Avaliação do impacto da comunicação intra e entre-nós em nuvens computacionais para aplicações de alto desempenho

Thiago Kenji Okada

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Pós-Graduação em Ciência da Computação
Orientador: Prof. Dr. Alfredo Goldman

Durante o desenvolvimento deste trabalho o autor recebeu auxílio financeiro da CAPES

São Paulo, Setembro de 2016

Avaliação do impacto da comunicação intra e entre-nós em nuvens computacionais para aplicações de alto desempenho

Esta versão da dissertação contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 07/11/2016. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Prof. Dr. Alfredo Goldman (orientador) - IME-USP
- Prof. Dr. Marco Aurélio Stelmar Neto - IME-USP
- Prof. Dra. Lúcia Maria de Assumpção Drummond - IBM

Agradecimentos

Gostaria primeiramente de agradecer minha mãe, Eliana Mayumi Hieda, e meu pai, Mauro Yukio Okada, pelo apoio durante meu Mestrado. Também gostaria de agradecer ao meu irmão, Felipe Okada, e todos os meus familiares por parte de mãe e pai.

Quero agradecer aos meus (melhores) amigos, Bruno Tinen, Eduardo Nito, Guilherme Yudy e Musashi Obara pela grande amizade e apoio.

Agradeço também ao Prof. Dr. Alfredo Goldman da Universidade de São Paulo (USP) pela orientação durante todo o meu Mestrado. E agradeço ao pessoal do Laboratório de Sistemas do IME-USP, em especial ao Nelson Lago, que me ajudou em diversos experimentos com a nuvem instalada na *revoada*, e aos meus colegas Marcos Amaris Gonzales, Pedro Bruel e Rogério Gonçalves, que me ajudaram em diversos experimentos. Além deles, os vários colegas de IME que conheci durante o Mestrado, muitos deles membros do Centro de Competência de Software Livre (CCSL), incluindo Albert Vonpupp, Hugo Braga, Diego Araújo, Diogo Pina, Elaine Naomi, Fellipe Souto, Jackson Souza, Pedro Scocco, Rafael Manzo.

Agradecimentos ao Prof. Dr. Gerson Geraldo H. Cavalheiro da Universidade Federal de Pelotas (UFPel) pelo apoio e orientação durante a minha estadia na cidade de Pelotas. Também agradeço a todo o pessoal do *Laboratory of Ubiquitous and Parallel Systems* (LUPS) da UFPel, incluindo o Rodrigo Duarte, Lucas Xavier, Vitor Ataides, Lucas Agostini, Fernando Angelin, Maicon Santos, Vinicius Santos, Juan Rios (Asaki), Giovane Torres, Leonardo Mendonça, Tainã Carvalho, João Oliveira, Michael Costa.

Finalmente, agradecimentos ao pessoal da UNESP de São José do Rio Preto: Prof. Aleardo Manacero e Profa. Renata Spolon Lobato, pela orientação durante a graduação, e os meus colegas de laboratório: Danilo Costa, Rafael Stabile, Leandro Barbosa, Gabriel Saraiva, Cássio Forte, entre outros.

Resumo

OKADA, T. **Avaliação do impacto da comunicação intra e entre-nós em nuvens computacionais para aplicações de alto desempenho** . 2016. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

Com o advento da computação em nuvem, não é mais necessário ao usuário investir grandes quantidades de recursos financeiros em equipamentos computacionais. Ao invés disto, é possível adquirir recursos de processamento, armazenamento ou mesmo sistemas completos por demanda, usando um dos diversos serviços disponibilizados por provedores de nuvem como a Amazon, o Google, a Microsoft, e a própria USP.

Isso permite um controle maior dos gastos operacionais, reduzindo custos em diversos casos. Por exemplo, usuários de computação de alto desempenho podem se beneficiar desse modelo usando um grande número de recursos durante curtos períodos de tempo, ao invés de adquirir um aglomerado computacional de alto custo inicial.

Nosso trabalho analisa a viabilidade de execução de aplicações de alto desempenho, comparando o desempenho de aplicações de alto desempenho em infraestruturas com comportamento conhecido com a nuvem pública oferecida pelo Google. Em especial, focamos em diferentes configurações de paralelismo com comunicação interna entre processos no mesmo nó, chamado de intra-nós, e comunicação externa entre processos em diferentes nós, chamado de entre-nós.

Nosso caso de estudo para esse trabalho foi o NAS Parallel Benchmarks, um benchmark bastante popular para a análise de desempenho de sistemas paralelos e de alto desempenho. Utilizamos aplicações com implementações puramente MPI (para as comunicações intra e entre-nós) e implementações mistas onde as comunicações internas foram feitas utilizando OpenMP (comunicação intra-nós) e as comunicações externas foram feitas usando o MPI (comunicação entre-nós).

Palavras-chave: nuvens computacionais, computação de alto desempenho, benchmark.

Abstract

OKADA, T.. **Evaluation of impact from inter and intra-node communication in cloud computing for HPC applications** . 2016Dissertação(Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2016.

With the advent of cloud computing, it is no longer necessary to invest large amounts of money on computing resources. Instead, it is possible to obtain processing or storage resources, and even complete systems, on demand, using one of the several available services from cloud providers like Amazon, Google, Microsoft, and USP.

Cloud computing allows greater control of operating expenses, reducing costs in many cases. For example, high-performance computing users can benefit from this model using a large number of resources for short periods of time, instead of acquiring a computer cluster with high initial cost.

Our study examines the feasibility of running high-performance applications, comparing the performance of high-performance applications in a known infrastructure compared to the public cloud offering from Google. In particular, we focus on various parallel configurations with internal communication between processes on the same node, called intra-node, and external communication between processes on different nodes, called inter-nodes.

Our case study for this work was the NAS Parallel Benchmarks, a popular benchmark for performance analysis of parallel systems and high performance computing. We tested applications with MPI-only implementations (for intra and inter-node communications) and mixed implementations where internal communications were made using OpenMP (intra-node communications) and external communications were made using the MPI (inter-node communications).

Keywords: cloud computing, high performance computing, benchmark.

Sumário

Glossário

API *Application Programming Interface*

GCE *Google Compute Engine*

HPC Computação de Alto Desempenho, ou *High Performance Computing*

IaaS *Infrastructure-as-a-Service*

IPC *Inter-Process Communication*

MPI *Message Passing Interface*

MV Máquinas Virtuais

MZ *Multi-Zone*

NIST *National Institute of Standards and Technology*

NPB *NAS Parallel Benchmarks*

OpenMP *Open Multi-Processing*

SO Sistema Operacional

SZ *Single-Zone*

TI Tecnologia da Informação

vCPU CPU virtual

vRAM RAM virtual

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o advento da computação em nuvem, não é mais necessário ao usuário investir grandes quantidades de recursos financeiros em equipamentos computacionais. Ao invés disto, é possível adquirir recursos de processamento e/ou armazenamento por demanda, usando um dos diversos serviços disponíveis por provedores de nuvem públicas como a Amazon EC2[1], o Google Compute Engine[2] e o Windows Azure[3]. Em grandes empresas ou em instituições de pesquisa, também podem ser utilizadas nuvens privadas para satisfazer diferentes demandas internas, com melhor aproveitamento dos recursos. Este é o caso da USP[4].

Um dos modelos de serviço provido pela nuvem é a de Infraestrutura como Serviço (*Infrastructure-as-a-Service* (IaaS)), que dá acesso aos usuários da nuvem a Máquinas Virtuais (MV) com certos recursos pré-determinados. O usuário da nuvem não possui conhecimento do hardware em que o serviço é executado, mas tem controle total sobre o software que será rodado sobre a MV, incluindo o Sistema Operacional (SO).

O usuário da nuvem pode, por exemplo, selecionar o número de processadores, quantidade de espaço livre em disco e memória RAM, largura de banda e uma das várias imagens de sistemas disponíveis. Uma facilidade administrativa adicional é que estas imagens de sistema podem já estar pré-configuradas, incluindo configurações mais comuns como Ubuntu Linux com servidor Apache, banco de dados MySQL e as linguagens PHP/Python já instalados, para servir páginas Web, ou um sistema base que permita a instalação dos programas que o usuário necessita.

Graças as vantagens citadas acima, tanto a comunidade de Tecnologia da Informação (TI) quanto a comunidade acadêmica passaram a enxergar os vários benefícios que a computação em nuvem trouxe, como garantia de disponibilidade, flexibilidade de recursos, eliminação dos custo relacionados a configuração do hardware e manutenção (AFG+10; MG11). Porém, problemas como baixo desempenho da rede, variação de desempenho (em especial durante horários de pico) e o *overhead* causado pela virtualização são os novos desafios trazidos na execução de aplicações com alta demanda computacional neste tipo de plataforma (WB10; CBT+14; NMNAJ12).

O foco desse trabalho são os usuários de Computação de Alto Desempenho, ou *High Performance Computing* (HPC), que podem por exemplo requisitar uma grande quantidade de recursos computacionais por um curto período de tempo. Isso possivelmente reduz os gastos que seriam usados no investimento de um aglomerado computacional, que tem alto custo inicial e poderia ficar ocioso a maior parte do tempo, por exemplo. Além disso, plataformas

[1] https://aws.amazon.com/pt/ec2/
[2] https://cloud.google.com/compute/
[3] https://azure.microsoft.com/pt-br/
[4] https://wiki.uspdigital.usp.br/nuvem/Paginas/Inicial.aspx

de computação em nuvem são uma alternativa viável para computação de alto desempenho. Por exemplo, a Amazon EC2 está na posição 240 da lista de Junho de 2016 do TOP500 [5], que é a lista dos 500 computadores mais poderosos do mundo.

1.1 Considerações preliminares

Computação em nuvem se utiliza da tecnologia de virtualização para permitir a execução de diversas máquinas virtuais em um mesmo servidor físico. Graças a isso, é possível aproveitar melhor os recursos computacionais de um data center típico. A alocação de diversos processos em uma mesma máquina física, por exemplo, permite que processos que precisam de muito poder de processamento executem de forma concorrente com processos que usem muito entrada e saída de dados. Além disso, é possível realocar processos de acordo com a demanda, de forma dinâmica, utilizando-se dos recursos que a virtualização oferece, como por exemplo migração de MVs em tempo real (Red15).

É possível escolher dentre diferentes objetivos ao gerenciar essas MVs. Por exemplo, pode-se minimizar o consumo de energia ao colocar diversas MVs em um mesmo servidor físico, porém se todas as MVs precisarem de muito poder de processamento ou entrada e saída, isso pode prejudicar o desempenho. Por outro lado, é possível instanciar apenas uma MV por servidor físico e garantir desempenho máximo, porém com possível desperdício de recursos. Isso acontece pois dificilmente uma aplicação consumirá 100% dos recursos da máquina durante longos períodos de tempo, e o tempo ocioso da máquina física poderia ser desperdiçado.

Outra questão a ser tratada na computação em nuvem trás é a variação do desempenho de acordo com os recursos alocados. Por exemplo, a alocação de múltiplas MVs na mesma máquina física possivelmente reduz a latência de comunicação, pois não é necessário utilizar a rede física para a troca de mensagens entre máquinas virtuais. Porém a utilização de máquinas virtuais pode causar variações no desempenho tanto de processamento como armazenamento. Isso porque múltiplos usuários podem compartilhar a mesma máquina física, que pode ser sobrecarregada caso exista muita demanda por parte de seus usuários. Infelizmente os usuários, por não terem acesso ao hardware, não possuem controle sobre isso, podendo tornar a computação de alto desempenho, nesse tipo de plataforma, um desafio (OIY+10). Assim, mesmo no caso em que a diminuição do custo é obtido pela alocação de MVs na mesma máquina física, o desempenho efetivo pode ser penalizado por outros fatores.

Ao usuário da nuvem não são fornecidas ferramentas que permitam o controle de todos os detalhes citados de maneira fina, pois caso contrário, estaríamos nos afastando do conceito inicialmente proposto para a computação em nuvem, que é de prover computação como um serviço, onde detalhes de baixo nível ficam por conta do provedor.

Portanto, executar aplicações de alto desempenho em nuvens computacionais se torna um desafio graças aos motivos mencionados acima. Desenvolver aplicações paralelas que executem de forma eficiente consiste, por si, uma tarefa bastante árdua. Prever o desempenho dessas aplicações em um ambiente de nuvem se torna um problema ainda maior.

Nosso trabalho estuda então como as características de um ambiente de nuvem computacional real, incluindo tanto uma nuvem privada quanto pública, se comportam com aplicações de alto desempenho. Temos como objetivo conseguir aproveitar melhor os recursos disponíveis nessa arquitetura, considerando suas características próprias.

[5] https://www.top500.org/system/178321

1.2 Objetivos

Este trabalho tem como objetivo principal o estudo de como as nuvens computacionais funcionam, quais as implicações desse modelo e como podemos utilizar as nuvens para a execução de aplicações de alto desempenho nas mesmas.

Graças as características únicas de um ambiente de nuvem, que não permite controle fino dos recursos computacionais utilizados, iremos estudar mais a fundo o impacto da execução de aplicações de alto desempenho na nuvem a partir da execução de um dos benchmarks mais conhecidos na comunidade científica, o *NAS Parallel Benchmarks* (NPB). A partir do conhecimento de seu padrão de comunicação, podemos entender o que está acontecendo em baixo nível durante sua execução.

O objetivo final será permitir a avaliação de uma determinada infraestrutura de nuvem e, a partir dessa avaliação, permitir a execução de aplicações de alto desempenho com maior eficiência. O beneficiário final deste estudo é o usuário de nuvens, que terá subsídios para efetuar o contrato de recursos para a sua demanda.

1.3 Contribuições

As principais contribuições deste trabalho são as seguintes:

1. Relacionar o desempenho de execução de aplicações de alto desempenho em infraestruturas com comportamento conhecido com o obtido em nuvens.

2. Analisar o impacto de diferentes proporções de paralelismo com comunicação intra-nós e entre-nós.

1.4 Organização do trabalho

No Capítulo 2 veremos os conceitos básicos necessários para o desenvolvimento desse trabalho, como computação em nuvem, computação de alto desempenho e benchmarks. No Capítulo 3 faremos uma revisão da literatura disponível sobre o tema. No Capítulo 4, veremos como podemos executar aplicações de alto desempenho em um ambiente de nuvem computacional, assim como as devidas implicações dessa execução. No Capítulo 5, veremos o *NAS Parallel Benchmarks* com detalhes, os benchmarks escolhidos para esse trabalho. No Capítulo 6, veremos os experimentos realizados e análise dos dados. No Capítulo 7, veremos as conclusões tiradas do trabalho e trabalhos futuros.

Capítulo 2

Conceitos

Neste capítulo apresentaremos os conceitos básicos que serão usados no trabalho. Começaremos fazendo uma revisão sobre o que é a computação em nuvem, incluindo seus modelos de serviço e implementação, além de uma breve definição de virtualização, tecnologia essencial para a computação em nuvem. Depois iremos revisar o conceito do que é a computação de alto desempenho e também o conceito de paralelismo, já que é o paralelismo, no contexto deste trabalho, que permite a computação de alto desempenho. Finalmente apresentaremos uma definição do que é benchmark, incluindo algumas características básicas esperadas em qualquer benchmark.

2.1 Computação em nuvem

Uma definição de computação dada pelo *National Institute of Standards and Technology* (NIST) é: "[. . .] um modelo que permite acesso via rede ubíquo, conveniente e sob demanda para um conjunto compartilhado de recursos computacionais (como redes, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente provisionados e liberados com mínimo esforço de gerenciamento ou interação com o provedor de serviço" (MG11):. Ou seja, a computação em nuvem permite seus usuários requisitar sistemas inteiros tanto de software como hardware rapidamente e por demanda, permitindo o uso de recursos computacionais como um serviço.

O NIST também define algumas características essenciais de uma nuvem computacional (MG11):

- **Autosserviço sob demanda:** um consumidor consegue sozinho provisionar recursos de uma nuvem, unilateralmente, sem necessidade de interação humana do provedor.
- **Amplo acesso a rede:** os recursos estão disponíveis para acesso através da rede e podem ser acessados por meio de mecanismos padronizados que promovem o uso de plataformas heterogêneas pelos clientes, como celulares, tablets, notebooks e estações de trabalho.
- **Agrupamento de recursos:** os recursos do provedor são agrupados para servir múltiplos consumidores usando um modelo multi-inquilinos, onde múltiplos consumidores possuem acesso ao mesmo recursos físicos porém isolados logicamente. Os recursos são realocados dinamicamente de acordo com a demanda do consumidor. O consumidor geralmente não possui controle ou conhecimento sobre a exata localização dos recursos, porém ele pode especificar uma localização de nível mais alto de abstração (exemplo: país, estado ou data center).

- **Rápida elasticidade:** os recursos podem ser provisionados e removidos de forma elástica, e em alguns casos, automaticamente, com o objetivo de escalar o sistema dinamicamente conforme a demanda. Para o consumidor, os recursos são aparentemente ilimitados e podem ser apropriado em qualquer quantidade a qualquer momento.

- **Serviço mensurado:** sistemas em nuvem automaticamente controlam e otimizam o uso de recursos, com um nível de abstração apropriado dependendo do serviço utilizado (exemplo: armazenamento, processamento, banda e contas de acesso). Recursos podem ser monitorados, controlados e reportados, oferecendo transparência tanto para o provedor quanto para o cliente do serviço.

No nosso trabalho, cada uma dessas características possui uma importância diferente. *Serviço mensurado* é um dos mais visíveis e importantes: com uma execução otimizada de aplicações de alto desempenho podemos reduzir o tempo total de execução de uma aplicação na nuvem. Como na nuvem os serviços são mensurados, essa redução do tempo total de execução pode reduzir os gastos totais com experimento, permitindo pesquisadores com orçamento limitado realizarem mais experimentos ou utilizarem o dinheiro economizado para outros gastos da pesquisa. Outra característica importante é a *rápida elasticidade*, permitindo o rápido provisionamento de recursos de acordo com as demandas das aplicações. *Autosserviço sob demanda* permite aos usuários de computação de alto desempenho que eles tenham acesso autônomo aos recursos, sem dependerem de terceiros. *Amplo acesso a rede* permite o acesso dos recursos providos na nuvem de qualquer lugar, usando diversos dispositivos. Isso pode ser interessante para o pesquisador, que não precisa mais ter acesso direto a um aglomerado computacional para executar experimentos.

Agrupamentos de recursos é uma característica menos visível aos usuários, porém pode afetar o desempenho de aplicações de alto desempenho de forma visível. *Amplo acesso a rede* implica possíveis problemas de banda e latência de rede, o que impacta aplicações de alto desempenho de maneira profunda. *Agrupamentos de recursos* significa que outros usuários usando recursos da mesma máquina física podem afetar a execução de sua aplicação, o que pode ser um problema dependendo do tipo de aplicação.

2.1.1 Modelos de serviço

Na computação em nuvem, o *modelo de serviço* é a divisão das diferentes nuvens de acordo com a abstração da capacidade computacional disponibilizada aos usuários. O NIST define três modalidades de serviço disponíveis em nuvens computacionais (MG11):

- *Software* como Serviço (SaaS, ou *Software as a Service*): o recurso fornecido ao consumidor é o uso de aplicações do fornecedor executando em uma infraestrutura na nuvem computacional. As aplicações são acessíveis por meio de diferentes clientes como um navegador ou interface de programação. O consumidor não gerencia ou controla a infraestrutura subjacente da nuvem, incluindo rede, servidores, sistemas operacionais, armazenamento ou mesmo controle dos recursos fornecidos individualmente pela aplicação, com a possível exceção de configurações específicas do usuário. Exemplos de SaaS: Google Apps (Gmail, YouTube, Google Docs), Microsoft Office 365, Salesforce.

- Plataforma como Serviço (PaaS, ou *Platform as a Service*): o recurso fornecido ao consumidor é a possibilidade de implementar na infraestrutura de nuvem aplicações desenvolvidas pelo próprio usuário ou adquirido por terceiros, usando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo provedor. O consumidor não deve gerenciar ou controlar a infraestrutura subjacente da nuvem, incluindo

rede, servidores, sistemas operacionais, ou armazenamento, mas controla suas aplicações implementadas sobre a infraestrutura e possivelmente das configurações do ambiente da aplicação. Exemplos de PaaS: Windows Azure, Heroku, Google App Engine, AWS Elastic Beanstalk.

- Infraestrutura como Serviço (IaaS, ou *Infrastructure as a Service*): o recurso fornecido ao consumidor é a provisão de processamento, armazenamento, rede e outros recursos computacionais fundamentais onde o consumidor pode implementar e executar *software* arbitrários, incluindo o sistema operacional e aplicações. O consumidor não controla a infraestrutura subjacente da nuvem mas possui o controle sobre os sistemas operacionais, armazenamento e aplicações implementadas; e possivelmente controle limitado sobre os componentes de rede (por exemplo, o firewall). Exemplos de SaaS: AWS EC2, Windows Azure, Rackspace, Google Compute Engine.

Neste trabalho, nosso foco será nas nuvens de tipo IaaS. Uma IaaS permite a flexibilidade necessária para aplicações do tipo HPC, permitindo a instalação de pacotes comumente usados em aplicações de computação de alto desempenho, incluindo Fortran, *Open Multi-Processing* (OpenMP), *Message Passing Interface* (MPI) e TAU.

Os outros modelos de serviço têm uma flexibilidade menor comparado ao modelo IaaS, exigindo a reescrita de aplicações em muitos casos. Porém eles abstraem diversos detalhes de baixo nível do usuário, e podem ser interessantes no futuro, por permitirem usuários leigos (que não tem conhecimento técnico) a utilizarem algoritmos otimizados para computação distribuída e de alto desempenho, e assim resolverem problemas maiores.

2.1.2 Modelos de implementação

Além dos modelos citados anteriormente, temos também o *modelo de implementação*, que são basicamente as diferentes políticas de acesso aos recursos computacionais de uma determinada nuvem. Novamente, de acordo com NIST, temos (MG11):

- *Nuvem privada*: a infraestrutura da nuvem é provisionada para uso exclusivo de uma única organização composta de múltiplos consumidores (como unidades de negócio). A sua propriedade, gerenciamento, e operação pode ser feita pela própria organização, um terceiro, ou uma combinação de ambos, e pode ou não estar dentro das instalações da organização.

- *Nuvem comunitária*: a infraestrutura de nuvem é de uso exclusivo para uma comunidade de consumidores de organizações que compartilham interesses (de missão, requisitos de segurança, políticas, e regulamentação). A sua propriedade, gerenciamento, e operação pode ser feito por uma ou mais organizações da comunidade, um terceiro ou uma combinação de ambos, e pode ou não estar dentro das instalações da organização.

- *Nuvem pública*: a infraestrutura de nuvem é provisionada de forma aberta e disponível ao público geral. A sua propriedade, gerenciamento, e operação pode ser feita por uma empresa, universidade ou pelo governo, ou qualquer combinação dos três. As instalações são localizadas dentro do provedor.

- *Nuvens híbridas*: a infraestrutura de nuvem é composta de duas ou mais distintas infraestruturas de nuvens (privada, comunitária, pública) que continuam sendo independentes, mas são interligadas com uma tecnologia padronizada ou proprietária que permite portabilidade de aplicações e dados (para, por exemplo, transferência de processamento para balanceamento de cargas entre nuvens).

Nossos experimentos são voltado tanto as nuvens privadas como as nuvens públicas. Cada uma têm suas vantagens e desvantagens. Em uma nuvem privada, nós temos o controle de todos os parâmetros usados durante o escalonamento e execução das máquinas virtuais. Graças a isso, podemos saber exatamente onde cada máquina virtual foi escalonada, e controlar seus recursos de maneira mais precisa. Porém, nuvens privadas exigem um investimento inicial muito mais alto, e graças a isso temos limitações a quantidade total de recursos que podemos usar em nuvens privadas. Isso remove a ilusão de "recursos ilimitados" que a nuvem provê. Já na nuvem pública, temos de fato a ilusão de "recursos ilimitados", ainda mais na utilização de grandes nuvens públicas como o Google Compute Engine. Porém não é possível ter acesso a todos os parâmetros usados durante a execução de um programa. Por exemplo, não é possível obter a informação de que máquina física uma determinada máquina virtual foi instanciada, ou mesmo se várias de nossas máquinas virtuais estão sobre um mesmo nó físico, diferentes nós físicos porém fisicamente próximos, ou nós diferentes nós físicos distantes. Isso acaba limitando a análise dos resultados.

2.1.3 Virtualização

Virtualização pode ser definida como "um arcabouço ou metodologia de divisão de recursos computacionais em múltiplos ambientes, por aplicação de um ou mais conceitos de tecnologias como particionamento de hardware e software, tempo compartilhado, simulação parcial ou total de máquina, emulação, qualidade de serviço, entre outros" (Sin04).

Virtualização é uma das tecnologias que permitem a computação em nuvem a existir. Por exemplo, sem a virtualização não seria possível termos o compartilhamento dos recursos físicos de forma logicamente isolada por múltiplos usuários. Para alguns autores, a nuvem inclusive é definida em termos da virtualização (VRMCL08).

Técnicas de virtualização eficientes, como a para-virtualização, são importantes, ou caso contrário, as aplicações teriam uma redução grande do desempenho em comparação com máquinas físicas. Um conceito semelhante ao da virtualização são os contêineres, que oferecem algumas das vantagens da virtualização (como isolamento de usuários compartilhando os mesmos recursos físicos) sem grande parte do *overhead* que a virtualização impõe.

2.2 Computação de alto desempenho e paralelismo

Computação de alto desempenho pode ser definida como "uma tecnologia que é usada para prover soluções para problemas que exigem poder computacional significativo e precisam acessar, ou processar, uma grande quantidade de dados rapidamente, ou precisam operar interativamente através um rede distribuída geograficamente" (FK99). Graças ao foco da computação paralela em problemas grandes e com grandes quantidades de dados, a computação de alto desempenho sempre teve uma grande ligação com a computação paralela e distribuída. Podemos ver o porque disso na Figura 2.1:

Na Figura 2.1, vemos que apesar do número de transistores continuar crescendo de acordo com a Lei de Moore, a frequência de relógio dos processadores e o desempenho em um único núcleo estabilizou. Mais interessante, temos um crescimento no número de núcleos físicos por processador, que reflete a tendência de termos processadores com múltiplos núcleos desde meados da década de 90. Isso mostra a importância do paralelismo e sistemas distribuídos: temos sistemas com um número cada vez maior de processadores, e uma diminuição no incremento de frequência de relógio. Então precisamos explorar o paralelismo para continuar aumentando o desempenho. E mesmo considerando que o número de transistores continua a

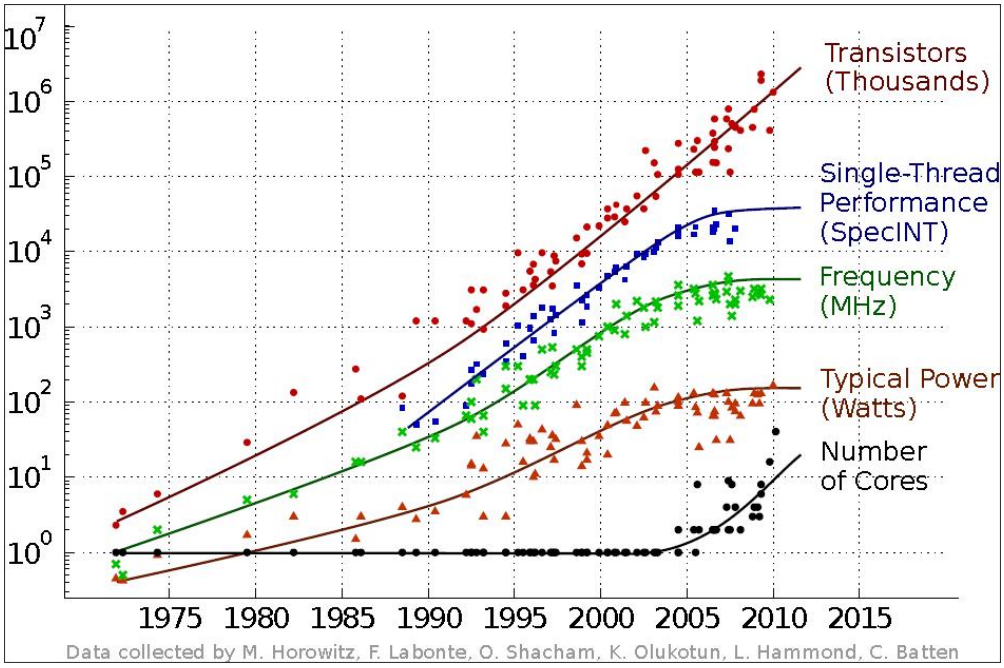


Figura 2.1: *35 anos de tendências de microprocessadores (Moo11).*

crescer, a demanda de processamento cresce de maneira mais rápida, por isso a necessidade de sistemas cada vez mais distribuídos.

A queda de frequência de relógio não é uma surpresa. Enquanto o aumento da frequência do relógio é uma das maneiras mais simples de aumentar o desempenho geral de processamento existem diversos problemas físicos que dificultam esse aumento (Pad11):

- Densidade de transistores em um circuito integrado limita a frequência do relógio;
- A velocidade da luz limita o tempo de transmissão ou latência;
- Custo de fabricação do hardware limitam a complexidade que pode ser incluído em um sistema.

