JULIANA BÄCHTOLD

UMA ARQUITETURA PARA AUDITORIA MULTIPARTES DOS SERVIÇOS PROVIDOS EM NUVEM COMPUTACIONAL

Dissertação apresentada em agosto/2012 ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná, como requisito parcial à obtenção do título de Mestre.

Área de Concentração: Ciências da Computação

Orientador: Prof. Dr Altair Olivo Santin

CURITIBA
2012

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná

Sistema

Bächtold, Juliana
B124m Uma arquitetura para auditoria multipartes dos serviços providos em nuvem
2012 computacional / Juliana Bärchtold ; orientador, Altair Olivo Santin. – 2012.
xii, 74 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2012
Bibliografia: f. 72-74

1. Informática. 2. Auditoria. 3. Computação em nuvem. 4. Contratos de nível de serviços. 5. Sistemas de recuperação da informação – Segurança. I. Santin, Altair Olivo. II. Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática. III. Título.

CDD 20. ed. – 004.068

Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central

Dedico este trabalho à minha filha, Gabriela, e ao meu marido, Alessandro, grandes incentivadores do meu projeto e alegrias da minha vida.

Agradecimentos

Agradeço a Deus, por conceder a graça da vida e da saúde.

Agradeço ao meu orientador Altair Olivo Santin que esteve ao meu lado, prestando auxílio, apoio e direcionamento durante toda minha jornada.

Agradeço ao programa de bolsas da CAPES e ao bolsista Eduardo Viegas, por suas contribuições e dedicação.

Agradeço à minha família pelo suporte e carinho, em especial aos meus pais e ao meu marido Alessandro.

Resumo

Este trabalho apresenta uma arquitetura para auditoria multipartes de SLA para computação em nuvem, onde são auditados provedor de IaaS, contratante de IaaS (provedor de SaaS) e cliente de SaaS. O objetivo é auditar problemas internos e externos ao ambiente de nuvem de modo incontestável pelas partes. A proposta emprega inspetores (agentes de coleta para auditoria) e auditor independente (terceiro), capazes de identificar desvios de SLA através de informações coletadas nos ambientes das partes. Os resultados mostram que é possível auditar e diagnosticar problemas na nuvem combinando informações através de uma autoria independente, sendo possível inclusive evitar conflito de interesse dos inspetores.

Palavras-chave: Auditoria, Nível de Serviço, SLA, Computação em Nuvem.

Abstract

This work purposes a multiparty SLA auditing architecture for cloud computing, where IaaS provider, IaaS client (SaaS provider) and SaaS client are audited. The objective is to audit the cloud environment, identifying both internal and external problems undisputedly by the parties. The proposal uses inspectors (auditing collector agents) and independent auditor (third party), capable of identifying SLA deviations through information collected on the environments of contractor, client and provider. The results indicate that it is possible to audit and diagnose problems in the cloud by combining information from the parties with an independent audit, moreover avoiding inspectors' conflict of interest.

Keywords: Auditing, Service Level, SLA, Cloud Computing.

Lista de Figuras

Lista de Tabelas e Quadros

Lista de Abreviações

Amazon EC2 – *Amazon Elastic Cloud Computing*

API – *Application Programming Interface* (Interface de Programação de Aplicativos)

AS – *Atomic Service* (Serviços Básicos)

BIOS – *Basic Input/Output System* (Sistema Básico de Entrada/Saída)

CLC – *Cloud Controller*

CPU – *Central Processing Unit* (Unidade Central de Processamento)

CS – *Composite Services* (Serviços Combinados)

DM&C – *Domain Management and Control*

DMTF – *Distributed Management Task Force* (Força Tarefa para Gerenciamento Distribuído)

FoSII – *Foundations of Self-governing ICT Infrastructures*

HTML – *HyperText Markup Language* (Linguagem de Marcação de Hipertexto)

HTTP – *Hypertext Transfer Protocol* (Protocolo de Transferência de Hipertexto)

HVM – *Host Virtual Machine*

I/O – *Input/Output* (Entrada/Saída)

IaaS – *Infrastructure as a Service* (Infraestrutura como um Serviço)

IS – *Infrastructure Services* (Serviços de Infraestrutura)

JVM – *Java Virtual Machine* (Máquina Virtual Java)

MIE – *Measurement Interference Error* (Erro de Interferência de Medida)

MTTF – *Mean Time To Failure* (Tempo Médio entre Falhas)

MTTR – *Mean Time To Repair* (Tempo Médio para Reparo)

NIST – *National Institute of Standards and Technology* (Instituto Nacional de Padrões e Tecnologia, EUA)

NTP – *Network Time Protocol*

PaaS – *Platform as a Service* (Plataforma como um Serviço)

PV – *Para Virtualizated* (Paravirtualizado)

QoE – *Quality of Experience* (Qualidade da Experiência)

QoS – *Quality of Service* (Qualidade de Serviço)

RS – *Resource Service* (Serviço de Recursos)

SaaS – *Software as a Service* (Software como um Serviço)

SGBD – Sistema de Gerenciamento de Banco de Dados

SLA – *Service Level Agreement* (Acordo de Nível de Serviço)

SLI – *Service Level Indicators* (Indicadores de Nível de Serviço)

SLO – *Service Level Objective* (Objetivo de Nível de Serviço)

SMTP – *Simple Mail Transfer Protocol*

SO – Sistema Operacional

SOAP – *Simple Object Access Protocol* (Protocolo Simples de Acesso a Objetos)

SRS – *System's Resource State* (Estado de Recursos do Sistema)

TPA – *Third-Party Auditor* (Auditor externo independente)

VM – *Virtual Machine* (Máquina virtual)

SUMÁRIO

1 Introdução

Computação em nuvem (*cloud computing*) é definida pelo NIST (*National Institute of Standards and Technology*, Instituto Nacional de Padrões e Tecnologia, EUA) como um modelo para habilitar acesso de qualquer lugar, sob demanda e conveniente a um *pool* de recursos computacionais que podem ser rapidamente provisionados e liberados com o mínimo de esforços de gerenciamento ou interação do provedor de serviço [1]. Cinco características essenciais devem estar presentes no serviço computacional para caracterizá-lo como em nuvem:

- Capacidade do próprio cliente provisionar recursos sob demanda;
- Acesso aos recursos por clientes heterogêneos;
- Disponibilidade de *pool* de recursos para vários locatários simultaneamente, alocados de acordo com a necessidade de consumo, independentemente de sua localização geográfica;
- Elasticidade rápida, de maneira que os recursos podem ser rapidamente ou até automaticamente redimensionados suportando a diminuição ou aumento da demanda por recursos computacionais, em qualquer momento;
- Serviços entregues quantificados, de acordo com a natureza do serviço contratado, para que sejam controlados e reportados, provendo transparência entre cliente e provedor de serviço [1].

Devido a estas características, a computação em nuvem passou a oferecer capacidade e ambiente para armazenamento de dados, bem como poder de processamento escalável, permitindo atender demandas elásticas enquanto diminui investimentos [2], o que explica o grande crescimento de seu uso.

Aos benefícios inerentes à sua própria definição, somam-se questões como a eficiência energética que o modelo oferece, ao compartilhar recursos, diminui a ociosidade de servidores, aumentando a eficiência energética do datacenter [3].

Computação em nuvem representa um dos mais significativos movimentos de Tecnologia da Informação nos últimos anos [4]. Nuvens constituem a próxima geração de *datacenters*, utilizando servidores virtuais, dinamicamente provisionadas sob demanda como um conjunto de recursos personalizados para atender um nível

de serviço específico [2]. “Tudo o que caracterizamos como computador atualmente, nada mais é do que apenas um dispositivo que nos conecta a um grande computador que estamos coletivamente criando (nuvem computacional)” [5].

O crescimento da computação em nuvem traz à tona antiga proposta da computação como a 5ª utilidade, conforme a visão de Leonard Kleinrock que já em 1969 declarou que “As redes de computadores ainda estão em sua infância, mas, com o seu crescimento e sofisticação (das tecnologias), nós provavelmente veremos o surgimento da computação como um serviço de utilidades da mesma forma que os serviços de energia elétrica e telefonia, e assim estarão disponíveis em residências e escritórios espalhados no país” [6].

Apesar do entusiasmo gerado pelo tema, as mesmas características que fazem o modelo de computação em nuvem ser atrativo, limitam a gestão dos recursos contratados se comparados ao modelo tradicional baseado em provisionamento local, em geral com prazo significativo para entrega do recurso solicitado (*on premise).* O controle do recurso computacional é limitado à interface de gerenciamento fornecida pelo provedor de serviço, restringindo as ações de administração dos recursos computacionais.

