identificar o impacto do consumo em sua execução.

```
# mpower system get_power
System Power Stats:
Instant                           : 57384.71
Minimum during sampling period    : 202.61
Maximum during sampling period    : 16217.39
Average during sampling period    : 12461.97
KwH during sampling period        : 340691.45
```

(a)

```
# mpower rack get_power rack1
r1lead:
Instant                           : 38145.33
Minimum during sampling period    : 216.78
Maximum during sampling period    : 16217.39
Average during sampling period    : 6694.35
KwH during sampling period        : 215816.91
Sampling period                   : 29014806.75
```

(b)

```
# mpower node get_power r1i3n9
r1i3n9        262W
```

(c)

Figura 2.6. Coleta de informações de consumo.

```
# mpower node set_limit r1i3n9 200
# mpower node get_power r1i3n9
r1i3n9        200W
```

Figura 2.7. Definindo um limite de consumo.

2.4. Desenvolvimento do Portal de Usuários e de Suporte

2.4.1. Portal de Usuários

O desenvolvimento de um portal para o Lobo Carneiro surgiu em função da necessidade de facilitar a troca de informações com os usuários. Esses tipos de Portais também estão presentes em várias instalações de centros de supercomputação ao redor do mundo, tais como no TACC [TACC] e no XSEDE [XSEDE]. Este portal deve permitir a integração com o supercomputador e a inclusão de scripts para facilitar a implantação de novas funcionalidades. Estas premissas foram importantes para a escolha do framework. O framework escolhido foi o Django [Django], que é desenvolvido em Python. As principais caraterísticas do portal devem ser:

- Atualização on-line de várias informações coletadas do supercomputador, por exemplo, o status do supercomputador, número de jobs em execução, número de jobs na fila etc.
- Informações das contas para os usuários, como o uso de espaço de armazenamento e o SU (system units) que é calculado em função do tempo de processamento por core alocado, isto é, *número de cores alocado* x *tempo de processamento* (*walltime*).
- Sumário dos dados dos usuários para os coordenadores.
- Notícias de modo geral, como instalação de softwares e bibliotecas, informes sobre o sistema etc.
- Permitir a abertura de chamados de suporte e seu respectivo acompanhamento.
- Permitir que os menus sejam sensíveis a conta, isto é, o menu disponibilizado para um usuário comum é diferente do aparece para o coordenador, que por sua vez também é diferente do que é disponibilizado para os administradores.
- Documentação geral do sistema.
- Informações de administração etc.

A seguir será descrita uma visão geral dos principais menus do Portal, tomando como referência o menu de administração. Alguns dessas opções podem não estar disponíveis para certos tipos de usuários.

- Perfil: mostra o perfil do usuário, suas estatísticas de uso e permite a abertura de chamado de suporte. Com base nos chamados feitos pelos usuários foi possível criar um FAQ, perguntas mais frequentes, do supercomputador Lobo Carneiro. Permitindo assim, uma melhor integração do usuário com o Portal.
- Coordenação: mostra a solicitação de registro, pois nesse portal o coordenador tem independência de controlar a sua equipe. Também mostra a equipe do projeto e sua estatística de uso.
- Recursos: mostra informações sobre os equipamentos disponíveis no centro de supercomputação. Mostra uma visão geral dos equipamentos, status dos servidores e nós computacionais, informações sobre a fila e lista de IPs que foram bloqueados durante a autenticação. Este bloqueio será explicado melhor em um item posterior.
- Administração: mostra todos os chamados de suporte incluindo o status deles, todos os projetos e usuários, permite postagem de notícias e acesso aos logs.
- Documentação: informações de acesso ao sistema, Guia do Usuário, documentação dos compilares Intel e PGI.
- Sobre: informações sobre registro e renovação de contas.

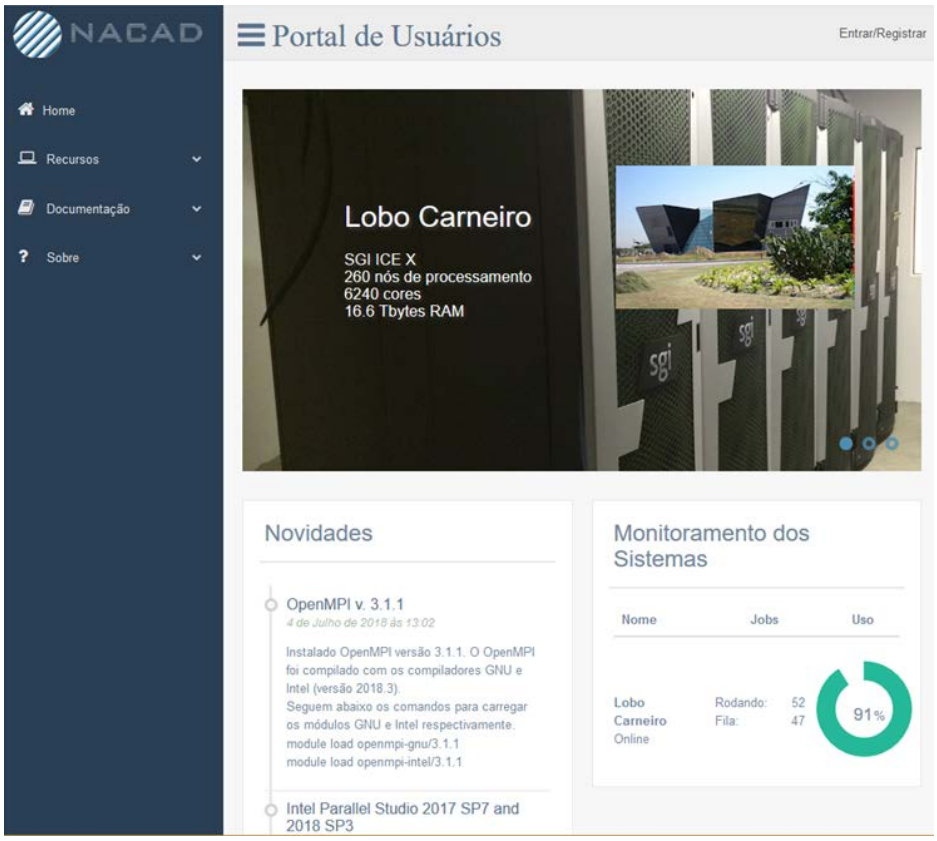


Figura 2.8: Portal de Usuários.

A Figura 2.8 mostra a página de entrada do Portal de Usuários. No canto superior direito é possível entrar na conta e com isso ter acesso a outras informações.

O Portal de Usuários está instalado em um servidor localizado dentro do laboratório do NACAD e não junto com o supercomputador. Isso traz algumas vantagens, por exemplo, se houver algum problema no site do supercomputador os usuários poderão obter essa informação consultando o portal.

2.4.2. Wiki de Suporte

Além do Portal de Usuários foi configurado em outra máquina um outro Portal contendo uma Wiki. A Wiki é usada apenas pela equipe de suporte e possui informações internas e restritas ao NACAD. Portanto, tais informações não são compartilhadas. A finalidade dela é de prover um aprendizado contínuo para a equipe de suporte. Toda a configuração do sistema, ajustes e correção de problemas são documentados nesta Wiki. Dessa forma, quando o problema se repetir os demais membros da equipe consultar o conteúdo da Wiki e podem corrigir o problema seguindo as orientações lá contidas. Seguem abaixo alguns dos principais tópicos que fazem parte desta Wiki:

- Instruções de boot e shutdown do supercomputador;
- Dicas e correções de problemas do sistema de fila PBS-Pro;
- Configurações específicas para os diversos servidores e nós computacionais, incluindo informações especificas de cada servidor e suas respectivas funções;
- Configuração, controle de cota e correções de problemas para os servidores de armazenamento, incluindo o *filesystem* paralelo Lustre;
- Instruções de desenvolvimento para compilação de forma otimizada para diversas ferramentas;
- Instruções sobre atualização e ajustes nos compiladores Intel, PGI e GNU;
- Instruções para o desenvolvimento de *hooks* para o PBS-Pro;
- Instruções para configuração do Portal de Usuários;
- Instruções de configuração de segurança para ambientes Linux;
- Informações sobre uso dos scripts desenvolvidos pela equipe;
- Outros tópicos.

O conteúdo da Wiki é continuamente atualizado. Qualquer intervenção no sistema motivada por atualizações, adaptações solicitadas pelos usuários, etc, e imediatamente documentada e incorporada a Wiki.

2.5. Segurança

No mundo atual, a questão de segurança está se tornando cada vez mais importante. No caso de um centro de supercomputação a importância aumenta. Entretanto, como o site do Lobo Carneiro é desassistido, o item de segurança se torna ainda mais fundamental. A segurança do Centro de Supercomputação deve levar em consideração tanto a parte física como a parte lógica.

Para endereçar a parte física foram instaladas câmeras de vídeos, que podem ser acompanhadas tanto pelos seguranças do Parque Tecnológico, como por computador e/ou celular, como mostra a Figura 2.9. Além das câmeras foram instalados vários sensores de

IoT (Internet das Coisas). Como por exemplo, sensores de temperatura que foram distribuídos na sala de operação, sala do nobreak, corredor frio e corredor quente.

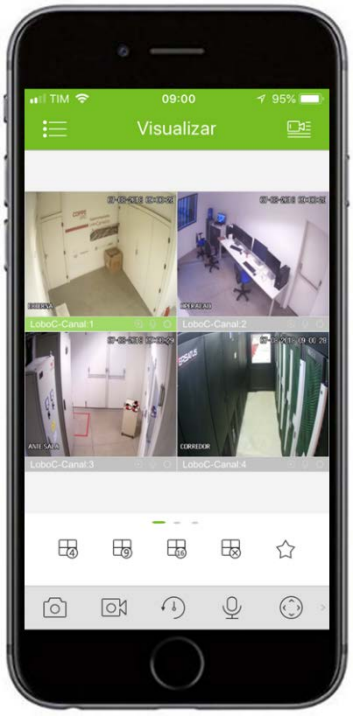


Figura 2.9: Monitoração das câmeras de vídeo através do celular.

2.5.1. Integração do supercomputador com o Portal

Para minimizar a superfície de ataque contra o supercomputador, toda a comunicação entre o Portal de Usuários e o supercomputador é feito por mão única, isto é, toda a comunicação é originada apenas do supercomputador. Ou seja, em nenhum momento o Portal envia dados para o supercomputador. Neste caso, se o Portal vier a ser comprometido, o invasor não terá acesso ao supercomputador. A Figura 2.10 ilustra de forma simplificada o sentido das comunicações entre os sistemas. O usuário pode acessar tanto o Portal como o supercomputador através do servidor de acesso. Este por sua vez, atualiza o Portal com os dados coletados do supercomputador. Vários dados coletados também são colocados na nuvem para poderem ser consumidos por outros dispositivos, tais como celular e *tablets*. Alguns desses itens serão descritos com mais detalhes posteriormente.

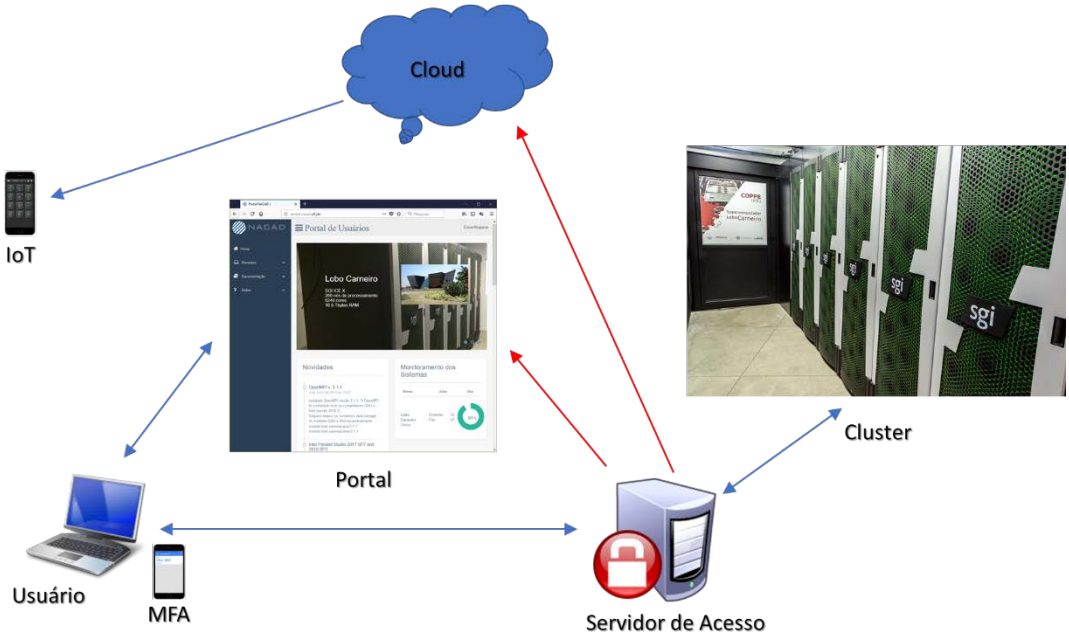
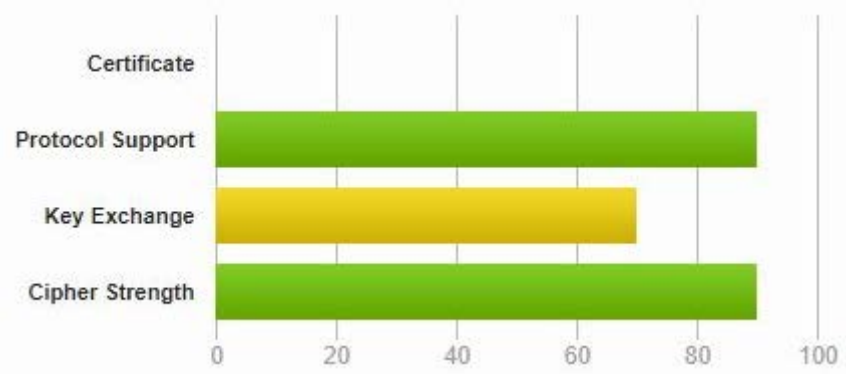


Figura 2.10. Infraestrutura lógica do supercomputador.

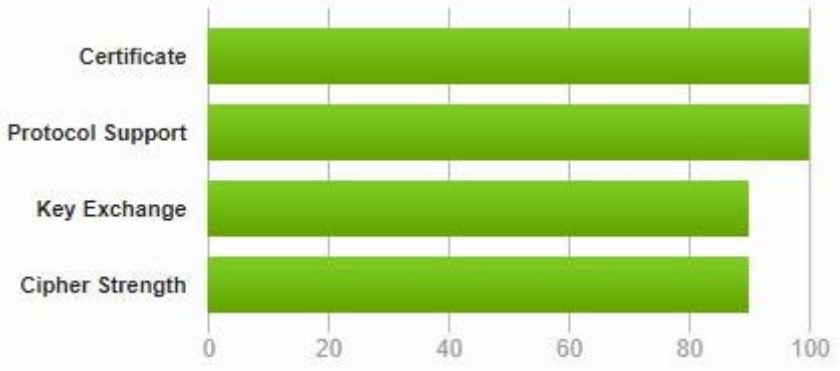
A fim de aumentar a segurança no Portal de Usuários foram implementados alguns ajustes no sistema. As configurações usuais não serão descritas neste item, apenas as mais importantes serão listadas e comentadas.

- Configuração do SELinux (*Security-Enhanced Linux*) que limita o escopo de possíveis danos que podem resultar da exploração de vulnerabilidades em aplicativos e serviços do sistema. O SELinux fornece um sistema flexível baseado no MAC (*Mandatory Access Control*), integrado ao kernel do Linux.
- Testes de penetração e outros testes. Para exemplificar, será mostrada a análise do item SSL do Portal. Durante a configuração, o Portal foi feito inicialmente usando a instalação padrão do servidor Web com um certificado auto assinado. O resultado do teste é mostrado na Figura 2.11(a). Com base nos resultados foram feitos ajustes no servidor Web, principalmente referentes aos protocolos suportados e chaves utilizadas, obtendo assim um aumento significativo na segurança do site, como mostrado na Figura 2.11(b).
- Configuração de um sistema de detecção de intrusão. Este sistema detecta qualquer alteração, inclusão ou remoção de arquivo do sistema. Ele utiliza uma base de dados para comparar com o sistema atual. Uma cópia dessa base de dados deve estar em outra máquina, pois no caso de uma invasão o invasor pode gerar outra base de dados diferente. A fim de mitigar esse problema foram criados alguns scripts que enviam diariamente um e-mail com o resultado da análise do sistema de intrusão e no assunto do e-mail vai junto o hash MD5 da base de dados em questão. Logo, se um invasor gerar uma nova base de dados, pode-se identificar rapidamente pelo cabeçalho do e-mail. A Figura 2.12 mostra o cabeçalho dos e-mails diários de duas máquinas, a do portal e a máquina de acesso com os respectivos hashes no campo assunto do e-mail. Na Figura 2.13 mostra

um trecho de uma mensagem padrão enviada por e-mail diariamente. Pode-se observar que este relatório é dividido em basicamente cinco partes, que são: sumário do relatório; arquivos que foram adicionados, removidos e modificados em relação ao banco de dados; e as informações com os detalhes das mudanças.