Para evitar os limites impostos pelo primeiro e terceiro pontos, paralelismo pode ser usado (Pad11). Esse paralelismo pode ser realizado multiplicando o número de núcleos em um único processador (como visto na Figura 2.1), usando vários processadores em um único sistema ou ainda conectando múltiplos computadores (cada um com seus vários processadores) em uma rede. No último caso temos um grande aumento no tempo de latência (o segundo ponto), porém ao mesmo tempo essa organização permite um grande aumento na densidade de processadores, além de outros recursos como memória RAM e espaço de armazenamento. Essa conexão de múltiplos computadores multiprocessados em rede é o que permitiu a computação de alto desempenho atual.

Por causa do problemas acima, durante mais de 60 anos, computadores voltados a computação de alto desempenho eram baseados em dois tipos de arquiteturas: supercomputadores vetoriais e aglomerados computacionais com um ou vários processadores escalares. Na grande maioria dos casos eram sistemas proprietários executando softwares proprietários. Porém na década de 2000 tivemos a transição dessas arquiteturas proprietárias para aglomerados computacionais do tipo *Beowulf*, que utilizam hardware e software comuns (*commodity*) e a filosofia do *faça você mesmo*, reduzindo os custos e tornando a computação de alto desempenho mais acessível (ST98).

Vendo a lista do TOP500[1], os 500 supercomputadores mais rápidos do mundo, temos grandes sistemas multiprocessados com milhares ou mesmo milhões de processadores, que consumindo milhares de kW de energia. Por exemplo, o atual sistema número 1 do TOP500, Sunway TaihuLight[2], possui 10.649.600 núcleos e consome 15.371 kW de energia. Esse grande número de núcleos e o grande consumo de energia reflete a dificuldades citadas anteriormente no desenvolvimento de novos sistemas.

Considerando computação num ambiente de nuvem, ao mesmo tempo que a nuvem oferece a *ilusão de recursos ilimitados*, ela não permite o controle dos detalhes de baixo nível de hardware (por exemplo, o modelo de CPU ou mesmo sua frequência de relógio), apenas detalhes de alto nível (número de núcleos, quantidade de RAM, etc.). Ou seja, para executar aplicações de maneira rápida e eficiente num ambiente de nuvem (e eficiência, nesse caso, podemos pensar numa redução no total gasto para execução) temos que se utilizar de aplicações paralelas.

2.2.1 Tipos de aplicações paralelas

Existem diversos aplicações paralelas. Algumas, por relações de dependência com outras aplicações, por serem aplicações de legado, ou pelo fato de serem aplicações intrinsecamente sequenciais (por exemplo, leitura e escrita no disco), o número de processadores a ser utilizados na aplicação é fixo e o escalonador não tem liberdade de alterar seu número de processadores. Outras aplicações são mais flexíveis, permitindo alterar o número de processadores executando a tarefa antes ou durante a execução do programa. Podemos definir três tipos de aplicações paralelas (Dro09):

- **Aplicações rígidas**: o número de processadores usados pela aplicação é fixo (não é possível alterar).
- **Aplicações paralelas moldáveis**: o número de processadores usados pela aplicação é escolhido antes da execução da aplicação, e não é alterado até o término da mesma.
- **Aplicações paralelas maleáveis**: o número de processadores usados pela aplicação é escolhido pelo escalonador, e pode ser mudado durante o tempo de execução. A aplicação pode adicionar mais processadores caso eles apareçam, e pode liberar processadores de acordo com a demanda.

Nossos benchmarks de uma forma geral são moldáveis, permitindo a escolha do número de processadores apenas antes da execução. Porém isso é suficiente para analisar como a aplicação se comporta em diferentes configurações de processadores, permitindo uma análise de escalabilidade com diferentes configurações de comunicação intra-nós e entre-nós.

2.2.2 Modelos de programação paralela

Para facilitar o desenvolvimento de aplicações paralelas, foram propostos diversos modelos de programação. Esses modelos existem para facilitar o desenvolvimento, criando uma nível de abstração entre a implementação e a arquitetura, além de tentar dar uma previsão de desempenho. Porém, não existe um modelo universal para programação paralela (ST98).

Modelos de programação paralela são classificados de acordo com a sua capacidade de abstração. Basicamente, quanto mais o modelo abstrair detalhes de baixo nível, como comunicação e paralelismo, menos o programador terá que se preocupar com esses detalhes e

[1] https://www.top500.org/
[2] https://www.top500.org/lists/2016/06/

consequentemente o desenvolvimento da aplicação paralela se torna mais simples. Em compensação, cada abstração feita em um modelo de programação impede o programador de realizar determinadas otimizações. Além disso, quanto mais abstrato o modelo, mais difícil é implementá-lo de maneira eficiente em máquinas reais (ST98).

Alguns dos modelos de programação paralela mais conhecidos são o PRAM, CGM, BSP e LogP. O modelo PRAM (*Parallel Random Access Machine*) foi um dos primeiros modelos de programação paralela a aparecerem (CKP+93). Esse modelo é uma extensão simples da computação paralela do modelo RAM para a computação sequencial, consistindo de um conjunto infinito de processadores, uma memória centralizada global e infinita, um conjunto de registradores de entrada e um programa finito representado por uma máquina de Turing, ou a aplicação paralela.

O modelo *Bulk Synchronous Parallel* (BSP) é dividido em diversos super passos, onde após cada passo temos uma barreira de sincronização, que garante que os processos estejam todos no mesmo estado antes do próximo super passos. A comunicação e computação das aplicações são executados entre os super passos, com a única limitação que todas as comunicações precisam ser terminadas antes da barreira de sincronização. O modelo BSP é interessante pois permite uma previsão de desempenho em determinada arquitetura desde que dois parâmetros da arquitetura sejam conhecidos: o parâmetro l (o tempo de duração de uma barreira de sincronização) e g (a taxa de banda que os dados podem ser transmitidos) (ST98). O modelo CGM (*Coarse Grained Multicomputer*) pode ser considerado como um caso especial do modelo BSP, basicamente impondo uma restrição adicional no tamanho do grão do algoritmo. Ou seja, algoritmos no modelo CGM podem ser portados facilmente para o modelo BSP.

O modelo LogP foi proposto por *Culler et al.* e recebeu o este nome por conta das variáveis do modelo: latência (L), *overhead* (o), banda de comunicação (g), número de processadores (P). Os autores do modelo perceberam que o modelo PRAM não é realista devido à falta de parâmetros para representar os custos de comunicação nas aplicações paralelas, especialmente para aplicações distribuídas (CKP+93). Apesar de interessante, o modelo LogP e BSP podem ser vistos como modelos próximo um do outro para modelagem de programas paralelos. Porém o modelo BSP é mais simples e portável, além de ser um pouco mais poderoso (BHP+96).

Os modelos anteriormente citados são modelos relativamente abstratos. O modelo BSP, por exemplo, decompõe o programa em super passos de forma explícita, porém o mapeamento de processos e sincronização são implícitos. Temos também modelos mais explícitos, como o modelo de *troca de mensagens* e *memória compartilhada*.

Troca de mensagens é uma das tecnologias padrões em sistemas distribuídos do tipo MIMD, e por isso estão disponíveis em praticamente qualquer arquitetura desse tipo. É uma interface de baixo nível, que usa funções do tipo *send* e *receive* para especificar a mensagem a ser enviada, o processo destinatário e seu destino (ST98).

Já o modelo de *memória compartilhada* permite o compartilhamento de informações entre processadores independentes por meio de um endereçamento comum de memória entre os diversos processadores (ST98). Também é uma das tecnologias mais utilizadas em sistemas do tipo MIMD.

Neste trabalho usaremos programas escritos em MPI, que se utiliza do mecanismo de troca de mensagens, e MPI, que se utiliza do mecanismo de memória compartilhada (ST98). Veremos mais sobre eles a seguir.

MPI

MPI é uma padrão para escrita de programas paralelos que define uma *Application Programming Interface* (API) implementando um modelo de programação paralela por troca de mensagens. MPI é bem sucedido e é um dos modelos dominantes para programas altamente escaláveis em computação científica (Pad11).

A popularidade do MPI torna ele uma das escolhas óbvias para a escrita de benchmarks para computação de alto desempenho. Clássicos benchmarks paralelos, como o *High Performance Linpack (HPL)*[3], *HPC Challenge Benchmark*[4], *PARKBENCH*[5] e o *NAS Parallel Benchmarks*[6] possuem implementações escritas usando o MPI.

OpenMP

OpenMP é uma API para paralelizar programas sequenciais escritos em C, C++ e Fortran em plataformas de memória compartilhada. Ele provê uma coleção de diretivas de compilação, uma biblioteca de execução, e variáveis de ambiente para permitir programadores especificar a exploração do paralelismo desejado em um programa (Pad11).

O fato do OpenMP permitir a paralelização de programas sequenciais com simples diretivas de compilação torna o OpenMP uma alternativa popular para computação de alto desempenho. Apesar de o OpenMP ser usado estritamente para ambientes de memória compartilhada, é possível combinar o OpenMP com o MPI para a criação de programas híbridos, onde o OpenMP é usado para comunicações internas via memória compartilhada (se aproveitando tanto da facilidade que o OpenMP oferece quanto da menor latência que a memória compartilhada oferece) e o MPI para comunicações externas via troca de mensagem. É possível encontrar diversos benchmarks que utilizam essa abordagem híbrida, como uma versão híbrida do *LINPACK* (WYB13) e o *NAS Parallel Benchmarks*[6].

2.3 Benchmarks

Benchmarks podem ser definidos como "[. . .] programas que formam um padrão de testes de desempenho para um computador e o software usado. Eles são escritos usando um modelo particular de programação e implementado por um software específico, que é o arbitro final do que o modelo de programação representa. Então um benchmark testa a interface de software com o computador, e não um tipo de arquitetura particular" (Pad11).

Cada área do conhecimento tem necessidades diferentes de desempenho. Por exemplo, temos aplicações que precisam de desempenho em ponto flutuante, outras em números inteiros; algumas aplicações são altamente paralelizáveis e outras inerentemente sequenciais; existem aplicações onde o maior tempo é gasto em Processamento, outras em Entrada/Saída. Por isso, o desempenho das aplicações não pode ser previsto e analisado usando um conjunto genérico de aplicações. Temos então que escolher um conjunto de características importantes para que nossa aplicação de *benchmark* represente bem nosso caso de uso (Pad11).

Infelizmente, a implementação de um bom benchmark é um processo complexo, geralmente com objetivos contrastantes. Porém podemos listar 5 aspectos que qualquer benchmark precisa ter (Hup09):

- **Relevância:** os resultados devem refletir algum ponto importante;

[3] http://www.netlib.org/benchmark/hpl/
[4] http://icl.cs.utk.edu/hpcc/
[5] http://www.netlib.org/parkbench/
[6] http://www.nas.nasa.gov/publications/npb.html

- **Repetibilidade:** existe confiança que o *benchmark* pode ser executado múltiplas vezes com o mesmo resultado;
- **Justo:** todos os sistemas participantes sendo comparados podem participar de forma igual;
- **Verificável:** existe confiança que o resultado documentado é real;
- **Econômico:** os testadores conseguem pagar a execução dos benchmark.

Comumente, em ordem de satisfazer os quatro últimos aspectos, o primeiro aspecto (considerado o mais importante) precisa ser sacrificado em partes. Isso não é necessariamente um problema, desde que se entenda as consequências dessa escolha (Hup09).

Além desses aspectos citados, alguns pontos adicionais são importantes no aspecto relevância: a métrica usada pelo benchmark precisa ser entendível, o uso dos recursos de software precisa ser feito de maneira realista, o uso do hardware precisa ser similar aos ambientes de consumo, longevidade da aplicação, aplicação ampla e um público-alvo bem definido (Hup09).

Considerando a popularidade de aplicações de benchmark para a análise de sistemas, não é de se surpreender que existam diversos benchmarks focados em paralelismo e/ou computação de alto desempenho. Entre alguns exemplos podemos citar o *HPC Challenge Benchmark*, *LINPACK*, *PARKBENCH* e o *NAS Parallel Benchmarks*.

2.4 Considerações sobre o capítulo

Neste capítulo definimos diversos conceitos que serão usados no trabalho. Nosso principal objetivo do trabalho é a execução eficiente de aplicações HPC em ambientes de nuvem computacional, e para isso usaremos de benchmarks paralelos para a avaliação de desempenho da nuvem.

Podemos perceber o papel central da paralelização nesse objetivo. Computação de alto desempenho exige paralelismo, graças as limitações físicas do hardware. Ao mesmo tempo, a computação em nuvem não oferece acesso aos detalhes de baixo nível do hardware (como o modelo de CPU ou sua frequência de relógio), apenas detalhes de nível mais alto (como número de núcleos de processamento em uma máquina virtual). Ou seja, para a execução eficiente de aplicações HPC na nuvem, temos que ter aplicações paralelas eficientes. No Capítulo 4 veremos possíveis problemas na execução de aplicações HPC nesse ambiente de nuvem, graças as suas características sistêmicas.

Também vimos diferentes tipos de aplicações paralelas, e as APIs de programação paralela MPI e OpenMP. MPI implementa um modelo de troca de mensagens, enquanto OpenMP implementa um modelo de memória compartilhada. Veremos mais detalhes sobre as vantagens de cada modelo também no Capítulo 4, mas por hora basta dizer que o modelo de troca de mensagens do MPI é mais versátil (permitindo o uso em diversas situações onde memória compartilhada não funcionaria), porém o modelo de memória compartilhada do OpenMP permite uma redução significativa da latência de comunicação.

Finalmente, vimos o conceito do que é um benchmark, e o que torna um benchmark bom. Veremos no Capítulo 5 mais detalhes do benchmark utilizado nesse trabalho, o NPB.

Capítulo 3

Trabalhos relacionados

Neste capítulo veremos um resumo da literatura corrente sobre a área. Começaremos vendo alguns trabalhos sobre a computação de alto desempenho na nuvens, passando para trabalhos que usam o mesmo benchmark escolhido neste trabalho, o NPB, em nuvens, explicitando o motivo porque o nosso trabalho é diferente do que já encontrado na literatura. Finalmente, veremos alguns trabalhos que analisam características do NPB, que foram usados como base deste trabalho.

3.1 Computação de alto desempenho na nuvem

Considerando a popularidade dos ambientes de nuvem computacional, é de se esperar que diversos trabalhos tenham estudado a viabilidade de aplicações de alto desempenho em uma nuvem, incluindo uma análise de seus possíveis problemas.

Em (AFG+09), descreve-se o que é a computação em nuvem, mostrando que ela não é uma ideia nova (a computação como uma utilidade, ou serviço, já é descrito pela literatura desde a década de 60), o motivo da nuvem ter se tornado popular agora, além dos modelos econômicos da nuvem. Mais importante no escopo deste trabalho, os autores listam 10 obstáculos para a computação em nuvem, o mais interessante para HPC sendo a imprevisibilidade de desempenho. Os autores observam que o escalonamento de máquinas virtuais, em especial para aplicações de alto desempenho, pode ser imprevisível. Eles também explicitam a importância da elasticidade da nuvem para HPC, considerando que não existe custo adicional de usar 20× mais recursos por 1/20 do tempo.

Em (VPB09), são apresentados as vantagens de se utilizar a nuvem para aplicações de HPC, incluindo a facilidade de operação comparado com aglomerados computacionais tradicionais e o provisionamento rápido. Eles mostram a viabilidade da nuvem para HPC com alguns casos de estudo, como a execução de classificação de genes e ressonância magnética na Amazon EC2, e mostram os ganhos de desempenho ao se utilizar mais MVs e o incremento no custo, concluindo que em alguns casos é interessante migrar sua aplicação na nuvem mesmo com o crescimento do custo de execução, desde que isso venha acompanhado de uma redução significativa do tempo total de execução.

3.2 Avaliação de desempenho na nuvem usando o NPB

O NPB é um conjunto de aplicações comumente usados para caracterizar o desempenho de arquiteturas paralelas em aplicações de computação de alto desempenho, desenvolvido pela NASA (BDBS92). Pelo fato da especificação das aplicações NPB serem feitas de

forma apenas algorítmica (BBB+91), existem diversas implementações diferentes, muitas delas desenvolvidas pela própria NASA, incluindo implementações de memória compartilhada (JJF+99), híbridas (Multi-Zone) (JdW04), e até heterogêneas (OpenCL) (SJL11). Com o advento da computação em nuvem, é natural encontrar diversos trabalhos que usam NPB para avaliar o desempenho em nuvens. Como resultado, existem diversos esforços diferentes para validar o desempenho de computação de alto desempenho em nuvens usando o NPB, que veremos nesta seção.

Em (GM12), reconhece-se a importância da possibilidade de execução de aplicações de alto desempenho em nuvens, considerando que os custos de manutenção na nuvem são basicamente zero e é uma solução efetiva tanto em relação ao custo total quanto rápida de se utilizar. Porém, de acordo com os autores, nuvens foram projetadas para negócios e aplicações Web, e não para as necessidades de aplicações de alto desempenho como baixa latência e alto uso de banda de comunicação. Então, os autores propõem que a nuvem pode ser usada em *alguns* tipos de aplicações de alto desempenho, não *todas*. Para testar sua hipóteses, os autores usam diferentes aplicações, incluindo as aplicações LU e EP do NPB. Os resultados mostram que as nuvens são viáveis para aplicações de alto desempenho que não fazem uso intensivo de comunicações.

Em (EH08), apresenta a Amazon EC2 como uma solução viável para a execução de aplicações de alto desempenho de pequeno tamanho. Eles testam essa hipótese com a execução de simulações de atmosfera oceânica em paralelo dentro da nuvem da Amazon. Os autores encontram um desempenho abaixo de supercomputadores dedicados, e no nível de aglomerados computacionais de baixo custo. Em especial, a deficiência surgem do desempenho de troca de mensagens, onde a latência e banda é entre uma e duas ordens de magnitude inferiores aos grandes computadores dedicados. Porém, os autores acham que os resultados são encorajadores, e preveem a criação de nuvens com desempenho otimizado a computação de alto desempenho, usando tecnologias de interconexão como Myrinet ou InfiniBand, além de otimizações nas camadas inferiores.

Em (AM10), é analisado a viabilidade de execução de aplicações de alto desempenho na nuvem, listando os pró e contras dessa abordagem. Eles avaliam o desempenho da Amazon EC2 executando aplicações de avaliação bem conhecidas, uma delas é o NPB. Eles observam que não existe nenhuma garantia de uma nuvem pública como a Amazon de não sobrecarregar uma máquina física, e isso pode afetar aplicações com uso mais intenso de processamento como é o caso de aplicações de alto desempenho. E novamente, eles apontam que o desempenho de comunicação é instável e isso impacta o desempenho de forma considerável. Uma nota interessante feita pelos autores desse trabalho é que o NPB usa a chamada de sistema `gettimeofday` para medir tempo. Isso é, a medida de tempo é baseada no tempo medido pelo relógio, ao invés de tempo de CPU. Isso evita inconsistências entre o tempo medido na máquina física e na máquina virtual, porém pode afetar a precisão da medida de tempo[1].

Em (HZK+10), são comparados diferentes provedores de nuvem com o supercomputadores da NCSA: Amazon EC2, GoGrid Cloud e IBM Cloud. Eles analisaram a latência de passagem de mensagens MPI e banda de saída, tanto nas versões NPB-OMP quanto NPB-MPI. Os autores não recomendam nenhuma nuvem na época em que o artigo foi feito, principalmente pela alta latência de passagem de mensagem do MPI. Eles concluem que, primeiro, a virtualização tem pouco impacto no desempenho total; segundo, a maioria das

[1] O uso da chamada `gettimeofday` para medida de tempo de execução pode ser afetado por alterações do relógio da máquina, por exemplo, feitas pelo administrador do sistema ou o serviço NTP (http://man7.org/linux/man-pages/man2/gettimeofday.2.html). Porém, no caso do NTP pelo menos, isso não deve afetar a medida de forma considerável, considerando que as correções de tempo do NTP são feitas em pequenos passos (http://doc.ntp.org/4.1.0/ntpd.htm).

nuvens públicas não são otimizadas para computação de alto desempenho.

Em (ZLZ+11), comparam-se instâncias otimizadas para computação de alto desempenho da Amazon EC2 (*Cluster Compute Instances*[2]) com um aglomerado computacional conectado com InfiniBand QDR. Eles usam aplicações do NPB, tanto classe C e D, usando 64 e 128 processadores lógicos. Os autores encontraram que a maior proporção da relação computação/comunicação da classe D melhore o desempenho relativo do NPB na Amazon EC2 comparado com a classe C, mais uma vez confirmando o problema de rede no desempenho da nuvem.

Em (MDH+12), compara-se o desempenho de instâncias otimizadas para computação de alto desempenho na Amazon EC2 contra o supercomputador Pleiades da NASA, um aglomerado computacional *SGI ICE*. Eles mostraram que o desempenho dos dois sistemas para um nó de núcleo único é equivalente. Porém, ao aumentar o número de núcleos, a diferença do EC2 para o Pleiades aumenta em função das características de rede do primeiro. Os autores testaram sua hipótese com o uso de diferentes aplicações de teste, incluindo as aplicações CG, FT, MG, EP, IS, BT, SP e LU do NPB, versões MPI. Os autores concluíram que a Amazon EC2 ainda não conseguem competir com sistemas orientados a computação de alto desempenho como o Pleiades.

Finalmente, em (LORZ13), é analisado o desempenho do Google Compute Engine para aplicações de computação de alto desempenho. Apesar dos esforços dos autores, esse foi o único trabalho encontrado com exceção do nosso focado nesse tema. Eles analisam diversas aplicações para medir a latência e banda de passagem da rede, e velocidade de processamento usando outras aplicações, entre elas o NPB. Os autores observam que o GCE tem, relativamente, uma alta banda de memória, e bom desempenho tanto em velocidade de processamento como armazenamento. Porém, existe uma escalabilidade ruim em relação ao número de vCPUs, provavelmente porque as instâncias do GCE são baseadas em *Hyper-Threading*. Os autores concluem que o GCE tem uma alta proporção desempenho/preço, sendo uma alternativa interessante e competitiva ao Amazon EC2 para computação de alto desempenho.

Nosso trabalho é relacionado com os outros trabalhos apresentados nesta seção, avaliando o desempenho de nuvens computacionais para computação de alto desempenho. Em especial, nosso trabalho foca em duas abordagens diferentes de comunicação em aplicações de alto desempenho: intra-nós, isso é, a comunicação entre processos no mesmo nó (no nosso caso, na mesma máquina virtual), via tanto multi-threading quanto troca de mensagens; e entre-nós, isso é, comunicação entre processos em diferentes nós (MVs), via troca de mensagens. Avaliamos o impacto do aumento ou diminuição da comunicação intra/entre-nós, variando o número de MVs disponíveis em um mesmo nó e seu número de CPU virtual (vCPU)s.

3.3 Análise de características do NPB

Como nosso trabalho foca no NPB para analisar as características únicas de ambientes de nuvem computacionais, é necessário um entendimento melhor das características de comunicação do NPB para realizar a análise. Por isso, os trabalhos dessa seção foram utilizados na nossa análise, e estão descritos com mais detalhes a seguir.

Em (MTM+09), é desenvolvido um abordagem para analisar padrões de comunicação semelhantes em aplicações científicas paralelas. Eles analisam quatro propriedades principais, chamadas *temporal, espacial, volume* e *grafos de comunicação*. Após isso, eles aplicam uma transformação matemática para normalizar os dados, permitindo comparar de similarida-

[2] https://aws.amazon.com/blogs/aws/the-new-amazon-ec2-instance-type-the-cluster-compute-instance/

des e contrastar as diferenças entre aplicações, independente do tamanho. A transformação aplicada é expressada por duas métricas: θ, onde $-1 \leq \theta \leq 1$, que quantifica a similaridade baseada nas propriedades temporais, espaciais ou de volume; e ϵ, onde $0 \leq \epsilon \leq 1$, que quantifica as similaridades entre dois grafos de comunicação (ou máximo subgrafo comum). Destas duas métricas, eles quantificam a similaridade entre aplicações. Eles testam essa abordagem usando quatro aplicações do *NAS Parallel Benchmarks* versão 3.0: BT, SP, MG e LU. Após a aplicação da metodologia descrita anteriormente nas aplicações citadas, os autores mostram que as aplicações têm padrões de comunicações semelhantes independentemente do tamanho de entrada (neste trabalho foram consideradas as classes A, B e C) e mesmo em relação ao número de processadores lógicos (eles documentam testes com 16 e 64 *processadores lógicos*). No entanto, os padrões de comunicação entre diferentes aplicações é diferente: BT e SP têm propriedades temporais e espaciais similares, porém diferentes volumes de dados. LU é completamente não relacionado com tanto BT quanto SP. Esse trabalho confirmam as afirmações dos autores originais do NPB (BBB+91).

Uma limitação de (MTM+09) é que eles consideram apenas duas funções MPI para obter o θ e ϵ, `MPI_Send` e `MPI_Recv`. Porém, apesar das aplicações NPB incluírem chamadas de comunicação coletivas (como o `MPI_Bcast`), em (FY02), analisou-se as características de comunicação das implementações MPI do NPB, para estudar a efetividade de otimizações de comunicação em tempo de compilação para aplicações MPI. Os testes dos autores mostram que funções de comunicação *coletivas* no NPB (incluindo `MPI_Allreduce`, `MPI_Bcast`, `MPI_Barrier`, etc.) são apenas uma pequena parte da comunicação em aplicações NPB, enquanto comunicações *ponto-a-ponto* (`MPI_Send` ou `MPI_Isend` e `MPI_Recv` ou `MPI_Irecv`) são a maior parte das comunicações no NPB, validando os resultados de (MTM+09) para a aplicação como um todo.

3.4 Considerações sobre o capítulo

Nesse capítulo vimos a literatura atual sobre o análise de desempenho de nuvens para aplicações de alto desempenho, um dos temas centrais do trabalho. Entre os diversos trabalhos analisados, nenhum deles foca no impacto da comunicação intra-nós e entre-nós, apesar de alguns trabalhos (como o (LORZ13)) analisarem o impacto da tecnologia Hyper-Threading, assim como esse trabalho.

Também vimos alguns trabalhos que analisam as características do benchmark escolhido, o NPB. Em especial, utilizaremos o trabalho de (MTM+09) para desenvolver parte do Capítulo 5, onde veremos uma análise mais profunda do NPB.

Capítulo 4

Processamento paralelo em ambiente de nuvem

Se formos utilizar a nuvem como uma infraestrutura para desenvolvimento de aplicações de alto desempenho, é necessário entender como o paradigma de programação paralela e/ou concorrente será explorado. Neste capítulo analisaremos este aspecto. Começaremos definindo alguns termos gerais usados no decorrer do trabalho, além de algumas considerações preliminares sobre nuvens computacionais. Depois veremos ambientes de execução de computação de alto desempenho, passando pelas arquiteturas de alto desempenho clássicas, a importância do paralelismo para a computação de alto desempenho e como a nuvem pode ser vista como um ambiente para execução de aplicações de alto desempenho. Depois veremos modelos clássicos de programação paralela, vendo o porque eles foram criados e breves descrições dos modelos mais conhecidos. Fecharemos esse capítulo falando como o modelo de programação paralela utilizado neste trabalho será adaptado para o ambiente de computação em nuvem.

4.1 Considerações preliminares sobre a nuvem

Em uma arquitetura em nuvem podemos instanciar múltiplas MVs, cada uma com um determinado número de vCPUs (CPUs alocada a uma determinada máquina virtual) e quantidade de RAM virtual (vRAM) (memória RAM alocada a uma determinada máquina virtual). Um detalhe interessante para se notar é que um recurso virtual, como uma vCPU ou vRAM, não necessariamente está alocada de forma exclusiva para determinada MV. Ou seja, um núcleo físico vai executar processos de múltiplas MV, dependendo do escalonador da máquina física, assim como um mesmo espaço de memória pode ser alocado por diferentes MV.

Outros recursos que são compartilhados entre as máquinas virtuais são os recursos de entrada e saída: rede e disco. A rede é compartilhada entre as MVs e máquinas físicas, e dentro de uma nuvem pública não temos muitas informações sobre a sua topologia: logicamente, temos acesso a qualquer MV a partir de qualquer MV. Porém, dependendo da topologia de rede temos diferentes latências dependendo da localização da máquina física. Isso significa que, embora não tenhamos informações detalhadas sobre a rede física em nuvens públicas, essa informação ainda é importante para aplicações de alto desempenho. Porém, só conseguimos essa informações de forma indireta, usando por exemplo aplicações que testam a latência como mostrado em (HZK+10; LORZ13). Como visto no Capítulo 3, o problema de latência e banda de rede é um problema real para aplicações de alto desempenho.

Quanto ao armazenamento ele geralmente é fornecido como um serviço separado, in-

tegrado com os outros serviços do IaaS. Nuvens geralmente oferecem diferentes tipos de armazenamento de acordo com a necessidade da aplicação. Por exemplo no *Google Compute Engine* (GCE), temos HDs tradicionais para armazenamento de informações de forma eficiente, SSDs para operações em disco que precisam de desempenho, *buckets* para armazenamento de informações a longo prazo e até mesmo "armazenamento" na memória RAM para máximo desempenho. Cada um dos diferentes tipos de armazenamento tem diferentes preços por GB, permitindo alta flexibilidade no uso dos recursos[1]. Interessante notar que a nuvem oferece espaços de armazenamento muitas vezes maior do que o disponível por um único disco comercialmente (por exemplo, o GCE oferece até 64TB de espaço de armazenamento em uma única instância), pois nuvens geralmente usam um sistema de arquivos distribuído, como o GlusterFS[2], ao invés de dar acesso direto ao disco.

