SINCRONIZAÇÃO GLOBAL DE RELÓGIOS FÍSICOS E AVALIAÇÃO DE AMBIENTES COMPUTACIONAIS ELÁSTICOS

Diego Leonel Cadette Dutra

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientador: Claudio Luis de Amorim

Rio de Janeiro
Março de 2015

SINCRONIZAÇÃO GLOBAL DE RELÓGIOS FÍSICOS E AVALIAÇÃO DE AMBIENTES COMPUTACIONAIS ELÁSTICOS

Diego Leonel Cadette Dutra

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Claudio Luis de Amorim, Ph.D.

Prof. Felipe Maia Galvão França, Ph.D.

Prof. Luis Felipe Magalhães de Moraes, Ph.D.

Prof. Noemi de La Rocque Rodriguez, D.Sc.

Prof. Jairo Panetta, Ph.D.

Prof. Alba Cristina Magalhães Alves de Melo, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2015

Dutra, Diego Leonel Cadette

Sincronização Global de Relógios Físicos e Avaliação de Ambientes Computacionais Elásticos/Diego Leonel Cadette Dutra. – Rio de Janeiro: UFRJ/COPPE, 2015.

XX, 140 p.: il.; 29, 7cm.

Orientador: Claudio Luis de Amorim

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2015.

Referências Bibliográficas: p. 122 – 138.

1. Avaliação de Desempenho. 2. InfraEstrutura como serviço. 3. Timekeeping. I. Amorim, Claudio Luis de . II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

À minha mãe pelo dom da vida,
pelo amparo ao longo desses
anos e paciência para escutar
minhas ideias, mesmo quando
não as entendia, e minha irmã
pela companhia e por todas as
conversas tarde da noite.

Agradecimentos

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e a Financiadora de Estudos e Projetos (FINEP) pelo suporte financeiro.

Agradeço aos meus professores de graduação na UFF que tanto me ensinaram principalmente os da área de arquitetura de computadores, sistemas operacionais e redes.

Agradeço aos Técnicos-Administrativos do Programa de Engenharia de Sistemas, que sempre auxiliaram e estiveram de prontidão para responder minhas duvidas, por mais bobas que fossem.

Agradeço aos professores do Programa de Engenharia de Sistemas e Computação da COPPE pelas aulas e discussões tão interessantes e divertidas que tive nesses 8 anos.

Um agradecimento especial aos colegas e colaboradores do Laboratório de Computação Paralela (LCP) que me ajudaram ao longo deste período de estudos e trabalho.

Um agradecimento especial ao Doutor Lauro Whatley, por tantos ensinamentos e discussões que em muito ajudaram a completar minha formação ao longo do mestrado e doutorado.

Um agradecimento especial à Professora Raquel Pinto, pelos ensinamentos e discussões, vindo do IME para COPPE para participar de diversas reuniões deste trabalho.

Um agradecimento especial para o Professor Claudio Luis de Amorim que tem me orientado desde meu primeiro dia na COPPE e aceitou me orientar durante meu doutorado. Muito obrigado pelo suporte, ajuda e ensinamentos que nunca faltaram nestes 8 anos.

Um agradecimento especial ao Professor Leonardo Pinho e sua família, amigos muito queridos que apesar da distância se fazem sempre presente em minha vida.

Um agradecimento especial à Mariana e Isabela Pinho por me fazerem rir, mesmo estando tão longe.

Um agradecimento a todos meus amigos e as pessoas que de alguma forma acabaram por novamente ficar em segundo plano durante o desenvolvimento deste tra-

balho.

Um agradecimento especial a meu tio Francisco de Paula Dutra Filho que sempre esteve presente em minha vida e foi o primeiro orientador que tive em minha carreira na computação.

Um agradecimento especial a minha irmã Vivian Marcia Cadette Dutra por me aturar nos momentos de estresse e sempre ter a piada perfeita para me fazer relaxar.

Um agradecimento especial a minha mãe Zenaide Cadette dos Santos Dutra que sempre cultivou em seus 2 filhos o prazer de estudar e aprender, mesmo que isso os tenha deixado com trauma de serem jogados pela janela se ficassem reprovados na escola.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SINCRONIZAÇÃO GLOBAL DE RELÓGIOS FÍSICOS E AVALIAÇÃO DE AMBIENTES COMPUTACIONAIS ELÁSTICOS

Diego Leonel Cadette Dutra

Março/2015

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

A eficiência energética é um novo desafio para o uso exigente de sistemas computacionais modernos. Nas aplicações paralelas, a eficiência energética reflete o percentual de tempo que os processadores gastam executando computação útil. Nos ambientes de nuvem (*Cloud Computing*), especialmente os ambientes de infraestrutura como serviço (em inglês, *Infrastructure as a Service* - IaaS), a eficiência energética deriva do percentual dos recursos computacionais alocados a uma aplicação e que efetuem computação útil.

Aplicações elásticas possuem as habilidades de requisitar novos recursos computacionais e de liberar recursos ociosos, ambas sob demanda. No contexto desta tese, a **elasticidade** é uma propriedade da aplicação que impacta diretamente sua eficiência energética e que depende tanto da forma como a aplicação foi construída como do ambiente de IaaS. Esta tese apresenta uma nova proposta para avaliar quantitativamente o desempenho de ambientes de IaaS através da aferição do impacto que tais ambientes exercem sobre uma aplicação elástica de *Video On Demand* (VoD).

Em um ambiente de computação paralela, os relógios locais de cada nó computacional precisam executar periodicamente operações de sincronização com um relógio global remoto para garantir medidas de tempo precisas e corretas mas ao custo de afetar negativamente sua eficiência energética. Esta tese propõe e avalia um novo mecanismo em *software* para a manutenção de relógios globais com capacidade de dispensar operações de sincronização, portanto permitindo aumentar a aceleração de aplicações paralelas e sua eficiência energética.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

GLOBAL PHYSICAL CLOCK SYNCHRONIZATION AND EVALUATION OF ELASTIC COMPUTATIONAL ENVIRONMENTS

Diego Leonel Cadette Dutra

March/2015

Advisor: Claudio Luis de Amorim

Department: Systems Engineering and Computer Science

Energy efficiency poses a new challenge to the demanding use of modern computing systems. In parallel applications, energy efficiency reflects the percentage of time the processors spent performing useful computation. In *Cloud Computing* environments, especially in *Infrastructure as a Service* (IaaS) environments, the energy efficiency derivatives of the percentage of the computing resources allocated to an application that are performing useful computation.

Elastic applications have the ability to request new computational resources, or release idle resources, both of them on demand. Within the scope of this thesis, the **elasticity** is an application's property that impacts directly its energy efficient and that depends as much on how the application is built as the IaaS environment. This thesis presents a new proposal to quantitatively evaluate the performance of IaaS environments by measuring the impact these environments has upon an elastic *Video On Demand* (VoD) application.

In a parallel computing environment, the local clocks of each compute node must periodically perform synchronization operations with a remote global clock to ensure accurate and correct time measurements but a the cost of negatively affect its energy efficiency. This thesis proposes and evaluates a new software mechanism for maintaining global clocks capable of avoiding synchronization operations, thus allowing to increase the acceleration of parallel applications and their energy efficiency.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

Cloud Computing Conjunto de serviços computacionais em larga escala distribuídos, dinamicamente reconfiguráveis e escaláveis, entregues sob demanda aos usuários e acessíveis através da Internet. vii, viii, 30

benchmark Programa utilizado para comparar o desempenho dos componentes de um ambiente computacional, sob condições de referência, em relação a uma avaliação de referência. x, xiv, xv, 1–9, 19–24, 39, 40, 42–45, 47, 51, 63, 65–76, 80, 108–114, 117–121

cluster Agrupamento de computadores ligados a um *Front-End*. ix–xi, xiii, 1, 6–8, 13, 16, 17, 19, 32–35, 37–41, 43, 47, 51, 77, 79, 81, 84–88, 90, 99–102, 106–108, 111, 115, 119–121

kernel Núcleo do sistema operacional (por exemplo, Kernel do Linux). xvi, 1, 13, 18, 24, 26–32, 62, 83, 91, 104, 115, 118, 121

API Application Programming Interface (ou Interface de Programação de Aplicações). 34, 36

elasticidade É a habilidade de o usuário ou aplicação requisitar novos recursos computacionais, ou liberar recursos ociosos, sob demanda. vii, 4, 8, 9, 32, 40–43, 45, 109, 112, 118

resiliência Propriedade da arquitetura computacional de atender as demandas de alocação e liberação de recursos por parte dos usuários ou aplicações sob demanda. 4, 8, 9, 32, 40–43, 47, 110, 112, 118, 119

XML-RPC Protocolo de chamada de procedimento remoto (CPR) que utiliza XML para codificar suas chamadas e HTTP como um mecanismo de transporte. 34

Acrônimos

AggBand Banda agregada total utilizada. 49

PcAf Percentual de clientes atendidos com falha. 49

PcBl Percentual de Clientes Bloqueados por latência de alocação. 49, 65–77

QtRvS Quantidade total de requisições de Vídeo recebidas. 49, 65, 66, 68, 69, 71, 73–77

SPLASH ***S****tanford* ***P****aralle****L*** ***A****pplications for* ***SH****ared memory*. 23

ACPI PMT *ACPI Power Management Timer*. 12

APIC *Local Advanced Programmable Interrupt Controller*. 12

CNK *Compute Node Kernel*. 18, 118

Dom0 Domínio Privilegiado. 28, 52, 53

DomU Domínio não Privilegiado. 28

DVFS *Dynamic Voltage and Frequency Scaling*. 2, 13, 19, 83, 92, 96, 114, 115

EC2 *Elastic Compute Cloud*. 33, 34, 38, 41, 107–110, 112, 113, 121

ECP Estritamente Crescente e Preciso. xi, xv, 8, 17, 81, 83, 86, 88, 90–93, 96–98, 103, 113–116, 119

FWK *Full-Weight Kernel*. 18

GPS *Global Positioning System*. xv, 9, 90, 91, 93–97, 106

HPC *High Performance Computing*. 108, 113

HPET *High Performance Event Timer*. 1, 2, 12, 90, 115

HTTP *Hypertext Transfer Protocol.* 7, 37, 42, 45–47, 49, 51, 52, 54–56, 62–65, 111, 119, 121

IaaS *Infrastructure as a Service.* vii, viii, x–xiii, xvi, 7–9, 20, 31–36, 39–51, 63–66, 69, 71–74, 76, 77, 79, 80, 107–114, 118–121

Intel VT *Intel Virtualization Technology.* 25

KVM ***K****ernel-based* ***V****irtual* ***M****achine.* xi, xiii–xvi, 26, 27, 29, 33, 35, 36, 60–68, 70–74, 76, 77, 80, 120

LWK *Light-Weight Kernel.* 18

LXC *Linux Containers.* xi, xvi, 29, 30, 36, 56–58, 60, 63, 80

MMV Monitor de Máquina Virtual. 25–28, 57, 62, 63

MPI *Message Passing Interface.* 38, 84, 87, 89, 103–105, 108, 116, 117

NCSA *National Center for Supercomputing Applications.* 107, 108

NPB *NAS Parallel Benchmarks.* 107, 113

NTP *Network Time Protocol.* xiii, xv, 2, 5, 9, 14–16, 87, 102, 104–106, 113–116

PaaS *Platform as a Service.* 31, 32, 109, 111, 113, 114

PARSEC ***P****rinceton* ***A****pplication* ***R****epository for* ***S****hared-m****E****mory* ***C****omputers.* xvi, 22, 24

PIT *Programmable Interval Timer.* 11, 12

POP *Parallel Ocean Program.* x, xi, xv, xvi, 3, 8, 9, 18, 19, 103–106, 117, 120

PTP *Precision Time Procol.* 16, 116

QoS Qualidade de Serviço. 38, 39, 41, 45

RGAP Relógio Global de Alta Precisão. xi, xii, xv, 8, 9, 81, 85–91, 98–106, 113, 116, 118–121

RTC *Real-Time Clock.* 2, 11

RTT *Round-Trip Time.* 85, 88

RVEC Relógio Virtual Estritamente Crescente. xi, xv, 8, 9, 81–88, 90, 91, 96–99, 103, 106, 116, 120, 121

S3 *Simple Storage Service*. 37

SaaS *Software as a Service*. 31, 32

SPEC ***S****tandard* ***P****erformance* ***E****valuation* ***C****orporation*. xvi, 21, 22

TaR Tempo para alocação de Recurso. 44, 50

TlR Tempo para liberação de Recurso. 44

TSC *Time Stamp Counter*. xv, 1, 2, 9, 13, 14, 81–83, 91–99, 106, 114, 115

VoD *Video On Demand*. vii, viii, xiii, 7, 9, 38, 39, 45, 48, 49, 119

Capítulo 1

Introdução

Este capítulo apresenta os assuntos desenvolvidos nesta tese, destacando os problemas que motivaram desenvolvimento das pesquisas sobre o tema, o contexto deste trabalho, os objetivos desta tese e delineia a organização dos demais capítulos.

1.1 Problemas de Interesse

Na computação, como em outras ciências, o tempo é uma dimensão fundamental, seja no escalonamento de tarefas em sistemas embarcados ou supercomputadores, depuração de aplicações paralelas, na avaliação de desempenho de novos ambientes computacionais ou melhorias propostas em ambientes existentes através de *benchmark*s. Nos sistemas computacionais, a aferição do tempo é feita através dos relógios de sistemas, estes implementados pelo sistema operacional sobre circuitos básicos presentes no *hardware* como o *High Performance Event Timer* (HPET) [1]. Através deles, as aplicações e *softwares* de sistema medem os tempos de execução e implementam operações de sincronização de relógios físicos em *cluster*s de computadores, usando uma fonte externa para manutenção de um relógio global. Contudo, a precisão dos relógios do sistema depende de fatores como, por exemplo, a quantidade de interrupções perdidas e a variação intrínseca nos intervalos de tempo entre interrupções.

Sistemas operacionais modernos como Linux implementam em seus *kernel*s mecanismo de interpolação através de circuitos como, por exemplo, o *Time Stamp Counter* (TSC), que por ser interno ao núcleo de processamento permite mitigar o impacto causado pelas interrupções e aumentar a precisão destes circuitos básicos. Este tipo de mecanismo é utilizado pela chamada de sistema gettimeofday [1]. Concretamente, a interpolação resultante é, em última análise realizada com base em

[1] arch/x86/vdso/vclock_gettime.c

valores gerados por um circuito em *hardware* que opera a base de interrupções, como o HPET ou o *Real-Time Clock* (RTC), que podem causar erros de ordem de $1\mu s$ por 50 ms, ou 20PPM [2]. O mecanismo de interpolação utiliza o TSC estando este marcado como estável, o que não ocorre em processadores com múltiplos núcleos se estes não estiverem sincronizados entre si. Contudo as implementações existentes dos mecanismos de temporização impactam diretamente no desempenho dos ambientes e das aplicações que executam sobre eles, tendo esse problema sido agravado nos sistemas computacionais energeticamente eficientes. Nos resultados dos *benchmark*s esse problema ocasiona uma redução no nível de confiança dos resultados, provocadas, por exemplo, pelo uso de diferentes taxas de atualização do contador em *hardware*, enquanto que nos ambientes paralelos esses mecanismos impactam diretamente na escalabilidade das aplicações, e nos ambientes de computação em nuvem aumentam a latência na alocação dos recursos.

Sistemas computacionais energeticamente eficientes empregam um número crescente de processadores com múltiplos núcleos, *Dynamic Voltage and Frequency Scaling* (DVFS) e suporte a virtualização. Contudo, os relógios atuais do sistema geralmente não foram projetados para lidar com a capacidade de tais mecanismos que podem desacelerar/acelerar a passagem do tempo. Especificamente, os processadores com DVFS podem reduzir a frequência de uma ou mais unidades de processamento ou mesmo desligar estas unidades, para efeitos de economia de energia ou proteção térmica, enquanto um mecanismo de suporte a virtualização permite que as máquinas virtuais migrem entre as unidades de processamento para fins de balanceamento de carga. No entanto, a operação de ambos os mecanismos está exposta a interrupções do sistema, o que contribui para o aumento da quantidade de desvios de tempo e reduzindo ainda mais a precisão dos relógios do sistema.

Por exemplo, TSC é frequentemente utilizado como uma solução de relógio para sistemas computacionais monoprocessados, em combinação com uma fonte externa para o tempo global, tal como um servidor *Network Time Protocol* (NTP) [3]. Apesar de ser simples e eficaz, este tipo de solução pode não ser aplicável a um processador com vários núcleos, em que cada núcleo tem seu próprio contador de tempo e em que um relógio de sistema tem de lidar com a sincronização de múltiplos contadores locais. Adicionalmente, ocorre um problema com interferência no decurso da computação que resulta da utilização de DVFS e do suporte de virtualização.

Além disso, o uso do NTP como mecanismo para manutenção do relógio global oferece baixa precisão e maior exposição a desvios de tempo e portanto inadequado às aplicações paralelas [4]. O NTP também aumenta o consumo de energia por causa das mensagens adicionais e as interrupções (de rede e *daemon*), reduzindo os períodos ociosos [5]. Em ambientes computacionais de alto desempenho, Ferreira et al. [6] mediram um acréscimo no tempo de execução que vai de 30% até 20 vezes

para o *Parallel Ocean Program* (POP) [7] executando em um sistema computacional com 10.000 nós por causa de diferentes fontes de ruído, fontes estas que provocam uma redução no percentual de tempo que a aplicação paralela gasta realizando computação útil. Além disso, ferramentas de análise de trace (*trace analysis tools*), tais como Intel Trace Collector [8] e o CrayPat [9] são dependentes de medidas de tempo precisas para seu correto funcionamento.

O conceito de percentual de tempo gasto em computação útil existente nos ambientes computacionais paralelos também é aplicável em ambientes de computação em nuvem, logo um ambiente computacional em nuvem é energeticamente eficiente quando este adapta o seu uso de recursos computacionais, liberando recursos ociosos ou alocando novos recursos, sob demanda, para atender a variação da demanda computacional. Nos projetos de sistemas computacionais elásticos, além do acréscimo na latência de alocação dos recursos, problemas com a temporização afetam as medidas de desempenho utilizadas para controlar a quantidade e uso dos recursos computacionais. Contudo, é importante observar o aspecto da avaliação de desempenho que o tempo influência ao lidar com esses novos ambientes computacionais. Esta mudança de paradigma na forma como são utilizados os recursos computacionais requer a construção de novos *benchmark*s, que consigam explorar os novos recursos disponíveis nestes sistemas e deste modo avaliarem o desempenho das diferentes soluções apresentadas para um mesmo problema. Essa abordagem demonstra ser altamente eficiente em sistemas monoprocessados, impulsionando o desenvolvimento de novas técnicas arquiteturais para o aumento do desempenho destes sistemas.

Os sistemas computacionais com vários processadores [2], por sua vez possuem características que tornam a análise individual do núcleo de processamento insuficiente para avaliar o desempenho do sistema computacional. Deste modo, é necessária a criação de novos *benchmark*s para os sistemas com vários processadores. Inicialmente foram utilizadas aplicações cientificas [10] que demandam grande poder computacional, surgindo posteriormente *benchmark*s formados por aplicações de outras classes como processamento multimídia e comércio eletrônico [11].

Sistemas computacionais escaláveis eram no final do século XX e na primeira década do século XXI projetados como agregados de computadores dedicados. Entretanto, os avanços técnicos dos últimos anos e a necessidade de otimizar o consumo energético produziram uma nova tendência no projeto desta classe sistema computacional, onde os recursos computacionais não possuem uma alocação estática, mas são requisitados ou liberados sob demanda de maneira a atender aos pedidos recebidos. A habilidade de um sistema computacional (usuário ou aplicação) requisitar novos

[2] Um sistema computacional com múltiplos processadores pode ser, um sistemas com múltiplas unidades de processamento interconectadas por um barramento de memória compartilhada (multiprocessador) ou por uma rede de interconexão (agregado de computadores ou multicomputador), como os que possuem unidades de processamento com múltiplos núcleos.

recursos computacionais, ou liberar recursos ociosos, sob demanda é denominada de **elasticidade** [12], e os sistemas computacionais que possuem esta habilidade são ditos sistemas computacionais elásticos. Neste modelo, o projetista do sistema computacional deve, além dos requisitos tradicionais do desenvolvimento de um sistema distribuído escalável, levar em consideração como o ambiente computacional que suporta seu sistema responde aos pedidos de alocação/liberação de recursos e de que forma isso impacta no sistema que está sendo projetado. A propriedade do ambiente computacional em atender as demandas de alocação e liberação de recursos por parte dos usuários ou aplicações sob demanda é denominada, dentro do escopo desta tese, de **resiliência** [3].

Observe ainda que o interesse em avaliar o comportamento de um dado ambiente computacional, que oferece suporte à elasticidade no uso dos recursos não é de interesse apenas do projetista da aplicação, mas também do provedor de serviço que oferece esses recursos. O projetista utiliza essas informações para diferenciar provedores de serviço de infraestrutura distintos e para adequar o sistema computacional às características impostas pelo ambiente computacional de um provedor. O provedor de serviço por sua vez deseja avaliar como os componentes e a organização de seu ambiente computacional responde às requisições feitas pelos seus usuários e como ele pode maximizar a utilização destes recursos.

Neste cenário, faz se necessário a construção de *benchmark*s que consigam explorar esta nova dimensão no projeto de sistemas computacionais seja do ponto de vista do projetista do sistema ou do projetista da ambiente computacional que executa esse sistema computacional, pois os *benchmark*s existentes não conseguem aferir esta nova característica e recordando o problema de temporização, construir novos mecanismos para a manutenção de relógios de sistemas e relógio global que minimizem seus impactos sobre os sistemas computacionais.

1.2 Contexto

O contexto desta tese compreende os ambientes computacionais energeticamente eficientes e as aplicações que executam sobre estes. Dentro deste escopo, um ambiente computacional energeticamente eficiente é definido como um sistema computacional que oferece suporte à alocação, liberação ou alteração da capacidade sob demanda dos seus recursos computacionais.

Nos ambientes de computação de alto desempenho, a quantidade de recursos alocados a computação paralela é geralmente constante, contudo em diversos momentos apenas um subconjunto destes recursos estão sendo utilizados pelos processos.

[3] resiliência, sub. fem.: Propriedade de um corpo de recuperar a sua forma original após sofrer choque ou deformação; capacidade de se recobrar facilmente ou se adaptar a mudanças

Sendo assim é possível reduzir a frequência, ou mesmo desligar, os núcleos ociosos, fornecendo margem para aumentar a frequência dos núcleos que estão realizando processamento útil à computação paralela. Contudo a utilização deste tipo de mecanismo oferece problemas para aplicações paralelas que necessitam de um sistema de temporização de alta precisão, tanto para medições de tempo locais como para as globais.

A razão desses problemas são os impactos negativos nas aplicações paralelas, decorrentes dos ruídos introduzidos no sistema computacional pelos mecanismos atuais utilizados para a manutenção de relógio global portanto, é necessário buscar novos mecanismos para a manutenção de relógio global para as aplicações paralelas que executam sobre ambientes computacionais energeticamente eficientes. Mecanismos estes que devem também buscar mitigar o impacto que serviços como o NTP impõem a um sistema de alto desempenho.

Por sua vez, nos ambientes de computação em nuvem utilizados neste trabalho os recursos são fornecidos na forma de computadores lógicos temporários, por exemplo, através de máquinas virtuais. Um computador lógico é uma entidade de processamento que executa um programa escrito para um sistema computacional tradicional, não devendo ser necessário para um usuário deste ambiente conhecer a forma como esse computador lógico é construído.

A existência desta indireção no acesso aos recursos físicos que realizam a computação propriamente dita, contudo, exige que esses computadores lógicos possam ter suas capacidades aferidas e comparadas com sistemas mais tradicionais como forma de fornecer dados para uma melhor utilização dos recursos por parte das aplicações. Felizmente, um computador lógico pode ser visto como um sistema computacional monoprocessado ou com múltiplos núcleos de processamento e com isso alguns dos *benchmark*s que existem para essas classes de sistemas computacionais podem ser aproveitados. Um complicador surge quando um mesmo conjunto de recursos físicos são multiplexados por diversos computadores lógicos, neste caso, faz se necessário observar além do desempenho isolado dos *benchmark*s, como uma combinação deles se comporta quando executa concorrentemente.

Embora exista uma crescente oferta de ambientes computacionais com suporte à alocação de recursos sob demanda, observa-se que os mesmos são construídos de maneira *ad-hoc* de modo que se torna difícil uma comparação imediata entre dois ou mais provedores de serviços de infraestrutura distintos. Um exemplo destas diferenças internas destes ambientes é a técnica de virtualização utilizada. Estes ambientes impõem ainda, novas restrições aos aplicativos que eles executam, restrições estas que devem ser levadas em consideração pelo projetista da aplicação. Entre as restrições específicas dos ambientes de infraestrutura computacionais com suporte à alocação de recursos sob demanda estão, por exemplo, a distribuição dos

computadores lógicos no ambiente físico, a topologia da rede de interconexão e o intervalo de tempo médio para alocação de recursos.

Desta forma, há a necessidade de se buscar novas métricas que permitam avaliar a capacidade de uma dada infraestrutura de reagir às variações de demanda das aplicações, uma vez que esta característica impacta diretamente o desempenho dessas aplicações. Tais características trazem à tona a necessidade da construção de novos *benchmark*s que em conjunto com os existentes permitam ao projetista do sistema computacional identificar como os diversos componentes estão impactando o desempenho geral do sistema.

1.3 Motivações

As motivações desta tese decorrem do contexto previamente apresentado e do resultado de uma ampla pesquisa bibliográfica que não encontrou procedimentos ou mecanismos que permitam lidar com os problemas de sincronização global de relógios físicos para *cluster* de computadores sem a necessidade de etapas de ressincronização ou que viabilizem uma avaliação que explore de forma significativa as diferentes dimensões impostas por essa nova forma de organização dos recursos computacionais.

A pesquisa bibliográfica com relação ao problema de sincronização global de relógios físicos para *cluster*s mostrou que existem trabalhos que buscam reduzir o impacto que o procedimento de ressincronização tem sob a computação paralela [4], contudo, as soluções existentes não eliminam completamente o problema da introdução de ruído causado pelas etapas de ressincronização no sistema computacional. A necessidade de ressincronização é causada pelas imprecisões dos atuais relógios de sistemas que tendem a se desviar ao longo do tempo. Outro problema também identificado nos ambientes computacionais de alto desempenho é o impacto que o ruído causado pelas interrupções de temporização tem sobre as aplicações paralelas [13].

A revisão da literatura mostra que o desempenho de tarefas executando sobre ambientes virtualizados [14] varia de acordo com a tecnologia e as configurações utilizadas [15] [16] [17], sendo esses resultados confirmados nos experimentos realizados nesta tese. Além disso, a indisponibilidade de aplicativos elásticos de código livre que permitam a avaliação sistemática destes ambientes e a observação do crescimento do tráfego de dados multimídia na Internet [18] levantam questionamentos de como esses ambientes computacionais se comportam, particularmente, quando fossem utilizados, por exemplo, para suportar um serviço de distribuição de vídeos.

1.4 Objetivos

Os objetivos desta tese são:

- Projetar e validar um método para avaliação do suporte à elasticidade oferecido pelos ambientes de IaaS, utilizando uma aplicação de VoD elástica como *benchmark*;
- Projetar e avaliar um novo mecanismo para a manutenção de relógios globais para *cluster* de computadores.

Dentro do escopo de ambientes de IaaS, o método proposto nesta tese utiliza uma aplicação elástica para transmissão de vídeo sobre o HTTP. O método utiliza, além do sistema de transmissão, *microbenchmarks* que avaliam a vazão do servidor HTTP nos ambientes virtualizados e a latência de alocação de novas máquinas virtuais. O novo mecanismo para manutenção de relógios globais tem como objetivo reduzir os ruídos de ressincronização impostos pelas atuais técnicas nos ambientes paralelos, mitigando deste modo o impacto que os mecanismos de temporização causam nas aplicações paralelas. A correlação entre as duas propostas decorre do conceito do percentual de tempo que uma aplicação gasta realizando computação útil. Nos ambientes de alto desempenho, este percentual é consequência da troca de mensagens entre as diversas tarefas de uma aplicação, já em ambientes de nuvem é consequência da diferença entre a quantidade de recursos alocados e dos recursos demandados por uma aplicação elástica ao longo do tempo.

1.5 Metodologia

Esta seção descreve sucintamente as metodologias utilizadas para avaliar as propostas desta tese, deixando para os seus respectivos capítulos a discussão aprofundada. A primeira proposta é um novo método e *benchmark* para ambientes de Infraestrutura como serviço (em inglês, *Infrastructure as a Service* (IaaS)) utilizando uma aplicação elástica que provê o serviço de transmissão de vídeo na Internet, descrita em mais detalhes no Capítulo 4. Em seguida, é apresentado um novo mecanismo para a manutenção um de relógio global em um *cluster* de computadores que não utiliza etapas de ressincronização e a introdução do mesmo dentro da biblioteca OpenMPI, ambas as soluções descritas em maiores detalhes no Capítulo 6.

1.5.1 Avaliação de desempenho de ambientes de Infraestrutura como Serviço

Dentro do escopo desta tese, o principal tema estudado é o impacto que a resiliência do ambiente computacional tem sobre as aplicações Web, tanto tradicionais como as com requisitos de tempo real, especificamente serviços de transmissão de vídeo sob demanda. Observe que a elasticidade na gerência dos recursos computacionais, oferecida em um ambiente de IaaS implica porém em um aumento na complexidade dos serviços construídos sobre estes ambientes, que passarão também a ser afetados pela forma como é implementada o ambiente de IaaS. Entre as questões que surgem estão:

- Quais os impactos do *middleware* para gerenciamento de recursos no desempenho das aplicações:

  Causados pelo escalonador de recursos de **IaaS**;

  Causados pelas técnicas de Virtualização;

  Causados pelos mecanismos de gerência da infraestrutura;

- Quais os impactos da topologia da Rede do *Datacenter* no desempenho das aplicações.

1.5.2 Relógio Global em *Cluster* de Computadores

O Relógio Global de Alta Precisão (RGAP) proposto e avaliado nesta tese tem como base os conceitos do relógio de sistemas Relógio Virtual Estritamente Crescente (RVEC), proposto originalmente em [19], que tem a propriedade de ser Estritamente Crescente e Preciso (ECP) [4] com relação a contagem de tempo, que é resultado de a operação do RVEC ser protegida de flutuações de tempo do sistema computacional sem com isso introduzir mais ruído no sistema.

Além disso, o novo mecanismo de temporização global é introduzido na biblioteca OpenMPI passando a fornecer um relógio global para a computação paralela que pode ser acessado através da função $MPI\_Wtime()$, sendo essa proposta avaliada através do *benchmark* para computação POP.

1.6 Contribuições

Em resumo, as principais contribuições presentes nesta tese são:

1. Proposta e desenvolvimento de um novo *benchmark* para ambientes de IaaS;

2. Avaliação deste novo *benchmark* em ambientes reais de IaaS privados;

[4] Um relógio do sistema é aderente a propriedade ECP se este garante que duas leituras consecutivas a este relógio, T_1 e T_2, retornarão medições de tempo $T_2 > T_1$ para qualquer intervalo de tempo entre as duas leituras.

3. Proposta de um novo método para comparação de ambientes de IaaS baseadas no novo *benchmark*;

4. Avaliação aprofundada utilizando *Global Positioning System* (GPS) da estabilidade em longos intervalos de tempo do circuito TSC;

5. Desenvolvimento e avaliação de um mecanismo para sincronização física de relógio global construído com base no RVEC, o RGAP;

6. Desenvolvimento e avaliação de um protótipo do RGAP integrado a biblioteca OpenMPI;

7. Avaliação experimental preliminar das diferenças no desempenho do POP entre o uso do RGAP e do NTP como serviços de sincronização de tempo para a biblioteca OpenMPI.

1.7 Organização da Tese

Esta tese apresenta no Capítulo 2 os conceitos básicos que permeiam o projeto mecanismos de sincronização de relógios físicos e o impacto destes mecanismos nos ambientes computacionais de alto desempenho. O Capítulo 3 trata dos conceitos centrais aos ambientes de computação em nuvem, em especial os ambientes de IaaS, com uma introdução aos conceitos de elasticidade e resiliência para estes ambientes computacionais. Ainda, o Capítulo 3 apresenta uma breve revisão de alguns dos métodos existentes para avaliação de desempenho de diferentes ambientes computacionais com ênfase no uso de *benchmark*s, e alguns dos conceitos que permeiam o projeto de serviços Web escaláveis, em especial sistemas de VoD. O Capítulo 4 apresenta o novo *benchmark* proposto, e constrói sobre este um método para avaliação de ambientes de IaaS e formalizar dentro do escopo deste trabalho, os conceitos de elasticidade e resiliência. O Capítulo 5 apresenta os resultados da avaliação empírica do mesmo. O Capítulo 6 apresenta o novo mecanismo para manutenção de relógios globais desenvolvido nesta tese, assim como formaliza suas premissas e limitações. Os experimentos utilizados na validação da proposta de relógio global são apresentados no Capítulo 7, enquanto que o Capítulo 8 contém alguns dos trabalhos relacionados com os conteúdos desta tese. O Capítulo 9 resume os principais resultados obtidos, apresenta as conclusões da tese e propõe alguns trabalhos futuros.

Capítulo 2

Revisão da Sincronização Global de Relógios Físicos

Este capítulo tem como objetivo revisar conceitos básicos, definições e questões envolvidas nesta tese dentro do escopo de Sincronização Global de Relógios Físicos. Ele também discute os impactos que a manutenção dos relógios físicos tem sobre as aplicações que executam nos ambientes computacionais de alto desempenho. Após, são feitas algumas considerações finais, encadeando os conteúdos abordados de forma a delimitar o escopo da pesquisa.

2.1 Aferindo o Tempo em Computadores Digitais

Nos computadores digitais modernos existem atividades que necessitam ser sincronizadas para seu correto funcionamento, sendo que diferentes circuitos digitais podem ser utilizados para este fim, os relógios do *hardware*. Além de sincronização, esses circuitos também são utilizados para medir a passagem do tempo de relógio. Com isso os diversos relógios diferem em como a informação é acessada, a precisão do valor lido e o custo de acesso [20].

É importante ter em mente a diferença entre a precisão da informação que pode ser obtida e o custo para obtê-la. A precisão leva em consideração apenas o número de casas decimais que o circuito do relógio oferece, ou seja, se o relógio envia um sinal a uma frequência de 100 MHz, a maior precisão que pode ser obtida por este relógio é 10 ns. O custo de acesso ao relógio é o tempo passado desde que a informação é requerida até esta encontrar-se disponível para a aplicação.

A arquitetura x86 possui seis circuitos de relógio [21], são eles:

- Relógio de Tempo-Real (*Real-Time Clock*);

- Relógio de Intervalo Programável (*Programmable Interval Timer*);
- Relógio Local do Processador (*Local Advanced Programmable Interrupt Controller*);
- Temporizador da Interface Avançada de Configuração e Potência (*Advanced Configuration and Power Interface Timer*);
- Temporizador de Eventos de Alta Precisão (*High Precision Event Timer*);
- Contador de Ciclos do Processador (*Time Stamp Counter*).

2.1.1 Relógio de Tempo-Real

O Relógio de Tempo-Real em inglês, *Real-Time Clock* (RTC), é o *hardware* para controle do tempo mais comum nos computadores digitais. Todos os computadores que seguem a arquitetura x86 devem possuir este relógio. O circuito que implementa o RTC não possui dependências com o processador ou outros circuitos de controle na placa-mãe, sendo integrado junto com o circuito CMOS RAM (BIOS) e alimentado por uma bateria.

O RTC funciona emitindo periodicamente interrupções de hardware na IRQ 8, sendo que essas interrupções podem ser enviadas com taxas que variam entre 2 Hz a 8.192 Hz (8 KHz). Sendo assim, o intervalo de tempo que pode ser medido com precisão de aproximadamente 122 μs [22].

2.1.2 Relógio de Intervalo Programável

Os computadores que seguem a arquitetura x86 contém também o circuito digital do Relógio de Intervalo Programável (*Programmable Interval Timer* ou PIT). Ele é implementado por um chip NMOS Intel 8253 (ou CMOS no caso do Intel 8254) [23].

O 8253 possui três relógios todos implementados com um contador de 16 *bits*. O primeiro é temporizador utilizado pelo sistema operacional (o Linux, por exemplo, configura para emitir interrupções a 1000 Hz na arquitetura x86) [22]), o segundo para controle de atualização das memórias RAM e o terceiro para a saída de áudio padrão (PC *Speaker*).

A frequência de trabalho máxima (taxa da geração de interrupções) é dada pelo oscilador interno do circuito, que é de 1.193.182 Hz para o 8253 e a frequência mínima de trabalho é de 18.2 Hz [24]. No 8254 a frequência de trabalho máxima é 10 MHz [25].

2.1.3 Relógio Local do Processador

Presente nos processadores recentes da família x86, o Relógio Local do Processador (*Local Advanced Programmable Interrupt Controller* - APIC) [26, 27], é um circuito de funcionamento similar ao PIT, porém a visibilidade das interrupções é restrita ao processador onde estas são geradas.

Os contadores deste relógio são registradores de 32 *bits* contra os 16 *bits* do PIT o que lhes c gerar interrupções com baixíssima frequência. Isso é feito durante a inicialização dos contadores do circuito, assim a interrupção somente é emitida para o processador quando o contador chegar a zero. A frequência máxima de atualização do contador é dada pelo barramento (ex.,100 MHz) e frequências menores podem ser obtidas esperando-se até 2^n ciclos do barramento (n pode variar de 0 a 7) [22] para emitir uma interrupção.

2.1.4 Temporizador da Interface Avançada de Configuração e Potência

O Temporizador da Interface Avançada de Configuração e Potência em inglês, *ACPI Power Management Timer* (ACPI PMT), é encontrado em placas-mães que suportam o mecanismo de controle de potência (*Advanced Configuration and Power Interface* - ACPI). Ele é composto por um contador de 32 *bits* que é incrementado a uma frequência fixa de $3,579545\ MHz$ [28]. A grande vantagem deste circuito é sua independência com relação aos controles de potência implementados nos computadores atuais, o que o torna atraente nas situações onde o processador pode modificar sua frequência de trabalho [22].

2.1.5 Temporizador de Eventos de Alta Precisão

O Temporizador de Eventos de Alta Precisão [1] em inglês, *High Performance Event Timer* (HPET), é um circuito para contagem de tempo desenvolvido pela Intel em parceria com a Microsoft. Como este circuito possui diversos contadores independentes o sistema operacional pode vincular um contador específico para cada aplicação em nível de usuário.

Seus requisitos de desenvolvimento foram especificados para prover uma granularidade de milissegundos nas medidas e precisão de nanosegundos. O controle do HPET é exclusividade do sistema operacional sendo este o responsável por salvar e restaurar as informações necessárias para o seu correto funcionamento quando o computador entra em um estado de baixo consumo.

O HPET funciona utilizando registradores mapeados na memória pela BIOS, sendo que este mapeamento é estabelecido durante a inicialização do computador

sendo posteriormente informado ao sistema operacional [22].

2.1.6 Contador de Ciclos do Processador

O Contador de Ciclos do Processador em inglês, *Time Stamp Counter* (TSC), é um registrador de 64 *bits* na arquitetura x86 que armazena o número de ciclos desde a inicialização do sistema. A lógica de atualização do contador incrementa em uma unidade o valor do registrador a cada ciclo da CPU, desta forma a frequência de atualização do contador é dada diretamente pela frequência de trabalho da CPU.

Devida a sua alta taxa de atualização e largura no número de bits, dentre os circuitos que são encontrados nas arquiteturas x86, o TSC é o que oferece a maior precisão. O TSC, ao contrário dos outros relógios da arquitetura x86, não gera interrupções que possam ser utilizadas para indicar a passagem do tempo, esta somente pode ser aferida executando a instrução RDTSC [29–32], oferecendo custo de acesso inferior aos outros circuitos.

Apesar das vantagens do TSC para temporização das aplicações, como sua frequência de atualização depende da atual de frequência de operação do processador, ele acaba por fornecer valores inconsistentes quando é utilizado em ambientes computacionais sujeitos a escalonamento de frequência (DVFS). Ambientes computacionais que utilizam máquinas multiprocessadas podem não conseguir garantir o correto funcionamento do TSC entre os diversos núcleos e processadores. O *kernel* do Linux, por exemplo, realiza verificações para evitar o uso deste circuito caso ele não esteja sincronizado entre os núcleos.

2.2 Temporizando Aplicações em um *Cluster* de Computadores

Esta seção descreve o problema de temporização global em *clusters* de computadores e como sob condições específicas as técnicas de sincronização global atuais resolvem o problema de temporização global descrito a seguir. Essas soluções são baseadas na sincronização global entre os nós do *cluster* e a chamada de sistema $gettimeofday()$ [33] ou $clock\_gettime()$.

2.2.1 Formalização do Problema

Considere dois nós computacionais, N_A e N_B, interconectados entre si e ambos executando a mesma versão do *kernel* do sistema operacional. Seja P_0 um processo

de uma aplicação paralela executando em N_A. P_0 envia uma mensagem para P_1 que encontra-se executando em N_B, o qual possivelmente já possui outros processos executando. Seja C_{P_n} a informação de tempo vinculada ao processo P_n, onde a informação contida em C_{P_n} é similar à do registrador TSC na arquitetura x86. Logo, utilizando dois valores consecutivos de C_{P_n}, é possível saber o intervalo de tempo entre esses valores. O problema é que, se C_{P_n} armazenar o mesmo valor que TSC, esta informação não é mais útil quando P_n realiza uma troca de mensagens com um processo executando em outro nó, já que o valor do TSC lido em N_A não tem significado em N_B.

Por exemplo, assuma que existe uma computação paralela onde o processo P_0 executa em N_A e que este troca mensagens com um processo P_1 executando em N_B: por hipótese temos que os processadores de N_A e N_B trabalham na mesma frequência e onde N_B é ligada depois de N_A. Sendo que, a aplicação paralela necessita de temporização global e que o *timestamp* é enviado nas mensagens, onde este *timestamp* é o resultado retornado pela instrução **RDTSC** ou outro relógio local. P_1 calcula a passagem do tempo decorrido na transmissão da mensagem enviada por P_0 utilizando o valor retornado pela leitura de C_{P_1}, que retornará o valor do TSC de N_B que é menor que o valor retornado pela leitura feita em N_A. Desta forma, o cálculo do intervalo de tempo resultará em um valor negativo, ou seja, absurdo [34].

É importante salientar que a principal propriedade que qualquer solução para este problema deve buscar garantir o comportamento estritamente crescente de C_P.

2.2.2 Network Time Protocol

O Protocolo de Tempo para Redes em inglês, *Network Time Protocol* (NTP), é um padrão do IETF [3, 35] para sincronização de nós na Internet. O NTP [3] versão 3 proposto em 1992 é uma aplicação cliente/servidor hierárquica que distribui o tempo universal (UTC), realizando sincronizações sobre o relógio local. O protocolo NTP executa como um serviço nos clientes, que periodicamente contactam um servidor NTP para atualizar seu relógio local baseando-se nas informações de tempo passadas pelo servidor. A troca de mensagens entre o servidor e o cliente NTP gera um valor Δ que é aplicado sobre o relógio local.

Os servidores NTP são organizados como em uma árvore (Figura 2.1), onde a raiz de uma árvore NTP é conhecida como servidor de estrato 0 e representa o relógio universal. O maior estrato possível é o 16 que representa um servidor que não esta respondendo. Os níveis de 1 a 15 representam os nós onde os clientes podem se conectar para sincronizar seus relógios. Em teoria, baseando-se na hierarquia apresentada, nós com estrato menores oferecem uma informação mais acurada.

Um cliente NTP pode se conectar a qualquer nó da árvore com exceção da raiz,

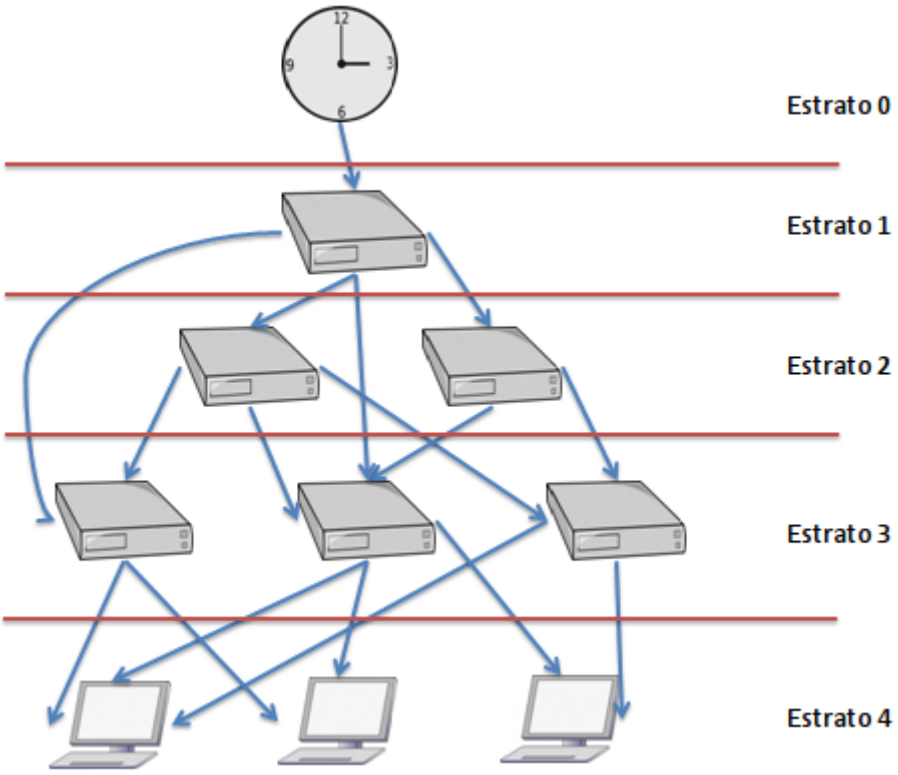


Figura 2.1: Organização do protocolo NTP

sendo que os parâmetros, em geral, mais relevantes no processo de seleção são os retardos de rede e carga do servidor. Uma vez que o servidor aceita a conexão do cliente algumas mensagens serão trocadas e, se necessário, o cliente atualizará seu relógio local.