A perda de controle direto dos recursos computacionais é relatada em [8] e [9] como uma das principais preocupações das organizações: Enquanto as preocupações de segurança e privacidade para serviços em computação em nuvem são similares aos serviços computacionais tradicionais, elas são ampliadas pelo fato da administração e controle serem externas à organização e sujeitas a má-administração sobre os ativos de informação. Migrar para um ambiente de nuvem pública requer transferir controle sobre a informação, assim como sobre os componentes de sistema para o provedor de serviços em nuvem que estavam anteriormente sobre o controle direto da organização. A perda de controle sobre aspectos físicos e lógicos do sistema e dos dados diminui a capacidade da organização manter-se alerta e informada sobre ocorrências, ponderar alternativas, definir prioridades e realizar alterações em políticas de segurança e uso.

Ao transferir o controle operacional, a organização passa a depender do nível de serviço entregue pelo provedor. A operação de uma organização e a retenção de informações críticas podem ser prejudicadas por problemas em serviços em nuvem,

ou por serviços em nuvem que não estejam ajustados às necessidades da empresa. Algumas iniciativas [9] tem como objetivo avaliar o serviço provido quanto à sua compatibilidade com normas e especificações de segurança, afim de auxiliar o cliente a escolher o provedor que atenda ao nível de segurança necessário ao seu negócio.

Usualmente em cenários *on premise* define-se um Acordo de Nível de Serviço (SLA, *Service Level Agreement*) para garantir que os serviços adquiridos sejam implantados e mantidos aderentes às necessidades de uma organização. Também aplicável para computação em nuvem, o SLA é um documento que contém os níveis de serviço esperados pelo cliente e garantidos pelo fornecedor, percorrendo os domínios de Qualidade de Serviço (QoS, *Quality of Service*) e outros critérios qualitativos definidos pelas duas entidades.

1.1 Motivação

A simples redação do SLA, entretanto, é insuficiente para prevenir desvios entre o nível de serviço entregue e o nível de serviço contratado. Mecanismos e ferramentas de auditoria são propostos pelo NIST em [1] como alternativa para validar serviços, verificar o cumprimento de políticas e monitorar o uso dos dados. De acordo com a Força Tarefa para Gerenciamento Distribuído (DMTF, *Distributed Management Task Force*) em [10], o provedor de serviço de computação em nuvem deve registrar todas as operações executadas pelo cliente e todos os eventos ocorridos na entrega do serviço. O DMTF também especifica que é tarefa do provedor disponibilizar estas informações sob a forma e pelo tempo que foi estabelecido pelo SLA. Adicionalmente, o DMTF recomenda a utilização de mecanismos de preservação e segurança destas informações como armazenamento de escrita única e múltiplas leituras e assinatura digital.

Dentro de uma nuvem, a relação cliente-provedor pode ser estendida, uma vez que uma entidade pode ser cliente em determinada camada de serviço, como de IaaS, e fornecedora de outra camada de serviço, como em SaaS. Estas relações caracterizam uma cadeia de interdependência, conforme Haeberlen em [11]. Em [12] os provedores sub-contratados são citados como menos controláveis, uma vez que a contratação não foi estabelecida diretamente com estes.

Dentro do nosso trabalho, denominaremos “cliente” o usuário final de SaaS; “contratante” a entidade que é ao mesmo tempo fornecedor de SaaS e usuário de IaaS; e “provedor” o fornecedor de IaaS.

Em nuvem computacional as ações de todas as partes (provedor, contratante e do próprio cliente), são responsáveis pelo desempenho final dos serviços computacionais [4]. Desta forma, o monitoramento de SLA neste ambiente deve ser capaz de capturar informações sobre as partes para permitir a auditoria. Adicionalmente, além dos fatores internos, fatores externos à nuvem, como a conexão de rede, por exemplo, podem alterar a experiência do cliente com relação ao serviço provido.

A implantação de auditoria de SLA deve ser capaz, portanto, de monitorar as relações de fornecimento de serviços entre contratante e provedor, dentro da nuvem, e considerar a visão do cliente, externa à nuvem, sobre a entrega do serviço.

Na ocorrência de experiências negativas do cliente no consumo de serviços, deve ser possível identificar de maneira inquestionável para todas as partes se o problema está na aplicação do contratante, na conexão com o ambiente de nuvem computacional ou no provedor de infraestrutura. Porém, o contratante não tem acesso à infraestrutura do provedor, apenas acesso à máquina virtual (VM, *Virtual Machine*) oferecida pelo mesmo. Já o provedor não tem acesso à aplicação do contratante, apenas sabe para qual contratante a VM foi fornecida. Adicionalmente, nenhum dos dois tem acesso ao ambiente do cliente para ter a experiência de percepção da conexão reclamada pelo mesmo.

Neste contexto, são necessários pontos de monitoramento com coleta contínua de métricas adequadas à auditoria dos serviços e recursos. Porém, como as partes têm restrições de acesso umas as outras, será necessário que um terceiro (“inspetor”) faça a coleta destas informações anonimamente providas para evitar conflito de interesses deste terceiro com qualquer das partes, minimizando eventuais conflitos de interesses entre as mesmas, um auditor independente deve consolidar tais informações para identificar desvio de SLA, reportando-os ao pessoal de governança de TI do contratante.

1.2 Objetivos

Este trabalho tem como objetivo geral apresentar um novo mecanismo de auditoria para serviços em nuvem computacional através da implantação de um auditor externo independente (*Third-Party Auditor*, TPA), que seja capaz de auditar os indicadores de nível de serviço. A auditoria deve ser implantada através de inspetores em diferentes camadas de provimento de serviço (SaaS e IaaS). Os dados são posteriormente consolidados em outra camada, separando as tarefas de obtenção e avaliação destes. Esta solução deve operar *online* e ser imparcial (livre de conflitos de interesse), sendo que os resultados consolidados da auditoria devem ser armazenados externamente à nuvem computacional que está sendo auditada.

O mecanismo de auditoria deverá ser capaz de obter Indicadores de Nível de Serviço (SLIs, *Service Level Indicators*) das camadas de serviço de SaaS e obter métricas de recurso da camada IaaS, isolando e detectando problemas específicos, de modo a identificar os responsáveis por ações dentro da nuvem (*accountability*), permitindo a atribuição de responsabilidades por eventuais falhas. Não pertence ao escopo deste trabalho, acoplar mecanismos de dimensionamento e alocação de recursos em IaaS para atender demandas de SaaS eventualmente registradas pelo mecanismo de auditoria.

A arquitetura a ser desenvolvida deve ser facilmente acoplada às soluções existentes, não sendo composta, portanto, de uma nova proposta de especificação para as soluções em nuvem no mercado. Para obtenção das informações necessárias os agentes devem, sempre que possível, se utilizar de recursos das tecnologias existentes.

Desta forma, os objetivos específicos deste trabalho são:

a) Apresentar as premissas necessárias para que um serviço em nuvem seja auditável.
b) Definir a arquitetura de auditoria para serviços de nuvem.
c) Criar um mecanismo de inspeção que capture os indicadores de nível de serviço percebidos pelo cliente.
d) Implementar o mecanismo capaz de obter informações nas diferentes camadas de nuvem computacional (SaaS e IaaS).

e) Construir um protótipo que implemente a proposta e testá-lo sob os cenários distintos para avaliar sua eficácia.

1.3 Contribuições

A principal contribuição deste trabalho é apresentar um novo mecanismo para a realização de auditoria de SLA em nuvem computacional através de inspetores que intervenham minimamente no funcionamento da nuvem para desenvolvedores, clientes, contratantes e provedores. Ao confrontar as medições obtidas na nuvem, o auditor deve ser capaz de identificar, com eficácia, os pontos de falha e atribuir responsabilidades sobre estas, sejam internas ou externas à nuvem.

Ao medir o nível de serviço utilizando-se de medições coletadas no cliente do serviço, espera-se que o mecanismo torne a avaliação de SLA fidedigna à percepção do usuário final.

O processamento das medições é realizado por diferentes entidades dentro da arquitetura (inspetores), mas as medições só se tornam são unificadas fora da visão das entidades responsáveis pela sua coleta. Esta estratégia previne conflito de interesse, uma vez que as entidades que são residentes no ambiente onde as medições são realizadas não conseguem tem informações sobre a avaliação geral do serviço e tampouco podem ser manipuladas para alterar estas informações.

2 Fundamentação

Neste capítulo serão apresentados os fundamentos relevantes para a compreensão deste trabalho. Nos itens 2.1 a 2.4 são apresentados conceitos utilizados na definição dos objetivos deste trabalho, assim como na definição da arquitetura do protótipo. Os itens 2.5 a 2.9 referem-se a conceitos e tecnologias que serão utilizadas na fase de desenvolvimento do protótipo.