(a) Configuração padrão do servidor Web



(b) Configuração ajustada após análise do teste

Figura 2.11: Resultados dos testes.

Figura 2.12: E-mails automáticos do sistema de detecção de intrusão.

```
Summary:
  Total number of files:        73904
  Added files:                  34
  Removed files:                31
  Changed files:                22

---------------------------------------------------
Added files:
---------------------------------------------------
added: /var/log/btmp-20180701
added: /var/log/cron-20180624.gz
added: /var/log/httpd/access_log-20180611.gz
added: /var/log/httpd/error_log-20180624.gz
...

---------------------------------------------------
Removed files:
---------------------------------------------------
removed: /var/log/btmp-20170301
removed: /var/log/cron-20170226.gz
removed: /var/log/httpd/access_log-20170220.gz
removed: /var/log/httpd/error_log-20170220.gz
...

---------------------------------------------------
Changed files:
---------------------------------------------------
changed: /etc/sysconfig/network-scripts
...
changed: /var/log/lastlog
changed: /var/log/maillog
changed: /var/log/messages
...

---------------------------------------------------
Detailed information about changes:
---------------------------------------------------
Directory: /etc/sysconfig/network-scripts
 Mtime    : 2017-02-13 08:01:31              , 2018-06-11 17:03:05
 Ctime    : 2017-02-13 08:01:35              , 2018-06-11 17:03:08
...
File: /var/log/lastlog
 Mtime    : 2017-03-04 08:39:26              , 2018-06-25 09:58:31
 Ctime    : 2017-03-04 08:39:26              , 2018-06-25 09:58:31
 SHA256   : eghs7kR66UF58Nqe1EiV+P8UKtjiEzZ+,
Lqgbms9f4eSz4wzuw/EkP2nCblQ3RkZm
 SHA512   : QnP7b4I9EyG5kZp2i++mjM8Fb3V5xJsA,
BFEUOYzQqSHmyCzUQMjpaR1jLlIcjsQO
...
```

Figura 2.13: Relatório diário de alterações do sistema.

2.5.2. Acesso e autenticação dos usuários

Diferentemente do Portal de Usuários, onde o usuário utiliza apenas uma interface web e ele não possui conta na máquina, o supercomputador Lobo Carneiro precisa fornecer uma conta para permitir o uso dos seus recursos. O usuário poderá fazer acesso ao supercomputador de casa, do laboratório, do trabalho, do cybercafé, etc. Isso pode gerar um grande problema, pois não há como garantir a segurança do lado do usuário. Em outras palavras, se o laboratório/acesso dele for comprometido, o invasor pode, por exemplo, trocar o cliente SSH, que é usado para acessar o supercomputador, e com isso coletar as contas e senhas das outras máquinas. De posse dessas senhas, o invasor poderia entrar no supercomputador com a conta de um usuário. Dessa forma a superfície de ataque do invasor aumentaria significativamente, pois ele já teria acesso ao supercomputador.

Para contornar esse problema adota-se um sistema conhecido como autenticação de multi-fator (MFA - *Multi-Factor Authentication*), no caso do Lobo Carneiro, de dois fatores. Logo, para ter acesso ao supercomputador o usuário deverá informar seu nome de usuário e senha (o primeiro fator – o que ele sabe) e também um código de autenticação de seu dispositivo de MFA (o segundo fator – o que ele tem). Juntos, esses vários fatores fornecem maior segurança para suas configurações e recursos de conta.

Atualmente o celular é considerado um produto essencial e praticamente todos possuem um smartphone. Logo, é totalmente factível a implementação de um sistema MFA usando o smartphone como o segundo fator de autenticação. Existem vários softwares que rodam tanto em IOS (Apple) como no Android, são eles: Google Authenticator, Microsoft Authenticator, FreeOTP e outros. O único requisito para o sistema funcionar corretamente é que o smartphone esteja com a hora correta. Basta então, que o usuário sincronize a hora do seu celular com a operadora de telefonia.

Do lado do supercomputador foi compilado, instalado e configurado o libpam do Google Authenticator e feitos os ajustes necessários no serviço NTP (*Network Time Protocol*) para que a hora esteja sempre sincronizada.

Durante a abertura de conta de um usuário o script executa várias etapas, tais como: criar a conta; criar grupo do projeto, caso necessário; configurar as cotas em disco (área home e área scratch). Ele configura também o Google Authenticator para o usuário gerando um QR-code que é enviado para o Portal de Usuários. O usuário por sua vez ao entrar no Portal e com um dos aplicativos do MFA instalado em seu celular, que deve ter acesso a câmera, deve apontar a câmera para o QR-code gerado pelo sistema a fim de ler e gerar a sua configuração automaticamente. A Figura 2.14 mostra a tela que aparece no Portal. Para aumentar ainda mais a segurança este QR-code só é mostrado apenas uma vez. Desta maneira, tenta-se evitar que outra pessoa possa usar o mesmo QR-code do usuário no caso de deixar o Portal aberto na sua conta ou tenha sua senha violada.

Figura 2.14: QR-code do MFA.

De posse do QR-code, o usuário já pode se autenticar no supercomputador. O processo é muito simples, basta o usuário se conectar no supercomputador usando a sua conta e senha e consultar o código no celular. Este código é mudado a cada minuto e só pode ser usado apenas uma vez, também por motivos de segurança. As Figuras 2.15 e 2.16 ilustram o processo de *login* do usuário.

Figura 2.15: Google Authenticator no smartphone.

```
login as: user
Using keyboard-interactive authentication.
Password: ********
Using keyboard-interactive authentication.
Verification code: 294380
------------------------------------------------------------------
          Welcome to the Lobo Carneiro Supercomputer
                      NACAD-COPPE/UFRJ
------------------------------------------------------------------

     ** WARNING: Unauthorized use/access is prohibited. **

If you log on to this computer system, you acknowledge your
awareness of and concurrence with the NACAD Acceptable Use
Policy. By attempting connection without permission, you are
in violation of federal law.

NACAD Usage Policies:
http://www.nacad.ufrj.br/informacoes/politica-de-uso
------------------------------------------------------------------
user@service:~>
```

Figura 2.16: Processo de login no supercomputador.

Além dos procedimentos de segurança descritos anteriormente foram implementadas outras medidas, tais como:

- Firewall GeoIP: São regras de firewall baseadas na localização física do IP, isto é, pode-se impedir que um ou mais países tentem se conectar ao supercomputador. Este tipo de filtragem diminui sensivelmente alguns tipos de ataques.
- Bloqueio e desbloqueio automático de tentativa de acesso: Para evitar ataques de força bruta para o SSH, tem-se um sistema que bloqueia automaticamente o IP por 30 minutos no caso de três erros da senha e/ou código. Isto permite que caso o evento em questão não seja um ataque, isto é, um falso positivo, o IP será liberado automaticamente sem a necessidade de intervenção da equipe de suporte.
- E outros processos de menor importância.

2.6. IoT (Internet das Coisas)

O uso de sensores externos implementados na infraestrutura do supercomputador juntos com outros sensores que faziam parte da solução permitiu um melhor controle sobre todas variáveis de ambiente. Desta forma, mesmo com uma equipe reduzida, é possível gerenciar um supercomputador remotamente. Apenas para enumerar, alguns dos sensores que fazem parte da solução são:

- Sensor de medição de energia, descrito anteriormente e mostrado na Figura 2.2.

- Sensores de temperatura dentro de todos os nós computacionais e demais servidores, que podem ser acessados pelo comando *ipmitool*.
- Sensores do nobreak que informa carga do nobreak, corrente, etc.

Com os diversos sensores disponíveis foi desenvolvida a coleta e tratamento desses dados de maneira centralizada, agregando informações úteis do sistema como um todo. Desta maneira foi disponibilizado várias informações sobre o supercomputador na nuvem, conforme ilustrado na Figura 2.10. Isso é feito através do protocolo MQTT (*Message Queue Telemetry Transport*), que se tornou o padrão para comunicações de IoT. Essas informações podem ser disponibilizadas como públicas ou privadas.

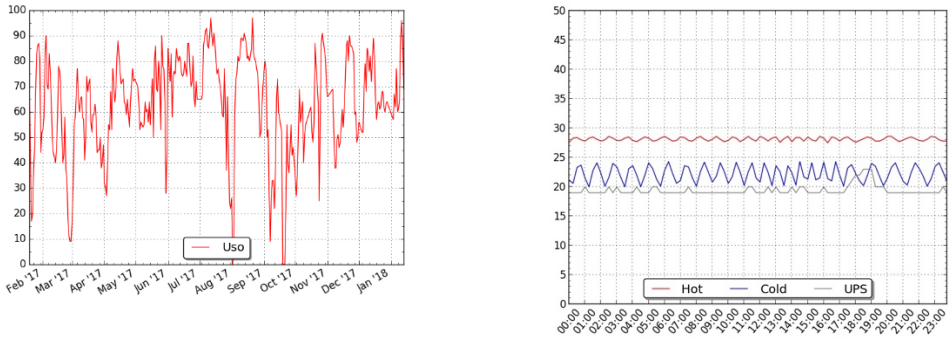
Este protocolo é oferecido em diversos provedores de serviços de nuvem pública, tais como: AWS (Amazon Web Services), Microsoft Azure, Google Cloud, etc. Também é possível, caso se queira, instalar um MQTT broker em um servidor próprio. Bastando para isso instalar o software de código aberto conhecido como Mosquitto [Mosquitto].

De posse dessas informações, elas podem ser consumidas de várias formas. A Figura 2.8 mostra a carga do sistema e a quantidade de jobs sendo executados e na fila. Na Figura 2.17(a) mostra-se a carga do sistema no período de fevereiro de 2017 a janeiro de 2018. Vários dos picos inferiores foram durante a manutenção programada do sistema de refrigeração, cuja capacidade é diminuída pela metade para que a operação de manutenção seja possível sem o desligamento total do sistema. Em consequência, deve-se reduzir a quantidade de nós de processamentos em operação. Os outros picos são devidos a falhas no fornecimento de energia elétrica e, nesses casos, o sistema de proteção desliga todos os equipamentos de forma segura. Note que, por segurança, o religamento é manual.

Alguns dos equipamentos de armazenamento do supercomputador não possuem um botão liga/desliga ou uma maneira de desligar o hardware via software. Neste caso, foi implementado um *smart relay* para desligar de forma automática o disjuntor de energia, após o término do correto do processo de shutdown do serviço. Esta controladora também é usada para ligar o disjuntor. A Figura 2.17 mostra uma foto do *smart relay*.

Figura 2.17: Smart Relay.

A Figura 2.17(b) mostra o histórico diário das temperaturas coletadas nos sensores localizados na sala do UPS e nos corredores quente e frio em um dia típico de operação. Este gráfico é gerado diariamente e enviado por e-mail para a equipe de suporte.



(a) (b)

Figura 2.17: (a) Utilização do supercomputador; (b) Histórico da temperatura da UPS, e dos corredores quente e frio.

Como essas informações estão disponibilizadas na nuvem, elas podem ser consumidas onde for necessário. A Figura 2.18 mostra um exemplo de coleta de informações utilizando um celular.

Figura 2.18: Exemplo de coleta de informações através de IoT usando o celular.

Segue abaixo uma pequena descrição dos dados mostrados na Figura 2.18.

- Primeira linha informa o status do supercomputador se ele está ou não ligado e informa a temperatura do corredor frio e quente.
- Segunda linha informa a quantidade de jobs que estão sendo executados, os jobs que estão na fila e a carga do sistema.
- Terceira linha informa carga do UPS e carga da bateria; e informação da última atualização das informações na nuvem.
- Quarta linha mostra informações sobre as tensões de entrada das três fases do nobreak.

Para consultar estas informações basta abrir o aplicativo e visualizar os dados, pois a autenticação é feita de forma automática.

2.7. Considerações Finais

O presente trabalho teve por finalidade trazer informações sobre integração sobre diversos serviços, como: Portal de Usuários, segurança e uso de IoT em ambientes de supercomputação.

Foram abordadas algumas tecnologias que podem ser utilizadas para aumentar a segurança, como por exemplo o MFA, que permite uma autenticação com dois fatores, automação do sistema de detecção de intrusão, firewall GeoIP, etc.

O uso de algumas tecnologias emergentes, que fazem parte da indústria 4.0, como o IoT e o uso da nuvem foram integradas e trazidas para dentro de um centro de supercomputação. A integração dessas informações promoveu um salto de qualidade na gerência e operação desassistida do supercomputador Lobo Carneiro. Desde a sua instalação e até agosto de 2018 o supercomputador Lobo Carneiro já processou mais de 108.000 jobs, correspondendo a mais de 1.600.000 SUs, todos com segurança, com a máxima qualidade no serviço, já que foi mantida a integridade do sistema durante todo o período. Já foram atendidas, também até a presente data, mais de 500 requisições de usuários submetidas através do Portal.

Neste trabalho não foram considerados vários itens relativos a ambientes de HPC e centros de supercomputação. Entre estes destacamos:

- Ensinar bons hábitos aos usuários permitindo assim, que se crie uma nova cultura para lidar com a segurança da informação.
- Ajustes no supercomputador para maximizar o desempenho.
- Desenvolvimento de *hooks* para atuar junto com o sistema de fila PBS-Pro.
- Desenvolvimento de diversos scripts para customizar e automatizar tarefas importantes na operação do supercomputador.
- Implementações corriqueiras em sistemas computacionais, principalmente relativas à segurança da informação.
- Detalhes sobre ações de auditoria dos sistemas.
- Listagens de todas as ferramentas utilizadas.
- Detalhes sobre a cibersegurança do supercomputador.

Estes itens, pertinentes, serão objeto de publicações e futuras para disseminarmos as boas práticas de operação e gerência de Centros de Supercomputação no Brasil.

Agradecimentos

Este trabalho contou com o apoio do CNPq, FAPERJ e da PETROBRAS.

Referências

Schwab, K., “A Quarta Revolução Industrial”, Editora Edipro, 2016.

TOP500: https://www.top500.org/system/176647, agosto, 2018.

Dongarra, Jack J., Piotr Luszczek, and Antoine Petitet. "The LINPACK benchmark: past, present and future." Concurrency and Computation: practice and experience 15.9 (2003): 803-820.

PMG: SGI Power Management Guide, version 002, 2016.

Canesin, F.C., “Algoritmos de Integração Temporal para Solução Adaptativa e Paralela das Equações de Navier-Stokes”, Tese de Mestrado, 2017, (http://www.coc.ufrj.br/pt/dissertacoes-de-mestrado/590-msc-pt-2017/8609-fabio-cesar-canesin).

Rajovic, N., et al, “The mont-blanc prototype: an alternative approach for HPC systems”, Proceeding SC '16 Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Article No. 38, Salt Lake City, Utah, November, 2016.

Martin, S., “Total Cost of Ownership and HPC System Procurement”, SC17: Birds of Feather, https://eehpcwg.llnl.gov/assets/sc17_bof_tco_procurement.pdf, 2017.

TACC User Portal, https://portal.tacc.utexas.edu/, agosto 2018.

XSEDE User Portal, https://portal.xsede.org/#/guest, agosto, 2018.

Django: https://www.djangoproject.com, agosto, 2018.

Mosquitto, https://mosquitto.org/, agosto, 2018.