Em uma primeira análise, o NTP consegue solucionar o problema de temporização descrito anteriormente nesta seção, uma vez que todos os computadores têm seus relógios sincronizados com a mesma fonte. Contudo, todos os relógios desviam inevitavelmente de UTC e com isso será possível que duas chamadas consecutivas em um mesmo nó da função $gettimeofday()$, onde a primeira chamada retorna $T1$, e segunda $T2$ tal que $T1 > T2$. Esta situação deve-se ao fato do NTP poder atualizar o relógio local para um tempo passado. Uma situação semelhante pode ocorrer em uma migração entre nós onde o destino possui uma referência de tempo mais antiga devido ao NTP.

O NTP usualmente viola a propriedade monotonicamente crescente quando a granularidade da medida é inferior a dezenas de milissegundos, de acordo com medidas feitas na rede do LCP/COMPASSO (Apêndice A). O problema ocorre quando os clientes encontram-se carregados e não conseguem executar o serviço do NTP com a frequência necessária. Nestes casos os desvios podem chegar a dezenas de segundos [36] e não mais milissegundos, resultado corroborado por Zhang e Schulz-

rinne [37], sendo que Hong et al. [38] apresentam um estudo mais detalhado de outras possíveis causas deste problema.

2.2.3 *Precision Time Protocol*

O *Precision Time Procol* (PTP) [39] é um padrão IEEE (IEEE 1588 e IEEE1588-2008 [40]) para sincronização de sistemas de tempo real críticos. O funcionamento básico deste algoritmo é que o relógio mais preciso dentro do *cluster* sincroniza todos os outros. O protocolo funciona utilizando dois tipos de relógios, mestre e escravo, onde inicialmente qualquer relógio pode exercer tanto a função de mestre quanto a de escravo. Neste protocolo, os relógios são classificados com relação a sua precisão em classes (estrato no NTP), onde a maior classe é o relógio atômico que possui estrato 1. Uma vez que o relógio mestre tenha sido definido, todos os outros relógios tornam-se escravos e devem determinar o deslocamento ($d(t)$) existente entre seu relógio local e o relógio mestre (Equação 2.1), onde $s(t)$ representa o tempo medido no relógio do escravo no instante t e $m(t)$ representa o tempo medido no relógio do mestre no mesmo instante t.

$$d(t) = s(t) - m(t) \tag{2.1}$$

O protocolo funciona através do envio periódico de mensagens a partir do mestre para os escravos. Desta forma, os dispositivos escravos podem recalcular o deslocamento com relação ao relógio mestre que tende a desviar com o tempo (*drift*). O PTP assume duas premissas para seu correto funcionamento. A primeira é que as mensagens são trocadas em intervalos pequenos de forma que o deslocamento aplicado sobre o relógio escravo pode ser considerado constante e a segunda é que a rede onde as mensagens são trocadas é simétrica. Desta forma, a precisão do IEEE 1588 depende do grau com o qual essas premissas são garantidas.

A precisão da sincronização é altamente dependente da rede e dos componentes utilizados na construção desta. O PTP é baseado em comunicação IP *multicast*, não sendo restrito apenas a *Ethernet*, mas podendo ser utilizado com qualquer padrão de rede que suporte *multicast*. Sendo que os serviços que implementam o protocolo executam como processos de baixa prioridade e o protocolo consegue uma precisão da ordem de $100\mu s$ para implementações totalmente em software, podendo melhorar esta precisão até $10\mu s$. Como o IEEE 1588 é executado como um serviço nos computadores do *cluster*, se o nó encontra-se sobrecarregado o PTP não conseguirá executar com a frequência necessária para manter o sistema sincronizado, similar ao que ocorre no NTP.

2.2.4 Limitações no projeto de Relógio Global em *cluster* de computadores

As soluções que necessitam de interação com *software* para manter os nós sincronizados não conseguem garantir a desigualdade descrita na Equação 2.2, para a condição descrita pela Equação 2.3 abaixo, quando sincronização global é utilizada. Isso ocorre principalmente pelo fato de os serviços em *software* não conseguirem executar com a frequência mínima necessária.

$$T1 < T2 \tag{2.2}$$

$$\lim_{(\forall T2, T1 | T2 > T1)} (T2 - T1) \longrightarrow 0 \tag{2.3}$$

As soluções puramente em *hardware* conseguem garantir a propriedade ECP, porém, num ambiente muito mais restrito, as soluções em *software* garantem a propriedade de crescimento estritamente crescente do tempo nos casos onde o intervalo entre consultas ao relógio de sistema é grande o suficiente para esconder as violações da propriedade ECP do relógio.

2.3 Sistemas computacionais energeticamente eficientes

O consumo de energia, as limitações na capacidade de resfriamento e as taxas de falhas previstas em sistemas *Exascale* [41], tornam muito atrativo o suporte do sistema operacional a mecanismos de *hotplug* para unidades de processamento [42] [43] e para memória principal [44]. Mecanismos de *hotplug*, fornecem meios para a troca ou inclusão de novos componentes com o sistema computacional em operação e já não mais são recursos disponíveis apenas em *mainframes*. Por exemplo, os processadores Xeon da serie 7500 oferecem suporte ao *hotplug* do processador [45], enquanto que a arquitetura big.LITTLE da ARM permite alterar dinamicamente o tipo de núcleo de processamento que está em uso. A disponibilidade de recursos *hotplug* nos processadores Xeon da serie 7500 [45] viabiliza o projeto de sistemas computacionais mais energeticamente eficientes, expandindo trabalhos como o de Barroso e Holzle [46] e Subramaniam e Feng [47] que tratam de computação energeticamente proporcional (do inglês Energy-proportional Computing).

Os ambientes de *Exascale* obrigam também uma reavaliação dos sistemas operacionais e *runtimes* para estas máquinas [48]. Os trabalhos FusedOS [49] e mOS [50] apresentam propostas de sistemas operacionais para máquinas *Exascale*. Park et al. [49], apresentam o protótipo do FusedOS, um sistema operacional baseado em

Linux que busca oferecer as funcionalidades de um *kernel* completo, ou em inglês *Full-Weight Kernel* (FWK), enquanto oferece desempenho computacional equivalente a um *kernel* para nós computacional de máquinas paralelas, também conhecido como *Light-Weight Kernel* (LWK). No FusedOS, os núcleos utilizados em uma computação paralela não executam código do sistema operacional Linux e sim uma biblioteca de computação (*Compute Library*) que encapsula as funcionalidades existentes no *Compute Node Kernel* (CNK). O ambiente computacional utilizando na validação do FusedOS é o Blue Gene/Q. Wisniewski et al. [50], apresentam outra proposta para execução simultânea de FWK e LWK sobre o mesmo nó computacional. O mOS particiona o nó computacional isolando os recursos a serem gerenciados pelo LWK evitando assim a disputa no acesso aos mesmos e com isso a introdução de ruídos na computação paralela. A comunicação entre os *kernel*s ocorre através de memória compartilhada, passagem de mensagens ou interrupções entre processos.

2.4 Ruídos de sistemas nos ambientes computacionais de alto desempenho

As aplicações paralelas mesmo quando executam em ambientes computacionais de alto desempenho como o Blue Gene/Q [51] sofrem impactos das diferentes fontes de ruídos existentes no sistema computacional [52]. A sensibilidade aos ruídos do sistema é consequência de como os processos da aplicação paralela trocam dados, da capacidade de rede do ambiente computacional e da própria capacidade de processamento do ambiente [53]. O impacto dos ruídos de sistema na execução das aplicações paralelas tende a crescer junto com o tamanho do sistema computacional, o que faz com que o desempenho esperado da execução de uma aplicação paralela possa não ser alcançado [54]. Sendo ainda importante salientar que o ruído provocado pelas interrupções de temporização impactam negativamente estes ambientes [13].

2.4.1 *Parallel Ocean Program*

O *Parallel Ocean Program* (POP) [55] é um programa de simulação oceânica desenvolvido pelo *Los Alamos National Laboratory* e utilizado em pesquisas nas áreas de oceanografia e previsão do clima. Ele realiza simulações oceânicas, recebendo variações da atmosfera como condições de contôrno. Devido à combinação da alta resolução exigida na solução dos vórtices oceânicos e simulações de longos intervalos de tempo, a demanda de poder computacional do POP cresce rapidamente. A grande demanda por poder computacional faz com que o POP tenha sido concebido para operar de forma eficiente em uma ampla variedade de ambientes computacionais. Nas mais recentes versões do POP, um esquema de decomposição de dados

flexível foi introduzido para aumentar ainda mais a portabilidade do modelo. Devido a sua demanda por poder computacional, a facilidade de configuração dos cenários de execução e a sua grande importância como aplicação cientifica, a comunidade de alto desempenho utiliza o POP como *benchmark* na avaliação de novas propostas para ambientes de computação paralela, ou na avaliação de ambientes computacionais paralelos existentes [7]. O POP é por sua vez uma aplicação paralela que demonstra alta sensibilidade aos ruídos introduzidos por *daemon* de sistema [6].

2.5 Escopo

O problema de Sincronização Global de Relógios Físicos é avaliado dentro do escopo desta tese sobre um ambiente computacional energeticamente eficiente. Sendo que dentro desta tese um ambiente computacional energeticamente eficiente é formado por *cluster* de computadores onde os nós são formados por múltiplas unidades de processamento, estas compostas por diversos núcleos de processo, e que utiliza de mecanismos como *Dynamic Voltage and Frequency Scaling* (DVFS), desligamento de unidades de processamento (hotplug no Xeon e ARM big.LITTLE). Dentro deste escopo os ruídos de sistemas impactam negativamente na execução das aplicações. Esses efeitos esse que tende a ser exacerbados nos ambientes computacionais de alto desempenho sejam esses dedicados como no Blue Gene/Q e ainda mais amplificados nos *cluster* de computadores comumente encontrados em instituições acadêmicas e empresas.

Capítulo 3

Revisão da Avaliação de Desempenho de ambientes de Computação em Nuvem

Este capítulo tem como objetivo revisar conceitos básicos, definições e questões envolvidas nesta tese dentro do escopo da Avaliação de Desempenho de os ambientes de *Infrastructure as a Service* (IaaS). Ele também apresenta o modelo de um sistema de distribuição de vídeo sob demanda e os questões que motivaram esta tese. Após, são feitas algumas considerações finais, encadeando os conteúdos abordados de forma a delimitar o escopo da pesquisa.

3.1 Avaliação de Desempenho de Ambientes Computacionais

O processo de inovação dos ambientes computacionais tem sido dirigido através de avaliações quantitativas das novas funcionalidades introduzidas no dado ambiente computacional. Tais avaliações são realizadas através de *benchmark*s que buscam aferir quantitativamente o impacto que a alteração proposta pela nova funcionalidade irá ter no desempenho de um dado ambiente computacional. Esta abordagem quantitativa impulsiona a construção de *benchmark*s para a avaliação de ambientes computacionais monoprocessados e multiprocessados. Deve-se ter em mente ainda que os *benchmark*s são utilizados tanto pelos projetistas de computadores quanto pelo usuário final do sistema computacional. Os projetistas os utilizam durante a etapa de projeto para quantificar o impacto que diferentes soluções ou funcionalidades têm sobre o ambiente de interesse, e após o sistema ter sido construído, para avaliar se as predições feitas durante o projeto se confirmaram. O usuário final por

sua vez deseja quantificar as vantagens que este irá obter ao modificar seu ambiente computacional atual [56].

3.1.1 Avaliação de Desempenho de Ambientes Computacionais Monoprocessados

O ***S**tandard **P**erformance **E**valuation **C**orporation* (SPEC) é um consórcio sem fins lucrativos, com a missão de desenvolver *benchmark*s de nível de sistema para múltiplos sistemas operacionais e arquiteturas, *benchmark*s esses que são derivados de aplicações reais [57]. A primeira suíte de *benchmark*s SPEC é de 89 [20] e ao longo dos últimos anos os *benchmark*s **SPEC CPU** tem sido utilizados amplamente por fabricantes e pela academia. A suíte é continuamente atualizada de modo a refletir os avanços nos ambientes computacionais, tais como desempenho da memória cache, da memória principal e arquitetura interna do processador (preditor de desvio e *pipeline*).

Tabela 3.1: Aplicações de processamento de inteiros da suíte SPEC (SPECint)

Aplicação	Descrição
400.perlbench	Linguagem de programação
401.bzip2	Compressão de dados
403.gcc	Compilador
429.mcf	Otimização combinatória
445.gobmk	Inteligência artificial
456.hmmer	Busca de sequencias de genes
458.sjeng	Inteligência artificial
462.libquantum	Computação quântica
464.h264ref	Compressão de vídeo
471.omnetpp	Simulação de eventos discretos
473.astar	Busca de menor caminho
483.xalancbmk	Processamento de XML

A escolha em usar códigos derivados de aplicativos reais tem como objetivo permitir que tanto projetistas de computadores como usuários tomem suas decisões com base em cargas de trabalho realistas. A suíte de *benchmark* SPEC é subdivido em dois grupos - o SPECint é composto pelas aplicações de processamento de inteiros, listadas na Tabela 3.1, enquanto que o SPECfp contem as aplicações de ponto flutuante, listadas na Tabela 3.2.

Contudo, apesar dos *benchmark*s serem utilizados de forma ubíqua na pesquisa em ambientes computacionais monoprocessados, nem sempre essa utilização é feita de forma crítica, ou seja, é realizada uma avaliação por parte dos pesquisadores de forma a determinar se efetivamente o *benchmark* escolhido é o mais apropriado aos experimentos feitos. A utilização de resultados obtidos através de um *benchmark*

Tabela 3.2: Aplicações de processamento de ponto flutuante da suíte SPEC (SPECfp)

Aplicação	Descrição
410.bwaves	Dinâmica de fluidos 3D
416.gamess	Química quântica
433.milc	Cromodinâmica quântica
434.zeusmp	Simulação de CD
435.gromacs	Dinâmica molecular
436.cactusADM	Simulação de Física relativística
437.leslie3d	Dinâmica de fluidos 3D
444.namd	Simulação de dinâmica molecular
447.dealII	Análise de elementos finitos
450.soplex	Programação Linear
453.povray	Ray-tracing
454.Calculix	Calculo de mecânica estrutural 3D
459.GemsFDTD	Simulação 3D das equações de Maxwell
465.tonto	Simulação de química quântica
470.lbm	Simulação de fluidos 3D com o Lattice-Boltzmann
481.wrf	Predição meteorologia
482.sphinx3	Reconhecimento de voz

que não realize uma avaliação ampla do problema estudado pode oferecer resultados questionáveis [58]. O SPEC é um exemplo do método de avaliação de desempenho que utiliza aplicações sintéticas (derivadas de aplicações reais). Além deste método existem outros métodos para avaliação de desempenho dos sistemas monoprocessados, como modelagem analítica, simulação, uso de mistura de instruções e *kernels* de aplicações (trechos de códigos de aplicações reais).

3.1.2 Avaliação de Desempenho de Ambientes Computacionais Multiprocessados

O procedimento de avaliação de desempenho em ambientes computacionais multiprocessados deve por sua vez levar em consideração como se comporta o desempenho destes quando se aumenta o número de núcleos de processamento, ou seja, como o poder computacional de um dado ambiente escala com acréscimo de novos núcleos. Duas suítes de *benchmark*s muito utilizadas são o **SPLASH-2** [59] e o LINPACK [60] baseados em aplicações cientificas, e o PARSEC [11], baseado em aplicações de processamento multimídia e comércio eletrônico.

SPLASH-2

Os trabalhos em ambientes multiprocessados, assim como ocorre nos casos de ambientes monoprocessados, utilizam aplicativos reais como *benchmark*s de modo a

permitir a avaliação quantitativa de novas ideias e o impacto que estas têm sobre o ambiente estudado [56]. Nos ambientes paralelos o ***S**tanford **P**aralle**L** **A**pplications for **SH**ared memory* (**SPLASH**) [10] viabiliza a avaliação quantitativa das diferentes características destes ambientes e tendo em vista a sistematização da suíte esta permite um maior grau de comparabilidade entre estudos distintos.

Tabela 3.3: Aplicações da suíte **SPLASH-2**

Aplicações	Descrição
Barnes	Simulação de corpos (galáxias ou partículas)
Cholesky	Fatoração de Cholesky
FFT	Processamento de Sinal
FMM	Simulação de corpos (galáxias ou partículas)
LU	Fatoração de LU
Ocean	Simulação de movimento dos oceanos
Radiosity	Equilíbrio de luminosidade
Radix	Ordenação através do Radix
Raytrace	Renderização usando traçado de raios
Volrend	Renderização usando traçado de raios
Water-Nsq	Simulação de moléculas de água
Water-Sp	Simulação de moléculas de água

A suíte de *benchmark*s **SPLASH-2** [59] é uma atualização do trabalho original [10], buscando ampliar a cobertura dos programas *multithreaded* com relação à computação científica. Além disso, também se fez necessário atualizar as implementações dos programas de modo a melhorar a interação deles como as inovações encontradas nos sistemas de memória e lidar com máquinas paralelas com um número maior de núcleos de processamento. O SPLASH-2 contém 12 *benchmark*s, listados na Tabela 3.3, que representam uma grande variedade de computações científicas, de engenharia e gráficas. Os aplicativos que compõem a suíte são caracterizados em quatro eixos:

- Concorrência e balanceamento de carga: Indicativo de escalabilidade do programa, assumindo sistema de memória e comunicação perfeitas;
- Tamanho da entrada: Indicativo de localidade temporal usado para verificar se uma entrada cabe na memória cache;
- Razão entre comunicação computação e tráfego: Avalia impacto potencial da latência de comunicação sobre o desempenho;
- Localidade espacial: Avalia o impacto que uma organização de cache tem sobre o ambiente, ex., falso compartilhamento.

PARSEC

A suíte ***P**rinceton **A**pplication **R**epository for **S**hared-m**E**mory **C**omputers* (PARSEC) [61] de *benchmark*s, é projetada para o estudo e avaliação quantitativa de processadores com múltiplos núcleos de processamento, ou *multicores*. Ao contrário do **SPLASH-2**, ela não possui um conjunto de programas que tende para o lado da computação de alto desempenho, mas oferece um conjunto de aplicações emergentes como *data mining*, processamento de imagens e aplicativos de sistemas comerciais. Além disso, o conjunto de aplicativos e *kernel*s que compõem o PARSEC mostram ser significativamente diferentes tanto dos da suíte SPLASH-2, como entre si. Em especial com relação a processadores com múltiplos núcleos o PARSEC demonstra uma maior variabilidade com relação a características dos aplicativos que o SPLASH-2 [11]. A Tabela 3.4 apresenta os aplicativos e que fazem parte da suíte de *benchmark*s PARSEC, é possível ver na segunda coluna de tabela que os aplicativos desta suíte cobrem uma amplitude maior das aplicações executadas em ambientes multiprocessados.

Tabela 3.4: Aplicações da suíte PARSEC

Aplicações	Descrição
blackscholes	Simulação do mercado de ações
bodytrack	Visão computacional (rastreamento do corpo humano)
canneal	*cache-aware Simulated Annealing*
dedup	Compressão através de deduplicação
facesim	Simulação Facial Realística
ferret	Busca de semelhaça em imagens
fluidanimate	Simulação de dinâmica de fluidos
freqmine	Mineração de dados
raytrace	Renderização usando traçado de raios
streamcluster	Mineração de dados - *online clustering problem*
swaptions	Simulação do mercado de ações com o método de Monte Carlo
vips	Processamento de Imagens
x264	Codificação de vídeo

3.2 Máquinas Virtuais e Técnicas de Virtualização

As tecnologias de virtualização encontram-se difundidas em diversos níveis dos sistemas digitais. Nos últimos anos, o crescente uso de máquinas virtuais decorre da disponibilidade cada vez maior de poder computacional dos processadores com

múltiplos núcleos e com o suporte, nestes novos processadores, para tecnologias de virtualização como as arquiteturas *Intel Virtualization Technology* (Intel VT) [62, 63] e *AMD Virtualization* **Pacifica** [64].

Dentro do escopo desta tese, máquinas virtuais e técnicas de virtualização aparecem como uma principal motivação do problema estudado. A razão é que, dentro do escopo deste trabalho os recursos computacionais disponíveis são organizados como máquinas virtuais e disponibilizados aos usuários do sistema computacional. Assim sendo, é importante entender esses mecanismos, como eles podem ser úteis na construção de sistemas computacionais e como afetam as aplicações de transmissão de vídeo e científicas.

3.2.1 Conceitos Gerais

Um dos conceitos básicos em computação é o uso de abstrações para esconder do desenvolvedor as complexidades do *hardware*. O uso de múltiplos níveis de abstração pretende criar, em cada nível, máquinas virtuais que estendem as funcionalidades disponíveis no nível abaixo, construindo desta forma uma hierarquia de abstrações [20].

Além das funcionalidades estendidas, uma máquina virtual deve criar isomorfismos entre as funcionalidades que ela oferece ao nível acima e o que pode ser executado na máquina virtual do nível abaixo. Dentro do escopo desta tese, virtualização é definida como a construção de um mapeamento entre o dispositivo virtual (funcionalidade) e o real [65].

3.2.2 Virtualização Total

A classe de Máquinas Virtuais que implementam a Virtualização Total oferece para as aplicações que executam sobre essa máquina virtual um *hardware* idêntico ao que é fornecido pela máquina física. Nela, existe uma camada de *software* ou *firmware*, que é responsável por gerenciar os recursos físicos e exportar para as camadas acima (SO e aplicações) uma interface igual a do *hardware* [66, 67]. Este modelo de virtualização pode ser implementado com auxílio da arquitetura do processador ou via tradução binária [67].

O VM/370 [66], apresentado na Figura 3.1, é um sistema operacional desenvolvido pela IBM em 1972. O ambiente computacional disponibilizado pelo VM/370 permitia a execução de múltiplos sistemas operacionais sobre um mesmo hardware. Entre os diferentes sistemas operacionais utilizados no ambiente VM/370 o CP (*Control Program*) e o CMS (*Conversational Monitor System*) possuem maior destaque. O CP é o sistema operacional responsável por criar a ilusão de múltiplas máquinas físicas dedicadas e este tipo de software veio a ficar conhecido como Monitor de

Máquina Virtual (MMV). O CPS, por sua vez, é o sistema operacional executado pela máquina virtual exposta aos usuários e onde as aplicações executam. O ambiente computacional VM/370 também utiliza outros sistemas operacionais executando dentro de máquinas virtuais para executar subsistemas como o de E/S.

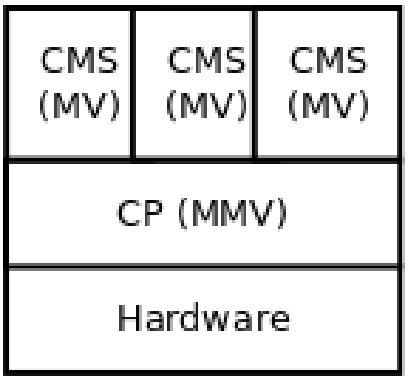


Figura 3.1: Organização de um sistema usando o **VM/370**

Kernel-based Virtual Machine

Com a chegada ao mercado de processadores da arquitetura x86 que possuem suporte em hardware para a virtualização [62, 63], é implementado no *kernel* do Linux um MMV, o ***K**ernel-based **V**irtual **M**achine* (KVM) [68]. O uso das extensões de virtualização dos novos processadores permite que uma instrução que viola as regras impostas pelos mecanismos de virtualização seja executada, sendo que a execução destas instruções causa uma interrupção que deve ser tratada pelo monitor. O **KVM** é um subsistema do Linux que se utiliza do suporte do *hardware* disponível nestes processadores para construir um monitor de máquina virtual, sendo que o suporte de entrada/saída (E/S) é realizado através **QEMU** [69]. O KVM é implementado como um módulo no *kernel* do Linux tradicional, que realiza a execução de uma máquina virtual através da criação de *threads* dentro do *kernel* e uma instância em modo usuário do QEMU.

A Figura 3.2 mostra uma representação de um sistema computacional utilizando o KVM como mecanismo de virtualização, sendo possível observar os três modos de operação utilizados. Ainda na Figura 3.2, os retângulos vermelhos escuros representam as aplicações dos usuários do sistema computacional, o grande retângulo azul representa o *kernel* do sistema operacional *host* (ex. Linux) que é executado em modo núcleo e os retângulos vermelhos claros representam duas máquinas virtual executando em modo virtual. Os processadores que oferecem este suporte implementam um novo modo de trabalho, o modo Virtual, onde as máquinas virtuais podem

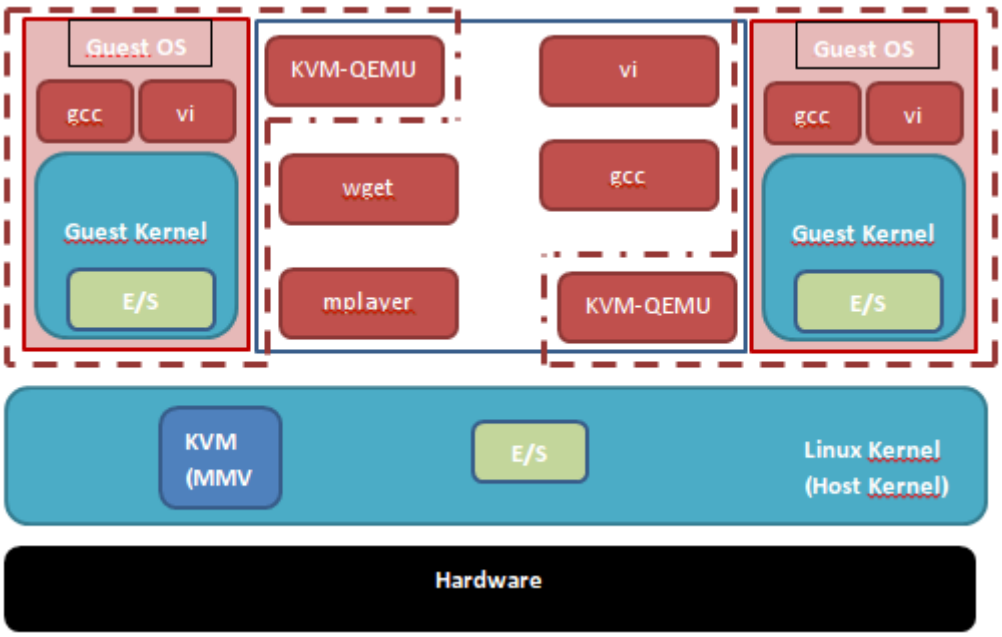


Figura 3.2: Organização de um sistema usando o **KVM**

executar nativamente e quando alguma instrução necessita de intervenção do MMV, esta provoca uma interrupção no sistema computacional e o controle do núcleo de processamento é entregue ao MMV. Um exemplo de instruções que exigem a intervenção do MMV são as instruções ligadas às chamadas de E/S da máquina virtual, enquanto que as instruções aritméticas não necessitam deste tipo de intervenção.

Ainda na Figura 3.2, a área circunscrita pelo tracejado em vermelho ressalta os componentes de uma máquina virtual KVM em execução, composta por um *Guest OS*, que se comunica com o modulo KVM no *kernel* do sistema operacional e uma instância da aplicação **KVM-QEMU**, executando no nível do usuário, a comunicação entre a máquina virtual e a sua instância do **KVM-QEMU** é realizada através do KVM. A instância do **KVM-QEMU** é o responsável por realizar as operações de E/S da máquina virtual, sendo deste modo parte fundamental de um sistema computacional virtualizado com o KVM.

3.2.3 Paravirtualização

As máquinas virtuais desta classe executam sobre um MMV que é responsável por gerenciar os acessos feitos pelas máquinas virtuais. O MMV permite a construção de máquinas virtuais que são semelhantes à máquina física que ele virtualiza, porém nesta classe de Máquinas Virtuais o conjunto de instruções original não se encontra todo virtualizado, o que obriga as máquinas virtuais, e por consequência o SO que executa sobre estas, a executarem estas instruções não suportadas através de chamadas ao MMV [67].

Xen

O **Xen** [70, 71] é um monitor de máquina virtual de código livre que implementa o modelo de paravirtualização. Desta forma o núcleo do sistema operacional deve ser modificado de forma a utilizar as chamadas que substituem os recursos que não são virtualizados. Entre estes recursos estão algumas instruções que acessam estados internos do processador sem com isso causar uma interrupção. A implementação do Xen consegue fornecer um mecanismo de virtualização de baixo custo computacional para aplicações que sejam dependentes apenas de CPU.

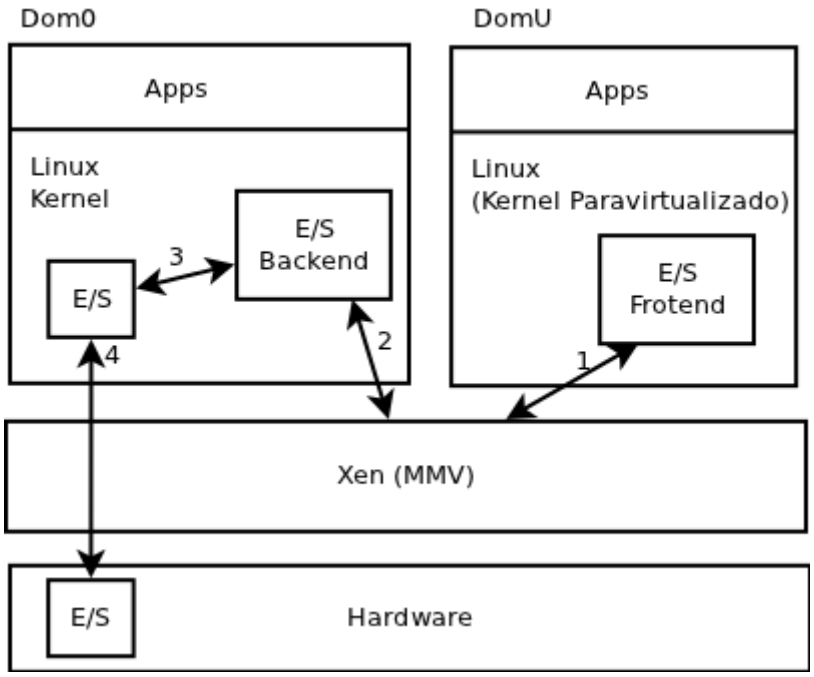


Figura 3.3: Organização de um sistema computacional usando **Xen**

A Figura 3.3 ilustra um ambiente computacional usando Xen. Pode-se observar o MMV e o Domínio Privilegiado (Dom0), responsável por realizar o acesso aos dispositivos de E/S no *hardware*. Por fim, neste ambiente, as máquinas virtuais, ou Domínio não Privilegiado (DomU), somente realizam acessos aos dispositivos de E/S através do **Dom0**. Ainda na Figura 3.3, as setas numeradas apresentam o caminho realizado para o atendimento de requisições de E/S. Em especial, o driver de E/S da máquina virtual (E/S *Front-End*) comunica-se com o **MMV** (1), o MMV repassa (2) essa requisição para o driver de E/S no Dom0 (E/S *BackEnd*), o *device driver* no Dom0 repassa o pedido (3) ao subsistema de E/S do *kernel*, para finalmente o subsistema de E/S realizar o acesso ao *hardware* (4). As etapas que ocorrem entre o Dom0, **DomU** e o MMV são realizadas através do canal de eventos do Xen e páginas de memória compartilhadas. O mecanismo que controla o procedimento

descrito anteriormente no Xen é a *grant table* [71].

3.2.4 Virtualização no Nível do Sistema Operacional

A Virtualização no Nível do Sistema Operacional constrói uma nova abstração dentro dos Sistemas Operacionais tradicionais, os contêineres de recursos (*resource containers*) ou simplesmente contêineres [72, 73]. Um contêiner isola dentro do núcleo do SO todas as estruturas de controle referentes às aplicações que ele encapsula, viabilizando desta forma a utilização de políticas diferentes para o gerenciamento de cada conjunto de aplicações e permitindo um melhor balanceamento de recursos computacionais entre os grupos isolados [74].

Uma das vantagens desta classe de virtualização é o baixo custo imposto por suas implementações, sendo esse o motivo de sua escolha para ambientes compartilhados como o PlanetLab [75], onde a demanda por capacidade de processamento é alta. O baixo custo computacional (consumo de CPU e memória) das implementações de contêineres é devido a ausência da necessidade de prover mecanismos para interpretação de instruções e outras complexidades necessárias para a virtualização total ou paravirtualização.

OpenVZ

O **OpenVZ** [76] é um dos diversos trabalhos [72, 73] na literatura que implementam essa classe de virtualização. Um diferencial com relação a outros trabalhos é que no OpenVZ apenas o núcleo do Linux é compartilhado entre os diversos contêineres. Assim, diversos Sistemas Operacionais (aplicativos de sistemas e bibliotecas) podem executar sobre o mesmo núcleo Linux compartilhado de forma transparente.

Com relação à tecnologia de migração, o OpenVZ utiliza um algoritmo do tipo suspender/resumir [77], onde se faz necessário serializar em um arquivo todo o estado do contêiner antes que esse possa ser recriado no nó computacional de destino. O uso deste tipo de migração faz com que o *downtime* percebido pelas aplicações que executam dentro do contêiner que está sendo migrado seja costumeiramente superior aos impostos por procedimentos migração similares aos utilizados no Xen ou no KVM.

Linux Containers

O projeto de *Linux Containers* (LXC), oferece o suporte à virtualização de contêineres no Linux. O suporte à esta classe de virtualização no *kernel* oficial do Linux (*kernel Vanilla*) é feito através de espaços de nomes (*namespaces*) [78], introduzidos apartir da versão 2.6.15 do Linux como pode ser visto na Tabela 3.5.

A implementação do **LXC** é feita através de alterações no *kernel* e aplicativos de gerenciamento no nível do usuário, aplicativos esses que utilizam as novas características construídas no *kernel* para suportar contêineres.

Tabela 3.5: Implementação do espaço de nomes usados pelo LXC no *kernel* do Linux ao longo de diferentes versões

Subsistema	Versão do Kernel
Shared SubTrees	2.6.15
UTSNAME	2.6.19
PID	2.6.24
IPC	2.6.19
USER	2.6.23
NETWORK	2.6.26
/PROC	2.6.26
RO BIND MOUNT	2.6.24

O LXC permite que tanto se isole apenas aplicativos, como um sistema operacional completo (aplicações e bibliotecas) de maneira similar ao **OpenVZ** (ver seção 3.2.4). No Linux os espaços de nomes dos recursos são implementados por processos e sendo que cada processo referencia um conjunto de espaços de nomes, normalmente compartilhados com outros processos de uma mesmo contêiner [79]. O uso de espaços de nomes provê tanto isolamento quanto virtualização, uma vez que processos em contêineres diferentes podem possuir recursos com o mesmo **ID**, como por exemplo, dois processos podem deter o mesmo **PID** desde que estejam em contêineres distintos.

A implementação de contêineres no *kernel* oficial do Linux, através do LXC, permitiu o surgimento de um grupo de trabalho que busca construir um mecanismo de migração, ou *checkpoint-restart* (**Linux-CR**) [80], para este sistema operacional. O projeto **Linux-CR** é baseado em experiências de projetos similares que não foram introduzidos no *kernel* [72, 76].

3.3 Serviços de computação em nuvem

De forma geral, o termo *Cloud Computing* [12, 81, 82], ou computação na nuvem, tem sido empregado para designar serviços computacionais em larga escala, distribuídos , dinamicamente reconfiguráveis e escaláveis, entregues sob demanda aos usuários e acessíveis através da Internet. Infelizmente, essa definição inicial abre uma gama de opções o que a torna confusa e dificulta o uso do termo. A Figura 3.4 apresenta os três diferentes tipos básicos de modelos que são chamados hoje de *cloud computing*. Como estes modelos podem designar componentes em diferentes níveis

do ambiente de um sistema computacional, faz-se necessário para este trabalho definir tanto quais são essas abordagens que hoje são encontradas na literatura e, em especial, qual delas é assumida no restante do texto.

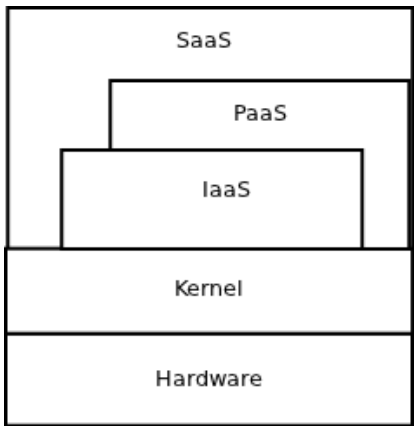


Figura 3.4: Hierarquia de serviços de computação na nuvem

3.3.1 Software como serviço - SaaS

Software como serviço em inglês, *Software as a Service* (SaaS) [81], é um modelo de distribuição onde aplicativos de propósito específico são armazenados no provedor de serviço e disponibilizados aos usuários através de uma rede de interconexão como a Internet. Na Figura 3.4 nota-se, que o SaaS fica no topo da hierarquia apresentada, podendo ser construído diretamente sobre o ambiente computacional (*hardware* e *kernel*), ou sobre um provedor de **PaaS** ou ainda sobre um provedor de IaaS. A *Salesforce.com* [83] é um exemplo de provedor de SaaS.

3.3.2 Plataforma como serviço - PaaS

Plataforma como serviço em inglês, *Platform as a Service* (PaaS) [81], é um modelo de distribuição de um ambiente integrado de alto nível para construir, testar e disponibilizar aplicativos customizados. Na Figura 3.4, o PaaS fica no meio da hierarquia apresentada, podendo ser construído diretamente sobre o ambiente computacional (*hardware* e *kernel*) ou sobre um provedor de **IaaS**. O desenvolvimento de aplicações sobre um **PaaS** oferece algumas vantagens para os desenvolvedores (usuários da plataforma), como o suporte à auto-escalabilidade da aplicação e atualização do sistema operacional sem a necessidade de intervenção dos desenvolvedores, sendo necessário porém que os desenvolvedores aceitem algumas restrições de como eles podem escrever seus aplicativos. Este tipo de serviço é provido pela Google em seu **App Engine** [84] e pelo sistema de código aberto **AppScale** [85].

3.3.3 InfraEstrutura como serviço - IaaS

O modelo de infraestrutura como serviço (em inglês, IaaS) [12, 81] apresenta ao usuário do sistema computacional uma interface onde ele pode solicitar a alocação de recursos computacionais de acordo com sua demanda sem a necessidade de uma prévia contratação destes recursos. Como ilustrado na Figura 3.4, IaaS é a camada de abstração mais próxima do *kernel* do sistema e do *hardware* em comparação com os modelos SaaS e PaaS. O cliente de **IaaS** administra um parque de máquinas virtuais que pode se expandir ou se contrair de acordo com a demanda, ou de acordo com a disponibilidade de recursos do provedor. Observe que um cliente de IaaS pode tanto ser um administrador, como um sistema computacional totalmente automatizado. Sendo que dentro do escopo desta tese um ambiente de IaaS é definido como o conjunto de *software* e *hardware* utilizado para construir o serviço de infraestrutura.

Elasticidade

Dentro do escopo deste trabalho, **elasticidade** é definida como a capacidade da aplicação requisitar, sob demanda, a alocação de recursos computacionais e liberação de recursos ociosos, de modo a atender a sua demanda computacional corrente [12]. A elasticidade de uma aplicação é limitada pelos algoritmos e estrutura de dados utilizados no desenvolvimento da mesma, é importante observar que o próprio ambiente IaaS também pode ser elástico, basta para isso que o software que gerencia um ambiente computacional tenha a capacidade de alocar ou liberar recursos computacionais. Baseado neste conceito de elasticidade propõe-se então o **ambiente resiliente** [1], como um ambiente computacional que oferece suporte a elasticidade. E define-se, dentro do escopo deste trabalho, **resiliência** como a propriedade do ambiente computacional de atender as demandas de alocação e liberação de recursos por parte dos usuários sob demanda. Observe ainda, que o termo resiliência é utilizado com significados diversos mesmo dentro da área de sistemas e computação [86], logo é imprescindível ter sempre em mente a definição dada nesta seção para melhor compreender o conteúdo deste trabalho.

Observe ainda, que o suporte à elasticidade é a característica inovadora dos ambientes de IaaS e a diferença fundamental entre estes e sistemas computacionais baseados em *cluster*, compartilhado ou não, e grades computacionais. Do ponto de vista do provedor de serviço, os recursos físicos podem ser compartilhados entre os usuários através da execução de diversas máquinas virtuais sobre a mesma máquina física, reduzindo desta forma os custos associados à compra, manutenção e utilização destes recursos computacionais. Contudo, deve-se ter em mente que, apesar dos

[1] resiliente, adj. 2 g.: Relativo à resiliência; que possui elasticidade = flexível

avanços nas tecnologias de virtualização, a execução simultânea de máquinas virtuais sobre o mesmo *hardware* ainda permite que ocorra interferência entre estas [87].

Entre os provedores deste tipo de serviço, estão a Amazon Web Services com seu serviço de EC2, o Google Compute Engine [88] e o **Enomaly** [89]. Além disso, existem sistemas de código aberto para a construção de ambientes de IaaS como o **Eucalyptus** [90], **Opennebula** [91] e **OpenStack** [92].

Eucalyptus

O **Eucalyptus** é um sistema de código aberto para a construção de ambientes de IaaS, sendo um dos primeiros ambientes abertos disponibilizados, e originalmente oferecia suporte apenas ao monitor de máquinas virtuais Xen. O suporte ao KVM [93] surge em sua versão **2.0**. A motivação para sua criação é fornecer meios para os pesquisadores estudasem os problemas que afetam os ambientes de IaaS [90]. O ambiente **Eucalyptus** é composto por quatro componentes principais, são eles:

- Controle de Nó computacional (NC): controla a execução, inspeção e término de uma máquina virtual em seu nó computacional;
- Controle do *Cluster* (CC): executa no *cluster front-end* e recolhe informações dos Controles de Nós e escalona máquinas virtuais em um Controle do Nó específico, além de gerenciar a rede virtual do *cluster*;
- Controle de Armazenamento (SC): Componente que executa *cluster front-end* e implementa o serviço de armazenamento *put/get*;
- (**Walrus**): um serviço de armazenamento *put/get* que implementa a interface do S3 [94] da Amazon;
- Controle da Cloud (CLC): ponto de entrada dos usuários e administradores na *Cloud* (ambientes de IaaS). Consulta o controle do *Cluster* para pegar informações sobre recursos e toma decisões de escalonamento de alto nível implementando estas decisões através de requisições ao controle do *Cluster*

A Figura 3.5 apresenta o ambiente físico de uma instalação do **Eucalyptus** com dois *cluster*s. Nela é possível observar o *Cloud Front-End* que executa o Controle da Cloud. O *Cloud Front-End* é interconectado com os *Clusters Front-Ends* através de um comutador. Ainda, cada *Cluster Front-End* executa o Controle do *Cluster*, sendo responsável por atender as requisições do *Cloud Front-End* e se comunicar com os Controle de Nó computacional que executam em cada um dos nós do *cluster*. O **Walrus** é o componente de alto nível do sistema de armazenamento que executa no *Cloud Front-End*, visto na Figura 3.5, enquanto que cada um dos *Cluster Front-Ends* do sistema deve executar o Controle de Armazenamento.

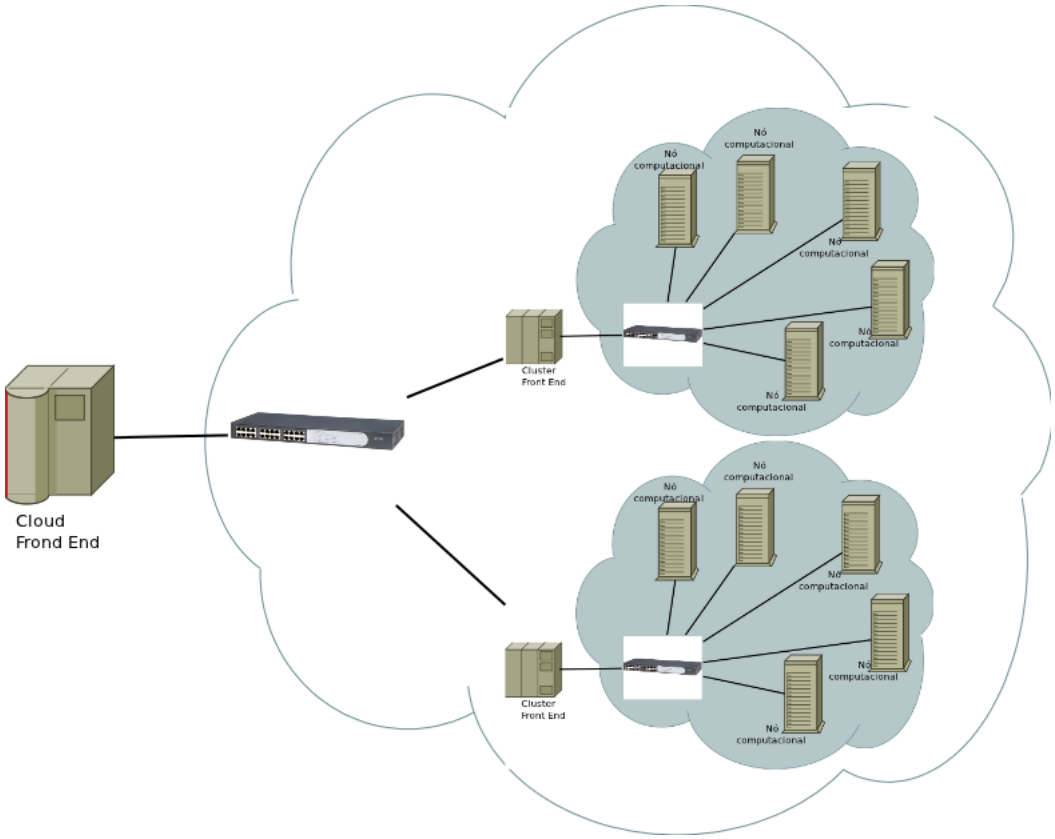


Figura 3.5: Organização de um ambiente de **IaaS** usando **Eucalyptus** com dois *cluster*s

OpenNebula

O **OpenNebula** é um sistema para gerenciamento de infraestrutura virtualizada, que facilita a construção de um ambiente de IaaS privado ou híbrido. Ele é um sistema *open source* que possibilita a uma organização implemente o modelo de uso de IaaS em seus *cluster*s, oferecendo deste modo flexibilidade e agilidade aos desenvolvedores e administradores de sistemas. O **OpenNebula** possui dois tipos de máquinas, o *front end*, responsável por receber as requisições dos clientes através de uma API XML-RPC e os nós computacionais que executam efetivamente as máquinas virtuais. Ele oferece ainda interface de controle compatível com a do EC2 onde os usuários podem requisitar alocação e liberação de recursos computacionais de acordo com sua demanda atual.