2.1 Tipos de nuvem computacional

Os serviços de computação em nuvem podem entregar serviços em diferentes formas. O NIST define três modelos de arquitetura de nuvem, de acordo com os recursos que são entregues para o cliente [7]:

1) **Software em Nuvem como um Serviço (*SaaS*).** O serviço oferecido ao cliente consiste em aplicações disponibilizadas pelo provedor rodando em uma infraestrutura em nuvem. As aplicações são acessíveis por vários dispositivos através de uma interface simples de cliente como um *browser web* (exemplo: *webmail*). O cliente não gerencia ou controla a infraestrutura adjacente na nuvem, incluindo rede, servidores, sistemas operacionais e armazenamento, nem mesmo as capacidades individuais da aplicação, com a exceção de alguns parâmetros de configuração específicos.
2) **Plataforma em Nuvem como um Serviço (*PaaS*).** O serviço oferecido ao cliente é a disponibilização de plataforma em nuvem para o desenvolvimento de aplicações usando linguagens de programação e ferramentas suportadas pelo provedor. O cliente não gerencia ou controla a infraestrutura adjacente na nuvem, incluindo rede, servidores, sistemas operacionais e armazenamento, mas tem controle sobre as aplicações implementadas e possivelmente sobre configurações do ambiente do

servidor. Em muitos casos, um cliente de PaaS pode ser um provedor de SaaS para usuários finais.

3) **Infraestrutura em Nuvem como um Serviço (*IaaS*).** A capacidade oferecida ao cliente é de provisionar recursos de processamento, infraestrutura física, armazenamento, redes e outros recursos computacionais fundamentais onde o cliente está apto a instalar e rodar os softwares que desejar, o que pode incluir sistemas operacionais e aplicações. O cliente não gerencia ou controla as camadas adjacentes da infraestrutura na nuvem, mas tem controle sobre o sistema operacional, armazenamento, aplicações desenvolvidas e possivelmente controle limitado de componentes específicos de rede (exemplo: *firewalls* no servidor). A Figura 2.1ilustra os recursos oferecidos por cada um dos modelos e ilustra a dependência dos serviços de SaaS e PaaS da camada de IaaS.

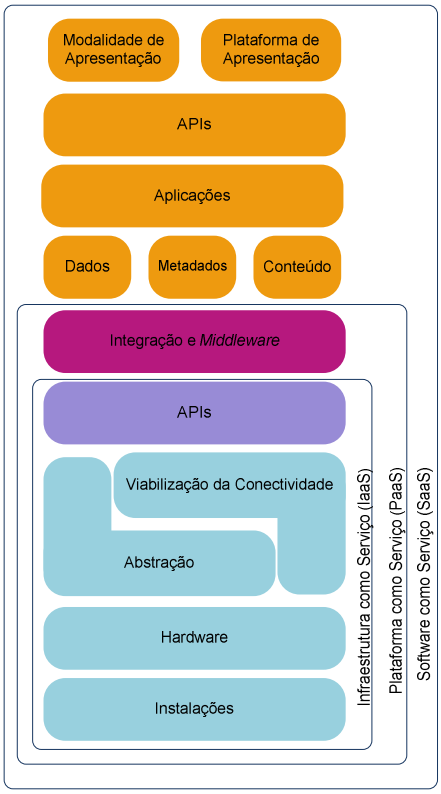

Figura 2.1 – Modelo de referência para computação em nuvem.
Adaptada de [16].

Os serviços de nuvem também podem ser categorizados de acordo com a propriedade e o uso compartilhado dos recursos entre clientes distintos:

- **Nuvem pública:** É a oferta de recursos computacionais e infraestrutura disponibilizados ao público em geral através da Internet. Por definição, os serviços são disponibilizados externamente à organização cliente, através de um provedor de serviços.
- **Nuvem privada:** Numa nuvem privada, o ambiente computacional é operado para a organização, podendo ser gerenciado exclusivamente por esta ou com auxílio de terceiros. O *datacenter* pode estar hospedado dentro da organização ou fora dela, mas, por ser um ambiente exclusivo e segregado, neste ambiente as organizações podem ter controle mais amplo sobre os recursos que em uma nuvem pública e estão menos sujeitos ao risco de ambientes multi-locatário.
- **Nuvem comunitária:** Se assemelha à nuvem privada, mas a infraestrutura e os recursos computacionais são compartilhados entre organizações que têm requisitos de privacidade, segurança e regulatórios comuns.
- **Nuvem híbrida:** É a composição de duas ou mais nuvens (privada, comunitária ou pública) que funcionam como entidades individuais, mas são disponibilizadas conjuntamente por tecnologia que permita interoperabilidade.

2.2 Conceitos Básicos de Segurança Computacional

Landwehr em [17] define segurança computacional como característica de um sistema que apresenta as propriedades abaixo:

- **Confidencialidade:** Garantia de que a informação será divulgada apenas para pessoas explicitamente autorizadas, podendo ser divulgada apenas a quem de direito. Muitas vezes confidencialidade é confundida com privacidade, onde a informação deve ser revelada a ninguém mais do que o dono da mesma. Confidencialidade pressupõe que ela possa ser tratada por um conjunto pré-definido de sujeitos.

- **Integridade:** Garantia de que a informação será protegida contra alterações não autorizadas, podendo ser alterada apenas por quem de direito.
- **Disponibilidade:** Garantia de que a informação estará acessível para o respectivo consumidor desta, pelo período necessário
    - Os sistemas podem ser classificados de acordo com a sua disponibilidade percentual durante determinado período, em geral dividindo-se o somatório dos períodos de indisponibilidade pelo período de um ano. Quanto mais próximo de 100%, mais controlado e tolerante a falha o sistema é considerado [18].
- **Autenticação:** Garantia de cada sujeito dentro de um sistema é realmente quem afirma ser.
- **Não-repúdio:** Garantia de que as ações realizadas dentro de um sistema não podem ser negadas posteriormente pelo seu executor.

As propriedades acima são suportadas por princípios de projeto no desenvolvimento de sistemas seguros:

- **Capacidade de atribuir responsabilidades por ações dentro de um sistema (*accountability*):** Capacidade de registrar e auditar as ações de cada sujeito dentro de um sistema.
- **Menor privilégio:** Atribuição apenas das mínimas permissões necessárias para o sujeito executar suas funções dentro do sistema.
- **Simplificação da infraestrutura de segurança:** Verificação de que todos os componentes, configurações e interações dentro de um perímetro de segurança foram corretamente especificados e implementados. Em ambientes menos complexos é mais simples e mais provável de se alcançar este princípio.
- **Segurança por padrão:** Implementação do princípio de menor privilégio em sistemas por *default*, ou seja, os sistemas devem ser pré-configurados de forma mais restritiva e segura possível. À medida que se torna necessário, configurações mais permissivas são adotadas.
- **Defesa em profundidade:** Assumindo-se que sistemas podem falhar, uma infraestrutura de segurança deve contar com componentes complementares,

de modo que determinado componente possa parar um ataque sofrido por outro componente vulnerável.

2.3 Acordo de Nível de Serviço

Conforme definição pela RFC2475 [19], o Acordo de Nível de Serviço (SLA, *Service Level Agreement*) é o resultado decorrente da negociação entre cliente e provedor de serviço que especifica atributos dos serviços providos como, por exemplo, disponibilidade e desempenho. De acordo com o NIST em [1], um SLA deve conter um conjunto de premissas a serem atendidas pelo provedor quanto ao fornecimento do serviço para o cliente; um conjunto de itens explicitamente não cobertos no provimento dos serviços; e as obrigações dos clientes.

Um SLA pode especificar as necessidades e níveis a serem obtidos no fornecimento do serviço, com alto nível de abstração. Os serviços são avaliados através de métricas de desempenho, conhecidas como Indicadores de Nível de Serviço (SLIs, *Service Level Indicators*). A junção das métricas de desempenho (SLI) com o seu valor objetivo é denominado Objetivo de Nível de Serviço (SLO, *Service Level Objective*).

“Disponibilidade” e “tempo médio de resposta do serviço” são exemplos de SLIs utilizados em um SLA; já os valores “95%” de disponibilidade ou “1,3 segundos” de tempo de resposta são exemplos de valores de um SLO [20].

Um SLO deve ser especificado de acordo com a expectativa do usuário sobre a entrega dos serviços. Antes de ser definido e negociado, é necessário que sejam conhecidos eventuais impactos no negócio no caso de falha do serviço [20].

Os SLIs e SLOs constituem a parte técnica mais relevante de um SLA; entretanto, existem outros itens que devem ser descritos em sua composição tais como penalidades a serem pagas pelo provedor do serviço quando este falhar em cumprir o nível de serviço prometido e recompensas que serão pagas pelo cliente do serviço quando o provedor do serviço superar as expectativas do nível de serviço acordado [20].

Para que seja possível definir os recursos necessários para uma aplicação atender aos SLOs estabelecidos no SLA, é necessário que a infraestrutura que irá hospedar a aplicação seja dimensionada. Cada recurso da infraestrutura pode ser

monitorado através de métricas, denominadas métricas de baixo nível ou métricas de recursos. Como exemplo, para que se cumpra um SLO “tempo de resposta < 3 segundos”, são necessários recursos de infraestrutura como processador, banda de rede e memória, que são dimensionados de acordo com a aplicação em questão. Cada recurso possui atributos, como utilização do processador, largura de banda e memória disponível. Cada atributo é definido como métrica de recurso. Exemplos de métricas de recurso são percentual de memória disponível, percentual de consumo de áreas de *cache*, fila de processos prontos para execução (*run queue*), quantidade de processos aguardando I/O, percentual de utilização de CPU, etc. Um conjunto de métricas de recurso reunidas para especificar requisitos de um serviço é denominado parâmetros de Qualidade de Serviço. Definições complementares sobre Qualidade de Serviço serão exploradas no item 2.4.