Capítulo 3

Introdução à Computação de Alto Desempenho na Nuvem Computacional

Edson Borin, Charles Boulhosa Rodamilans e Jeferson Rech Brunetta

Abstract

HPC systems are expensive and the access to them is limited to some communities, however, the recent advances in cloud computing technologies are allowing anyone to access HPC resources by paying only for the use of the system. The main cloud computing providers are offering high performance computing services, such as machines with GPUs and machines with high computational power interconnected by a high performance network to support the execution of tightly coupled applications. Regarding conventional HPC, performed on clusters of computers, the user's benefits by using cloud computing services are: (a) no wait on queues to use the system; (b) significant time reduction in system deployment; (c) access to high performance resources without a high initial investment; (d) use of specific resources for each application, such as the use of machines with GPUs. The main goal of this course is to provide an introduction to high performance computing on cloud computing resources and demonstrate for the participants how cloud computing can be used to execute high performance programs that make use of technologies such as GPU and MPI.

Resumo

Sistemas de HPC são custosos e o acesso aos mesmos é geralmente limitado a uma pequena parcela da comunidade, entretanto, avanços recentes em tecnologias de computação na nuvem estão permitindo que qualquer um possa acessar recursos de HPC pagando apenas pelo uso do sistema. Os principais provedores de computação em Nuvem estão fornecendo serviços voltados para a computação de alto desempenho, tais como máquinas com GPU e máquinas com alto poder computacional interligadas por uma rede de alto desempenho para dar suporte à execução de aplicações fortemente acopladas. Em relação à HPC convencional, realizada em aglomerados de computadores, ao utilizar estes serviços da Nuvem Computacional, o usuário se beneficia por: (a) não esperar na fila

para utilização do sistema; (b) haver uma redução significativa no tempo para implantação do sistema; (c) ter acesso aos recursos de alto desempenho sem um investimento inicial elevado; e (e) possibilitar o uso de recursos específicos para cada aplicação, como, por exemplo, o uso de máquinas com GPU. O principal objetivo deste minicurso é prover uma introdução à computação de alto desempenho em recursos na nuvem computacional e demonstrar aos participantes como a nuvem pode ser usada para executar programas de alto desempenho que fazem uso de tecnologias como GPU e MPI.

3.1. Introdução a HPC e tipos de aplicações

Desde a invenção dos primeiros computadores eletrônicos de propósito geral, na década de 40, até o início dos anos 90, os supercomputadores, utilizados na computação de alto desempenho, ou *High Performance Computing* (HPC), eram máquinas altamente especializadas, com processadores vetoriais projetados especificamente para executar rapidamente um nicho de aplicações científicas e de engenharia. Com a evolução e popularização dos processadores destinados a computadores pessoais e estações de trabalho, a partir da década de 90 os supercomputadores passaram a ser projetados como aglomerados de computadores com processadores de propósito geral, disponíveis no mercado.

Aglomerados, ou ***clusters***, de computadores são sistemas computacionais formados por computadores organizados em *racks*, sendo que cada computador possui memória própria. Ao longo dos anos surgiram diversas abordagens para facilitar a programação deste tipo de sistema, mas a que se tornou mais popular foi a ***Message Passing Interface* (MPI)**, que oferece funções para que o programador coordene manualmente o fluxo de dados entre os processos que executam em paralelo nos diferentes nós de processamento do *cluster*.

Nos anos 90, o desempenho dos processadores de propósito geral crescia exponencialmente, dirigido principalmente pelo aumento da frequência de operação dos processadores. Entretanto, no início dos anos 2000, a indústria de semicondutores encontrava cada vez mais dificuldades para aumentar a frequência de operação dos processadores e sinalizou que a maneira de continuar oferecendo processadores com desempenho cada vez melhor seria através da introdução de múltiplos núcleos computacionais em um mesmo processador, os processadores ***multi-core***. Dessa forma, a partir de 2005, os fabricantes de processadores começaram a produzir processadores *multi-core*, sendo que os núcleos computacionais compartilhavam a memória do computador. Nesta mesma época, os supercomputadores, que eram baseados em processadores de propósito geral, também começaram a fazer uso de processadores *multi-core*.

Os programas que eram codificados com MPI podiam tirar proveito naturalmente dos múltiplos núcleos computacionais dos supercomputadores através do lançamento de múltiplos processos no mesmo nó computacional (um processo para cada núcleo computacional, ou *core*). No entanto, esta abordagem consome mais memória e, em alguns casos, é mais lenta do que as alternativas que fazem uso direto da memória compartilhada entre os *cores* e o modelo de programação para memória compartilhada **OpenMP** se tornou mais popular para a programação paralela dos múltiplos núcleos dentro de cada computador do *cluster*. Dessa forma, os programas desenvolvidos para os supercomputadores passaram a fazer uso de MPI para a comunicação entre os nós do *cluster* e OpenMP

para a programação paralela dos múltiplos núcleos computacionais dentro de um mesmo nó.

Nos anos 2000 também houve uma evolução significativa no *hardware* das placas de processamento gráfico, ou **GPUs**[1], que deixaram de ter funcionalidade fixa e passaram a ser programáveis. Neste mesmo período, pesquisadores verificaram que era possível e vantajoso utilizar o poder computacional destes dispositivos para realizar processamento científico [Luebke et al. 2004]. Como consequência, a indústria evoluiu o *hardware* das GPUs e desenvolveu linguagens e ferramentas para facilitar a programação destes dispositivos para resolver problemas de propósito geral, dando origem ao termo ***General Purpose Graphics Processing Unit* (GPGPU)** [Owens et al. 2007]. A partir de 2009 as GPGPUs passaram a ser integradas nos supercomputadores para serem utilizadas como aceleradores de código. Em resposta a este movimento, a Intel desenvolveu o Xeon PHI, um acelerador em *hardware* feito para competir com as GPGPUs.

As GPGPUs possuem um *hardware* bem diferente dos processadores *multi-core* e no início não era possível converter automaticamente programas em C, C++ ou Fortran para serem executados de forma eficiente nas GPGPUs, dessa forma, novas linguagens de programação surgiram. Uma das primeiras linguagens foi o **CUDA**, da NVidia, no entanto, outras abordagens como **OpenCL** [Du et al. 2012], **OpenACC** e **OpenMP 4.0** surgiram ao longo do tempo. Neste novo ecossistema, os supercomputadores passaram a ser programados com MPI + X, onde X pode ser uma composição de OpenMP, CUDA, OpenCL, *etc*. A Figura 3.1 ilustra a evolução dos supercomputadores ao longo dos anos, incluindo os dispositivos computacionais e suas respectivas tecnologias de programação.

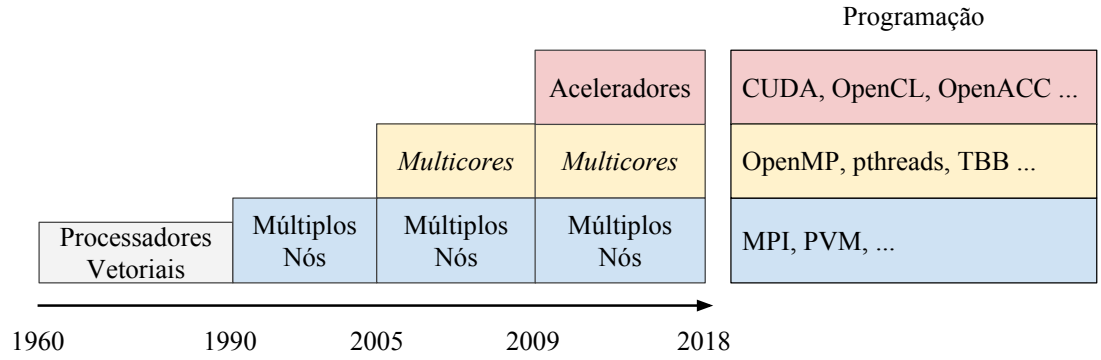


Figura 3.1: Evolução dos supercomputadores ao longo dos anos, incluindo os dispositivos computacionais e suas respectivas tecnologias de programação.

As aplicações que demandam processamento de alto desempenho podem ser classificadas em "aplicações fortemente acopladas" e "aplicações fracamente acopladas", descritas a seguir.

3.1.1. Aplicações fortemente acopladas vs fracamente acopladas

A evolução dos supercomputadores foi dirigida principalmente por aplicações caracterizadas como "**fortemente acopladas**", ou *Tightly-Coupled Application* (TC-A), que exigem uma rede de interconexão de alto desempenho. Muitas destas aplicações imple-

[1] do inglês: *Graphic Processing Unit*.

mentam algoritmos projetados com o modelo de programação ***Bulk Synchronous Parallel* (BSP)** [Valiant 1990], que trabalha em rodadas intercaladas de a) computação isolada nos nós computacionais e b) troca de informações entre os nós, como ilustrado na Figura 3.2. Neste modelo de programação, o algoritmo faz uso de barreiras de sincronização para fazer com que o programa aguarde até que todos os nós terminem uma rodada antes de iniciar a próxima.

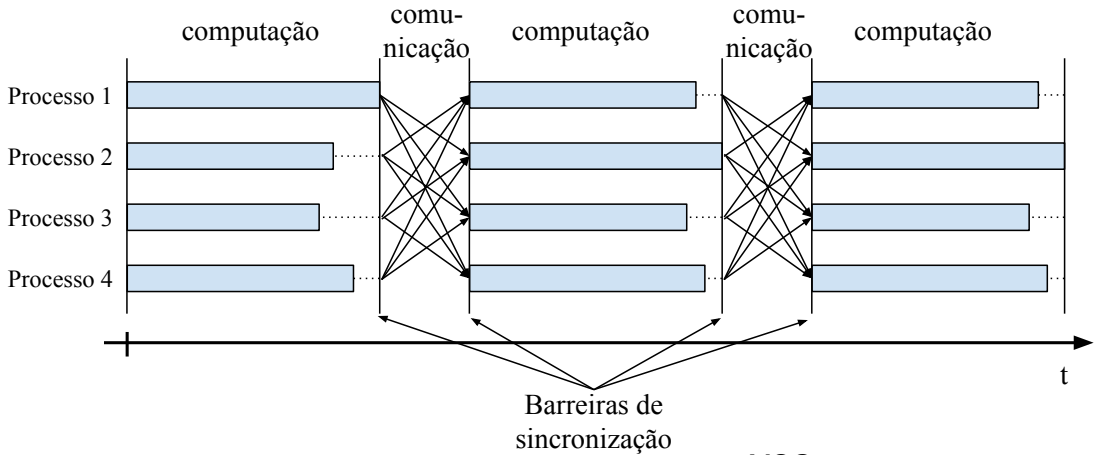


Figura 3.2: Execução de aplicações fortemente acopladas em rodadas de computação e comunicação.

Note que uma rede de interconexão lenta poderia fazer com que o tempo gasto nas rodadas de comunicação de aplicações fortemente acopladas (TC-A) aumentasse demasiadamente o tempo de execução da aplicação. Como consequência, os *clusters* de HPC são geralmente compostos por computadores de alto desempenho conectados por uma rede especializada de alto desempenho, com o padrão InfiniBand, por exemplo. A biblioteca de programação MPI oferece suporte à sincronização e comunicação entre os processos de aplicações que executam em *clusters* de HPC e tem sido amplamente utilizada para codificação de aplicações fortemente acopladas.

Existem diversos tipos de problemas que demandam um alto poder computacional mas que podem ser paralelizados sem que haja muita comunicação entre os diferentes nós de computação. Estes problemas, também conhecidos como "**embaraçosamente paralelos**", permitem a implementação de aplicações "**fracamente acopladas**", ou *Loosely-Coupled Application* (LC-A), que são facilmente paralelizáveis e podem ser executadas de forma eficaz em aglomerados de computadores conectados por uma rede de interconexão comum, sem garantias de desempenho.

Por não precisarem de comunicação de alto desempenho, as aplicações fracamente acopladas (LC-A) também podem ser executadas em federações de *clusters* e estações de trabalhos conectados através da Internet. Este tipo de computação, que pode fazer uso de sistemas ociosos e geograficamente dispersos, é conhecido como **computação em grade**[2]. Este modelo de computação paralela se tornou amplamente conhecido por aplicações de computação voluntária famosas como o projeto SETI@home, que faz uso

[2]do inglês: *grid computing*.

de computadores ociosos[3] para processar dados em busca de inteligência extraterrestre.

Na computação em grade o sistema computacional pode aumentar ou diminuir de tamanho dinamicamente, durante a execução das aplicações. Dessa forma, é importante que as aplicações desenvolvidas para este tipo de sistema sejam **tolerantes a falhas**, para que a falha ou remoção de nós do sistema não cause falha ou interrompa a aplicação como um todo. Também é importante que as aplicações sejam capazes de fazer uso de novos recursos computacionais, integrados ao sistema após a aplicação ter sido iniciada.

3.1.2. Aplicações com acesso intensivo a dados

Aplicações com acesso intensivo a dados fazem acesso a arquivos de dados intensivamente durante a execução. Para este tipo de aplicação, é importante que o sistema computacional tenha mecanismos de alto desempenho para acesso aos arquivos. Caso contrário, o tempo de execução total da aplicação poderia ser afetado significativamente pelo atraso no acesso aos dados. As seguintes abordagens podem ser utilizadas para permitir o acesso com alto desempenho a arquivos:

1. Armazenamento dos arquivos em um sistema de arquivos centralizado e compartilhado. Neste caso, os arquivos são armazenados em um subsistema especializado e, à medida que os nós computacionais tentam acessar os arquivos, o conteúdo destes é transferido pela rede.

2. Armazenamento dos arquivos em um sistema de arquivos distribuído e compartilhado. Neste caso, os arquivos são armazenados de forma distribuída nos próprios nós computacionais. À medida que os processos que executam nos nós computacionais tentam acessar os arquivos, os dados são lidos da unidade de armazenamento local (caso haja uma cópia do arquivo na máquina) ou transferidos de outro nó computacional pela rede. A replicação parcial ou total de arquivos é comum nesta abordagem.

3. Armazenamento de cópias dos arquivos na unidade de armazenamento local do nó computacional. Neste caso, os dados devem ser copiados para as unidades de armazenamento dos nós computacionais antes da computação ser iniciada.

A primeira e a segunda abordagem são transparentes para o usuário, já que o mesmo acessa os arquivos como se eles existissem no nó computacional onde o processo é executado, mas elas colocam mais pressão na rede de comunicação do sistema, já que esta é utilizada para transmitir o conteúdo dos arquivos à medida que eles são lidos. A terceira abordagem, por outro lado, não utiliza a rede do sistema para transferência de arquivos durante a execução da tarefa e pode oferecer maior paralelismo, no entanto, o usuário deve coordenar a cópia dos dados de entrada (saída) para os (dos) nós computacionais do sistema.

A intensidade com que a aplicação acessa os dados é inversamente proporcional à "**intensidade operacional**" da aplicação, que corresponde ao número de operações realizadas para cada *byte* de dados lido. Quanto maior a "intensidade operacional", mais

[3] A ativação do protetor de tela do computador do usuário é muitas vezes usada como gatilho para indicar que o sistema está ocioso.

tempo o sistema passa processando e menos tempo acessando dados dos arquivos. Dessa forma, a intensidade de acesso aos dados não está ligada diretamente ao tamanho do dado de entrada. Observe, por exemplo, que uma aplicação que acessa dados grandes (p.ex. dados com *terabytes* ou *petabytes*) mas tem alta intensidade operacional não faz acesso intensivo aos dados.

Aplicações com baixa intensidade operacional, que fazem poucas operações de processamento para cada *byte* lido do arquivo, são sensíveis ao desempenho do subsistema de leitura e escrita de arquivos. Aplicações de *Big Data*, que fazem busca e mineração em grandes volumes de dados, são exemplos de aplicações com baixa intensidade operacional pois acessam um volume muito grande de dados e a quantidade de operações realizadas por *byte* lido é pequena. Este tipo de aplicação é comumente acelerado com sistemas que mantêm os dados armazenados de forma distribuída, em múltiplos nós de processamento, e movem o código para execução nos nós distribuídos. O sistema Hadoop é um exemplo de sistema que processa dados com esta abordagem.

3.1.3. Considerações sobre o desempenho de aplicações de alto desempenho

O desempenho de uma aplicação de alto desempenho depende das características da aplicação e do sistema computacional que será utilizado para a execução desta. Características como o número de núcleos computacionais, a frequência de operação da CPU, a quantidade de vias nas unidades vetoriais, o tamanho das *caches* do processador, a vazão e a latência da memória principal, dos dispositivos de armazenamento secundários e da rede de interconexão e até mesmo as características de aceleradores em *hardware* (p.ex. GPUs) podem afetar o desempenho de uma aplicação de alto desempenho. Entretanto, geralmente é um subconjunto pequeno destas características (muitas vezes apenas uma) que causa o maior impacto no desempenho da aplicação e este subconjunto depende das características da aplicação. Por exemplo, como discutido anteriormente, o desempenho de aplicações fortemente acopladas pode ser fortemente afetado pelo desempenho da rede de interconexão.