A Figura 3.6 apresenta uma visão simplificada do ambiente computacional do **OpenNebula**, nela é possível observar no nível mais baixo a infraestrutura local, dela faz parte também o sistema operacional, o monitor de máquina virtual e um possível provedor externo de IaaS, sobre estes encontra-se o **OpenNebula** propriamente dito, este ainda subdividido em *drivers*, escalonador e núcleo. Os *drivers* do **OpenNebula** servem como ”cola”, ou seja eles são responsáveis por efetivamente

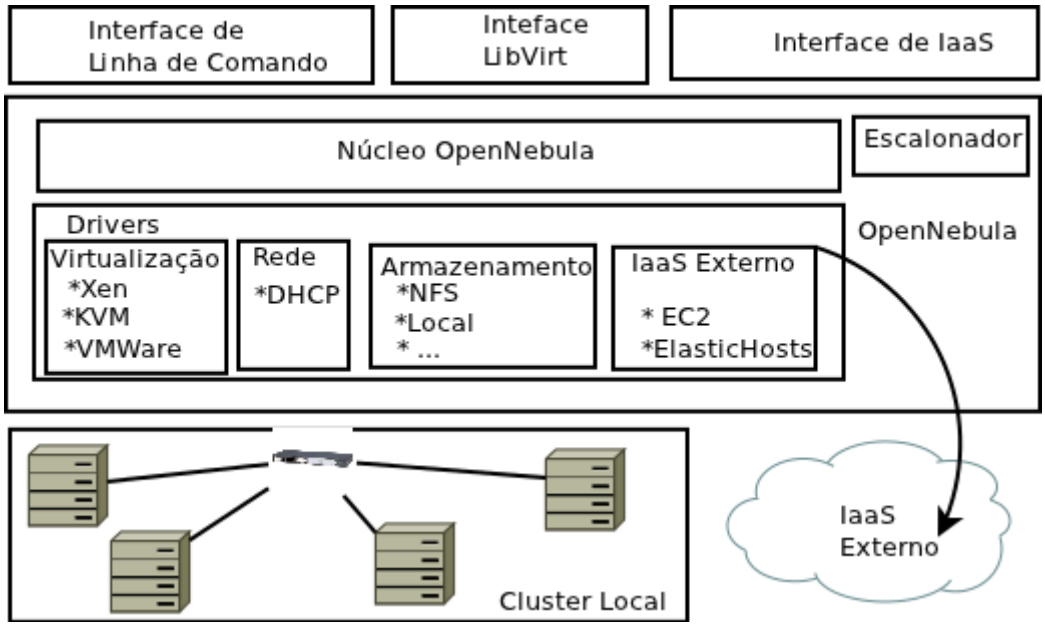


Figura 3.6: Organização de um ambiente de IaaS **OpenNebula**

executarem as tarefas do sistema de gerenciamento sobre a infraestrutura física, ainda com relação aos *drivers* deve ser observado que em sua versão atual o **OpenNebula** oferece suporte a três monitores de máquinas virtuais, são eles: Xen, KVM e VMWare.

O núcleo do **OpenNebula** é o componente do ambiente responsável por receber as requisições das interfaces com o usuário e realizar as operações necessárias para servir essas requisições. Nos casos onde a requisição é uma alocação de máquina virtual ele deve primeiro contatar o Escalonador para descobrir em qual recurso físico a nova máquina virtual deve ser iniciada. Por fim, no topo da Figura 3.6, encontram-se as interfaces com o **OpenNebula**, são através destas que os usuários do sistema interagem com o mesmo.

O ambiente modular do **OpenNebula** facilita a introdução de novos recursos como novos monitores de máquina virtuais ou a alteração do escalonador utilizado por este. Um exemplo deste tipo de alteração é a substituição do escalonador padrão do sistema de gerenciamento pelo **Haizea** no projeto **RESERVOIR** [91, 95]. Sendo que assim como o **Eucalyptus**, o **OpenNebula** oferece suporte a múltiplos *clusters* [96].

OpenStack

O **OpenStack** [92] é um sistema de código livre para o desenvolvimento de ambientes de computação na nuvem escaláveis. Ele pode ser utilizado para o projeto de ambientes de IaaS públicos ou privados. A Figura 3.7 apresenta uma das possíveis organizações de uma instalação deste sistema de gerenciamento, sendo necessário

utilizado quatro projetos correlacionados, são eles:

- *OpenStack Compute* [97] (Nova): Responsável por gerenciar as instâncias de máquinas virtuais;
- *OpenStack Networking* (Nova-Network): Responsável por gerenciar a rede das instâncias de máquinas virtuais;
- *OpenStack Identity service* [98] (KeyStone): Responsável por gerenciar as identidades dos diferentes usuários do ambiente de IaaS;
- *OpenStack Image Service* (Glance): Responsável por gerenciar as diversas imagens de máquinas virtuais disponíveis no ambiente.

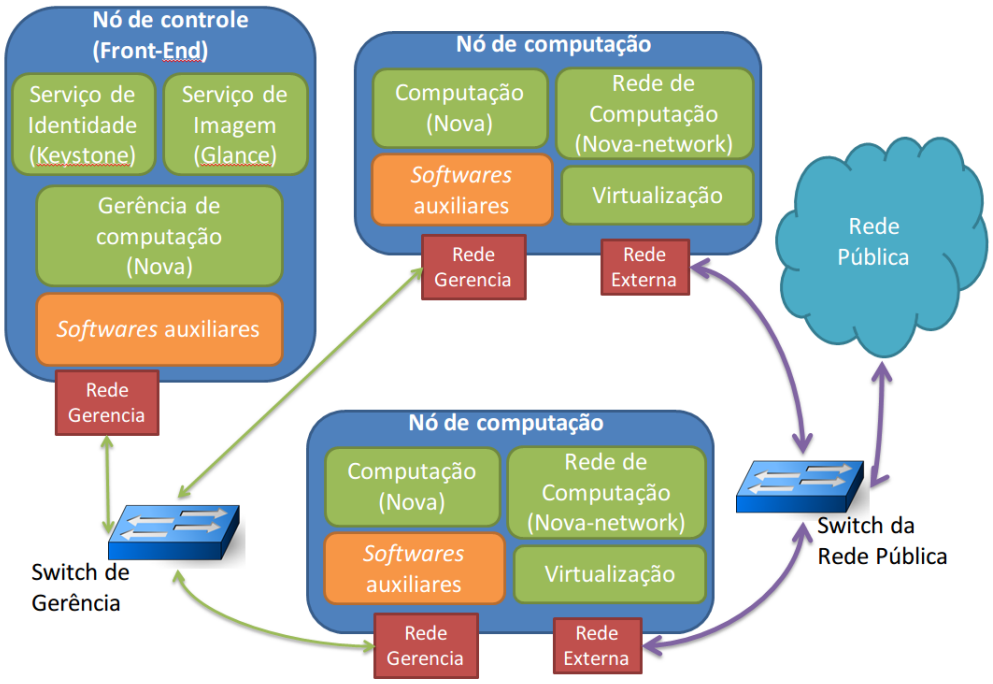


Figura 3.7: Organização de um ambiente de IaaS usando o **OpenStack**

O *OpenStack Compute*, também conhecido como **Nova**, é uma ferramenta para provisionar e gerenciar grandes redes de máquinas virtuais. Ela oferece o suporte necessário para o gerenciamento de um ambiente de IaaS, como por exemplo, a execução de máquinas virtuais e gerenciamento de redes virtuais. Ela tem como objetivo ser independente da tecnologia de virtualização de modo que define um conjunto de drivers que acessam a técnica de virtualização e exportam as funcionalidades através de sua API. Atualmente o **Nova** oferece suporte a sete monitores de máquinas virtuais distintos, são eles: Xen, KVM, LXC, Hyper-V 2008, QEMU, User Mode Linux, VMWare.

Ainda com relação ao projeto **Nova**, as imagens das máquinas virtuais podem ser gerenciadas de duas maneiras distintas. A primeira é através do serviço *OpenStack Image* [98], ou **Glance**. O **Glance** é responsável por rotear as requisições dos clientes para um dos seus serviços de armazenamento, que podem ser um sistema de arquivo, armazenamento no **S3** da *Amazon*, *Hypertext Transfer Protocol* (HTTP) ou *OpenStack Object Storage*. A outra forma como as imagens podem ser armazenadas é utilizando diretamente o serviço de armazenamento *OpenStack Object Storage*.

- *OpenStack Object Storage* [97]: Serviço de armazenamento redundante e escalável;
- *OpenStack Networking* (Neutron):

O serviço de armazenamento, o *OpenStack Object Storage*, também chamado de **Swift**, é um aplicativo de código livre para construir um sistema de armazenamento de objetos escalável e redundante usando *cluster* de computadores. O **Swift** não é um sistema de arquivos ou um sistema de armazenamento de tempo real, mas um sistema de armazenamento de longo prazo e escalável. O sistema fica responsável pela replicação e consistência das escritas realizadas. O **Swift** oferece uma interface similar ao **S3**, ou seja os usuários podem escrever pares de chave/valor (limite de 4 KB) ou buscar um dado valor (objeto) fornecendo sua chave como parâmetro.

3.4 Serviços Web escaláveis

A criação e manutenção de serviços Web escaláveis representa ainda um grande desafio para projetistas, desenvolvedores e administradores destes serviços, pois não se pode prever o crescimento da demanda por poder computacional relacionado ao aumento do número de usuários do serviço [12]. Somando-se a isso existe a dificuldade inerente de se projetar e construir sistemas escaláveis com o número de usuários e os custos associados ao provisionalmente errado da capacidade computacional para o novo serviço.

Atualmente, os recursos computacionais utilizados na construção de serviços escaláveis são organizados como *cluster*s de computadores, sistemas par-a-par ou mesmo ambientes computacionais híbridos. Os *cluster*s, por exemplo, têm sido utilizados com sucesso tanto na academia quanto na indústria como forma de oferecer serviços que necessitem de mais poder computacional do que pode ser encontrado em um servidor comum. Contudo, apesar de sua larga adoção, a desenvolvimento de sistemas escaláveis neste ambiente computacional está longe de ser uma tarefa trivial. O projeto de serviços computacionais como o *Simple Storage Service* (S3)

e *Elastic Compute Cloud* (EC2), bem como o uso destes serviços, fazem com que as dificuldades inerentes ao projeto de sistemas computacionais escaláveis tornem-se comuns no projeto de aplicações Web [99]. Um destes problemas é a como tratar a consistência dos dados [100], bem com a disponibilidade e a tolerância a particionamentos [101].

Nos ambientes distribuídos a comunicação entre os diferentes componentes do sistema computacional é realizada usando passagem de mensagem. Por exemplo, na computação científica essa troca de mensagens é feita através do padrão *Message Passing Interface* (MPI). Existem ainda trabalhos [102][103][104] que buscam tornar mais simples o desenvolvimento de sistemas para essa classe de computadores em troca de um aumento no *overhead* mais nenhum ainda desponta como uma solução de propósito geral. Olhando do para as aplicações distribuídas, é possível observar que na Internet o serviço que mais tem crescido é o de transmissão de dados multimídia [18].

3.4.1 Organização Lógica de um Sistema de Transmissão de vídeo sob demanda

Um sistema de transmissão de vídeo sob demanda ou *Video On Demand* (VoD) é um serviço Web de tempo real não-crítico (*soft real-time*), modelado conceitualmente como uma aplicação cliente/servidor, onde o cliente solicita ao servidor que esse lhe envie o arquivo de vídeo desejado. Os requisitos de tempo real deste tipo de serviço surgem da necessidade de o servidor transmitir os dados enquanto concorrentemente o cliente está exibindo o vídeo [105]. Além dos requisitos de tempo real não-crítico outra caraterística importante dos sistemas VoD é o conceito de Fator de Paciência, que é o intervalo de tempo que o cliente aceita esperar para ver um vídeo[106].

Os requisitos de tempo real e Qualidade de Serviço (QoS), somados às questões de escalabilidade, acabam por aumentar a complexidade da construção deste tipo de serviço Web. Uma possível maneira de se projetar este serviço de modo a atender estes requisitos é através da introdução, por exemplo, servidores de cache entre o servidor de vídeo e os clientes.

SMART: Um sistema de Tempo-Real Não-crítico para Transmissão de vídeo sob demanda na Internet usando virtualização

O **SMART** [107] sistema para transmissão de vídeo sob demanda escalável, projetado em torno de técnicas de virtualização. Nele, os nós do *cluster* executam um ou mais contêineres, como pode se observado na Figura 3.8, cada contêiner executando

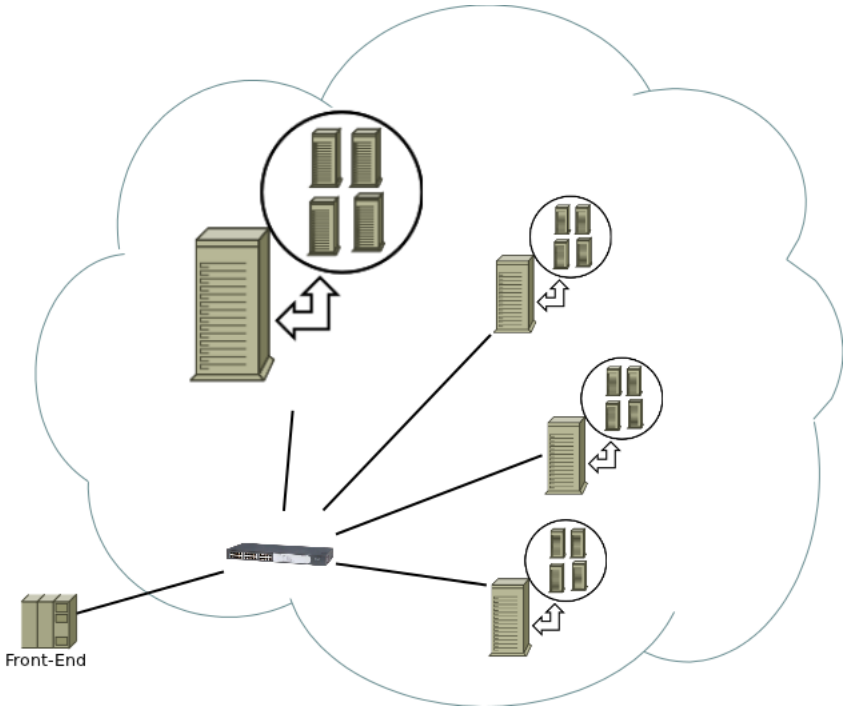


Figura 3.8: Cluster de servidores virtualizados utilizados no sistema de transmissão de vídeo sob demanda **SMART**

uma instância da aplicação de transmissão de vídeo e assumindo ainda que, cada instância do aplicação de transmissão ficaria responsável por apenas um único vídeo.

O uso de virtualização permitiu ao sistema **SMART** fornecer as garantias de QoS necessárias para a construção de um serviço de transmissão de vídeo sob demanda. Com relação a escalabilidade, o projeto do **SMART** o torna escalável uma vez que para aumentar a capacidade de transmissão do sistema basta executar um novo contêiner, dado é claro que existam recursos computacionais disponíveis no *cluster* para que o contêiner seja executado.

3.5 Escopo

O problema da avaliação de desempenho dos ambientes computacionais de IaaS é abordado dentro do escopo do projeto utilizando uma aplicação elástica como *benchmark* para quantificar as diferenças entre os ambientes computacionais. Sendo que dentro desta tese um ambiente computacional de IaaS é definido como um *cluster* sobre o qual é executado o *software* de gerenciamento de IaaS e cujos nós computacionais oferecem suporte de algumas das técnicas de virtualização descritas neste capítulo. Dentro deste escopo é relevante observar as aplicações Web escaláveis e em especial a aplicação de VoD que devido existência do conceito de Fator de Paciência mostra-se promissora no projeto de um *benchmark*.

Capítulo 4

Metodologia para avaliação de desempenho de ambientes de Infraestrutura como Serviço

Neste capítulo são abordadas inicialmente as principais características que permeiam os ambientes de *Infrastructure as a Service* (IaaS). Em particular, são descritas as principais diferenças entre este ambiente e os paradigmas de computação em *cluster* e em grades. Em seguida, são apresentados os aspectos que são levados em consideração no projeto de *benchmark*s para avaliação quantitativa de desempenho para ambientes de IaaS, focando as questões da elasticidade da aplicação e da resiliência do ambiente computacional. Posteriormente é apresentado o novo *benchmark* para ambientes de IaaS baseado em um sistema de distribuição de vídeo. Por fim, as questões abordadas no decorrer do capítulo são colocadas em perspectiva dentro do escopo desta tese.

4.1 Visão Geral

Os ambientes de computação nas nuvens, e em particular as que oferecem o serviço de Infraestrutura como Serviço (IaaS), carregam muitos dos objetivos dos projetos de computação utilitária (*computing as a utility*) [12]. Estes ambientes trazem novos problemas do ponto de vista da organização do sistema computacional, como a necessidade das aplicações escalarem horizontalmente de forma eficiente e de maneira elástica, seja para adquirir recursos ou para liberá-los e os sistemas do nível de infraestrutura, como o sistema operacional [108], precisam estar cientes de que não mais são executados diretamente sobre o nó físico. Em termos da organização do *hardware*, os ambientes de IaaS são construídos sobre *cluster*s de computadores,

sejam estes de pequeno porte para ambientes privados ou *DataCenters* para ambientes públicos como o EC2. Do ponto de vista do sistema de gerenciamento de ambiente computacional de IaaS o computador é o *DataCenter* [109].

4.1.1 A Elasticidade das aplicações nos ambientes de IaaS

Como definido na Seção 3.3.3, a **elasticidade** é a denominação dada à capacidade de uma aplicação, em execução em um dado provedor de IaaS, requisitar sob demanda, a alocação ou liberação de um conjunto de máquinas virtuais cuja capacidade pode ser especificada no momento da alocação ou liberação, doravante denominado de sistema virtual. A **elasticidade** de uma dada aplicação é impactada por fatores que podem ser divididos em dois grupos. O primeiro é determinado pelo tipo de sistema virtual que a aplicação demanda e o segundo pela **resiliência** do ambiente de IaaS.

O tipo de sistema virtual que uma aplicação demanda é resultado dos algoritmos, bibliotecas e requisitos de Qualidade de Serviço (QoS) utilizados na execução da aplicação em um dado ambiente de IaaS, e a resiliência é a capacidade de resposta do ambiente de IaaS em atender a demanda de sistemas virtuais feita pela aplicação. Portanto, o impacto na **elasticidade** de uma aplicação é também determinado pela **resiliência** do ambiente de IaaS na qual ela for executada. A resiliência é consequência da organização do *hardware* e do conjunto de *softwares* usados para gerenciamento do ambiente de IaaS. Os *softwares* de gerenciamento podem estar tanto no *front-end* do *cluster* como nos nós computacionais que compõem um dado ambiente de IaaS. No *front-end*, por exemplo, encontram-se os algoritmos de escalonamento e provisionamento de máquinas virtuais, o sistema de coleta e o mecanismo de requisição de informações aos nós computacionais. Nestes nós, os componentes compreendem o sistema de gerenciamento e coleta de informações local, a tecnologia de virtualização utilizada e o sistema operacional.

A Figura 4.1 mostra o comportamento de uma aplicação elástica fictícia, onde o eixo horizontal mostra a duração da execução em horas e o eixo vertical a quantidade de recursos computacionais alocados à computação. Ainda nesta figura, a curva em azul representa a quantidade de recursos computacionais disponíveis e a cor vermelha a quantidade de recursos computacionais utilizados para atender a demanda da aplicação. Ainda na Figura 4.1 a latência de alocação dos recursos computacionais é zero, ou seja, a resiliência do ambiente de IaaS não impacta na elasticidade, contudo é importante ter em mente que nos sistemas computacionais reais esta latência de alocação não é nula.

Observe que, atualmente apesar dos benefícios teóricos que as aplicações elásticas

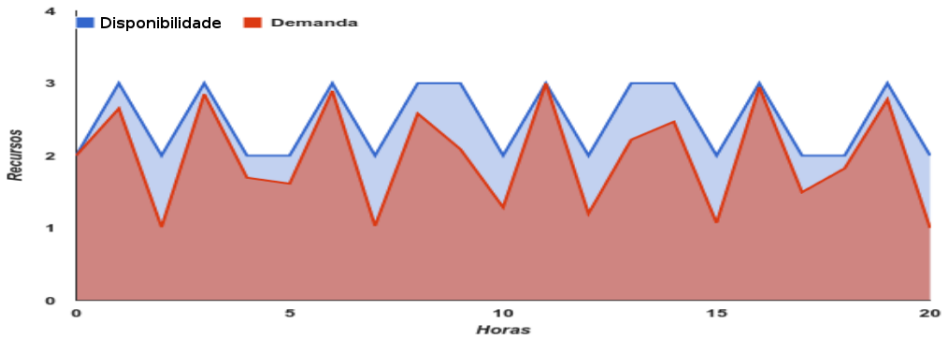


Figura 4.1: Exemplo do uso do recurso computacional de uma aplicação elástica no tempo: Disponibilidade X Demanda

possuem com relação as suas contra partes não elásticas [12], como uma melhor utilização do *hardware* disponível e aumento automático da capacidade computacional da aplicação distribuída, há dificuldade em encontrar aplicações de código livre disponíveis que utilizem a elasticidade fornecida pelos ambientes de IaaS, de forma que não é possível utilizar uma aplicação existente como *benchmark*s de elasticidade para os diferentes ambientes de IaaS. Para isso, é necessário reescrever as aplicações existentes para os ambientes de IaaS, em alguns casos para cada um dos ambientes de IaaS no qual essa aplicação será executada. Este trabalho, propõe, desenvolve e avalia o uso de uma aplicação elástica para a distribuição de vídeo na Internet sobre o protocolo *Hypertext Transfer Protocol* (HTTP), como meio de aferir quantitativamente as vantagens do uso da elasticidade no projeto das aplicações e os impactos da resiliência de um dado ambiente de IaaS na mesma.

4.2 Descrição do Problema

O surgimento de sistemas de código aberto para a construção de ambientes de Infraestrutura como Serviço [90–92], bem como o crescente número de provedores que oferecem este tipo de serviço através da Internet, levantam questões tanto para o dono da infraestrutura como para seus usuários de como quantificar o desempenho de dois ambientes de infraestrutura distintos. Apesar de existir estudos acadêmicos, métodos e ferramentas que avaliam individualmente os componentes de um ambiente de IaaS, são necessárias novas ferramentas e métodos para avaliar as questões de elasticidade das aplicações. A complexidade de construir um ambiente de IaaS torna necessário um entendimento mais profundo de como as características intrínsecas dos diferentes componentes impactam um ambiente de IaaS como um todo e de que forma essas sobrecargas podem ser minimizadas ou eliminadas.

A busca por uma nova metodologia para avaliação de ambientes de IaaS pres-

supõe que a resiliência impacta diretamente a elasticidade das aplicações, sendo assumido que o suporte à elasticidade é a característica que define tais ambientes, ou seja, com a ausência deste suporte, um ambiente de IaaS não é diferente de um *cluster* de computadores tradicional. A avaliação da resiliência dos ambientes de IaaS passa então por identificar uma metodologia e aplicativos que permitam aferir quantitativamente as diferenças entre ambientes distintos. Em particular, consiga aferir quantitativamente o impacto que o suporte oferecido a elasticidade tem sobre os aplicativos que executam em um ambiente de IaaS.

4.3 Metodologia

Uma aplicação elástica deve incluir algoritmos e estruturas para lidar com a gerência dos recursos computacionais aumentado assim a complexidade da mesma. O impacto da adição ou remoção de recursos computacionais é consequência da combinação destes algoritmos e da resiliência dos ambientes IaaS. Assim, a metodologia proposta avalia dois entre os quatro itens listados na Seção 1.5.1 ambos vinculados ao *middleware* para gerenciamento de recursos, são eles:

- Impacto das Técnicas de Virtualização;
- Impacto dos Mecanismos de gerência da infraestrutura.

O método de avaliação proposto nesta tese é composto por três etapas: primeira é a pré-avaliação dos mecanismos de virtualização, seguida pela pré-avaliação da latência de alocação do recurso computacional e por fim a execução do *benchmark* proposto. A avaliação preliminar dos mecanismos de virtualização busca identificar como as diferentes técnicas e suas devidas implementações impactam nas métricas de desempenho do componente elástico do *benchmark*. Por exemplo, na aplicação elástica para a distribuição de vídeo na Internet esse componente é o servidor que envia os vídeos aos clientes e a métrica de interesse é a vazão de rede do servidor. A etapa de pré-avaliação da latência de alocação do recurso computacional busca quantificar os tempos médios que podem ser esperados de um ambiente de IaaS, dada a combinação do software de gerenciamento e da técnica de virtualização, que é utilizada para a configuração da execução do *benchmark*. A última etapa do método de avaliação utiliza as informações coletadas nas etapas anteriores para configurar o controle de elasticidade da aplicação de transmissão de vídeo, proposta nesta tese como um *benchmark* para ambientes de IaaS. Esses passos são listados a seguir:

1. Pré avaliação dos mecanismos de virtualização;

2. Pré avaliação da latência de alocação;

3. Execução do *benchmark* ElasticMovie.

O método de avaliação descrito anteriormente assume que a aplicação escolhida como *benchmark* para o ambiente de IaaS, possui um limite com relação ao tempo que pode esperar por um novo recurso computacional antes de ter sua operação impactada negativamente. No caso do ElasticMovie esse limite é dado pelo Fator de Paciência do cliente e o impacto negativo é o bloqueio de um cliente, assim outra aplicação que possua limitação similar também pode ser utilizada. Contudo é importante observar que nos casos onde a aplicação escolhida como *benchmark* não possui um limite no tempo que ela pode aguardar o novo recurso computacional não é possível garantir que esse método possa ser utilizado corretamente.

4.3.1 Métricas de Avaliação para ambientes de IaaS

A metodologia para avaliação de ambientes de IaaS é baseada na latência de alocação e liberação das máquinas virtuais. A primeira métrica do ambiente é o **Tempo para liberação de Recurso (TlR)**, definida como o intervalo de tempo que leva para que uma máquina virtual seja liberada pelo ambiente. A segunda métrica de ambiente o **Tempo para alocação de Recurso (TaR)**, que representa a duração total da alocação da nova máquina virtual, onde **TaR**, pode ser subdivido em:

- Tempo para escolha de nó físico (**TeF**): É o intervalo de tempo que o sistema de gerenciamento leva para escolher onde uma nova máquina virtual deve ser instanciada a partir de uma requisição do usuário;
- Tempo para cópia de imagem (**TcI**): É o intervalo de tempo que leva para transferir a imagem da nova máquina virtual ao nó onde será executada;
- Tempo para inicialização da imagem (**TiI**): É o intervalo de tempo que leva para inicializar uma máquina virtual, depende da tecnologia de virtualização e do método de contextualização.

4.3.2 Pré-avaliação dos componentes

As primeiras duas etapas da metodologia de avaliação passam por avaliar os principais elementos que compõem o ambiente de IaaS, tendo em vista a aplicação

escolhida. Para o serviço de transmissão de vídeo isto implica em aferir o comportamento com relação ao componente de transmissão de vídeo, por exemplo, o servidor HTTP, das diferentes máquinas virtuais que compõem o ambiente de IaaS. A latência media de alocação dos recursos computacionais é aferida e utilizada para configurar a aplicação e definir os parâmetros do *benchmark*. Por exemplo, assumindo um ambiente onde a latência media é de 30 segundos, se a maior latência de alocação suportada por uma aplicação for de 20 segundos ela não poderá utilizar este ambiente diretamente. Observe que esse problema pode ser contornado com a utilização de um conjunto de recursos pré-alocados, mas não em uso pela aplicação, reduzindo assim a eficiência da aplicação elástica no uso dos seus recursos computacionais.

Esta avaliação preliminar é fundamental, pois é através desta que são levantados os valores referentes à capacidade individual das máquinas virtuais. Outra avaliação preliminar importante é a identificação da latência de alocação média fornecida pelo ambiente de IaaS para aplicação de interesse. Estas informações são necessárias para o correto funcionamento dos mecanismos de gerência de recursos da aplicação elástica.

4.3.3 *Benchmark*: Serviço de Transmissão de vídeo sob demanda Elástico

A avaliação de ambientes de IaaS realizada através da implementação de um serviço de transmissão de *Video On Demand* (VoD) organizado de forma simular ao apresentado na seção 3.4.1, que utiliza os recursos de elasticidade do ambiente de IaaS de forma a atender a demanda dos usuários sem com isso desperdiçar recursos computacionais mantendo máquinas ociosas ativas. Esse tipo de serviço Web foi escolhido devido aos requisitos de tempo real e QoS exigirem tanto desempenho quanto tempo de resposta na alocação de novos recursos por parte do ambiente de IaaS, em especial a existência do conceito de Fator de Paciência, que é o intervalo de tempo que o cliente aceita esperar para ver um vídeo [106]. Outro fator relevante na escolha da aplicação de transmissão de vídeo como *benchmark* é a crescente importância da aplicação de transmissão de conteúdo de tempo real de entretenimento na Internet, que já é responsável por mais de 63% do trafego de *downstream* [18].

O sistema de transmissão elástico é uma das possíveis implementações do serviço de transmissão de vídeo de descrito na Seção 3.4.1, utilizando como base os resultados obtidos por Whately [107] que mostram a viabilidade do projeto de um sistema de transmissão utilizando máquinas virtuais. O ElasticMovie é o sistema elástico de transmissão de vídeos projetado para ser utilizado como um *benchmark* dos am-

bientes de IaaS. Neste cenário os servidores virtualizados assumem, por exemplo, a função dos servidores cache e passariam então a responder pelo envio dos fluxos de vídeos aos clientes, desta maneira o serviço de transmissão será composto por três componentes principais, são eles:

- Servidor de gerenciamento - **ElasticMovie** (*Front*);
- Servidor de distribuição - **ElasticMovie Node**;
- Cliente.

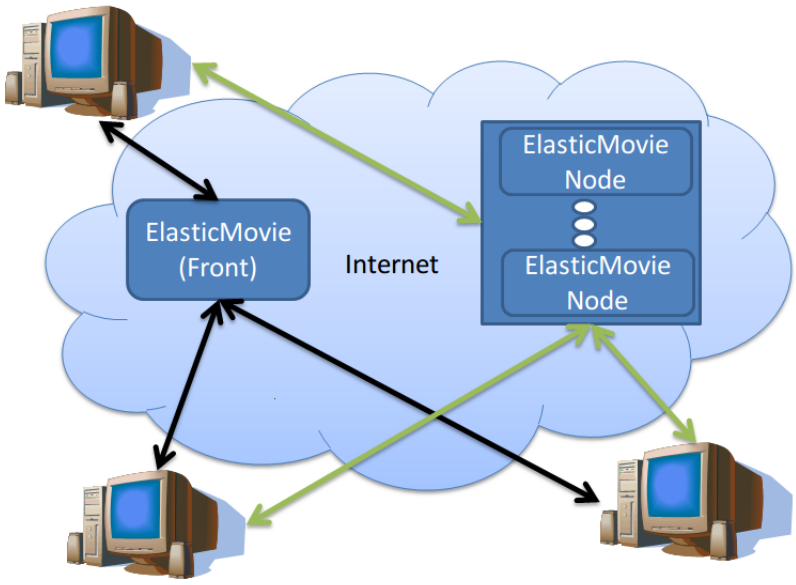


Figura 4.2: Organização lógica do sistema ElasticMovie

A Figura 4.2 apresenta a organização lógica dos componentes do serviço de transmissão de vídeo. Ainda na Figura 4.2 as setas de cor preta representam o contato inicial entre os clientes e o servidor de gerenciamento, as setas verdes representam o momento posterior quando os clientes contatam os servidores de distribuição e os servidores de distribuição transmitem o vídeo solicitado através do protocolo HTTP.

O **servidor de gerenciamento** é o componente responsável pelo gerenciamento das máquinas virtuais no ambiente de IaaS e definição de qual vídeo ele deve exibir [1], ele recebe requisições de vídeo dos clientes e as direciona a um **servidor de distribuição**. Ele deve determinar se existe um **servidor de distribuição** que possa atender este **cliente** e, caso exista, informar ao cliente qual será este servidor. No caso onde não for possível utilizar um servidor de distribuição ativo, o servidor

[1] Pode ser definido através da imagem da máquina virtual ou pela aplicação em tempo de execução

de gerenciamento deverá alocar uma nova máquina virtual. O **servidor de distribuição** é o componente do sistema de transmissão que executa obrigatoriamente dentro do ambiente de IaaS, ele é responsável pela transmissão do vídeo para os clientes, ao final de uma transmissão o cliente solicita a sua retirada do sistema de distribuição de vídeo.

Protótipo: ElasticMovie

O **ElasticMovie** é o protótipo de um sistema de transmissão de vídeo sobre HTTP projetada para servir como *benchmark* dos ambientes de IaaS. O principal objetivo do ElasticMovie é aferir quantitativamente o impacto que a resiliência de um ambiente de IaaS impacta no desempenho da aplicação de transmissão de vídeo, mas especificamente, na taxa de bloqueio de clientes no sistema de transmissão. Assim o ElasticMovie não utiliza preditores para controlar a alocação dos recursos computacionais com base na taxa de chegada de clientes, pois estes esconderiam a latência de alocação dos recursos computacionais. O protótipo é composto pelos componentes a seguir:

- ElasticMovie (*Front*);
- ElasticMovie Node;
- Cliente;
- IaaS *Front-End*
- IaaS *cluster* (*compute nodes*)

A Figura 4.3 apresenta como os diferentes componentes utilizados no projeto do serviço **ElasticMovie** encontram-se organizados. A seta preta representa o contato inicial entre o cliente e serviço de gerenciamento do ElasticMovie e a seta verde representa a transmissão de vídeo para os clientes. A seta laranja representa comunicação e entre o serviço de gerenciamento do ElasticMovie com o *Front-End* do serviço de IaaS, enquanto que a vermelha representa à comunicação entre os componentes do ambiente IaaS, sendo que estas somente ocorrem, relativas ao ElasticMovie, apenas quando da necessidade da criação ou destruição de uma instância.

Em seu estado estacionário, o sistema de distribuição de vídeo conta apenas com o componente ElasticMovie (*Front*) em execução, sendo novos recursos alocados de acordo com a demanda exigida. A Figura 4.4 apresenta o fluxograma realizado no tratamento de uma requisição, nela a seta de cor preta representa o começo do processamento, as setas de cor verde o caminho seguido para tratar uma nova solicitação

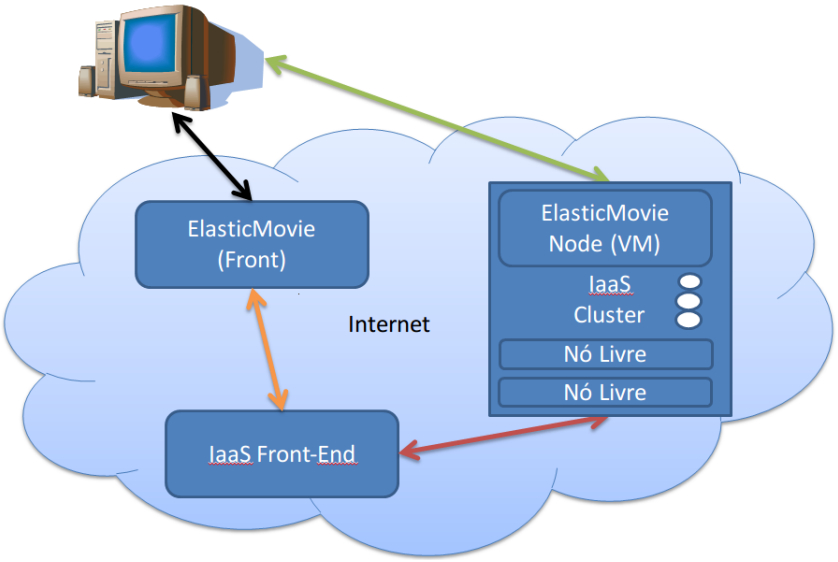


Figura 4.3: Organização dos componentes da implementação do sistema de VoD ElasticMovie em um ambiente de IaaS

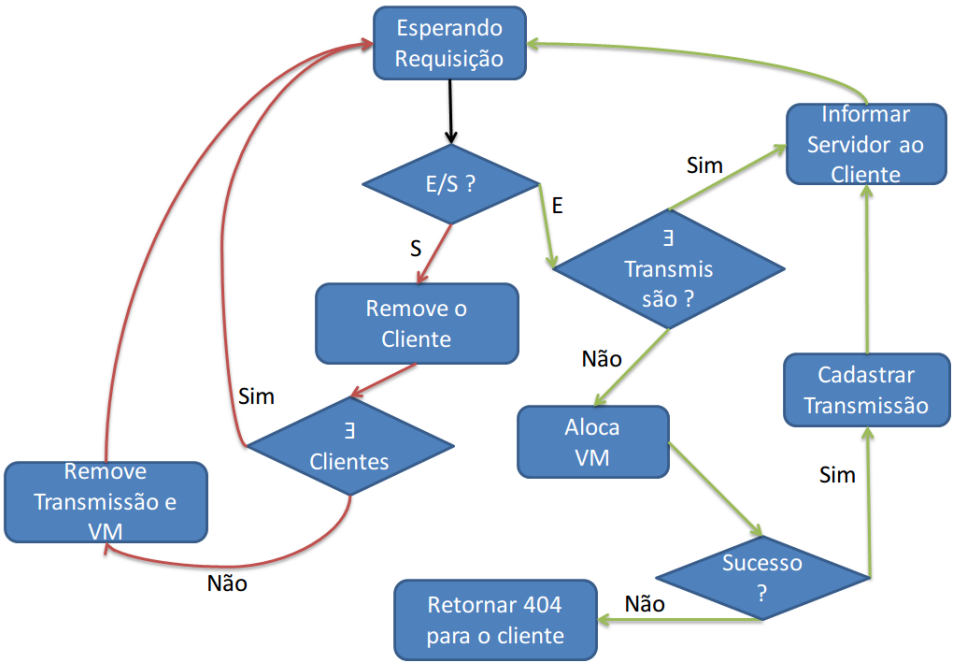


Figura 4.4: Fluxograma do processamento realizado ElasticMovie (*Front*) por requisição

de um cliente para transmissão de vídeo e as setas de cor vermelha o procedimento realizado quando um cliente finaliza sua transmissão. Nesta figura a interação entre o ElasticMovie (*Front*) e o ambiente de IaaS ocorrem no momento de alocação e na remoção de máquinas virtuais, sendo este o ponto de controle da elasticidade do ElasticMovie. O controle de elasticidade utilizado no protótipo inicializa uma

nova máquina virtual apenas quando no processamento de uma nova requisição de entrada não existe servidor que possa atender a transmissão ou quando no caso de uma requisição de saída o servidor de transmissão ficará sem clientes. Ainda nesta figura é importante ter em mente que uma transmissão somente será efetivamente inicializada se o cliente acessar o servidor de transmissão que ele recebeu como resposta, o que é feito de automaticamente através do código de redirecionamento do HTTP.

Métricas para Avaliação do ElasticMovie

O uso do ElasticMovie permite aferir o impacto que a latência de alocação das máquinas virtuais causa nos clientes. Na aplicação de transmissão de VoD essas métricas são:

- Quantidade total de requisições de Vídeo recebidas (**QtRvS**): Quantidade total de requisições de vídeo recebida pelo ElasticMovie (*Front*);
- Percentual de clientes atendidos com falha (**PcAf**): Percentual de clientes que receberam vídeo, mas tiveram ao menos uma violação de tempo real;
- Percentual de Clientes Bloqueados por latência de alocação (**PcBl**): Percentual dos clientes bloqueados por demora na alocação de um novo servidor de transmissão;
- Banda agregada total utilizada (**AggBand**): Soma das bandas de rede de todas as máquinas virtuais no sistema dada em Mbps.

Essas métricas são contextualizadas com as métricas de desempenho dos componentes individuais descritas na Seção 4.3.2. No caso de um sistema de VoD, elas são: quantidade de clientes atendidos por máquina virtual e quantidade de máquinas virtuais por servidor.

4.4 Considerações Finais

Este capítulo detalha o trabalho realizado no projeto de um novo método para avaliação do desempenho de ambientes de IaaS utilizando um sistema de distribuição de VoD, o ElasticMovie. A principal característica do ElasticMovie na avaliação dos ambientes de IaaS é a utilização do Fator de Paciência como mecanismo de controle na avaliação da latência de alocação de novas máquinas virtuais. Esta escolha tem como consequência o surgimento de uma correlação entre as métricas da aplicação e do ambiente de IaaS, por exemplo, existe uma correlação entre a métrica **PcBl**

do serviço de transmissão vídeo e a métrica **TaR** do ambiente de IaaS, isto ocorre por que quanto maior for o tempo de alocação, maior é a probabilidade de clientes serem bloqueados.

Por fim, observe que uma aplicação elástica é uma aplicação energeticamente eficiente, pois ao manter em uso apenas os recursos necessários para atender a demanda corrente requerida do sistema computacional esta permite que os recursos liberados possam ser utilizados por outras aplicações ou mesmo desligados.

Capítulo 5

Avaliação Experimental IaaS

Neste capítulo são apresentados os resultados dos experimentos realizados na avaliação do novo *benchmark* proposto para ambientes de *Infrastructure as a Service* (IaaS), discutido na Seção 4.3.3. Sendo assim primeiramente são apresentados os resultados da avaliação das diferentes técnicas de virtualização no contexto da transmissão de vídeo sobre o *Hypertext Transfer Protocol* (HTTP), fornecendo assim o embasamento necessário para a avaliação dos ambientes de IaaS. Posteriormente os ambientes de IaaS Openstack e OpenNebula são avaliados utilizando o *benchmark* proposto.

5.1 Ambiente Experimental

O ambiente experimental utilizado é composto por oito nós computacionais homogêneos, interconectados por um *switch gigabit ethernet*, cujas características de *hardware* e *software* dos nós encontram-se descritas na Tabela 5.1 (com o tipo do componente na primeira coluna e o modelo utilizado na segunda). Um segundo *cluster* homogêneo composto por cinco nós é utilizado para gerar a carga de clientes nos experimentos feitos apresentados neste capítulo.

5.2 Avaliação das técnicas de virtualização utilizando um servidor de streaming de vídeos

Nesta seção estão sumarizados os resultados experimentais para transmissão de um vídeo com taxa de **2,8 Mbps**. Em todos os experimentos são realizadas **10** execuções de cada um dos experimentos e com intervalo de coleta do perfil da trans-

Tabela 5.1: Nó computacional Dual Xeon Hexa (Ivy Bridge-EN)

Componente	Modelo
Processador	Intel Xeon (6 cores) $E5-2420$ $1,90$ GHz
Duração Teórica do Ciclo	$0,53$ ns
Cache L2	15 MB
FSB	1333 MHz
Memória DRAM	$6 * 4$ GB DDR3 ECC
Interface de Rede	$2 * 10/100/1000$ Mbps
Unidade de Disco	1 TB SATA 7200 RPM
Sistema Operacional	Ubuntu 14.04 (3.13.0)
Versão do LXC	3.13.0
Versão do Xen	4.4.1
Versão do KMV	3.13.0

missão de **60** segundos. O experimento de transmissão utiliza o servidor HTTP **NGINX** com suporte para transmissão de *streaming* de vídeos codificados em **H.264** [110] e utilizado a ferramenta de *profiling* **OProfile** para se contabilizar os tipos de eventos (chamadas).

5.2.1 Transmissão sem virtualização

A execução do servidor de vídeo sem uso de técnicas de virtualização afere a capacidade efetiva, neste caso, definida como o total de clientes atendidos simultaneamente. Sendo observado que a capacidade de transmissão do Linux sobre o nó físico e do Domínio Privilegiado (Dom0) do Xen são as mesmas.

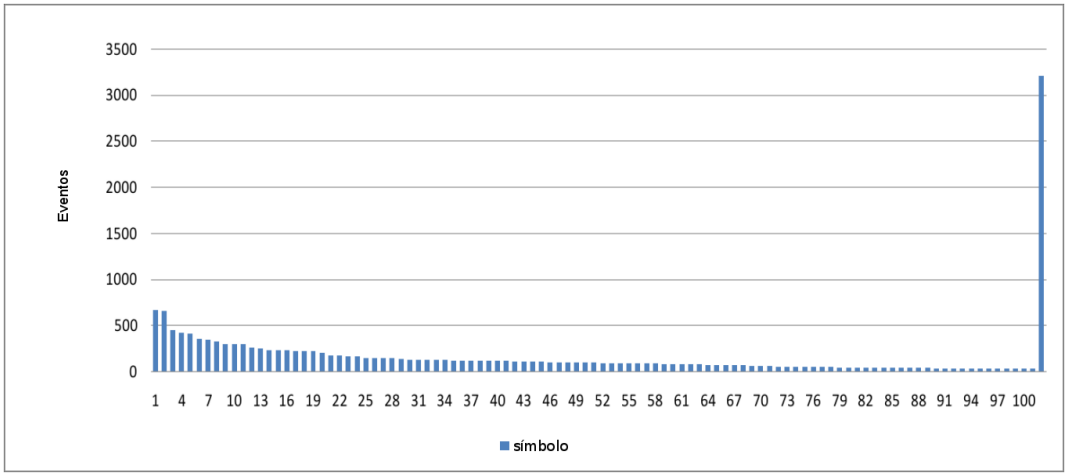


Figura 5.1: Distribuição de eventos para 320 clientes no servidor de vídeo sem virtualização

O servidor atende um total de **320 clientes** assistindo um vídeo com taxa de **2,8 Mbps**, onde os clientes entram no sistema sucessivamente com intervalos fixos entre

eles. Deste modo, a banda passante em uso no servidor, após a entrada do ultimo cliente, durante os experimentos é de 896 Mbps. A Figura 5.1 [1] apresenta a distribuição de eventos capturados pela ferramenta de *profiling* durante uma execução do sistema de distribuição de vídeos. Neste cenário são capturados para o perfil **16990** eventos divididos em **533** símbolos distintos durante os 60 segundos considerados. Na figura são apresentados apenas os 99 maiores símbolos e os demais eventos, estão agrupados sob o índice **100** do eixo horizontal.