Tradicionalmente os recursos necessários para atender um SLA são definidos por especialistas, que utilizam sua experiência com aplicações anteriores para dimensionar a infraestrutura necessária. Em alguns casos, as aplicações podem ser testadas através de *benchmarking*, uma análise comparativa que possibilita a simulação do consumo de recursos necessários.

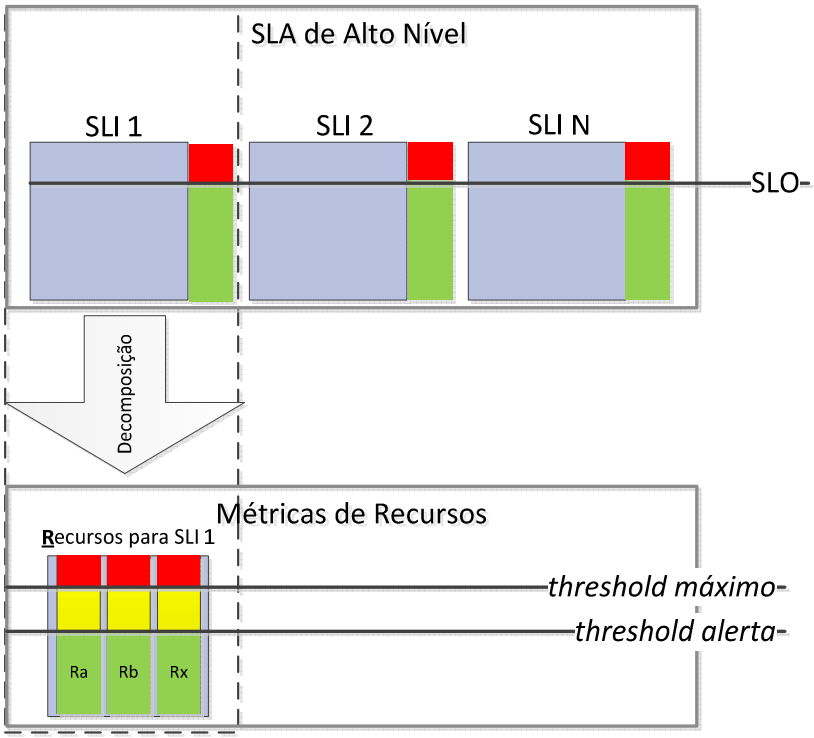

Figura 2.2 – Decomposição de SLI/SLO em métricas de recursos/*thresholds*. Adaptada de [20].

As métricas de recursos devem ter limites (*thresholds*) estipulados para manutenção de cada SLO. É possível definir *thresholds* de alerta e máximo para cada métrica de recurso como forma de monitorar ou prevenir potenciais desvios do SLA acordado. A Figura 2.2 mostra o SLI sendo suportado pelos recursos e suas respectivas métricas.

2.4 Qualidade de Serviço

Qualidade de serviço é um termo utilizado na composição de SLAs de serviços computacionais por especificar claramente as métricas de recursos e seus valores objetivos. Em [21] discute-se a criação de uma ontologia padrão para ser utilizada nos três eventos comuns em a toda contratação de serviços na Web: a aquisição de serviços baseado em QoS, monitoramento de QoS e adaptação de QoS.

Na aquisição de serviços baseada em QoS, utilizando-se as mesmas definições e métricas de QoS para todos os serviços e provedores, torna-se possível a comparação entre os requerimentos do usuário em tempo real e o QoS ofertado pelos diferentes provedores.

No monitoramento de QoS de serviços contratados, as métricas de QoS são utilizadas para monitorar a Qualidade de Serviço efetivamente entregue e detectar uma eventual violação de SLA. As medições coletadas são comparadas aos acordos medidos no SLA.

Na implantação de QoS adaptativa são coletadas informações em tempo real sobre QoS do serviço consumido. Determinados parâmetros de QoS são controlados e podem disparar reações a uma possível degradação da qualidade, automaticamente solicitando mais recursos de infraestrutura ou reduzindo o número de conexões ao serviço.

De acordo com a ontologia proposta, todo item de QoS deve possuir:

- **Característica ou atributo:** Uma categoria de serviço deve estar relacionada a um conjunto de características que afetam sua qualidade. A característica deve ser um atributo mensurável de um serviço.

  Exemplo: Tempo de processamento, % de CPU utilizada.

- **Valor ou medição:** O resultado da medição de uma característica.
  Exemplo: tempo de processamento para “Tarefa A” = 2500 mSeg
- **Medição:** Especificação do método utilizado para medir uma característica. Exemplo: Tempo de processamento da Tarefa A é obtido pelo intervalo entre o evento de pressionar o botão “Salvar” e retorno do comando “*Comitt*” no banco de dados.
- **Função:** Especificação das combinações de medições para descrever critérios de QoS.
- **Perfil de QoS:** Conjunto de valores métricos para um serviço, compostos por nível de referência e nível das medições.
- **Caracterização da medição:** Especificação de onde, como (utilizando quais funções) e quando (início/fim e intervalos de amostragem) a medição é obtida.
- **Parte envolvida:** As medidas podem ser obtidas pelo provedor, cliente ou uma terceira parte no serviço.
- **Tipo de valores:** As medições devem ser especificadas em tipos de valores aceitos, podendo ser, por exemplo, inteiros, *strings* e %, entre outros, expressos por unidades de medidas padrão.

2.5 Java *Servlets*

Servlets são classes Java desenvolvidas de acordo com uma estrutura bem definida, e que, quando instaladas junto a um Servidor que implemente um *servlet container* (um servidor que permite a execução de *servlets*, muitas vezes chamado de Servidor de Aplicações Java com suporte a *servlets*), podem tratar requisições recebidas de clientes.

Servlets em geral são desenvolvidos para responder às requisições HTTP. Um cenário típico de funcionamento de uma aplicação desenvolvida com *servlets* é o seguinte: ao receber uma requisição, um *servlet* pode capturar parâmetros desta requisição (muitas vezes transmitidos num formulário HTML), efetuar qualquer processamento inerente a uma classe Java e devolver uma página HTML de

resposta. *Servlets* rodam dentro da máquina virtual do servidor. Cada requisição inicia uma nova *thread* (processo) dentro da máquina virtual. A Figura 2.3 representa o mapeamento entre requisições, *threads* e *servlets* dentro de um servidor de aplicações Java, demonstrando que para cada requisição uma nova *thread* é iniciada instanciando o *servlet* invocado.

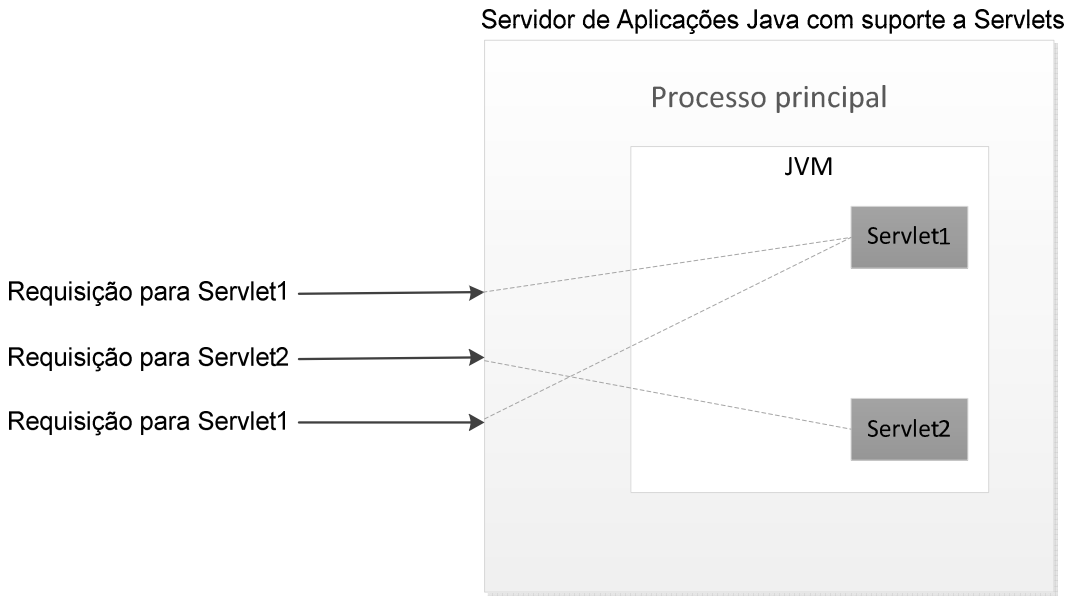

Figura 2.3 – Mapeamento entre requisições, *threads* e *servlets*. Adaptada de anônimo.