Pelo fato da computação em nuvem ser baseada na tecnologia de virtualização, existe um possível *overhead* causado pela virtualização de hardware. Porém, apesar desse *overhead* ser mensurável (AA06), ele é pequeno o suficiente a ponto de não inviabilizar o seu uso em aplicações de alto desempenho (JDV+09; HZK+10; GBC12).

Um exemplo real da arquitetura descrita acima pode ser vista na implementação do OpenStack. O módulo de computação, chamado de *nova*, é responsável pelo escalonamento das MVs requisitadas pelos usuários nas máquinas físicas, e por padrão faz o balanceamento de carga por um algoritmo similar ao *round-robin*. A rede é gerenciada por outro módulo, conhecido como *neutron*, que permite a criação de redes privadas entre as MVs. O *cinder* é o módulo de armazenamento, permitindo que clientes requisitem recursos de rede e conectem-se nas MVs (Pep11).

O provedor de nuvem possui mecanismos para promover o balanceamento de carga gerada pelas MVs sobre a infraestrutura física disponível, graças ao escalonador de máquinas virtuais disponível no ambiente. Isso pode resultar em escolhas ruins do ponto de vista de desempenho, como por exemplo, duas aplicações com alto uso de CPU sendo escalonadas em uma mesma máquina física.

Do ponto de vista do escalonador de MVs, podemos pensar na granularidade das tarefas, que nesse caso são MVs. Uma MV grande ou de grão-grosso é uma MV com um grande número de vCPUs, enquanto uma MV pequena ou de grão-fino é uma MV com um pequeno número de vCPUs. Esse número "grande" ou "pequeno" é relativo à maior MV disponível em determinado ambiente de nuvem. Então, por exemplo, considerando que o Google oferece máquinas virtuais de até 32 vCPUs, uma MV de grão-grosso teria 32 vCPUs, enquanto uma MV de grão-fino teria 1 ou 2 vCPUs cada. Esse conceito de granularidade faz sentido, em especial se pensarmos que uma MV de grão-grosso faz a maior parte das suas comunicações internamente, já que seus processos estão na sua maioria dentro da mesma MV, enquanto uma MV de grão-fino faz a maior parte das suas comunicações externamente, já que os processos estão distribuídos em múltiplas MVs. Veremos isso posteriormente no Capítulo 6.

A granularidade das MVs pode ter um impacto significativo no desempenho, principalmente em aplicações de alto desempenho (WHV10). Por exemplo uma MV de grão-grosso exige uma quantidade maior de recursos de hardware (mais vCPUs, mais vRAM, etc.), e isso reduz as opções do escalonador para a sua alocação, pois será necessário encontrar máquinas físicas relativamente livres. MVs de grão-fino, com uma ou duas vCPUs por exemplo, podem ser escalonadas em outras máquinas físicas com MVs já em execução.

[1] https://cloud.google.com/compute/docs/disks/
[2] https://www.gluster.org/

4.2 Ambientes de execução de computação de alto desempenho

Uma arquitetura paralela moderna para processamento de alto desempenho consiste de vários nós multiprocessados, interligados por rede, podendo ser ou não ser Hyper-Threading. Tipicamente esses nós possuem uma arquitetura do tipo NUMA (*Non-Uniform Memory Address*, ou seja, os acessos a memória têm latências diferentes dependendo do tipo de memória sendo acessada). Eventualmente, esses nós dispõem também de capacidade de processamento heterogênea, tanto em quantidade de memória e poder de processamento (número de núcleos, frequência e/ou diferentes CPUs), como diferenças arquiteturais (arquiteturas híbridas misturando CPU e GPU, por exemplo).

A materialização dessa arquitetura pode ser na forma de um supercomputador, como por exemplo o TaihuLight [3] ou o Santos Dumont [4] ou mesmo de um aglomerado computacional *Beowulf*. As últimas são populares em função do custo, porém requerem manutenção e administração constantes. Supercomputadores, por outro lado, geralmente estão disponíveis apenas a um grupo seleto de pessoas, exige um alto investimento, tem altos custos operacionais (um exemplo foi o desligamento do Santos Dumont graças aos altos custos energéticos [5]) e exige aplicações altamente escaláveis para tirar proveito dos recursos.

Tanto um supercomputador quanto um aglomerado computacional permitem aos seus usuários submeterem tarefas, geralmente do tipo *bag-of-tasks* (um conjunto com várias tarefas com um determinado objetivo), que são executadas de acordo com o escalonador de tarefas do sistema. Isso significa que tarefas podem demorar um certo tempo até serem executadas, caso o sistema esteja ocupado com outras tarefas. Ainda existe, dentro do sistema operacional de cada nó, um escalonador de tarefas que fica responsável por escalonar os diferentes *threads*/processos nas CPUs.

As nuvens públicas apresentam uma alternativa interessante para supercomputadores e aglomerados computacionais. As duas características que dão a nuvem uma vantagem sobre essas infraestruturas clássicas são a *rápida elasticidade* e o *serviço mensurado*. A rápida elasticidade permite ao usuário, na prática, a alocação de recursos ilimitados. O serviço mensurado permite um baixo investimento inicial e nenhum custo de manutenção. Seu custo operacional depende exclusivamente do uso. Essas duas características dão ao usuário uma flexibilidade muito grande. Na prática, não há diferenças de preço entre usar 1 CPU por 1000 horas ou 1000 CPUs por 1 hora.

Enquanto não é de se esperar que todos os usuários de computação de alto desempenho migrem completamente para a nuvem, a nuvem oferece algumas alternativas interessantes mesmo para usuários de instituições que investiram uma grande quantidade de recursos em aglomerados computacionais ou supercomputadores. Um exemplo é um instituto que tem acesso a um pequeno aglomerado com 8 máquinas de 32 CPUs cada, e quer adquirir 8 novas máquinas para diminuir o tempo de execução de uma aplicação. Ao invés de simplesmente adquirir mais 8 máquinas, com o potencial de ver a aplicação executando no mesmo ou até em pior tempo, uma alternativa é executar um teste em uma nuvem com configuração semelhante e verificar se o aplicativo escala como deveria.

Podemos comparar uma nuvem computacional com os ambientes de computação de alto desempenho mais tradicionais. Em um alto nível elas são bastante semelhantes: temos um grande conjunto de nós multiprocessados interligados via rede, possível heterogeneidade de

[3] https://www.top500.org/system/178764

[4] https://www.top500.org/system/178570

[5] http://oglobo.globo.com/sociedade/tecnologia/supercomputador-brasileiro-pode-ser-desligado-por-falta-de-recursos-19558324

processamento entre diferentes nós, arquiteturas com diferentes latências de memória, e mais recentemente até arquiteturas híbridas com GPU na nuvem. Porém, dentro de um ambiente de nuvem computacional temos uma camada adicional, que vem do uso da tecnologia de virtualização. As várias máquinas físicas de uma nuvem são gerenciadas por um escalonador de tarefas, que atribui às requisições dos usuários da nuvem. Dentro de cada uma das máquinas físicas estão sendo executadas múltiplas máquinas virtuais, cada um com seu escalonador de processos que gerencia as máquinas virtuais, enquanto as próprias máquinas virtuais têm um escalonador de processos que gerencia os processos do usuário.

Uma alternativa para essa arquitetura acima é a utilização de contêineres ao invés de virtualização. Os contêineres isolam cada um dos ambientes de execução no espaço de usuário, assim como no ambiente de virtualização, porém o kernel usado entre a máquina física e o contêiner é o mesmo, portanto existe apenas um escalonador de processos para todos os contêineres da máquina física. Uma outra opção ainda é o provisionamento direto de máquinas físicas, conhecido como *bare-metal*, o que pode ser interessante para ser avaliado futuramente, considerando que alguns provedores de nuvem fornecem essa opção.

4.3 Modelo considerado

Com base no que foi visto neste capítulo, podemos descrever um modelo de execução de aplicações de alto desempenho em ambientes de nuvem. Temos o *escalonador de máquinas virtuais*, que recebe as requisições de MVs dos usuários, cada uma com suas vCPUs e determinada quantidade de vRAM, e escalona elas em suas devidas *máquinas físicas*, de acordo com alguma estratégia de escalonamento. Porém, nenhuma premissa é assumida para a estratégia de escalonamento de MVs sobre as máquinas físicas providas pela infraestrutura, considerando que nuvens públicas não fornecem esse tipo de mecanismo. Cada máquina virtual executa processos que são gerenciados tanto pelo escalonador de processos da máquina virtual quanto da máquina física. Na Figura 4.1 apresentamos um esquema desse modelo.

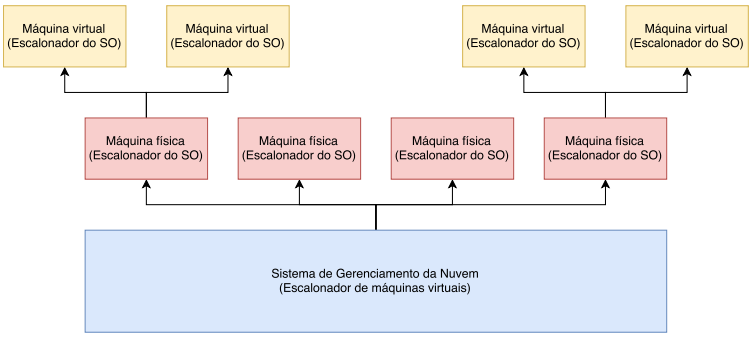


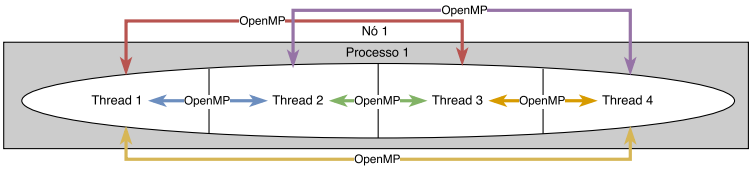
Figura 4.1: *Esquema de escalonadores em um ambiente de nuvem.*

As aplicações são escritas utilizando MPI ou OpenMP de forma única, ou ainda em combinação de OpenMP+MPI, como descrito na Seção 2.2.2. Aplicações que utilizam MPI puro ou OpenMP puro serão chamados de implementações *Single-Zone* (SZ), ou zona única. Elas possuem esse nome pois do ponto de vista do desenvolvedor, o acesso a informação de qualquer um dos processos/*threads* disponíveis é feito da mesma maneira. Aplicações que utilizam uma combinação de OpenMP+MPI serão chamadas de *Multi-Zone* (MZ), ou zona múltipla. Nesses casos o acesso de dados entre *threads* (OpenMP) e processos (MPI) é feito de maneira diferente.

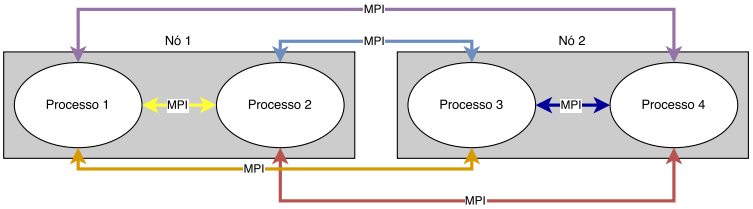
Quanto as comunicações feitas pelas aplicações paralelas utilizadas, podemos classificar as comunicações em dois:

- Intra-nós: a comunicação intra-nós ocorre quando um processo se comunica com outro dentro do mesmo nó. Essa comunicação pode ocorrer tanto por memória compartilhada quanto por troca de mensagens;
- Entre-nós: a comunicação entre-nós ocorre quando um processo de um nó se comunica com outro processo de outro nó. Essa comunicação acontece, mais comumente, por troca de mensagens.

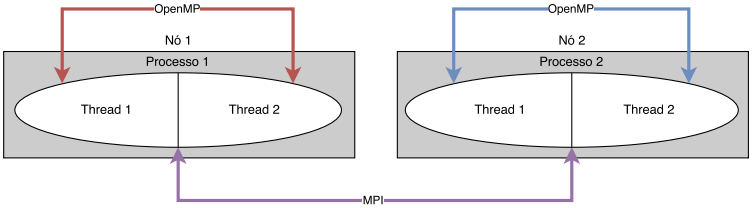
A Figura 4.2 mostra um esquema de cada um dos tipos de comunicação citados. Aplicações OpenMP puras utilizam-se apenas de comunicação intra-nós (Figura 4.2a), pelo fato de se utilizarem de mecanismos de memória compartilhada. Aplicações MPI puras fazem comunicações tanto intra-nós como entre-nós, sempre utilizando trocas de mensagens (Figura 4.2b). Aplicações OpenMP+MPI fazem comunicações intra-nós usando OpenMP e comunicações entre-nós utilizando o MPI (Figura 4.2c).



(a) Single-Zone (OpenMP).



(b) Single-Zone (MPI).



(c) Multi-Zone.

Figura 4.2: *Esquema de comunicação de aplicações NPB.*

4.4 Considerações sobre o capítulo

Vimos neste capítulo um pouco sobre ambientes de execução de aplicações de alto desempenho. Mostramos como um ambiente de nuvem se compara com uma clássica arquitetura de computação de alto desempenho, um aglomerado *Beowulf*, além de vermos modelos de programação voltados para aplicações paralelas e suas implicações. Finalmente, mostramos um modelo do que esperamos ver durante a execução de aplicações paralelas na nuvem.

Um dos pontos mais importantes levantados é o compartilhamento de recursos em um ambiente de nuvem. Como cada máquina virtual executa sobre uma máquina física, que pode ser compartilhada com outras máquinas virtuais, isso torna o desempenho de uma nuvem

menos previsível que um aglomerado *Beowulf*, por exemplo. No aglomerado computacional, por geralmente ser um ambiente privado, pode-se isolar o ambiente para apenas um ou poucos experimentos, permitindo garantia de desempenho. Mesmo quando o ambiente é compartilhado por múltiplos usuários existe algum sistema de gerenciamento que permite a garantia do desempenho. Em um ambiente de nuvem esse tipo de garantia é feito pelo provedor de serviços, sem o controle do usuário.

O uso de uma técnica de memória compartilhada (OpenMP), pelo menos em teoria, deve reduzir a latência de comunicação. Porém isso não é o único fator no desempenho final da aplicação. Pelos fatos citados anteriormente no capítulo, usar somente OpenMP em um ambiente de nuvem, com objetivo de reduzir a latência, pode nem sempre resultar no melhor desempenho.

Isso acontece pois o uso de OpenMP exige máquinas virtuais de grão-grosso em aplicações massivamente paralelas, pois não podemos usar memória compartilhada em um ambiente distribuído. Porém, como consequência de se utilizar MVs maiores podemos ter um uso ineficiente dos recursos computacionais que a nuvem oferece, considerando que o escalonador de MVs terá menos opções de máquinas físicas para escalonar determinada MV.

Por outro lado, MVs de grão-fino podem ser escalonadas de maneira mais simples e eficiente, porém ao mesmo tempo podem aumentar a latência de comunicação de forma significativa, tanto pelo uso de mecanismos de troca de mensagem ao invés de memória compartilhada, quanto pelo fato de que duas máquinas virtuais podem ser escalonadas em máquina físicas distantes. Para conseguir o melhor desempenho, é necessário conseguir um equilíbrio entre a latência induzida e granularidade de máquinas virtuais.

Capítulo 5

NAS Parallel Benchmarks

O *NAS Parallel Benchmarks* (NPB) é um conjunto de programas desenvolvidos para avaliar o desempenho de computadores e supercomputadores paralelos, desenvolvido pela NASA. Esses benchmarks são derivados de aplicações de fluidodinâmica computacional. Eles consistem de cinco núcleos de aplicação paralelos (pequenas aplicações que testam um comportamento específico) e três aplicações simuladas (BBB+91).

O NPB foi desenvolvido com diversos objetivos. (BBB+91) não consideram benchmarks clássicos como o Livermoore Loops, LINPACK e NAS Kernels como apropriados para máquinas paralelas modernas por não terem requerimentos computacionais e de memória compatíveis com o vasto incremento computacional das últimas décadas. Por outro lado, os autores não consideram aplicações científicas completas apropriadas também, pela dificuldade de se adaptar um grande programa para uma nova arquitetura de forma eficiente, e a dificuldade da tarefa para se obter um parâmetro comparativo com outras arquiteturas. Outras considerações feitas por (BBB+91) são:

- Sistemas paralelos avançados requerem novos algoritmos e abordagens de software, e esses métodos geralmente são diferentes de métodos convencionais;
- Benchmarks precisam que ser genéricos e não devem favorecer nenhuma arquitetura paralela;
- A corretude dos resultados e os números de desempenho apresentados devem ser facilmente verificados;
- O uso de memória e a necessidade total de processamento devem ser facilmente ajustados para novos sistemas com maiores poderes de processamento;
- O benchmark precisa que ser facilmente distribuído.

Podemos comparar as considerações do (BBB+91) quanto ao desenvolvimento do NPB com os aspectos que qualquer benchmark deve apresentar por (Hup09) mostradas no Capítulo 2. O aspecto *Justo* é comparável ao segundo ponto; o aspecto *Verificável* é comparável ao terceiro ponto; o aspecto *Econômico* não aparece nesses pontos acima, porém entra na ideia do porque usar um benchmark e não portar uma aplicação científica completa.

Temos aqui dois pontos que não foram cobertos em (Hup09): o primeiro e quarto pontos. O quarto ponto é o que permite o NPB ser relevante até hoje. A versão de referência do NPB foi atualizado de forma a ser utilizado até hoje (como vimos no Capítulo 3). Existem tamanhos de entrada para o NPB, como a classe D ou E, que utilizam dezenas ou centenas de GB de memória RAM, além de computação que demoram várias horas ou mesmo

dias mesmo em grandes sistemas modernos. O primeiro ponto é o que permitiu as diversas implementações diferentes do NPB, incluindo OpenMP, MPI e até mesmo OpenCL.

Neste trabalho, selecionamos as três aplicações simuladas, BT, LU e SP. Essas aplicações resolvem as equações instáveis, compressíveis de Navier-Stokes em três dimensões espaciais (dWH03), cada uma usando uma estratégia diferente (BBB+91):

- **LU:** uma solução regular esparsa, em bloco (5×5), triangular inferior e superior para um sistema de equações lineares. Este problema exibe uma quantidade limitada de paralelismo se comparada com o SP;
- **SP:** solução de múltiplos, sistemas independentes não diagonalmente dominantes, escalar, de equações penta diagonais.
- **BT:** solução de múltiplos, sistemas independentes não diagonalmente dominantes, de equações bloco tridiagonais com tamanho de bloco (5×5). De acordo com (BBB+91), o SP e o BT são similares em diversos aspectos, com exceção da proporção entre comunicação e computação.

Nós escolhemos as aplicações BT, LU e SP do NPB, porque essas aplicações possuem implementações tanto SZ (OpenMP pura ou MPI pura) e MZ (OpenMP+MPI). Como vimos no Capítulo 4, aplicações híbridas que utilizam tanto OpenMP e MPI não são incomuns. Os próprios autores do NPB reconhecem a importância desse tipo de implementação híbrida (dWH03), e por isso escolhemos as versões MZ das aplicações citadas.

As versões SZ e MZ possuem aproximadamente o mesmo tamanho, e as soluções apresentadas são equivalentes (dWH03). Isso permite a comparação entre as versões SZ e MZ, permitindo analisar o impacto da redução de latência nas versões MZ comparado com a SZ (em especial, na versão MPI pura). A versão OpenMP SZ serve como base para uma versão com o menor impacto possível de latência de comunicação e maior banda, porém é menos escalável dado o uso de mecanismos de memória compartilhada.

Como vimos no Capítulo 4, temos comunicações intra-nós e entre-nós. Como estamos considerando máquinas virtuais nesse caso, dizemos que a comunicação intra-nós é entre processos numa mesma MV, enquanto entre-nós é entre processos em MVs distintas.

As aplicações do NPB são moldáveis, ou seja, permitem a alteração do número de processos a serem usados pela aplicação antes da execução da aplicação. Porém, a moldabilidade de algumas aplicações do NPB é limitada. Por exemplo, nas versões MPI algumas aplicações exigem número quadrático de processos. As versões MZ (OpenMP+MPI) possuem limite máximo do número de processos MPI.

5.1 Padrões de comunicação no NPB

Como visto no Capítulo 2, um dos principais problemas de executar aplicações de alto desempenho na nuvem são causados pela comunicação. Então, avaliar como os processos de uma aplicação sem comunicam entre si é importante para a análise dos dados dos experimentos.

Uma das maneiras de caracterizar as comunicações que uma aplicação fazendo uma análise do seu padrão de comunicação.

Uma das maneiras de caracterizar as comunicações que uma aplicação paralela e distribuída realiza é fazendo uma análise do seu padrão de comunicação. É de se esperar que aplicações com padrões de comunicação semelhantes tenham um comportamento semelhante durante a execução, salvo o tempo de processamento do problema em si. Como desenvolvido

em (MTM+09), podemos comparar o padrão de comunicação de duas aplicações distintas coletando os seguintes dados de comunicação brutos:

- **Temporal:** é um vetor representando a relação entre a quantidade de dados enviados por um determinado processo sobre a quantidade de trabalho que o mesmo realiza. Ou seja, um valor *temporal* alto indica um processo que passa muito tempo comunicando e pouco tempo processando, enquanto um valor baixo indica o contrário e 0 indicaria um processo que não se comunica. Esse parâmetro é definido pelo seguinte vetor:

$$temporal =< temporal_1, temporal_2, ..., temporal_n >$$

  Onde $temporal_1$ é a propriedade temporal do *processo 1*, $volume_2$ é o mesmo para o *processo 2* e assim por diante. Se considerarmos $temporal_j$ como a propriedade temporal do *processo j*, temos então:

$$temporal_j = |enviadas_j|/trabalho_j$$

  Onde $|enviadas_j|$ é a quantidade de mensagens enviadas pelo *processo j*, e o $trabalho_j$ é a quantidade de trabalho realizada pelo mesmo;

- **Espacial:** é um vetor representando com quais processos vizinhos determinado processo se comunica. Esse parâmetro é definido pelo seguinte vetor:

$$espacial =< espacial_1, espacial_2, ..., espacial_n >$$

  Onde $espacial_1$ é a propriedade espacial do *processo 1*, $volume_2$ é o mesmo para o *processo 2* e assim por diante. Se considerarmos $temporal_j$ como a propriedade espacial do *processo j*, temos então:

$$espacial_j = |vizinhos_j|$$

  Onde $vizinhos_j$ indica o número de processos que se comunica com o *processo j* (tanto envio quanto recebimento);

- **Volume:** é um vetor representando o total de volume de comunicação trocado por cada processo da aplicação. Esse parâmetro é definido pelo seguinte vetor:

$$volume =< volume_1, volume_2, ..., volume_n >$$

  Onde $volume_1$ é a propriedade de volume do *processo 1*, $volume_2$ para o *processo 2* e assim por diante. Se considerarmos $volume_j$ como a propriedade temporal do *processo j*, temos então:

$$volume_j = \sum_{m=1}^{|enviadas_j|} tamanho\_da\_mensagem_{j,m}$$

  Onde $|enviadas_j|$ é a quantidade de mensagens enviadas pelo *processo j*, e o $tamanho_d a_m ensagem_{j,m}$ é o tamanho da mensagem enviada do *processo j* ao *processo m*;

- **Grafo de comunicação:** um grafo que mostra as relações de comunicação entre cada um dos processos da aplicação.

Após isso, eles aplicam uma transformação matemática para normalizar os dados, permitindo comparar de similaridades e contrastar as diferenças entre aplicações, independente do tamanho. A transformação aplicada é expressada por duas métricas: θ, onde $-1 \leq \theta \leq 1$, que quantifica a similaridade baseada nas propriedades temporais, espaciais ou de volume; e ϵ, onde $0 \leq \epsilon \leq 1$, que quantifica as similaridades entre dois grafos de comunicação (ou máximo subgrafo comum).

Com isso, (MTM+09) gerou a Tabela 5.1, comparando a similaridade de aplicações do NPB em diferentes tamanhos de classe.

Tabela 5.1: *Padrões de similaridade na comunicação com problemas em diferentes classes de tamanho (16 processos MPI) por (MTM+09).*

Aplicação	Problema	$\theta_{temporal}$	θ_{volume}	$\theta_{espacial}$	(θ, ϵ)
BT	A-B	1,000	1,000	1,000	(1,000, 1,000)
	B-C	1,000	1,000	1,000	(1,000, 1,000)
	C-A	1,000	1,000	1,000	(1,000,1,000)
LU	A-B	0,992	0,998	1,000	(0,997, 1,000)
	B-C	0,998	1,000	1,000	(0,999, 1,000)
	C-A	0.997	0,999	1,000	(0,997,1,000)
SP	A-B	1,000	0,822	1,000	(0,941, 1,000)
	B-C	1,000	0,682	1,000	(0,894, 1,000)
	C-A	1,000	0,655	1,000	(0,885,1,000)

Em uma primeira observação dos dados da Tabela 5.1 destaca-se a correlação positiva entre as propriedades temporal e espacial (θ próximo ou igual à 1), o que indica uma semelhança nas propriedades de comunicação temporal e espacial, independente do tamanho do problema. Ou seja, mesmo com o crescimento do tamanho de entrada, cada processo continua enviando os dados em períodos de tempo semelhantes (propriedade temporal) e se comunica com os mesmos processos (propriedade espacial). A propriedade de volume, ou seja, a quantidade de dados enviados, também é semelhante nas aplicações BT e LU, porém não no SP, o que indica que existem algumas diferenças no volume total de dados enviados pelo SP dependendo do tamanho da classe utilizada.

(MTM+09) também fez uma análise comparando a similaridade das diferentes aplicações do NPB, variando o número de processos MPI. Vemos essa comparação na Tabela 5.2.

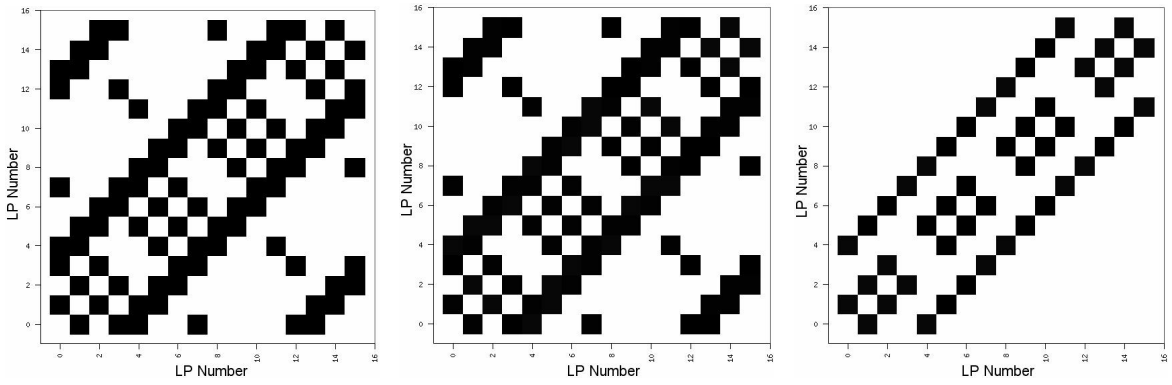
Tabela 5.2: *Padrões de similaridade na comunicação com diferentes problemas com diferente número de processos MPI por (MTM+09).*

Aplicações	Processos MPI	$\theta_{temporal}$	θ_{volume}	$\theta_{espacial}$	(θ, ϵ)
BT-SP	16	1,000	0,000	1,000	(0,667, 1,000)
	64	1,000	0,000	1,000	(0,667, 1,000)
BT-LU	16	0,000	0,000	0,000	(0,000, 0,667)
	64	0,072	0,000	0,000	(0,024, 0,737)
LU-SP	16	0,000	0,163	0,000	(0,054, 0,667)
	64	0,072	-0,308	0,000	(-0,079, 0,737)

Ao observar os dados da Tabela 5.2, podemos retirar algumas conclusões. As aplicações BT e SP são similares, se excluirmos a propriedade de volume. Isso significa que BT e SP se

comunicam durante períodos similares e com os mesmos vizinhos, porém possuem volumes de comunicação completamente diferentes. Isso confirma a afirmação de (BBB+91), onde BT e SP são semelhantes com exceção da proporção de comunicação e computação. Além da similaridade de BT e SP, também podemos concluir que LU é bem diferente das outras duas aplicações, mesmo resolvendo um problema semelhante, explicitando a diferença de abordagem da aplicação LU frente a BT e SP. Outro ponto a destacar é propriedade de volume entre as aplicações LU-SP, o que indica que elas tem alguma semelhança no volume de comunicação, apesar de pequena. O $\theta_{temporal} > 0$ indica que com 16 processos MPI a aplicação LU tem um volume de comunicação maior, porém com 64 processos MPI a aplicação SP tem um volume de comunicação maior (pois temos $\theta_{temporal} < 0$).