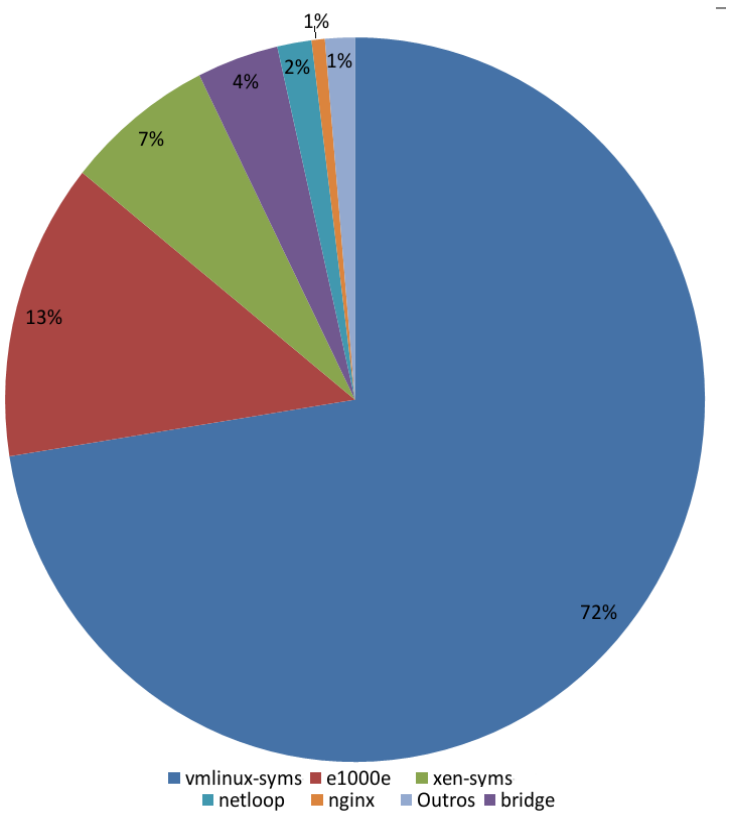


Figura 5.2: Amostragem de eventos para 320 clientes no servidor de vídeo sem virtualização

Os eventos de maior ocorrência na série de dados apresentada pela Figura 5.1 são os causados pelo envio de mensagens do *kernel* do Linux para a interface de rede como, por exemplo, os símbolos [2] dos eventos ligados à interrupção da interface de rede como o *e1000_irq_enable*, que aparece na Figura 5.1 como o evento classificado como índice **1** do eixo horizontal. Devido à quantidade de símbolos e eventos capturados, a Figura 5.1 apenas fornece a distribuição quantitativa dos mesmos. Na Figura 5.2, verifica-se quais são os sete programas que mais geram símbolos no perfil, sendo possível observar que o Dom0 (xen-syms) gera 7% dos eventos da Figura 5.1.

Na Figura 5.2, pode-se observar que os eventos ocorridos com maior frequência

[1] Uso números no rotulo para facilitar a leitura da figura
[2] Para o **OProfile** um símbolo pode ser tanto uma aplicação, como uma chamada de função

no intervalo de tempo considerado são os eventos ligados ao envio de pacotes pela interface de rede, sejam os vinculados à interface propriamente dita e classificados como e1000e que representam 13% do total ou os executados pelo sistema operacional como parte do processamento da pilha de protocolos com 72%. Por fim, é importante salientar que a aplicação de transmissão de *streaming* de vídeo (servidor web NGINX) gera apenas 1% do total de eventos no intervalo de interesse. Isso mostra que a maior parte do processamento é ligada ao dispositivo de entrada/saída, no caso, a interface de rede. É importante manter em mente esses resultados, pois os mesmos facilitarão a comparação com as outras técnicas de virtualização.

O perfil de uso dos núcleos de processamento foi coletado durante os experimentos descritos anteriormente, desta forma são apresentados os perfis de uso das unidades de processamento para os casos onde existem 100, 200 e 300 clientes recebendo vídeos do servidor (neste caso o número de clientes é igual ao número de fluxos mantidos pelo servidor) nas Figuras 5.3, 5.4 e 5.5, respectivamente. Por fim, é importante observar que o **NGINX** é um servidor HTTP baseado em eventos e deste modo não possui múltiplas *threads* ou processos para realizar o atendimento das requisições HTTP como no caso do Apache. Isso implica que o desempenho do servidor HTTP é diretamente ligado ao desempenho do núcleo de processamento e uso da hierarquia de memória. Se necessário, o uso de múltiplos núcleos para a utilização de 100% da interface de rede no **NGINX** é feito através do uso de *workers*.

A Figura 5.3 apresenta em vermelho a carga média de processamento no sistema e em azul a carga de processamento no núcleo em que o **NGINX** executa, para o intervalo de tempo entre 50 e 150 segundos de um perfil total de 300 segundos capturado durante os experimentos. Os clientes começavam a requisitar vídeos depois de 10 segundos e aos 50 segundos todos os clientes já estão no sistema. O pico observado na figura deve-se às requisições HTTP dos últimos clientes que chegaram e que começam a receber pedaços de vídeo com taxa superior a da exibição do vídeo. A Figura 5.3 também mostra que o servidor **NGINX** com a configuração utilizada consegue utilizar apenas um dos 12 núcleos de processamento disponíveis, o que não é problema com base no percentual de uso do núcleo.

A Figura 5.4 apresenta a carga média de processamento no sistema ainda em vermelho e a carga de processamento no núcleo em que o **NGINX** executa em azul, para o intervalo de tempo entre 50 segundos e 150 segundos de um perfil total de 300 segundos capturado durante os experimentos, sendo que os clientes começavam a entrar a requisitar vídeo depois de 10 segundos e aos 90 segundos todos os clientes já estão no sistema.

A Figura 5.5 apresenta a carga média de processamento no sistema em vermelho e a carga de processamento no núcleo em que o **NGINX** executa em azul, para o intervalo de tempo entre 50 segundos e 150 segundos de um perfil total de 300

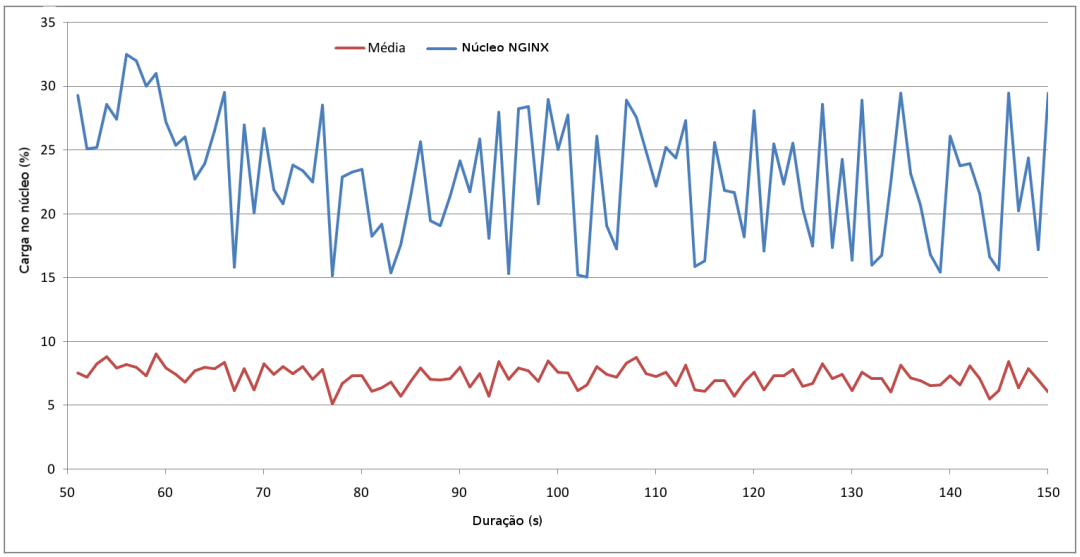


Figura 5.3: Amostragem da carga de processamento média e carga de processamento no núcleo em que executa o NGINX para 100 clientes no servidor de vídeo sem virtualização

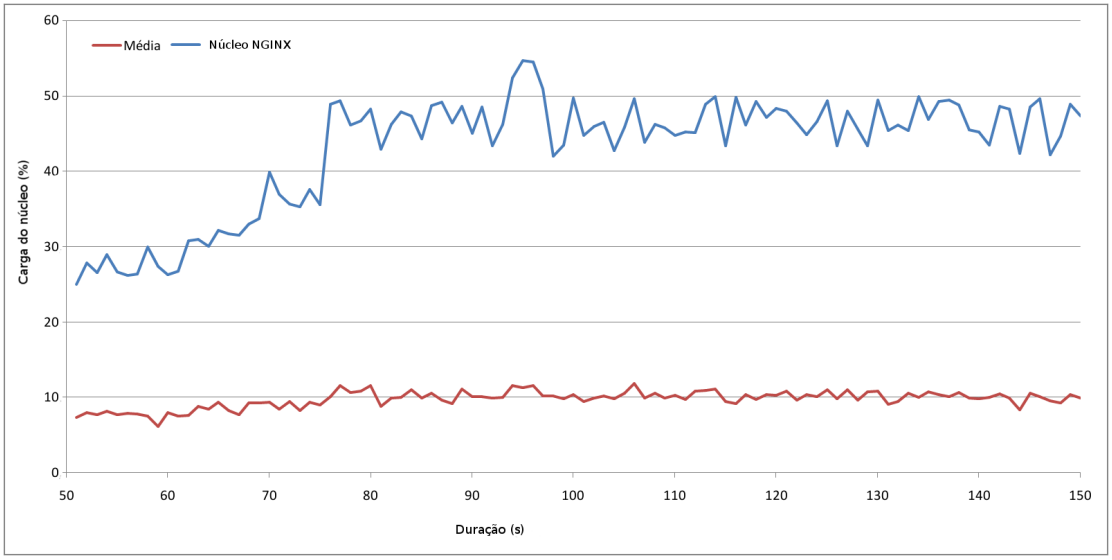


Figura 5.4: Amostragem da carga de processamento média e carga de processamento no núcleo em que executa o NGINX para 200 clientes no servidor de vídeo sem virtualização

segundos capturado durante os experimentos, sendo que os clientes começavam a entrar a requisitar vídeo depois de 10 segundos e aos 120 segundos todos os clientes já estão no sistema.

Por fim, uma análise das Figuras 5.3, 5.4 e 5.5, permite inferir que o custo computacional para realizar o atendimento das requisições HTTP para envio de vídeo é proporcional ao número de clientes que encontram-se recebendo vídeo do servidor no cenário experimental utilizado. Observe ainda que, apesar do servidor

Figura 5.5: Amostragem da carga de processamento média e carga de processamento no núcleo em que executa o NGINX para 300 clientes no servidor de vídeo sem virtualização

suportar enviar o vídeo para 320 clientes, para a análise de custo computacional, apenas são apresentados resultados para 100, 200 e 300 clientes.

5.2.2 Transmissão de vídeo usando a virtualização do tipo contêiner com o LXC

Os experimentos realizados utilizando o *Linux Containers* (LXC) como técnica de virtualização e executando o servidor HTTP **NGINX** dentro de um contêiner obtém resultados similares à execução do servidor diretamente sobre o Linux não virtualizado, não apresentando sobrecarga considerável, e conseguindo alcançar uma vazão efetiva de 320 clientes também para um vídeo com taxa de **2.8 Mbps**. Com base nestes resultados são apresentados para o servidor de vídeo executando dentro de um contêiner LXC apenas os totais de clientes atendidos. A Tabela 5.2 apresenta na primeira coluna o número de contêiners LXC que são criados, sendo na segunda coluna apresentando o total de clientes que são servidos por contêineres e a terceira coluna apresenta apenas o total de clientes atendidos pelo nó físico, ou seja, Total de Clientes = Número de VM's * Média de Clientes por VM.

Como pode ser observado na Tabela 5.2 o uso da tecnologia de contêiner, implementado pelo LXC, não apresenta sobrecarga significativa na transmissão dos fluxos de vídeo, permitindo ao servidor enviar vídeo para 320 clientes simultaneamente. De modo que, com base neste resultado e em descrições da implementação do LXC disponíveis na literatura, não foi necessário realizar uma avaliação de perfil para

Tabela 5.2: Sumário do experimento para o LXC com vídeo de taxa de 2.8 Mbps

Número de VM's	Média de Clientes por VM	Total de Clientes no *hardware*
1	320	320
2	160	320
4	80	320
8	40	320
320	1	320

essas execuções, pois os resultados experimentais obtidos mostram que a indireção introduzida nos descritores de processos e escalonamento das tarefas em dois níveis não impactam o desempenho da aplicação de transmissão de *streaming* de vídeo.

5.2.3 Transmissão de vídeo usando a virtualização do tipo paravirtualizada com o Xen

Os experimentos com o Monitor de Máquina Virtual (MMV) **Xen** possuem dois objetivos distintos. Em primeiro lugar deseja-se avaliar comparativamente os efeitos que a implementação da técnica de paravirtualização do **Xen** tem sobre um sistema de transmissão de vídeo e em segundo lugar verificar como se comporta essa mesma implementação quando o número de máquinas virtuais concorrentemente ativas no *hardware* é variado.

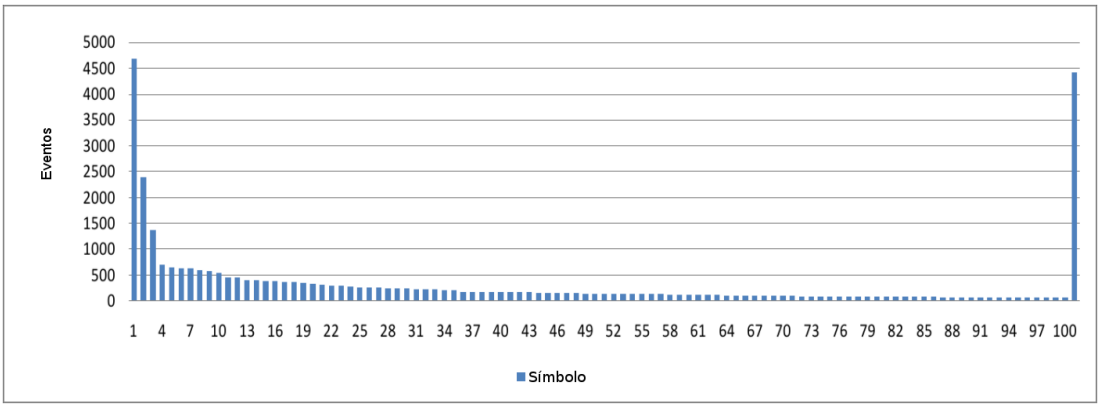


Figura 5.6: Distribuição de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais Xen e uma única máquina virtual

Como comentado anteriormente na Seção 3.2.3, no **Xen** os domínios não privilegiados não possuem acesso direto aos dispositivos do sistema computacional, sendo necessário passarem pelo domínio privilegiado. Isto pode ser observado nos experimentos pelo fato dos eventos que mais ocorrem no sistema durante os experimentos terem sido o de acesso a *grant table* do **Xen**. Na Figura 5.6 esses eventos estão

classificados como os eventos de ìndice **1** do eixo horizontal ou *do_grant_table_op*. A Figura 5.6 representa o experimento com apenas uma máquina virtual atendendo a todos os 242 clientes. Observe ainda, que como no cenário com o LXC, devido à grande quantidade de símbolos e eventos capturados durante o intervalo de interesse dos experimentos, a Figura 5.6 fornece a distribuição quantitativa dos mesmos.

Essa forma de controle ao acesso dos dispositivos imposta pela implementação de **Xen** faz com que a quantidade máxima de clientes suportados por uma única máquina virtual no *hardware* utilizado durante os experimentos nunca fosse superior a 242 clientes. A Tabela 5.3 apresenta os resultados obtidos para diferentes configurações no número de máquinas virtuais para o **Xen**, sendo na primeira coluna o número máquinas virtuais executadas concorrentemente, a segunda coluna contem o total de clientes servidos por cada máquina virtual nesta configuração e a terceira coluna apresenta apenas o total de clientes atendidos pelo no físico, ou seja, Total de Clientes = Número de VM's * Média de Clientes por VM. Ainda sobre a Tabela 5.3 os casos de tese onde são apresentados números de clientes por VM fracionários decorre de um desbalanceamento no número de cliente atendidos por máquina virtuai.

Tabela 5.3: Sumário do experimento para o Xen com vídeo de taxa de 2.8 Mbps

Número de VM's	Média de Clientes por VM	Total de Clientes no *hardware*
1	242	242
2	121	242
4	60, 5	242
8	30, 25	242

Com relação à quantidade de máquinas virtuais que são executadas concorrentemente sobre o hardware durante os experimentos sumarizados na Tabela 5.3 verifica-se que ao menos para o serviço de transmissão de vídeos, a execução de 12 máquinas virtuais afeta a capacidade do **Xen** de transmitir os vídeos é comprometida e ele passa a atender menos do que 242 clientes, sendo esse total de clientes dado pela capacidade de atendimento concorrente dos clientes na máquina virtual.

A Figura 5.7 apresenta o perfil geral do programas ou aplicativos que geram os eventos detectados pela ferramenta de **OProfile**. Nela é possível observar que a maior parte dos símbolos capturados pertencem ao **Xen**, sendo desse modo considerados como sobrecarga para o processamento.

A Figura 5.8 apresenta os sete programas ou símbolos com maior quantidades de eventos vistos na Figura 5.6. É possível observar na Figura 5.8 que os eventos que ocorrem com maior frequência no intervalo de tempo considerado são os eventos ligados ao gerenciamento do Xen para controle de acesso aos dispositivos de entrada/saída como o, *do_grant_table_op* e *__flush_tlb_mask*. Esse resultado explica para o

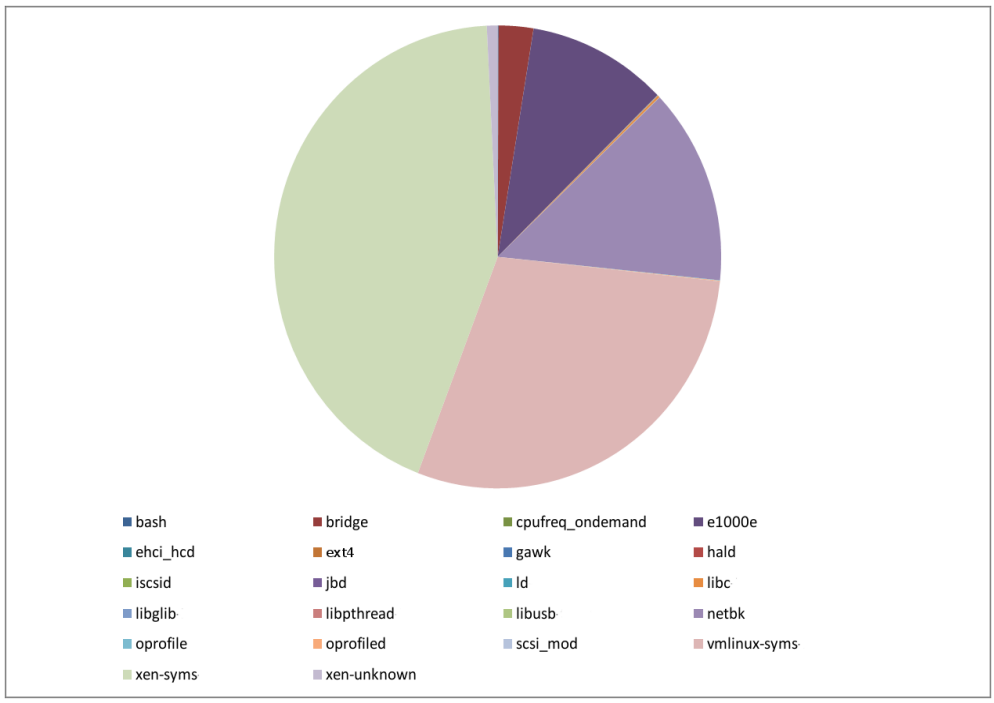


Figura 5.7: Aplicativos que geram os eventos detectados no sistema para 242 clientes do nó servidor de vídeo usando o monitor de máquinas virtuais Xen e uma única máquina virtual

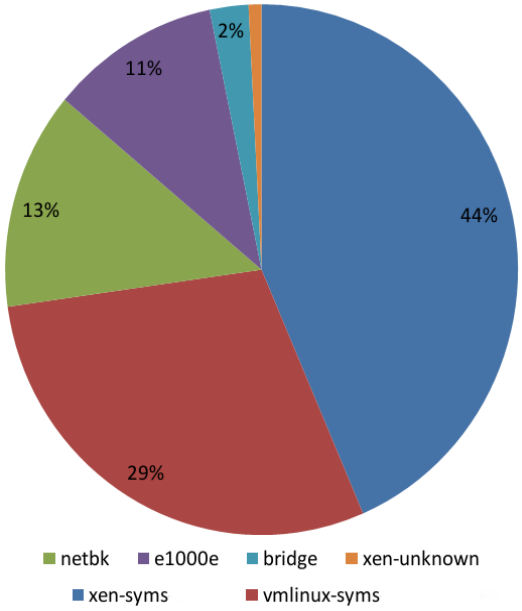


Figura 5.8: Amostragem de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais Xen e uma única máquina virtual

sistema computacional usado nos experimentos a incapacidade do **Xen** de utilizar mais da banda passante disponível no hardware. Nos experimentos apresentados, a banda passante total em uso pelo **Xen** nunca é superior a $677,6$ **Mbps** o que é

equivalente a aproximadamente 76% da capacidade total disponível no *hardware*.

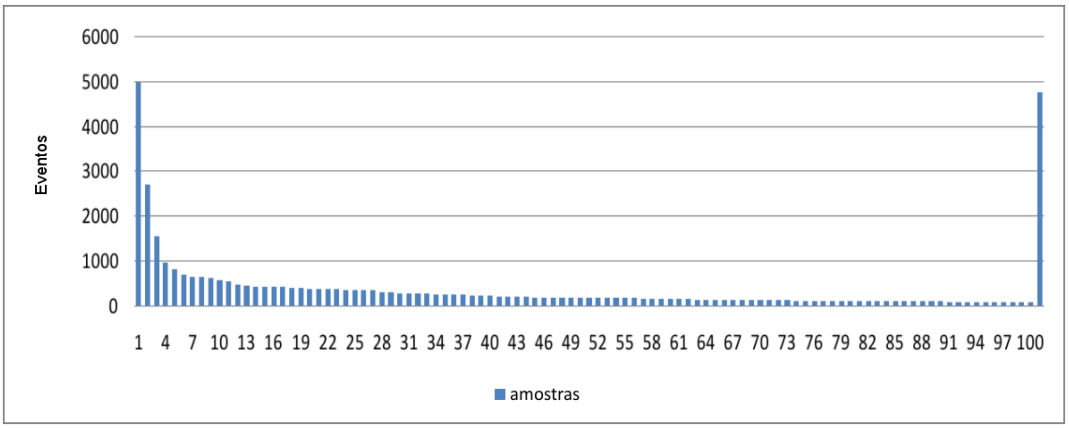


Figura 5.9: Distribuição de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais Xen e duas máquinas virtuais

No cenário experimental representado pela Figura 5.6, são capturados um total de 32287 eventos em um intervalo de 60 segundos, por sua vez o mesmo experimento no cenário da Figura 5.9 onde usam-se duas máquinas virtuais distintas, cada uma conseguindo tratar 121 clientes, a distribuição dos eventos que mais ocorrem no intervalo de interesse não se altera. A única diferença é a ocorrência de um número de eventos capturados ser 13% superior, 36493 eventos em um intervalo de 60 segundos.

5.2.4 Transmissão de vídeo usando a virtualização total com o KVM

Os experimentos com o monitor de máquinas virtuais ***K**ernel-based **V**irtual **M**achine* (KVM) buscam avaliar comparativamente os efeitos que a implementação da técnica de virtualização total do KVM causam em um sistema de transmissão de vídeo e verificar como se comporta essa mesma implementação quando o número de máquinas virtuais concorrentemente ativas no *hardware* é variado.

Como descrito anteriormente na Seção 3.2.2, no KVM os domínios não privilegiados não acessam diretamente os dispositivos de entrada/saída, realizando este acesso através do **QEMU** que executa no domínio privilegiado, isso pode ser observado nos experimentos pelo fato de os eventos que mais ocorrem no sistema serem ligados ao aplicativo **qemu-kvm** marcados na Figura 5.10 como os eventos de índice **1** do eixo horizontal. Note que a Figura 5.10 representa o experimento com apenas uma máquina virtual atendendo a todos os 242 clientes. Observe ainda, assim como no cenário com o LXC, devido a grande quantidade de símbolos e eventos capturados durante o intervalo de interesse dos experimentos, a Figura 5.10 fornece

a distribuição quantitativa dos mesmos.

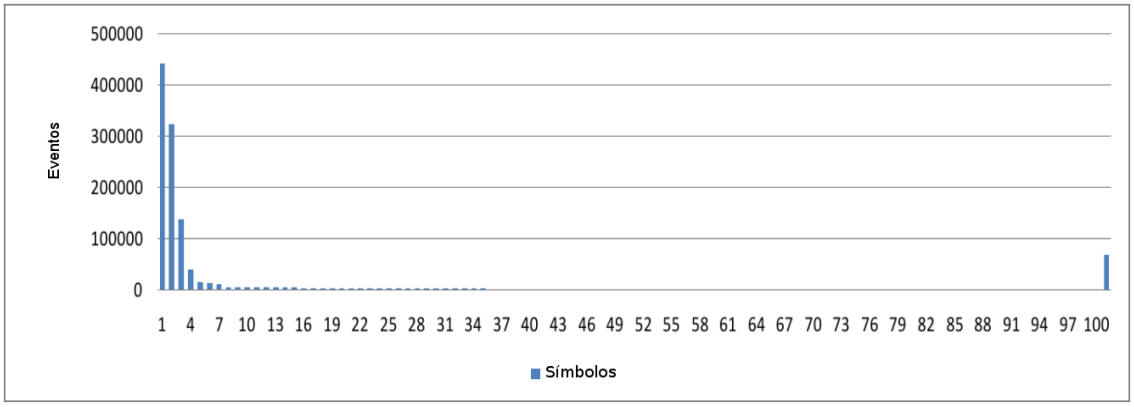


Figura 5.10: Distribuição de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais KVM e uma única máquina virtual

Essa forma de controle ao acesso dos dispositivos imposta pela implementação de KVM faz com que a quantidade máxima de clientes suportados por uma única máquina virtual no *hardware* utilizado nunca fosse superior a 242 clientes. A Tabela 5.4 apresenta os resultados obtidos para diferentes configurações no número de máquinas virtuais para o KVM, onde na primeira coluna é o número máquinas virtuais executadas concorrentemente, na segunda a média de clientes servidos por cada máquina virtual e a terceira coluna apresenta apenas o total de clientes atendidos pelo nó físico, ou seja, Total de Clientes = Número de VM's * Média de Clientes por VM. Ainda sobre a Tabela 5.4, os casos em que são apresentados o número de clientes fracionários significando que duas das máquinas virtuais recebem um cliente a mais que as demais.

Tabela 5.4: Sumário do experimento para o KVM com vídeo de taxa de 2.8 Mbps

Número de VM's	Média de Clientes por VM	Total de Clientes no *hardware*
1	242	242
2	141	282
4	80	320
8	40	320

Com relação à quantidade de máquinas virtuais que são executadas concorrentemente durante os experimentos sumarizados na Tabela 5.4, verifica-se que o KVM, ao contrário do **Xen**, consegue alcançar o número máximo de clientes que o servidor suporta com a utilização de quatro ou mais máquinas virtuais. Isso ocorre porque no KVM cada máquina virtual possui um processo no nível do usuário do domínio privilegiado responsável por realizar as suas operações de entrada/saída, o que possivelmente aumenta o grau de concorrência dentro do sistema e permite uma utilização melhor dos recursos de rede disponíveis.

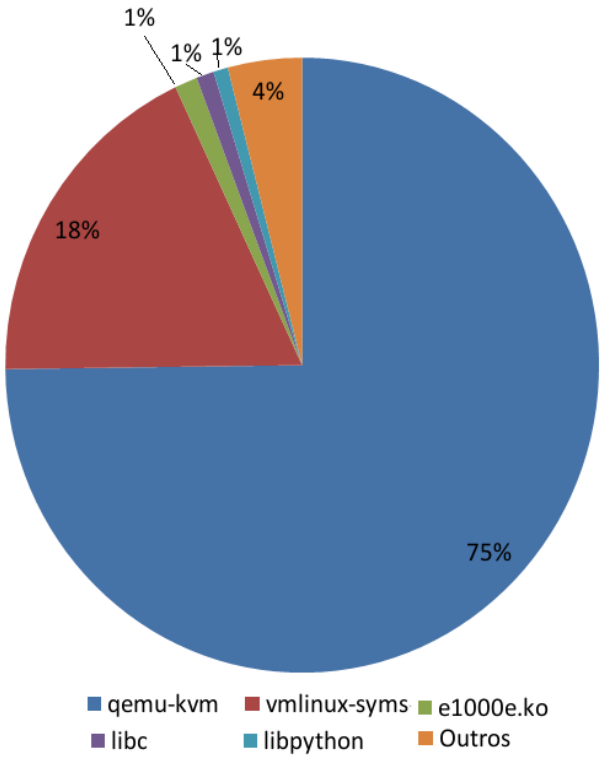


Figura 5.11: Amostragem de eventos para 242 clientes nó servidor de vídeo usando o monitor de máquinas virtuais KVM e uma única máquina virtual

A Figura 5.11 apresenta os sete programas ou símbolos com maior quantidades de eventos vistos na Figura 5.10. É possível observar na Figura 5.11 que os eventos que ocorrem com maior frequência no intervalo de tempo considerado são os eventos ligados ao gerenciamento do **QEMU-KVM** para controle ao acesso aos dispositivos de entrada/saída, eventos do processamento do *kernel* do Linux ligados a copia de páginas e do *driver* da interface de rede. Com o aumento no número de máquinas virtuais, é possível utilizar a capacidade total do servidor com relação ao número de clientes atendidos concorrentemente, nos experimentos apresentados a banda passante em uso por uma máquina virtual no KVM nunca é superior a $677,6$ **Mbps** o que é equivale a aproximadamente 76% da capacidade total disponível no *hardware*.

5.2.5 Discussão dos resultados da avaliação das técnicas de virtualização

Os resultados apresentados nesta seção permitem observar como as diferentes técnicas de virtualização afetam o desempenho de um sistema de transmissão de vídeo sobre o protocolo HTTP. Neste sentido, os resultados coletados através da ferramenta **OProfile** permitem concluir que a considerável diferença de desempenho ocorre devido às cópias de dados realizada pelas MMV para garantir o isola-

mento entre as diferentes máquinas virtuais. Como o LXC não necessita destas cópias adicionais, o desempenho do servidor HTTP não sofre impactos mensuráveis e apresenta resultados similares aos obtidos na execução direta sobre o ambiente não virtualizado, ou seja, consegue atender 320 clientes. Os ambientes virtualizados com o Xen e o KVM apresentam ambos um limite de apenas 242 clientes atendidos com uma única máquina virtual, consequência das copias de dados entre a máquina virtual e o MMV.

As diferenças entre o Xen e o KVM começam a aparecer quando se aumenta o número de máquinas virtuais executando concorrentemente. O ambiente computacional utilizando o Xen, não consegue atender mais de 242 clientes, em decorrência da serialização existente no acesso ao anel responsável pela comunicação entre as máquinas virtuais e o monitor. O KVM por sua vez apresenta uma implementação mais modular, criando uma aplicação responsável pelas operações de Entrada/Saída por máquina virtual, o que permite que o ambiente virtualizado com o KVM consiga atender os mesmos 320 clientes que o ambiente não virtualizado quando se aumenta o número de máquinas virtuais sobre o *hardware*. Com base nestes resultados os ambientes computacionais virtualizados mais promissores para o projeto de ambiente de IaaS são o LXC e o KVM.

5.3 Avaliação de ambientes de IaaS com o ElasticMovie

Esta seção apresenta a avaliação experimental do *benchmark* ElasticMovie previamente descrito na Seção 4.3.3 para os ambientes de IaaS. Os ambientes IaaS que são avaliados são compostos pelos nós descritos na Tabela 5.1, os *softwares* de gerenciamento de IaaS **OpenStack** e **OpenNebula**, com os sistemas de virtualização **Xen** e KVM.

A Figura 5.12 apresenta a organização do ambiente experimental usado na avaliação dos ambientes de IaaS utilizando o ElasticMovie como *benchmark*. Em todos os experimentos utilizando o ElasticMovie um cliente realiza duas requisições HTTP: a primeira é para solicitar a transmissão de um vídeo e a segunda para se remover do sistema de transmissão. O nó Controle Experimental é responsável por disparar os utilizados no experimento de acordo com a taxa de chega dada por uma variável aleatória exponencial, onde um cliente é uma instância do aplicativo wget, modificado para verificar se a taxa do vídeo é respeitada. O servidor HTTP utilizado pelos nós de transmissão é o **NGINX** e o vídeo utilizado nos experimentos tem duração de 30 minutos, codificado no padrão **H.264** com uma taxa de $2,698$ $Mbps$,

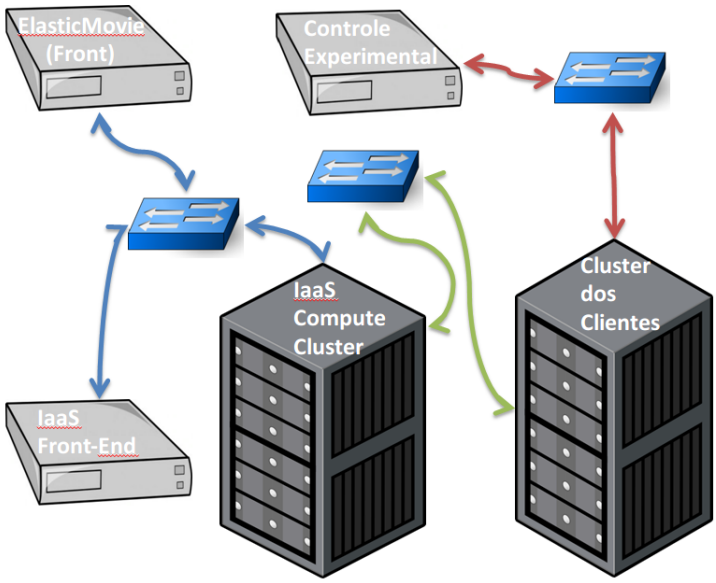


Figura 5.12: Organização física do ambiente experimental utilizado nos testes do ElasticMovie

a transmissão aos clientes utiliza uma taxa de $2,968$ $Mbps$.

5.3.1 Pré-avaliação da latência de alocação

Como visto na Seção 4.3.3, o método de avaliação de um ambiente de IaaS utilizando o ElasticMovie passa por uma avaliação preliminar dos componentes do ambiente computacional, na seção anterior foi realizada a avaliação preliminar das técnicas de virtualização no contexto de transmissão de vídeo sobre o protocolo HTTP. Nesta seção apresenta a avaliação preliminar da latência de alocação. Os experimentos são feitos utilizando uma imagem de máquina virtual de $2,8GB$ no formato QCOW2.

Tabela 5.5: Latência de alocação da máquina virtual

Ambiente	Média(s)	Desvio Padrão(s)
KVM OpenStack	$10,729$	$1,439$
Xen OpenStack	$71,965$	$6,531$
KVM OpenNebula	$59,447$	$1,312$
Xen OpenNebula	$101,016$	$14,749$

A Tabela 5.5 apresenta na primeira coluna o ambiente computacional, na segunda o tempo médio de alocação e na terceira coluna o desvio padrão correspondente, ambos calculados sobre 20 execuções. Como é possível observar o KVM apresenta

tempos de *boot* muito inferiores aos do Xen. Por sua vez, o ambiente usando OpenStack apresenta tempos muito menores dos obtidos com o OpenNebula. Isto ocorre, pois a configuração de rede do OpenStack é feita posteriormente ao *boot*, enquanto no OpenNebula ela ocorre durante o *boot* através dos *templates* de máquina virtual que necessitam ser configurados a priori no ambiente. Em termos práticos, a etapa de contextualização de rede do OpenStack acrescenta 30 segundos, intervalo esse que deve ser levado em consideração para o correto funcionamento da aplicação elástica.

5.3.2 Avaliação do ambiente OpenStack com o ElasticMovie

O ambiente de IaaS OpenStack é mantido pela versão Juno do *software* de gerenciamento de ambiente de IaaS. Nos experimentos descritos a seguir são realizadas 10 repetições para cada experimento usando dois padrões de carga distintos, ambos segundo um processo de chegada de clientes com distribuição Poisson: no primeiro o sistema recebe um total de 200 clientes com intervalo entre chegadas de 10 segundos; no segundo são utilizados 400 clientes com intervalo entre chegadas de 5 segundos.

KVM

A pré-avaliação da capacidade de rede do ambiente OpenStack com KVM mostra que cada instância de máquina virtual consegue manter uma taxa de transmissão de 40 $MBps$ usando o HTTP, o que confirma os resultados obtidos na Seção 5.2.4. Com base nesta informação e tendo em vista que o vídeo utilizado nos experimentos tem uma taxa de $2,968$ $Mbps$, o ElasticMovie (*Front*) é configurado para repassar até 100 clientes por nó de transmissão. Com base nos resultados obtidos na Seção 5.3.1, o gerador de clientes é configurado para gerar carga para os experimentos cujo Fator de Paciência do cliente iguais a 60 e 120 segundos.

O primeiro experimento de avaliação do ambiente de IaaS composto pelo KVM e OpenStack utiliza 200 clientes por repetição, totalizando 2.000 clientes, obtendo uma Quantidade total de requisições de Vídeo recebidas (**QtRvS**) para este experimento de 2.000 clientes o que representa 100% dos clientes gerados. A taxa de bloqueio dada pelo Percentual de Clientes Bloqueados por latência de alocação (**PcBl**) é de $2,05\%$, sendo que a melhor das execuções bloqueia dois clientes, enquanto que a pior bloqueia 6, em média são bloqueados $4,1$ clientes por execução com um desvio padrão de $1,446$.

A Figura 5.13 mostra o perfil de alocação dos recursos da melhor execução *benchmark* ElasticMovie, com relação à métrica **PcBl**, para o cenário experimental do OpenStack usando KVM, com Fator de Paciência de 120 segundos e taxa de chegada de $0,1$ clientes por segundo. Na figura, os pontos em azul representam o número

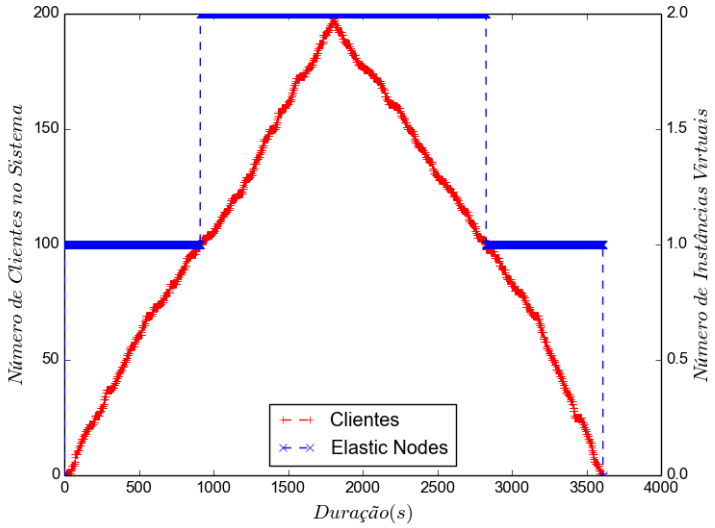


Figura 5.13: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenStack usando o KVM, com Fator de Paciência de 60 segundos e 200 clientes

de máquinas virtuais alocadas em um dado instante do experimento e os pontos de cor vermelha representam a quantidade de clientes assistindo vídeo em um dado instante. Esta execução bloqueia dois clientes de um total de 200 requisições recebidas o que representa uma taxa de bloqueio de 1%.

O segundo experimento de avaliação do ambiente de IaaS composto pelo KVM e OpenStack, utiliza 400 clientes por repetição, com taxa de chegada de $0,2$ clientes/s, totalizando 4.000 clientes, onde o **QtRvS** do experimento é 4.000 clientes o que representa 100% dos clientes gerados. A taxa de bloqueio dada pelo **PcBl** é de $23,57\%$, sendo que a melhor das execuções bloqueia 45 clientes, enquanto que a pior bloqueia 113 clientes, em média são bloqueados $94,3$ clientes por execução com um desvio padrão de $17,321$.

A Figura 5.14 mostra o perfil de alocação dos recursos da melhor execução do ElasticMovie, com relação à métrica **PcBl**, para este o cenário experimental do OpenStack usando KVM, com Fator de Paciência de 60 segundos e taxa de chegada de $0,2$ clientes por segundo. Na figura, dois momentos distintos chamam a atenção: o primeiro é um período onde ocorre uma série de alocações e liberações das máquinas virtuais no intervalo entre 1.000 e 2.000 segundos, o que é a causa do número alto de clientes bloqueados; o segundo ponto é a abrupta liberação dos recursos ao final, consequente dos intervalos de chegada para este experimento. Essa execução bloqueia 45 clientes de um total de 400 requisições recebidas o que representa uma taxa de bloqueio de $11,25\%$.

O terceiro experimento com o ambiente de IaaS composto pelo KVM e OpenStack, gera 200 clientes por repetição, com taxa de chegada de $0,1$ clientes/s, totalizando 2.000 clientes, onde o **QtRvS** do experimento é 2.000 clientes o que representa

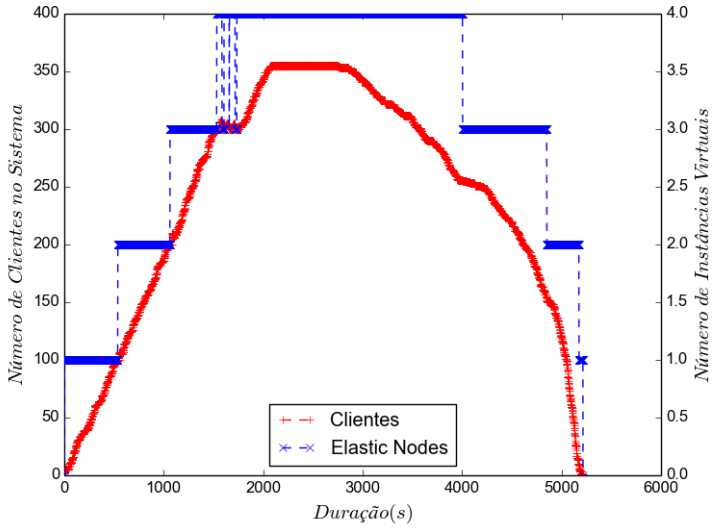


Figura 5.14: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenStack usando o KVM, com Fator de Paciência de 60 segundos e 400 clientes

100% dos clientes gerados. A taxa de bloqueio dada pelo **PcBl** é de $2,05\%$, sendo que a melhor das execuções bloqueia 25 clientes, enquanto que a pior bloqueia 10 clientes, em média são bloqueados $4,1$ clientes por execução com um desvio padrão de $3,176$.

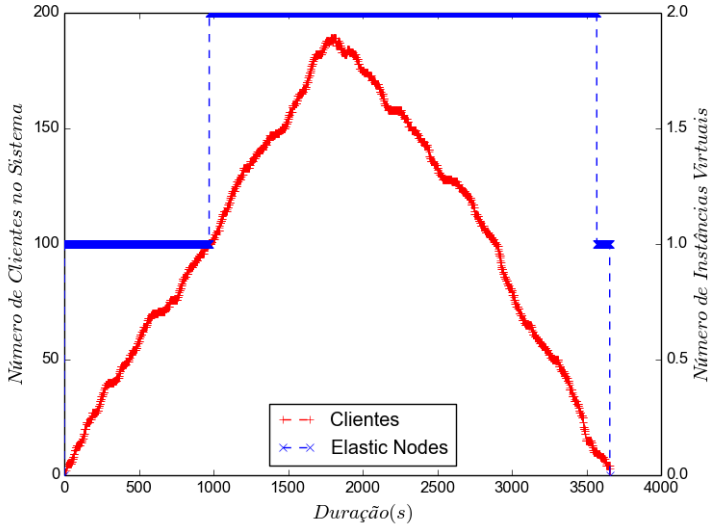


Figura 5.15: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenStack usando o KVM, com Fator de Paciência de 120 segundos e 200 clientes

A Figura 5.15 mostra o perfil de alocação dos recursos da melhor execução do ElasticMovie, com relação à métrica **PcBl**, para o cenário experimental do OpenStack usando KVM, com Fator de Paciência de 120 segundos e taxa de chegada de $0,1$ clientes por segundo. A longa demora em liberar recurso computacional vista

no experimento é consequência do padrão de chegas e do algoritmo do ElasticMovie que prioriza alocar um cliente no servidor que estiver mais ocupado. Esta execução bloqueia 2 clientes de um total de 200 requisições recebidas o que representa uma taxa de bloqueio de 1%.

O ultimo experimento utilizando a combinação OpenStack e KVM é realizado com uma taxa de chega de $0,2$ clientes/s, totalizando 400 clientes por repetição, onde o **QtRvS** do experimento é 4.000 clientes o que representa 100% dos clientes gerados. A taxa de bloqueio dada pelo **PcBl** é de $8,3\%$, sendo que a melhor das execuções bloqueia 16 clientes, enquanto que a pior bloqueia 43 clientes, em média são bloqueados $33,2$ clientes por execução com um desvio padrão de $7,291$.

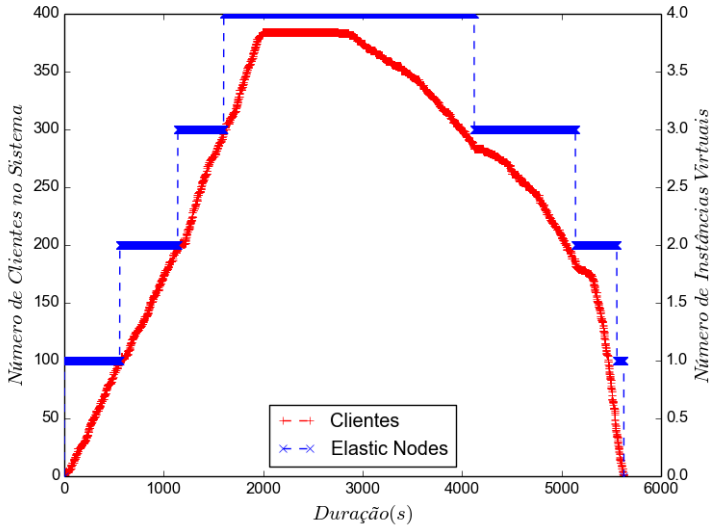


Figura 5.16: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenStack usando o KVM, com Fator de Paciência de 120 segundos e 400 clientes

A Figura 5.16 mostra o perfil de alocação dos recursos da melhor execução do *benchmark* ElasticMovie, com relação à métrica **PcBl**, para o cenário experimental do OpenStack usando KVM, com Fator de Paciência de 120 segundos e taxa de chegada de $0,2$ clientes por segundo. Esta execução bloqueia 16 clientes de um total de 400 requisições recebidas o que representa uma taxa de bloqueio de 4%, sendo possível observar que, apesar do experimento não ser aquele com a menor taxa de bloqueio, a quantidade elevada de clientes faz com que as curvas de demanda e oferta de recursos computacionais fiquem mais próximas.