2.6 Java *Applets*

Applets são pequenos programas que podem ser adicionados a páginas *web*, tornando-se uma parte desta página *web*, como uma imagem ou uma linha de texto. Da mesma forma que um *browser* interpreta uma referência de uma imagem em um documento HTML, o *browser* identifica o *applet* Java e o encaminha para execução pela JVM (*Java Virtual Machine*). Quando o *browser* carrega o documento HTML, o *applet* Java é também carregado e executado.

Por questões de segurança não é permitido que o *applet* seja executado até que ele tenha confirmação da verificação de seu *byte-code*. Dependendo da política de segurança, um *applet* assinado pode acessar diretamente a máquina virtual Java

(JVM). Quando não for identificado como confiável, fica sem acesso às ações diretamente no sistema operacional, como tentar acesso a uma porção de memória. O ambiente protegido onde é executado chama-se *sandbox* (caixa de areia).

Um *applet* é essencialmente composto por quatro métodos, descritos abaixo:

- ***Applet.init*():** Quando um *applet* é carregado pela primeira vez para o sistema, o método init() é chamado pelo ambiente de execução (*browser* ou *appletviewer*) para inicializar o *applet*. Esta função é chamada apenas uma vez, antes que o *applet* se torne visível, servindo como um método de configuração para inicializar variáveis e outros objetos.
- ***Applet.start*():** Este método é chamado pelo *browser* sempre que o *applet* torna-se ativo. O *applet* pode ser ativado logo após a inicialização, ou quando uma página HTML que contém a *tag applet* é exibida novamente. Os *applets* que possuem códigos para o processamento *multi-threaded* (vários processos) utilizarão este método para criar ou reiniciar a execução de tais *threads*.
- ***Applet.stop*():** Quando o usuário sai da página que continha o *applet*, o *browser* efetua a chamada a este método. *Threads* e conexões a sistemas através da rede são geralmente suspendidas quando este método é acionado.
- ***Applet.destroy*():** Este método é chamado pelo *browser* a todo momento que um *applet* necessita ser descartado. Uma chamada ao método *destroy*() é seguida pelo despejo da instância do *applet* da memória, portanto este método é utilizado principalmente para o propósito de "limpeza" da memória.

2.7 Eucalyptus

Eucalyptus é um *framework* de código aberto para computação em nuvem que implementa serviços em nuvem sob a forma de *Infrastructure as a Service* (IaaS) [22].

A arquitetura do Eucalyptus é composta por quatro componentes, cujo diagrama esquemático é apresentado na Figura 2.4 :

- ***Node Controller*:** Componente responsável por controlar a execução de cada instância de máquina virtual (VM, *Virtual Machine*). O *Node Controller* geralmente é executado no servidor físico onde são instanciadas as máquinas virtuais.
- ***Cluster Controler*:** Gerencia a rede virtual entre diversas VMs e é capaz de obter informações ou agendar a execução destas.
- ***Storage Controller*:** Implementa o serviço de armazenamento virtual.

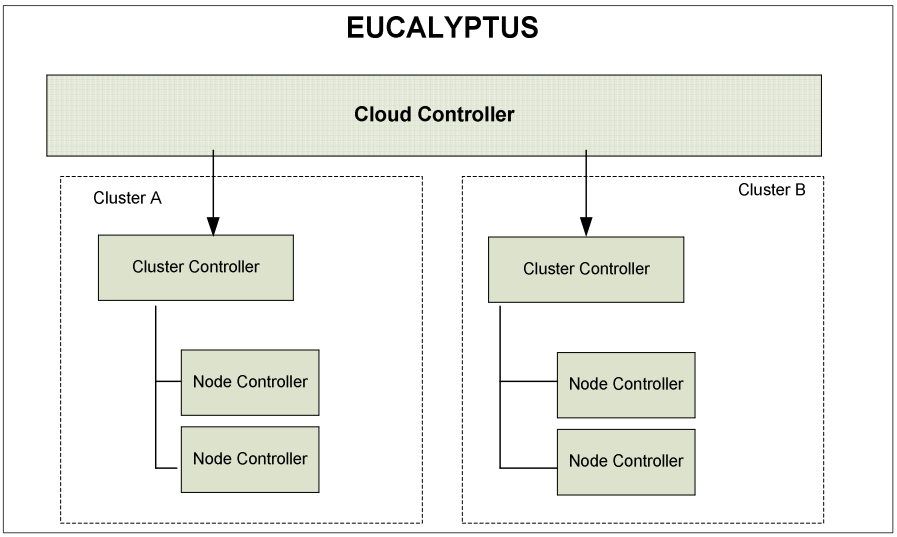

Figura 2.4 – Relação entre os serviços de *Cloud Controller*, *Cluster Controller* e *Node Controller*. Adaptada de [22].

- ***Cloud Controller* (CLC):** É o serviço responsável por gerenciar todos os recursos da nuvem, sendo capaz de alocar/desalocar recursos e obter informações sobre os serviços existentes. Ele também apresenta a interface de gerenciamento para os usuários da nuvem.

O CLC é uma coleção de *web services*. Eles são agrupados em serviços gerenciadores de recursos, serviços de dados (gerencia parâmetros de configuração do usuário e sistema) e serviços de interface (expõe a interface para o usuário e os serviços de gerenciamento).

O Serviço de Recursos (RS, *Resource Service*) é quem interage com o *Cluster Controller* e o *Node Controller* para verificar o estado de VMs e alocar/desalocar recursos físicos. Todo o registro de alocação de recursos é registrado em uma entidade lógica, o Estado de Recursos do Sistema (SRS, *System's Resource State*). O SRS pode fornecer informações importantes para

controle de SLAs, embora nem sempre represente acuradamente os recursos alocados.

O Eucalyptus utiliza a mesma interface provida pela *Amazon Elastic Cloud Computing* (Amazon EC2) para acesso programável aos recursos computacionais através de mensagens SOAP e *Query*. É o Serviço de Interface (*Interface Service*), que atende as requisições de usuários, sendo responsável por responder as consultas. São trocadas mensagens SOAP entre as interfaces programáveis e os *webservices*.

2.8 Virtualização

"Virtualização de servidores" significa abstrair os recursos físicos de um servidor, de modo a parecerem diferentes logicamente do que são fisicamente, em geral representando uma fração do hardware real. Tradicionalmente é utilizada como sinônimo da tecnologia de máquinas virtuais por ser a forma de virtualização mais comum na plataforma x86 de servidores.

Num ambiente virtualizado, a camada de software abstrai o hardware físico e é capaz de instanciar várias máquinas virtuais, cada uma com seus processos, sistema operacional e aplicações. Esta camada de abstração pode também conter funções de gerenciamento, sendo denominada *hypervisor* (hipervisor). É o *hypervisor* que disponibiliza a abstração de recursos de hardware para os sistemas operacionais.

Um sistema operacional (SO) que é executado em uma instância de máquina virtual é chamado de SO hóspede (*guest*). As máquinas virtuais instanciadas no mesmo hardware compartilham os mesmos recursos. Esse compartilhamento de recursos pode significar um melhor aproveitamento do hardware, uma vez que os servidores oscilam quanto ao consumo de recursos, apresentando picos de uso e ociosidade em outros momentos. Na Figura 2.5 está representado um diagrama que ilustra a relação entre hardware, *hypervisor* e diferentes sistemas operacionais.

Nos ambientes x86, existem variações dentro da camada de software cuja abstração pode ser classificada desta forma:

- **Tradução binária da CPU:** O *hypervisor* intercepta determinadas instruções para a CPU e realiza traduções de modo que o SO hóspede acredita ter completo controle sobre o hardware do servidor.

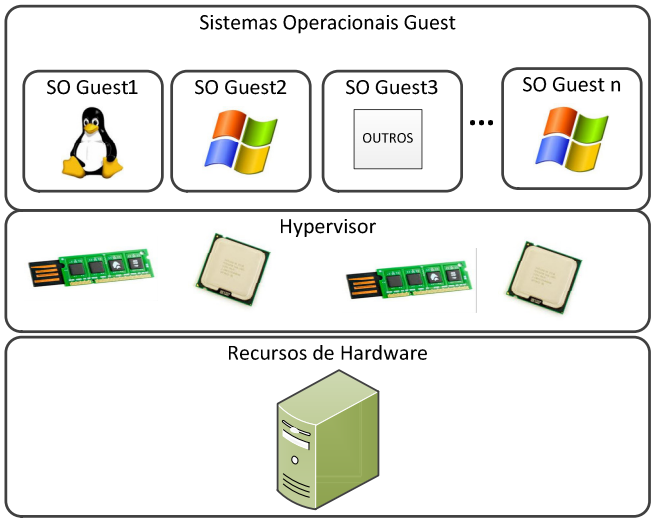

Figura 2.5 – Abstração criada pelo *hypervisor* para os sistemas operacionais hospedeiros. Adaptada de anônimo.