(MTM+09) também gerou os grafos de comunicação de cada problema testado, que são reproduzidos na Figura 5.1. Apesar de (MTM+09) gerar um grafo para cada tamanho de problema, reproduzimos aqui apenas o da classe A pois os outros grafos são semelhantes, e os dados da Tabela 5.1 corroboram isso. Analisando a Figura 5.1, vemos como o BT e SP realmente são semelhantes quanto ao padrão de suas comunicações, corroborando os dados da Tabela 5.2 também.



(a) BT **(b)** SP **(c)** LU

Figura 5.1: *Grafos de comunicação das aplicações NPB por (MTM+09).*

5.2 Considerações sobre o capítulo

Este capítulo faz uma análise das aplicações do *NAS Parallel Benchmarks*, o benchmark usado para a realização desse trabalho. Vimos que o NPB, apesar de um benchmark antigo, ainda faz bem seu objetivo de analisar novas arquiteturas paralelas, e isso é graças ao design do benchmark que permitiu a escalabilidade das aplicações escritas.

Mais importante, fizemos uma análise dos padrões de comunicação do NPB com o auxílio do trabalho (MTM+09), mostrando as semelhanças e diferenças em cada uma das aplicações testadas. Isso será de suma importância para analisarmos o que está acontecendo na hora da execução desses benchmarks.

Capítulo 6

Experimentos e Resultados

Neste capítulo faremos uma avaliação de nuvens computacionais usando o NPB, permitindo analisar seu desempenho em aplicações HPC.

Começaremos descrevendo os experimentos realizados e mostraremos seus resultados. Após isso, analisaremos os resultados usando o conhecimento adquirido no Capítulo 4 sobre a arquitetura de nuvem e aplicações HPC, assim como o conhecimento do NPB e seus padrões de comunicação no Capítulo 5.

No fim, queremos entender melhor os detalhes de implementação de uma nuvem comercial, como o Google Compute Engine, a partir da análise dos resultados obtidos.

6.1 Metodologia dos experimentos

Neste trabalho utilizamos as implementações do NPB providas pela NASA, versão 3.3.1[1]. Os *Makefiles* foram modificados de forma apropriada para cada experimento, e todas os benchmarks foram testados com a otimização `-O3` do GCC. Todas as máquinas virtuais e contêineres estão executando Ubuntu 14.04, GCC 4.8.4 e OpenMPI 1.6.5. Para mais detalhes das configurações utilizadas e para permitir reprodutibilidade dos resultados, os scripts usados nesse trabalho estão disponíveis sobre uma licença de código aberto (MIT) no GitHub[2].

Nós focamos na medição de três parâmetros para avaliar o comportamento durante a execução dos benchmarks: *tempo total de execução*, *volume total de comunicação* e *número de chamadas de comunicação*.

O *tempo total de execução* é o tempo medido pela própria aplicação, já que cada aplicação NPB reporta seu tempo de execução após o termino da execução.

O *volume total de comunicação* é medido executando uma instância da aplicação com 32 contêineres com uma única vCPU cada em nosso sistema NUMA, e executando o `sysstat`[3] em plano de fundo. Isso força toda comunicação feita usando primitivas MPI em trocar dados pela interface de rede, ao invés de realizar essa troca de informações usando algum mecanismo de *Inter-Process Communication* (IPC). Com o uso do `sysstat`, podemos medir a banda usada na interface de rede. Após processar os dados brutos, nós calculamos o total de volume de comunicação medindo a banda usada por cada um dos nós, considerando a interface de rede `eth0` (o tráfego gerado na interface `lo` não é considerada). Já que estamos considerando os dados transferidos no nó inteiro ao invés de medir diretamente na aplicação, esse método de medição não é preciso. Porém, a diferença entre cada aplicação é significativa, como pode ser visto na Tabela 6.1. Para nossa análise, é suficiente saber qual aplicação possui o

[1] http://www.nas.nasa.gov/publications/npb.html
[2] https://github.com/m45t3r/naspb-tests
[3] http://sebastien.godard.pagesperso-orange.fr/

maior volume de comunicação, não o valor exato de comunicação. Pelo mesmo motivo, nós mostramos apenas o valor nas versões SZ, já que é mais difícil medir a transferência de dados entre nós nas versões MZ (que dependem da comunicação entre processos via OpenMP).

O *número de chamadas de comunicação* é o número de vezes que funções MPI são chamadas em cada aplicação. Ela é medida usando o TAU[4], um *profiler* para aplicações paralelas. Compilando o NPB com o TAU, é possível observar o número de chamadas de qualquer função no programa, incluindo chamadas as funções de comunicação do MPI. Nós obtemos o número de chamadas de comunicação contando o número de chamadas das funções `MPI_Send` ou `MPI_Isend`. Foram contadas apenas as chamadas de funções de envio ponto a ponto (`MPI_Send`/`MPI_Isend`) pois temos um número similar de chamadas às funções de recebimento. Novamente, o importante aqui é saber qual aplicação usa mais chamadas de comunicação, não qual o número de chamadas exato usado em cada aplicação. Como vimos no Capítulo 3, as chamadas para funções de comunicação em grupo são pequenas comparadas as chamadas de comunicação ponto a ponto (FY02), por isso não são consideradas.

Tanto o *volume total de comunicação* quanto o *número de chamadas de comunicação* são constantes para determinada configuração de aplicação. Isso significa que, por exemplo, executar a aplicação LU-SZ com tamanho de classe D e 32 processadores lógicos sempre vai resultar em números similares de volume e chamadas de comunicação, independente da infraestrutura usada para executar a aplicação. Porém, o *tempo total de execução* varia dependendo da configuração usada, e é a partir dessa variação que tentamos entender o que ocorre em cada aplicação.

Na Tabela 6.1 apresentamos o *volume total de comunicação* e o *número de chamadas de comunicação* para todos as aplicações testadas. Estes números são constantes para determinada configuração, ou seja, lu.D.32 sempre terá 121GB de *volume total de comunicação* independente de ser executado com 2 MVs de 16 vCPUs ou 1 MV de 32 vCPUs, ou de ser executado no Google ou na Amazon.

Tabela 6.1: *Volume total de comunicação e número de chamadas de comunicação em cada aplicação usada.*

	Volume Total de Comunicação (GB)	Número de Chamadas de Comunicação
lu.D.32	121,73	12.741.100
bt.D.25	217,06	188.400
bt.D.36	275,06	325.512
sp.D.25	407,60	375.900
sp.D.36	508,99	649.512

Nós avaliamos as aplicações de benchmark BT, LU e SP em diferentes configurações de máquinas virtuais nas infraestruturas de nuvem disponíveis. Escolhemos essas aplicações por serem as únicas com implementação MZ, além da SZ. Todas as aplicações testadas são escritas em Fortran-77. As aplicações SZ usam as primitivas do MPI para realizar comunicação entre processos (nesse caso, processos lógicos MPI). Já as aplicações MZ usam primitivas MPI para realizar a comunicação entre-nós (um processo MPI para cada máquina virtual) e OpenMP para realizar as comunicação entre processos, com o uso de memória compartilhada ao invés de troca de mensagens.

Google Compute Engine não oferece instâncias de máquinas virtuais específicas para uso em aplicações de alto desempenho, diferente do que ocorre na Amazon EC2[5] ou Microsoft

[4] https://www.cs.uoregon.edu/research/tau/home.php
[5] https://aws.amazon.com/en/ec2/instance-types/

Tabela 6.2: *Tamanhos de problema e parâmetros de entrada usado nos benchmarks, tamanho D.*

benchmark	Tamanho da Matriz	Número de Iterações	Tamanho do Passo (s)
LU	408 x 408 x 408	300	1,0
BT	408 x 408 x 408	250	0,00002
SP	408 x 408 x 408	500	0,0003

Azure[6]. No caso do Microsoft Azure, por exemplo, a rede em instâncias de alto desempenho usam 10 Gbps Ethernet ou InfiniBand, de forma explícita. O Google não oferece nenhuma informação adicional sobre a rede, e oferece somente uma recomendação de como configurar uma aplicação de alto desempenho no GCE[7]. Então nesses experimentos foram utilizados máquinas virtuais de uso geral do GCE, conhecidas como `n1`.

Nós modificamos o número de máquinas virtuais e o tamanho de cada um por experimento, sempre mantendo 32 vCPUs. Então, se usamos apenas uma única MV, temos uma MV de 32 vCPUs; se usamos duas MVs cada uma possui 16 vCPUs em cada, e assim por diante. O número de MVs seguido do tipo de MV usado no GCE estão listado a seguir:

- 1 × `n1-highcpu-32`: 32 vCPUs e 28,8GB de vRAM;
- 2 × `n1-highmem-16`: 16 vCPUs e 104GB de vRAM;
- 4 × `n1-highmem-8`: 8 vCPUs e 52GB de vRAM;
- 8 × `n1-highmem-4`: 4 vCPUs e 26 GB de vRAM;
- 16 × `n1-highmem-2`: 2 vCPUs e 13GB de vRAM.

A mudança do `n1-highcpu` para `n1-highmem` foi necessária pelo fato das aplicações NPB exigirem consideravelmente mais memória do nó *mestre*, comparado com outros nós. Então com instâncias de MVs menores alguns testes começaram a falhar por falta de memória, já que as instâncias `n1-highcpu` possuem uma menor quantidade de memória RAM comparada com as outras instâncias. Porém, não há nenhuma diferença em termos de processamento no GCE, e a redução de memória RAM vem com o simples objetivo de reduzir o preço da instância. Seria possível executar os testes que falharam usando as instâncias *standard*, que possuem mais memória que a *highcpu* porém menos que a *highmem*, como foi verificado depois, o que reduziria o preço de execução por aplicação.

De acordo com o painel de controle do GCE, todas as máquinas virtuais foram instanciadas em máquinas físicas com CPUs Intel® Haswell, porém de acordo com a documentação do GCE[8] elas poderiam ter sido instanciadas em máquinas físicas usando a arquitetura Sandy Bridge ou Ivy Bridge também. O preço em todos os casos é o mesmo.

Nosso aglomerado computacional é chamado de *revoada*, uma nuvem privada baseada no OpenStack Liberty. O sistema hospedeiro está executando o Ubuntu 14.04 (trusty), Linux kernel 3.19, e virtualizador KVM. Como o *revoada* é a nossa nuvem privada, nós tivemos acesso a mais informação que no *GCE*. Por exemplo, na *revoada* nós conhecemos o algoritmo de escalonamento usado para MVs e conhecemos qual máquina física cada MV está escalonada. Essa informação não está disponível para nós no *GCE*. Cada um dos 7 nós físicos na *revoada* estão conectados numa rede com topologia estrela, e possuem a seguinte configuração de hardware:

[6] https://azure.microsoft.com/en-us/pricing/details/cloud-services/
[7] https://cloud.google.com/solutions/architecture/highperformancecomputing
[8] https://cloud.google.com/compute/docs/machine-types#standard_machine_types

- 2 × Intel® Xeon® CPU E5645 @ 2.40GHz[9] (2 processadores físicos com 6 núcleos físicos e 12 *threads* cada, resultando em 12 núcleos físicos e 24 *threads* por nó físico);
- 8 × 4GB (32GB no total) de RAM;
- 1 Gbps Ethernet;
- 1 nó com 4x3TB HDD, 6 nós com 4x500GB HDD, via CephFS 0.94.6.

Finalmente, temos o nosso sistema NUMA, chamado de *revoada*. O sistema operacional é o Ubuntu 14.04 (trusty), executando o Linux kernel 3.13 e o LXC 1.0.8. Ele possui a seguinte configuração de hardware:

- 4 × AMD Opteron™ 6276 @ 1,4GHz[10] (16 processadores físicos, resultando em 64 núcleos físicos);
- 32GB de memória RAM conectado diretamente em cada processador, resultando em 128GB (4×32GB) de memória total no sistema;

O uso do LXC no *revoada* é justificado pelo fato do mesmo ser um sistema de gerenciamento de contêineres, usado para limitar o número de CPU em cada experimento e simular máquinas virtuais separadas. Por exemplo, para simular 4 MVs com 8 vCPUs cada, nós usamos 4 contêineres de 8 vCPUs cada, limitando o uso de CPU usando cgroups do Linux. Cgroups limitam os recursos forçando os processos executando dentro do contêiner para usar apenas os recursos alocados nele. Então, por exemplo, se alocarmos as CPUs 0-3 para a MV-0, o processo executando na MV-0 não pode usar a CPU 4.

O esquema de arquitetura de uma CPU física do hardware da *hydra*, incluindo a hierarquia da memória principal e caches individuais do processador, pode ser visto na Figura 6.1. Na máquina atual temos 4 CPUs iguais a essa conectadas por um barramento na placa-mãe.

Os experimentos foram executados no *GCE*, *hydra* e *revoada*. Os experimentos menores (classe A, B e C) foram repetidos pelo menos 20 vezes. Os experimentos maiores (classe D) foram repetidos 5 vezes, uma vez que cada execução de um aplicativo na classe D consome na ordem de uma hora. Por outro lado, para os experimentos menores, os dados de múltiplas execuções foram analisados estatisticamente. Na Figura 6.2, mostramos o gráfico da variação dos tempos de execução da aplicação LU classe C executado com 32 processos MPI, enquanto na Figura 6.3, mostramos o mesmo gráfico para o BT classe C executado com 25 processos MPI no ambiente *hydra*. Comportamentos semelhantes podem ser observados nas outras configurações, mostrando que os experimentos são estáveis.

Neste capítulo, a discussão limita-se a análise do desempenho das aplicações na classe D. No Apêndice A os dados para execução das demais classes são apresentados.

Nos gráficos que serão apresentados nas próximas seções, o eixo vertical é o tempo de execução em segundos (s), enquanto no eixo horizontal temos cada uma das arquiteturas testadas. *GCE* representa o Google Compute Engine, *hydra* representa nossa arquitetura NUMA, *revoada* representa nossa nuvem privada OpenStack; C representa o número de vCPUs em cada MV, enquanto V representa o número de MVs usados em cada experimentos. Então, *GCE-2Cx16V* representa as execuções no Google Compute Engine, usando 16 MVs de 2 vCPUs cada, GCE-4Cx8V usa 4 vCPUs e 8 MVs e assim vai. Note que, para qualquer $xV \times yC$, sempre temos $x \times y = 32$ processadores.

[9] http://ark.intel.com/products/48768/Intel-Xeon-Processor-E5645-12M-Cache-2_40-GHz-5_86-GTs-Intel-QPI

[10] http://www.cpu-world.com/CPUs/Bulldozer/AMD-Opteron%206276.html

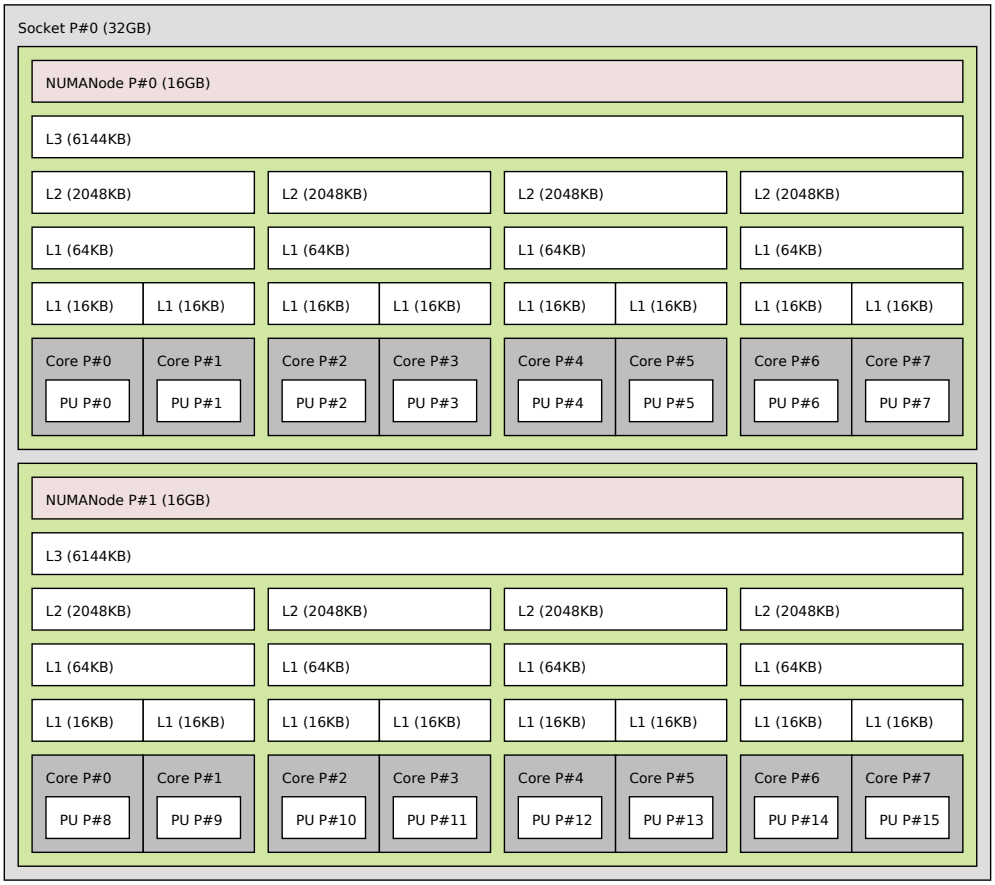


Figura 6.1: *Esquema de arquitetura de hardware da* hydra*, nosso sistema NUMA.*

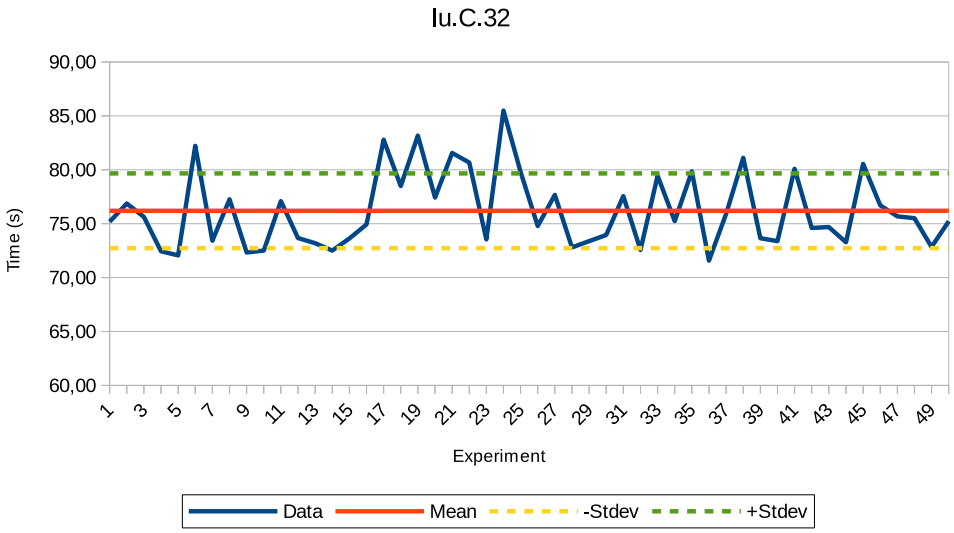


Figura 6.2: *Variação dos tempos de execução durante 50 repetições da aplicação LU-C-32 no ambiente* hydra.

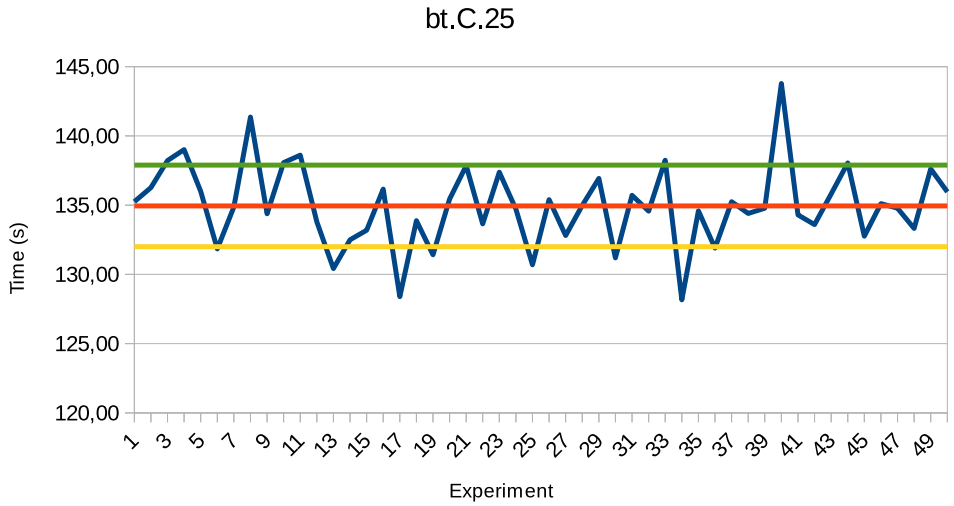


Figura 6.3: *Variação dos tempos de execução durante 50 repetições da aplicação BT-C-25 no ambiente* hydra.

Com o aumento do número de MVs (e a consequente redução do número de vCPUs), temos um aumento da comunicação entre-nós e diminuição da comunicação intra-nós. Podemos ver isso na Figura 6.4, onde temos o *volume de comunicação* em cada uma das configurações de MVs no experimento LU-D-32, executado no ambiente *hydra*, enquanto o mesmo tipo de gráfico nos outros experimentos podem ser encontrados no Apêndice A. Nesse gráfico, temos a diminuição do volume de comunicação com a diminuição do número de MVs, o que reflete a diminuição da comunicação entre-nós. Note que a diminuição do volume de comunicação não é linear, pois as comunicações refletem os padrões apresentados na Figura 5.1 do Capítulo 5, e a alteração no número de nós (nesse caso, MVs) altera a distribuição desse padrão de comunicação no espaço.

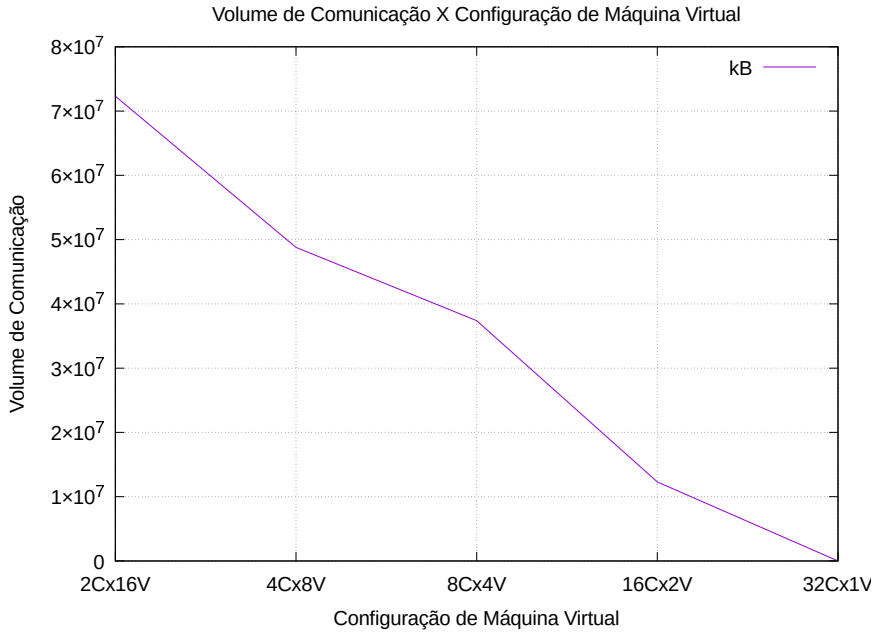


Figura 6.4: *Volume de Comunicação X Configuração de Máquina Virtual no experimento LU-D-32 executando na* hydra.

Os benchmarks BT-SZ e SP-SZ exigem número quadrático de processadores MPI. Por esse motivo, não foi possível executar esses benchmarks com exatamente 32 processadores MPI, e foram usados 25 e 36.

Nós gostaríamos de ressaltar que não estamos comparando a diferença entre tempos de execução em diferentes infraestruturas. Nosso atual objetivo é analisar a diferença de comportamento entre as infraestruturas.

6.2 Resultados

6.2.1 LU-SZ

A Figura 6.5 mostra os resultados da aplicação LU-SZ classe D, que explora o paralelismo intra-nós usando MPI. Em uma primeira observação, se destaca que os tempos de execução têm pouca variação, com exceção do GCE-32Cx1V e o revoada-16Cx2V, ambos no tamanho D, que são o menor número possível de MVs em cada uma das respectivas infraestruturas. Nossa hipótese é que em ambos esses casos, LU-SZ é afetado pela tecnologia Intel® Hyper-Threading, então em ambos os casos não existem núcleos físicos suficientes para o número de vCPUs requisitados pela máquina virtual. Note que o mesmo comportamento não ocorre na *hydra*, uma infraestrutura onde todos os núcleos são físicos (não existe Hyper-Threading).

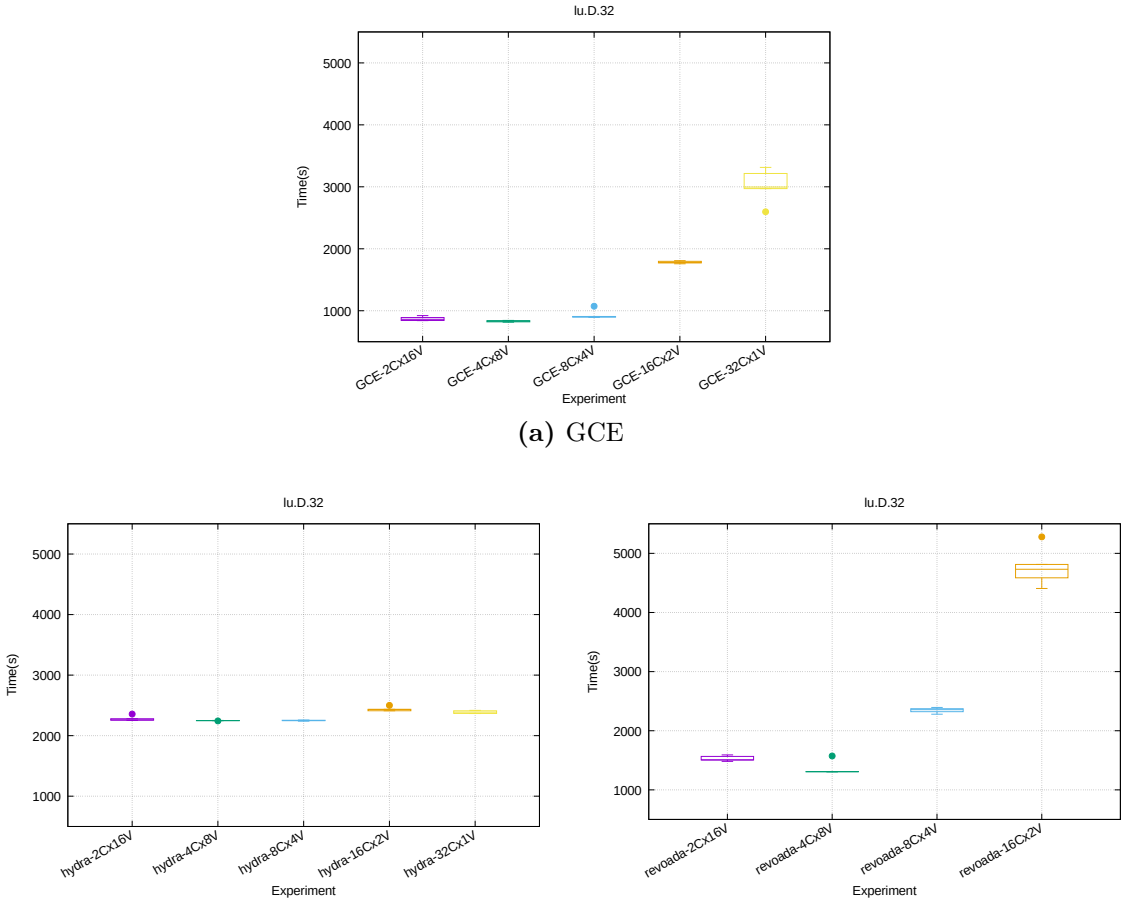
Como não temos controle da alocação de MVs no *GCE*, nossas MVs podem ter sido instanciadas em máquinas físicas ocupadas, ou as máquinas físicas podem estar sendo super provisionadas. Essa hipótese é suportada pelo fato que, em nossos testes, todas as máquinas virtuais foram alocadas em máquinas físicas com CPUs Intel® Haswell. Enquanto não temos o modelo exato usado pelo GCE, o máximo número de núcleos físicos disponíveis em qualquer CPU Haswell é 18[11]. Mesmo considerando que o GCE pode ter um nó com múltiplos nós físicos (permitindo que um único nó tenha mais que 18 CPUs, de forma semelhante o que ocorre com a *hydra*), ainda assim é mais difícil alocar uma MV grande com 32 vCPUs, comparado com múltiplas MVs menores.

Olhando com mais detalhes a *hydra* (Figura 6.5b) nós temos uma infraestrutura com baixos custos de comunicação, já que não existe comunicação ocorrendo na rede física. Nossa análise dos resultados na *hydra* nos permite concluir que em uma infraestrutura com baixos custos de comunicação, não existe diferença significativa no desempenho (a maior diferença entre médias foi 7,0 segundos para a classe C, 189,8 segundos para a classe D), pelo menos nesse experimento.