Xen

Como no caso do KVM, a pré-avaliação da capacidade de rede do ambiente OpenNebula com Xen definiu como limite 100 clientes por máquina virtual, logo a configuração do ElasticMovie (*Front*) permanece inalterada. Com base nos resultados

obtidos na Seção 5.3.1, o gerador de clientes é configurado para gerar carga para os experimentos com Fator de Paciência do cliente igual a 120 segundos.

No primeiro experimento de avaliação do ambiente de IaaS composto pelo Xen e OpenStack, gera 200 clientes por repetição, totalizando 2.000 clientes, a **QtRvS** para este experimento é de 1.992 clientes o que representa $99,6\%$ dos clientes gerados, onde em média são atendidos $199,2$ clientes com um desvio padrão de $1,6$. A taxa de bloqueio dada pelo **PcBl** é de $1,908\%$, sendo que a melhor das execuções não bloqueia clientes, enquanto que a pior bloqueia 7, em média são bloqueados $3,8$ clientes por execução com um desvio padrão de $2,786$.

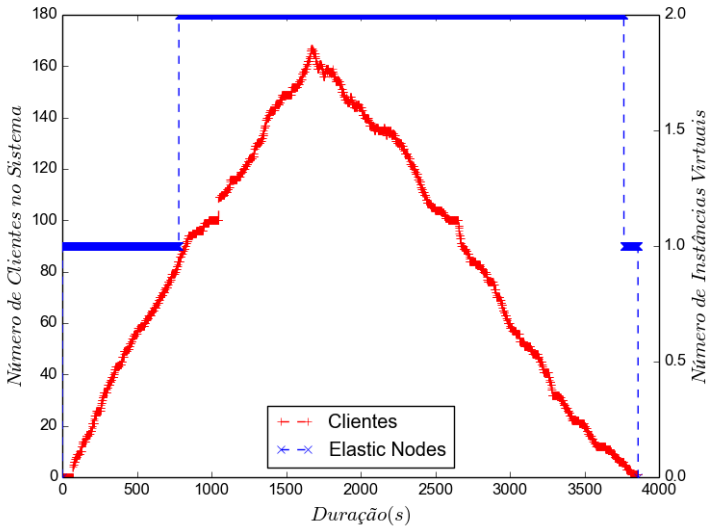


Figura 5.17: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenStack usando o Xen, com Fator de Paciência de 120 segundos e 20 clientes

A Figura 5.17 mostra o perfil de alocação dos recursos da melhor execução do *benchmark* ElasticMovie, com relação à métrica **PcBl**, para o cenário experimental do OpenStack usando Xen, com Fator de Paciência de 120 segundos e taxa de chegada de $0,1$ clientes por segundo. Esta execução é a única em todo conjunto experimental que consegue atender todas as requisições geradas, como consequência da maior duração do Fator de Paciência e da baixa taxa de requisições. Por sua vez, a longa demora na liberação do recurso computacional é similar ao visto na Figura 5.15, ocorrendo pelo mesmo motivo. A descontinuidade observada próxima a 1000 segundos é decorrente do processo de Poisson que rege a entrada de clientes no experimento.

O segundo experimento de avaliação do ambiente de IaaS composto pelo Xen e OpenStack utiliza uma taxa de entrada de clientes de $0,2$ clientes/s, gerando 400 clientes por repetição, totalizando 4.000 clientes, a **QtRvS** para este experimento é de 3.988 clientes o que representa $99,7\%$ dos clientes gerados, onde em média são atendidos $398,8$ clientes com um desvio padrão de $1,47$. A taxa de bloqueio dada

pelo **PcBl** é de $2,1063\%$, sendo que a melhor das execuções bloqueia 5 clientes, enquanto que a pior bloqueia 13, em média são bloqueados $3,441$ clientes por execução com um desvio padrão de $1,47$.

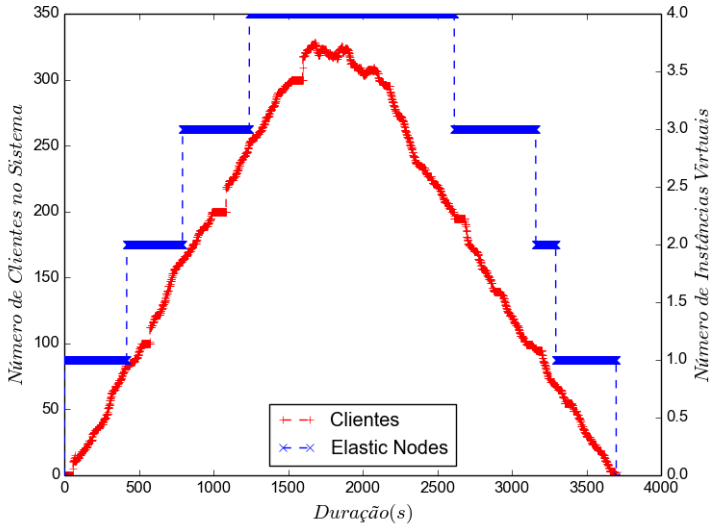


Figura 5.18: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenStack usando o Xen, com Fator de Paciência de 120 segundos e 400 clientes

A Figura 5.18 mostra o perfil de alocação dos recursos da melhor execução do *benchmark* ElasticMovie, com relação à métrica **PcBl**, para o cenário experimental do OpenStack usando Xen, com Fator de Paciência de 120 segundos e taxa de chegada de $0,2$ clientes por segundo. Esta execução bloqueia 5 clientes de um total de 400 requisições recebidas o que representa uma taxa de bloqueio de $1,25\%$. É possível observar a semelhança das curvas deste experimento com as curvas da Figura 5.16, consequência do uso de um Fator de Paciência maior.

Considerações sobre a avaliação usando o ElasticMovie no OpenStack

A Tabela 5.6 apresenta um sumário dos experimentos realizados com o ElasticMovie no ambiente computacional gerenciado pelo OpenStack, onde todos os valores apresentados representam as médias de 10 execuções. A coluna Fator de Paciência (**FP**), mostra que apenas o KVM conseguiu executar com Fator de Paciência de 60 segundos, sendo esse o menor Fator de Paciência possível de ser utilizado no ambiente com OpenStack. A diferença entre o menor Fator de Paciência utilizado e a latência média de alocação apresentada na Tabela 5.5 é causada pela forma como o OpenStack contextualiza a interface de rede virtual das suas VM's: essa contextualização ocorre depois da VM ser inicializada, durante o processo de *boot* do sistema operacional através de um *daemon* de sistema.

Ainda na Tabela 5.6, agora observando apenas os experimentos com F.P. igual

Tabela 5.6: Resumo da avaliação usando ElasticMovie no OpenStack - Caso Médio

Ambiente	Virt.	Fator de Paciência (FP)	**QtRvS**	**PcBl**
OpenStack	KVM	60	200	$2,05\%$
OpenStack	KVM	60	400	$23,57\%$
OpenStack	KVM	120	200	$2,05\%$
OpenStack	KVM	120	400	$8,30\%$
OpenStack	Xen	120	$199,2$	$1,91\%$
OpenStack	Xen	120	$398,8$	$2,11\%$

a 120 é possível observar que os dois melhores resultados obtidos são o do Xen e KVM, ambos com 200 clientes. O Xen apresenta um resultado melhor de acordo com a definição da métrica **PcBl**, contudo o Xen, ao contrário do KVM, não conseguiu atender todos os clientes gerados. Com relação ao uso do Fator de Paciência presente na aplicação de transmissão de vídeo, os resultados obtidos com o OpenStack mostram que este consegue capturar os impactos causados pelas técnicas de virtualização e alteração na demanda computacional exigida do sistema (número de clientes gerados), o que fornece indícios da viabilidade do uso da aplicação ElasticMovie como um *benchmark* para ambiente de IaaS.

5.3.3 Avaliação do ambiente OpenNebula com o ElasticMovie

O ambiente de IaaS OpenNebula é mantido com sua versão 4.10. Nos experimentos descritos a seguir são realizadas 10 repetições para cada um dos experimentos usando dois padrões de carga distintos, ambos seguindo um processo de chegada de clientes com distribuição Poisson: no primeiro o sistema recebe um total de 200 clientes com intervalo entre chegadas de 10 segundos; no segundo são utilizados 400 clientes com intervalo entre chegadas de 5 segundos.

KVM

A pré-avaliação da capacidade de rede do ambiente OpenNebula com KVM mostra que cada máquina virtual consegue atender apenas 40 clientes assistindo um vídeo com taxa de $2,968$ $Mbps$, assim o ElasticMovie (*Front*) é configurado para repassar até 40 clientes para cada nó de transmissão. Com base nos resultados obtidos na Seção 5.3.1, o gerador de clientes é configurado para gerar carga somente para os experimentos cujo Fator de Paciência do usuário é igual há 120 segundos.

O primeiro experimento de avaliação do ambiente de IaaS composto pelo KVM e OpenNebula utiliza 200 clientes por repetição, totalizando 2.000 clientes, contudo **QtRvS** para este experimento é de 1952 clientes o que representa $97,6\%$ dos clientes

gerados, onde em média são atendidos 195, 2 clientes com um desvio padrão de 3, 112. A taxa de bloqueio dada por **PcBl** é de 9, 734%, sendo que a melhor das execuções bloqueia 10 clientes, enquanto que a pior bloqueia 27, em média são bloqueados 19 clientes por execução com um desvio padrão de 3, 112.

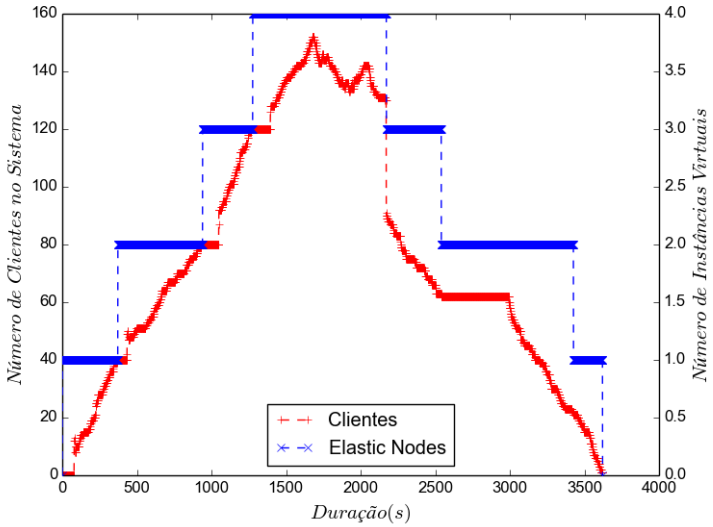


Figura 5.19: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenNebula usando o KVM, com Fator de Paciência de 120 segundos e 200 clientes

A Figura 5.19 apresenta o resultado da melhor execução do ElasticMovie para o cenário experimental do OpenNebula usando KVM, com Fator de Paciência de 120 segundos e taxa de chegada de 0, 2 clientes por segundo. É possível observar que ElasticMovie, ajusta-se rapidamente às mudanças na demanda computacional exigidas. A diferença entre os instantes que novos nós de transmissão são adicionados e os clientes passam a estar no sistema, como visto na figura na alocação que ocorre próxima ao ponto referente a 1.000 segundos, é consequência da latência de alocação, ou seja, o cliente que causa a criação de uma nova instância virtual somente entra no sistema depois que esta instância estiver criada, e apta a transmitir vídeo. Esta execução bloqueia 10 clientes de um total de 192 requisições recebidas o que representa uma taxa de bloqueio de 5, 208%.

A Figura 5.20 mostra o perfil de alocação dos recursos da pior execução do ElasticMovie, com relação à métrica **PcBl**, para este ambiente de IaaS. É possível perceber que nessa execução a liberação dos recursos é mais lenta, isto ocorre devido à distribuição dos clientes nos diferentes servidores de transmissão, pois em virtude da tentativa de minimizar o número de recursos virtuais, o ElasticMovie aloca um novo cliente no servidor de transmissão que estiver mais carregado. É possível ainda observar nessa figura que próximo a 1.500 segundos de duração do experimento ocorre a liberação de um servidor de transmissão, que é seguido de uma nova alocação. Este efeito é consequência da escolha feita no projeto do Elas-

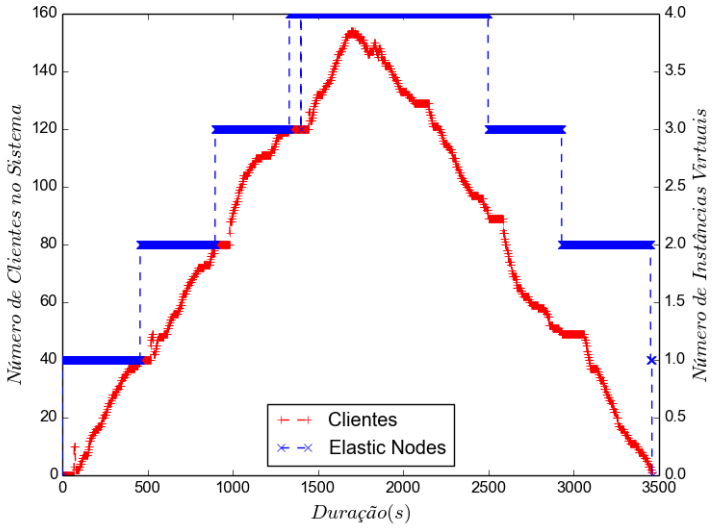


Figura 5.20: Pior execução *benchmark* ElasticMovie para o ambiente OpenNebula usando o KVM, com Fator de Paciência de 120 segundos e 200 clientes

ticMovie de não utilizar preditores para a chegada de clientes, pois esta abordagem permite avaliar melhor o impacto que a latência de alocação tem sobre a taxa de bloqueio. Ainda, nesta figura, é possível observar dois picos rápidos, próximos ao zero e logo após o instante 500 segundos, ambos representam clientes bloqueados devido à latência de alocação. Esta execução bloqueia 27 clientes de um total de 196 requisições recebidas o que representa uma taxa de bloqueio de $13,775\%$.

O segundo experimento de avaliação do ambiente de IaaS composto pelo KVM e OpenNebula, utiliza 400 clientes por repetição, com taxa de chegada de $0,2$ clientes/s, totalizando 4.000 clientes, onde o **QtRvS** do experimento é 3903 clientes o que representa $97,57\%$ dos clientes gerados, onde em média são atendidos $390,3$ clientes com um desvio padrão de $4,662$. A taxa de bloqueio dada por **PcBl** é de $12,426\%$, sendo que a melhor das execuções bloqueia 14 clientes, enquanto que a pior bloqueia 76, em média são bloqueados $48,5$ clientes por execução com um desvio padrão de $25,505$.

A Figura 5.21 mostra o perfil de alocação dos recursos para melhor execução do *benchmark* ElasticMovie no cenário com 400 clientes. Nesta figura dois pontos chamam a atenção: o primeiro ocorre no intervalo entre 1.000 s e 1.500 s, nele é possível observar uma alocação e liberação de recursos, isto ocorre devido a chega de um cliente que vem a ser bloqueado, como é mostrado na figura; o segundo ponto é a rápida saída dos clientes que é consequência do platô que ocorre após 1.500 segundos. Esta execução bloqueia 14 clientes de um total de 393 requisições recebidas o que representa uma taxa de bloqueio de $3,562\%$.

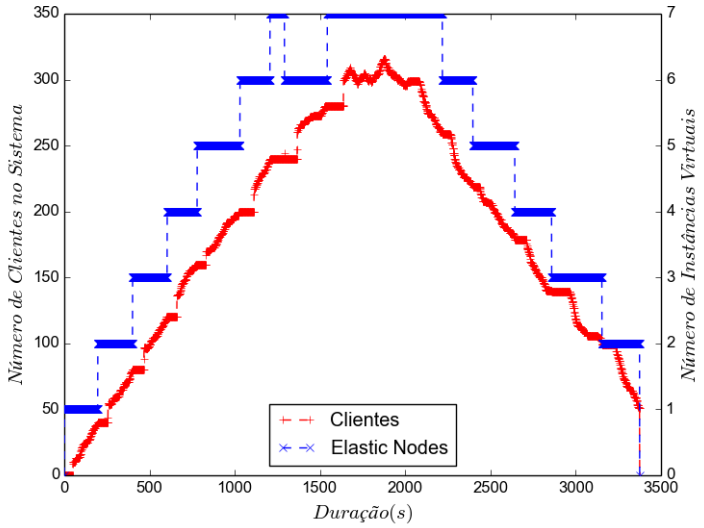


Figura 5.21: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenNebula usando o KVM, com Fator de Paciência de 120 segundos e 400 clientes

Xen

Como no caso do KVM, a pré-avaliação da capacidade de rede do ambiente OpenNebula com Xen definiu como limite 40 clientes por máquina virtual, logo a configuração do ElasticMovie (*Front*) permanece inalterada. E mesmo estando mais próxima do limite do Fator de Paciência, a latência de alocação obtida na Seção 5.3.1, ainda mantém o gerador de clientes é configurado para gerar carga somente para os experimentos utilizando o Fator de Paciência do usuário é igual há 120 segundos.

O primeiro experimento de avaliação do ambiente de IaaS composto pelo Xen e OpenNebula, trata 200 clientes por repetição, totalizando 2.000 clientes, contudo **QtRvS** para este experimento é de 2.000 clientes o que representa 100% dos clientes gerados. A taxa de bloqueio dada pelo **PcBl** é de $2,1\%$, sendo que a melhor das execuções bloqueia apenas um cliente, enquanto que a pior bloqueia 9, em média são bloqueados $4,2$ clientes por execução com um desvio padrão de $3,2496$.

A Figura 5.22 apresenta o resultado da melhor execução do *benchmark* ElasticMovie para o cenário experimental do OpenStack usando Xen, com Fator de Paciência de 120 segundos e taxa de chegada de $0,1$ clientes por segundo. É possível observar que o ElasticMovie, apesar do aumento na latência de alocação de máquinas virtuais, o sistema mantém a curva de recursos alocados próxima a curva da demanda exigida pelas transmissões para os clientes. Esta execução bloqueia um cliente de um total de 200 requisições recebidas o que representa uma taxa de bloqueio de $0,5\%$.

O segundo experimento de avaliação do ambiente de IaaS composto pelo Xen e OpenNebula, utiliza 400 clientes por repetição, com uma taxa de chegada de $0,2$

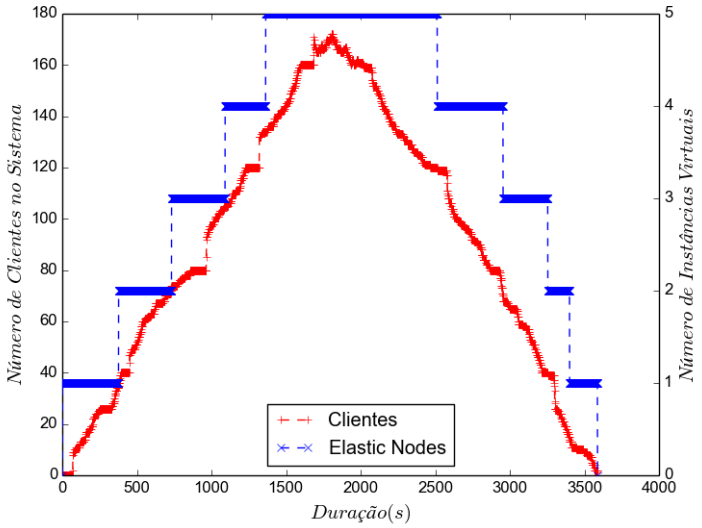


Figura 5.22: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenNebula usando o Xen, com Fator de Paciência de 120 segundos e 200 clientes

clientes/s, totalizando 4.000 clientes, onde o **QtRvS** do experimento é 3990 clientes o que representa 99, 75% dos clientes gerados, onde em média são atendidos 399 clientes com um desvio padrão de 2, 0. A taxa de bloqueio dada por **PcBl** é de 3, 007%, sendo que a melhor das execuções bloqueia apenas um cliente, enquanto que a pior bloqueia 28, em média são bloqueados 12 clientes por execução com um desvio padrão de 9, 338.

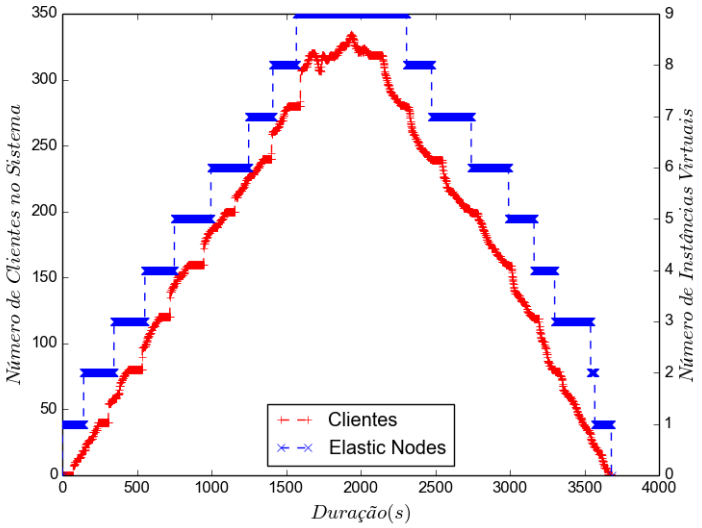


Figura 5.23: Melhor execução do *benchmark* ElasticMovie para o ambiente OpenNebula usando o Xen, com Fator de Paciência de 120 segundos e 400 clientes

A Figura 5.23 mostra o perfil de alocação dos recursos para melhor execução do *benchmark* ElasticMovie no cenário com 400 clientes com o Xen. Nesta o ponto que

mais chama a atenção é a rápida liberação da máquina virtual que ocorre em torno de 3.500 segundos, consequência da oscilação vista na curva de cliente entre 1.500 e 2.000 segundos. Como no cenário anterior a curva de recursos se manteve próxima à curva da demanda. Esta execução bloqueia apenas um cliente de um total de 400 requisições recebidas o que representa uma taxa de bloqueio de $0,25\%$.

Considerações sobre a avaliação usando o ElasticMovie no OpenNebula

A Tabela 5.7 apresenta um sumário dos experimentos realizados com o ElasticMovie no ambiente computacional gerenciado pelo OpenNebula, onde todos os valores apresentados representam as médias de 10 execuções. A coluna **F.P** (Fator de Paciência), mostra que apenas os experimentos com Fator de Paciência igual a 120 segundos executam com sucesso no ambiente de IaaS gerenciado pelo OpenNebula, sendo isso consequência da avaliação preliminar da latência média de alocação apresentada na Tabela 5.5.

Tabela 5.7: Resumo da avaliação usando ElasticMovie no OpenNebula - Caso Médio

Ambiente	Virt.	Fator de Paciência (FP)	**QtRvS**	**PcBl**
OpenNebula	KVM	120	$195,2$	$9,73\%$
OpenNebula	KVM	120	$390,3$	$12,426\%$
OpenNebula	Xen	120	200	$2,10\%$
OpenNebula	Xen	120	399	$3,01\%$

Ainda, na Tabela 5.7, é possível observar que nos ambientes de IaaS gerenciados pelo OpenNebula o Xen obteve resultados melhores, contudo é importante salientar que para ambas as técnicas de virtualização os melhores resultados obtidos são aqueles em que a taxa de chega de clientes é de $0,1$ clientes/s. Com relação ao uso do Fator de Paciência presente na aplicação de transmissão de vídeo os resultados obtidos com o OpenNebula mostram que este consegue capturar o impacto causados pela alteração na demanda computacional exigida do sistema (número de clientes gerados) nos ambientes de IaaS utilizados, o que fornece indícios da viabilidade do uso da aplicação ElasticMovie como um *benchmark* para ambiente de IaaS.

5.4 Aplicação da Metodologia para avaliação de desempenho de ambientes IaaS

A Tabela 5.8 apresenta de forma consolidada os resultados previamente mostrados nas Tabelas 5.6 e 5.7 permitindo assim uma comparação direta entre as diferentes combinações de *software* de gerenciamento de ambiente de IaaS e as técnicas de

virtualização. Tomando como base apenas os resultados apresentados, e usando a métrica **PcBl**, o melhor ambiente avaliado foi o OpenStack utilizando o Xen como técnica de virtualização. Contudo é importante relembrar que o método de avaliação proposto na Seção 4.3 para ser utilizado junto com o ElasticMovie é composto por três etapas, onde os resultados obtidos nas duas primeiras etapas são utilizados para configurar a execução do ElasticMovie e que também podem ser utilizados como critérios de desempate.

Tabela 5.8: Resumo da avaliação usando ElasticMovie - Caso Médio

Ambiente	Virt.	Fator de Paciência (FP)	**QtRvS**	**PcBl**
OpenStack	KVM	60	200	$2,05\%$
OpenStack	KVM	60	400	$23,57\%$
OpenStack	KVM	120	200	$2,05\%$
OpenStack	KVM	120	400	$8,30\%$
OpenStack	Xen	120	$199,2$	$1,91\%$
OpenStack	Xen	120	$398,8$	$2,11\%$
OpenNebula	KVM	120	$195,2$	$9,73\%$
OpenNebula	KVM	120	$390,3$	$12,426\%$
OpenNebula	Xen	120	200	$2,10\%$
OpenNebula	Xen	120	399	$3,01\%$

Com base nos resultados apresentados na Tabela 5.8, na avaliação preliminar da largura de banda utilizada do servidor apresentada na Seção 5.2 e na avaliação preliminar da latência de alocação de recursos apresentada na Tabela 5.5 o melhor ambientes de IaaS é o OpenStack com KVM. O ambiente formado pelo OpenStack com KVM é o único que consegue responder quando o Fator de Paciência é menor do que 120 segundos. Sendo que a análise dos experimentos mostra que o aumento da métrica **PcBl** foi consequência do procedimento de contextualização de rede da máquina virtual. Observe ainda que as semelhanças observáveis nas Figuras 5.18, 5.16 e 5.17, 5.15 respectivamente, indicam que o uso de um valor elevado para o Fator de Paciência, tende a reduzir a capacidade do ElasticMovie em avaliar e diferenciar os ambientes de IaaS.

5.5 Estimando o custo energético do *cluster* de IaaS

Um importante conceito que falta avaliar dentro do escopo dos ambientes de IaaS é o perfil do consumo energético do ambiente experimental, especificamente a sua eficiência energética. O consumo energético é estimado através de um *nobreak* APC Back-UPS 1500 RS, dedicado ao servidor em teste, que fornece 865 $Watts$ em suas

linhas de saída. A escolha deste *nobreak* é decorrente da presença de um circuito que avalia a demanda de energia dos equipamentos ligados nele e da viabilidade de acessar esta informação através da porta serial do mesmo.

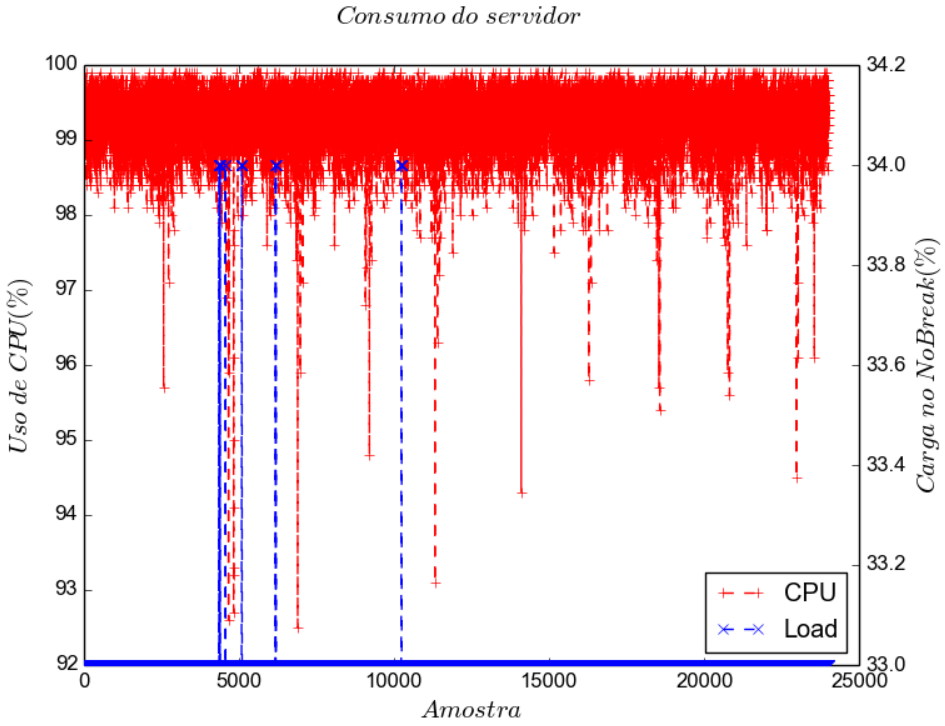


Figura 5.24: Consumo do servidor quando em pleno uso

A Figura 5.24 apresenta o perfil de consumo de energia e uso de CPU quando o servidor está sendo utilizado ao máximo. A carga de processamento é gerada através da transcodificação continua de múltiplos vídeos com o programa FFmpeg [111] e as amostras são coletadas com um intervalo de $0,5\ s$, com um total de 24.020 amostras coletadas em um intervalo de $3,336$ horas. Na figura os pontos de cor vermelha representam a média do consumo dos processadores enquanto os de cor azul representam a carga imposta pelo servidor sobre o *nobreak* (consumo de energia). Como pode ser observado, quando a demanda por processamento está próxima de 100% a carga sobre o *nobreak* fica em torno de 33% indo a 34% em alguns picos. A média da carga sobre nobreak é de $33,002\%$, o que representa um consumo médio de $285,467\ Watts$, o que no ambiente experimental descrito na Seção 5.1 representa um consumo total de $2.283,738\ Watts$.

A Figura 5.25 apresenta o perfil de consumo de energia e uso de CPU quando o servidor está ocioso. As amostras são coletadas novamente com um intervalo de $0,5\ s$, somando um total de 24.020 amostras coletadas em um intervalo de $3,336$ horas. Como pode ser observado apesar de os processadores estarem a maior parte do tempo ocioso, com exceção de alguns picos de 3% e um 6%, a carga imposta

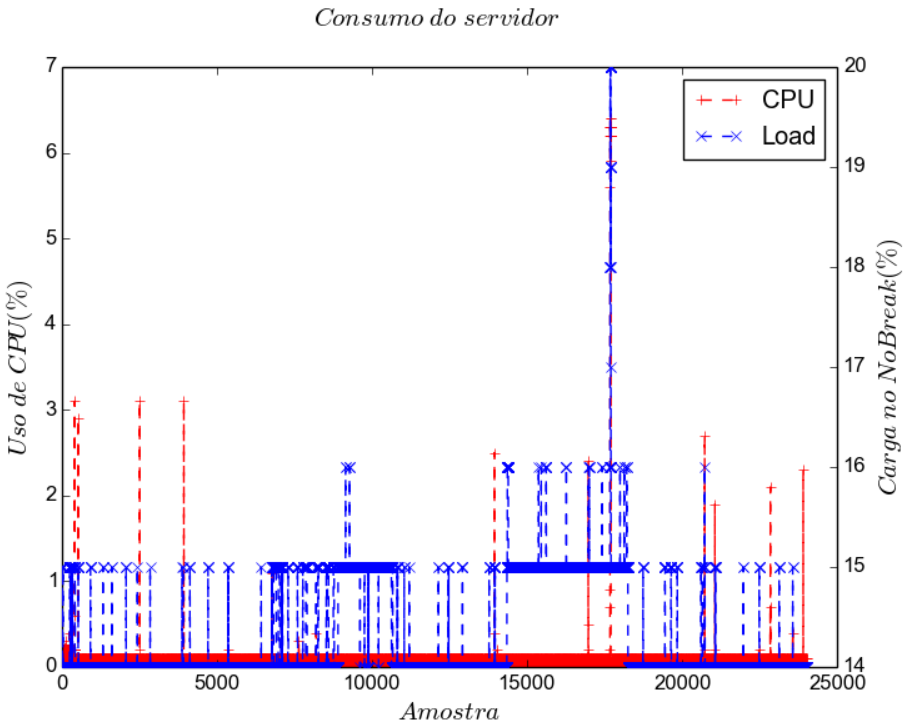


Figura 5.25: Consumo do servidor quando ocioso

sobre o *nobreak* nunca fica abaixo dos 14%. No intervalo aferido o carga média dos processadores é de $0,030\%$ enquanto que a carga sobre o *nobreak* é de $14,263\%$ o que representa um consumo médio de $123,375$ $Watts$ por servidor.

A pequena diferença entre o consumo dos servidores nestes dois estados é uma das motivações para o projeto e desenvolvimento de aplicações elásticas. Em um cenário com aplicações tradicionais para *cluster* tradicional estes servidores ficam dedicados à aplicação. As aplicações elásticas por outro lado liberam estes recursos computacionais caso não necessitem deles. Estes recursos podem então ser provisionados para outras aplicações, desligados ou colocados em um modo de baixo consumo, se este estiver disponível no *hardware*. As opções de aumentar a eficiência energética do ambiente computacional hibernando ou mesmo desligando parte dos servidores de um *cluster* impactam diretamente na resiliência do ambiente de IaaS que executa sobre esses servidores, devido às diferentes latências para que o servidor retorne ao estado ativo.

5.6 Discussão dos Resultados

Neste capítulo foram apresentados resultados que avaliam como diferentes técnicas de virtualização apresentadas na Seção 3.2 afetam o desempenho de um servidor Web para transmissão de vídeo. Os resultados obtidos através dos experimentos mostram

que em termos de utilização efetiva do *hardware*, em relação à banda passante de rede em Mbps ou número de clientes atendidos pela mesma máquina física, apenas o LXC consegue ter um desempenho comparável com o Linux sem virtualização quando é utilizada apenas uma máquina virtual. Contudo, um resultado interessante é o fato do KVM ter conseguido utilizar melhor o *hardware* quando se executam mais máquinas virtuais, chegando ao mesmo desempenho de rede do Linux sem virtualização a partir de quatro máquinas virtuais, isso é consequência dos processos em nível de usuário, descritos na Seção 3.2.2, que conseguem assim fazer uso efetivo da largura de banda disponível.

É possível observar com base nos resultados apresentados nas Seções 5.2.3 e 5.2.4, que mesmo em um cenário onde apenas um servidor físico é utilizado, o uso de uma aplicação de transmissão de vídeo permite diferenciar as técnicas de virtualização utilizadas. Esses resultados mostram ainda que apesar do aumento no número de máquinas virtuais sobre o mesmo nó computacional, o ambiente com KVM consegue utilizar 100% da largura de rede disponível, o que não ocorre para o Xen. Este resultado corrobora inicialmente a hipótese levantada na Seção 4.1.1 de que um serviço de vídeo consegue diferenciar, por exemplo, um ambiente computacional de IaaS usando Xen de outro que utiliza KVM.

Os resultados apresentados na Seção 5.3 mostram que a aplicação ElasticMovie consegue aferir quantitativamente as diferenças entre os ambientes de IaaS avaliados. Também, os experimentos também mostram que a utilização de valores elevados no parâmetro Fator de Paciência reduz a eficácia do ElasticMovie como *benchmark* para ambientes de IaaS. Por fim, a Seção 5.5 quantifica o custo energético do ambiente experimental e discute brevemente como a elasticidade das aplicações pode ser um caminho para aumentar a eficiência energética do ambiente computacional.

Capítulo 6

Relógio Global de Alta Precisão

Neste capítulo é abordado inicialmente o problema para temporização estritamente crescente e precisa para *cluster* computadores. Em particular, uma solução previamente proposta é adaptada para essas arquiteturas que garante a propriedade ECP, e sobre esse novo relógio de sistema é apresentada a solução para construção e manutenção de relógio global, o Relógio Global de Alta Precisão (RGAP). O RGAP difere dos mecanismos em *software* existentes para a manutenção de relógio global por não utilizar etapas de ressincronização.

6.1 Relógio Virtual Estritamente Crescente

O principal objetivo no design do Relógio Virtual Estritamente Crescente (RVEC) [112, 113] é evitar desvios de tempo de um sistema de computacional, tornando-o aderente à propriedade Estritamente Crescente e Preciso (ECP), sem incorrer em ruído adicional no sistema. O uso do *Time Stamp Counter* (TSC) como referência para o contador de tempo ECP atende estas demandas, dado que ele funciona sem o uso de interrupções, é interno ao núcleo de processamento, opera na frequência do núcleo e pela alta estabilidade do oscilador [2] [114] [115]. A escolha do TSC faz com que as únicas duas fontes potenciais de desvios de tempo sejam: gerenciamento de execução sobre os múltiplos núcleos e alterações na frequência destes núcleos.

Código fonte 6.1: Estrutura de dados do RVEC

```
struct tb{
  u64 base_counter;
  u64 age_time_ns;}
```

A solução RVEC é baseada no conceito de construir um relógio virtual, de modo que sua estrutura armazene dois valores, o da última leitura do TSC realizada pela lógica de controle do RVEC e o valor da passagem de tempo consolidado até o instante desta última operação de leitura. Assim, o RVEC é representando em um sistema computacional pela estrutura de dados *struct tb* apresentada no Código 6.1, onde o campo *base_counter* armazena o último valor lido do TSC e o campo *age_time_ns* armazena o tempo consolidado que é mantido pela lógica representada no Código 6.2, a ser explicada posteriormente neste capítulo.

Código fonte 6.2: Procedimento de atualização do RVEC

```
void update_rvec(struct tb *ptb, u64 CoreHZ){
  aux_tsc = get_counter();
  ptb->age_time_ns += (aux_tsc - ptb->base_counter)/CoreHZ;
  ptb->base_counter = aux_tsc;}
```

A Figura 6.1 ilustra a contagem de tempo utilizando o RVEC. No instante A, uma instância de RVEC é criada, armazenando o valor atual do TSC 10 no campo *base_counter* do RVEC e o valor 0 no campo *age_time_ns*. No instante B, a frequência do núcleo é alterada de 2 para 1, o que obriga a execução do Código 6.2 para atualizar os valores dos campos *base_counter* e *age_time_ns* para 20 e $5*10^9$ (5 segundos), respectivamente. Sendo importante observar que a instância do RVEC não pode ser lida durante o intervalo B-B' pois o fluxo de execução no sistema com relação a esta instância do RVEC está executando o procedimento descrito no Código 6.2.

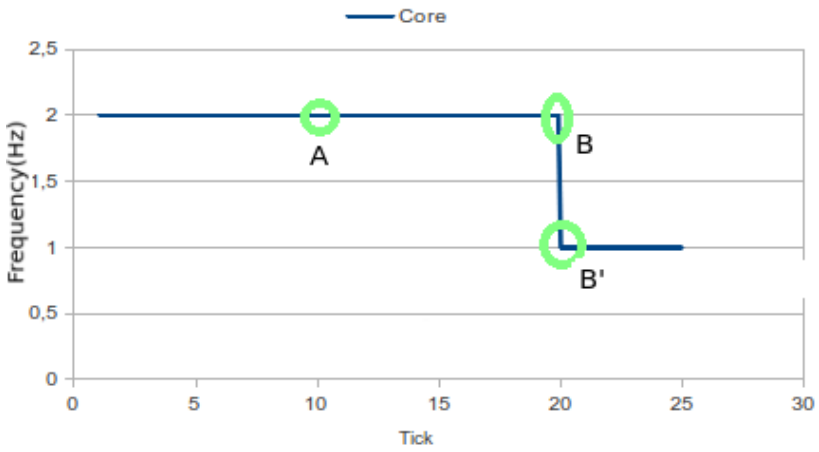


Figura 6.1: Contagem de tempo utilizando o RVEC: Inicialização (tick A) and atualização (tick B)

Portanto, RVEC permite a criação de uma abstração de contagem de tempo específica para o processo, desde o instante da sua criação. Especificamente, os pontos de atualização são os instantes em que ou o núcleo tem sua frequência alterada

ou quando um processo migra para outro núcleo. É extremamente importante que esses pontos de atualização sejam inseridos nos lugares apropriados de um sistema operacional para que, durante a sua execução, todas as instâncias de RVEC estejam protegidas de possíveis desvios de tempo.

6.1.1 RVEC no Linux

No Linux, RVEC é integrado com os subsistemas de gerenciamento do núcleo e de escalonamento de tarefas do *kernel*. Para esta finalidade, a criação de RVEC para os múltiplos núcleos de processamento é integrada ao procedimento de inicialização do Linux. Além disso, essa integração permite ao sistema oferecer simultaneamente suporte RVEC para *threads* tanto no nível de *kernel* como no nível de usuário. Portanto, o mecanismo resultante não impõe limites com relação ao número de instâncias de RVEC que podem ser criadas no sistema.

Código fonte 6.3: Procedimento de leitura do RVEC para um núcleo pré-selecionado

```
return u64 rvec_core_gettime(void){
  aux_tsc = (get_counter() - ptb->base_counter);
  return ptb->age_time_ns + aux_tsc/get_CoreHZ();}
```

A integração inicial do RVEC dentro do *kernel* é feita através da inserção da estrutura de dados *struct tb* na fila de execução de cada núcleo de processamento, representada no Linux pela estrutura de dados *struct struct rq*. Deste modo, o RVEC vinculado a um núcleo de processamento deve ter sua estrutura de dados atualizada sempre que um núcleo tem a sua frequência de operação alterada, assim garantindo a corretude da instância do RVEC. Esta garantia é dada através da atualização dos campos *base_counter* e *age_time_ns* realizada pelo Código 6.2, implementado no *driver Dynamic Voltage and Frequency Scaling* (DVFS) do Linux (CPUFreq). A estrutura do RVEC também mantém o tempo consolidado desde a iniciação de um núcleo de processamento, o qual é calculado a partir dos valores das leituras TSC através da passagem do tempo, a frequência atual da unidade do núcleo, e os campos RVEC que estão associados com o núcleo.

O RVEC também é instanciado para todas as tarefas (*threads*) dentro de um sistema Linux, realizado através da integração de *struct tb* com a estrutura de dados *struct task_struct* do *kernel* do Linux. Especificamente, após a inicialização de uma nova tarefa a executar em um núcleo específico, o valor corrente de RVEC associado com este núcleo irá ser copiado e armazenado no campo *base_counter* do RVEC que está associado com a nova tarefa. Assim, uma atualização de *base_counter* deve ser realizada sempre que a tarefa sofre uma migração entre núcleos de processamento, garantindo assim que o RVEC mantenha a propriedade ECP. Mais importante ainda,

este controle de atualização é executado pelo escalonador de tarefas do Linux, de modo que o RVEC está protegido de uma potencial fonte de desvios de tempo.

Entretanto, observe que a operação de atualização causada por um evento de migração gera uma chamada a um núcleo remoto, o que em sistemas Linux sobrecarregados pode fazer com que o sistema pare de funcionar (*Kernel PANIC*). Esta falha ocorre devido ao aumento na quantidade de chamadas a núcleos remotos que faz com que a temporização do escalonador de tarefas seja violada. Como forma de evitar esta situação irrecuperável, o controle de migração do RVEC posterga a atualização dos campos *base_counter* e *age_time_ns*, como descrito em [113].

Esta implementação RVEC permite que diferentes *threads* no sistema possam verificar os seus RVECs através da chamada de sistema *clock_gettime*(), que vai diretamente para o subsistema temporização (*timekeeping*) do Linux. Neste caso, uma aplicação chama esta função com o identificador do relógio $CLOCK\_RVEC$ como o parâmetro de entrada do relógio desejado.

6.2 RGAP para *Beowulf Clusters*

Aplicações paralelas e outras que executam em diversos nós computacionais quando necessitam realizar medições temporais dependem da existência de um relógio global visível a todos os nós. Por exemplo, nos ambientes computacionais de alto desempenho que utilizam o MPI [116] o acesso ao relógio global é feito através da chamada MPI_Wtime(), contudo o padrão MPI deixa para o projetista do ambiente computacional a responsabilidade de manter o relógio global. Como o padrão MPI espera que exista um mecanismo para a manutenção de relógio global em operação no ambiente computacional, este deve estar em operação mesmo que uma dada aplicação paralela não utilize o relógio global. Contudo, para as aplicações paralelas, estes mecanismos podem causar um impacto significativo na escalabilidade destas aplicações [6]. Uma das fontes para esse problema são as etapas de ressincronização existentes nas soluções atuais [117] e *"tick"* periódico do temporizador utilizado para manter a temporização do sistema computacional [52]. Deste modo o projeto de uma solução em software que consiga reduzir ou mesmo eliminar essas etapas de ressincronização oferecerá melhorias significativas para os ambientes de computação de alto desempenho atuais.

O uso RVEC permite garantir que a diferença de tempo entre leituras simultâneas de duas instâncias de RVEC distintas tendem a ser constantes, como descrito na Equação 6.1. Na Equação 6.1 são representadas duas tarefas A e B, onde $RVEC_n(A)$ e $RVEC_n(B)$ são os valores retornados pela leitura do RVEC local destas tarefas no mesmo instante n e, K uma constante que reflete a diferença entre os instantes

de criação das duas tarefas.

$$\forall_{n\in\mathbb{N}\|A\neq B}(|RVEC_n(A) - RVEC_n(B)|) \rightarrow K \tag{6.1}$$

Partindo desta premissa é proposto nesta tese o RGAP para *cluster* de computadores, que é baseado em um modelo cliente-servidor onde cada nó cliente sincroniza uma única vez com o nó escolhido como servidor RGAP no *cluster*. Especificamente, utilizou-se uma sincronização de cliente-servidor algoritmo probabilístico como o de [118] para determinar o tempo global, com relação a sincronização feita pelo RGAP, de um nó através da Equação 6.2.