- **Abstração na camada de aplicação:** A camada de software de abstração reside em uma aplicação em execução no SO hospedeiro. Esta camada serve para fazer uma tradução binária para outras aplicações e não necessita executar outro SO hóspede. Sua utilização permite que aplicações desenvolvidas para determinado SO possam ser executadas em outro SO.
- **Virtualização suportada pelo hardware:** Novos processadores como o AMDV e o Intel VTx trazem tecnologias embarcadas estendendo o conjunto de instruções x86, de modo a auxiliar o trabalho do *hypervisor* e contribuir na distribuição da carga. Estas novas instruções permitem que alguns controles de supervisão possam acontecer na camada de hardware de CPU, tornando o trabalho do *hypervisor* mais eficiente.
- **Paravirtualização:** O SO hóspede é modificado para utilizar *drivers* paravirtualizados para I/O. O SO hóspede modificado passa a chamar diretamente os serviços de I/O e outras operações em modo privilegiado, ao invés de ter todas as instruções privilegiadas interceptadas e traduzidas pelo

hypervisor. Pela adição dos *drivers* paravirtualizados, o SO hóspede passa a estar ciente da presença do *hypervisor*.

2.9 *Hypervisor* Xen

Xen é um *hypervisor* que implementa abstração do hardware para o SO através de paravirtualização [23]. O ambiente virtual Xen é composto pelos módulos descritos abaixo, representados na Figura 2.6:

- Xen Hypervisor
- SO Guest – Domínio 0 (Domain 0)
    - No Domínio 0 fica hospedado o serviço de gerenciamento e controle dos demais SO hóspedes, denominado *Domain Management and Control* (Xen DM&C).
- SO Guest – Domínio U (Domain U)
    - PV Guest: Cada SO PV hóspede é um SO hóspede paravirtualizado (*Paravirtualizated Guest*).
    - HVM Guest: Um SO HVM Guest (*Host Virtual Machine Guest*) é um SO plenamente virtualizado

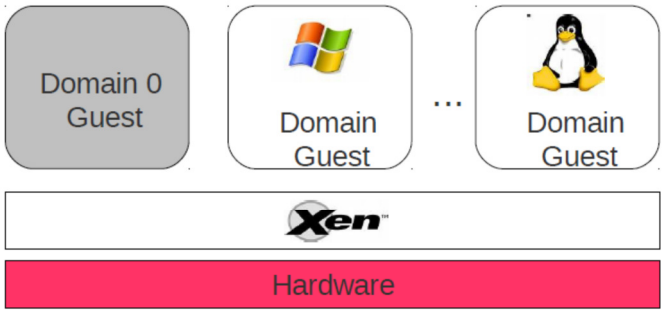

Figura 2.6 – Módulos que compõem o Xen. Adaptada de anônimo.

O *hypervisor* Xen é a camada de abstração básica de software que é instalada diretamente sobre o hardware e fica sob os SOs. É responsável por escalonamento da CPU, e pelo endereçamento e particionamento de memória das várias máquinas virtuais instanciadas pelo *hypervisor*. O *hypervisor* não só abstrai o

hardware para as máquinas virtuais, mas também controla a utilização do processamento pelas máquinas virtuais.

No Domínio 0 fica hospedada a VM em execução no *hypervisor* Xen. Esta VM tem direitos especiais para acesso direto a recursos de I/O e é capaz de interagir com as outras máquinas virtuais (pertencentes ao Domínio U sejam do tipo PV ou HVM Guests) em execução no sistema. Antes de qualquer outra máquina virtual ser iniciada, todos os ambientes de virtualização Xen requerem a execução do Domínio 0. Os *drivers Network Backend Driver* e *Block Backend Driver* estão inseridos no Domínio 0 para suportar as solicitações de acesso de rede e disco local oriundas das máquinas no Domínio U.

As VMs hospedadas no Domínio U não podem ter acesso direto ao hardware do computador hospedeiro, ao contrário do Domínio 0, onde este acesso é permitido. Todas as VMs paravirtualizadas rodando no *hypervisor* Xen são referenciadas como Domínio U PV Guests, podendo ser Linux, Solaris, FreeBSD, ou outro Unix (em todos estes casos, necessariamente com modificações nas versões no sistema operacional instalados). O *hypervisor* Xen também pode hospedar VMs totalmente virtualizadas, permitindo que os sistemas operacionais em execução nestas VMs não precisem ser modificados, rodando em sua versão padrão, como no caso de versões de Windows. As VMs que são executadas no modo de virtualização total são identificadas como Domínio U HVM *Guests*.

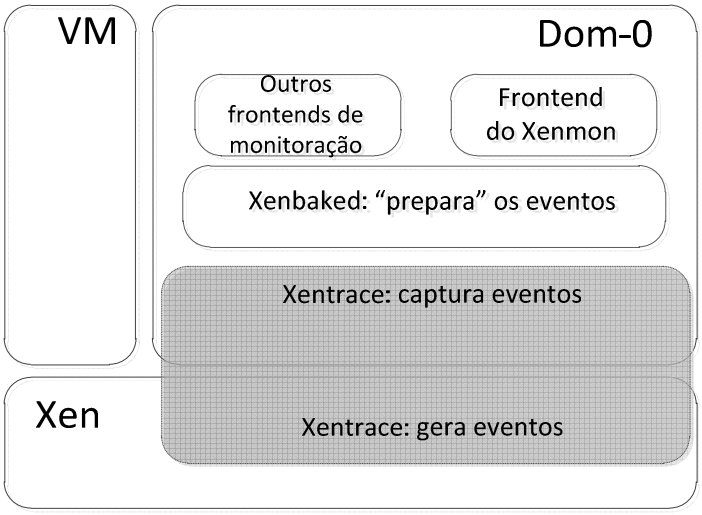

Figura 2.7 – Componentes do XenMon.
Adaptada de [23].

As VMs hospedadas no Domínio U PV estão cientes da limitação de acesso direto ao hardware e têm ciência que existem outras VMs em execução no mesmo computador. Já as VMs hospedadas no Domínio U HVM não estão cientes da virtualização e enxergam o hardware entregue pelo *hypervisor* como o hardware físico real. Uma VM no Domínio U PV utiliza dois *drivers* para rede e acesso a disco, o PV *Network Driver* e PV *Block Driver*.

As VMs hospedadas no *Domain* U HVM não possuem PV *drivers* dentro de suas instâncias; neste caso um processo especial (qemu-dm) é iniciado para cada VM no Domínio 0. O processo qemu-dm dá suporte as VMs hospedadas no Domínio U HVM para rede e acesso a disco, abstraindo destas VMs o acesso ao hardware. Estas VMs inicializam da mesma forma que inicializariam num computador típico. Para isso, é adicionado o Xen virtual *firmware* em cada VM, possibilitando a simulação do funcionamento de uma BIOS.

A interface de monitoramento implementada para o Xen é o XenMon, que reporta o uso de recursos de diferentes VMs e provê informações adicionais sobre o compartilhamento e o escalonamento de recursos no Xen. O XenMon é essencialmente composto por três componentes, demonstrados na Figura 2.7 e detalhados abaixo:

- **Xentrace:** O modulo é um gerador de eventos, presente no Xen. Usando o Xentrace é possível disparar eventos em pontos de controles arbitrariamente definidos no *hypervisor*. Xentrace permite que atributos específicos sejam associados com cada evento. Por exemplo, para um agendamento específico para um evento num domínio, os eventos podem ser associados a atributos como o ID do próprio domínio e o *timestamp* do evento.
- **Xenbaked:** O *event stream* gerado pelo Xentrace não é útil sem o acoplamento a outros serviços. O Xenbaked é um processo em modo usuário que captura os eventos gerados pelo Xentrace e os processa, possibilitando a interpretação das informações. Pode-se confrontar e agregar eventos de alterações de *status* de domínios, de “dormindo” para “acordado”, por exemplo, com o intuito de determinar o intervalo de tempo no qual o domínio estava bloqueado.

- **Xenmon:** É a interface para exibir e registrar os dados. O XenMon reporta várias métricas que são acumuladas sobre período de execução (intervalo de tempo para o qual o domínio estava agendado para usar a CPU); domínio que estava utilizando a CPU no último segundo; e domínio que estava usando a CPU nos últimos 10 segundos.

Há um conjunto de métricas disponíveis para medir o consumo de tempo por domínio:

- **Uso de CPU (*usage time*):** mostra o percentual de tempo da CPU utilizado por um domínio conforme medida padrão (por exemplo, sobre 1 segundo).
- **Tempo bloqueado (*blocked time*):** reflete um percentual de tempo gasto por um domínio enquanto estava bloqueado em algum evento de I/O (isto é, o domínio não estava na *run queue*).
- **Tempo em espera (*waiting time*):** mostra o percentual de tempo de um domínio gasto na *run queue* para ser escalonado no uso da CPU.