Entender como a aplicação interage com os diferentes recursos do sistema computacional, identificar os principais gargalos (limitantes de desempenho) e otimizar o código para o sistema em questão é geralmente uma tarefa que demanda bastante conhecimento da aplicação e do sistema computacional. Esse processo pode envolver o desenvolvimento de modelos de desempenho, a realização de experimentos de desempenho e a modificação do código da aplicação. Dessa forma adequar uma aplicação a um *cluster* de alto desempenho pode ser um desafio à parte.

Apesar do processo de otimização de aplicações para *clusters* específicos estar fora do escopo deste minicurso, podemos fazer as seguintes considerações gerais:

- Aplicações fortemente acopladas geralmente dependem de um bom desempenho da rede de interconexão. Dessa forma, a execução deste tipo de aplicação em sistemas com uma rede de interconexão de alto desempenho pode ser mais vantajosa. É importante notar que, em alguns casos, a rede só se torna o gargalo do sistema quando o número de nós computacionais utilizados para a computação é muito grande. Dessa forma, aplicações fortemente acopladas também podem funcionar

bem em sistemas com redes de interconexão simples quando a quantidade de nós computacionais não for muito grande.

- Aplicações fracamente acopladas geralmente não dependem muito do desempenho da rede de interconexão. Nestes casos, é mais vantajoso investir em outros tipos de recursos computacionais, como CPUs, memória e/ou dispositivos de armazenamento secundários de alto desempenho.

- Aplicações programadas com MPI podem tirar proveito de sistemas com múltiplos nós e de nós com múltiplos núcleos computacionais, ou *multi-core*. Para isso, basta executar um processo por núcleo computacional, ou *core*. Neste cenário, há aplicações que atingem melhor desempenho em sistemas com muitos nós com poucos núcleos computacionais e outras que executam de forma mais rápida em poucos nós com muitos núcleos computacionais. Vale à pena realizar testes para determinar qual é a melhor estratégia para a aplicação de interesse.

- Aplicações com baixa intensidade operacional colocam mais pressão no sistema de entrada e saída. Nestes casos, é importante otimizar o acesso aos arquivos; por exemplo, através da replicação destes nos dispositivos de armazenamento local dos nós computacionais. A aquisição de dispositivos de armazenamento secundários com melhor desempenho, como *Flash Drives*, pode melhorar ainda mais o desempenho nestes casos.

- Aplicações ***CPU-bound***, que dependem mais do desempenho da CPU do que dos outros componentes do sistema, geralmente podem ser aceleradas significativamente com o uso de aceleradores em *hardware*, como GPUs. No entanto, para isso, a aplicação deve ter sido codificada para fazer uso destes recursos com linguagens como CUDA ou OpenCL, por exemplo.

3.2. Computação na Nuvem

3.2.1. Definição e modelos de negócios

De acordo com a definição do NIST [Mell and Grance 2011], computação na Nuvem é um modelo para permitir acesso ubíquo, conveniente e sob demanda via rede a um conjunto de recursos computacionais configuráveis (p. ex: redes de computadores, servidores, armazenamento, aplicações, e serviços) que possam ser provisionados e descartados rapidamente e com pouco esforço de gerenciamento ou interação com o provedor de serviço. Este modelo é composto por cinco características essenciais:

- Autosserviço sob demanda: O cliente pode provisionar ou descartar recursos de computação, como armazenamento e processamento, sob demanda sem a necessidade de interagir com humanos durante o processo;

- Acesso amplo pela rede: O serviço deve estar disponível para uso através da rede de computadores por mecanismos padronizados;

- *Pool* de recursos: Os recursos computacionais do provedor devem atender a múltiplos clientes, com diferentes recursos físicos e virtuais atribuídos e re-atribuídos dinamicamente para os clientes sob demanda;

- Elasticidade rápida: Recursos devem ser provisionados e descartados de forma elástica e rápida, e até automaticamente em alguns casos, para permitir um ajuste dinâmico à demanda de serviço. Este tipo de característica permite que clientes adequem os recursos computacionais à sua necessidade ao longo do tempo. A Figura 3.3 ilustra como a elasticidade rápida pode trazer benefícios como redução de custo ou aumento da qualidade de serviço. A Figura 3.3a representa um cenário onde o sistema foi dimensionado para suprir a demanda de pico e nos momentos onde a demanda não é alta o recurso computacional é subutilizado, representado pela área escura. A Figura 3.3b ilustra as situações onde as projeções iniciais de demanda de recursos foram infladas e o sistema adquirido passa a ser sempre subutilizado. A Figura 3.3c ilustra o caso oposto, onde o sistema é subdimensionado. Neste caso, os usuários têm uma qualidade de serviço inferior durante os períodos de alta demanda, o que pode causar atrasos ou mesmo decréscimo na produtividade. Um sistema com elasticidade rápida é ilustrado na Figura 3.3d, onde a quantidade de recursos é ajustada rapidamente para se adequar à demanda do cliente;
- Serviços mensuráveis: Os serviços oferecidos pelo provedor devem ser mensuráveis, de forma a permitir o monitoramento e controle automático do uso dos recursos.

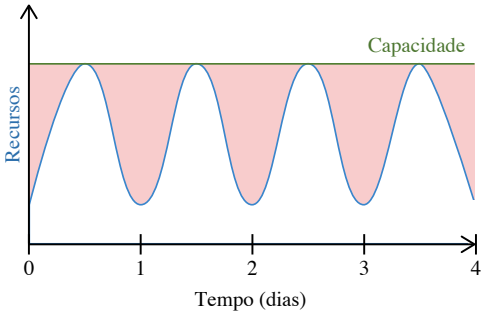
Em suma, estas características permitem que a Nuvem ofereça acesso sob demanda, conveniente e ubíquo a seus serviços de forma que a quantidade de recursos possa ser incrementada ou reduzida rapidamente com pouco esforço de gerenciamento e nenhuma interação humana com o provedor de serviços.

3.2.1.1. Modelos de serviço e implantação

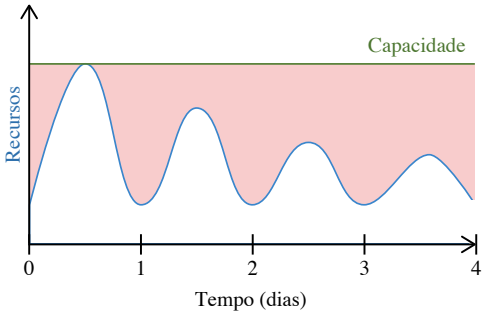
Os serviços oferecidos em uma Nuvem Computacional são classificados em três modelos principais [Mell and Grance 2011, Puthal et al. 2015]:

- *Software* como Serviço, ou *Software as a Service* (SaaS): neste modelo o serviço oferecido é um *software* que é executado pelo provedor da Nuvem. Gmail, Google Drive e Bing são exemplos de aplicações que empregam o modelo *Software* como Serviço.
- Plataforma como Serviço, ou *Platform as a Service* (PaaS): neste modelo o serviço oferecido é uma plataforma de *software* base para o desenvolvimento ou instalação de aplicações pelo cliente. Neste modelo, a maior parte da pilha de *software*, como o sistema operacional e as bibliotecas de *software*, é gerenciada pelo provedor e o cliente não tem que se preocupar com atividades como atualizações das bibliotecas e do sistema operacional. O Google App Engine[4] é um exemplo de plataforma como serviço.
- Infraestrutura como Serviço, ou *Infrastructure as a Service* (IaaS): neste modelo de serviço o cliente recebe a infraestrutura bruta, como máquinas virtuais e espaços

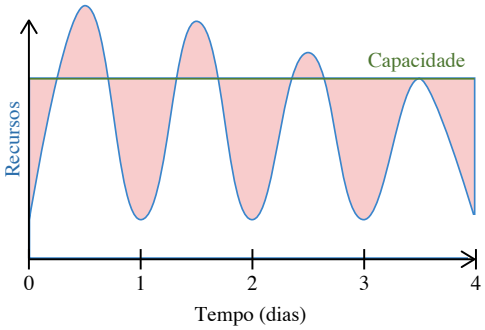
[4] `http://code.google.com/appengine`



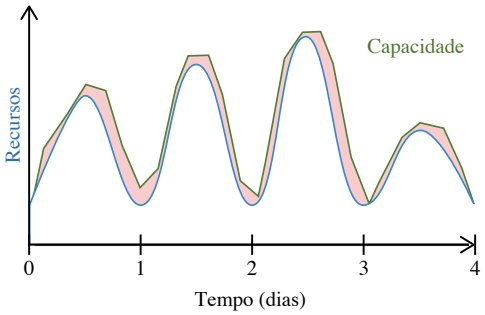
(a) Infraestrutura de computação adquirida com dimensionamento baseado na demanda de pico. Ocorre o desperdício de recursos durante períodos de baixo uso.



(b) Infraestrutura de computação adquirida com superdimensionamento causado por projeções infladas de demanda.



(c) Infraestrutura subdimensionada. A qualidade de serviço inferior durante os períodos de alta demanda pode causar atrasos ou mesmo decréscimo na produtividade.



(d) Infraestrutura com ajuste elástico rápido, que permite o ajuste dinâmico dos recursos de acordo com a demanda.

Figura 3.3: Cenários de provisionamento de recursos. A área escura representa recursos não utilizados ou sub-provisionados. Figuras (a), (b) e (c) mostram casos onde não há elasticidade, fazendo com que os recursos sejam sub ou superdimensionados. A Figura (d) mostra um caso onde é possível realocar recursos ao longo do tempo de acordo com a demanda real.

para armazenamento, e é responsável pela instalação e gerenciamento da pilha de *software* que será executada no sistema. Este modelo é geralmente mais flexível e possui um custo de uso menor, entretanto, os custos associados ao gerenciamento da pilha de *software* do sistema ficam a cargo do cliente.

A Figura 3.4 compara a responsabilidade do gerenciamento dos recursos em um *cluster* privado e nos diferentes modelos de serviços na Nuvem Computacional. Note que no *cluster* privado a responsabilidade de gerenciamento dos recursos (incluindo *software* e *hardware*) é integralmente do usuário (proprietário) enquanto que no modelo Infraestrutura como Serviço na Nuvem Computacional o usuário é responsável por gerenciar apenas parte do *software* (Sistema Operacional - S.O., *Runtime*, Dados e Aplicações).

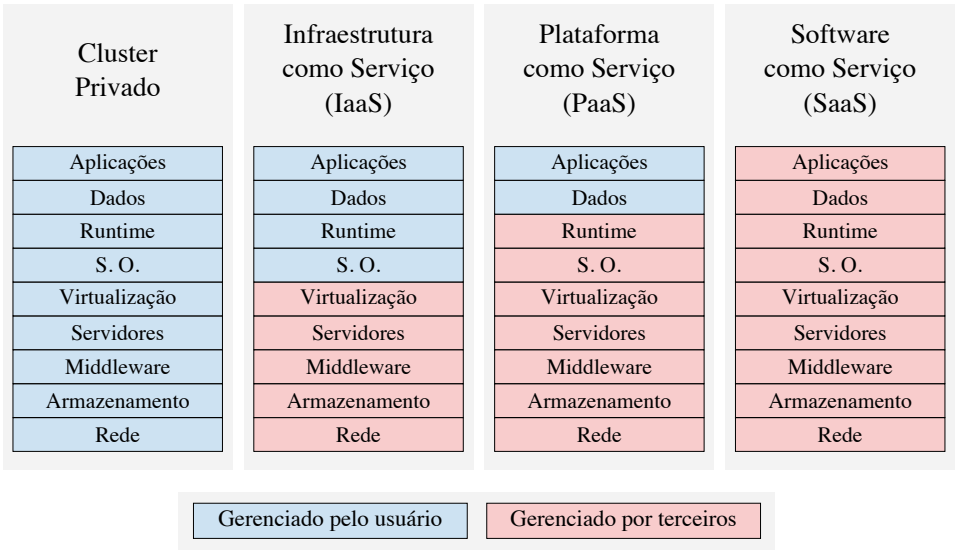


Figura 3.4: Gerenciamento de recursos em um *cluster* privado e nos diferentes modelos de serviço na nuvem.

As nuvens computacionais também podem ser classificadas com relação ao modelo de implantação. No modelo de implantação "Privado", a Nuvem construída sobre uma infraestrutura privada e é acessada exclusivamente pelos proprietários da infraestrutura. Uma empresa grande com diversos setores que fazem uso de recursos computacionais pode fazer uso do modelo "Privado" de Nuvem Computacional para o compartilhamento dos recursos computacionais entre os diferentes setores. No modelo "Público", os serviços são oferecidos ao público em geral e tipicamente cobrado com base nas métricas de uso. A Amazon AWS[5] e a Microsoft Azure [6] são exemplos de nuvens públicas que cobram em função do uso. No modelo de implantação "Híbrido" há a junção de nuvens privadas com nuvens públicas de forma que a Nuvem privada possa ter seus recursos computacionais ampliados em momentos de maior demanda a partir da contratação de recursos da Nuvem pública.

Historicamente, a nuvem computacional tem sido usada amplamente para aplicações *web*, onde *sites* se aproveitam da alta disponibilidade e dos mecanismos de auto dimensionamento, que provedores de nuvem fornecem, para manter os sistemas operantes e com cargas de trabalho distribuída entre diversos nós de computação. Nos primórdios, as tecnologias de emulação de máquina virtual não eram tão eficientes e a rede de intercomunicação entre os nós era rápida para aplicações da Internet, mas não o suficiente para aplicações de alto desempenho. Com os avanços nas tecnologias de virtualização e as vantagens da nuvem computacional, aplicações de alto desempenho, que não requerem uma rede de alta velocidade, começaram a tirar proveito do uso de recursos da nuvem

[5] `http://aws.amazon.com/`
[6] `http://www.microsoft.com/windowsazure/`

computacional. Do outro lado, com a demanda, os provedores destes serviços passaram a disponibilizar infraestruturas de rede de alta velocidade - como InfiniBand.

3.2.2. Considerações de custo e desempenho na Nuvem Computacional

Como discutido na Seção 3.1.3, adequar uma aplicação a um *cluster* de alto desempenho pode ser um desafio à parte. Entretanto, o modelo de serviço da Nuvem Computacional permite que o usuário realize o processo inverso, ou seja, adeque o *cluster* às necessidades da aplicação. Neste contexto, em vez do usuário tentar adequar a aplicação para executar de forma eficaz no *cluster*, o usuário pode testar a aplicação em diferentes configurações de máquinas virtuais na Nuvem Computacional e escolher o "*cluster*" que melhor satisfaz as necessidades da aplicação.

No modelo de Infraestrutura como Serviço, ou *Infrastructure as a Service* (IaaS), o cliente geralmente possui à disposição uma ampla gama de tipos de máquinas virtuais para contratar. Cada tipo possui características de recursos (p. ex.: tipo e número de núcleos computacionais, quantidade de memória, *etc*) e custos diferentes. Sendo que o custo pode variar de poucos centavos até dezenas de dólares por hora de uso. Dessa forma, é muito importante que o usuário escolha máquinas apropriadas para a aplicação que deseja executar. As principais variáveis que afetam o custo das máquinas virtuais são:

- O modelo do processador;
- A quantidade de núcleos (*cores*) do processador;
- A quantidade de memória RAM;
- A quantidade e o modelo dos aceleradores (p.ex. GPUs e FPGAs);

Para a escolha do tipo de máquina virtual com o melhor custo-benefício, o usuário deve analisar as características da aplicação de interesse. Por exemplo, analisar se a aplicação foi codificada para tirar proveito de múltiplos núcleos computacionais ou de GPUs. Note que executar uma aplicação que não foi codificada para usar GPUs em uma máquina virtual que possui GPUs não oferecerá melhor desempenho enquanto que o custo será maior. Alternativamente, em vez de analisar a aplicação, o usuário pode simplesmente "testar" o desempenho da aplicação em diferentes configurações e selecionar a configuração que oferece o melhor custo-benefício.