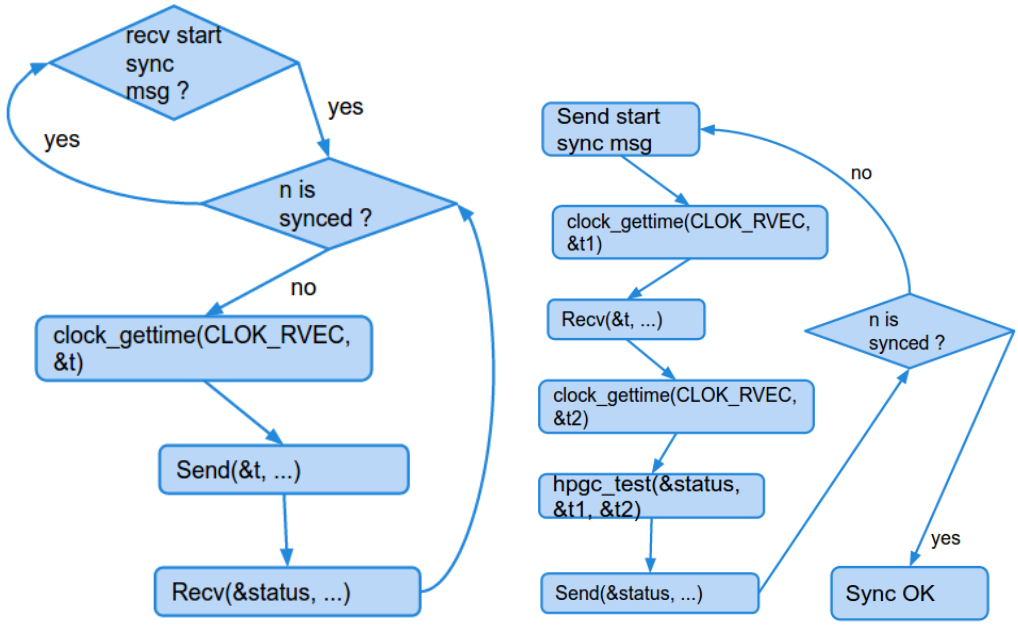
$$RVEC_{Server}(j) \approx (RVEC_{Client_i}(j) + RTT/2), (j = 1, m(i)) \tag{6.2}$$

Na Equação 6.2, $RVEC_{Server}$ é o RVEC do nó servidor RGAP e $RVEC_{Client_i}$ é o RVEC do nó cliente i; j é a medida de RVEC j-ésimo que é transmitido a partir do cliente i para o servidor RGAP. O *round-trip time* RTT mensagem é estimado com base na Equação 6.3 e assumindo que x, y são os RTTs mínimos entre um cliente i e o servidor RGAP; então por hipótese, RTT_x e RTT_y tenderão para um valor constante K_i depois de $m(i)$ mensagens terem sido trocadas com o servidor RGAP.

$$\forall_{x<>y\in\mathbb{Z}_+}(|RTT_x - RTT_y|) \rightarrow K_i \tag{6.3}$$

O algoritmo de sincronização RGAP para um nó cliente é apresentando na Figura 6.2, nela é possível observar os passos executados pelo servidor RGAP e por um cliente que deseja sincronizar seu relógio. O servidor RGAP como pode ser visto na Figura 6.2a fica esperando a chegada de uma mensagem solicitando a execução do procedimento de sincronização de um cliente. O cliente RGAP como visto na Figura 6.2b que deseja sincronizar globalmente seu relógio envia uma mensagem de sincronização para o servidor RGAP. Uma vez a tendo recebido a mensagem de início de sincronização o servidor RGAP realiza uma leitura do seu RVEC e envia uma mensagem informando o valor lido ao cliente. O cliente RGAP calcula então o tempo que a resposta demorou para ser recebida, o *Round-Trip Time* (RTT), e utiliza essa informação como entrada para o critério de parada. As mensagens do procedimento de sincronização são trocadas entre o cliente e o servidor RGAP até o desvio padrão RTTs alcançar um valor inferior ao valor pré-definido. Neste ponto o nó cliente é considerado sincronizado, assim ele armazena em sua estrutura RGAP o RTT mínimo, o valor RVEC servidor, e o valor do seu RVEC. Deste ponto em diante, o cliente pode calcular o valor de tempo global localmente usando a função descrita no Código 6.4.

O algoritmo básico descrito anteriormente permite sincronizar pares de nós den-



(a) Fluxograma do procedimento de sincronização do servidor RGAP

(b) Fluxograma do procedimento de sincronização do cliente RGAP

Figura 6.2: Fluxograma do procedimento de sincronização usando o RGAP

tro de um *cluster*. O RGAP funciona como mecanismo para manutenção de relógio global ao sincronizar todos os nós do *cluster* que necessitam de um relógio global com um único nó servidor RGAP. Assim o relógio global, representando pelo termo $RVEC_{Server}$ na Equação 6.2, pode ser localmente computado pelos nós clientes RGAP. No caso de múltiplos nós clientes, o servidor RGAP irá processar as solicitações de sincronização em ordem crescente do tempo chegada do pedido.

Código fonte 6.4: Função get_time do RGAP

```
u64 get_global_time(...){
  time_now = get_time(...) - gC.initTimeLocal + gC.minimalRTT/2;
  time_now = time_now + gC.initTimeRemote;
  return time_now;}
```

Assumindo que o RVEC consiga garantir a propriedade ECP do RVEC, os relógios RVECs locais não desviam ao longo do tempo, o que permite inferir que o RGAP também é aderente à propriedade ECP e logo o Código 6.4 não sofre desvios ao longo do tempo. Ainda com base na hipótese de aderência à propriedade ECP do RVEC e por consequência a do RGAP é possível inferir que a mesma leve a um relógio global no *cluster* livre de operações de ressincronização.

6.3 OpenMPI+ : OpenMPI com suporte de RGAP

OpenMPI+ é a versão da biblioteca para computação paralela OpenMPI básica estendida com um serviço de sincronização de tempo baseado no RGAP para a função MPI_Wtime(), observe que como informado em [119] o padrão *Message Passing Interface* (MPI) [116] não define um ponto fixo no passado para o início do tempo global. Atualmente, esta função requer que todos os relógios locais dos nós que executam uma aplicação MPI estejam externamente globalmente sincronizados, seja por meio um hardware dedicado (por exemplo, uma rede de sincronização) ou uma solução de software, como NTP que é normalmente utilizado em *clusters Beowulf*.

O fato de RGAP ser livre de ressincronização e portanto evitar a introdução de ruídos extras de sistema causados por *daemons* pode ser explorado de várias maneiras. Por exemplo, ele pode fornecer *timestamps* globalmente corretos para as mensagens e beneficiar os procedimentos de análise de traços e avaliação de desempenho com base no tempo mais preciso das aplicações paralelas. No OpenMPI+ a função original MPI_Wtime() é modificada. Especificamente, em vez de utilizar gettimeofday() em OpenMPI+, MPI_Wtime() chama a função get_global_time() do RGAP implementada dentro da biblioteca, que fornece uma referência de tempo global para a computação paralela.

Código fonte 6.5: Estrutura de dados do RGAP em OpenMPI+

```
struct rgap_t{
  u64 localBaseTime;
  u64 remoteBaseTime;
  u64 minimalRTT;}
```

Isto é realizado com a introdução da estrutura de dados rgap_t no OpenMPI. A estrutura de dados é apresentada no Código 6.5, onde localBaseTime é o valor RVEC local no instante da sincronização, remoteBaseTime é o valor RVEC remoto, tal como indicado pelo nó de referência, e minimalRTT é o round time trip mais curto calculado durante a fase sincronização do processo. A estrutura rgap_t é então adicionada na estrutura de dados ompi_communicator_t e o algoritmo de sincronização dentro da função ompi_mpi_init(), através da função OpenMPIplus_Boot().

O Código 6.6 mostra o algoritmo de sincronização RGAP implementando para o OpenMPI, a função **OpenMPIplus_Boot()**. Diferentemente da implementação original do RGAP, apresentando na seção anterior, para o OpenMPI o processo de referência do relógio global (servidor RGAP) é o processo com ***rank* 0**, sendo ele que

Código fonte 6.6: Procedimento de inicialização do RGAP para o OpenMPI

```
int OpenMPIplus_Boot(...){
  struct rgap_t rgap;

  if (0 == ompi_comm_rank(MPI_COMM_WORLD))
    for(i=1;i < ompi_comm_size(MPI_COMM_WORLD);i++){
        while(rgapOUT(client_status, ...))
          clock_gettime(CLOCK_RVEC, &time_now);
          time_ns = TimeToInt(&time_now)
          MPI_Send(&time_ns, ...);
          MPI_Recv(&client_status, ...);}
  else
    while(rgapOUT(client_status, ...)){
      clock_gettime(CLOCK_RVEC, &tpre);
      MPI_Recv (&time_ns, ...);
      clock_gettime(CLOCK_RVEC, &tpos);
      client_status = rgap_test(tpre, tpos, &rgap, &minRTTs);
      MPI_Send(&client_status, ...);}
  return OMPI_SUCCESS;}
```

inicia a sincronização com os outros nós que fazem parte de uma mesma computação paralela. Observe que a função OpenMPIplus_Boot() possui duas funções auxiliares: a função rgapOUT() é responsável por decidir com base na variável client_status se deve finalizar a execução do laço de sincronização para um processo; e a função rgap_test() que com base nos valores **tpre** e **tpos** calculam o valor corrente do RTT e decide se o nó já se encontra sincronizado, armazenando o resultado correspondente na variável client_status.

O procedimento mostrado no Código 6.6 deve ser executado para todos os nós de uma computação paralela, descartando as primeiras **D** mensagens de cada nó uma vez que essas mensagens geralmente têm valores mais elevados de RTT e rgap_test() só leva em conta os 20 menores valores de **RTT** para calcular o desvio padrão utilizado como critério de parada. O Código 6.6 não foi projetado para tratar as mensagens enviadas através MPI_Bcast() por esta forma de envio de mensagem não ser utilizado durante esta etapa do procedimento de inicialização da biblioteca OpenMPI.

6.4 Considerações Finais

Este capítulo detalhou o projeto e construção do novo mecanismo para criação e manutenção de relógios globais em *cluster*s de computadores energeticamente eficiente. Inicialmente o capítulo apresenta uma breve revisão do RVEC e como este é integrado ao sistema operacional Linux, e da propriedade ECP oferecida por ele. É a propriedade ECP que viabiliza o projeto do RGAP como um mecanismo para construção de relógios globais em *cluster*s de computadores *software* livre de operações

de ressincronização, operações essas que existem em outras soluções baseadas em *software* como é apresentado na Seção 2.2. Por fim, este capítulo detalha como o RGAP foi integrado à biblioteca MPI OpenMPI como mecanismo para fornecer o relógio global exigido para o correto funcionamento da mesma.

Capítulo 7

Avaliação Experimental do Relógio Global de Alta Precisão

Este capítulo apresenta os resultados de uma avaliação experimental do Relógio Global de Alta Precisão (RGAP): primeiramente são apresentados os dados da avaliação do Relógio Virtual Estritamente Crescente (RVEC) no *hardware* utilizado para os experimentos, em especial a propriedade Estritamente Crescente e Preciso (ECP), corroborando estes resultados com uma placa *Global Positioning System* (GPS), fundamental para o correto funcionamento do RGAP. Tendo demonstrado experimentalmente a confiabilidade do RVEC são apresentados os resultados da avaliação do algoritmo básico do RGAP descrito na Seção 6.2, para em seguida realizar a avaliação experimental da implementação do RGAP dentro da biblioteca OpenMPI, como descrito na Seção 6.3.

7.1 Ambiente experimental do RGAP

O ambiente experimental utilizado é composto por um *cluster* homogêneo de computadores com quatro nós como pode ser observado na Figura 7.1 e cujas características estão descritas na Tabela 7.1 (com o tipo do componente na primeira coluna e o modelo utilizado na segunda). O circuito de *hardware* que mantém os relógios de sistema durante os experimentos é o *High Performance Event Timer* (HPET).

Tabela 7.1: Nó computacional Dual Xeon Quad (Harpertown)

Componente	Modelo
kernel	Linux 3.1.10
Processador	Intel Pentium Xeon $E5410$ $2,33$ GHz
Duração Teórica do Ciclo	$0,43$ ns
Cache L2	12 MB
FSB	1333 MHz
Memória DRAM	$6 * 4$ GB DDR3 ECC
Interface de Rede	$2 * 10/100/1000$ Mbps
Unidade de Disco	1 TB SATA 7200 RPM
Placa GPS	TSync-PCIe-001 (Acurácia de 50 ns) [120]

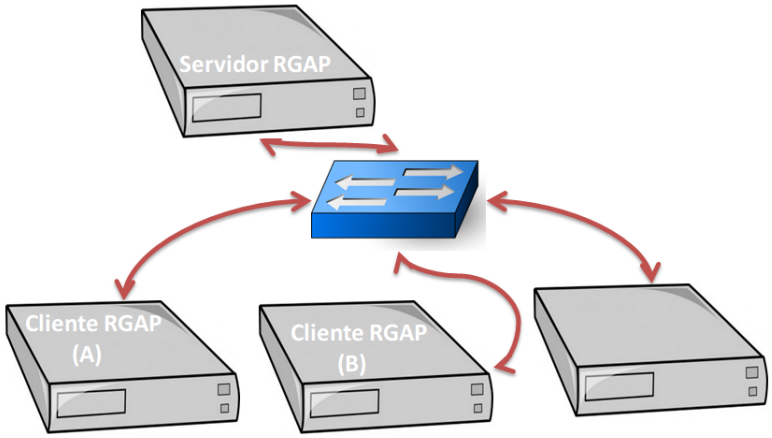


Figura 7.1: Ambiente experimental utilizado na avaliação do RGAP

7.2 Avaliação do RVEC no ambiente experimental

O correto funcionamento do RGAP depende da garantia de aderência do seu relógio base, o RVEC, à propriedade Estritamente Crescente e Preciso (ECP), quando executado no *hardware* utilizado nos experimentos, o processador Intel Xeon E5410. Assim os resultados apresentados nesta seção primeiramente avaliam a aderência à propriedade ECP do circuito de hardware básico (TSC) e da implementação do RVEC sobre esse circuito. Em seguida são aferidos os custos relativos ao uso do RVEC como mecanismo de temporização.

7.2.1 Aderência à propriedade ECP

Os resultados apresentados a seguir avaliam a aderência à propriedade ECP do *Time Stamp Counter* (TSC), do RVEC e **RVEC+Mig** (RVEC com migração) no

hardware descrito anteriormente. Os outros relógios do sistema foram excluídos deste experimento por eles dependem de ressincronização com uma fonte de relógio global externa para corrigir seus desvios de tempo. Todos os resultados apresentados a seguir são os valores médios e o intervalo de confiança é de $99,9\%$.

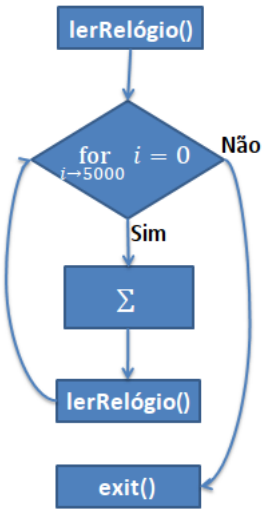


Figura 7.2: *Microbenchmark* utilizado para avaliar a aderência à propriedade ECP

O experimento é realizado através de um *microbenchmark* visto na Figura 7.2. Ele é composto por um laço **for** com um bloco simples de álgebra matricial (15 operações de adições por iteração) executadas 5.000 vezes em cada iteração. O *microbenchmark* é executado 100 vezes para cada um dos relógios.

TSC

A avaliação da propriedade ECP para o TSC é necessária para garantir que o programa de teste fique restrito a um núcleo e que o sistema não esteja utilizando *Dynamic Voltage and Frequency Scaling* (DVFS). Nestas condições o tempo médio de execução é de $93,649$ ms. A Figura 7.3 traça a evolução do tempo médio e desvio padrão coletados pelo *microbenchmark* para o TSC. É importante salientar que em nenhum momento ocorreu violação da propriedade ECP, tanto para as execuções individuais, como para os tempos médios consolidados, como é possível observar mais claramente no intervalo em zoom na Figura 7.3. Por ser uma avaliação experimental estes resultados não excluem à possibilidade de violações a propriedade ECP ocorrer em outras condições diferentes das avaliadas, contudo estes resultados fornecem indícios de que o mesmo é pouco provável.

O resultado anterior confirma que no experimento descrito o TSC é estritamente crescente. Para aumentar a confiabilidade deste experimento, em especial com relação à precisão do TSC para longos intervalos de tempo é necessário avaliar o comportamento do circuito base em comparação com uma fonte externa precisa.

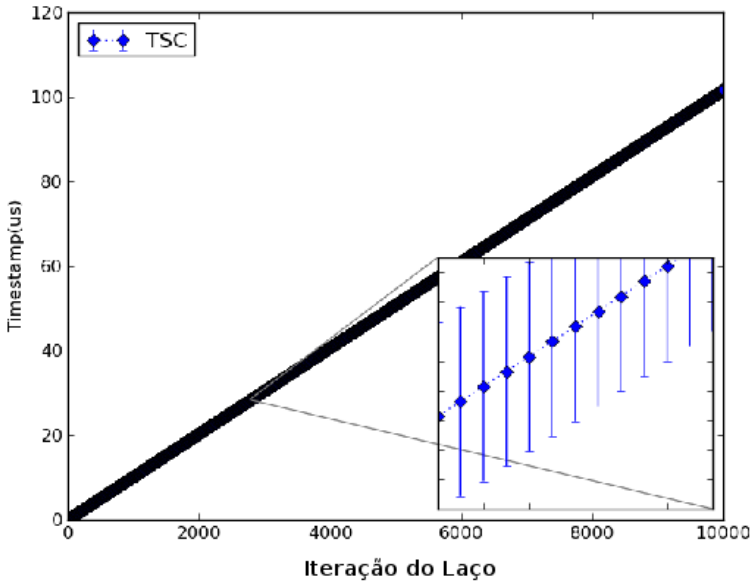


Figura 7.3: Avaliando à propriedade ECP do TSC

A seguir são apresentados os resultados do experimento realizado com uma placa GPS listada na primeira seção deste capítulo. Os experimentos são realizados com uma versão do *microbenchmark* descrito anteriormente, executando neste experimento 10K instruções de soma, sendo calculado *a posteriori* qual a duração de cada iteração do laço. Este experimento é repetido 644 vezes para cada um dos relógios, o que representa uma duração total de $15,17$ horas para TSC e $15,85$ para o GPS.

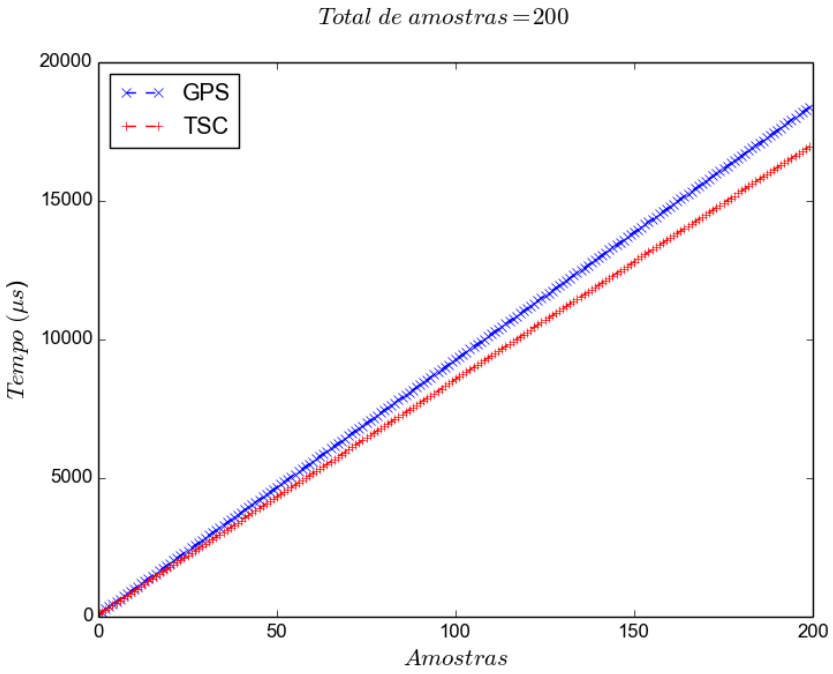


Figura 7.4: Amostra da progressão da execução no tempo para o laço de 10K instruções: GPS x TSC

A Figura 7.4 apresenta uma das comparações entre a progressão da execução ao longo do tempo do TSC e do GPS para as primeiras 200 iterações do laço. Nela, é possível observar que apesar de ambos os relógios se manterem estritamente crescentes, o valor aferido pelo GPS tende a divergir do valor aferido pelo TSC, corroborando deste modo a diferença no tempo total de execução obtido para o experimento. A duração média do laço utilizando o TSC é de $86,163\ \mu s$, com um desvio padrão de $1,460\ \mu s$, já o experimento utilizando o GPS obteve uma média de $92,668\ \mu s$ com um desvio padrão de $1,038\ \mu s$. A diferença entre as curvas do TSC e GPS vista na Figura 7.4 é decorrente do custo de acesso à placa GPS que é realizado através do barramento PCIe, que impõe em média uma sobrecarga de $6,505\ \mu s$, como mostra a Figura 7.5, sendo ela referente ao mesmo intervalo que o mostrado na Figura 7.4.

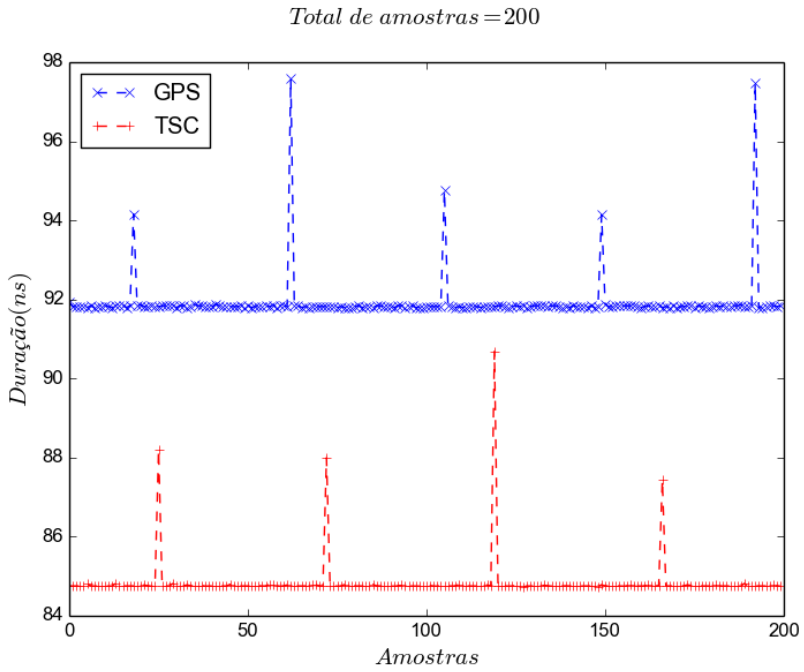


Figura 7.5: Amostra da duração do laço de 10K instruções: GPS x TSC

A Figura 7.6 apresenta o histograma para o TSC dos resultados obtidos para o experimento, onde dos 615.998.768 valores gerados, $98,53\%$ deles estão agrupados na primeira barra do histograma. Em comparação, a segunda e terceira barra do histograma concentram juntas apenas $1,46\%$ dos resultados obtidos com a execução do *microbenchmark*.

A Figura 7.7 apresenta o histograma para o GPS deste experimento. Dos 615.998.768 pontos gerados $98,264\%$ deles estão agrupados na primeira barra do histograma, com as duas barras seguintes concentrando juntas apenas $1,717\%$ dos resultados obtidos.

A Figura 7.8, apresenta uma execução do *microbenchmark* usado anteriormente

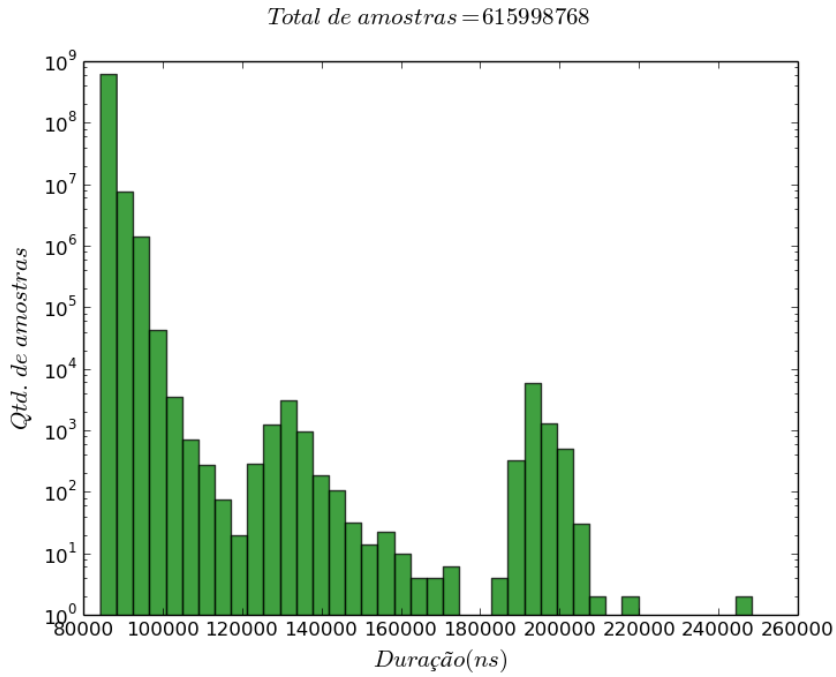


Figura 7.6: Histograma da duração do laço de 10K instruções para o TSC

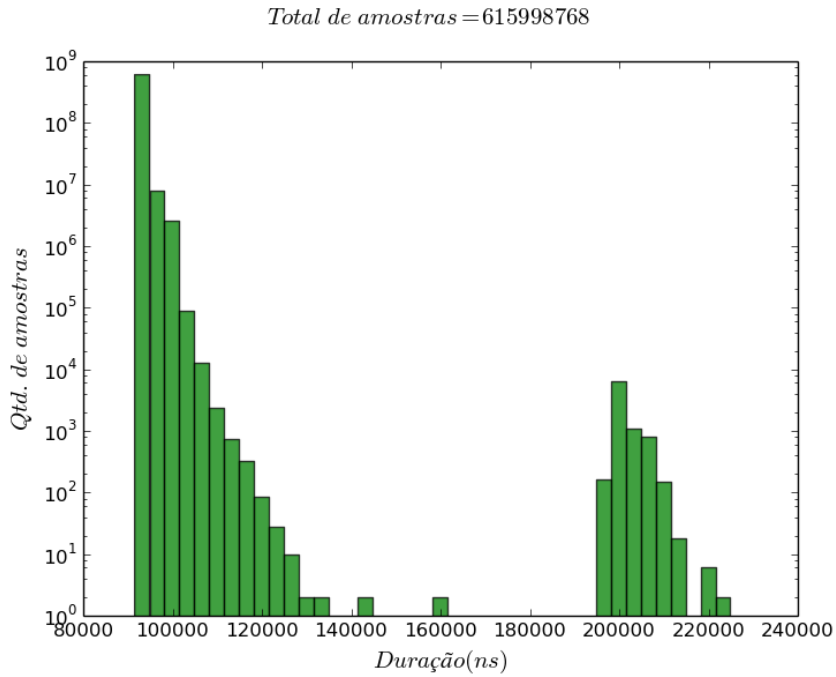


Figura 7.7: Histograma da duração do laço de 10K instruções para o GPS

agora utilizando um laço com 100K instruções de soma. A duração média de uma iteração do laço usando o TSC é de $829,454\ \mu s$, com um desvio padrão de $4,749\ \mu s$. Utilizando o GPS a duração média foi de $835,997\ \mu s$, com um desvio padrão de $20,366\ \mu s$, o que representa uma diferença média de $6,543\ \mu s$ na duração dos laços.

Por fim, a Figura 7.9 apresenta uma execução do mesmo *microbenchmark*, agora

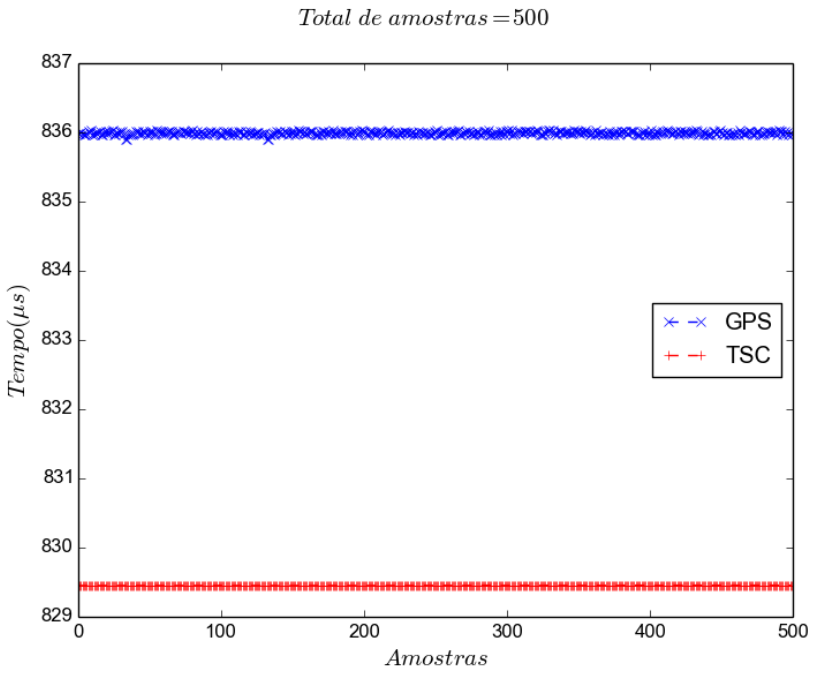


Figura 7.8: Duração do laço de 100K instruções: GPS x TSC

utilizando um laço com 1M instruções de soma. A duração média de uma iteração do laço usando o TSC é de $7.855,891\ \mu s$, com um desvio padrão de $77,646\ \mu s$. Utilizando o GPS a duração média foi de $7.862,533\ \mu s$, com um desvio padrão de $31,523\ \mu s$, o que representa uma diferença média de $6,642\ \mu s$ na duração dos laços. Neste experimento o aumento dos desvios padrão para o TSC e o GPS decorrem do maior número de interrupções no processamento das iterações do laço.

Com base na observação dos resultados apresentados é possível concluir que o TSC é um contador de tempo estável durante intervalos longos. Esta conclusão surge da observação dos resultados de estabilidade realizados com um laço de 10K instruções de soma e dos resultados que mostram que a diferença aferida entre o TSC e o GPS não cresce com o aumento do laço aferido para 100K e 1M instruções.

RVEC

Tendo em vista que o TSC se comporta como esperado no hardware utilizado para os experimentos, deve-se validar também se a implementação do **RVEC** obedece a propriedade ECP. A validação do RVEC para este ambiente experimental será dividida em duas etapas. Inicialmente ela é feita limitando a execução do programa de teste a apenas um núcleo e sem DVFS, como forma de limitar o ruído introduzido no sistema computacional.

O tempo médio de execução aferido pelo RVEC é de $99,834\ ms$. A Figura 7.10 mostra a evolução do tempo médio e do desvio padrão coletados pelo *microbenchmark*, tais como indicados pelo RVEC. O *zoom* apresentando na Figura 7.10 mostra

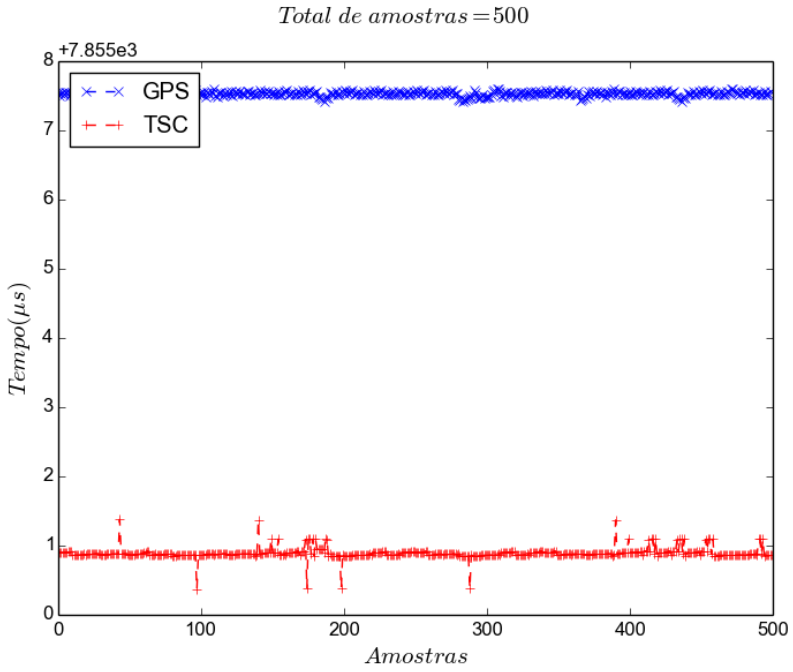


Figura 7.9: Duração do laço de 1M instruções: GPS x TSC

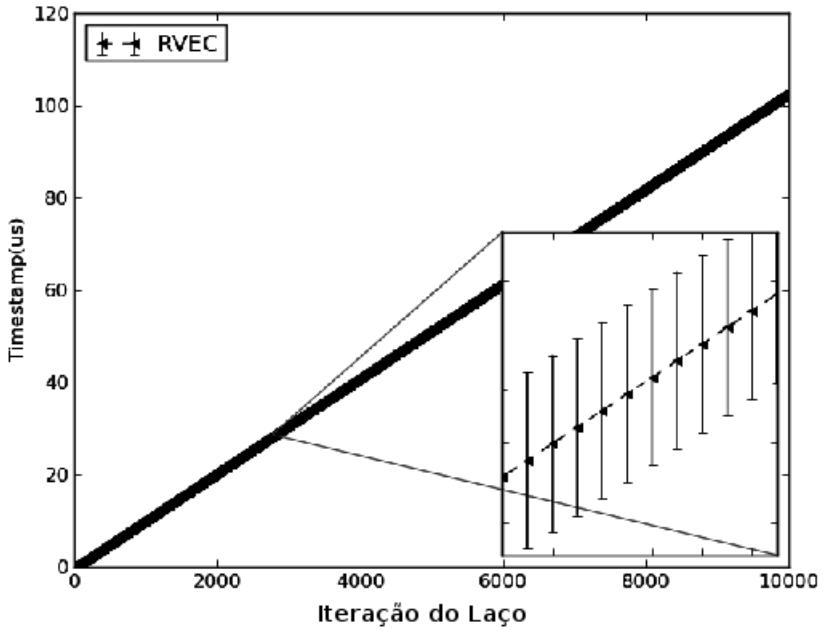


Figura 7.10: Avaliando a propriedade ECP do RVEC

que não ocorrem violações da propriedade ECP durante os experimentos.

A avaliação da propriedade ECP do RVEC, quando da ocorrência de migração de processos (RVEC+Mig) é realizada limitando a execução do *microbenchmark* a apenas um núcleo e alterando este núcleo durante a execução do programa de teste. O tempo médio de execução do *microbenchmark* é de $98,225\ ms$ quando o processo

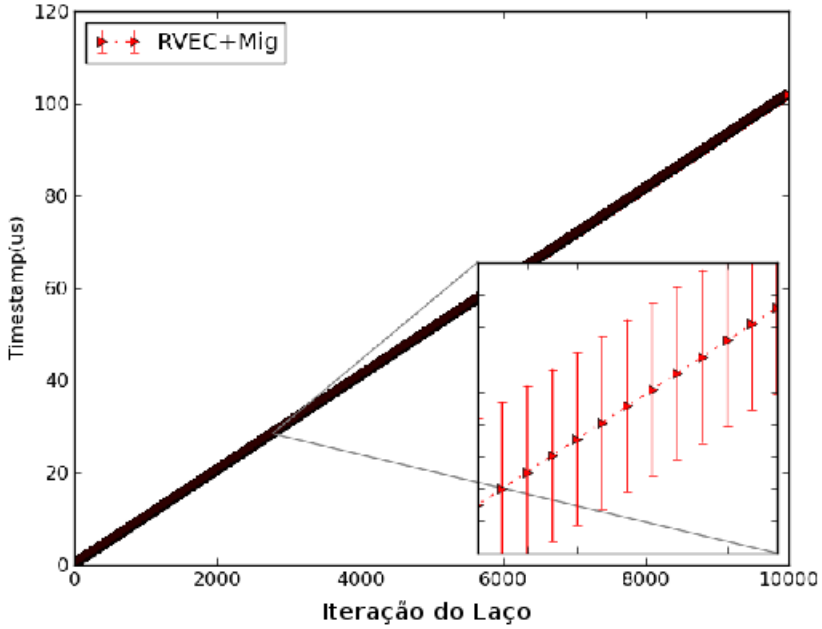


Figura 7.11: Avaliando a propriedade ECP do RVEC quando na presença de migração

é migrado. A Figura 7.11 apresenta a evolução do tempo médio e do desvio padrão coletados pelo *microbenchmark*, tais como indicados pelo RVEC para o experimento do RVEC na presença de migração de processos (RVEC+Mig). Portanto, o resultado mostrado na figura e mais claramente no zoom do intervalo de tempo da Figura 7.11 confirma que RVEC também obedeceu à propriedade ECP nesse experimento sob a condição de migração de processos. Interessantemente, o tempo médio de execução RVEC sem migração é maior do que com a migração. A razão é que, neste último caso, o Linux acaba por migrar o processo de teste para um núcleo menos sobrecarregado, com o objetivo de equilíbrio de carga.

7.2.2 Quantificando o custo de acesso ao RVEC

O incremento no tempo de execução observado no experimento anterior é explicado pela sobrecarga devido ao uso do RVEC em comparação com a TSC (instruções adicionais necessárias para acessar as informações dentro do Linux), de maneira análoga ao que ocorreu com a comparação entre o TSC e o GPS. Deste modo, quantificar o custo computacional de consultar o relógio de sistema RVEC e a precisão garantida são passos fundamentais para uma correta implementação do RGAP.

Para quantificar o custo desta sobrecarga sobre uma aplicação que utiliza o relógio de sistema RVEC, a chamada de sistema **clock_gettime** é introduzida dentro de um bloco contendo 5.000 instruções de soma, com e sem chamadas a um relógio do sistema após 2.500 instruções, ou seja, no meio do laço. O laço é exe-

cutado 10.000 vezes, e em cada iteração, são avaliados o TSC, o relógio do sistema **MONOTONIC** e o relógio do sistema RVEC, os dois últimos através da chamada de sistema *clock_gettime*(). Os resultados a seguir são o tempo médio e o desvio padrão, ambos os valores descritos na Tabela 7.2 estão em nanossegundos, para 100 repetições do experimento descrito anteriormente.

Tabela 7.2: Tempo de execução x Relógio

Clock option	µ(ns)	$\sigma(ns)$
TSC	257	$2,47$
MONOTONIC	575	$12,08$
RVEC	420	$3,12$

O tempo médio de execução do programa de teste sem ler os relógios do sistema é de $10,063$ μs com um desvio padrão de $0,762$ μs. A Tabela 7.2 apresenta um aumento no tempo médio de execução de 257 ns, o que representa uma sobrecarga de $2,554\%$ causada pelo uso do TSC. O relógio do sistema MONOTONIC aumentou o tempo médio de execução em 575 ns ou $5,714\%$ e o RVEC incrementou o tempo médio de execução em $4,174\%$. O incremento da consulta ao **RVEC** sobre a execução base representa um acréscimo de 420 ns para cada consulta feita ao RVEC por uma aplicação que o utilize como relógio de sistema. Os obtidos mostram que o RVEC apresenta uma sobrecarga menor quando comparado ao do relógio do sistema MONOTONIC, bem como uma variação no custo de execução mais próxima ao do TSC como visto na Tabela 7.2.

7.3 Manutenção de relógio global em *cluster*s usando RGAP

Esta seção apresenta os resultados da solução RGAP para sincronizar temporalmente os nós de um *cluster* de computadores rodando Linux sem a necessidade de ressincronização. O experimento utiliza um nó do *cluster* como servidor RGAP e dois nós clientes, o que permite verificar se as instâncias de RVEC em ambos os nós ficam sincronizadas ao longo do tempo. Em primeiro lugar, o servidor RGAP recebe uma solicitação de sincronização do nó cliente **A** e, em seguida, um pedido do nó cliente **B**, onde o algoritmo do cliente utilizado para o experimento é mostrado no Código 7.1. Essa implementação do cliente RGAP tem como objetivo avaliar como o procedimento de sincronização responde dentro do ambiente experimental, sendo assim o controle de saída escolhido foi limitar o número total de mensagens que podem ser trocadas durante uma tentativa de sincronização, e como mostrado no Código 7.1 o limite escolhido foi de 10.000 mensagens. A chamada de sistema **nanosleep()** (recebendo como parâmetro entrada 1 μs) é utilizada para

evitar inundação de mensagens de sincronização na rede, inundação essa que ocorreu durante execuções preliminares deste experimento quando essa chamada de sistema não foi utilizada.

Código fonte 7.1: RGAP – Algoritmo do cliente

```
void rgap_Client(...){
  for (i = 0; i < 10000; i++) {
    nanosleep(...);
    time1 = clock_gettime(clk_id);
    sendto(...);
    recvfrom(...);
    time2 = clock_gettime(clk_id);
    *(results+i) = (time2 - time1)/2;}}
```

O experimento é repetido 200 vezes e mede o número necessário de mensagens para que as variações do $RTT/2$ serem menores do que $K = 500\ ns$, que é então escolhido como critério de sincronização para o RGAP. O servidor RGAP inicia primeiro a sincronização do nó **A**, onde o valor de $RTT/2$ para o servidor é de $44,51\ \mu s$ para o **A** e de $44,47\ \mu s$ para o nó **B**. Em relação à inicialização da sincronização global, cada um dos dois nós, **A** e **B**, trocou um total de 481 mensagens com o servidor RGAP em menos de 1 ms utilizando um intervalo de 1 μs entre cada interação cliente-servidor e 3 mensagens por interação. Após a inicialização da sincronização com o RGAP, cada um de nós **A** e **B** usou sua instância do RGAP para enviar 2.000 mensagens *timestamp* para o servidor RGAP, enviando essas mensagens sempre após a execução de um laço contendo 5.000 instruções de soma, com cada experimento durando aproximadamente 22 ms.

A Figura 7.12 mostra os *timestamps* globais enviados por **A** e **B** em suas respectivas mensagens. É possível observar na figura que a enésima mensagem do nó **A** sempre é enviada primeiro que a enésima mensagem do nó **B**, uma vez que as mensagens enviadas por **A** sempre possuem um valor de *timestamp* global fornecido por RGAP menor do que as enviadas pelo nó **B**, o que confirma que o três nós do *cluster* estão globalmente sincronizados. Sendo importante salientar que o custo de 1 ms por nó sincronizado fornece indícios que o RGAP pode ser uma solução de relógio global altamente escalável.

7.3.1 Escalabilidade teórica da sincronização global usando RGAP

Com base no resultado anterior e assumindo um *cluster* cujos nós são interligados por uma rede não-bloqueante, o algoritmo de sincronização RGAP pode,

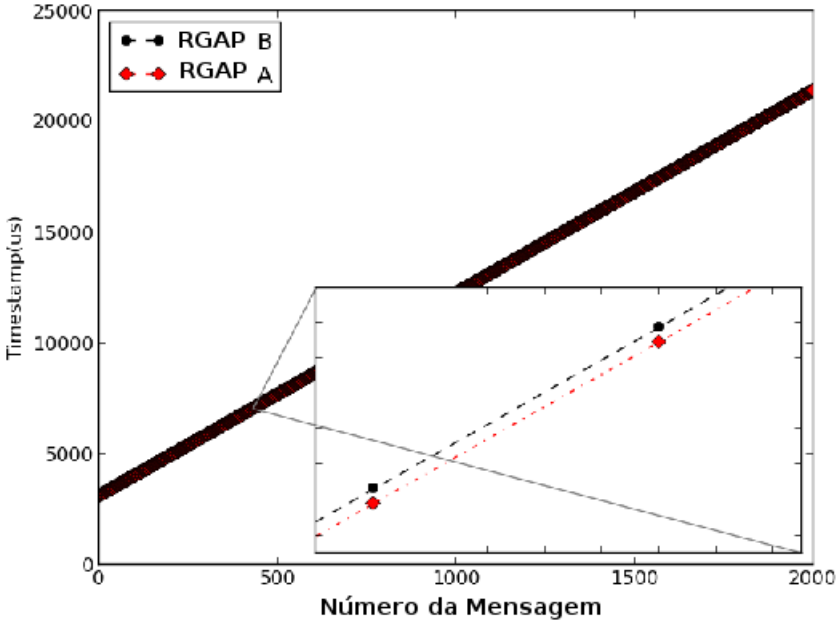


Figura 7.12: *Timestamps* globais de **A** e **B** usando RGAP

teoricamente, ser uma solução extremamente escalável para sincronização global. Assumindo que qualquer nó cliente uma vez sincronizado utilizando o RGAP pode se transformar em um servidor RGAP de outro grupo de nós clientes que ainda estão à espera de realizar a sua sincronização. A Equação 7.1 mostra que uma solução do algoritmo RGAP em dois níveis, poderia sincronizar mais de 500.000 nós por segundo, enquanto a Expressão 7.2 mostra que uma solução totalmente hierárquica utilizando RGAP poderia sincronizar 10^{301} nós por segundo, assumindo que o custo de sincronização permaneça 1 ms por nó, como aferido no primeiro experimento da Seção 7.3.

$$1000 + 999 + 998 + ... + 1 \Rightarrow \sum_{n=1}^{1000} n = 500,500 \tag{7.1}$$

$$\left.\begin{array}{l} t = 0 \Rightarrow 1 \text{ nó} \\ t = 1 \Rightarrow 2 \text{ nós} \\ t = 2 \Rightarrow 4 \text{ nós} \\ \vdots \\ t = 1000 \Rightarrow 2^{1000} \simeq 10^{301} \text{ nós} \end{array}\right\} \tag{7.2}$$

Na prática, um *cluster* de 1.000 nós pode ser sincronizado globalmente dentro de 1 segundo, bastando apenas cada um dos nós se sincronizar com o servidor RGAP.

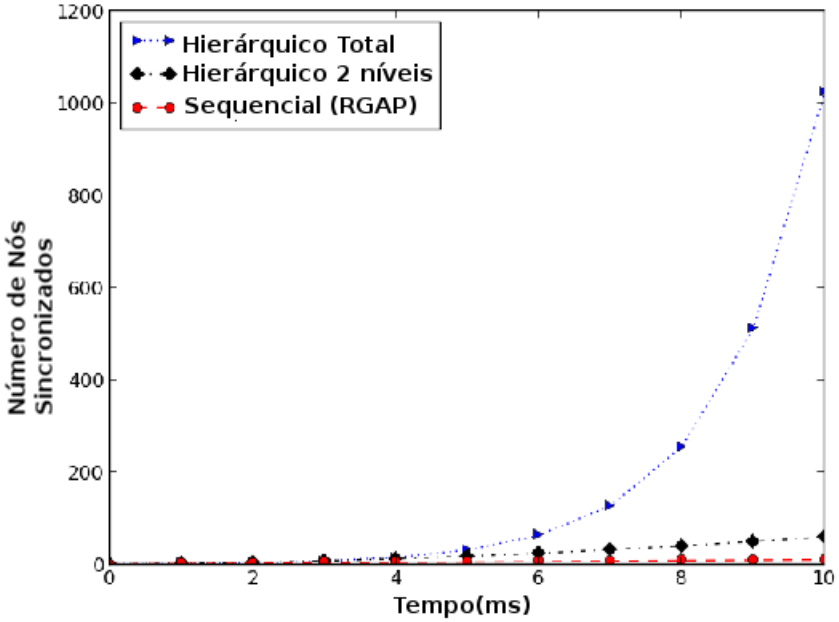


Figura 7.13: Escalabilidade teórica de três algoritmos baseados no RGAP

Em contraste, a solução *Network Time Protocol* (NTP) [1] oferece uma precisão muito mais baixa, assim como exige a ressincronização de todos os nós a cada 10 segundos, com o servidor NTP através da troca de quatro mensagens.