Na *revoada* (Figura 6.5c), nós temos 12 núcleos físicos e 24 núcleos lógicos em cada um dos nossos 7 nós, e existe um custo não desprezível de comunicação entre MVs em diferentes máquinas físicas (pois temos o uso da rede física para essa comunicação). O escalonador padrão do OpenStack (usado no *revoada*) é baseado no algoritmo *round-robin*[12] para alocação de MVs, preferindo alocar novas MVs na próxima máquina física ao invés da última. Então as configurações com mais MVs forçam o OpenStack a alocar mais MVs em uma mesma máquina física, reduzindo os custos de comunicação e, consequentemente, reduzindo o tempo total de execução. Esse é o caso tanto para o revoada-2Cx16V e revoada-4Cx8V.

Finalmente, avaliamos os resultados do *GCE* (Figura 6.5a). Comparando GCE-16Cx2V com outros resultados no *GCE* permite inferirmos que ambas as MVs estão rodando em diferentes máquinas físicas, considerando que o GCE-32Cx1V está executando em uma única máquina física. Essa hipótese reforça os resultados da *hydra*, que mostra uma diferença

[11] https://en.wikipedia.org/wiki/Haswell_(microarchitecture)
[12] http://docs.openstack.org/liberty/config-reference/content/section_compute-scheduler.html



(a) GCE

(b) hydra

(c) revoada

Figura 6.5: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-SZ, tamanho D com 32 processos MPI.*

insignificante no tempo total de execução quando os custos de comunicação são baixos.

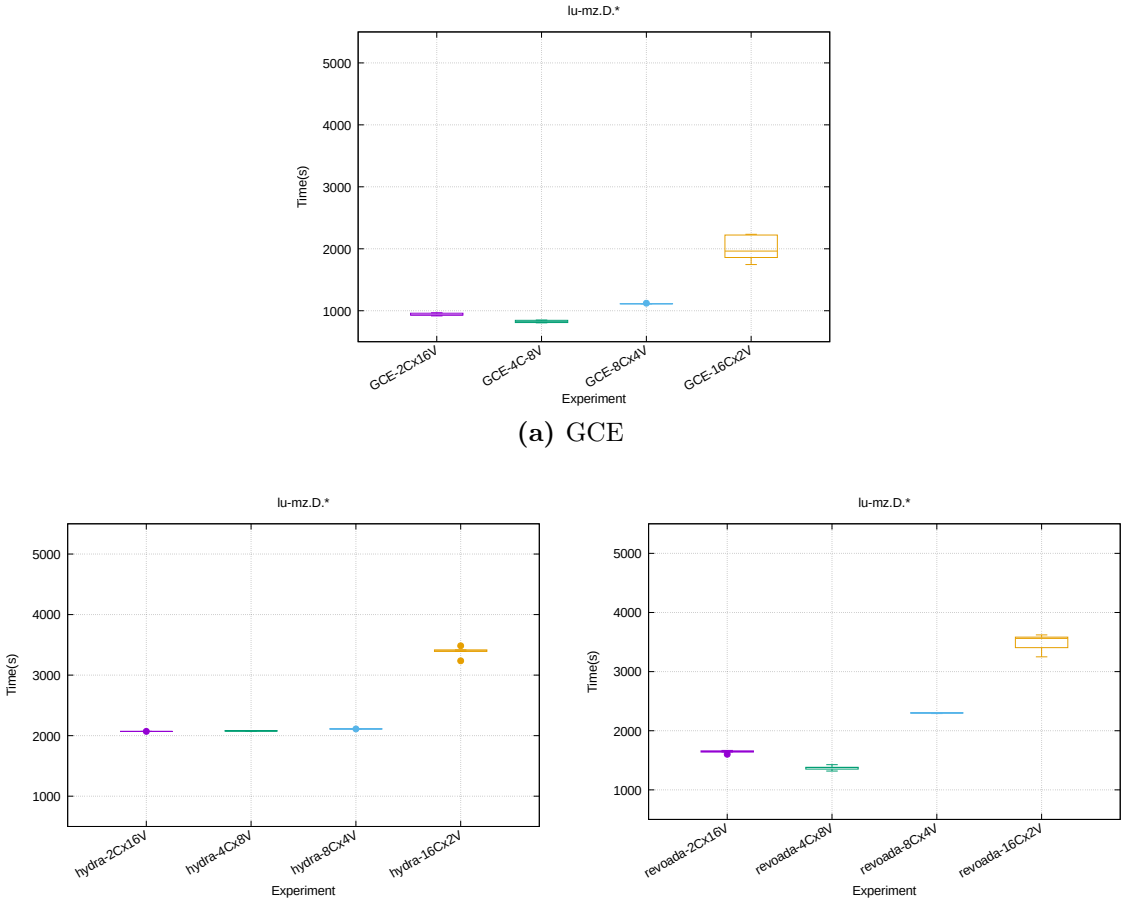
Nós não sabemos o número de núcleos físicos disponíveis em cada nó físico no *GCE*, ou mesmo se os data centers do Google são homogêneos. Porém, podemos concluir que o escalonamento de MVs de grão-fino, ou seja, com pequeno número de vCPUs, é mais eficiente nessa infraestrutura. Lembre-se que, atualmente, a maior MV oferecida no *GCE* tem 32 vCPUs, e o tamanho do grão é em relação com esse tamanho.

É possível que, eventualmente, o GCE-16Cx2V seja escalonado em duas máquinas físicas diferentes. Porém os autores não acreditam que isso ocorreu nos experimentos, baseado na evidência encontrada. Se isso tivesse ocorrido, existiria uma discrepância (por exemplo, um valor atípico no diagrama de caixa) nos nossos resultados.

6.2.2 LU-MZ

Figura 6.6 mostra os resultados da aplicação LU-MZ tamanho D. É esperado que nas aplicações MZ, pelo menos parcialmente, que os custos de comunicação sejam sobrepostos por processamento efetivo ao usar multi-thread (Val90), considerando que *threads* travam o acesso a região crítica, enquanto outras *threads* ocupam o processador, evoluindo o cálculo.

No caso em que existe máquinas virtuais de grão-grosso, ou seja, com um grande número de vCPUs e baixo número total de MVs, o número de canais de comunicação entre MVs é pequeno (canais de comunicação MPI). Então a competição entre *threads* pode ocorrer pelo



(a) GCE

(b) hydra

(c) revoada

Figura 6.6: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-MZ, tamanho D.*

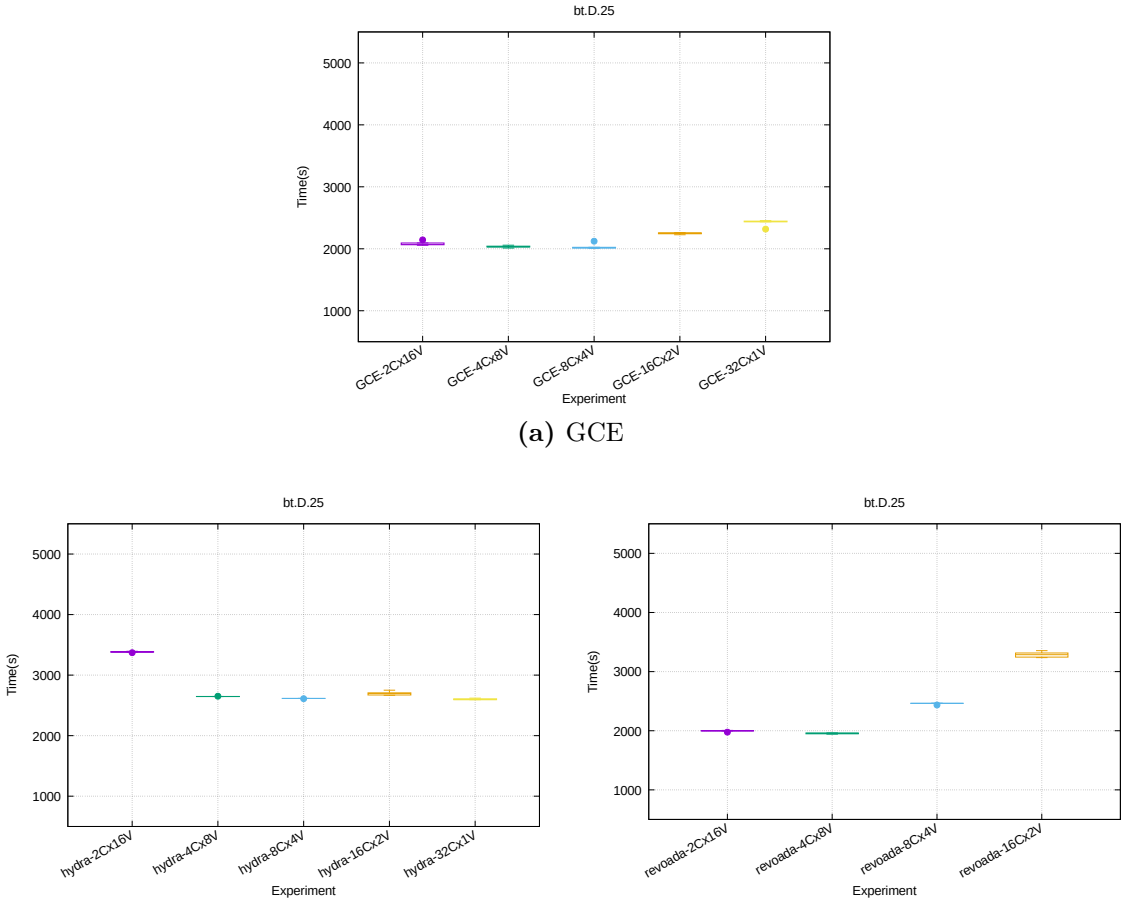
acesso ao canal de comunicação MPI. Nós observamos essa situação quando temos 2 MVs com 16 vCPUs cada: nesse caso, nós temos apenas um canal de comunicação MPI compartilhado por 16 *threads* em cada MV. Quando aumentamos a granularidade de MV, aumentando o número de MVs, nós temos mais canais de comunicação MPI para ser explorado por cada *thread*. Isso reduz a competição em cada canal MPI, consequentemente aumentando o desempenho total.

De acordo, os resultados que se destacam no LU-MZ são GCE-16Cx2V, hydra-16Cx2V e revoada-16Cx2V, primeiramente por causa da dispersão dos tempos de execução, e depois pelos tempos de execução significantemente maiores que o resto. A dispersão dos tempos de execução pode ser atribuído pelas *threads* tentando conseguir acesso à região crítica, enquanto o baixo desempenho foi explicado anteriormente.

Finalizando a análise do LU-MZ, olhando nos resultados do GCE, quando usamos MVs de grão-fino (MVs com 2, 4 e 8 vCPUs), várias MVs podem ser escalonadas na mesma máquina física, permitindo um desempenho relativamente estável. Então 4 MVs parecem ser o mínimo para evitar o congestionamento do canal MPI, pelo menos nessa configuração.

6.2.3 BT-SZ

No BT-SZ temos duas configurações de processos MPI, 25 (Figura 6.7) e 36 (Figura 6.8). Na *hydra* com 25 processos MPI (Figura 6.7b) o resultado que se destaca é o hydra-2Cx16V,



(a) GCE

(b) hydra

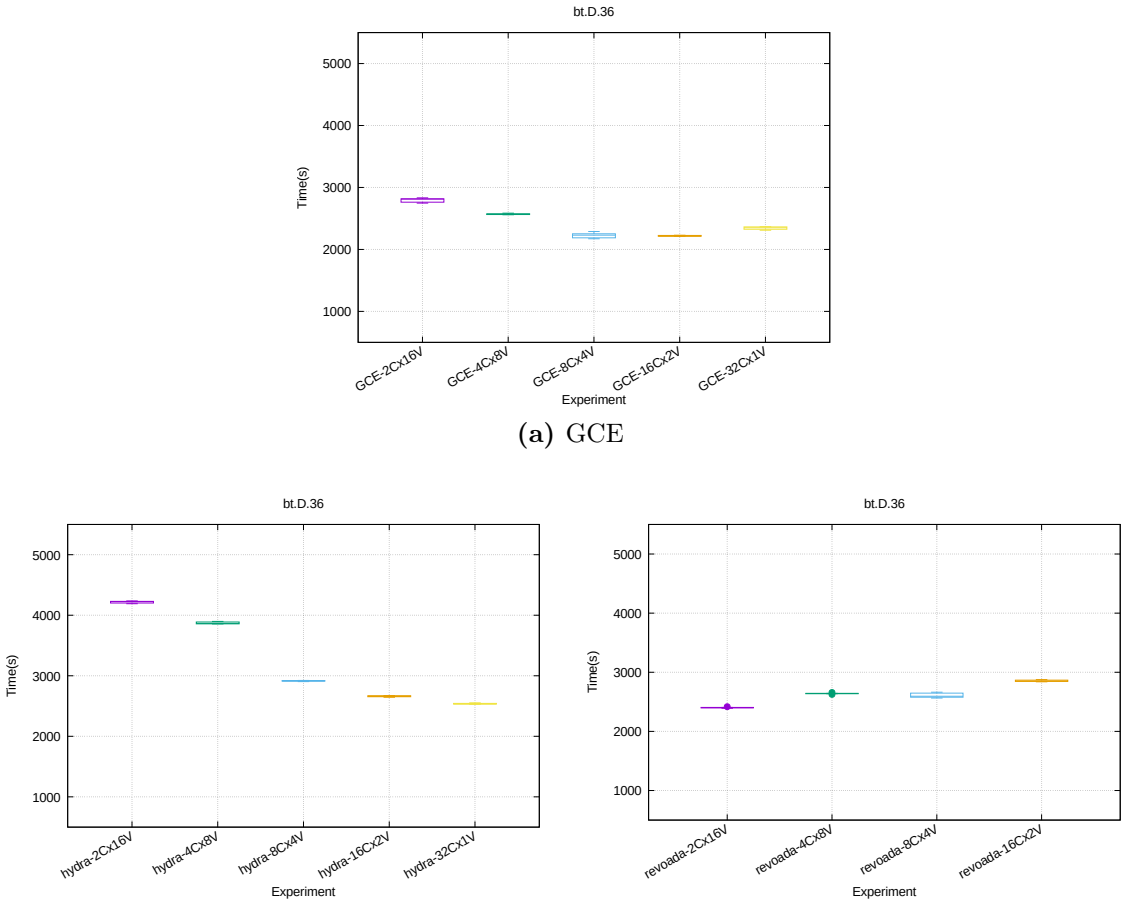
(c) revoada

Figura 6.7: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho D com 25 processos MPI.*

onde o tempo total de execução é maior que nos outros casos. Isso pode ser graças aos diversos contêineres se comunicando pela rede local, pois como podemos ver no gráfico da Figura A.1a, o maior volume de comunicação ocorre justamente nesse caso. Isso se reflete de forma ainda maior no caso de 36 processos MPI: o volume de comunicação cresce ainda mais (Figura A.1b), o suficiente para esse resultado se repetir também no caso hydra-4Cx8V além do hydra-2Cx16V.

Na *revoada* (Figura 6.7c), o resultado é semelhante ao que vimos na LU-SZ, ou seja, o impacto do Hyper-Threading afetou o desempenho nos casos hydra-16Cx2V e, de maneira menor, hydra-8Cx4V. Interessante notar que no caso com 36 processos MPI, o tempo de execução revoada-16CxV diminui de forma considerável comparando com 25 processos MPI. Provavelmente esse benchmark apresenta características que o fazem aproveitar a tecnologia Hyper-Threading de maneira eficiente, algo que não foi observado em outros testes.

No *GCE* (Figura 6.7a), também temos um pequeno impacto do Hyper-Threading com MVs de grão-grosso (GCE-32Cx1V), pelo menos no caso com 25 processos MPI. Com 36 processos MPI, provavelmente pelo aumento do volume de comunicação (Figura A.1b), temos que as MVs de grão-fino são levemente penalizadas comparando com as MVs de grão-grosso. Diferente do que aconteceu em outros testes, a MV de grão-grosso (GCE-32Cx1V) não teve um grande impacto do desempenho por causa do Hyper-Threading, como ocorreu em outros testes, o que confirma a hipótese que tivemos na *revoada* que esse benchmark em específico aproveita essa tecnologia de maneira eficiente.

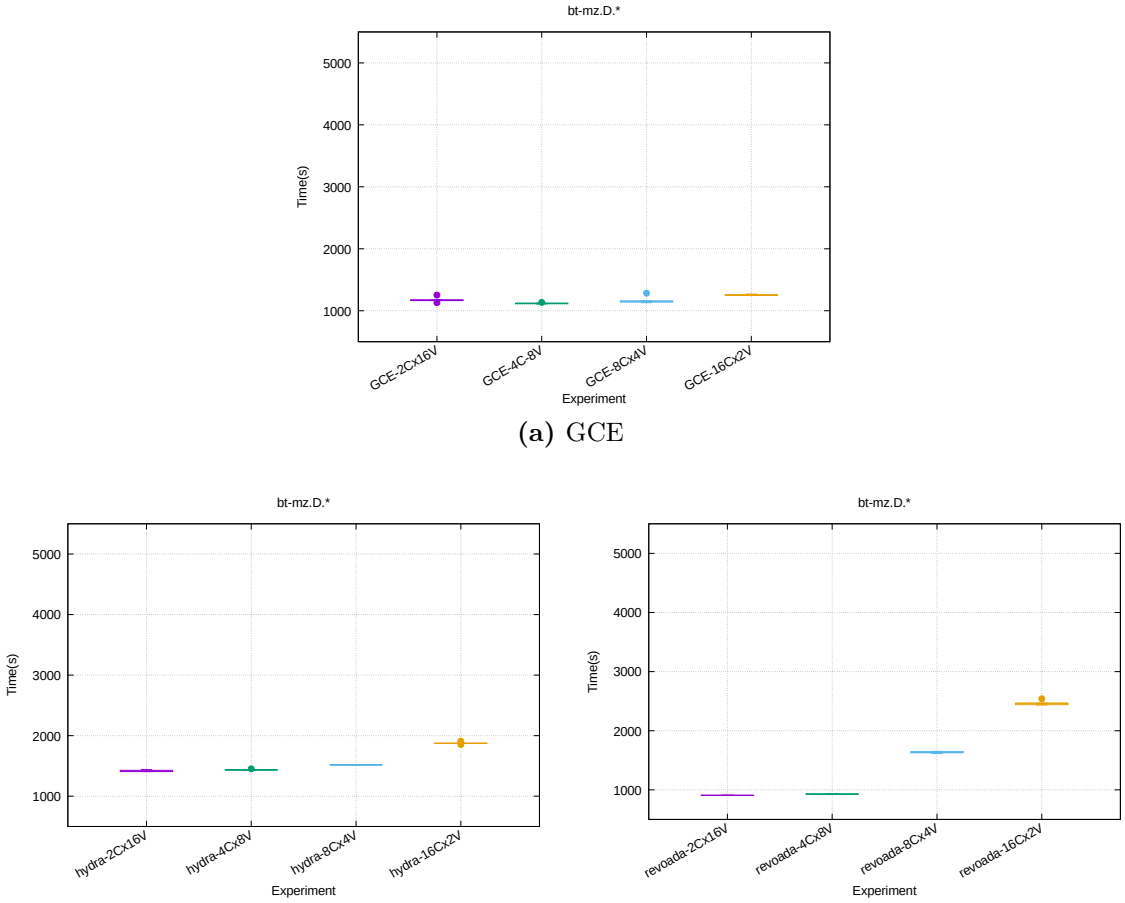


(a) GCE

(b) hydra

(c) revoada

Figura 6.8: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho D com 36 processos MPI.*



(a) GCE

(b) hydra

(c) revoada

Figura 6.9: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-MZ, tamanho D.*

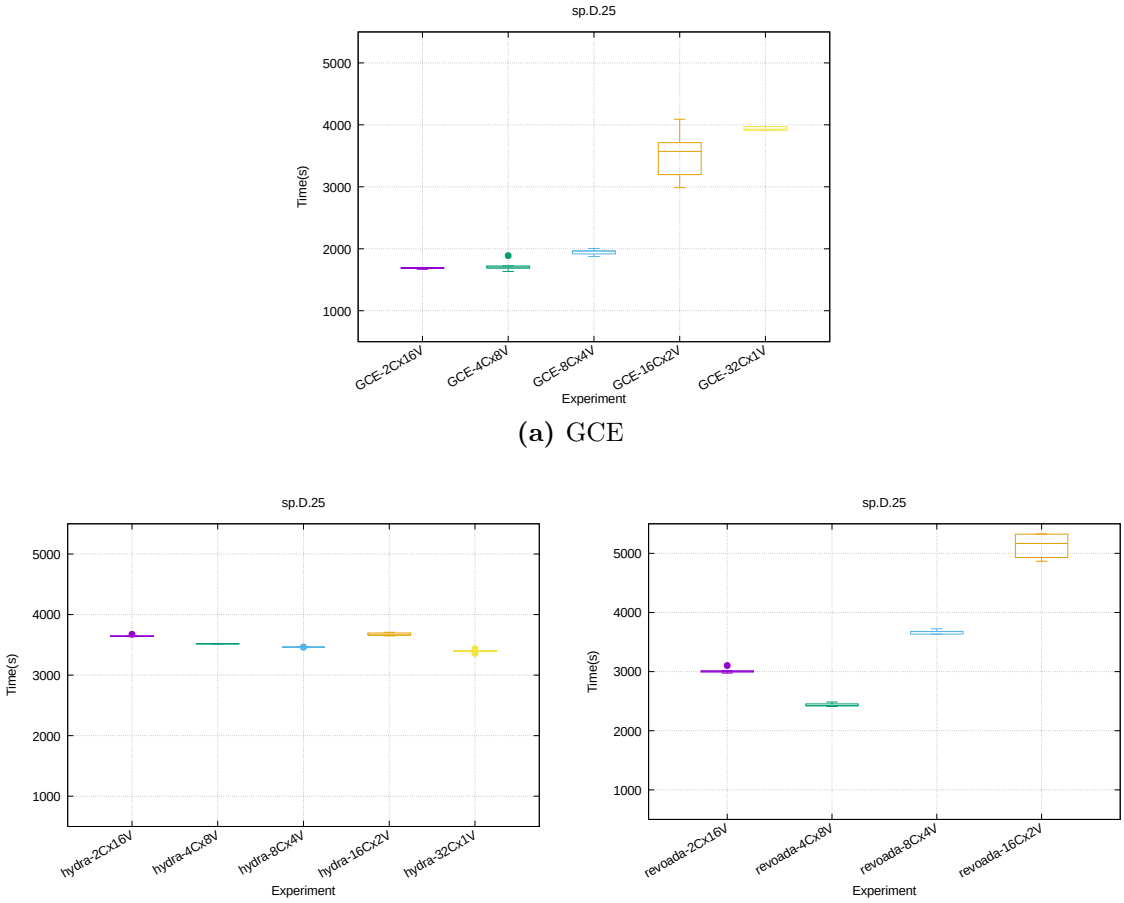
6.2.4 BT-MZ

Figura 6.9 mostra os resultados da aplicação BT-MZ (MPI+OpenMP). Com a redução da latência de comunicação (com o uso do OpenMP no lugar do MPI para as comunicações intra-nós), observamos algo similar ao que aconteceu no LU-SZ, pelo menos na infraestrutura *hydra*: temos poucos canais de comunicação MPI para a comunicação entre-nós, o que causa uma queda de desempenho nas configurações de grão mais grosso. Isso não foi observado nas outras duas infraestruturas, o que sugere que o *GCE* tem um melhor desempenho de rede que a *revoada* (a *hydra* de fato têm um desempenho superior, graças ao fato de todas os contêineres estarem no mesmo nó físico).

Uma outra explicação para o menor desempenho da *revoada* é o Hyper-Threading, considerando que não existe capacidade suficiente nos nós físicos para instanciar MVs com 16 vCPUs.

6.2.5 SP-SZ

Figura 6.10 mostra os resultados da aplicação SP-SZ (MPI) com 25 processos MPI, Figura 6.11 mostra os resultados com 36 processos MPI. SP-SZ tem um tempo total de execução maior, de forma geral, comparado com o LU. O *total volume de comunicação*, como mostrado na Tabela 6.1, é superior também, porém temos um menor *número de chamadas de comunicação*. No caso com 25 processos MPI, assim como o LU-SZ, nós temos uma baixa



(a) GCE

(b) hydra

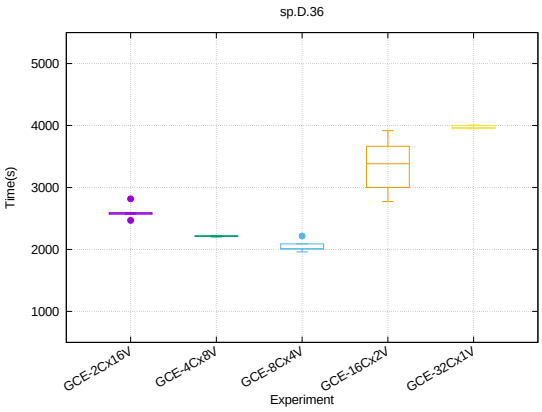
(c) revoada

Figura 6.10: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho D com 25 processos MPI.*

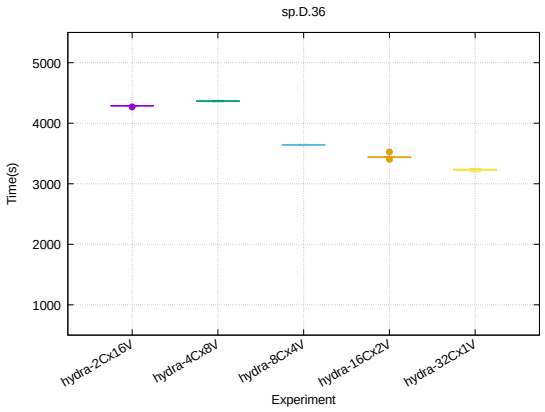
variação dos tempos de execução na infraestrutura *hydra* (a maior diferença entre médias é 212,15 segundos). Porém com 36 processos MPI temos uma variação bem mais significativa, 1136,32 segundos, com os menores tempos de execução nas configurações de grão mais grosso, ou seja, quando temos menor comunicação intra-nós.

Todavia, olhando nos resultados do *revoada* com 36 processos MPI, vemos um impacto mais significativo do Hyper-Threading que o impacto da comunicação intra-nós causa no tempo total de execução. Isso provavelmente causou, tanto no *revoada* quanto no *GCE*, um tempo de execução significantemente maior nas MVs com grão mais grosso, mesmo considerando que tivemos o resultado contrário na *hydra*.

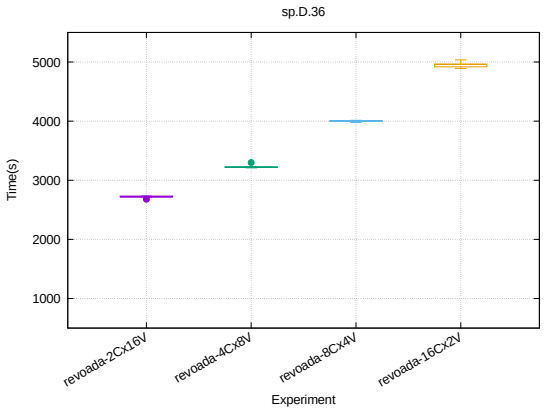
Destaca-se o "largo" diagrama de caixa no caso GCE-16Cx2V, indicando uma variabilidade grande no tempo de execução entre as repetições. Isso pode ter sido causado por uma migração de MV durante os experimentos, pois o fenômeno foi observado tanto com 25 processos MPI quanto 36. O mesmo comportamento não é observado no *revoada*, reforçando a nossa hipótese já que o OpenStack não faz balanceamento de carga por padrão. No *GCE* com MVs de grão mais fino, isso é, com 4 ou mais MVs, é possível observar o escalonamento de MVs na mesma máquina física ou máquinas físicas próximas.