Além das características mencionadas anteriormente, o desempenho do sistema de armazenamento e o desempenho da rede (ambos contratados pelo cliente), também afetam o custo do processamento. Em particular, o desempenho da rede era uma das características que dificultavam bastante a execução de aplicações de alto desempenho na Nuvem Computacional. De fato, há alguns anos atrás, quando os provedores públicos de serviço de Nuvem Computacional estavam se consolidando, era difícil utilizar serviços da Nuvem Computacional para computação de alto desempenho com aplicações fortemente acopladas por causa da latência da rede. Naquela época, diversos estudos indicavam que as tecnologias de virtualização afetavam significativamente a latência de comunicação entre as máquinas virtuais [He et al. 2010, Mag 2011, Gupta et al. 2013]. O relatório desenvolvido pela Magellan [Mag 2011], por exemplo, apontava que a Nuvem Computacional

poderia ser até 50 vezes mais lenta para um determinado tipo de padrão de comunicação e que as aplicações mais penalizadas eram aquelas que dependiam de baixa latência de rede (fortemente acopladas) enquanto que aplicações que dependiam de vazão da rede, mas não muito da latência (poucas trocas de mensagens grandes), não eram tão afetadas.

Atualmente, o serviço de rede na Nuvem Computacional melhorou e os principais provedores de serviço já oferecem opções especializadas para melhorar o desempenho de aplicações de alto desempenho. Por exemplo, a AWS oferece o Elastic Network Adapter (ENA) [Barr 2016, AWS 2018], um serviço de rede que pode ser configurado para trabalhar com 10 ou 25 Gbps. A Azure oferece o serviço RDMA (Remote Direct Memory Access)[Karmarkar 2015, Azure 2018], que implementa uma rede Infiniband com desempenho de 32 Gbit/s Estes serviços mostram que os provedores estão oferecendo serviços cada vez mais compatíveis com as demandas de aplicações fortemente acopladas e a tendência é que a Nuvem Computacional poderá ser utilizada para qualquer tipo de aplicação de alto desempenho.

Estes são aspectos importantes que devem ser levados em consideração pelo usuário quando estiver contratando serviços na Nuvem Computacional para a execução de aplicações de alto desempenho.

3.3. Executando aplicações de HPC na nuvem

Nesta seção iniciaremos a parte prática do minicurso, entrando em contato com um provedor de nuvem, inicializando e explorando a utilização de sua infraestrutura no modelo IaaS. Atualmente os dois principais fornecedores de serviços de Nuvem Computacional são a Microsoft, com a plataforma *Azure*, e a Amazon com a plataforma *Amazon Web Services* (AWS). O conceito de uso de ambas plataformas é semelhante em muitos aspectos e uma noção do uso em uma delas pode guiar para o uso na outra. Dessa forma, concentraremos as atividades práticas deste minicurso apenas na AWS. Ressalta-se que o processo de escolha do melhor tipo de serviço (incluindo tipos de máquina virtual) para cada aplicação não faz parte do escopo deste minicurso.

3.3.1. Fluxo de trabalho no modelo IaaS

Para se executar aplicações em máquinas virtuais na Nuvem Computacional, o usuário deve realizar uma sequência de passos. Estas atividades estão representadas na Figura 3.5 e incluem:

1. Criar uma conta no *website* do provedor de serviço na Nuvem Computacional;

2. Selecionar a **imagem de máquina virtual** para ser instanciada. A imagem é um arquivo que contém o sistema operacional e pacotes de *software*. Geralmente os provedores oferecem diversas opções de imagens, incluindo imagens com o S.O. Linux e o S.O. Windows. Além disso, algumas dessas imagens possuem *software* pré-instalado. Nestes casos, o provedor pode cobrar a mais pelo uso da imagem em função do conjunto de *software* instalados;

3. Selecionar o **tipo de máquina virtual** que executará a imagem escolhida no passo anterior. O tipo de máquina virtual define os recursos em *hardware*, incluindo o

número de núcleos computacionais, a quantidade de memória, as interfaces de rede, os aceleradores (e.g. GPUs), entre outros;

4. Configurar as interfaces de rede, chaves de acesso, usuários, *etc*;
5. Ligar a máquina virtual, ou VM[7];
6. Conectar-se e utilizar a VM com *software* de acesso remoto, por exemplo, *secure shell* (SSH) ou Remote Desktop;
7. Desconectar e desligar a VM;
8. Destruir a VM.

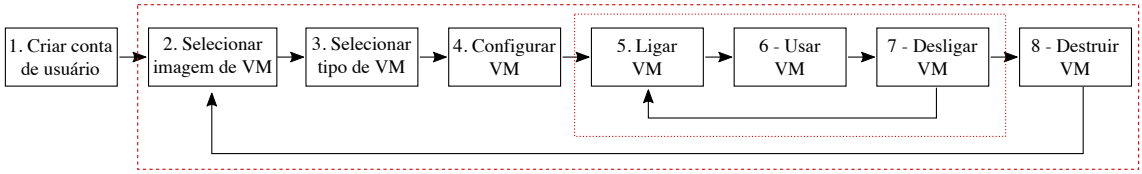


Figura 3.5: Diagrama do fluxo de trabalho no modelo IaaS.

3.3.2. Passos na AWS

3.3.2.1. Criando uma conta no provedor

Como descrito na Seção 3.3.1, o primeiro passo para a execução de código é a criação de uma conta de usuário no *website* do provedor de serviço na Nuvem Computacional. Como este processo pode diferir significativamente para cada provedor, não detalharemos o mesmo aqui.

3.3.2.2. Selecionando um imagem na AWS

Para selecionar uma imagem de VM na AWS, o primeiro passo é acessar os serviços EC2[8] no menu do provedor, como indicado na Figura 3.6.

Após acessar o serviço EC2, você deve acessar a opção "*Launch Instance*" para dar início ao processo de seleção de imagem e tipo de VM, como indicado na Figura 3.7.

Uma vez que a opção "*Launch Instance*" foi acessada, o provedor mostrará uma lista de imagens (chamadas AMIs[9] na AWS) com diferentes configurações de sistemas operacionais e *softwares*, como ilustrado na Figura 3.8.

Selecione a imagem denominada "**Deep Learning AMI (Ubuntu)**" na versão mais recente. A imagem na versão 13 possui o sistema operacional Linux Ubuntu na versão 16.04 LTS e os drivers de CUDA e OpenCL para GPUs NVidia previamente instalados.

[7] do inglês: *Virtual Machine*
[8] do inglês: *Elastic Cloud Computing*.
[9] do inglês: *Amazon Machine Images*.

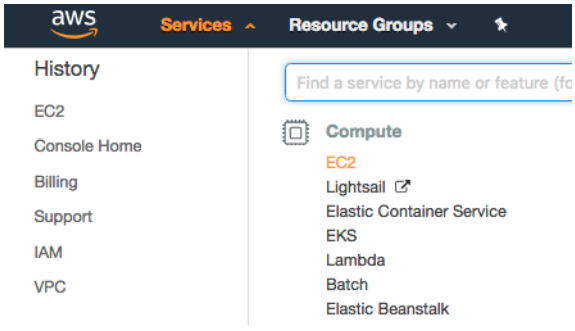


Figura 3.6: Seleção do serviço EC2.

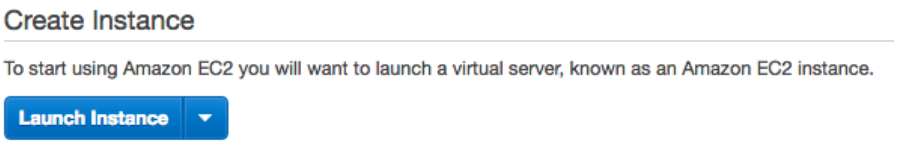


Figura 3.7: Criação de máquina virtual.

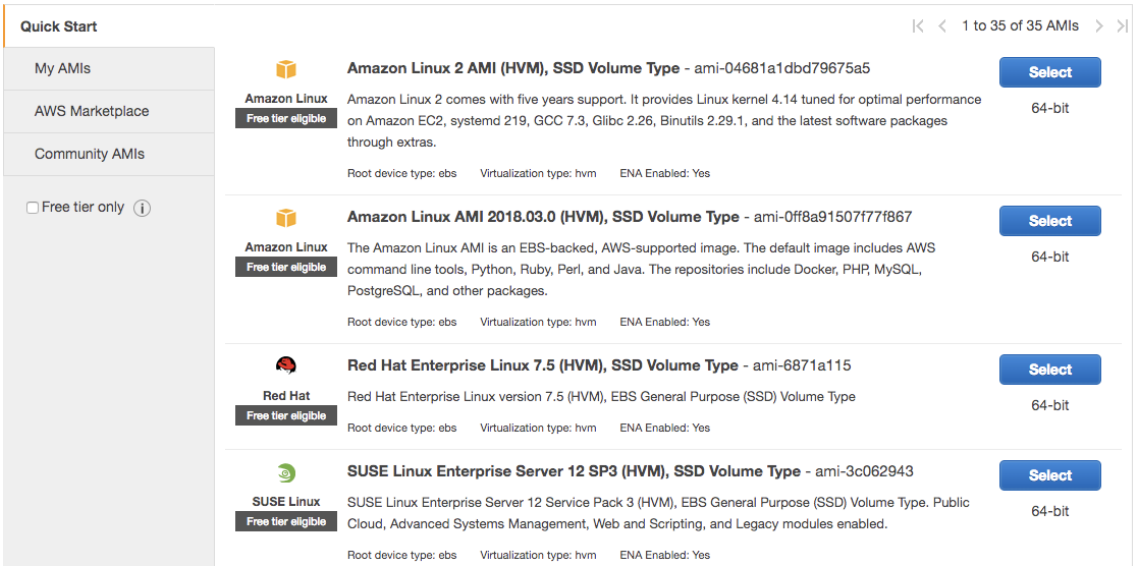


Figura 3.8: Lista de imagens (AMIs) disponíveis.

3.3.2.3. Selecionando um tipo de VM na AWS

Após selecionar a imagem, o provedor exibirá uma tela com múltiplas opções onde você poderá selecionar o tipo de máquina virtual que deseja instanciar, como mostrado na Figura 3.9

Selecione o tipo de máquina virtual intitulada "p2.xlarge", que possui uma GPU NVidia K80, e clique no botão "*Next: Configure Instance Details*" para prosseguir para a etapa de configuração da VM. (**NOTA: cuidado para não clicar em "*Review and Launch*", pois pretendemos configurar a VM antes**)

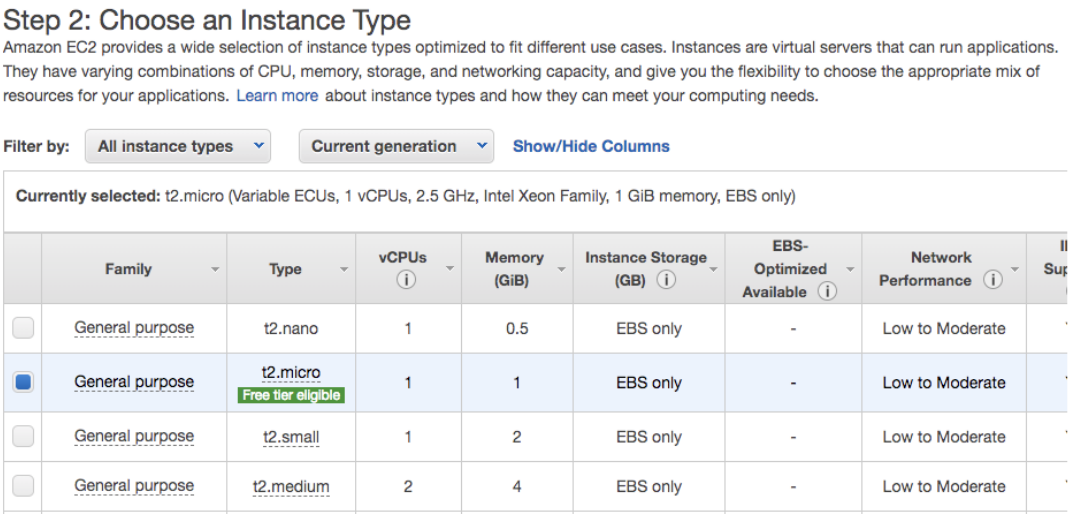


Figura 3.9: Lista de tipos de máquinas virtuais disponíveis.

3.3.2.4. Configurando a VM selecionada

Após selecionar o tipo de máquina virtual, o provedor mostrará um conjunto de telas com parâmetros para configuração da máquina virtual. A Figura 3.10 mostra a primeira tela deste conjunto.

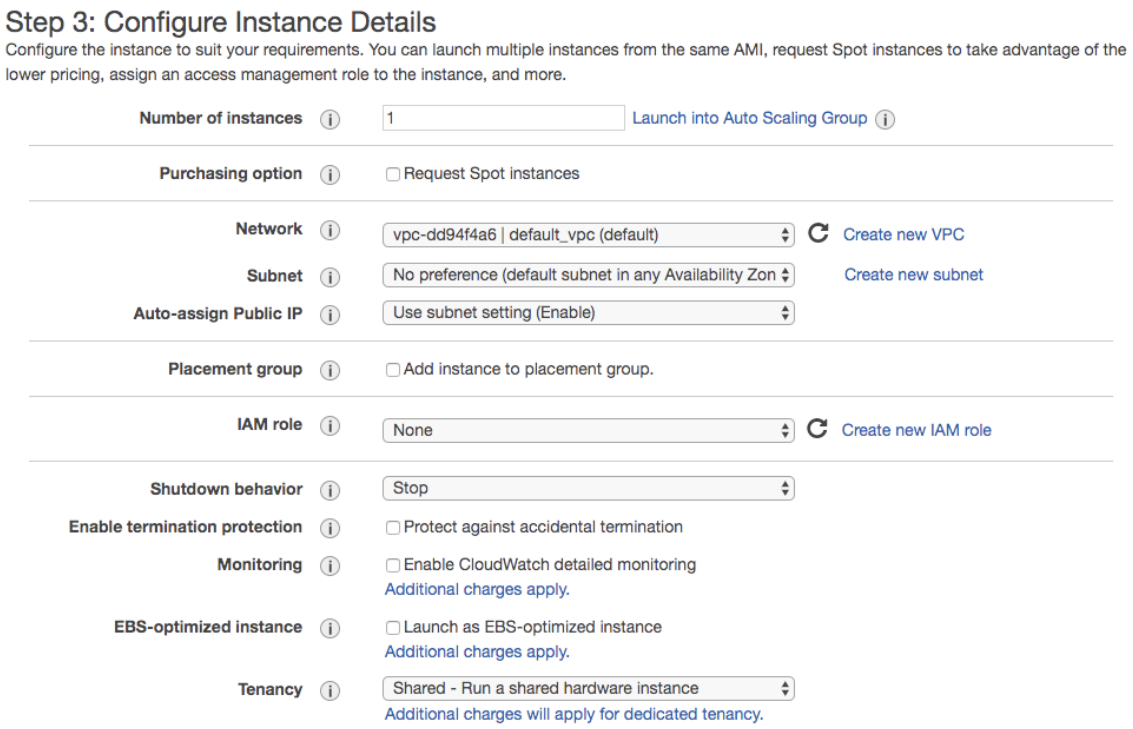


Figura 3.10: Configuração de detalhes da instância.

Para nosso experimento, você não precisará mudar os parâmetros desta primeira tela, que devem se parecer com os da Figura 3.10. Neste ponto, você deve clicar em "*Next:*

Add Storage" (este botão foi omitido da Figura 3.10) para configurarmos o dispositivo de armazenamento da máquina virtual.

A tela seguinte, intitulada "*Add Storage*" permite que o usuário selecione e configure os dispositivos de armazenamento da VM. Para nossos experimentos, utilizaremos o dispositivo padrão, como ilustrado na Figura 3.11. Para prosseguirmos com a configuração selecione a opção "*Next: Add Tags*".