Outra vantagem significativa da ausência de ressincronização do RGAP é que este pode ser realizado para sincronizar nós *offline*, o que pode também beneficiar de sistemas embarcados com limitação de energia, tais como as redes de sensores sem fios. A Figura 7.13 ilustra a estimativa de escalabilidade para as três soluções utilizando **RGAP**.

7.4 OpenMPI+

A primeira avaliação para o **OpenMPI+**, descrito na Seção 6.3, busca identificar o número de mensagens que devem ser descartadas no início do ciclo de sincronização para cada nó do *cluster*. O experimento é executado 100 vezes, e utilizando os resultados anteriores da Seção 7.3 a função rgap_Client() é alterada para passar a trocar 500 mensagens com o servidor RGAP. A Figura 7.14 mostra os resultados de apenas 20 mensagens para um processo MPI remoto de um total de 8 processos utilizados para o experimento. Como pode ser visto na figura, só é necessário descartar, no máximo, as primeiras cinco mensagens, ou seja, $D = 5$.

Tendo configurado o número de mensagens a serem descartadas é possível fazer

[1] O apêndice A, apresenta uma avaliação do NTP utilizando o mesmo ambiente experimental que o descrito no inicio deste capítulo.

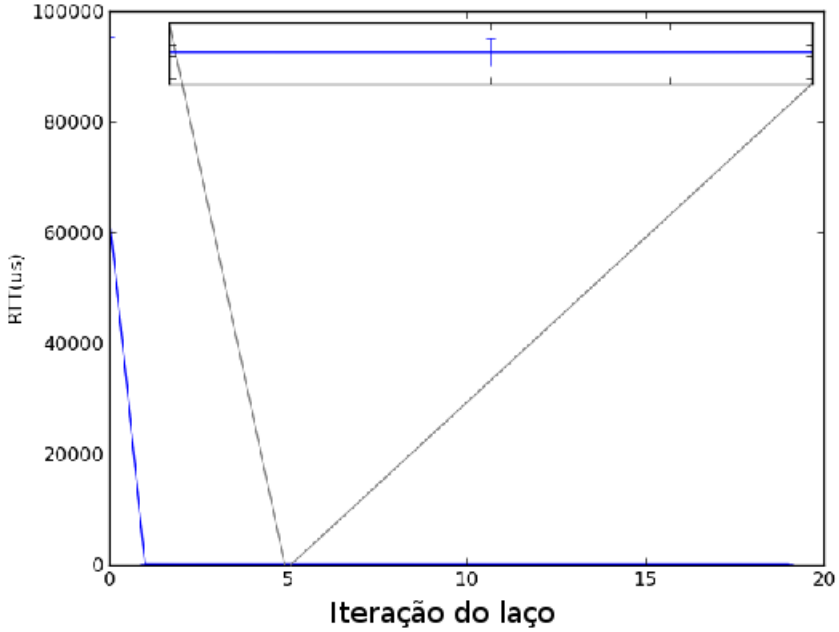


Figura 7.14: RGAP para OpenMPI

a avaliação experimental da aderência à propriedade ECP do **OpenMPI+**. Este experimento é feito através da geração de 100.000 *timestamps* e usa 4 nós, com 32 processos *Message Passing Interface* (MPI). O *microbenchmark* utilizado é similar ao utilizado na Seção 7.2.1 para os experimentos com o RVEC, contudo neste experimento todos os processos MPI remotos são previamente sincronizados com o processo de referência 0(*rank*). Ao final de cada iteração, cada processo remoto envia uma mensagem com o seu *timestamp* global para o nó de referência que armazena estas mensagens. O resultado é apresentado na Figura 7.15 e mais claramente no zoom dos dois intervalos na figura para três processos distintos, *rank* 8, *rank* 16 e *rank* 32. Os resultados mostram que o protótipo **OpenMPI+** obedece à propriedade ECP.

O tempo médio de sincronização por nó neste experimento é de $10,733\ \mu s$ e $2,952\ ms$ para uma computação com 8 e 32 processos respectivamente, com uma sobrecarga de 30 mensagens trocadas para a sincronização. Esse aumento na sobrecarga é consequência do impacto que a comunicação com o nó remoto tem sobre a computação paralela. Os valores são resultados de 100 execuções do *microbenchmark*, com **K** menor que 500 ns.

7.4.1 OpenMPI+: *Parallel Ocean Program*

Ferreira et al. [6] relatam o impacto negativo que os ruídos do sistema operacional provocam sobre o desempenho dos programas utilizados na computação de alto desempenho, consequência em grande parte da interferência de *daemons*, execução das

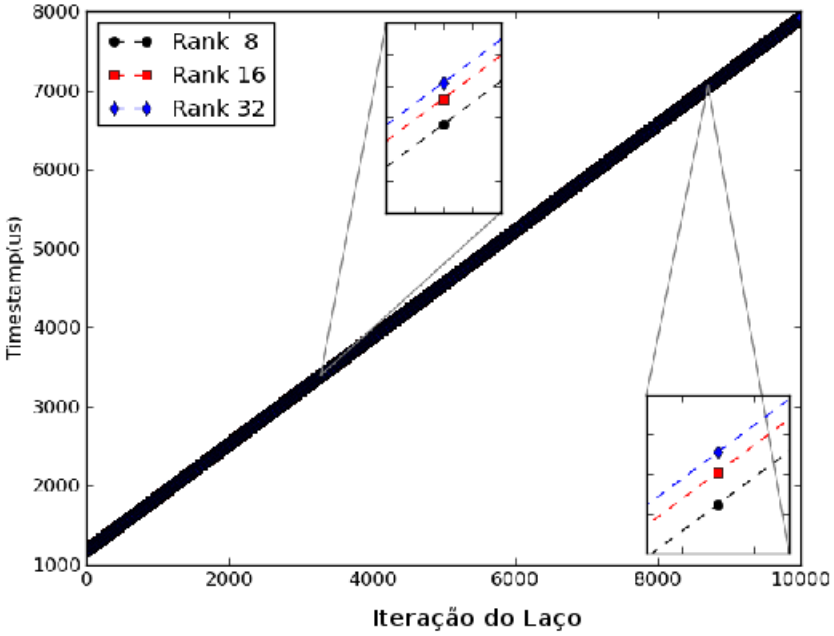


Figura 7.15: RGAP para o OpenMPI

threads do sistema operacional. Em particular, o *Parallel Ocean Program* (POP) é a aplicação mais impactada pela assinatura de ruído referente a um *daemon* de sistema. Os resultados a seguir apresentam a execução do POP utilizando os dois mecanismos de distintos para a manutenção do relógio global: RGAP e NTP. Como a solução RGAP proposta neste trabalho não necessita de mensagens de ressincronização ou *daemon* para seu correto funcionamento, ela deixa de introduzir ruídos adicionais no sistema computacional quando comparada com o NTP.

Tabela 7.3: Configuração do experimento com o POP

Parâmetro	Valor
stop_count	36000
solv_max_iters	10000
nprocs_clinic	4 - 48
nprocs_tropic	4 - 48
nx_global	12288
ny_global	8192

O ambiente experimental utilizado para avaliação preliminar do POP executando sobre **OpenMPI+** é formado por quatro nós, cada um com dois processadores hexa-core de 64 bits Intel Xeon E5-2420 ($1,90\ GHz$), Linux (Ubuntu 11.10 com *kernel* 3.1.10), interconectados por uma rede *gigabit Ethernet* e com o NTP sendo ressincronizando com um intervalo de 10 segundos. O POP é executado 10 vezes, variando o número de processos **MPI** de 4 à 48. A Tabela 7.3 apresenta os parâmetros de configuração do POP utilizados nos experimentos, observe que as primeiras quatro linha

são parâmetros de configuração estão localizados no arquivo **pop_in**. Os parâmetros **nprocs_clinic** e **nprocs_tropic** representam o número de processos MPI utilizados na computação, que neste experimento variou de 4 à 48 processos, já os parâmetros **nx_global** e **ny_global** definem o tamanho da grade simulada. A Figura 7.16 mostra que para todos os cenários o POP executando sobre o RGAP é mais rápido do que a execução utilizando o NTP.

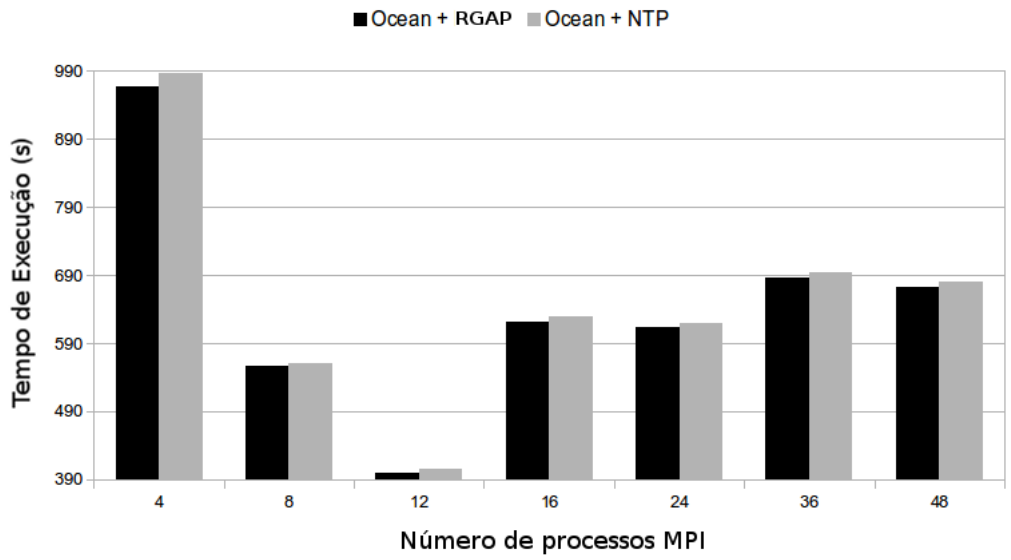


Figura 7.16: Tempo médio de execução do POP: RGAP (OpenMPI+) x NTP

A Figura 7.16 apresenta o tempo de execução de POP utilizando o mecanismo RGAP em comparação com a execução utilizando o NTP, os resultados mostram que o uso do **OpenMPI+** acelerou a computação entre 5 e 19 segundos quando comparado com o mesmo ambiente executando o NTP. No cenário com quatro processos, o POP termina após 966, 708 segundos utilizando o RGAP, contra 985, 880 segundos usando o NTP. O ruído introduzido pelo NTP aumenta em 19 segundos o tempo de execução de POP.

Ainda na Figura 7.16, o cenário com 8 processos o tempo de execução com o OpenMPI+ é 556, 786 segundos contra 561, 322 segundos do NTP, sendo que no cenário com 12 processos (uma máquina), o tempo de execução com o OpenMPI+ é de 399, 221 segundos contra 405, 458 segundos. No cenário experimental com 16 processos a execução com OpenMPI+ é 7, 721 segundos mais rápida do que com o NTP, tendo durado 621, 762 segundos. O cenário com 24 processos a execução sobre o OpenMPI+ é 7, 359 segundos mais rápida que a execução com o NTP. Sendo que a diferença do tempo de execução entre o OpenMPI+ e o NTP é de 6, 948 segundos para o cenário com 36 processos e para o cenário com 48 processos a execução com o OpenMPI+ é 7, 984 segundos mais rápida.

O efeito global dos ruídos introduzidos pelo NTP nos resultados obtidos para o tempo de execução do POP produz um desvio padrão $8,8$ vezes maior do que o encontrado nos cenários utilizando o RGAP. Enquanto que devido a utilização de apenas 4 nós nos experimentos preliminares limitam o impacto observado na aplicação POP, a redução no desvio padrão indica que a versão OpenMPI estendida com o RGAP introduz uma quantidade inferior de ruído na computação paralela quando comparada a execução no ambiente configurado com o NTP.

7.5 Discussão dos Resultados

Os resultados apresentados neste capítulo mostram que a solução para relógio global em *cluster* de computadores consegue manter os nós sincronizados sem a necessidade de ressincronização. Mais especificamente os resultados preliminares da avaliação da implementação do RGAP dentro da biblioteca OpenMPI mostram que o mesmo consegue reduzir o impacto causado pela manutenção do relógio global na execução do POP. Especificamente, mesmo usando um número pequeno de nós o tempo de execução foi reduzido em 2%. O algoritmo básico do RGAP consegue sincronizar um em menos de 1 ms, sendo que a versão atual implementada dentro da biblioteca OpenMPI leva $2,9$ ms.

Este capítulo também apresentou uma avaliação mais profunda da técnica RVEC, em especial as avaliações utilizando a placa GPS. Onde os resultados mostraram que mesmo ampliando em duas ordens de grandeza o numero de instruções, a medida de tempo realizada através do GPS condiz com o que é aferido utilizando o TSC, como pode ser visto nas Figuras 7.5, 7.8 e 7.9.

Capítulo 8

Trabalhos Relacionados

Este capítulo visa relacionar proposta desta tese com outros trabalhos relevantes encontrados na literatura, divididos em duas áreas: avaliação de ambientes de *Infrastructure as a Service* (IaaS) e mecanismos para manutenção de relógios global em *cluster* de computadores.

8.1 Avaliação nos ambientes de IaaS

Esta seção apresenta alguns dos trabalhos que avaliam sistematicamente os ambientes de IaaS. Essas avaliações são realizadas tanto no ambiente de IaaS da Amazon, o *Elastic Compute Cloud* (EC2), como em ambientes de IaaS locais construídos sobre um *cluster* usando pacotes como o **OpenStack** e o **OpenNebula**. Estas avaliações são motivadas pela disponibilidade de poder computacional de maneira instantânea e com baixo custo financeiro dos ambientes de IaaS, somada as experiências passadas de execução de aplicativos *cluster*s, levanta questões de como aplicações que exigem uma grande demanda computacional se comportam quando executadas em ambientes similares ao do EC2. A seguir, são apresentados alguns trabalhos que avaliam experimentalmente o impacto deste novo ambiente sobre esses serviços.

Walker [121], apresenta uma comparação experimental entre a execução da suíte *NAS Parallel Benchmarks* (NPB) no *cluster* **Abe** do *National Center for Supercomputing Applications* (NCSA) e um *cluster* construído com instâncias do EC2. Para os experimentos, são utilizadas instâncias do EC2 com capacidade computacional similar a dos nós do Abe, com 4 núcleos por processador e dois processadores por nó. A avaliação inicial dos nós computacionais usando uma versão OpenMP do NPB porém demonstrou uma degradação no desempenho que variou entre 7% e 21% das instâncias do EC2 em comparação com um nó do Abe. Os resultados obtidos com a execução dos aplicativos do NPB em ambos os *cluster*s, porém, mostram

que a degradação no *cluster* do EC2 variou entre 40% e 1000%. O autor investigou essa diferença comparando o desempenho do MPI em termos de largura de banda e latência para um *cluster* de 4 nós e 32 núcleos, os resultados obtidos mostram que com o aumento do tamanho da mensagem a largura de banda de bisseção no *cluster* do NCSA que utiliza Infiniband é uma ordem de grandeza maior que a do EC2, e a latência no Abe é uma ordem de grandeza menor.

Napper e Bientinesi [122], apresentam um estudo experimental executando o *benchmark* do Linpack [123] em *cluster*s de diferentes tamanhos no EC2. O número de nós em cada *cluster* variou de 2 a 16 e os *cluster*s foram construídos usando nós com grande capacidade de processamento compostos por 2 Intel Xeon quad-core de 2.3 GHz com 7 GB de memória principal e pelos maiores nós disponíveis das instâncias padrão do EC2 que continham 2 AMD Opteron dual-core de 2.60 GHz com 15 GB de memória principal. Os resultados obtidos pelos autores mostraram que apesar de individualmente um nó computacional (instância) do EC2 ser comparável a um nó de um *cluster* de um ambiente de HPC, o mesmo desempenho não consegue ser obtido por um *cluster* formado por instâncias do EC2. A execução do Linpack no EC2 apresentou severas perdas de desempenho quando se aumentou o número de nós do *cluster*, apesar de outros estudos mostrarem que o Linpack escala linearmente em ambientes de HPC [124]. Em termos de custo, os autores mostraram que ao contrário dos ambientes de HPC, no *cluster* utilizado no EC2 o desempenho obtido por dólar investido caiu exponencialmente com o aumento do número de nós.

Yigitbasi et al. [125], constroem um sistema de avaliação dos ambientes de IaaS como extensão do *benchmark* GrenchMark, que é um mecanismo para submissão de trabalho para ambientes de computação em grades. O C-Meter tem como objetivo permitir aos usuários aferir as sobrecargas ligadas à aquisição e liberação de máquinas virtuais. Para isso, os autores realizam uma comparação de diferentes configurações de instâncias virtuais e algoritmos de escalonamento. O sistema C-Meter é dividido em três subsistemas: o núcleo que interage com o GrenchMark, o módulo de interação com o ambiente de IaaS e o subsistema de utilitários que fornece ferramentas de configuração e análise dos experimentos. Com exceção dos experimentos onde são avaliadas as sobrecargas de aquisição e liberação de recursos computacionais usando SSH, os experimentos reportados no trabalho são todos realizados com um conjunto estático de máquinas. Com base nos resultados reportados, os autores concluem que o ambiente do EC2 é tanto horizontalmente escalável, ou seja, o aumento no número de instâncias representa também um aumento no desempenho, como verticalmente escalável, ou seja, a aquisição de recursos computacionais mais poderosos causa um aumento no desempenho medido pelo *benchmark*.

Mao e Humphrey [126] apresentam um estudo da latência de alocação de novas máquinas virtuais nos ambientes EC2, Azure e Rackspace. São avaliados fatores

como horário do dia, tamanho da imagem, tipo da instância e quantidade de pedidos simultâneos. Os resultados mostram que os principais fatores que impactam o tempo de alocação são o tipo da instância, tanto com relação o sistema operacional como a capacidade computacional e o tamanho da imagem. Diferentemente de [126], o método de avaliação proposto nesta tese busca aferir o impacto que a latência de alocação tem sobre uma aplicação elástica real.

Kossmann et al. [127], realizam um estudo de diferentes ambientes de computação na nuvem para aplicativos de banco de dados, usando o *benchmark* TPC-W. Como o objetivo dos autores era avaliar o desempenho de ambientes de computação na nuvem para aplicativos de processamento transacional, eles avaliam tanto soluções de IaaS como as de PaaS. Os cenários utilizados nos experimentos variam de soluções puras de IaaS como o EC2, o uso do EC2 com outros serviços de armazenamento (S3) ou banco de dados relacional na nuvem (RDS), a ambientes de PaaS como o Azure da Microsoft e o AppEng da Google. Para realizar os experimentos os autores reimplementaram o TPC-W, para que este pudesse utilizar as características de cada um dos ambientes, sendo que para o Azure é necessário reescrever o mesmo em outra linguagem. O *benchmark* escolhido reporta informações com relação a vazão total de consultas válidas por segundo e custo de consulta por cliente (desempenho). Para os cenários do EC2 e do Azure, as máquinas virtuais tinham aproximadamente a mesma capacidade computacional. O artigo reporta resultados para custo e escalabilidade, este último como volume total de requisições atendidas com sucesso, sendo que os únicos cenários que mostram escalabilidade são os do EC2 usando sistema de armazenamento S3 e o do Azure. Ainda, mostram como em um mesmo ambiente, no caso o EC2, diferenças nos serviços utilizados afetam o desempenho do aplicativo.

Sobel et al. [128], apresentam o Cloudstone, um *benchmark* para ambientes de IaaS. O Cloudstone é composto por um aplicativo Web 2.0, um conjunto de ferramentas para automatizar a geração de carga e medições de desempenho em diferentes cenários. O aplicativo Web 2.0 é o Olio e para este trabalho são utilizadas duas implementações do Olio, uma em Ruby e outra em PHP. Um ponto importante do referido trabalho é a recomendação por parte dos autores de se utilizar métricas baseadas em custo em dólares como parâmetro de avaliação. O artigo apresenta resultados de ambas às versões do Cloudstone para diferentes configurações no EC2; são executados seis cenários para Ruby e um para PHP, e com base neles é possível observar que os cenários com Ruby obtém um desempenho pior em média que os com PHP e que o desempenho do Ruby no EC2 é melhor com máquinas de maior capacidade de processamento. Contudo, os experimentos com o *benchmark* utilizado não permitem aos autores avaliar os efeitos da elasticidade na aplicação, uma vez que para o uso do Cloudstone o conjunto de máquinas deve ser estático.

Ueda e Nakatani [129] apresentam uma comparação experimental entre as pla-

taformas para construção de ambientes de IaaS Eucalyptus e OpenNebula, ambas usando o Xen como monitor de máquina virtual, realizando ainda uma comparação destes resultados com os obtidos no EC2, onde ainda para o OpenNebula são feitas avaliações alterando a forma como as imagens das máquinas virtuais são armazenadas. Eles constroem um *benchmark* que modela o serviço web Wikipédia utilizando ferramentas de código livre. Além disso, os autores realizam uma avaliação do tempo médio de provisionamento de uma instância virtual nos três ambientes. Os resultados obtidos pelos autores mostram, para as soluções de código livre que são avaliadas a fundo, como diferenças na construção do ambiente de IaaS impactam tanto o desempenho da aplicação quando o sistema esta em funcionamento (número de máquinas fixo), como quando se necessita provisionar novas máquinas virtuais, contudo não é realizado um estudo de como a latência de alocação impacta o serviço que executa sobre um ambiente de IaaS.

Ferdman et al. [130] propõem a suíte de *benchmark* **CloudSuite** composta por conjunto de *benchmark*s representativos da classe de aplicação *scale-out*, como MapReduce, *Web Search*, analise de dados e transmissão de vídeo. No artigo eles realizam um profundo estudo de como essas aplicações executam nos processadores existentes através de métricas como cache miss e número de instruções por ciclo. Apesar de o trabalho original não possuir foco diretamente em ambientes de IaaS, a suíte de *benchmark* proposta tem sido utilizada em outros trabalhos como parte da metodologia para avaliação de ambientes de infraestrutura como serviço como em [131] e [132]. A metodologia para avaliação de ambientes de IaaS proposta nesta tese também utiliza um serviço de transmissão de vídeo, contudo diferente do disponível no CloudSuite que não aloca novas máquinas virtuais dinamicamente, o ElasticMovie é projetado para ser um serviço elástico e com isso avaliar o comportamento da resiliência do ambiente de IaaS.

Gambi et. al. [133] atacam o problema do controle automático de elasticidade em ambientes de IaaS reais, ou seja, ambientes onde a latência de alocação não é nula. Neste trabalho os autores abordam o problema de como estimar esta latência e apresentam resultados do uso desta estimativa para o controle de elasticidade da aplicação Cassandra [134] nos ambientes OpenStack e Reservoir. Os resultados apresentados no trabalho mostram que o impacto no desempenho da aplicação que as operações de alocação e liberação de máquinas virtuais provocam. A escolha do Cassandra como aplicação de avaliação causa um efeito interessante, pois a mesma para temporariamente de responder as requisições quando se retira um nó do sistema, somente retornando a responder as requisições após sua etapa de reconfiguração. Apesar de a proposta dos autores não ser de um *benchmark* para ambientes de IaaS, a avaliação experimental deles afere os impactos das operações elásticas sobre a aplicação. Diferente do que ocorre com o Cassandra nos experimentos apresenta-

dos pelos autores, o novo *benchmark* ElasticMovie continua atendendo aos clientes durante a reconfiguração dos recursos disponíveis as aplicações elásticas.

Em [135], Dory et. al. apresentam uma avaliação experimental do comportamento dos aplicativos Cassandra, HBase e mongoDB, todos eles representantes da classe de base de dados NoSQL [136]. No trabalho é avaliado apenas o impacto provocado pela adição de novos recursos computacionais ao *cluster* que mantem essas aplicações, a comparação entre os resultados realizada pelos autores corrobora a ideia de que elasticidade da aplicação é dependente das escolhas técnicas feitas no projeto das mesmas. As duas principais criticas a este trabalho estão na ausência de uma avaliação do impacto da retirada de recursos do *cluster* e da ausência de estudos do impacto que a variação na latência de alocação de recursos causa na aplicação, em comparação os resultados apresentados nesta tese utilizando a proposta ElasticMovie avalia os impactos provocados pela latência de alocação e pela liberação de máquinas virtuais.

Li et. al. [137] apresentam CloudCmp uma ferramenta para comparar sistematicamente o desempenho e custo de provedores de nuvem. CloudCmp avalia o poder computacional, a latência de alocação, armazenamento persistente e serviços de rede oferecidos, por diferentes ambientes de computação em nuvem. Diferente dos trabalhos citados anteriormente, CloudCmp busca avaliar ambientes de tanto de IaaS quanto de PaaS, sendo assim os autores utilizaram como base da ferramenta de comparação uma versão alterada por eles da suíte de *benchmark* SPECjvm2008 [138]. A latência de alocação de recursos é avaliada através de um *microbenchmark* que solicita a alocação de 20 instâncias e calcula o intervalo de tempo entre a solicitação e a instância estar disponível ao usuário, no correlacionando a latência aos impactos causados na aplicação elástica. O uso do ElasticMovie como *benchmark* proposto nesta tese diferentemente da proposta CloudCmp, pois busca quantificar o impacto que a latência de alocação tem sobre uma aplicação elástica e nos usuários desta aplicação, deste modo ele não cobre ambientes de PaaS como a CloudCmp. Como o método de avaliação proposto nesta tese está limitado a ambientes de IaaS, a avaliação do desempenho computacional das máquinas virtuais é realizado com *benchmark*s tradicionais, como o SPECcpu [57] ou pelo componente da aplicação elástica que executa sobre as máquinas virtuais, que no caso do ElasticMovie é o servidor HTTP NGINX.

Barker et. al. [139] apresentam ShuttleDB, um mecanismo para automatizar a elasticidade de banco de dados na nuvem. Este mecanismo é baseado em técnicas de migração e replicação tanto no nível de máquinas virtuais como de banco de dados. A avaliação experimental é feita com uma versão modificada do YCSB [140] e os resultados apresentados mostram que o uso da solução hibrida chega a reduzir o tempo de alocação de novas máquinas virtuais em 87%. A técnica ShuttleDB

também permite a liberação de recursos através da remoção de réplicas, sendo isto experimentalmente demonstrado no trabalho. Assim como o ElasticMovie é possível utilizar o ShuttleDB para avaliar o impacto que a alocação ou liberação de máquinas virtuais tem sobre a aplicação elástica. Contudo como os autores não buscavam criar um *benchmark* para ambientes de IaaS, não avaliando como ShuttleDB é impactado pela variação no tempo de alocação das máquinas virtuais, sendo que, além disso, o ShuttleDB necessita ter acesso ao monitor de máquinas virtuais para seu correto funcionamento, o que não é factível em ambientes de IaaS compartilhados.

Varghese et. al. [141] propõem uma metodologia de *benchmarking* dividida em seis etapas em que um usuário fornece um conjunto de pesos que indicam a importância que a memória, o processador, de comunicação e armazenamento tem para o aplicativo que precisa ser executado no ambiente de IaaS. A avaliação destes componentes individuais é realizada através de *benchmark*s tradicionais como o SPECcpu [57] e o BONNIE++ [142]. Utilizando as informações fornecidas pelo usuário e pelos *benchmark*s, o método classifica as máquinas virtuais com o objetivo de localizar a máquina virtual que oferece o melhor desempenho para a aplicação. Diferente deste trabalho o método de avaliação para ambientes de IaaS proposto nesta tese busca aferir o impacto que a resiliência do ambiente tem sobre a elasticidade da aplicação e como esta é percebida pelos usuários.

Gillam et. al. [143] apresentam uma avaliação de desempenho dos ambientes de IaaS da IBM, EC2 e RackSpace, além de uma avaliação de uma instalação privada do OpenStack. São aferidas características dos nós virtuais como rede, CPU, disco e memória, utilizando um conjunto de *benchmark*s públicos. Entre os *benchmark*s utilizados estão o STREAM [144], LINPACK, BONNIE++ e MPPTEST [145]. Diferentemente do ElasticMovie, os *benchmark*s utilizados neste trabalho não avaliam o impacto do ambiente de IaaS na elasticidade de aplicação.

As Tabelas 8.1 e 8.2 apresentam um resumo dos trabalhos relacionados apresentados nesta seção, elas representam respectivamente os trabalhos que não possuem avaliação de elasticidade e os que possuem. A principal desvantagem dos trabalhos dispostos na Tabela 8.1 é o fato de nenhum deles buscou aferir o impacto que a alocação de novas máquinas virtuais causa nas aplicações utilizadas como *benchmark*s. Na Tabela 8.2, os trabalhos que possuem na coluna "Elástico"a entrada parcial são aqueles em que a avaliação foi feita com um *microbenchmark* que apenas aferiu o tempo de criação de uma máquina virtual, chamado ao longo do texto de latência de alocação. Ainda na Tabela 8.2, os últimos três trabalhos utilizam aplicações elásticas e fornecem resultados de como a latência de alocação das máquinas virtuais impacta no desempenho das aplicações escolhidas como *benchmark*s, sendo que a desvantagem deles no âmbito da avaliação de desempenho dos ambientes de IaaS foi não terem explorado como a variação da latência de alocação

impacta o *benchmark* escolhido.

Tabela 8.1: Artigos relacionados que não realizam avaliação da elasticidade

Paper	Tipo de Cloud	*Benchmark*	Objetivo
[121]	IaaS (EC2)	NPB	Avaliar desempenho do ambiente para HPC
[122]	IaaS (EC2)	LINPACK	Avaliar desempenho do ambiente para HPC
[125]	IaaS (EC2)	GrenchMark	Avaliar desempenho do ambiente para computação em grades
[127]	IaaS/PaaS	TCP-W	Avaliação de desempenho de servidor Web e de banco de dados
[128]	IaaS (EC2)	Cloudstone	Avaliação de desempenho com uma aplicação Web 2.0
[129]	IaaS	Wikipédia	*Benchmark* construído para modelar uma base de dados da Wikipédia
[130]	IaaS (EC2)	CloudSuite	Suíte de *benchmark*s para avaliar o desempenho dos processadores que suportam os ambientes de IaaS
[141]	IaaS	SPECcpu	Avaliação do desempenho de ambientes de IaaS

8.2 Trabalhos relacionados ao RGAP

Esta seção apresenta alguns trabalhos relacionados com a técnica de temporização global desenvolvida nesta tese, o **R**elógio **G**lobal de **A**lta **P**recisão (RGAP). Os trabalhos aqui descritos estão divididos em duas categorias: primeiro são apresentados os trabalhos que lidam com o projeto de mecanismos para a manutenção de relógios globais, onde se observa a aderência destas técnicas à propriedade ECP; posteriormente são discutidos os trabalhos que investigam o impacto que a manutenção dos relógios globais tem sobre a computação paralela.

8.2.1 Projeto de Relógio Global

Em sistemas de computacionais, a manutenção dos relógios do sistema geralmente é apoiada por um *daemon Network Time Protocol* (NTP) que ressincroniza periodicamente com um servidor remoto [3]. Sob uma pesada carga de trabalho, no entanto, a execução *daemon* NTP muitas vezes é retardada, o que pode causar desvios de

Tabela 8.2: Artigos relacionados que realizam algum tipo avaliação da elasticidade nos ambientes de IaaS

Paper	Tipo de Cloud	Elástico	*Benchmark*	Objetivo
[137]	IaaS	Parcial	SPECjvm2008	Avaliação com o SPECjvm2008 e aplicação sintética para avaliar latência de alocação
[126]	IaaS/PaaS	Parcial	Sintético	Avaliação da latência de alocação de máquinas
[143]	IaaS	Parcial	Diversos	Avaliação de ambientes de IaaS usando diversos *benchmark*, com avaliação sintética da latência de alocação de máquinas virtuais
[133]	IaaS	Sim	Cassandra	Estimativa da latência de alocação e projetar um mecanismo para com base nesta estimativa alterar a quantidade de máquinas virtuais alocadas a aplicação
[135]	IaaS	Sim	NoSQL	Avalia o impacto que a latência de alocação tem sobre banco de dados NoSQL
[139]	IaaS	Sim	MySQL	Automatização da elasticidade de bancos de dados

tempo de até dezenas de segundos [36] nos relógios do sistema. Além disso, esses relógios do sistema são aderentes à propriedade Estritamente Crescente e Preciso (ECP) quando leituras sucessivas ao relógio são realizadas em intervalos de tempo menores do que dezenas de milissegundos [36].

O trabalho de Tian et al. [146] apresenta o desenvolvimento de um mecanismo para manutenção de relógio global usando o *Time Stamp Counter* (TSC) como o contador base, juntamente com um algoritmo de sincronização remota que é semelhante ao do NTP. A diferença marcante entre este trabalho de sincronização de outras abordagens puramente em software é que ele não utiliza uma fonte de tempo externa. Devido à utilização direta do TSC, um relógio global pode não funcionar corretamente em processadores que utilizam *Dynamic Voltage and Frequency Scaling* (DVFS) ou múltiplos cores ou unidades de processamento. O protocolo de sincronização apresentado pelos autores, assim como o NTP, não pode garantir a aderência à propriedade ECP quando o intervalo entre as requisições ao relógio de sistema for pequeno. A propriedade é violada devido ao fato do nó ter que ajustar

seu relógio global baseado na troca de mensagens com o servidor, sendo o ajuste feito aplicando-se um Δ ao valor do TSC. Por motivos de precisão, o protocolo é projetado para executar dentro do *kernel*, contudo, isso não impossibilita que em nós muito carregados o fluxo de execução (*kernel thread*) possa não conseguir executar com a frequência necessária para manter os nós globalmente sincronizados.

Sobeih et al. [147] apresentam uma proposta de um protocolo para a manutenção de um relógio global. O protocolo por eles proposto funciona elegendo um líder que funciona como se fosse um nó estrato 1 no NTP, sendo que a maior diferença entre o procedimento de eleição de líder nesta proposta e outros algoritmos de eleição de líder em sistemas de sincronização é que no neste trabalho existe a possibilidade de existir estados transientes no sistema onde existem múltiplos líderes, um para cada partição da rede. O artigo apresenta os resultados das simulações utilizadas para avaliar o protocolo. Como este protocolo realiza trocas periódicas de mensagens entre seus pares, sua implementação sofre obrigatoriamente da mesma deficiência do NTP, descrita em [36–38], não garantindo deste modo aderência à propriedade ECP.

O RADClock [148] é um sistema de sincronização distribuída sem estado construído sobre relógios os sistema (por exemplo, *High Performance Event Timer* (HPET)) ou contadores de tempo (por exemplo, TSC), que mantem um relógio global relativo, bem como um relógio global absoluto para os nós de uma rede de sincronização. No entanto, o RADClock depende de um *daemon* para ressincronização periódica. Além disso, o uso direto de TSC limita sua aplicabilidade em processadores com múltiplos núcleos. No trabalho [149] os autores utilizam a solução RADClock como proposta de solução para manutenção dos relógios de sistemas das máquinas virtuais, com a vantagem de reduzir o número de violações de tempo quando comparado com o Clocksource Xen e NTP. Entretanto, esta solução mantem as limitações encontradas do trabalho original do RADClock com respeito aos processadores com múltiplos núcleos e suporte a DVFS.

Em [150], os autores propõem uma rede de sincronização em hardware mantida através de um gerador de pulso remoto. O *cluster* que utiliza o relógio Global possui uma rede secundária especial para sincronização, onde o *hardware* desta rede de sincronização recebe os tiques de um oscilador remoto (relógio). A rede de sincronização é organizada como uma árvore onde a raiz é o oscilador remoto, onde os cabos desta rede devem ter o mesmo comprimento [151]. O *hardware* proposto para suportar a rede de sincronização não necessita de interferência das camadas de *software* para realizar a atualização da informação do relógio local, logo as únicas funcionalidades que envolvem *software* são as operações de *reset* e leitura da instância local do relógio global (*time stamp*), garantindo deste modo a propriedade ECP aos relógios locais.

O *Precision Time Procol* (PTP) [152] é um padrão IEEE (IEEE 1588 e IEEE1588-2008 [40]) projetado para a manutenção de um relógio preciso para testes e medições em rede de computadores. O PTP é normalmente utilizado com suporte de *hardware* na interface física de rede e nos *switches*, contudo também existem implementações apenas em um *software* [153]. A principal desvantagem do PTP em relação ao RGAP é a necessidade de *hardware* dedicado para que este seja aderente à propriedade ECP.

Jones e Koenig [117] propõem um mecanismo para a manutenção de um relógio global para aplicações MPI, baseado no conceito de cliente-servidor. Um dos resultados do trabalho mede uma melhora na variação média de tempo para um conjunto de nós que passa de $20,0\ ms$ utilizando o padrão NTP para até $2,29\mu s$. O mecanismo proposto é baseado em uma fase de sincronização inicial entre o nó de referência e todos os outros nós de uma computação paralela, sendo que este mecanismo conta com etapas de ressincronização realizadas durante as chamadas coletivas do MPI. Em comparação, a solução RGAP só requer uma fase de sincronização inicial entre o do RVEC nó de referência e os RVECs dos outros nós, isso é consequência da propriedade ECP do RVEC.

As soluções em *software* garantem a propriedade de crescimento estritamente crescente do tempo nos casos onde o intervalo entre consultas ao relógio de sistema é grande o suficiente para esconder as violações da propriedade ECP do relógio de sistema. As soluções em *hardware* são imunes a este problema, contudo são menos flexíveis. A Tabela 8.3 sumariza esses trabalhos sendo possível observar através dela que as soluções existentes para a manutenção de relógio global aderente à propriedade ECP para qualquer intervalo de tempo entre medidas só é possível com a utilização de *hardware* dedicado. Nas soluções em *software*, a existência de etapas de ressincronização impende a aderência destas à propriedade ECP, diferentemente destes trabalho,s o Relógio Global de Alta Precisão (RGAP) é uma solução em *software* que não utiliza etapas de ressincronização e é aderente à propriedade ECP.

Tabela 8.3: Artigos que tratam da manutenção e relógio global

Paper	ECP	*Hardware*
[146]	Não	Não
[147]	Não	Não
[148]	Não	Não
[150]	Sim	Sim
[152]	Sim	Sim
[117]	Não	Não

8.2.2 Impacto dos mecanismos de manutenção de relógios de sistema no desempenho das aplicações paralelas

Ferreira et al. [6] aprestam um estudo sistemático dos efeitos dos ruídos do sistema operacional sobre o desempenho de algumas aplicações de alto desempenho em um sistema de computacional de grande escala. O estudo utiliza o sistema operacional Catamount, por este introduzir poucos ruídos na computação paralela. Por exemplo, o trabalho mostrou que a interferência, consequência de uma assinatura de ruído de $100Hz/25\mu s$, semelhante ao do perfil gerado pelo tratamento das interrupções de tempo quando medido em um sistema Linux carregado, pode causar 30% de redução no desempenho de uma aplicação em execução num sistema com dez milhares de nós. Especificamente, o trabalho também mostra que o *Parallel Ocean Program* (POP) sofre uma redução de 2000%, devido ao efeito global de uma assinatura ruído de alta duração e baixa frequência. Especificamente, a dramática desaceleração do POP é consequência de uma aplicação paralela que concorre com uma *thread* intermitente do *kernel* ou *daemon* sistema com frequência de execução de 10 Hz e duração de $2500\mu s$.

Em [52] Ferreira et al. avaliam como o equilíbrio do ambiente paralelo (relação entre computação e banda disponível), distribuição dentro da computação dos nós que sofrem interferência causada por ruídos de sistema e como o algoritmo que implementa as funções coletivas do MPI atenuam o impacto destes ruídos na computação paralela. Os autores mostram que mesmo com um percentual pequeno dos nós suscetíveis a fontes de ruídos o impacto na aplicação paralela é significativo. A variação na distribuição dos nós que sofrem interferência, se estes possuem um *rank* MPI próximo ao do *rank 0* mostra um impacto menor na computação paralela quando comparado ao mesmo percentual de nós sofrendo interferência quando estes possuem *ranks* mais distantes do zero. Um resultado importante deste trabalho é a analise do impacto que o balanceamento do ambiente paralelo tem sobre a aplicação. Os autores mostram que as aplicações que executam em ambientes bem balanceados sofrem um impacto maior dos ruídos do sistema operacional em comparação com ambientes paralelos onde existe mais capacidade computacional do que largura de banda de rede. O trabalho mostra que o impacto do ruído SO não é apenas uma propriedade do comportamento do padrão de comunicação de uma aplicação paralela, sendo influenciado por outras propriedades do ambiente e do sistema operacional.

Petrini, Kerbyson e Pakin [54] apresentam uma avaliação de desempenho do supercomputador ASCI Q. O trabalho identifica um grave problema de desempenho causado pelos ruídos do sistema operacional, usando como *benchmark* o SAGE. Por exemplo, eles identificam que um dos *heartbeats* utilizados no monitoramento dos nós é responsável por 75% da degradação no desempenho provocada pelos ruídos do

kernel. A redução nos ruídos do SO, consequência da retirada de parte dos *daemons* e redução na frequência dos *heartbeats*, causa uma melhoria de $2,21$ vezes, para 7.680 processadores, sobre a configuração original do ASCI Q.

Park et. al. [49] apresentam a proposta de sistema operacional FusedOS para os nós computacionais de um supercomputador que combina as abordagens de um *kernel* completo e de um *kernel* leve (este último mais comum). Neste trabalho os processadores são divididos em dois grupos: o primeiro executa o *kernel* do Linux e o segundo grupo o *kernel* leve, representado pela Compute Library (CL). O protótipo do sistema FusedOS avaliado em um sistema computacional Blue Gene/Q e cujos experimentos demonstraram que esta abordagem introduz uma baixa quantidade de ruído, compatível com os ruídos do sistema operacional CNK [154].

8.3 Considerações Finais

Neste capítulo foram apresentados alguns trabalhos relacionados encontrados ao longo da pesquisa bibliográfica feita nesta tese. Com relação à avaliação de desempenho para ambientes de IaaS, como é possível observar nos trabalhos apresentados na Seção 8.1, apesar de já existirem propostas de aplicativos para funcionar como *benchmark*s para estes ambientes computacionais, ainda não está definido um conjunto de aplicativos relevantes e que explorem as novas dimensões desses ambientes, como a elasticidade dos aplicativos e a resiliência do ambiente de IaaS. Com relação à avaliação de elasticidade é importante ressaltar os trabalhos de Dory et. al. [135], Gambi et. al. [133]. Já a Seção 8.2 apresenta alguns dos trabalhos relacionados com a proposta de relógio global em software desta tese, entre estes trabalhos dois trabalhos que estão mais próximos ao RGAP são Ferreira et al. [6] e Jones e Koenig [117].

Capítulo 9

Conclusão e Trabalhos Futuros

Neste capítulo são apresentadas as principais conclusões decorrentes desta tese, bem como trabalhos futuros identificados como relevantes.

9.1 Resumo

Está tese propôs e implementou duas novas ferramentas para ambientes computacionais energeticamente eficientes. Uma dessas ferramentas é um novo *benchmark* para ambientes de *Infrastructure as a Service* (IaaS) denominado ElasticMovie. Ele é um sistema de distribuição de *Video On Demand* (VoD) elástico usando o protocolo HTTP, que utilizando o conceito de Fator de Paciência consegue aferir quantitativamente a resiliência dos ambientes de IaaS. A outra ferramenta é um mecanismo para manutenção de relógios globais em *software* para *cluster*s de computadores, o **R**elógio **G**lobal de **A**lta **P**recisão. O RGAP é aderente à propriedade Estritamente Crescente e Preciso (ECP) e viabiliza a construção de relógios globais dispensando o uso das etapas de ressincronização, comuns às soluções em *software*, o que o torna atraente para ambientes computacionais de alto desempenho.

O ElasticMovie foi projetado para ser facilmente portado entre ambientes de IaaS, tendo sido portado e avaliado em ambientes mantidos por softwares de gerenciamento distintos, bastando para isso apenas configurar as chamadas de alocação e liberação de recursos computacionais. A avaliação experimental mostrou que o mesmo consegue estressar os ambientes de IaaS, quando o Fator de Paciência é próximo da latência de alocação de novos recursos. O protótipo inicial do RGAP inicial construído para avaliar a viabilidade da solução conseguiu sincronizar globalmente o relógio de um nó remoto por milissegundo, sendo assim o RGAP foi posteriormente portado para dentro da biblioteca OpenMPI. A implementação dentro do OpenMPI busca oferecer suporte à chamada MPI_Wtime(), sem que seja necessária a execução de *daemons* no sistema computacional ou de *hardware* dedicado, sendo

que esta versão do RGAP sincroniza um nó remoto em $2,9\ ms$, utilizando um *cluster* com 4 nós construído com interconexão *gigabit ethernet*.

9.2 Conclusões

Com base nos experimentos realizados, tanto a proposta ElasticMovie como o RGAP conseguem alcançar seus objetivos. Mais especificamente no caso do ElasticMovie é possível observar que a utilização deste, passando pelas avaliações preliminares descritas no Capítulo 5 viabiliza uma avaliação sistemática dos ambientes de IaaS, para as aplicações elásticas. Os resultados experimentais das Seções 5.2.3 e 5.2.4, onde é possível observar que o uso de diferentes técnicas de virtualização em combinação com a variação no número de máquinas virtuais apresentou uma utilização melhor da banda de rede disponível no servidor para o ***K**ernel-based **V**irtual **M**achine* (KVM) em comparação com o Xen. Os resultados da taxa de bloqueio correlacionados com o Fator de Paciência do usuário do sistema indicam que o sistema elástico de transmissão consegue capturar impactos que as diferenças dos ambientes de IaaS causam na aplicação de vídeo, escolhida nesta tese como *benchmark*.