(a) GCE

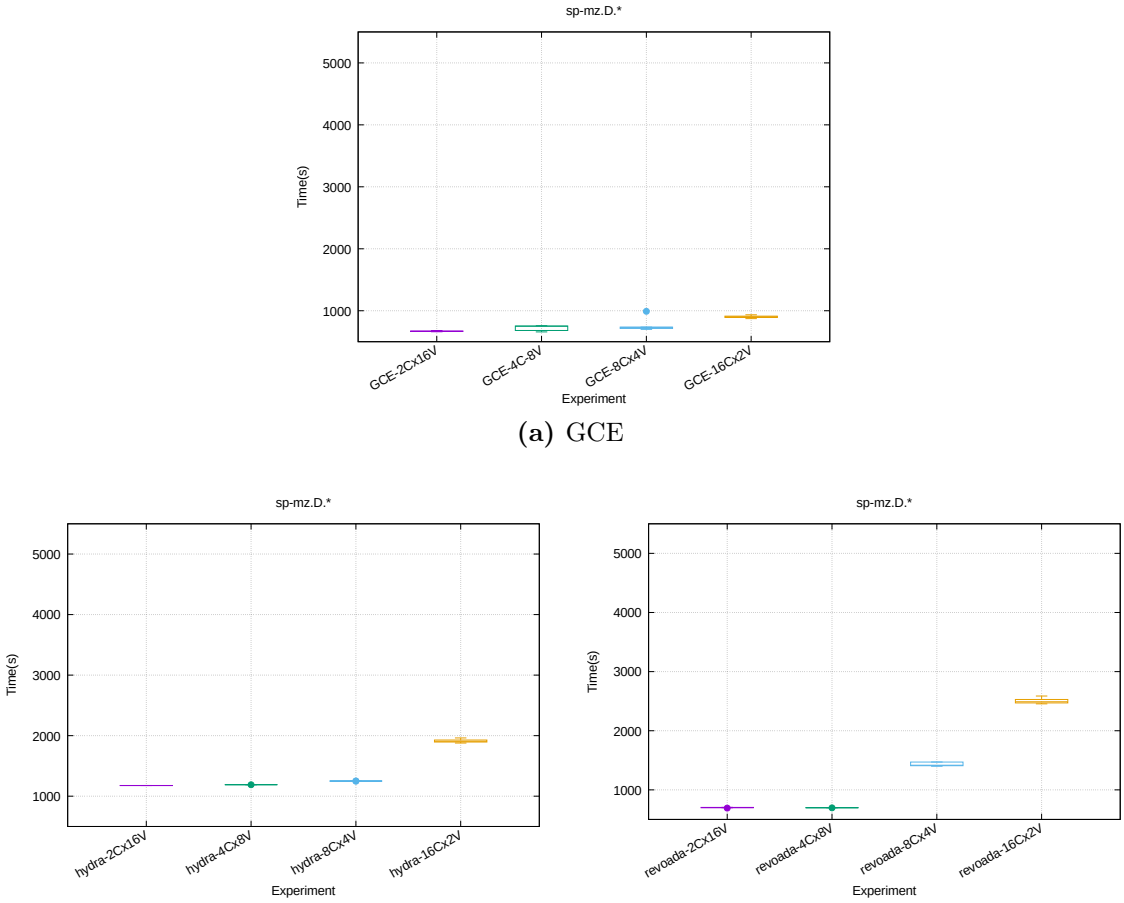


(b) hydra



(c) revoada

Figura 6.11: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho D com 36 processos MPI.*



(a) GCE

(b) hydra

(c) revoada

Figura 6.12: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-MZ, tamanho D.*

6.2.6 SP-MZ

Figura 6.12 mostra os resultados da aplicação SP-MZ (MPI+OpenMP). Os maiores tempos de execução são observados com os menores números de MVs, similar ao LU-MZ. Também similar ao LU-MZ, nós encontramos que custos de comunicação não afetam o desempenho em configurações com menores números de vCPUs.

É interessante notar o valor atípico no diagrama de caixa no GCE-8Cx4V: é possível que uma execução dessa aplicação tenha sido afetada pelos efeitos de migração de MV ou por outra(s) MV(s) executando na mesma máquina física durante a execução. Uma situação similar pode ter acontecido no GCE-4Cx8V no SP-SZ.

6.3 Redução do custo operacional

Vimos na seção anterior reduções significativas no tempo total de execução em alguns casos. O Google Compute Engine é uma nuvem cuja sua utilização é cobrada por minuto, após os primeiros 10 minutos (que são cobrados de forma fixa)[13]. Isso é diferente da Amazon EC2 que é cobrado por hora[14], então, ao optar pelo uso do *GCE*, é de se esperar que o usuário deseje minimizar seus gastos considerando margens menores de tempo. Outro ponto

[13] https://cloud.google.com/compute/pricing
[14] https://aws.amazon.com/pt/ec2/pricing/

importante é que no *GCE*, para um número fixo de vCPUs, independente do número de MVs usada pela aplicação, o preço por minuto é o mesmo. Portanto, essa redução no tempo total de execução representa uma economia significativa para o usuário.

Por exemplo, nossos experimentos mostram que no caso sp.D.25, diminuímos o tempo de execução (médio) em 2253,63 segundos (ou uma redução de 57,5% do tempo de execução), quando comparado a configuração de MVs mais otimizada (GCE-2Cx16V) comparada com a pior configuração (GCE-32Cx1V). Em termos de economia, isso representa uma redução de aproximadamente 38 minutos, ou uma economia de US$1,31 por execução[15]. Considerando uma aplicação que demore 100 horas para executar no GCE-32Cx1V, uma redução de tempo semelhante economizaria US$92,00.

Vemos outras reduções significativas no tempo total de execução no casos lu.D.32 (redução de 72,5%, comparando GCE-32Cx1V e GCE-4Cx8V), lu-mz.D.* (redução de 58,8%, comparando GCE-4Cx8V e GCE-16Cx2V), bt.D.36 (redução de 20,6%, comparando GCE-2Cx16V e GCE-16Cx2V).

Por isso, é importante descobrir a configuração correta de máquinas virtuais antes de executar um experimento na nuvem. Uma das maneiras de se descobrir essa configuração correta é executando instâncias menores do problema, de preferência que demorem cerca de 10 minutos de execução (valor mínimo cobrado no GCE). 32 vCPUs executando durante 10 minutos resultam em US$0,27[16]. Ou seja, é possível analisar todas as configurações possíveis de MVs com 32 vCPUs com apenas US$1,62, o que em uma execução que demore horas ou dias, pode resultar em uma economia considerável.

6.4 Considerações sobre o capítulo

Foi apresentada uma comparação de desempenho da execução de aplicações do NPB em duas arquiteturas conhecidas (*revoada* e *hydra*) e o *GCE*. Pode-se observar o impacto de diferentes variáveis nas aplicações testadas, e sua diferença no desempenho final.

Verificamos o impacto da tecnologia de *threads* virtuais, o *Hyper-Threading*, nas diferentes aplicações e também o impacto da comunicação intra-nós frente a comunicação entre-nós. Também observamos o impacto do uso de técnicas de memória compartilhada (usando o OpenMP) no lugar de troca de mensagens (MPI) para as comunicações intra-nós. Considerando o *GCE*, instanciar uma grande MV, como o GCE-32Cx1V, não parece ser uma boa opção para aplicações de alto desempenho, mesmo quando a aplicação executa de forma eficiente usando Hyper-Threading, como é o caso do BT-SZ. O GCE-32Cx1V nunca foi a configuração mais rápida em nenhuma situação.

Com os resultados apresentados, conclui-se que os data centers do Google devem utilizar máquinas físicas com um número relativamente pequeno de CPUs, pelo menos comparado com a nossa infraestrutura NUMA, a *revoada*. Isso porque ao se utilizar MVs de grão-grosso, quase sempre tivemos um problema com escalonamento de MVs ou a tecnologia Hyper-Threading, algo que não aconteceu na *hydra*, um sistema NUMA com grande número de CPUs e sem a tecnologia Hyper-Threading (por usar processadores AMD).

Mesmo GCE-16Cx2V parece encontrar esses dois problemas, em menor caso, mas ainda assim existe. Porém em pelo menos um caso GCE-16Cx2V mostrou ter um bom desempenho, que foi justamente no caso BT-SZ com 36 processos MPI, que como já foi comentado, aparenta ter um bom desempenho com Hyper-Threading. Nosso trabalho confirma em parte o que foi encontrado em (LORZ13), onde o Hyper-Threading no *GCE* parece impactar o

[15] https://cloud.google.com/products/calculator/#id=fcac81db-0018-41f8-ac3b-b41619157ef8
[16] https://cloud.google.com/products/calculator/#id=d59b9d25-2633-4c00-8deb-be33a72091e2

desempenho de aplicações de alto desempenho.

Porém, simplesmente usar as configurações de MV de grão mais fino não é necessariamente a melhor maneira de executar aplicações de alto desempenho. Em alguns casos, como o LU-MZ no GCE-2Cx16V, nós tivemos um desempenho menor que o comparado com o GCE-4Cx8V, o que provavelmente ocorreu por causa da latência na comunicação entre-nós, por existirem muitos processos fazendo comunicação entre-nós. Os melhores resultados no GCE, mesmo considerando os experimentos com classes menores do Apêndice A, geralmente apontam para um equilíbrio entre o número de vCPUs e o número de MVs, ou seja, um equilíbrio da comunicação entre-nós e intra-nós.

Finalmente, vimos o impacto econômico de se utilizar uma configuração apropriada de MVs, graças ao fato que reduzir o tempo total de execução pode representar uma economia significativa, principalmente em uma nuvem como o Google Compute Engine, que realiza cobrança de instâncias por minuto.

Capítulo 7

Conclusões e trabalhos futuros

Nesta seção serão apresentados as conclusões obtidas a partir dos resultados e comentar sobre possíveis trabalhos futuros.

7.1 Conclusões

Com o crescente uso de tecnologias de computação em nuvem tanto em ambientes de TI quanto na comunidade acadêmica, é importante entender de forma mais profunda como um ambiente de nuvem pode impactar o desempenho de aplicações. Em especial, como a nuvem permite a computação como um serviço, otimizar a execução de uma aplicação na nuvem economiza dinheiro e recursos.

Neste trabalho analisamos a viabilidade de execução de aplicações de alto desempenho em IaaS públicas, comparando o desempenho de aplicações de alto custo computacional em infraestruturas com comportamento conhecido com a nuvem pública oferecida pelo Google. Nós também analisamos diferentes configurações de paralelismo com comunicação intra-nós e entre-nós. Nosso caso de estudo para esse trabalho foi o NPB, um benchmark bastante popular para a análise de desempenho de sistemas paralelos e de alto desempenho. Utilizamos com implementações deste benchmark implementadas puramente com o MPI (conhecidas como SZ) e implementações mistas onde as comunicações internas foram feitas utilizando OpenMP (comunicação intra-nós) e as comunicações externas foram feitas usando o MPI (comunicação entre-nós).

Nossos resultados mostram que o desempenho de uma aplicação de alto desempenho pode ser afetada, positivamente, com o uso de um número apropriado de vCPUs em cada MV usado durante a execução de uma aplicação. Nas aplicações LU e SP, foi possível verificar comportamento semelhante tanto na infraestrutura *GCE* quanto na *revoada* e *hydra*, pelo menos considerando MVs de grão-fino (número pequeno de vCPUs por MV). Nas máquinas virtuais de grão-grosso (número grande de vCPUs por MV) o desempenho foi penalizado, graças a inúmeros fatores, incluindo a falta de núcleos físicos de CPU (com o consequente uso da tecnologia Hyper-Threading) e o escalonamento de MV pelo provedor de nuvem. Já na aplicação BT, tivemos um efeito oposto, em especial quando o número de processos MPI era maior que o número de *threads* de execução disponíveis. Isso pode ter acontecido pelo fato dessa aplicação ter um bom aproveitamento da tecnologia Hyper-Threading, e a diminuição de latência ao seu utilizar uma MV de grão-grosso melhorou o resultado nesse caso específico.

O fato de algumas aplicações de alto desempenho terem um bom aproveitamento da tecnologia Hyper-Threading (como o BT) e outras não (como o LU e SP) não é incomum, como mostrado em (LAH+02). Isso aumenta a importância de se conhecer como a arquitetura

da nuvem funciona com mais detalhes no baixo nível: algumas aplicações se beneficiam de alguns tipos de característica mais do que outras.

Nesse trabalho, nossa análise focou no Google Compute Engine. Como vimos anteriormente, o GCE é uma nuvem interessante para realizar os experimentos considerando que sua utilização é cobrada por minuto. Como tivemos reduções significativas no tempo total de execução, isso gera uma economia interessante para o usuário que deseja executar aplicações de alto desempenho na nuvem. Considerando que essa economia vem apenas por se utilizar a configuração adequada de máquinas virtuais, sem necessitar mudanças no código-fonte, é interessante ao usuário de nuvem fazer uma avaliação semelhante das possíveis configurações de máquinas virtuais a serem utilizadas na nuvem pública.

Considerando o GCE, os melhores resultados geralmente foram obtidos com o GCE-4Cx8V ou o GCE-8Cx4V, e quando eles não foram o melhor resultado chegam perto. Acreditamos que o 8 vCPUs é o mínimo para evitar o problema de Hyper-Threading do GCE, enquanto até 8 MVs é o mínimo para se evitar um aumento de latência na comunicação entre-nós.

O autor gostaria de salientar que o estudo está atado a configuração atual do data center do Google, e pode mudar no futuro. Porém nossa análise pode ser aplicada mesmo que o Google faça uma atualização no seu data center, ou em outras nuvens públicas como o Amazon EC2 e o Microsoft Azure.

7.2 Trabalhos futuros

Gostaríamos de aplicar a mesma análise em outras nuvens públicas, como a Amazon EC2 e o Microsoft Azure, para compará-las com o Google Compute Engine.

Um outro trabalho futuro é expandir a suíte de benchmarks a serem utilizados, incluindo um teste de armazenamento. Nuvens computacionais oferecem diversos tipos de armazenamento a escolha do usuário, e diferentes combinações podem oferecer diferentes resultados. Um exemplo é o uso de SSDs contra HDs tradicionais, porém uma nuvem comercial como o Google oferece ainda mais opções, como o uso de memória RAM para armazenamento. Para isso, será necessário primeiro observar diferentes requerimentos para aplicações de alto desempenho, como latência ou banda.

Finalmente, a partir desse trabalho queremos expandir a análise para o impacto da comunicação via troca de mensagens e memória compartilhada num ambiente de nuvem computacional. Em ambientes tradicionais de computação de alto desempenho, o uso de memória compartilhada sempre foi mais interessante pois reduz a latência das comunicações. Porém esse trabalho mostra que, em um ambiente de nuvem, onde o usuário não tem controle total do ambiente, essa vantagem não é mais tão óbvia assim. Por isso queremos estudar mais a fundo o impacto da memória compartilhada em ambientes de HPC.

Apêndice A

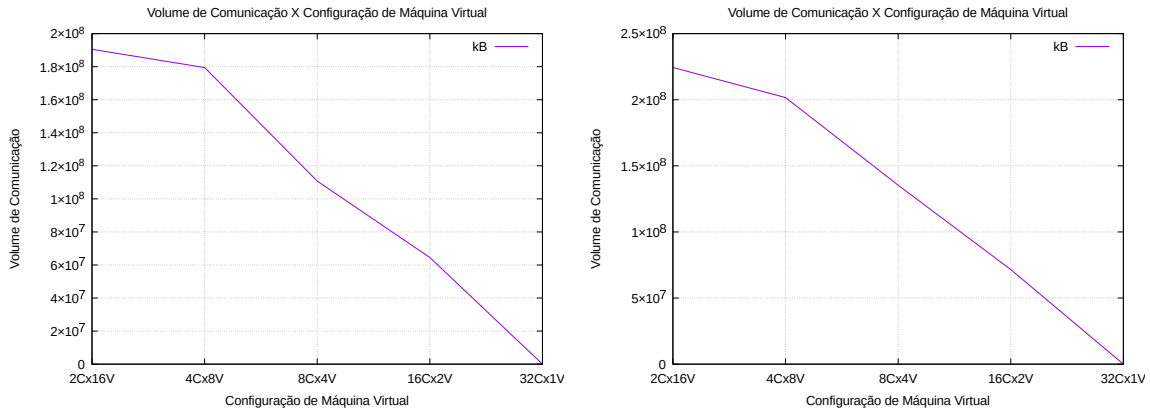
Resultados adicionais

Neste apêndice colocaremos resultados extras que não foram analisados de forma mais profunda no decorrer do texto, porém ainda podem ser interessantes para uma análise futura dos dados.

A.1 Volume de Comunicação X Configuração de Máquina Virtual

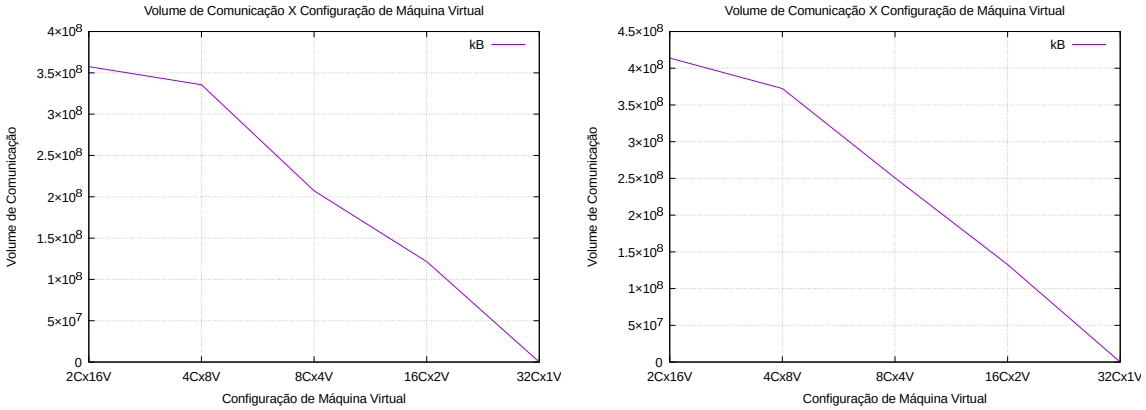
Nos gráficos desta seção temos o *volume de comunicação* em cada uma das configurações de MVs disponíveis nos experimentos realizados (LU, BT e SP), classe D. Estes experimentos foram realizadas na infraestrutura *hydra*.

É possível verificar a diminuição do volume de comunicação com a diminuição do número de MVs, o que reflete a diminuição da comunicação entre-nós. Lembrando que a diminuição do volume de comunicação não é linear, como se pode esperar, pois as comunicações refletem os padrões apresentados na Figura 5.1 do Capítulo 5.



(a) *BT-D-25* **(b)** *BT-D-36*

Figura A.1: *Volume de Comunicação X Comunicação de Máquina Virtual no experimento BT executando na* hydra.



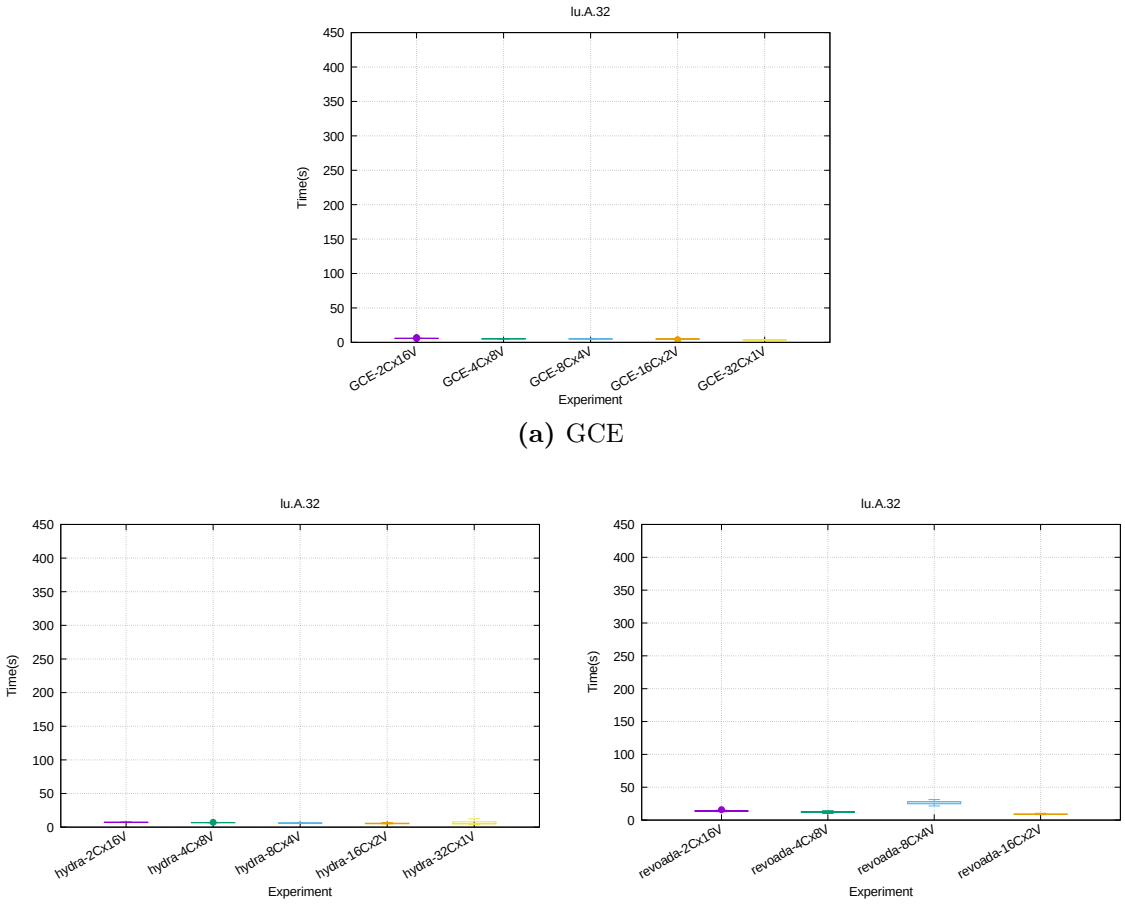
(a) *SP-D-25* **(b)** *SP-D-36*

Figura A.2: *Volume de Comunicação X Comunicação de Máquina Virtual no experimento SP executando na* hydra.

A.2 Experimentos classes A, B e C

Os gráficos desta seção refletem os experimentos realizados (LU, BT e SP) nas três arquiteturas (*GCE*, *hydra* e *revoada*) nas classes A, B e C.

Lembrando que nestes casos, os gráficos refletem os resultados após 20 execuções dos experimentos, contra 5 repetições nos experimentos da classe D. O motivo disso foi a demora de execução dos experimentos da classe D comparado com as classes A, B e C, conforme explicado no Capítulo 6.

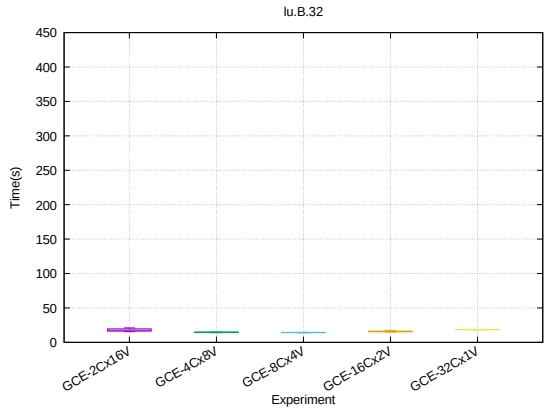


(a) GCE

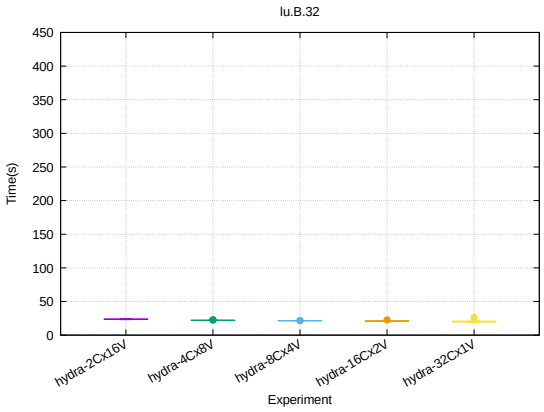
(b) hydra

(c) revoada

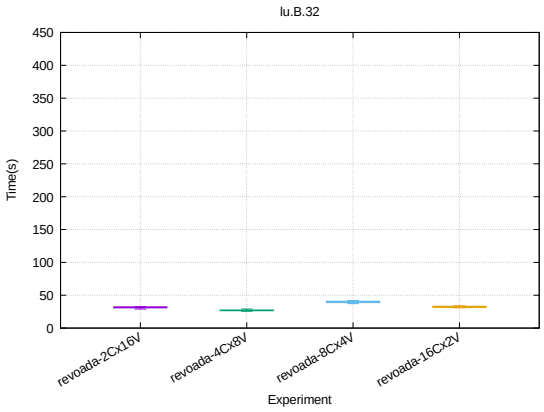
Figura A.3: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-SZ, tamanho A com 32 processos MPI.*



(a) GCE

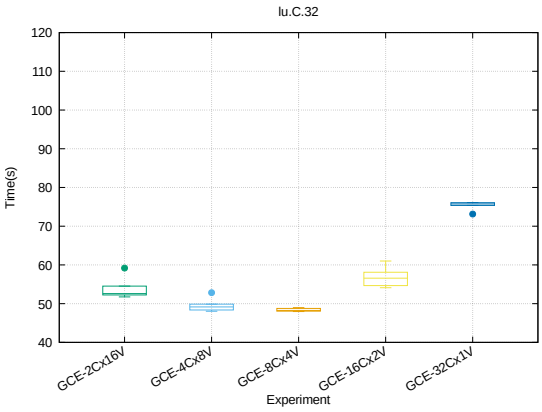


(b) hydra

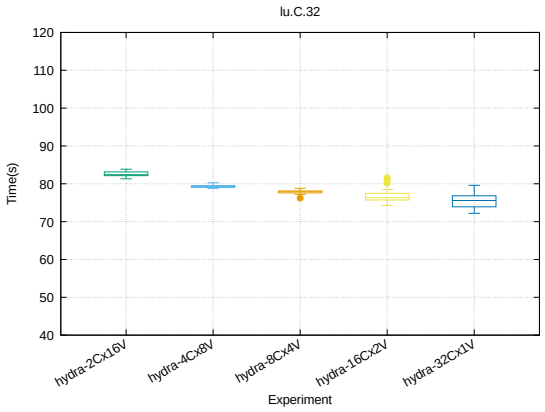


(c) revoada

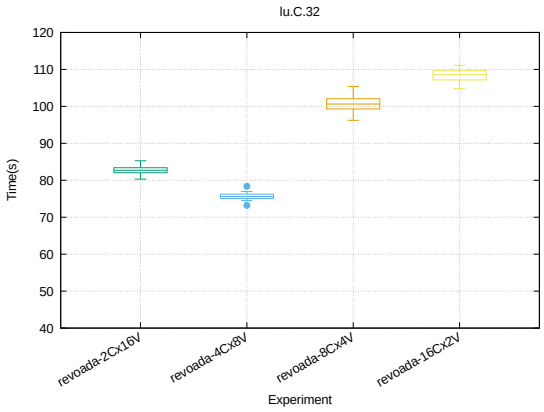
Figura A.4: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-SZ, tamanho B com 32 processos MPI.*



(a) GCE

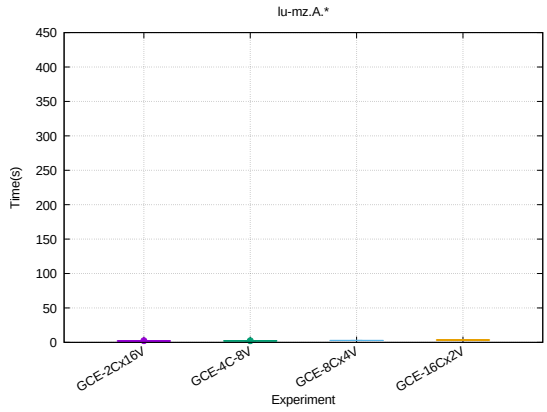


(b) hydra

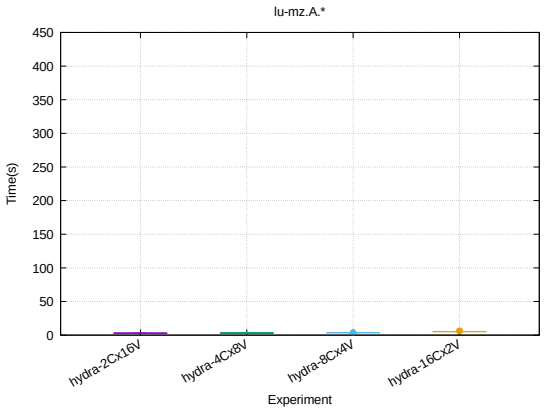


(c) revoada

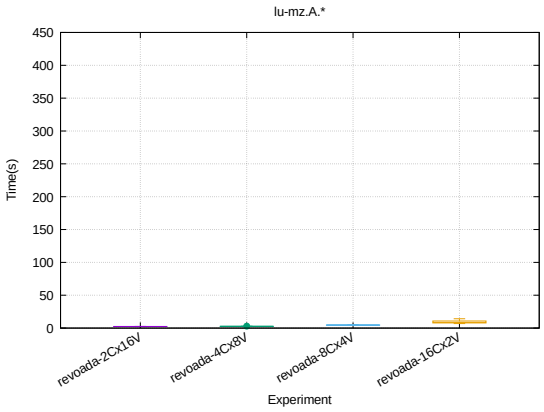
Figura A.5: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-SZ, tamanho C com 32 processos MPI.*



(a) GCE

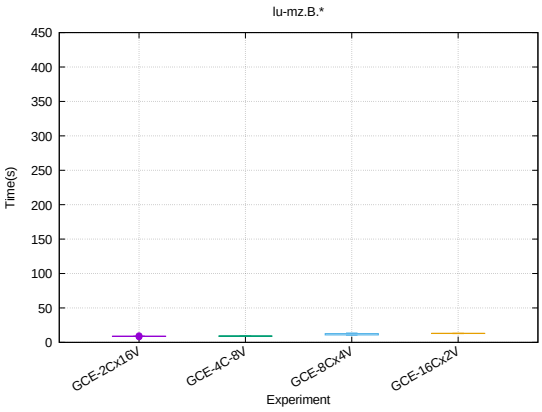


(b) hydra

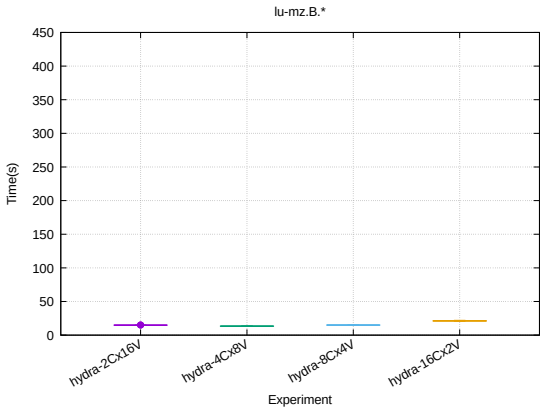


(c) revoada

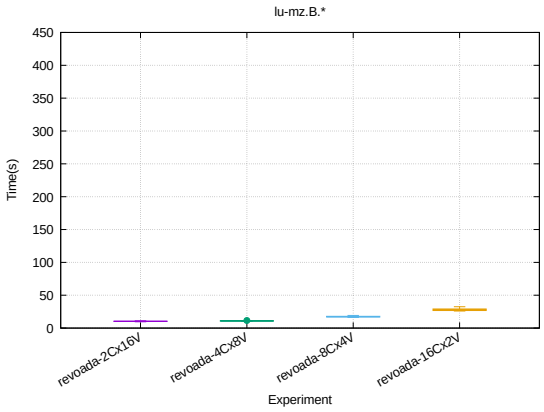
Figura A.6: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-MZ, tamanho A.*