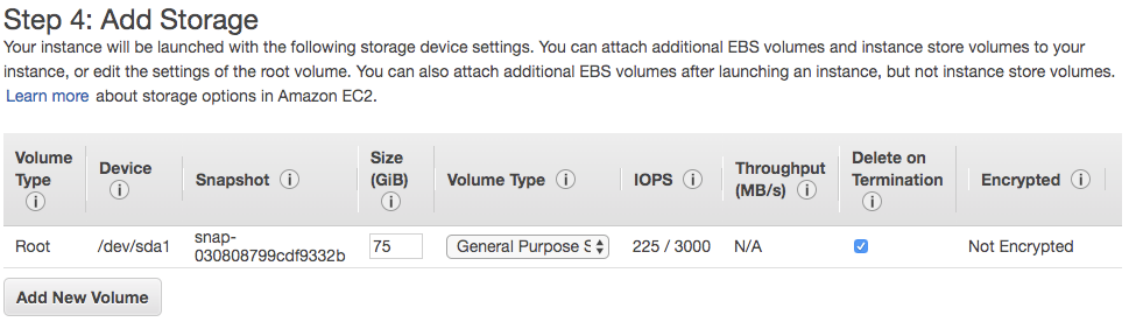


Figura 3.11: Configuração de armazenamento.

Em muitas situações é útil você atribuir *Tags* (etiquetas) às máquinas virtuais. Nesta atividade, você criará uma *Tag* que nos ajudará a identificar os usuários das máquinas virtuais durante o minicurso. Para isso, escreva "*Student*" na caixa "*Key*" e o seu nome (*e.g.* Edson) na caixa "*Value*", como indicado na Figura 3.12. Por fim, clique em "*Next: Configure Security Group*" para avançar para a próxima tela de configuração.

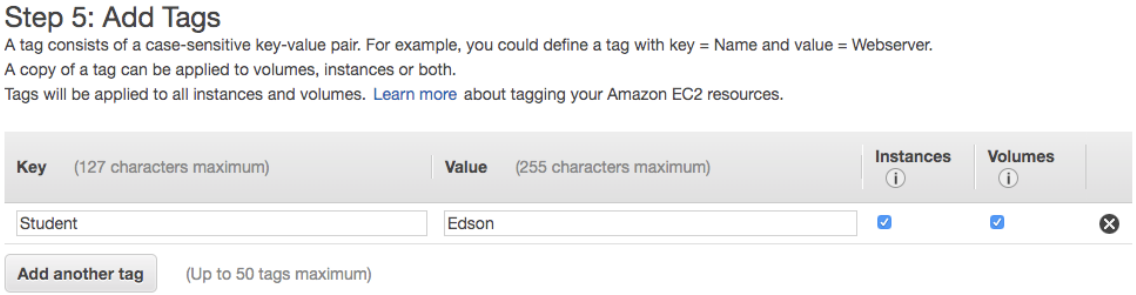


Figura 3.12: Definição de rótulos.

Na tela "*Configure Security Group*", você terá a oportunidade de configurar regras de acesso à máquina, como mostrado na Figura 3.13. Por padrão, o protocolo SSH já está habilitado na porta 22. Para nossos experimentos, não há necessidade de modificar estas configurações. Dessa forma, basta clicar em "*Review and Launch*" para exibir um resumo geral das configurações.

A tela intitulada "*Review Instance Launch*" mostra um resumo das configurações, como ilustrado na Figura 3.14. Você pode revisar as opções selecionadas e, por fim, iniciar a execução da máquina virtual clicando no botão "*Launch*".

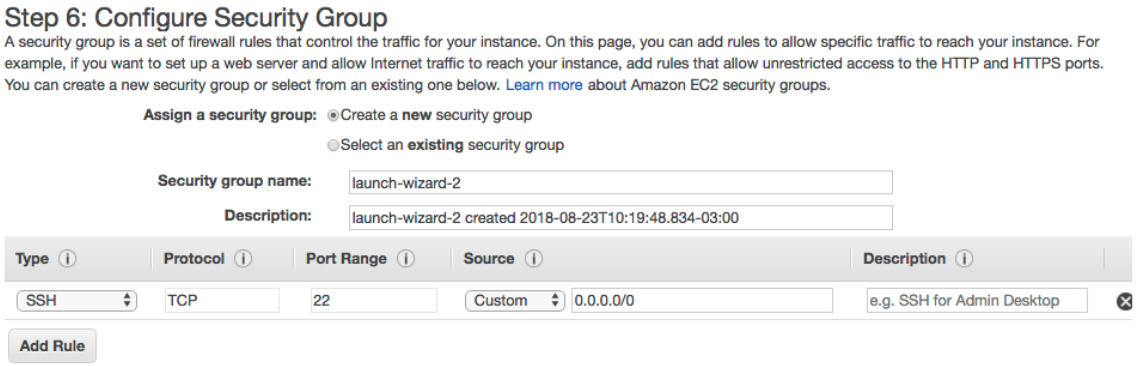


Figura 3.13: Configuração das regras de acesso à rede.

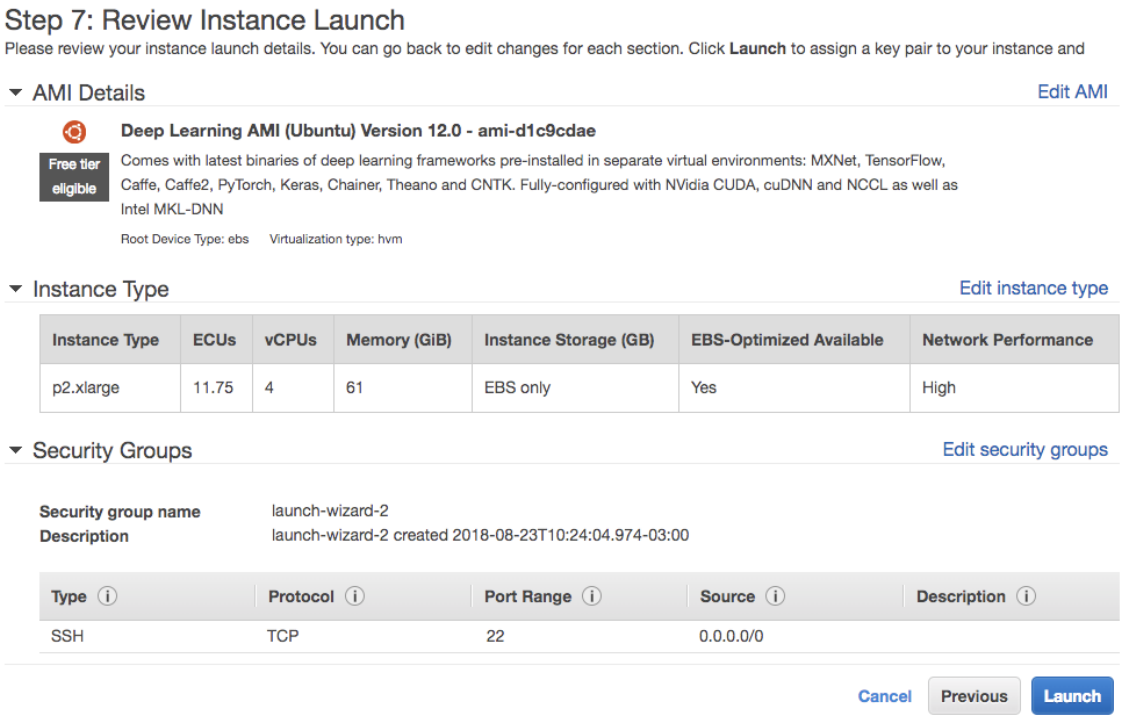


Figura 3.14: Resumo das configurações selecionadas.

3.3.2.5. Ligando a VM

Ao clicar em "*Launch*", antes de iniciar a máquina virtual, o provedor solicitará que você forneça a chave de acesso. Esta chave é importante para garantir que apenas você tenha acesso a esta máquina. Caso você não tenha uma chave, você pode criar selecionando a opção "*Create a new key pair*" e digitando um nome para a sua nova chave, como indicado na Figura 3.15. Após dar um nome à nova chave, você deve realizar o "*download*" desta chave para o seu computador clicando em "*Download Key Pair*". Dica: evite o uso de caracteres especiais como nome de chave.

Após realizar o *download* da chave, clique em "*Launch Instances*" para iniciar a máquina virtual. Neste momento, a máquina virtual será iniciada e o provedor exibirá

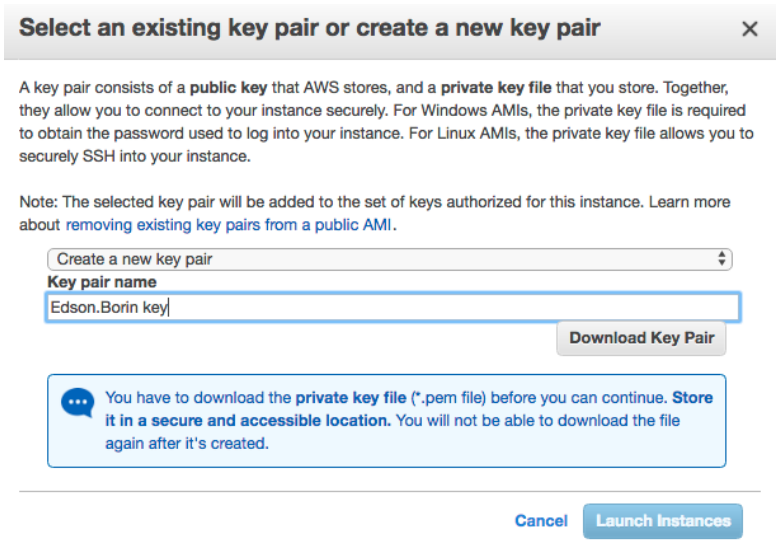


Figura 3.15: Definição da chave de acesso para a máquina virtual.

uma tela com informações sobre o processo, como mostrado na Figura 3.16. Por fins de segurança, esta é a única oportunidade que você terá para realizar o *download* desta chave.

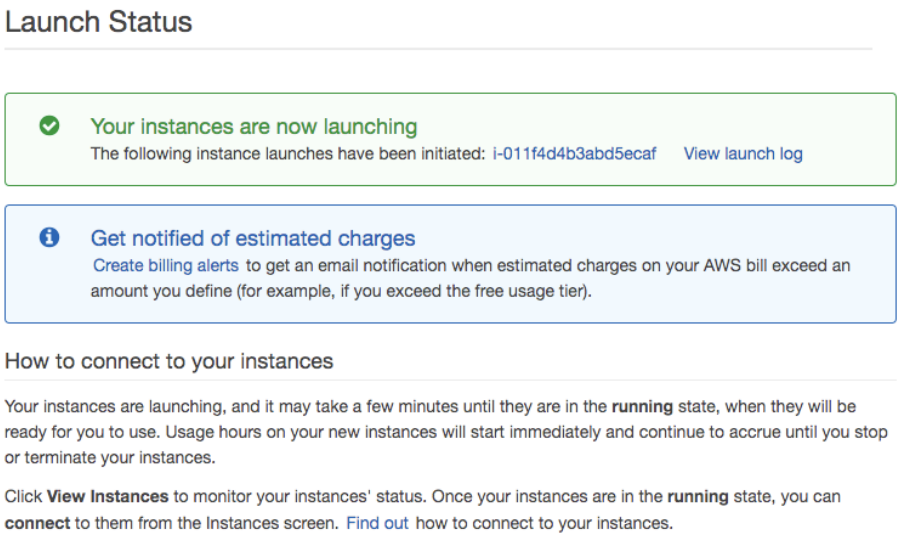


Figura 3.16: Informações da máquina virtual em criação.

Note que enquanto o sistema operacional estiver sendo iniciado (processo de *boot*), a máquina ainda não é acessível. Você pode clicar em "*View Instances*" para visualizar a máquina virtual criada e seu estado. Após o campo "*Status Checks*" exibir 2/2 *checks*, a máquina virtual estará pronta para o acesso remoto.

3.3.2.6. Acessando e utilizando a VM remotamente

A tela "Instances", exibida após você clicar em "*View Instances*" no passo anterior, também pode ser acessada pelo menu "*Instances*", à esquerda do *Dashboard* exibido após a escolha do serviço EC2. Para se conectar remotamente a uma instância, selecione a instância e clique em "*Connect*", como indicado na Figura 3.17

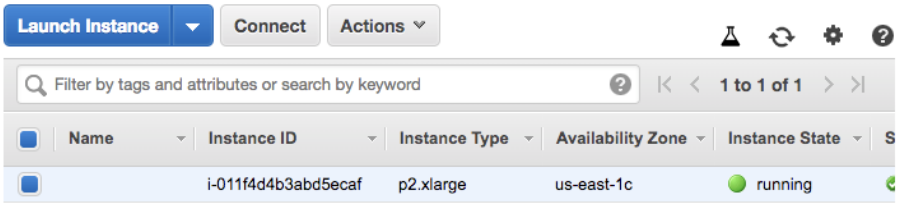


Figura 3.17: Instâncias criadas.

Após clicar em "*Connect*", o *website* exibirá uma tela com instruções de conexão, como mostrado na Figura 13. Observe neste exemplo, que a opção padrão é "*A standalone SSH client*", que pode ser o programa ssh em linha de comando disponível nas distribuições Linux e no sistema OSX, ou o programa PuTTY, comumente utilizado nos sistemas Windows.

Caso você esteja usando o Linux ou o OSX, você deve ajustar as permissões do arquivo de chaves que você obteve nos passos anteriores (Veja os passos 2 e 3 na Figura 3.18). Uma vez que você ajustou as permissões, basta acessar a máquina virtual com o comando sugerido. No caso abaixo foi o comando:

```
ssh -i "Edson.Borin key.pem" ubuntu@ec2-34-227-227-217.compute...
```

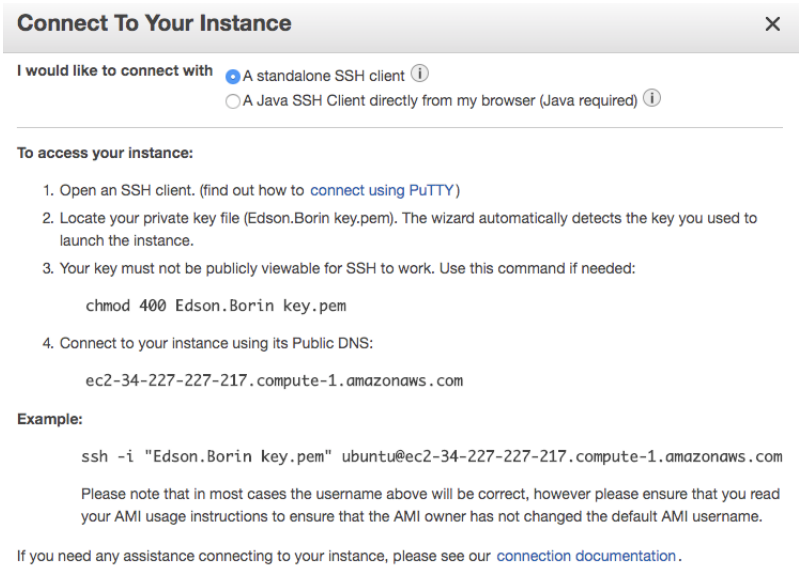


Figura 3.18: Instruções de conexão com a máquina.

Os próximos acessos a esta máquina virtual podem ser realizados repetindo-se este passo. Neste ponto, você pode instalar ou utilizar o seu *software* na máquina virtual.

3.3.2.7. Desligando a VM

Enquanto a máquina virtual estiver ligada, o provedor cobrará pelo serviço. Isso acontece mesmo que você não esteja executando programas dentro da máquina virtual. Por isso, sempre que finalizar o trabalho, é importante desligar a máquina virtual. Para isso, você pode ir na tela "Instances", selecionar a máquina virtual de interesse, clicar em "Actions" e selecionar as opções "Instance State" e "Stop", como ilustrado na Figura 3.19.

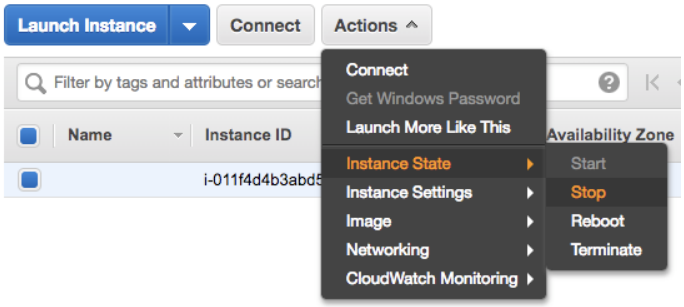


Figura 3.19: Interrupção da máquina virtual.

Uma vez que a máquina virtual foi desligada, o provedor cobra apenas pelo espaço de armazenamento necessário para armazenar o disco da máquina virtual. Observe, que você pode religar a máquina virtual posteriormente que seus dados e programas instalados estarão preservados.