As três métricas descritas acima representam completamente como o tempo é gasto por um domínio. Adicionalmente, o XenMon reporta a métrica contador de execução (*execution count)* que reflete em qual frequência um domínio foi escalonado para a CPU durante um intervalo de mensuração (por exemplo, 1 segundo). Quando a leitura desta métrica é combinada com as métricas de uso de CPU e tempo em espera por período, obtemos informações importantes sobre o comportamento do escalonador.

Para aplicações com I/O intensivo, o XenMon prove a métrica de contador de I/O (*I/O count)* que é uma medida aproximada de I/O solicitado pelo domínio. É o número de trocas de página de memória entre um domínio e Domínio 0, uma vez que o acesso a disco e memória é abstraído pelo Domínio 0 disponibilizando as páginas de memória solicitadas pelo SO hospedeiro nos outros domínios.

3 Trabalhos Relacionados

3.1 Auditoria de Acordo de Nível de Serviço em *Grid Computing*

O artigo [24] avalia arquiteturas para auditoria de SLA, considerando desempenho, introduzindo a questão de relação de confiança entre as partes envolvidas numa auditoria de SLA e os impactos da implantação de uma arquitetura de auditoria numa infraestrutura existente.

Contratos de SLA declaram as expectativas do cliente quanto ao provedor, cuja relação inversa também pode ser prevista no mesmo acordo. Cada SLA é composto por um conjunto de indicadores de nível de serviço que possuem valores alvo para o serviço contratado. Cada SLI associado ao seu valor alvo constitui um SLO.

A relação de confiança num ambiente de *grid computing* considera que provedor e usuário não necessariamente possuem uma relação de confiança entre si, e que o cliente pode não confiar nos dados de auditoria de SLA fornecidos pelo provedor e vice-versa. Adicionalmente, é comum haver percepções diferentes entre níveis de serviço pelo fornecedor e pelo cliente, sendo causadas por fatores externos à relação cliente/provedor, como por exemplo, aquelas motivadas por latência de rede. Não obstante, instrumentais diferentes para medição de SLA podem gerar diferentes resultados.

Assumindo a necessidade de uma visão imparcial, o artigo propõe a participação de um TPA. O TPA deve ser uma entidade confiável tanto para o cliente como para o provedor e deve ser responsável pela instrumentação e execução da auditoria em si.

- **Arquitetura simplificada:** Numa primeira proposta, o auditor é introduzido consumindo recursos do provedor, com o intuito de simular a ação do cliente. A desvantagem óbvia neste cenário é a geração de *overhead*. Outra desvantagem é a possibilidade que o inspetor seja privilegiado no fornecimento do serviço, impedindo uma visão real do mesmo, ainda que especificamente esta desvantagem pudesse ser contornada através de

anonimização de requisições.. Um diagrama esquemático desta arquitetura é apresentado na Figura 3.1.

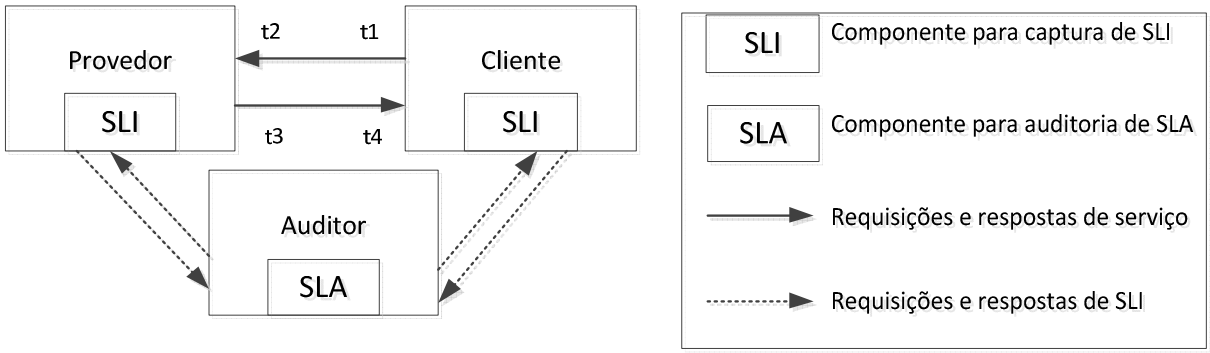

Figura 3.1 – Arquitetura simplificada de auditoria de SLA, com a participação do TPA. Adaptada de [24].

- **Arquitetura de *sniffing* de pacotes:** Nesta arquitetura, o inspetor (serviço adicional responsável pela coleta de SLIs para envio para consolidação pelo auditor) precisa estar conectado nas redes locais do cliente e provedor, através do espelhamento de portas (*port mirroring*) do *switch* onde os mesmos estão conectados. A vantagem é a inexistência de *overhead*, uma vez que não são adicionadas novas requisições ao serviço provido, e o fato de que nesta arquitetura torna-se possível medir SLIs tanto de cliente como de fornecedores. Para que não sejam violados, é necessário utilizar *sniffers* resistentes à violação ou adicionar um coprocessador seguro para criptografar e assinar digitalmente a informação coletada que é enviada ao inspetor. A principal desvantagem do modelo é que o auditor teria acesso potencial a todas as informações em circulação nas redes locais do provedor e do cliente, uma vez que o processo de captura de pacotes não é seletivo e captura toda informação que a transmitida para o *host* conectado naquela porta. Outro ponto negativo é que o *sniffer* não seria capaz de capturar informações contidas em pacotes criptografados. Um diagrama representativo desta arquitetura é apresentado na Figura 3.2.

- **Arquitetura de *host decorator*:** Nesta arquitetura os inspetores estão implantados nos *hosts* do provedor e do cliente e participam da comunicação em nível de serviço, intermediando todas as interações cliente-servidor. Cada

mensagem partindo do cliente é transmitida ao inspetor instalado no respectivo *host*, que interage com o inspetor residente no *host* do provedor, que por sua vez encaminha a mensagem para o provedor. Mensagens do provedor para o cliente realizam caminho inverso, conforme apresentado na Figura 3.3. Esta proposta sobrepõe a limitação da incapacidade de detecção por *sniffing*, pois irá inspecionar somente os pacotes de interesse da aplicação, uma vez que os auditores estão posicionados em nível de aplicação. A contrapartida é que dados das transações entre clientes e provedores ficariam expostos neste modelo, o que poderia ser mitigado pela criptografia dos dados enviados para o provedor utilizando sua chave pública.

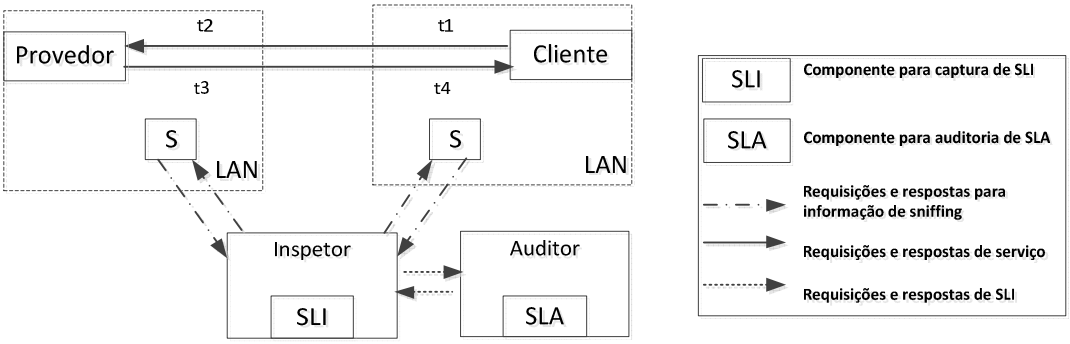

Figura 3.2 – Arquitetura de auditoria de SLA utilizando *sniffing* de pacotes em rede local. Adaptada de [24]

Após avaliação das arquiteturas, o artigo mede o impacto causado pela inserção dos serviços de auditoria de SLA no desempenho do serviço, através do Erro de Interferência de Medida (*Measurement Interference Error*, MIE), e compara o desempenho das diferentes propostas na arquitetura de *grid computing*.

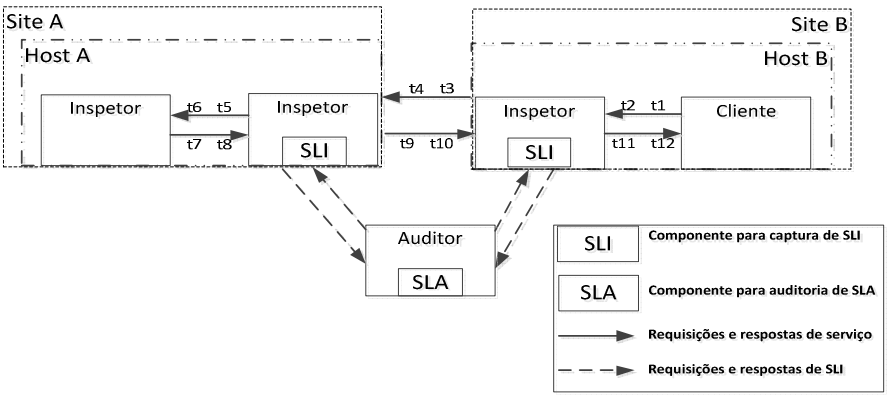

Figura 3.3 – Arquitetura de auditoria de SLA utilizando agentes Inspetores e Auditor. Adaptada de [24]

A visão comparativa de desempenho nesta arquitetura não foi transcrita por não ser relevante para o escopo deste trabalho. Este artigo é uma referência para esta dissertação por introduzir a presença do auditor terceiro no provimento de serviços em computação em *grid*, que é um sistema distribuído predecessor à computação em nuvem.