(a) GCE

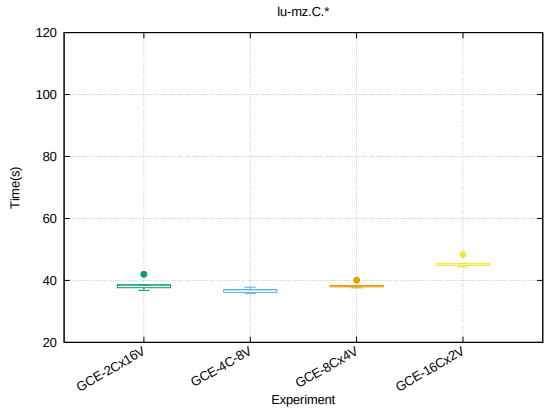


(b) hydra

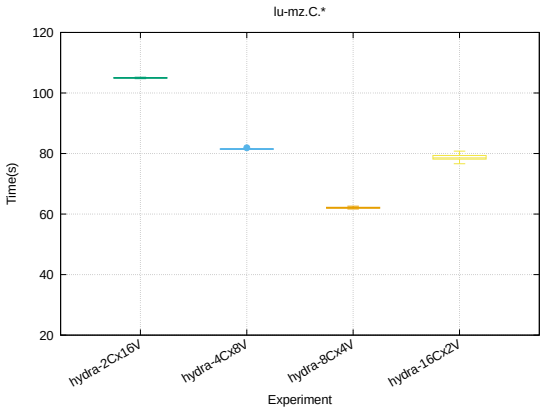


(c) revoada

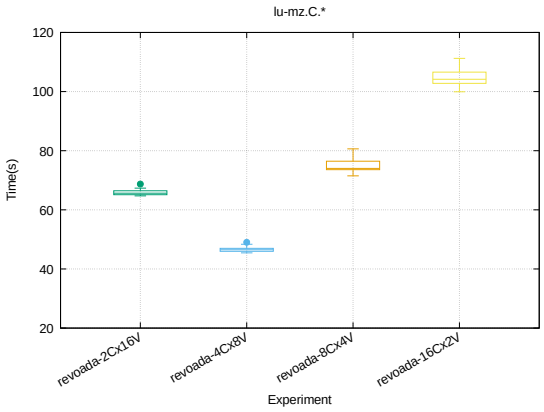
Figura A.7: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-MZ, tamanho B.*



(a) GCE

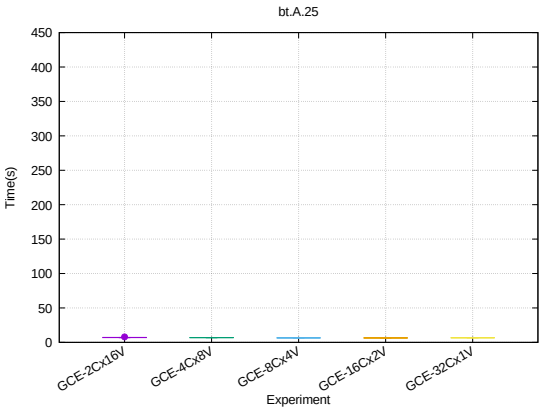


(b) hydra

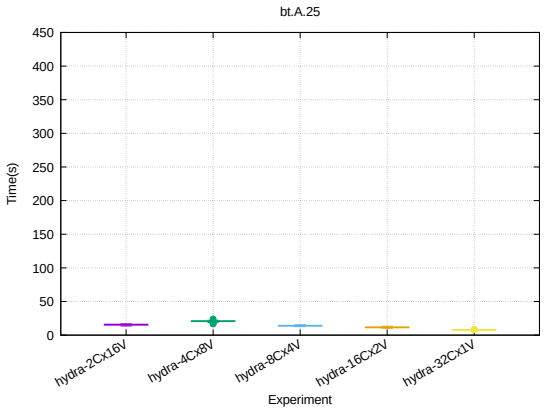


(c) revoada

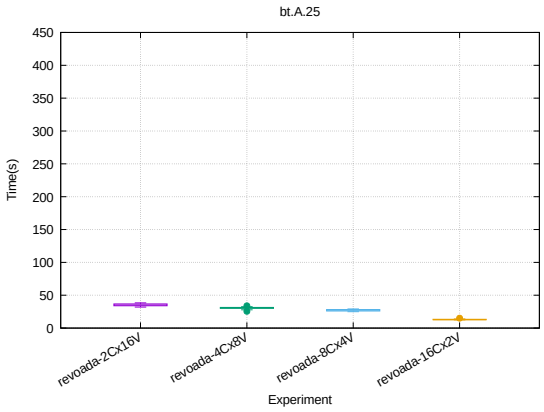
Figura A.8: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas LU-MZ, tamanho C.*



(a) GCE

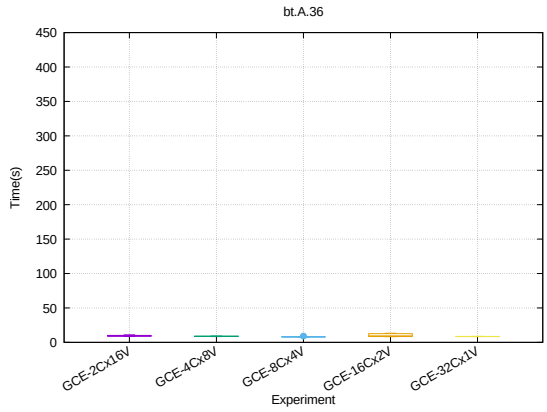


(b) hydra

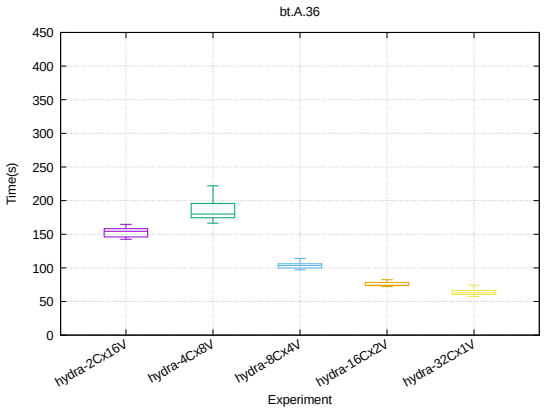


(c) revoada

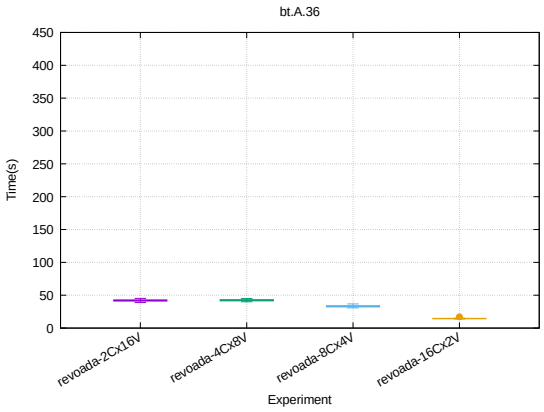
Figura A.9: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho A com 25 processos MPI.*



(a) GCE

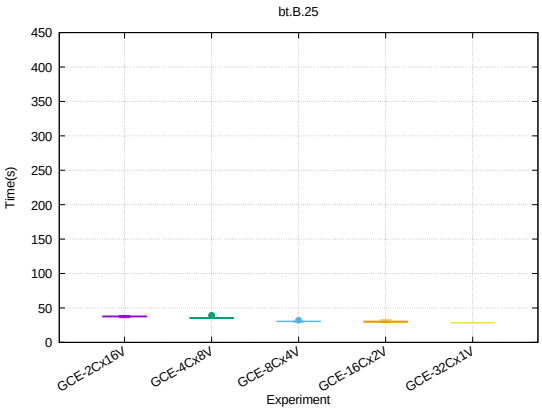


(b) hydra

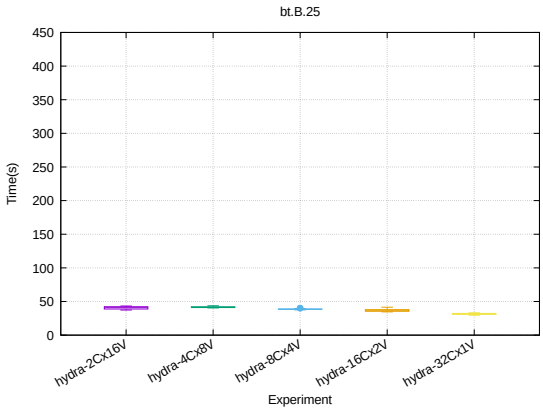


(c) revoada

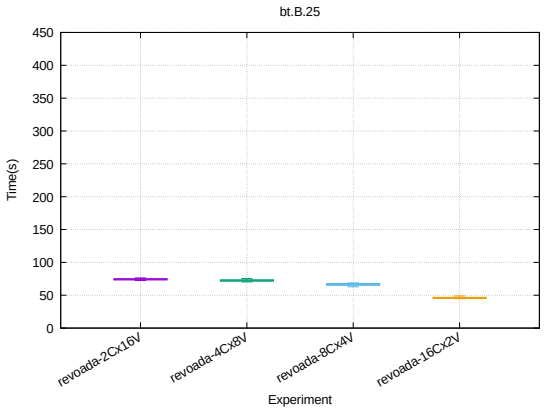
Figura A.10: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho A com 36 processos MPI.*



(a) GCE

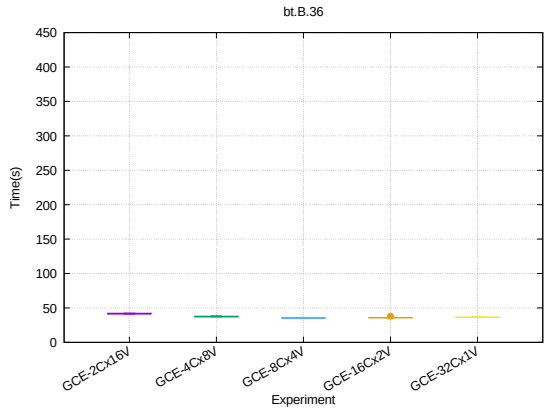


(b) hydra

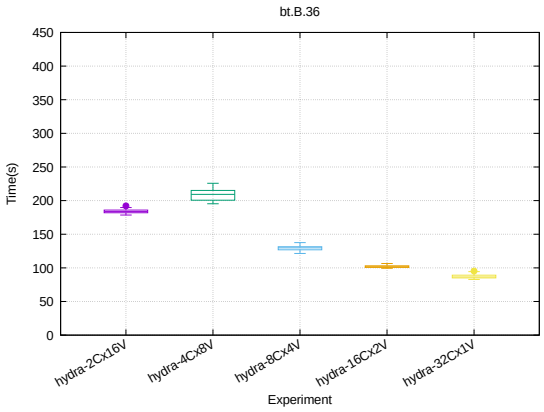


(c) revoada

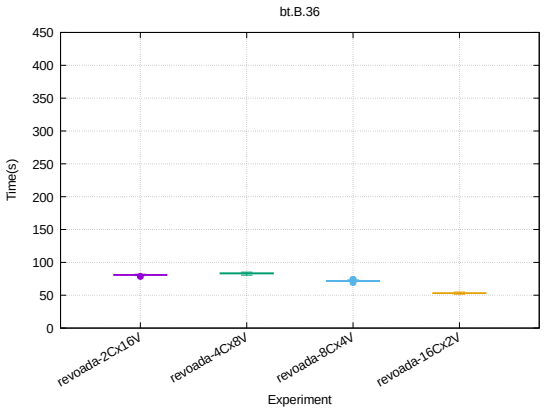
Figura A.11: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho B com 25 processos MPI.*



(a) GCE

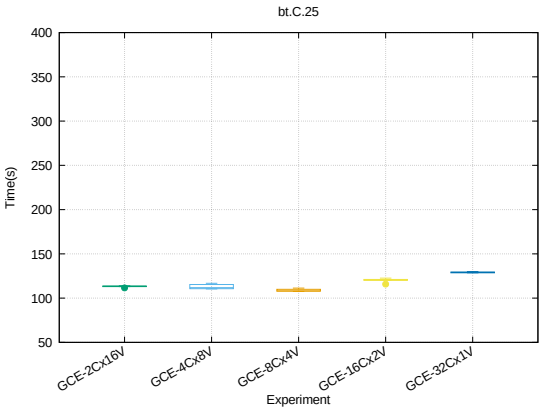


(b) hydra

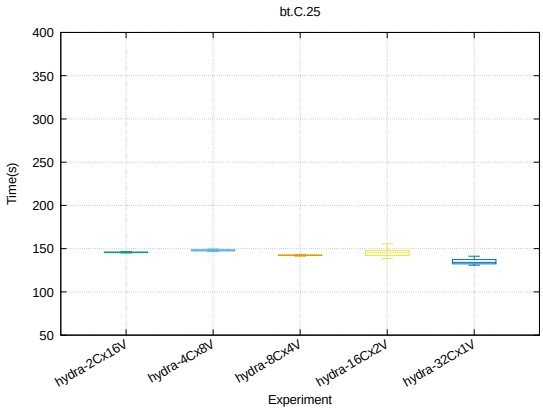


(c) revoada

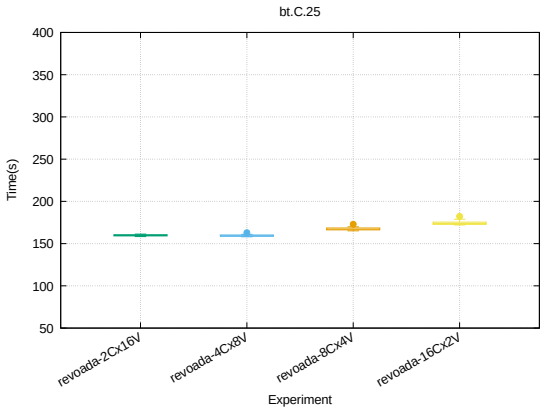
Figura A.12: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho B com 36 processos MPI.*



(a) GCE

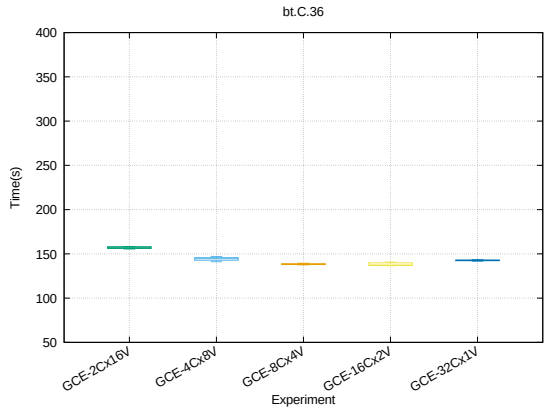


(b) hydra

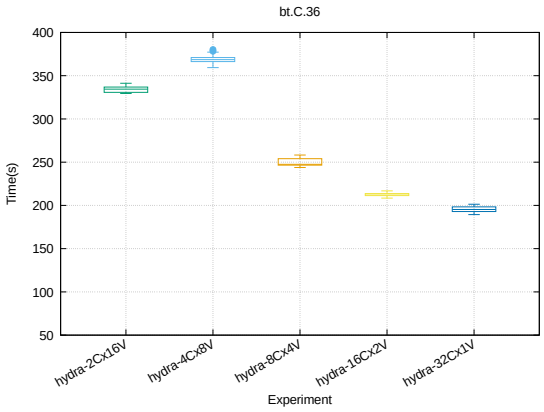


(c) revoada

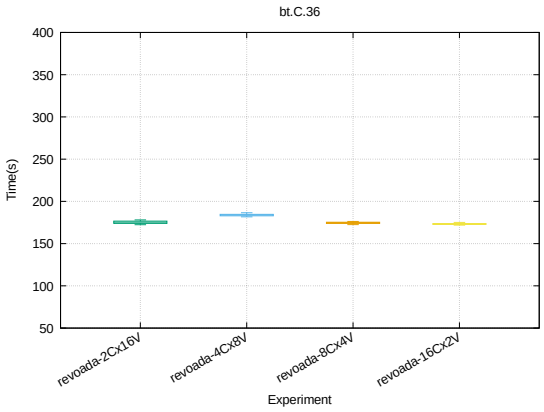
Figura A.13: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho C com 25 processos MPI.*



(a) GCE

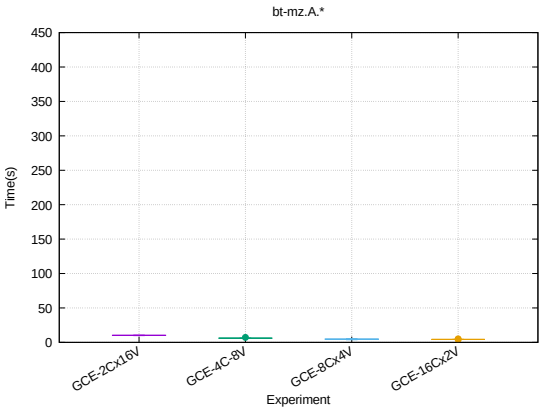


(b) hydra

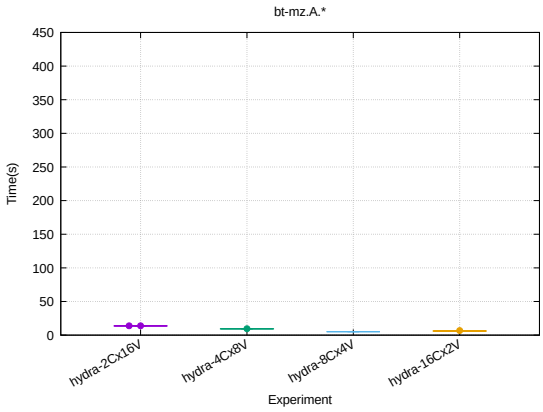


(c) revoada

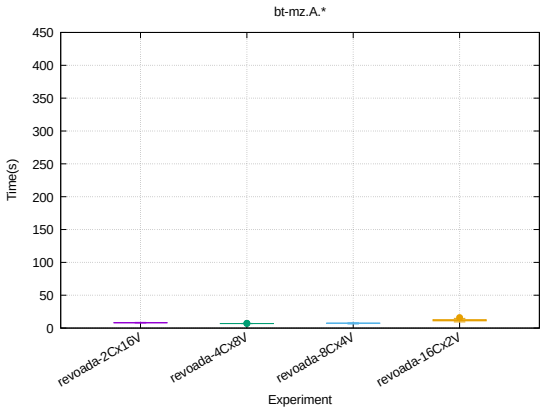
Figura A.14: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-SZ, tamanho C com 36 processos MPI.*



(a) GCE

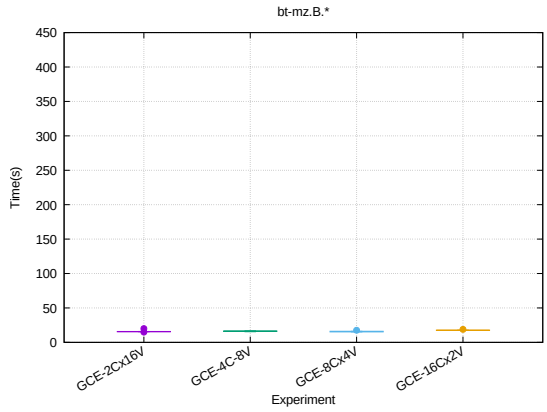


(b) hydra

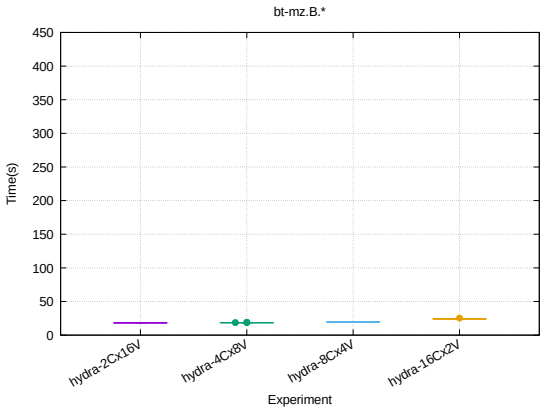


(c) revoada

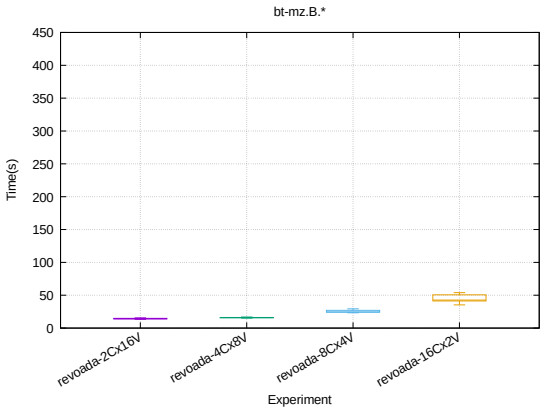
Figura A.15: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-MZ, tamanho A.*



(a) GCE

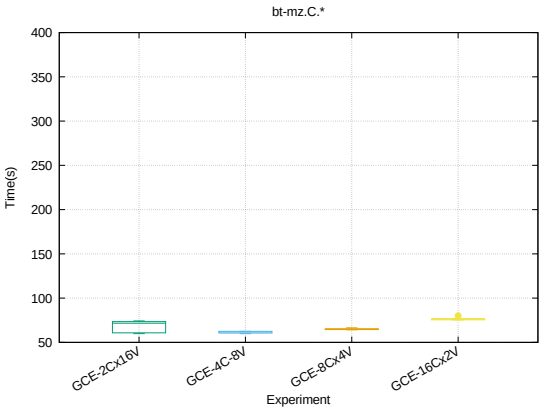


(b) hydra

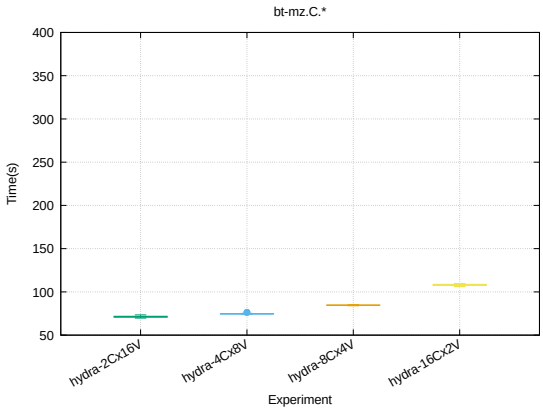


(c) revoada

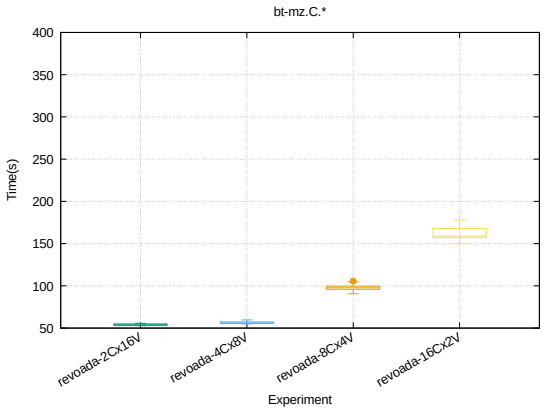
Figura A.16: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-MZ, tamanho B.*



(a) GCE

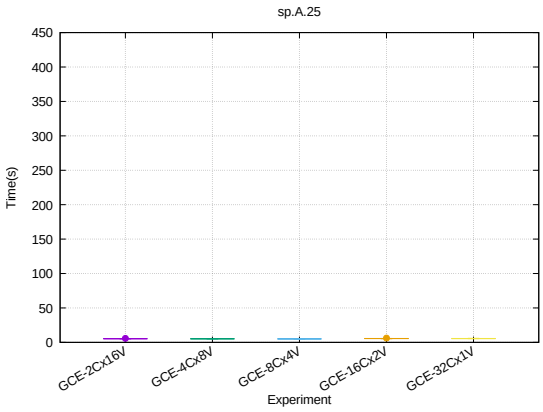


(b) hydra

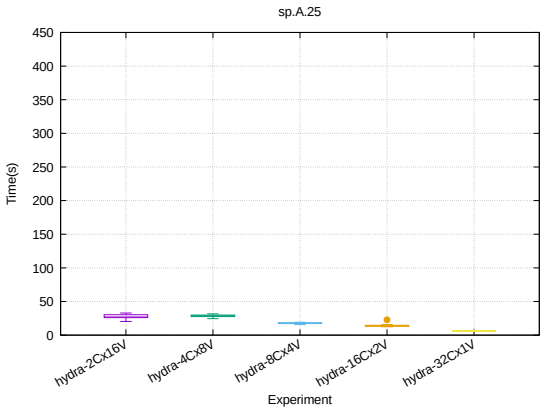


(c) revoada

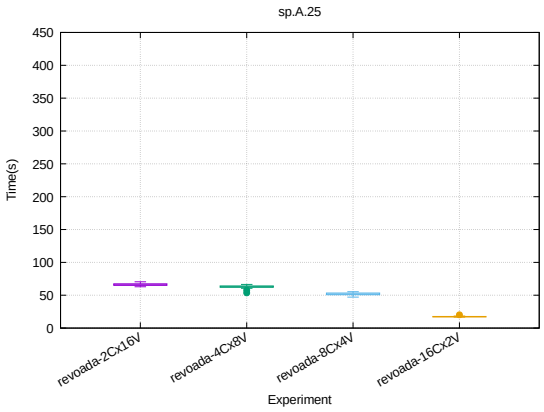
Figura A.17: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas BT-MZ, tamanho C.*



(a) GCE

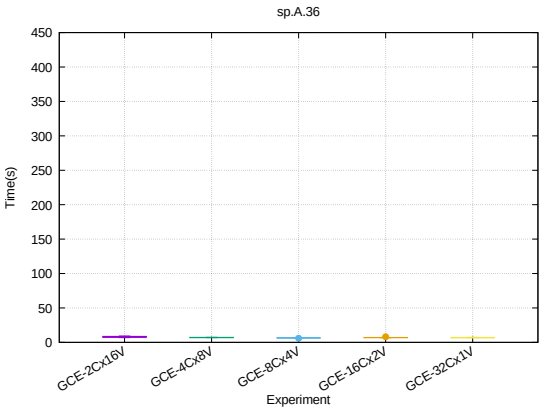


(b) hydra

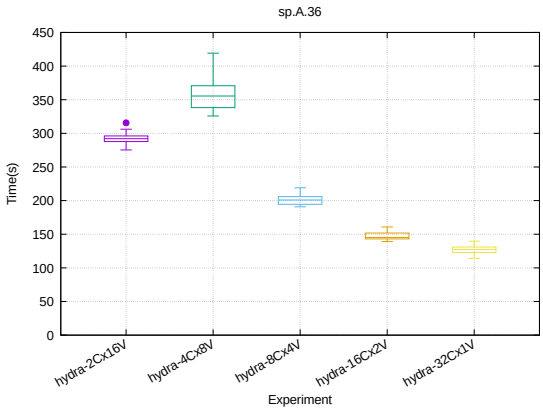


(c) revoada

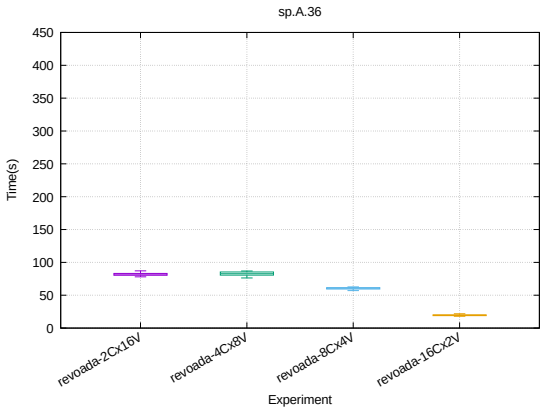
Figura A.18: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho A com 25 processos MPI.*