3.3.2.8. Destruindo a VM

Quando a máquina virtual não tem mais utilidade, é importante destruí-la para que o armazenamento da mesma não gere custos. Este processo é muito similar ao processo descrito na Seção 3.3.2.7 ("Desligando a VM"). Neste caso, basta você selecionar a opção "*Terminate*" em vez de "*Stop*". Ainda não destrua esta máquina virtual, a utilizaremos para os demais passos. Note que, uma vez destruída, o conteúdo da máquina virtual não pode ser recuperado.

3.3.3. Estudo de caso 1: Executando a aplicação CMP em máquinas com GPU

Neste estudo de caso, executaremos uma aplicação que faz uso de GPUs em uma máquina virtual. Inicialmente, você deve selecionar a imagem e o tipo de máquina virtual, configurar, ligar e acessar remotamente a máquina virtual, como descrito na Seção 3.3.2. Para este experimento utilizaremos a imagem intitulada "**Deep Learning AMI (Ubuntu)**" na versão mais recente e o tipo de máquina virtual "p2.xlarge".

3.3.3.1. Transferindo e compilando a aplicação CMP

Após acessar a máquina virtual, faça clone do repositório "GPU Seismic Processing" usando o comando:

```
git clone https://github.com/hpg-cepetro/IPDPS-CRS-CMP-code.git
```

Este repositório contém código de duas aplicações de processamento sísmico que funcionam com OpenMP, OpenACC, CUDA e OpenCL [Gimenes et al. 2018]. Estas aplicações foram desenvolvidas por pesquisadores do *High Performance Geophysics lab* (HPG), que se localiza no Centro de Estudos de Petróleo (CEPETRO) da Universidade Estadual de Campinas (Unicamp). Neste minicurso nós executaremos a versão OpenCL/OpenMP da aplicação CMP na GPU/CPU.

Utilizaremos um dado de entrada chamado "simple-synthetic", produzido pelos pesquisadores do HPG para a validação de métodos de processamento sísmico. O dado pode ser copiado com o seguinte comando:

```
wget www.ic.unicamp.br/~edson/minicurso-hpc/simple-synthetic.su
```

Após copiar o repositório, você deve compilar a aplicação. Em nossos experimentos utilizaremos o compilador GCC, disponível na imagem de VM selecionada. Para compilar a aplicação, primeiro você deve entrar no diretório que contém a implementação do CMP em OpenCL utilizando o comando:

```
cd ~/IPDPS-CRS-CMP-code/CMP/OpenCL/
```

Depois, você deve criar um subdiretório intitulado "`bin`", entrar neste diretório, realizar a configuração do sistema de *build* e disparar a compilação com os seguintes comandos:

```
mkdir bin
cd bin
cmake ../
make -j
```

3.3.3.2. Executando a aplicação CMP

Para manter o diretório organizado, criaremos um subdiretório intitulado "`results`" e executaremos a aplicação a partir deste local. Para isso, execute os seguintes comandos:

```
mkdir ../results
cd ../results
```

A aplicação CMP, a ser utilizada em nossos experimentos, requer um conjunto de parâmetros de entrada para configurar o processamento. Estes parâmetros informam propriedades do dado de entrada e definem o espaço de busca do problema a ser resolvido pela ferramenta. Caso seja de interesse, o artigo de Gimenes e outros [Gimenes et al. 2018] apresenta mais detalhes sobre o funcionamento e a implementação desta ferramenta. Em nossos experimentos utilizaremos o dado `simple-synthetic.su` com os parâmetros listados na Tabela 3.1.

Tabela 3.1: Parâmetros de execução da aplicação CMP com o dado `simple-synthetic.su`.

Parâmetro	Valor
`-aph`	600
`-c0`	1.98e-7
`-c1`	1.77e-6
`-d`	1
`-i`	simple-synthetic.su
`-nc`	300
`-tau`	0.002
`-v`	3

Para executar a ferramenta CMP com os parâmetros listados na Tabela 3.1 execute o seguinte comando:

```
time ../bin/cmp-ocl2 -aph 600 -c0 1.98e-7 -c1 1.77e-6 -d 1 \
      -i ~/simple-synthetic.su -nc 15 -tau 0.002 -v 3
```

Note que o caractere '\' é um caractere especial para escapar a quebra de linha e não precisa ser digitado se o comando estiver em apenas uma linha.

O comando `time` mede o tempo de execução da ferramenta e o reporta no terminal como indicado abaixo:

```
real    0m1.753s
user    0m0.928s
sys     0m0.524s
```

Além do tempo reportado pela ferramenta `time`, você observará que o program CMP reporta outras métricas de desempenho a respeito do principal ponto da aplicação, como indicado abaixo:

```
[INFO]: ~/IPDPS-CRS-CMP-code/CMP/OpenCL/src/main.cpp:281:
Total Execution Time: 0.981616:
Giga-Semblances-Trace/s: 7.646577:
Kernel Execution Time: 0.912309:
Kernel Giga-Semblances-Trace/s: 8.227475
```

Podemos comparar essa implementação com a sua versão OpenMP executada na CPU da máquina. O processo de compilação e execução é similar, a diferença é que a execução é feita sem o uso do parâmetro '`-d`', que determina o dispositivo OpenCL.

```
cd ~/IPDPS-CRS-CMP-code/CMP/OpenMP/
mkdir bin
cd bin
cmake ../
make -j
mkdir ../results
cd ../results
time ../bin/cmp-omp2 -aph 600 -c0 1.98e-7 -c1 1.77e-6 \
      -i ~/simple-synthetic.su -nc 15 -tau 0.002 -v 3
```

3.3.3.3. Desligando a máquina virtual da aplicação CMP

Após realizar este experimento e antes de ir para o próximo estudo de caso, desligue a máquina virtual, conforme apresentado na subseção 3.3.2.7.

3.3.4. Estudo de caso 2: Executando o NPB em múltiplas VMs com MPI

Neste estudo de caso, será descrito o processo de criação de uma máquina virtual, sua configuração para execução de um *benchmark* paralelo, a generalização desta imagem e a execução deste tal *benchmark* em um *cluster* na Nuvem Computacional composto por múltiplas VMs. O *benchmark* é o *NAS Parallel Benchmarks* (NPB) [Bailey et al. 1991], composto por um conjunto de programas paralelos. Nosso foco será a utilização de sua versão MPI.

Neste estudo de caso, teremos dois nós: um nó Mestre/Trabalhador e o outro nó somente como Trabalhador. A Figura 3.20 apresenta o diagrama do experimento na AWS e será detalhado nas próximas subseções.

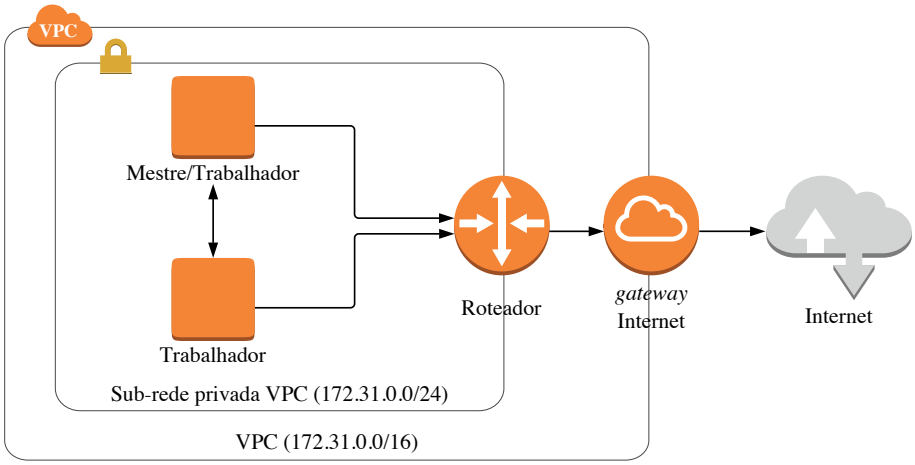


Figura 3.20: Diagrama do experimento na AWS.

3.3.4.1. Configurando uma máquina para geração de imagem

Para a etapa de configuração, uma máquina simples é suficiente. Para este experimento utilizaremos a imagem intitulada "**Ubuntu Server 16.04 LTS**" e o tipo de máquina virtual "t2.micro".

Na próxima etapa, em "*Advanced Details*", na seção "*user data*" pode ser configurado um *script* a ser executado em sua primeira inicialização, onde pacotes necessários podem ser instalados ou atualizados. Selecione a opção "*As text*" e cole as seguintes linhas de comando:

```
#!/bin/bash
sudo apt-get update
sudo apt-get install -y wget make gcc libgfortran3 \
   sysstat libibnetdisc-dev openmpi-bin libopenmpi-dev \
   libhdf5-openmpi-dev gfortran build-essential git
```

Nenhuma outra configuração especial é necessária nesta esta etapa. Dessa forma, avance para a próxima etapa clicando no botão "*Next: Add Storage*" e então avance novamente clicando em "*Next: Add Tags*". Defina os rótulos assim como anteriormente e conclua a inicialização da máquina. Como uma chave de acesso já foi criada na Seção 3.3.2.5, ao lançar a máquina, você pode reutilizar esta mesma chave.

3.3.4.2. Configurando a chave RSA

Após a criação da máquina o acesso pode ser feito por SSH, conforme descrito nas opções do botão conectar. A fim de permitir o acesso por SSH livre entre as máquinas, pode-se criar uma chave de RSA e compartilhá-la entre as demais máquinas. Para tanto, podemos criar a chave utilizando o seguinte comando na máquina virtual criada.

```
ssh-keygen
```

Você não precisa preencher nenhum dos campos solicitados, basta prosseguir pressionando a tecla *Enter*.

Como esta será uma imagem base para todas as máquinas virtuais podemos liberar o acesso dela para ela mesma utilizando o comando:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

3.3.4.3. Transferindo, compilando e executando o NPB

Para testes de aplicações MPI utilizaremos o *benchmark* paralelo NAS. A primeira etapa é a transferência e a descompressão do *benchmark* na máquina virtual. Estas são feitas utilizando as seguintes linhas de comando:

```
wget https://www.nas.nasa.gov/assets/npb/NPB3.3.1.tar.gz
tar -zxf NPB3.3.1.tar.gz
cd NPB3.3.1/NPB3.3-MPI/
```

Os *benchmarks* serão compilados utilizando um arquivo de descrição da suíte localizado no diretório "*config*". Uma cópia do modelo pode ser feita usando o comando:

```
cp config/make.def.template config/make.def
```

As duas únicas alterações que serão feitas neste arquivo são: a) a troca do compilador Fortran para o mpif90 e b) a troca do compilador C para o mpicc. Estas alterações podem ser realizadas com os seguintes comandos:

```
cat config/make.def.template | \
sed "s/MPIF77 = f77/MPIF77 = mpif90/g" | \
sed "s/MPICC = cc/MPICC = mpicc/g" > config/make.def
```

A suíte de aplicativos a ser compilada também é definida em um arquivo de configuração no mesmo diretório. Uma cópia do *template* pode ser feita usando o comando:

```
cp config/suite.def.template config/suite.def
```

Cada linha deste arquivo descreve uma das opções de compilação, combinando tamanhos de execução (A, B, C e D), com os *benchmarks* e a quantidade de processos que serão lançados durante a execução. Altere o arquivo copiado para gerar executáveis para múltiplos tamanhos de entrada de todas as aplicações com 4 processos. Um editor de texto pode ser utilizado (p.ex. `vim` ou `nano`) para editar o arquivo. Alternativamente, você pode executar o *script* seguinte para realizar esta edição:

```
np=4
for bench in bt cg ep ft is lu mg sp; do
    for size in A B C; do
        echo "$bench    $size    $np" >> config/suite.def
    done
done
```

A compilação pode ser feita utilizando-se o comando:

```
make suite
```

Para um teste simples de execução, execute o comando:

```
mpirun -n 1 bin/cg.S.1
```

Um relatório da execução deve ser exibido no terminal, incluindo informações como tempo de execução e checagem de erros.

3.3.4.4. Criando a imagem base

Uma vez com tudo configurado, está máquina será utilizada como base para a criação de futuras máquinas virtuais. Para isso, selecione a máquina virtual e escolha a opção "*Actions*", "*Image*", "*Create Image*", conforme a Figura 3.21.

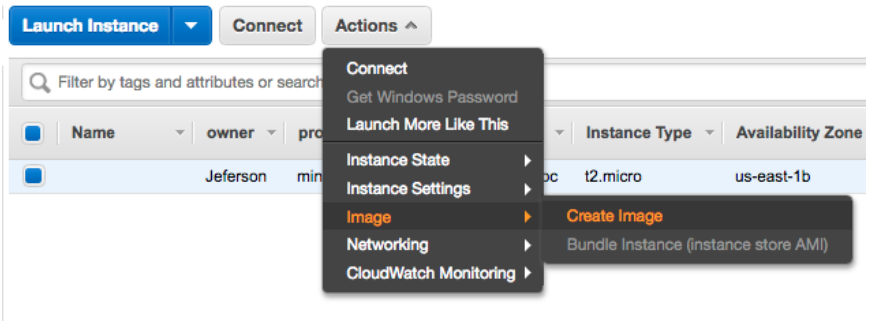


Figura 3.21: Criação de AMI.

Na janela de criação, preencha o campo "*Image name*" com o nome da imagem desejado e o campo "*Image Description*" com uma pequena descrição de sua AMI, conforme apresentado na Figura 3.22. Logo após esta etapa, a instância que estava em uso já pode ser desalocada (*Teminate*).

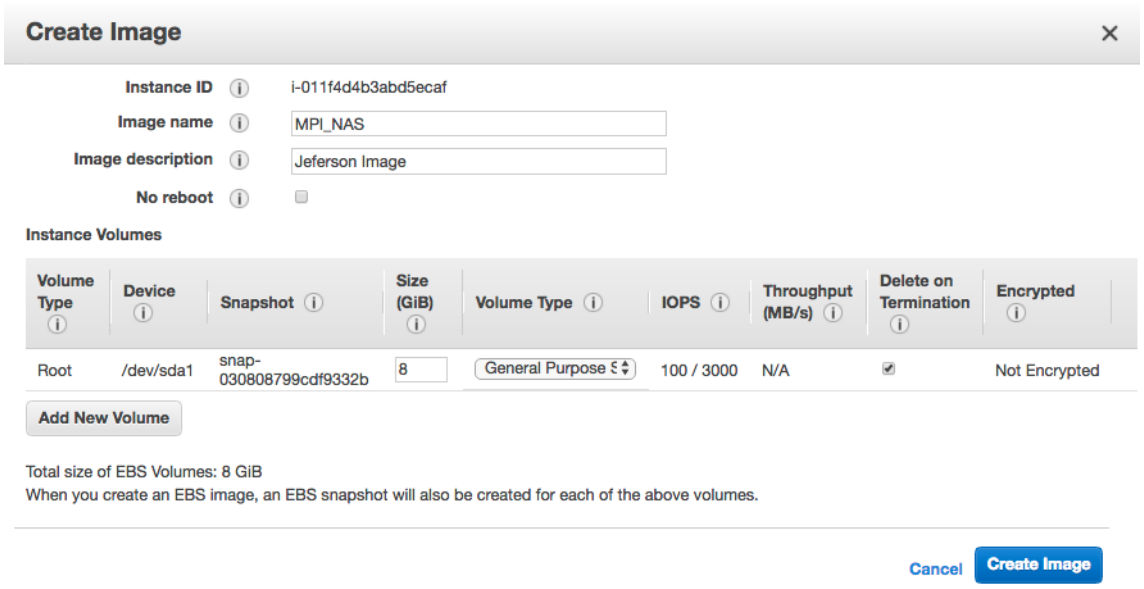


Figura 3.22: Detalhes da criação da AMI.

3.3.4.5. Configurando a rede de interconexão

Para a distribuição da aplicação MPI, todas as máquinas virtuais da rede devem ter livre acesso entre elas. Tal liberação é feita através dos *Security Groups*. A configuração pode

ser feita na parte de "*Security Groups*" do *Dashboard* EC2. Então criamos um novo grupo clicando no botão "*Create Security Group*".

Na criação de um grupo, solicite a liberação apenas da porta 22 a partir de qualquer lugar, conforme ilustrado na Figura 3.23. Se atente à VPC que usará para criar o *Create Security Group*, a mesma VPC deve ser selecionada no momento da criação da máquina.

Figura 3.23: Criação de um *Security Group*.