O Relógio Global de Alta Precisão (RGAP), proposto e avaliado nesta tese, é baseado no relógio de sistema Relógio Virtual Estritamente Crescente (RVEC) para construir um mecanismo de manutenção de relógios globais livre de ressincronização para um *cluster* de computadores. Foi também avaliada uma implementação preliminar deste mecanismo dentro da biblioteca OpenMPI. A avaliação experimental de uma implementação do RVEC e RGAP no Linux usando um *cluster* de referência com quatro nós, mostrou que RVEC teve sobrecarga insignificante e foi altamente preciso em comparação com um relógio do sistema representativo Linux. Além disso, os resultados indicaram que a solução de sincronização RGAP pode ser altamente escalável, e potencialmente ele pode sincronizar até 1000 nós/s em um *cluster* como visto na Seção 7.3.1. Além disso, a aplicação preliminar do RGAP na biblioteca OpenMPI mostrou que a sincronização de um processo remoto leva apenas $2,9\ ms$, embora seja três vezes mais longo do que o protótipo para *cluster*.

Mais importante ainda, a avaliação do impacto da RGAP sobre a redução do ruído do sistema em um aplicativo *Parallel Ocean Program* (POP), usando um *cluster* quatro nós com 12 núcleos por processador mostrou uma redução perto 2% no tempo de execução. Estes resultados promissores sugerem que o RGAP é uma alternativa eficaz para a manutenção de relógio global. A proposta RGAP apresenta resultados promissores com relação a sua capacidade de manter o tempo sincronizado sem com isso introduzir ruídos no sistema computacional. Esses resultados, tendo em vista os trabalhos relacionados encontrados, fornecem fortes indícios de que a utilização do RGAP em ambientes paralelos pode reduzir os ruídos de sis-

temas ligados à manutenção dos relógios globais e com isto aumentar a eficiência energética destes ambientes ao permitir que os nós computacionais passem mais tempo realizando computação útil à aplicação paralela.

9.3 Trabalhos Futuros

Os trabalhos futuros para ElasticMovie se concentram em três linhas distintas: a primeira é dar sequencia nas avaliações para diferentes ambientes IaaS, tanto privadas como públicas; a segunda linha de trabalho trata de reorganizar o componente ElasticMovie *Front* para permitir que o *benchmark* possa ser utilizado na avaliação de grandes datacenters que implementem ambientes de IaaS; por ultimo existe a linha de alterar o componente de transmissão, que pode ser deste passar a oferecer suporte ao *streaming* dinâmico em HTTP ou soluções mais disruptivas. A avaliação de diferentes ambientes IaaS inclui a avaliação do mecanismo Neutron para o Openstack e como este impacta o desempenho da configuração de rede das máquinas virtuais, avaliar como outros mecanismos de contextualização para as máquinas virtuais, ou mesmo fazer avaliações em ambientes de IaaS totalmente diferentes como o EC2 ou o Eucalyptus. Com relação às melhorias no ElasticMovie, elas têm como alvo manter a elasticidade do mesmo, se possível ampliá-la, para viabilizar a avaliação de ambientes de IaaS emergentes que consigam entregar latências de alocação de recursos cada vez menores.

Para o RGAP existe a necessidade de experimentos em ambientes dedicados à computação de alto desempenho, o que pode fazer com que seja necessário reimplementar o RVEC em um novo *kernel*. Além de ser necessária uma investigação para identificar quais são os gargalos na atual implementação no OpenMPI, essa investigação passa por realizar avaliações do RGAP com o POP em ambientes computacionais paralelos maiores e com diferentes níveis de ruídos provocados pelos *daemons* de sistema. Outra linha de pesquisa importante dentro do escopo do RGAP é investigar a escalabilidade da solução em *cluster* de computadores maiores e, em especial, como o RGAP pode ser utilizado nos futuros ambientes *Exascale*.

Referências Bibliográficas

[1] INTEL. *IA-PC HPET (High Precision Event Timers) Specification*, Oct.. 2004.

[2] PÁSZTOR, A., VEITCH, D. "PC based precision timing without GPS", *SIGMETRICS Perform. Eval. Rev.*, v. 30, n. 1, pp. 1–10, jun. 2002. ISSN: 0163-5999. doi: 10.1145/511399.511336.

[3] MILLS, D. L. "Network Time Protocol (Version 3) Specification, Implementation and Analysis". 1992. Network Working group report RFC 1305.

[4] JONES, T., KOENIG, G. A. "A Clock Synchronization Strategy for Minimizing Clock Variance at Runtime in High-End Computing Environments". In: *SBAC-PAD'10*, pp. 207–214. IEEE CS, 2010. ISBN: 978-0-7695-4216-4. doi: 10.1109/SBAC-PAD.2010.33.

[5] MEISNER, D., GOLD, B. T., WENISCH, T. F. "The PowerNap Server Architecture", *ACM Trans. Comput. Syst.*, v. 29, n. 1, pp. 3:1–3:24, fev. 2011. ISSN: 0734-2071. doi: 10.1145/1925109.1925112.

[6] FERREIRA, K. B., BRIDGES, P., BRIGHTWELL, R. "Characterizing application sensitivity to OS interference using kernel-level noise injection". In: *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pp. 19:1–19:12, Piscataway, NJ, USA, 2008. IEEE Press. ISBN: 978-1-4244-2835-9. Disponível em: <`http://dl.acm.org/citation.cfm?id=1413370.1413390`>.

[7] KERBYSON, D. J., JONES, P. W. "A Performance Model of the Parallel Ocean Program", *Int. J. High Perform. Comput. Appl.*, v. 19, n. 3, pp. 261–276, ago. 2005. ISSN: 1094-3420. doi: 10.1177/1094342005056114. Disponível em: <`http://dx.doi.org/10.1177/1094342005056114`>.

[8] INTEL. *Intel Trace Collector 8.1 Reference Guide*, 2013. Disponível em: <`http://software.intel.com/sites/products/documentation/hpc/ics/itac/81/ITC_Reference_Guide`>.

[9] DEROSE, L., POXON, H. “A Paradigm Change: From Performance Monitoring to Performance Analysis”. In: *Computer Architecture and High Performance Computing, 2009. SBAC-PAD ’09. 21st International Symposium on*, pp. 119–126, 2009. doi: 10.1109/SBAC-PAD.2009.28.

[10] SINGH, J. P., WEBER, W.-D., GUPTA, A. “SPLASH: Stanford parallel applications for shared-memory”, *SIGARCH Comput. Archit. News*, v. 20, pp. 5–44, March 1992. ISSN: 0163-5964. doi: http://doi.acm.org/10.1145/130823.130824. Disponível em: <http://doi.acm.org/10.1145/130823.130824>.

[11] BIENIA, C., KUMAR, S., LI, K. “PARSEC vs. SPLASH-2: A quantitative comparison of two multithreaded benchmark suites on Chip-Multiprocessors”. In: *IISWC-2008, IEEE International Symposium on Workload Characterization.*, pp. 47 –56. IEEE, sept. 2008. doi: 10.1109/IISWC.2008.4636090.

[12] ARMBRUST, M., FOX, A., GRIFFITH, R., et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Relatório técnico, University of California at Berkeley, 2009. Disponível em: <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>.

[13] MORARI, A., GIOIOSA, R., WISNIEWSKI, R., et al. “A Quantitative Analysis of OS Noise”. In: *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pp. 852–863, May 2011. doi: 10.1109/IPDPS.2011.84.

[14] GOLDBERG, R. P. “Architecture of virtual machines”. In: *Proceedings of the June 4-8, 1973, national computer conference and exposition*, AFIPS ’73, pp. 309–318, New York, NY, USA, 1973. ACM. doi: http://doi.acm.org/10.1145/1499586.1499669. Disponível em: <http://doi.acm.org/10.1145/1499586.1499669>.

[15] TICKOO, O., IYER, R., ILLIKKAL, R., et al. “Modeling Virtual Machine Performance: Challenges and Approaches”, *SIGMETRICS Perform. Eval. Rev.*, v. 37, n. 3, pp. 55–60, jan. 2010. ISSN: 0163-5999. doi: 10.1145/1710115.1710126. Disponível em: <http://doi.acm.org/10.1145/1710115.1710126>.

[16] CHAUHAN, J., MAKAROFF, D. “Performance Evaluation of Video-on-demand in Virtualized Environments: The Client Perspective”. In: *Proceedings of the 6th International Workshop on Virtualization Technologies in*

Distributed Computing Date, VTDC '12, pp. 29–36, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-1344-5. doi: 10.1145/2287056.2287066. Disponível em: <http://doi.acm.org/10.1145/2287056.2287066>.

[17] DING, X., GIBBONS, P. B., KOZUCH, M. A. "A Hidden Cost of Virtualization When Scaling Multicore Applications". In: *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, Berkeley, CA, 2013. USENIX. Disponível em: <https://www.usenix.org/conference/hotcloud13/workshop-program/presentations/Ding>.

[18] "Global Internet Phenomena Report". https://www.sandvine.com/downloads/general/global-internet-phenomena/2014/1h-2014-global-internet-phenomena-report.pdf. Acessado em 13-02-2015.

[19] DUTRA, D. L. C. "Contagem de Tempo Em Sistemas de Virtualizaçãos". March 2009. Disponível em: <http://objdig.ufrj.br/60/teses/coppe_m/DiegoLeonelCadetteDutra.pdf>.

[20] PATTERSON, D. A., HENNESSY, J. L. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann, August 2004). ISBN: 1558606041.

[21] TANENBAUM, A. S., WOODHULL, A. S. *Operating Systems Design and Implementation (3rd Edition)*. Upper Saddle River, NJ, USA, Prentice-Hall, Inc., 2005. ISBN: 0131429388.

[22] BOVET, D. P., CESATI, M. *Understanding the LINUX KERNEL*. O'Reilly Media, 2006.

[23] INTEL. *8254/82C54: Introduction to Programmable Interval Timer*, Nov. 2008. http://www.intel.com/design/archives/periphrl/docs/7203.HTM.

[24] OSDEV.ORG. "Intel 8253". Wiki, Nov. 2008. http://wiki.osdev.org/$Programmable\_Interval\_Timer$.

[25] INTEL. *8254/82C54 Programmable Interval Timer*, Nov. 2008. http://www.intel.com/design/archives/periphrl/docs/7178.htm.

[26] INTEL. *Intel 64 and IA-32 Architectures Software Developers Manual. Volume 1: Basic Architecuture*, Nov. 2007.

[27] INTEL. *Intel 64 and IA-32 Architectures Software Developers Manual. Volume 3A: System Programming Guide, Part 1*, Feb. 2008.

[28] HP, INTEL, MICROSOFT, et al. *Advanced Configuration and Power Interface Specification*, December 2005. Revision 3.0a.

[29] AMD. *AMD64 Technology AMD64 Architecture Programmers Manual Volume 2*, September 2007.

[30] AMD. *AMD Technical Bulletin - TSC Dual-Core Issue and Utility Fix*, June 2007.

[31] INTEL. *Intel 64 and IA-32 Architectures Software Developers Manual. Volume 2B: Instruction Set Reference, N-Z*, Nov. 2007.

[32] INTEL. *Intel 64 and IA-32 Architectures Optimization Reference Manual*, Nov. 2007.

[33] THOMPSON, K., RITCHIE, D. M. *gettimeofday*, Nov. 1971.

[34] SCHNEIDER, F. B. "Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial", *ACM Comput. Surv.*, v. 22, n. 4, pp. 299–319, dez. 1990. ISSN: 0360-0300. doi: 10.1145/98163.98167. Disponível em: <http://doi.acm.org/10.1145/98163.98167>.

[35] MILLS, D. L. *Network Time Protocol (NTP)*. Relatório técnico, Nov. 1985. http://www.faqs.org/rfcs/rfc958.html.

[36] MUIR, S. "The Seven Deadly Sins of Distributed Systems". In: *First Workshop on Real, Large Distributed Systems, WORLDS'04*, Dec. 2004. http://www.usenix.org/events/worlds04/tech/muir.html.

[37] ZHANG, X., SCHULZRINNE, H. G. *Voice over TCP and UDP*. Relatório Técnico MSU-CSE-00-2, Department of Computer Science, Columbia University, Sept. 2004. Disponível em: <http://hdl.handle.net/10022/AC:P:29235>.

[38] HONG, C.-Y., LIN, C.-C., CAESAR, M. "Clockscalpel: Understanding Root Causes of Internet Clock Synchronization Inaccuracy". In: Spring, N., Riley, G. (Eds.), *Passive and Active Measurement*, v. 6579, *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 204–213, 2011. ISBN: 978-3-642-19259-3. doi: 10.1007/978-3-642-19260-9_21. Disponível em: <http://dx.doi.org/10.1007/978-3-642-19260-9_21>.

[39] SOCIETY, I. I. "Precision Clock Synchronization Protocol for Networked Measurement and Control Systems", *IEC 61588 First edition 2004-09; IEEE 1588*, 2004. http://ieee1588.nist.gov/.

[40] EIDSON, J. C. *Measurement, Control, and Communication Using IEEE 1588*. Springer, 2006. ISBN: 1846282500.

[41] SNIR, M., WISNIEWSKI, R. W., ABRAHAM, J. A., et al. "Addressing Failures in Exascale Computing", *International Journal of High Performance Computing Applications*, v. 28, n. 2, pp. 129–173, 2014. doi: 10.1177/1094342014522573. Disponível em: <http://hpc.sagepub.com/content/28/2/129.abstract>.

[42] GLEIXNER, T., MCKENNEY, P. E., GUITTOT, V. "Cleaning Up Linux's CPU Hotplug for Real Time and Energy Management", *SIGBED Rev.*, v. 9, n. 4, pp. 49–52, nov. 2012. ISSN: 1551-3688. doi: 10.1145/2452537.2452547. Disponível em: <http://doi.acm.org/10.1145/2452537.2452547>.

[43] MWAIKAMBO, Z., RAJ, A., RUSSELL, R., et al. "Linux kernel hotplug CPU support". In: *Linux Symposium*, v. 2, 2004.

[44] SCHOPP, J., HANSEN, D., KRAVETZ, M., et al. "Hotplug memory redux". In: *Linux Symposium*, p. 151, 2005.

[45] KUMAR, M., DEMSHKI, M., SHIVELEY, R. "Advanced Reliability for Intel Xeon Processor-based Servers". March 2010.

[46] BARROSO, L., HOLZLE, U. "The Case for Energy-Proportional Computing", *Computer*, v. 40, n. 12, pp. 33–37, Dec 2007. ISSN: 0018-9162. doi: 10.1109/MC.2007.443.

[47] SUBRAMANIAM, B., FENG, W.-C. "Towards Energy-proportional Computing for Enterprise-class Server Workloads". In: *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE '13, pp. 15–26, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-1636-1. doi: 10.1145/2479871.2479878. Disponível em: <http://doi.acm.org/10.1145/2479871.2479878>.

[48] DONGARRA, J., BECKMAN, P., MOORE, T., et al. "The International Exascale Software Project Roadmap", *Int. J. High Perform. Comput. Appl.*, v. 25, n. 1, pp. 3–60, fev. 2011. ISSN: 1094-3420. doi: 10.1177/1094342010391989. Disponível em: <http://dx.doi.org/10.1177/1094342010391989>.

[49] PARK, Y., VAN HENSBERGEN, E., HILLENBRAND, M., et al. "FusedOS: Fusing LWK Performance with FWK Functionality in a Heterogeneous Environment". In: *Proceedings of the 2012 IEEE 24th International*

Symposium on Computer Architecture and High Performance Computing, SBAC-PAD '12, pp. 211–218, Washington, DC, USA, 2012. IEEE Computer Society. ISBN: 978-0-7695-4907-1. doi: 10.1109/SBAC-PAD.2012.14. Disponível em: <http://dx.doi.org/10.1109/SBAC-PAD.2012.14>.

[50] WISNIEWSKI, R. W., INGLETT, T., KEPPEL, P., et al. "mOS: An Architecture for Extreme-scale Operating Systems". In: *Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '14, pp. 2:1–2:8, New York, NY, USA, 2014. ACM. ISBN: 978-1-4503-2950-7. doi: 10.1145/2612262.2612263. Disponível em: <http://doi.acm.org/10.1145/2612262.2612263>.

[51] EVANGELINOS, C., WALKUP, R. E., SACHDEVA, V., et al. "Determination of Performance Characteristics of Scientific Applications on IBM Blue Gene/Q", *IBM J. Res. Dev.*, v. 57, n. 1, pp. 99–110, jan. 2013. ISSN: 0018-8646. doi: 10.1147/JRD.2012.2229901. Disponível em: <http://dx.doi.org/10.1147/JRD.2012.2229901>.

[52] FERREIRA, K., BRIDGES, P., BRIGHTWELL, R., et al. "The impact of system design parameters on application noise sensitivity", *Cluster Computing*, v. 16, n. 1, pp. 117–129, 2013. ISSN: 1386-7857. doi: 10.1007/s10586-011-0178-3. Disponível em: <http://dx.doi.org/10.1007/s10586-011-0178-3>.

[53] HOEFLER, T., SCHNEIDER, T., LUMSDAINE, A. "Characterizing the Influence of System Noise on Large-Scale Applications by Simulation". In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–11, Washington, DC, USA, 2010. IEEE Computer Society. ISBN: 978-1-4244-7559-9. doi: 10.1109/SC.2010.12. Disponível em: <http://dx.doi.org/10.1109/SC.2010.12>.

[54] PETRINI, F., KERBYSON, D. J., PAKIN, S. "The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q". In: *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*, SC '03, pp. 55–, New York, NY, USA, 2003. ACM. ISBN: 1-58113-695-1. doi: 10.1145/1048935.1050204. Disponível em: <http://doi.acm.org/10.1145/1048935.1050204>.

[55] JONES, P. W., WORLEY, P. H., YOSHIDA, Y., et al. "Practical performance portability in the Parallel Ocean Program (POP)", *Concurrency*

and Computation: Practice and Experience, v. 17, n. 10, pp. 1317–1327, 2005. ISSN: 1532-0634. doi: 10.1002/cpe.894. Disponível em: <http://dx.doi.org/10.1002/cpe.894>.

[56] CULLER, D., SINGH, J., GUPTA, A. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1998. ISBN: 1-55860-343-1.

[57] HENNING, J. L. "SPEC CPU2000: Measuring CPU Performance in the New Millennium", *Computer*, v. 33, pp. 28–35, July 2000. ISSN: 0018-9162. doi: 10.1109/2.869367. Disponível em: <http://portal.acm.org/citation.cfm?id=619053.621510>.

[58] WEICKER, R. "On the use of SPEC benchmarks in computer architecture research", *SIGARCH Comput. Archit. News*, v. 25, pp. 19–22, March 1997. ISSN: 0163-5964. doi: http://doi.acm.org/10.1145/250015.250018. Disponível em: <http://doi.acm.org/10.1145/250015.250018>.

[59] WOO, S. C., OHARA, M., TORRIE, E., et al. "The SPLASH-2 programs: characterization and methodological considerations", *SIGARCH Comput. Archit. News*, v. 23, pp. 24–36, May 1995. ISSN: 0163-5964. doi: http://doi.acm.org/10.1145/225830.223990. Disponível em: <http://doi.acm.org/10.1145/225830.223990>.

[60] DONGARRA, J. J., LUSZCZEK, P., PETITET, A. "The LINPACK Benchmark: past, present and future", *Concurrency and Computation: Practice and Experience*, v. 15, n. 9, pp. 803–820, 2003. ISSN: 1532-0634. doi: 10.1002/cpe.728. Disponível em: <http://dx.doi.org/10.1002/cpe.728>.

[61] BIENIA, C., KUMAR, S., SINGH, J. P., et al. "The PARSEC benchmark suite: characterization and architectural implications". In: *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pp. 72–81, New York, NY, USA, 2008. ACM. ISBN: 978-1-60558-282-5. doi: http://doi.acm.org/10.1145/1454115.1454128. Disponível em: <http://doi.acm.org/10.1145/1454115.1454128>.

[62] UHLIG, R., NEIGER, G., RODGERS, D., et al. "Intel virtualization technology", *Computer*, v. 38, n. 5, pp. 48–56, May 2005. ISSN: 0018-9162. doi: 10.1109/MC.2005.163.

[63] NEIGER, G., SANTONI, A., LEUNG, F., et al. "Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization", *In-*

tel Technology Journal, v. 10, n. 3, Aug. 2006. ISSN: 1535-864X. doi: 10.1535/itj.1003.01.

[64] AMD. *AMD64 Virtualization Codenamed "Pacifica"Technology: Secure Virtual Machine Architecture Reference Manual*, May 2005.

[65] POPEK, G. J., GOLDBERG, R. P. "Formal requirements for virtualizable third generation architectures", *Commun. ACM*, v. 17, n. 7, pp. 412–421, 1974. ISSN: 0001-0782. doi: http://doi.acm.org/10.1145/361011.361073.

[66] CREASY, R. J. "The Origin of the VM/370 Time-Sharing System", *IBM Journal of Research & Development*, v. 25, n. 5, pp. 483–490, Sept. 1981.

[67] SMITH, J. E., NAIR, R. *Virtual Machines: Versatile Platforms for Systems and Processes*. Morgan Kaufmann, June 2005. ISBN: 1558609105. Disponível em: <`http://www.amazon.ca/exec/obidos/redirect?tag=citeulike09-20&path=ASIN/1558609105`>.

[68] KIVITY, A., KAMAY, Y., LAOR, D., et al. "kvm: the Linux virtual machine monitor". In: *OLS '07: The 2007 Ottawa Linux Symposium*, pp. 225–230, jul 2007.

[69] BELLARD, F. "QEMU, a fast and portable dynamic translator". In: *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '05, pp. 41–41, Berkeley, CA, USA, 2005. USENIX Association. Disponível em: <`http://portal.acm.org/citation.cfm?id=1247360.1247401`>.

[70] BARHAM, P., DRAGOVIC, B., FRASER, K., et al. "Xen and the art of virtualization". In: *ACM Symposium on Operating Systems Principles*, pp. 164–177, 2003.

[71] PRATT, I., FRASER, K., HAND, S., et al. "Xen 3.0 and the Art of Virtualization". In: *Proceedings of Linux Symposium 2005*, July 2005.

[72] OSMAN, S., SUBHRAVETI, D., SU, G., et al. "The design and implementation of Zap: a system for migrating computing environments", *SIGOPS Oper. Syst. Rev.*, v. 36, n. SI, pp. 361–376, 2002. ISSN: 0163-5980. doi: http://doi.acm.org/10.1145/844128.844162.

[73] SWSOFT. "Virtuozzo Containers 4". May 2008. http://www.parallels.com/en/products/virtuozzo/.

[74] HELSLEY, M. *LXC: Linux container tools*. IBM, Fevereiro 2009. Disponível em: <http://download.boulder.ibm.com/ibmdl/pub/software/dw/linux/l-lxc-containers/l-lxc-containers-pdf.pdf>.

[75] CHUN, B., CULLER, D., ROSCOE, T., et al. "PlanetLab: An Overlay Testbed for Broad-coverage Services", *SIGCOMM Comput. Commun. Rev.*, v. 33, n. 3, pp. 3–12, jul. 2003. ISSN: 0146-4833. doi: 10.1145/956993.956995. Disponível em: <http://doi.acm.org/10.1145/956993.956995>.

[76] SWSOFT. *OpenVZ Users Guide*, Nov. 2005.

[77] KOZUCH, M., SATYANARAYANAN, M. "Internet suspend/resume". In: *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on*, pp. 40–46, 2002. doi: 10.1109/MCSA.2002.1017484.

[78] "Linux container". Web. Disponível em: <http://lxc.sourceforge.net/>.

[79] BHATTIPROLU, S., BIEDERMAN, E. W., HALLYN, S., et al. "Virtual servers and checkpoint/restart in mainstream Linux", *SIGOPS Oper. Syst. Rev.*, v. 42, pp. 104–113, July 2008. ISSN: 0163-5980. doi: http://doi.acm.org/10.1145/1400097.1400109. Disponível em: <http://doi.acm.org/10.1145/1400097.1400109>.

[80] LAADAN, O., HALLYN, S. E. "Linux-CR: Transparent Application Checkpoint-Restart in Linux". In: *The 2010 Ottawa Linux Symposium, OLS '10*, jul 2010.

[81] FOSTER, I., ZHAO, Y., RAICU, I., et al. "Cloud computing and grid computing 360-degree compared". In: *IEEE 2008 Grid Computing Environments Workshop, GCE '08*. IEEE, 2008. ISBN: 978-1-4244-2860-1. doi: http://dx.doi.org/10.1109/GCE.2008.4738445. Disponível em: <http://dx.doi.org/10.1109/GCE.2008.4738445>.

[82] VAQUERO, L. M., RODERO-MERINO, L., CACERES, J., et al. "A break in the clouds: towards a cloud definition", *SIGCOMM Comput. Commun. Rev.*, v. 39, pp. 50–55, December 2008. ISSN: 0146-4833. doi: http://doi.acm.org/10.1145/1496091.1496100. Disponível em: <http://doi.acm.org/10.1145/1496091.1496100>.

[83] "Salesforce Customer Relationships Management (CRM) System". Web, 2011. Disponível em: <http://www.salesforce.com/>.

[84] "Google App Engine". Web, 2011. Disponível em: <http://code.google.com/appengine/>.

[85] CHOHAN, N., BUNCH, C., PANG, S., et al. "AppScale: Scalable and Open AppEngine Application Development and Deployment". In: *First International Conference on Cloud Computing, CloudComp'09*. ICST, 2009. Disponível em: <http://www.cs.ucsb.edu/~ckrintz/papers/cloudcomp09.pdf>.

[86] TRIVEDI, K. S., KIM, D. S., GHOSH, R. "Resilience in computer systems and networks". In: *Proceedings of the 2009 International Conference on Computer-Aided Design*, ICCAD '09, pp. 74–77, New York, NY, USA, 2009. ACM. ISBN: 978-1-60558-800-1. doi: http://doi.acm.org/10.1145/1687399.1687415. Disponível em: <http://doi.acm.org/10.1145/1687399.1687415>.

[87] YOUSEFF, L., BUTRICO, M., DA SILVA, D. "Toward a Unified Ontology of Cloud Computing". In: *IEEE 2008 Grid Computing Environments Workshop, GCE '08*. IEEE, 2008. ISBN: 978-1-4244-2860-1. doi: http://dx.doi.org/10.1109/GCE.2008.4738443. Disponível em: <http://dx.doi.org/10.1109/GCE.2008.4738443>.

[88] "Google Cloud Platform: Compute Engine". Web, 2012. Disponível em: <https://cloud.google.com/compute/>.

[89] "Enomalism elastic computing infrastructure". Web, 2011. Disponível em: <http://www.enomaly.com/>.

[90] NURMI, D., WOLSKI, R., GRZEGORCZYK, C., et al. "The Eucalyptus Open-Source Cloud-Computing System". In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pp. 124–131, Washington, DC, USA, 2009. IEEE Computer Society. ISBN: 978-0-7695-3622-4. doi: http://dx.doi.org/10.1109/CCGRID.2009.93. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2009.93>.

[91] SOTOMAYOR, B., MONTERO, R. S., LLORENTE, I. M., et al. "Virtual Infrastructure Management in Private and Hybrid Clouds", *IEEE Internet Computing*, v. 13, pp. 14–22, September 2009. ISSN: 1089-7801. doi: 10.1109/MIC.2009.119. Disponível em: <http://portal.acm.org/citation.cfm?id=1638588.1638692>.

[92] "Openstack cloud software". Web, 2015. Disponível em: <http://www.openstack.org/>.

[93] *Eucalyptus Administrator's Guide (2.0).* Eucalyptus, 2011. Disponível em: <http://open.eucalyptus.com/wiki/EucalyptusAdministratorGuide_v2.0>.

[94] "Amazon Web Services - Amazon Simple Storage Service". Web, 2011. Disponível em: <http:/s3.amazonaws.com>.

[95] SOTOMAYOR, B., MONTERO, R. S., LLORENTE, I. M., et al. "Capacity Leasing in Cloud Systems using the OpenNebula Engine". In: *The First Workshop on Cloud Computing and its Applications, CCA'08*, 2008.

[96] *Managing Hosts and Clusters 2.2.* OpenNebula.org, 2011. Disponível em: <http://opennebula.org/documentation:rel2.2:cluster_guide>.

[97] *OpenStack Cloud Administrator Guide.* OpenStack.org, Abril 2015. Disponível em: <http://docs.openstack.org/admin-guide-cloud/admin-guide-cloud.pdf>.

[98] "OpenStack Image - Glance". Web, 2015. Disponível em: <http://www.openstack.org/projects/image-service/>.

[99] KILLALEA, T. "Building Scalable Web Services", *Queue*, v. 6, n. 6, pp. 10–13, out. 2008. ISSN: 1542-7730. doi: 10.1145/1466443.1466447. Disponível em: <http://doi.acm.org/10.1145/1466443.1466447>.

[100] DECANDIA, G., HASTORUN, D., JAMPANI, M., et al. "Dynamo: Amazon's Highly Available Key-value Store", *SIGOPS Oper. Syst. Rev.*, v. 41, n. 6, pp. 205–220, out. 2007. ISSN: 0163-5980. doi: 10.1145/1323293.1294281. Disponível em: <http://doi.acm.org/10.1145/1323293.1294281>.

[101] GILBERT, S., LYNCH, N. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services", *SIGACT News*, v. 33, n. 2, pp. 51–59, jun. 2002. ISSN: 0163-5700. doi: 10.1145/564585.564601. Disponível em: <http://doi.acm.org/10.1145/564585.564601>.

[102] DEAN, J., GHEMAWAT, S. "MapReduce: Simplified Data Processing on Large Clusters", *Commun. ACM*, v. 51, n. 1, pp. 107–113, jan. 2008. ISSN: 0001-0782. doi: 10.1145/1327452.1327492. Disponível em: <http://doi.acm.org/10.1145/1327452.1327492>.

[103] BAUER, M., CLARK, J., SCHKUFZA, E., et al. "Programming the Memory Hierarchy Revisited: Supporting Irregular Parallelism in Sequoia", *SIGPLAN Not.*, v. 46, n. 8, pp. 13–24, fev. 2011. ISSN: 0362-1340. doi:

10.1145/2038037.1941558. Disponível em: <http://doi.acm.org/10.1145/2038037.1941558>.

[104] GELADO, I., STONE, J. E., CABEZAS, J., et al. "An Asymmetric Distributed Shared Memory Model for Heterogeneous Parallel Systems", *SIGPLAN Not.*, v. 45, n. 3, pp. 347–358, mar. 2010. ISSN: 0362-1340. doi: 10.1145/1735971.1736059. Disponível em: <http://doi.acm.org/10.1145/1735971.1736059>.

[105] PINHO, L. B., AMORIM, C. L. "Assessing the efficiency of stream reuse techniques in P2P video-on-demand systems", *Journal of Network and Computer Applications (JNCA)*, v. 29, n. 1, pp. 25–45, January 2006. ISSN: 1084-8045, Academic Press - Elsevier.

[106] ABRAM-PROFETA, E. L., SHIN, K. G. "Scheduling Video Programs in Near Video-on-demand Systems". In: *Proceedings of the Fifth ACM International Conference on Multimedia*, MULTIMEDIA '97, pp. 359–369, New York, NY, USA, 1997. ACM. ISBN: 0-89791-991-2. doi: 10.1145/266180.266387. Disponível em: <http://doi.acm.org/10.1145/266180.266387>.

[107] WHATELY, L. L. A. *Uso de Virtualização na construção de serviços de streaming escaláveis*. Tese de Doutorado, COPPE/Sistemas, UFRJ, Rio de Janeiro, RJ, Brasil, July 2008. (in Portuguese).

[108] GORDON, A., BEN-YEHUDA, M., FILIMONOV, D., et al. "VAMOS: Virtualization Aware Middleware". In: *Proceedings of the 3rd Conference on I/O Virtualization*, WIOV'11, pp. 3–3, Berkeley, CA, USA, 2011. USENIX Association. Disponível em: <http://dl.acm.org/citation.cfm?id=2001555.2001558>.

[109] HOELZLE, U., BARROSO, L. A. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009. ISBN: 159829556X, 9781598295566.

[110] WIEGAND, T., SULLIVAN, G., BJONTEGAARD, G., et al. "Overview of the H.264/AVC video coding standard", *Circuits and Systems for Video Technology, IEEE Transactions on*, v. 13, n. 7, pp. 560–576, July 2003. ISSN: 1051-8215. doi: 10.1109/TCSVT.2003.815165.

[111] FFMPEG. Disponível em: <http://www.ffmpeg.org>. Acessado em 13-02-2015.

[112] DUTRA, D. L. C., WHATELY, L. L. A., AMORIM, C. L. D. "Attaining Strictly Increasing and Precise Time Count in Energy-Efficient Computer Systems". In: *Computer Architecture and High Performance Computing (SBAC-PAD), 2013 25th International Symposium on*, pp. 65–72, 2013. doi: 10.1109/SBAC-PAD.2013.3.

[113] DUTRA, D., WHATELY, L., AMORIM, C. *A Highly Effective System Clock for Energy-efficient Computer Systems*. Relatório Técnico ES-745/13, Systems and Computer Engineering Prog., COPPE,UFRJ, Jan. 2013.

[114] VEITCH, D., BABU, S., PÁSZTOR, A. "Robust synchronization of software clocks across the internet". In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, IMC '04, pp. 219–232, New York, NY, USA, 2004. ACM. ISBN: 1-58113-821-0. doi: 10.1145/1028788.1028817.

[115] ELSON, J. *Time Synchronization in Wireless Sensor Networks*. Tese de Doutorado, University of California, May 2003.

[116] FORUM, M. P. I. *MPI: A Message-Passing Interface Standard Version 3.0*. Relatório técnico, 2012.

[117] JONES, T., KOENIG, G. A. "Clock synchronization in high-end computing environments: a strategy for minimizing clock variance at runtime", *Concurrency and Computation: Practice and Experience*, v. 25, n. 6, pp. 881–897, 2013. ISSN: 1532-0634. doi: 10.1002/cpe.2868.

[118] CRISTIAN, F. "A probabilistic approach to distributed clock synchronization". In: *ICDCS'89*, pp. 288–296, jun 1989. doi: 10.1109/ICDCS.1989.37958.

[119] MESSAGE PASSING INTERFACE FORUM. *MPI: a message passing interface standard version 3.0*. Relatório técnico, Sept 2012.

[120] CORPORATION, S. *TSync-PCIe PCI Express TIME CODE PROCESSOR with OPTIONAL GPS and TSync-PCIe-PTP*, May. 2014.

[121] WALKER, E. "Benchmarking Amazon EC2 for high-performance scientific computing", *USENIX Login*, v. 33, n. 5, pp. 18–23, oct 2008.

[122] NAPPER, J., BIENTINESI, P. "Can cloud computing reach the top500?" In: *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW

'09, pp. 17–20, New York, NY, USA, 2009. ACM. ISBN: 978-1-60558-557-4. doi: http://doi.acm.org/10.1145/1531666.1531671. Disponível em: <http://doi.acm.org/10.1145/1531666.1531671>.

[123] "Linpack". http://www.netlib.org/benchmark/hpl/. Acessado em 13-02-2015.

[124] DONGARRA, J. J., VAN DE GEIJN, R. A., WALKER, D. W. "Scalability Issues Affecting the Design of a Dense Linear Algebra Library", *J. Parallel Distrib. Comput.*, v. 22, n. 3, pp. 523–537, set. 1994. ISSN: 0743-7315. doi: 10.1006/jpdc.1994.1108. Disponível em: <http://dx.doi.org/10.1006/jpdc.1994.1108>.

[125] YIGITBASI, N., IOSUP, A., EPEMA, D., et al. "C-Meter: A Framework for Performance Analysis of Computing Clouds". In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pp. 472–477, Washington, DC, USA, 2009. IEEE Computer Society. ISBN: 978-0-7695-3622-4. doi: http://dx.doi.org/10.1109/CCGRID.2009.40. Disponível em: <http://dx.doi.org/10.1109/CCGRID.2009.40>.

[126] MAO, M., HUMPHREY, M. "A Performance Study on the VM Startup Time in the Cloud". In: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pp. 423–430, June 2012. doi: 10.1109/CLOUD.2012.103.

[127] KOSSMANN, D., KRASKA, T., LOESING, S. "An evaluation of alternative architectures for transaction processing in the cloud". In: *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, pp. 579–590, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0032-2. doi: http://doi.acm.org/10.1145/1807167.1807231. Disponível em: <http://doi.acm.org/10.1145/1807167.1807231>.

[128] SOBEL, W., SUBRAMANYAM, S., SUCHARITAKUL, A., et al. "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0". In: *The First Workshop on Cloud Computing and its Applications, CCA'08*, 2008.

[129] UEDA, Y., NAKATANI, T. "Performance variations of two open-source cloud platforms". In: *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*, IISWC '10, pp. 1–10, Washington, DC, USA, 2010. IEEE Computer Society. ISBN: 978-1-4244-9297-

8. doi: http://dx.doi.org/10.1109/IISWC.2010.5650280. Disponível em: <http://dx.doi.org/10.1109/IISWC.2010.5650280>.

[130] FERDMAN, M., ADILEH, A., KOCBERBER, O., et al. "Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware". In: *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pp. 37–48, New York, NY, USA, 2012. ACM. ISBN: 978-1-4503-0759-8. doi: 10.1145/2150976.2150982. Disponível em: <http://doi.acm.org/10.1145/2150976.2150982>.

[131] LI, Z., ZHANG, H., O'BRIEN, L., et al. "On Evaluating Commercial Cloud Services: A Systematic Review", *J. Syst. Softw.*, v. 86, n. 9, pp. 2371–2393, set. 2013. ISSN: 0164-1212. doi: 10.1016/j.jss.2013.04.021. Disponível em: <http://dx.doi.org/10.1016/j.jss.2013.04.021>.

[132] YANG, H., BRESLOW, A., MARS, J., et al. "Bubble-flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers". In: *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ISCA '13, pp. 607–618, New York, NY, USA, 2013. ACM. ISBN: 978-1-4503-2079-5. doi: 10.1145/2485922.2485974. Disponível em: <http://doi.acm.org/10.1145/2485922.2485974>.

[133] GAMBI, A., MOLDOVAN, D., COPIL, G., et al. "On Estimating Actuation Delays in Elastic Computing Systems". In: *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '13, pp. 33–42, Piscataway, NJ, USA, 2013. IEEE Press. ISBN: 978-1-4673-4401-2. Disponível em: <http://dl.acm.org/citation.cfm?id=2487336.2487345>.

[134] LAKSHMAN, A., MALIK, P. "Cassandra: A Decentralized Structured Storage System", *SIGOPS Oper. Syst. Rev.*, v. 44, n. 2, pp. 35–40, abr. 2010. ISSN: 0163-5980. doi: 10.1145/1773912.1773922. Disponível em: <http://doi.acm.org/10.1145/1773912.1773922>.

[135] DORY, T., MEJÍAS, B., ROY, P. V., et al. "Measuring Elasticity for Cloud Databases". In: *Proceedings of the The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, 2011.

[136] CATTELL, R. "Scalable SQL and NoSQL Data Stores", *SIGMOD Rec.*, v. 39, n. 4, pp. 12–27, maio 2011. ISSN: 0163-5808. doi: 10.1145/1978915.1978919. Disponível em: <http://doi.acm.org/10.1145/1978915.1978919>.

[137] LI, A., YANG, X., KANDULA, S., et al. "Comparing Public-Cloud Providers", *Internet Computing, IEEE*, v. 15, n. 2, pp. 50–53, March 2011. ISSN: 1089-7801. doi: 10.1109/MIC.2011.36.

[138] CORPORATION, S. P. E. "SPECjvm2008". Disponível em: <http://www.spec.org/jvm2008>. Acessado em 13-02-2015.

[139] BARKER, S., CHI, Y., HACIGÜMÜS, H., et al. "ShuttleDB: Database-Aware Elasticity in the Cloud". In: *11th International Conference on Autonomic Computing (ICAC 14)*, pp. 33–43, Philadelphia, PA, jun. 2014. USENIX Association. ISBN: 978-1-931971-11-9. Disponível em: <https://www.usenix.org/conference/icac14/technical-sessions/presentation/barker>.

[140] COOPER, B. F., SILBERSTEIN, A., TAM, E., et al. "Benchmarking Cloud Serving Systems with YCSB". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pp. 143–154, New York, NY, USA, 2010. ACM. ISBN: 978-1-4503-0036-0. doi: 10.1145/1807128.1807152. Disponível em: <http://doi.acm.org/10.1145/1807128.1807152>.

[141] VARGHESE, B., AKGUN, O., MIGUEL, I., et al. "Cloud Benchmarking for Performance". In: *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pp. 535–540, Dec 2014. doi: 10.1109/CloudCom.2014.28.

[142] "Bonnie++". http://www.coker.com.au/bonnie++/. Acessado em 13-02-2015.

[143] GILLAM, L., LI, B., O'LOUGHLIN, J., et al. "Fair benchmarking for cloud computing systems", *Journal of Cloud Computing: Advances, Systems and Applications*, v. 2, n. 1, pp. 6, 2013.

[144] "STREAM". http://www.cs.virginia.edu/stream/. Acessado em 13-02-2015.

[145] "MPPTEST". http://www.mcs.anl.gov/research/projects/mpi/mpptest/. Acessado em 13-02-2015.

[146] TIAN, G.-S., TIAN, Y.-C., FIDGE, C. "High-Precision Relative Clock Synchronization Using Time Stamp Counters". In: *ICECCS '08: Proceedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems (iceccs 2008)*, pp. 69–78, Washington, DC, USA, 2008. IEEE Computer Society. ISBN: 978-0-7695-3139-7. doi: http://dx.doi.org/10.1109/ICECCS.2008.39.

[147] SOBEIH, A., HACK, M., LIU, Z., et al. "Almost Peer-to-Peer Clock Synchronization", *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pp. 1–10, March 2007. doi: 10.1109/IPDPS.2007.370211.

[148] VEITCH, D., RIDOUX, J., KORADA, S. B. "Robust synchronization of absolute and difference clocks over networks", *IEEE/ACM Trans. Netw.*, v. 17, pp. 417–430, April 2009. ISSN: 1063-6692. doi: 10.1109/TNET.2008.926505.

[149] BROOMHEAD, T., CREMEAN, L., RIDOUX, J., et al. "Virtualize everything but time". In: *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pp. 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[150] DE SOUZA, A. F., DE SOUZA, S. F., DE AMORIM, C. L., et al. "Hardware Supported Synchronization Primitives for Clusters". In: *PDPTA'08*, pp. 520–526, 2008.

[151] AMORIM, C. L., DE SOUZA, A. F. "Distributed global clock for clusters of computers". United States Patent No. 20060212738, September 2006.

[152] RATZEL, R., GREENSTREET, R. "Toward higher precision", *Commun. ACM*, v. 55, n. 10, pp. 38–47, out. 2012. ISSN: 0001-0782. doi: 10.1145/2347736.2347750.

[153] CORRELL, K., BARENDT, N. "Design Considerations for Software Only Implementations of the IEEE 1588 Precision Time Protocol". In: *In Conference on IEEE 1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2006.

[154] GIAMPAPA, M., GOODING, T., INGLETT, T., et al. "Experiences with a Lightweight Supercomputer Kernel: Lessons Learned from Blue Gene's CNK". In: *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pp. 1–10, Washington, DC, USA, 2010. IEEE Computer Society. ISBN: 978-1-4244-7559-9. doi: 10.1109/SC.2010.22. Disponível em: <http://dx.doi.org/10.1109/SC.2010.22>.

Apêndice A

Avaliando as limitações do NTP no *cluster* local

Dentre as motivações deste trabalho encontra-se as limitações do uso do NTP como mecanismo de sincronização para aplicações de tempo-real em um *cluster* de computadores compartilhados, descritas por Muir [36]. Apesar da informação qualitativa apresentada no referido artigo, optou-se por avaliar como o NTP funciona no *cluster* utilizado nos experimentos desta tese. O experimento foi projetado para avaliar quanto apenas o atraso na execução do serviço do NTP pode degradar a medida temporal para uma aplicação, dado que em um sistema real a degradação esperada é ainda maior devido ao tempo gasto pelas interrupções, e de contenção na rede e nos barramentos internos do nó.

Para o experimento foram construídos três cenários distintos, onde variou-se o intervalo de tempo entre as execuções do cliente NTP e cada cenário foi repetido 200 vezes, onde cada repetição realizando 20 execuções do serviço NTP. Em todos os cenários é importante ressaltar que, ao contrario da situação descrita no referido artigo, os nós computacionais e a rede encontravam-se inteiramente disponíveis para o NTP, não ocorrendo contenções devido a sobrecarga do nó ou da rede.

Inicialmente, o experimento foi realizado utilizando um intervalo de 0 s entre as execuções do serviço do NTP, esse cenário representa a situação ideal, onde o cliente do NTP é executado consecutivamente sem executar a chamada de sistema nanosleep() entre as execuções. O segundo cenário utilizou um intervalo de 10 s, ou seja a aplicação esperava 10 s entre as execução do cliente NTP, este cenário representa o funcionamento do protocolo NTP em nós computacionais que não encontram-se sobrecarregados. O terceiro e último cenário utilizou um intervalo de tempo entre as execuções de 600 s, sendo que o intervalo foi escolhido por representar o perfil de execução do NTP em nós computacionais que encontram-se sobrecarregados [36].

A Tabela A.1 sumariza os resultados obtidos para o experimento onde na pri-

meira coluna tem-se o cenário testado, na segundo coluna o intervalo entre execuções do *deamon* do NTP, na terceira a média dos valores dos Δ's aplicados sobre o relógio local e na quarta coluna o desvio padrão associado a média.

Tabela A.1: Estatísticas dos Δs aplicados pelo protocolo NTP sobre o relógio local

Cenário	Intervalo entre execuções (s)	Média (μs)	Desvio Padrão
1	0 s	$503,17$	$111,97$
2	10 s	$660,11$	$235,27$
3	600 s	$10.169,35$	$651,26$

Uma aplicação de tempo-real, cujas tarefas devam ocorrer em intervalos de tempo iguais ou menores que 1 ms não executará corretamente sobre esse *cluster*, pois duas atualizações sucessivas de $503,17$ μs fazem com que o tempo medido pela aplicação seja superior a 1 ms. Durante os experimentos o maior Δ aplicado sobre o valor corrente do relógio local realizado pelo NTP foi de $1,879$ s. Com base nestes resultados é possível afirmar que os problemas descritos por Muir continuam a ocorrer em sistemas computacionais que utilizam o NTP como mecanismo de suporte as aplicações sensíveis ao tempo.

Os resultados obtidos (Tabela A.1) demonstraram que mesmo sem a degradação provocada pela perda de interrupções em nós sobrecarregados as atualizações feitas pelo protocolo NTP o tornam uma solução arriscada para sistemas de tempo-real.