(a) GCE

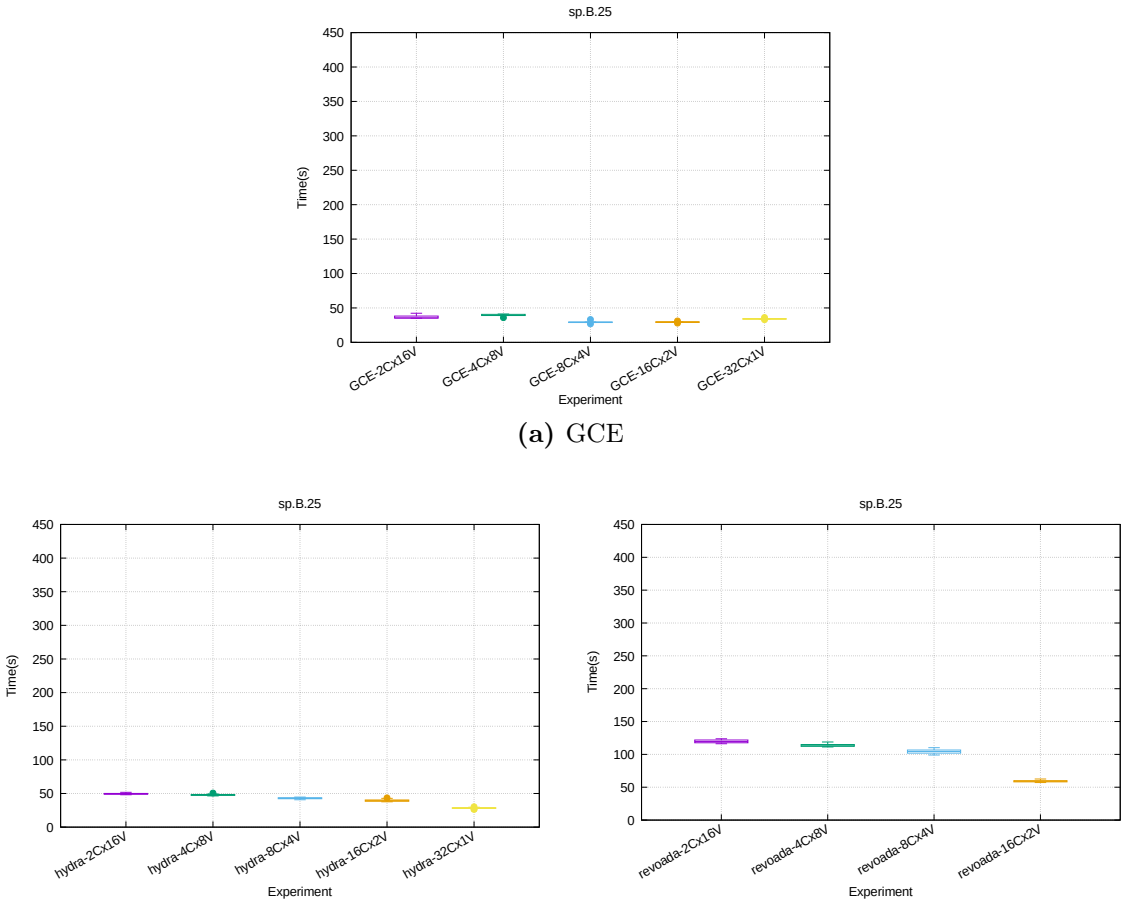


(b) hydra



(c) revoada

Figura A.19: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho A com 36 processos MPI.*

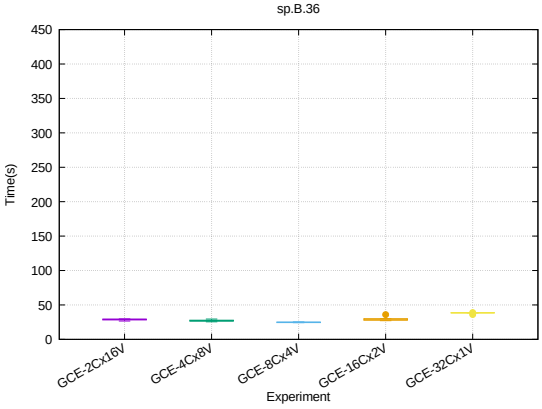


(a) GCE

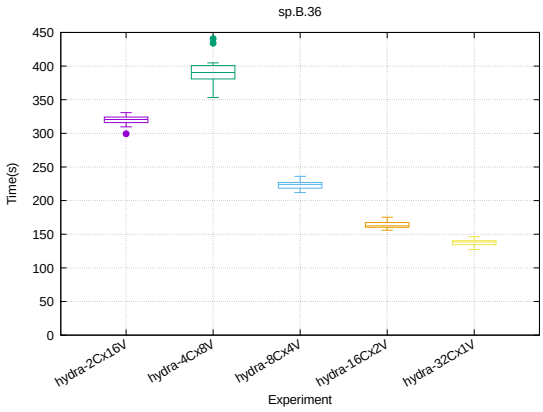
(b) hydra

(c) revoada

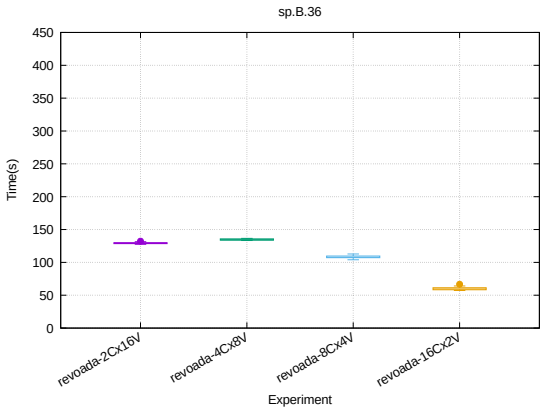
Figura A.20: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho B com 25 processos MPI.*



(a) GCE

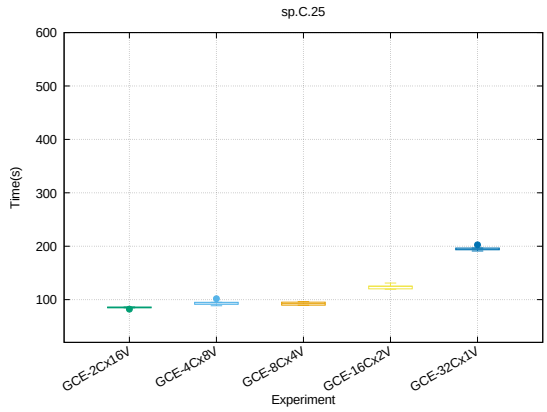


(b) hydra

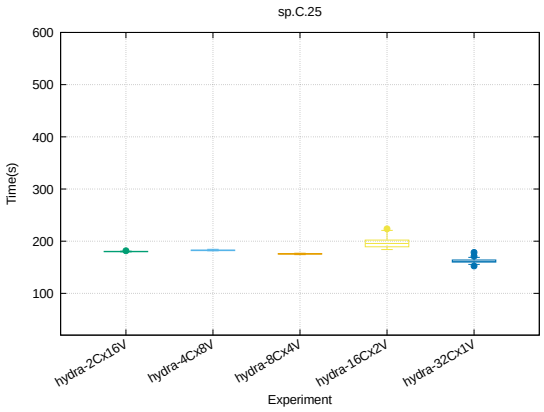


(c) revoada

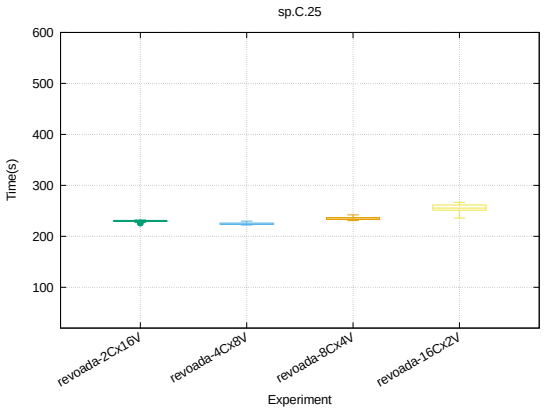
Figura A.21: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho B com 36 processos MPI.*



(a) GCE

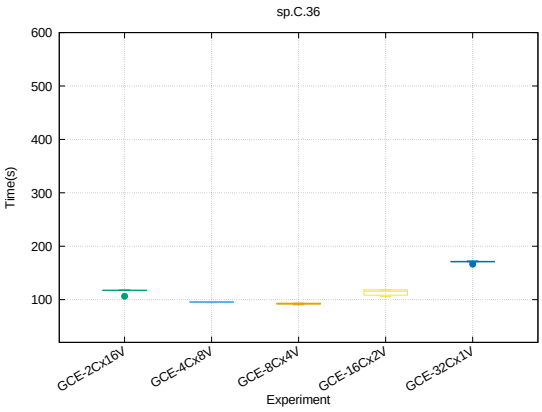


(b) hydra

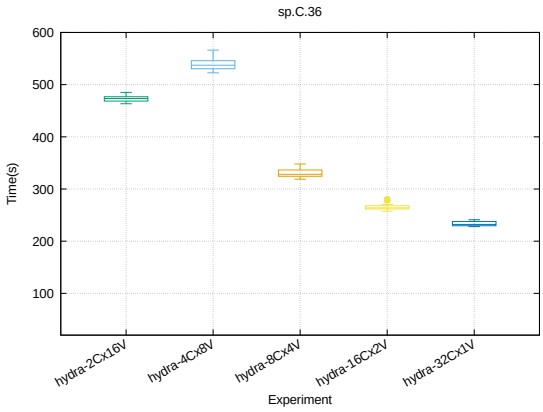


(c) revoada

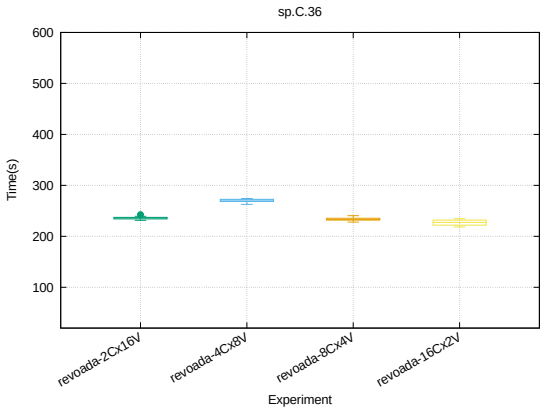
Figura A.22: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho C com 25 processos MPI.*



(a) GCE

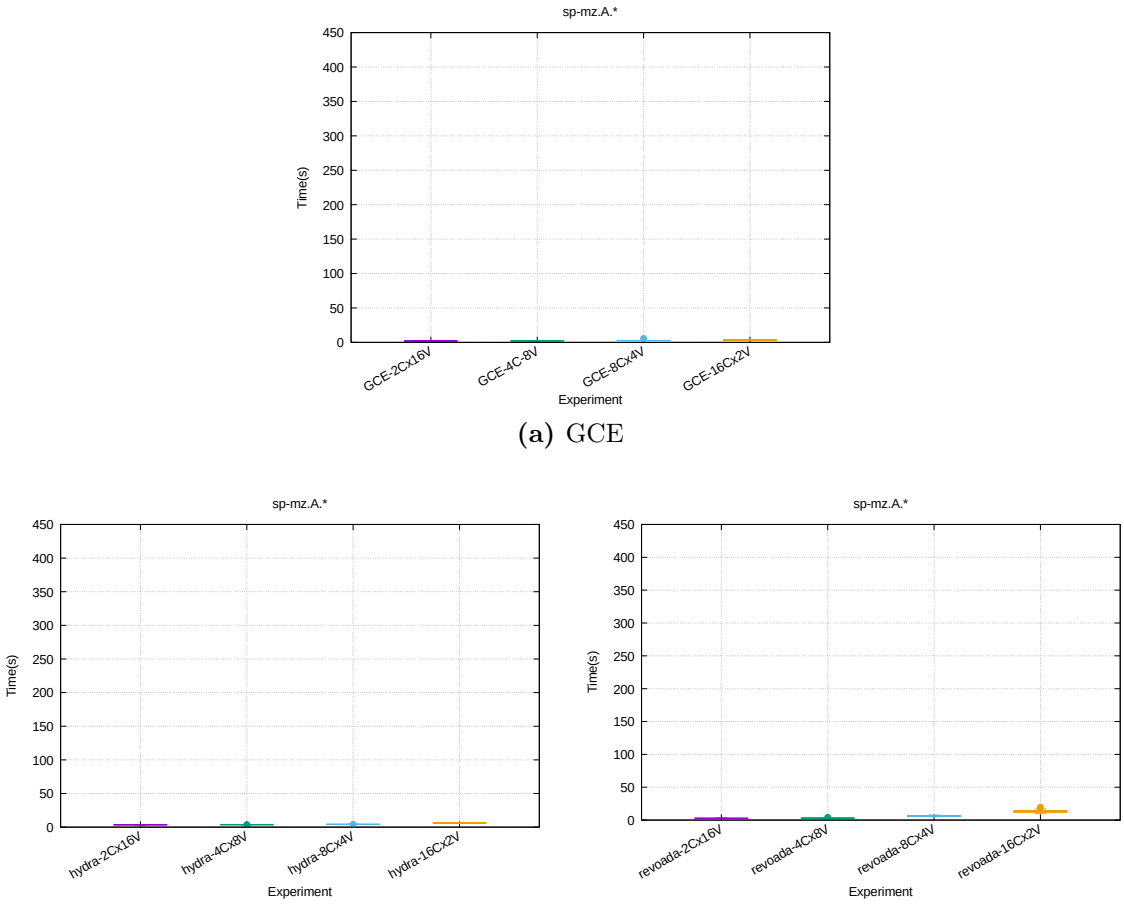


(b) hydra



(c) revoada

Figura A.23: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-SZ, tamanho C com 36 processos MPI.*

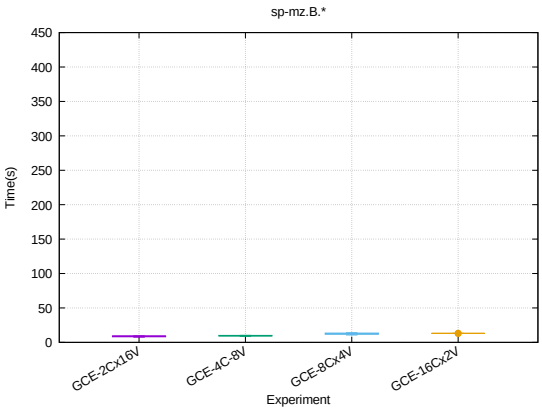


(a) GCE

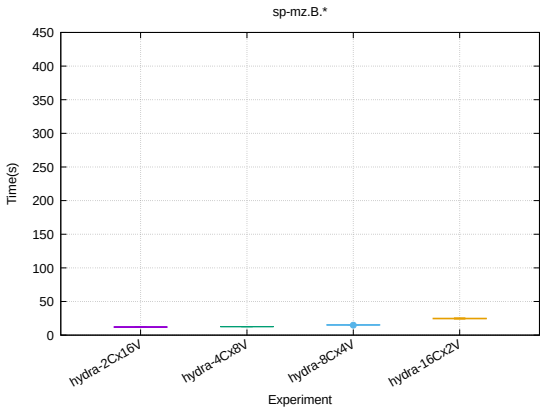
(b) hydra

(c) revoada

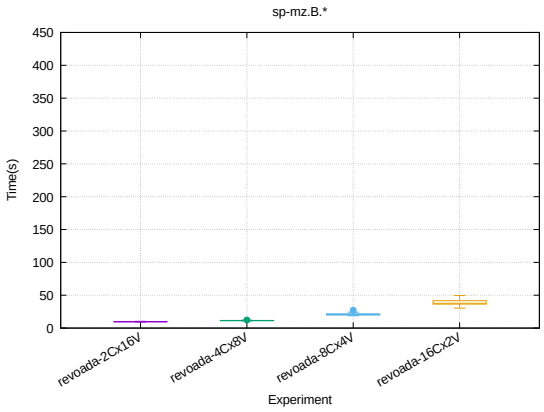
Figura A.24: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-MZ, tamanho A.*



(a) GCE

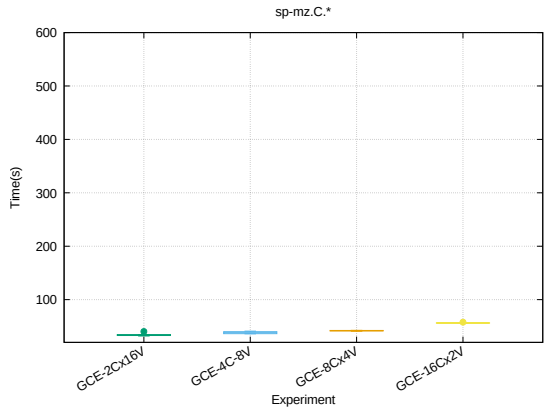


(b) hydra

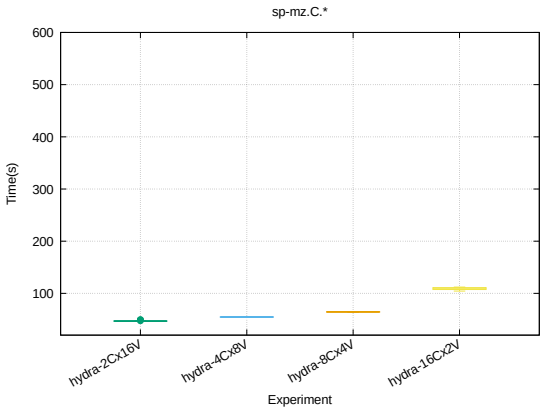


(c) revoada

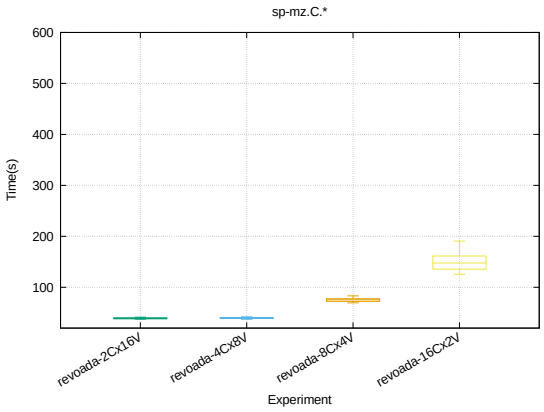
Figura A.25: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-MZ, tamanho B.*



(a) GCE



(b) hydra



(c) revoada

Figura A.26: *Diagrama de caixa do tempo total de execução nas infraestruturas testadas SP-MZ, tamanho C.*

Apêndice B

Usabilidade em nuvens computacionais

Tanto a Microsoft Azure[1] quanto o Google Compute Engine[2] oferecem ferramentas de linha de comando para permitir o gerenciamento de suas nuvens públicas. Essas ferramentas permitem instanciar, destruir, modificar, migrar máquinas virtuais, além de recursos relacionados (como discos virtuais ou imagens de sistemas operacionais), de forma programática, usando a linha de comando.

Durante o desenvolvimento desse trabalho, foram utilizadas as ferramentas citadas para permitir instanciar, executar e destruir, de forma automática, um grande número de máquinas virtuais. De uma forma geral, ambas as ferramentas se comportaram muito bem, porém as ferramentas fornecidas pelo Google se mostraram mais simples de usar e mais estáveis. Em um caso específico (quando o autor tentou subir uma imagem na nuvem), a ferramenta fornecida pela Microsoft para a tarefa começou a consumir uma quantidade grande de memória RAM (passando da casa dos 4GB), até ser necessário matar o processo antes que o sistema operacional acabasse travando.

Tanto a Microsoft[3] quanto o Google[4] também oferecem SDKs (*Software Development Kit*) escritas na linguagem de programação Python. Grande parte das ferramentas usadas nesse trabalho foram escritas nessa linguagem, então faria sentido essas SDKs serem usadas ao invés de se basear nas ferramentas de linha de comando. Porém o autor teve grandes dificuldades para gerenciar máquinas virtuais usando o SDK disponível para o Microsoft Azure, onde para fazer tarefas simples como instanciar e destruir máquinas virtuais, foram necessárias um grande número de linhas de código. Em especial, o SDK que a Microsoft provê necessita de muita intervenção manual com o sistema, e retorna erros basicamente devolvendo o XML do servidor para a aplicação ao invés do uso de exceções que o próprio Python fornece, por exemplo. Por isso o autor resolveu as ferramentas de linha de comando como base, o que reduziu o número de linhas de código e tornou a aplicação como um todo mais estável. Aparentemente o SDK que o Google provê é melhor nesse quesito, porém ela não foi utilizada graças a frustração com o SDK da Microsoft.

Uma última interface que ambos os serviços fornecem é a interface Web. Ambas oferecem os recursos básicos de gerenciamento, como criar e instanciar máquinas virtuais. Porém a interface que o Google fornece é claramente superior em alguns aspectos: ela permite acesso às máquinas virtuais via SSH usando o próprio navegador Web, permite gerencia de múltiplas máquinas virtuais ao mesmo tempo e a interface em si parece mais rápida e responsiva. A interface Web da Microsoft não fornece nada disso, em especial o fato de você não conseguir

[1] https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-command-line-tools/
[2] https://cloud.google.com/compute/docs/gcloud-compute/
[3] https://azure.microsoft.com/pt-br/develop/python/
[4] https://cloud.google.com/compute/docs/tutorials/python-guide

destruir múltiplas máquinas virtuais de uma vez torna o processo desgastante quando você tem que remover, por exemplo, 32 MVs de uma vez. Porém, a Microsoft está desenvolvendo uma nova interface Web, que atualmente está em Beta[5], e por isso não foi utilizada durante o decorrer desses experimentos, mas que pode resolver vários dos problemas que a interface atual possui.

[5] https://azure.microsoft.com/en-us/services/preview/

Referências Bibliográficas

[AA06] Keith Adams e Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. *SIGARCH Comput. Archit. News*, 34(5):2–13, Outubro 2006. 18

[AFG+09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy H Katz, Andrew Konwinski, Gunho Lee, David A Patterson, Ariel Rabkin, Ion Stoica et al. Above the clouds: A Berkeley view of cloud computing. 2009. 13

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010. 1

[AM10] Sayaka Akioka e Yoichi Muraoka. HPC benchmarks on Amazon EC2. *2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, páginas 1029–1034, 2010. 14

[BBB+91] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan e S. Weeratunga. The NAS parallel benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, 1991. 14, 16, 23, 24, 27

[BDBS92] D. H. Bailey, L. Dagum, E. Barszcz e H. D. Simon. NAS Parallel Benchmark results. Em *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, Supercomputing '92, páginas 386–393, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press. 13

[BHP+96] Gianfranco Bilardi, Kieran T. Herley, Andrea Pietracaprina, Geppino Pucci e Paul Spirakis. BSP vs LogP. Em *Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '96, páginas 25–32, New York, NY, USA, 1996. ACM. 10

[CBT+14] Aftab Ahmed Chandio, Kashif Bilal, Nikos Tziritas, Zhibin Yu, Qingshan Jiang, Samee U Khan e Cheng-Zhong Xu. A comparative study on resource allocation and energy efficient job scheduling strategies in large-scale parallel computing systems. *Cluster Computing*, 17(4):1349–1367, 2014. 1

[CKP+93] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian e Thorsten Von Eicken. LogP: Towards a realistic model of parallel computation. Em *ACM Sigplan Notices*, volume 28, páginas 1–12. ACM, 1993. 10

[Dro09] Maciej Drozdowski. *Scheduling for Parallel Processing*. Springer London, 2009. 9

[dWH03] Rob F Van der Wijngaart e Jin Haopiang. NAS Parallel Benchmarks, multi-zone versions. Relatório técnico, NASA, 2003. 24

[EH08] Constantinos Evangelinos e Chris Hill. Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2. *October*, 2(2.40):2–34, 2008. 14

[FK99] Sándor Forrai e Péter Kacsuk. Industrial supercomputing center in hungary-pre-feasibility study. Em *International Conference on High-Performance Computing and Networking*, páginas 1242–1245. Springer, 1999. 7

[FY02] Ahmad Faraj e Xin Yuan. Communication characteristics in the NAS Parallel Benchmarks. *Proceedings of the Parallel and Distributed Computing and Systems (PDCS) '02*, páginas 729–734, 2002. 16, 29

[GBC12] Andrzej Goscinski, Michael Brock e Philip C Church. *High performance computing clouds*. CRC Press, 2012. 18

[GM12] Abhishek Gupta e Dejan Milojicic. Evaluation of HPC applications on cloud. *Proceedings - 2011 6th Open Cirrus Summit, OCS 2011*, páginas 22–26, 2012. 14

[Hup09] Karl Huppler. *The Art of Building a Good Benchmark*, páginas 18–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 11, 12, 23

[HZK+10] Qiming He, Shujia Zhou, Ben Kobler, Dan Duffy e Tom McGlynn. Case study for running HPC applications in public clouds. Em *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, página 395, New York, New York, USA, jun 2010. ACM Press. 14, 17, 18

[JDV+09] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman e Phil Maechling. Scientific workflow applications on amazon ec2. Em *2009 5th IEEE International Conference on E-Science Workshops*, páginas 59–66. IEEE, 2009. 18

[JdW04] H. Jin e R. F. Van der Wijngaart. Performance characteristics of the multi-zone NAS parallel benchmarks. Em *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, páginas 6–, April 2004. 14

[JJF+99] H. Jin, H. Jin, M. Frumkin, M. Frumkin, J. Yan e J. Yan. The OpenMP implementation of NAS Parallel Benchmarks and its performance. Relatório técnico, NASA, 1999. 14

[LAH+02] Tau Leng, Rizwan Ali, Jenwei Hsieh, Victor Mashayekhi e Reza Rooholamini. An empirical study of hyper-threading in high performance computing clusters. *Linux HPC Revolution*, 2002. 45

[LORZ13] Zheng Li, Liam OBrien, Rajiv Ranjan e Miranda Zhang. Early observations on performance of Google Compute Engine for scientific computing. Em *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 1, páginas 1–8. IEEE, 2013. 15, 16, 17, 43

[MDH+12] Piyush Mehrotra, Jahed Djomehri, Steve Heistand, Robert Hood, Haoqiang Jin, Arthur Lazanoff, Subhash Saini e Rupak Biswas. Performance evaluation of Amazon EC2 for NASA HPC applications. Em *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, páginas 41–50. ACM, 2012. 15

[MG11] Peter Mell e Tim Grance. The NIST definition of cloud computing. Relatório técnico, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011. 1, 4, 5, 6

[Moo11] Chuck Moore. Data processing in exascale-class computer systems. http://web.archive.org/web/20121016140030/http://www.lanl.gov/orgs/hpc/salishan/salishan2011/3moore.pdf, 2011. vii, 8

[MTM+09] Chao Ma, Yong Meng Teo, Verdi March, Naixue Xiongy, Ioana Romelia Pop, Yan Xiang He e Simon See. An approach for matching communication patterns in parallel applications. *IPDPS 2009 - Proceedings of the 2009 IEEE International Parallel and Distributed Processing Symposium*, 2009. vii, x, 15, 16, 25, 26, 27

[NMNAJ12] Klaithem Al Nuaimi, Nader Mohamed, Mariam Al Nuaimi e Jameela Al-Jaroodi. A survey of load balancing in cloud computing: Challenges and algorithms. Em *Proceedings of the 2012 Second Symposium on Network Cloud Computing and Applications*, NCCA '12, páginas 137–142, Washington, DC, USA, 2012. IEEE Computer Society. 1

[OIY+10] Simon Ostermann, Alexandria Iosup, Nezih Yigitbasi, Radu Prodan, Thomas Fahringer e Dick Epema. A performance analysis of EC2 cloud computing services for scientific computing. Em *Cloud Computing*, páginas 115–131. Springer, 2010. 2

[Pad11] David Padua. *Encyclopedia of parallel computing*, volume 4. Springer Science & Business Media, 2011. 8, 11

[Pep11] K. Pepple. *Deploying OpenStack*. O'Reilly Media, 2011. 18

[Red15] Red Hat. Chapter 5. KVM live migration - Red Hat Customer Portal. https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Virtualization_Administration_Guide/chap-Virtualization_Administration_Guide-KVM_live_migration.html, 2015. Último acesso em 29/06/2015. 2

[Sin04] Amit Singh. An introduction to virtualization. http://www.kernelthread.com/publications/virtualization/, 2004. Último acesso em 16/08/2016. 7

[SJL11] Sangmin Seo, Gangwon Jo e Jaejin Lee. Performance characterization of the NAS parallel benchmarks in OpenCL. Em *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, páginas 137–148. IEEE, 2011. 14

[ST98] David B. Skillicorn e Domenico Talia. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2):123–169, Junho 1998. 8, 9, 10

[Val90] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, Agosto 1990. 35

[VPB09] Christian Vecchiola, Suraj Pandey e Rajkumar Buyya. High-performance cloud computing: A view of scientific applications. Em *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, páginas 4–16. IEEE, 2009. 13

[VRMCL08] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres e Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008. 7

[WB10] Yi Wei e M. Brian Blake. Service-oriented computing and cloud computing: Challenges and opportunities. *IEEE Internet Computing*, 14(6):72–75, Novembro 2010. 1

[WHV10] P. Wang, W. Huang e C. A. Varela. Impact of virtual machine granularity on cloud computing workloads performance. Em *2010 11th IEEE/ACM International Conference on Grid Computing*, páginas 393–400, Oct 2010. 18

[WYB13] Feng Wang, Canqun Yang e Juncheng Bai. Hybrid MPI/OpenMP optimization in Linpack benchmark on multi-core platforms. Em *Computer Science & Education (ICCSE), 2013 8th International Conference on*, páginas 917–920. IEEE, 2013. 11

[ZLZ+11] Yan Zhai, Mingliang Liu, Jidong Zhai, Xiaosong Ma e Wenguang Chen. Cloud versus in-house cluster: Evaluating Amazon cluster compute instances for running MPI applications. *Proceedings 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, páginas 1–10, 2011. 15