Com o "*Security Group*" criado, selecione o grupo, no botão "*Actions*" e clique m "*Edite inbound rules*", como mostrado na Figura 3.24.

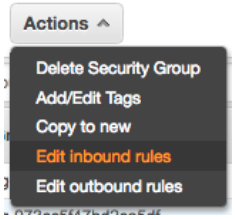


Figura 3.24: Menu de edição das regras do *Security Group*.

Agora você deve criar uma regra para liberar todas as portas entre as máquinas que pertencentes a aquele "*Security Group*". Selecione "*Add Rule*", preencha os campos conforme ilustrado na Figura 3.25. Enquanto você estiver digitando o nome do seu "*Security Group*", aparecerá uma caixa de texto com o identificador, apenas clique na caixa e o campo se preencherá. Em seguida, salve as alterações clicando no botão "*Save*"

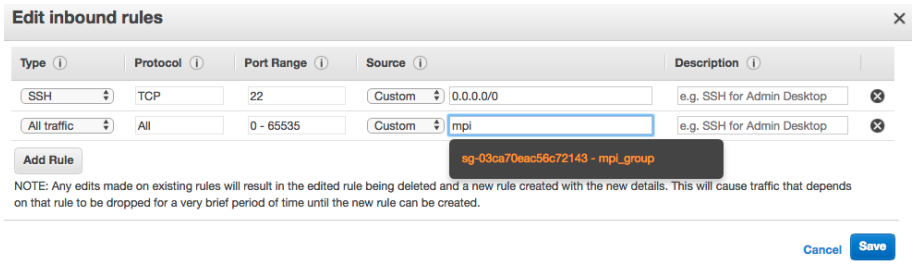


Figura 3.25: Edição de regras do *Security Group*.

3.3.4.6. Selecionando e configurando múltiplas instâncias

Agora, criaremos duas VMs do tipo "M5.large" para executar o *benchmark* NAS de forma paralela. Inicie o processo de criação de uma Máquina Virtual e, para selecionar uma AMI criada por você, escolha no menu lateral a opção "*My AMIs*", conforme a Figura 3.26. Para a criação de um *cluster*(*Cluster Placement Group*), utilizaremos máquinas virtuais otimizadas para Computação, do tipo C5. Os outros tipos de máquinas virtuais que podem ser utilizadas para o *cluster* são:

- Propósito Geral: M4, M5, M5d;
- Otimizada para Computação: C3, C4, C5, C5d, cc2.8xlarge;
- Otimizada para Memória: cr1.8xlarge, R3, R4, R5, R5d, X1, X1e, z1d;
- Otimizada para Armazenamento:: D2, H1, hs1.8xlarge, I2, I3, i3.metal;
- Accelerated computing: F1, G2, G3, P2, P3.

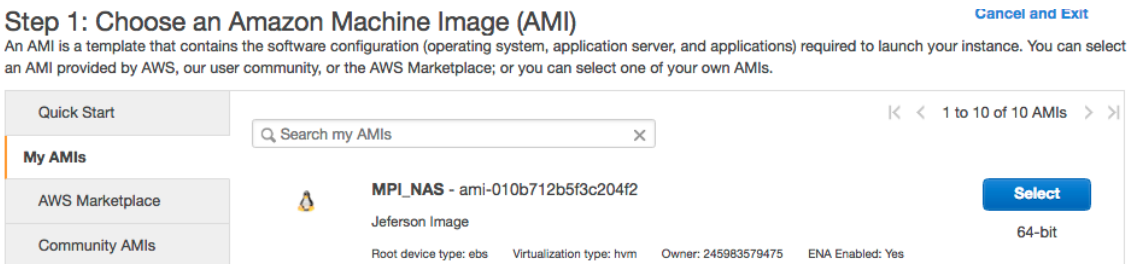


Figura 3.26: Lista de AMIs customizadas.

Na tela de configuração, defina o campo "*Number of instances*" para 2, assim podemos criar as duas máquinas com a mesma configuração de forma simultânea. Marque a caixa "*Add instance to placement group*", para que as máquinas sejam inicializadas no mesmo "*Placement Group*" e possam tirar proveito de uma rede com desempenho melhor. Um grupo pode ser criado selecionando-se a opção "*Add a new placement group*" com o nome definido na caixa de texto abaixo. Assim como apresentado na Figura 3.27.

Ainda nesta etapa, você deve selecionar a mesma VPC de seu "*Security Group*". Então, avance até a etapa de seleção do "*Security Group*" e marque a opção "*Select an existing security group*". Em seguida, selecione o grupo que você criou na Seção 3.3.4.5, como indicado na Figura 3.27.

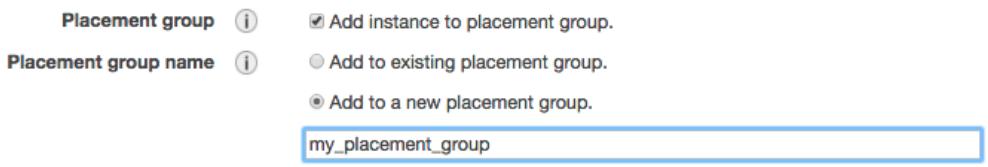


Figura 3.27: Seleção de *Placement Group*.

Após esta etapa, você já pode finalizar a criação das máquinas virtuais, utilizando a sua chave de acesso criada na Seção 3.3.2.5.

Faça o acesso remoto em uma das máquinas, assim como na Seção 3.3.2.6. Como as devidas permissões já foram dadas a chave, basta apenas se conectar a máquina da mesma forma, apenas alterando o endereço. Note que você deve usar o mesmo usuário anterior, `ubuntu`.

Para executar a aplicação nas múltiplas máquinas, primeiro você deve coletar os endereços de IP privados das máquinas. Faça isso selecionando a máquina no *Dashboard* EC2 e no menu inferior (na aba "*Description*" o campo "*Private IPs*" exibe endereços da rede privada). Agora, execute o *benchmark* substituindo os endereços coletados nos campos <IPx>. Note que cada uma das duas máquinas aparece duas vezes na lista, isso ocorre pois cada nó de processamento tem dois núcleos:

```
mpirun -np 4 --host <IP1>,<IP1>,<IP2>,<IP2> \
NPB3.3.1/NPB3.3-MPI/bin/cg.A.4
```

Para não inserir os endereços de IP diretamente na linha de comando, você poderia criar um arquivo para armazena-los, indicando a quantidade de núcleos de processamento de cada nó, através do comando:

```
echo "<IP1> slots=2" > hostfile
echo "<IP2> slots=2" >> hostfile
```

Assim, é possível executar a aplicação, indicando o arquivo de endereços recém criado, usando-se o comando:

```
mpirun -np 4 --hostfile hostfile NPB3.3.1/NPB3.3-MPI/bin/cg.A.4
```

3.3.4.7. Criando o sistema de arquivo compartilhado com o EFS

Quando trabalhamos com sistemas distribuídos, muitas vezes precisamos compartilhar arquivos entre os nós de processamento. O *Elastic File System* (EFS) é um serviço de sistema de de arquivos pela rede fornecido pela AWS. Este serviço possui a vantagem da elasticidade, não sendo necessário dimensionar o tamanho total a ser utilizado.

Para criar um EFS, no menu superior esquerdo, selecione "*Services*" e busque pelo serviço EFS, conforme ilustrado na Figura 3.28.

Em seguida, selecione a opção "*Create File System*", conforme mostrado na Figura 3.29

Selecione a VPC em que você criou o grupo. No destino de montagem, modifique o "*Security Group*" para utilizar o que foi criado anteriormente, no nosso caso "*mpi_group*", e então avance clicando no botão "*Next Step*". Você pode deixar o EFS disponível apenas na mesma zona em que suas máquinas estão executando. Continue a

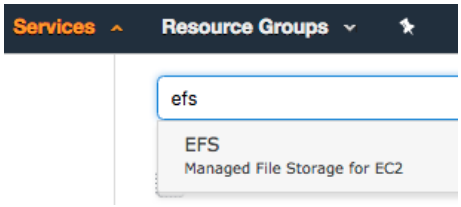


Figura 3.28: Seleção do serviço EFS.

Figura 3.29: Criação de um *Elastic File System*.

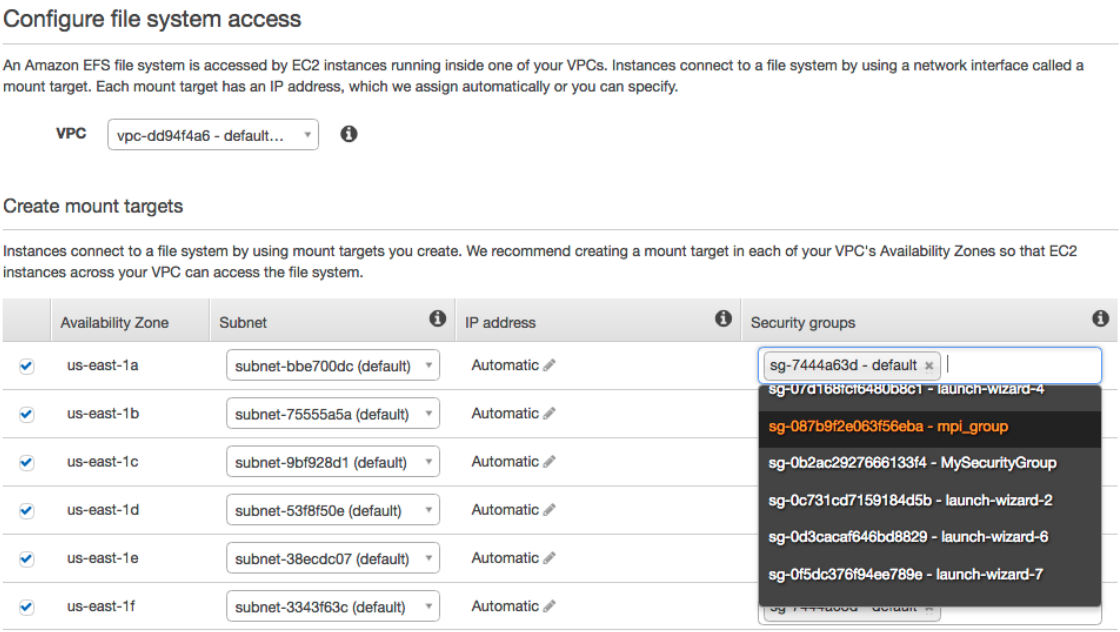


Figura 3.30: Configuração do *Elastic File System*.

criação do sistema de arquivos adicionando etiquetas como na Seção 3.3.2.4 e avance até concluir a criação do EFS.

Depois de analisar e criar o EFS, certifique-se de que o estado do ciclo de vida do destino de montagem esteja disponível, assim como na Figura 3.31.

Após a verificação, siga as Instruções de montagem no Amazon EC2 clicando em "*Amazon EC2 mount instructions*", conforme apresentado na Figura 3.32.

Os comandos utilizados para instalação dos pacotes e montagem foram:

VPC	Availability Zone	Subnet	IP address	Mount target ID	Network interface ID	Security groups	Life cycle state
	us-east-1a	subnet-bbe700dc (default)	172.31.9.104	fsmt-53ae9f1b	eni-0bbf0068fbdc1b2f8	sg-087b9f2e063f56eba - mpi_group	Available

Figura 3.31: Ciclo de vida do EFS.

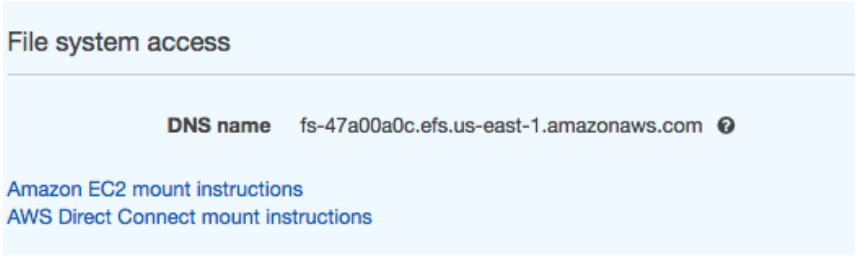


Figura 3.32: Instruções de montagem do EFS.

```
sudo apt-get install nfs-common -y
sudo mkdir ~/efs
sudo mount -t nfs4 -o nfsvers=4.1,rsize=1048576,\
wsize=1048576,hard,timeo=600,retrans=2,noresvport\
fs-aebf63d7.efs.us-east-2.amazonaws.com:/ ~/efs
```

Agora você pode montar este mesmo EFS em suas duas máquinas e observar que que o sistema compartilhado funcionando. Isso pode ser feito criando-se uma conexão para uma máquina e executando-se o comando

```
sudo sh -c "echo 'Ola mundo' > ~/efs/meu_arquivo.txt"
```

Agora, conecte na segunda máquina e, depois de montar o EFS criado, veja o conteúdo do seu arquivo com o seguinte comando:

```
cat ~/efs/meu_arquivo.txt
```

3.3.4.8. Desligando as máquinas virtuais com MPI

Após realizar este experimento, desligue as máquinas virtuais, conforme apresentado na subseção 3.3.2.7.

3.4. Considerações finais

Este minicurso apresentou uma introdução aos conceitos de computação de alto desempenho, uma visão geral dos serviços oferecidos na Nuvem Computacional e como estes serviços podem ser utilizados para realizar computação de alto desempenho. Além disso, o minicurso apresentou dois casos de uso no provedor de serviços Amazon Web Services (AWS): o primeiro mostrando a execução de uma aplicação de alto desempenho em uma máquina virtual com GPU e o segundo executando uma aplicação paralela com MPI em um *cluster* formado por um grupo de máquinas virtuais.

Referências

[Mag 2011] (2011). The Magellan Report on Cloud Computing for Science. Technical report, U.S. Department of Energy Office of Science Office of Advanced Scientific Computing Research (ASCR).

[AWS 2018] AWS, A. W. S. (2018). Enhanced networking on linux. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html. Acessado em 29 agosto de 2018.

[Azure 2018] Azure, M. (2018). Azure - cloud services pricing. https://azure.microsoft.com/en-us/pricing/details/cloud-services/. Acessado em 29 agosto de 2018.

[Bailey et al. 1991] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., et al. (1991). The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73.

[Barr 2016] Barr, J. (2016). Elastic network adapter – high performance network interface for amazon ec2. https://aws.amazon.com/pt/blogs/aws/elastic-network-adapter-high-performance-network-interface-for-amazon-ec2/. Acessado em 29 agosto de 2018.

[Du et al. 2012] Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., and Dongarra, J. (2012). From cuda to opencl: Towards a performance-portable solution for multi-platform gpu programming. *Parallel Computing*, 38(8):391–407.

[Gimenes et al. 2018] Gimenes, T. L., Pisani, F., and Borin, E. (2018). Evaluating the performance and cost of accelerating seismic processing with cuda, opencl, openacc, and openmp. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 399–408. IEEE.

[Gupta et al. 2013] Gupta, A., Kale, L. V., Gioachin, F., March, V., Suen, C. H., Lee, B.-S., Faraboschi, P., Kaufmann, R., and Milojicic, D. (2013). The who, what, why, and how of high performance computing in the cloud. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, pages 306–314. IEEE.

[He et al. 2010] He, Q., Zhou, S., Kobler, B., Duffy, D., and McGlynn, T. (2010). Case study for running hpc applications in public clouds. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 395–401. ACM.

[Karmarkar 2015] Karmarkar, T. (2015). Availability of linux rdma on microsoft azure. https://azure.microsoft.com/en-us/blog/azure-linux-rdma-hpc-available/. Acessado em 29 agosto de 2018.

[Luebke et al. 2004] Luebke, D., Harris, M., Krüger, J., Purcell, T., Govindaraju, N., Buck, I., Woolley, C., and Lefohn, A. (2004). Gpgpu: General purpose computation

on graphics hardware. In *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, New York, NY, USA. ACM.

[Mell and Grance 2011] Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. Technical report, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg.

[Owens et al. 2007] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J. (2007). A survey of general-purpose computation on graphics hardware. 26(1):80–113.

[Puthal et al. 2015] Puthal, D., Sahoo, B., Mishra, S., and Swain, S. (2015). Cloud Computing Features, Issues, and Challenges: A Big Picture. In *Computational Intelligence and Networks (CINE), 2015 International Conference on*, pages 116–123.

[Valiant 1990] Valiant, L. G. (1990). A bridging model for parallel computation. *Communications of the ACM*, 33.