3.2 Qualidade da Experiência: Obtendo a Qualidade de Serviço percebida pelo usuário

O fornecimento de serviços computacionais utiliza-se de métricas de QoS para a definição de expectativas e formulação do SLA.

A definição de Qualidade da Experiência (QoE, *Quality of Experience*) é explorada no artigo [25]. É considerada complementar à definição de QoS, uma vez que QoE representa o conjunto de métricas utilizadas para se obter a percepção do cliente sobre a Qualidade do Serviço consumido. Os dois conceitos diferenciam–se essencialmente porque as medições de QoE podem ser influenciadas por qualquer sistema entre o provedor e o usuário final, pois o QoS avalia unicamente o serviço tal como é entregue pelo provedor. Adicionalmente, as métricas de QoE podem possuir elementos subjetivos que não serão discutidos no escopo deste trabalho. A Tabela 3.1 exemplifica as métricas mais comuns para QoS e QoE.

Tabela 3.1 – Métricas de exemplo de QoS e QoE

QoS		QoE
Sistema	**Tarefa**	
Capacidade de processamento	Tempo de processamento da tarefa	Tempo de resposta fim-a-fim
Memória total disponível **Capacidade de *throughput*** **Disponibilidade total**	Tempo de espera (para início do processamento)	Acesso bem sucedido (disponibilidade percebida)

Outros artigos, como [26] e [27], exploram as relações quantitativas entre QoS e QoE. Em [27] define-se a relação QoS e QoE, comparando desvios de QoS e o aspecto subjetivo de QoE. Também em [26], testou-se a sensibilidade do usuário à latência de rede.

A Figura 3.4 foi adaptada de [26], revelando que as taxas de cancelamento por parte do usuário aumentaram significantemente após 3 segundos para *download* do último byte na interface do usuário.

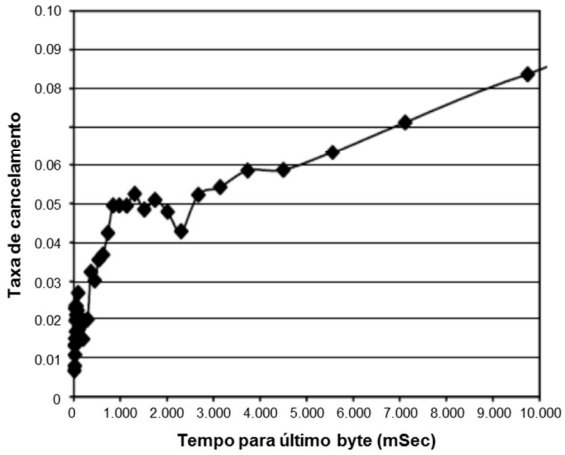

Figura 3.4 – Relação entre tempo para recebimento do último byte pelo cliente e % de cancelamento de transações. Adaptada de [26]

Neste trabalho será utilizado o SLI de tempo máximo de resposta fim-a-fim como 03 segundos, parâmetro escolhido observando-se as experiências sumarizadas no gráfico da Figura 3.4. Despreza-se o tempo de processamento necessário ao navegador para exibição da resposta após o recebimento do último byte. O protótipo desenvolvido neste trabalho obterá medidas de QoE exemplificadas neste artigo.

3.3 Capacidade de Monitoramento (Monitorabilidade) de SLA como requisito para aquisição de serviços em computação em nuvem

Em cenários heterogêneos e fracamente acoplados, o gerenciamento de SLA requer mecanismos de monitoramento avançados para garantir que as violações de SLA durante a provisão do serviço sejam registradas. O artigo [28] propõe que a

negociação de SLA deva incluir obrigatoriamente as condições de monitoramento deste, avaliando-as antes do estabelecimento do acordo. O artigo propõe um *framework*, SLA@SOI, capaz de implantar estas características.

Um cenário genérico decompõe o SLA em serviços combinados (CS, *Composite Services*) compostos de serviços básicos (AS, *Atomic Service*). CSs são implantados como coleções de ASs, que por sua vez são implantados em Serviços de Infraestrutura (ISs, *Infrastructure Services*). Desta forma, o SLA determinado na verdade é a combinação hierárquica dos SLAs do serviço até o hardware. Todo o conjunto de SLAs precisa ser garantido para assegurar a experiência do usuário. Para isso, são utilizados termos de acordo, instrumentos para evidenciar a garantia entre os serviços negociados e monitorados.

Para o apuramento dos SLAs são necessárias definições formais de qualidade de serviço (*Quality of Service*, QoS), descritas abaixo:

- **Disponibilidade:** Assumindo serviço S; instante T_1 como início do período de monitoramento; instante T_2 como período para analisar disponibilidade; duração do monitoramento $T = T_2-T_1$; b_i como o período em que os clientes não podem mais invocar S (exceto quando há problemas de conectividade), onde $T_1 \leq b_i \leq T_2$; e_i como o momento em que S torna-se novamente utilizável, após a ocorrência de b_i, onde $T_1 \leq e_i \leq T_2$; $d_i = e_i-b_i$; $d = \Sigma d_i$; nós então definimos disponibilidade do serviço S como $A = (T-d)/T$.
- **Acessibilidade:** Assumindo a operação O do serviço S; instante T_1 como início do período de monitoramento; instante T_2 como período para analisar acessibilidade; a duração do monitoramento como $T = T_2-T_1$; R_a como a quantidade de invocações de O durante o intervalo T; R_d como o número de invocações que não foram respondidas, ou seja, que foram descartadas durante o período T; nós então definimos acessibilidade para operação O como $C_O = (R_a-R_d)/R_a$.
- ***Throughput*:** Assumindo a operação O do serviço S; unidade de tempo t; taxa de requisições entrantes R = N/t, N = Número de requisições por unidade de tempo t, $N \in \mathbf{I}$; acessibilidade C = 1 para $R = R_1$; acessibilidade C<1 para $R = R_2$, $R_2 > R_1$, nós então definimos *throughput* para operação O como $H_O = R_1/t$.

- **Tempo de Conclusão e Tempo Médio de Conclusão (*Average Completion Time*):** Assumindo operação O do serviço S; mensagem de requisição M_Q do cliente do serviço S para invocação da operação O; mensagem de resposta M_R; M_Q recebida por completo pelo serviço no tempo t_I; M_R é devolvida à rede completamente no tempo t_O; definimos então Tempo de Conclusão da operação O como $TC_O = t_O - t_I$. Assumindo várias medições contínuas através do monitoramento da infraestrutura, TC_{O1}, TC_{On}, definimos Tempo Médio de Conclusão como $TA_O = (\Sigma TC_{Oi})/n$.
- **Tempo Médio para Reparo (*Mean Time To Repair*):** Assumindo o serviço S, no tempo um momento de tempo t_b, no qual o serviço se torna indisponível; o respectivo momento no tempo t_e, que ele se torna disponível novamente o período (duração) da indisponibilidade, $t = t_e - t_b$; uma série de períodos, $T = (t_1, t_2, \ldots, t_n)$ capturados através do monitoramento da infraestrutura; o total de tempo de indisponibilidade, $u = \Sigma t_i$; definimos MTTR = u/n.
- **Tempo Médio entre Falhas (*Mean Time To Failure*):** Assumindo o serviço S; restauração do serviço após falha, ocorrendo no tempo t_b; uma consecutiva falha no serviço, iniciando ao tempo t_e; o respectivo tempo de disponibilidade $t = t_e - t_b$; uma série de períodos, $T = (t_1, t_2, \ldots, t_n)$ capturados através do monitoramento da infraestrutura; a duração total da disponibilidade do serviço, $u = \Sigma t_i$; definimos MTTF = u/n.

A contribuição do artigo baseia-se no desenvolvimento do *framework* SLA@SOI, que estende as funcionalidades de um publicador/revendedor de serviços de computação em nuvem, passando a identificar a monitorabilidade dos sub-componentes de SLA antes de validar uma negociação. O *framework* é formado pelos componentes a seguir:

- ***Event Bus*:** Captura informações de SLA registradas durante o provisionamento, através de componentes que identificam eventos denominados *event captors* e disponibiliza-as para os demais componentes. Através destes eventos, os monitores podem detectar violações em termos de SLA. O *Event Bus* foi implementado sobre a arquitetura publicação/ assinatura (*publish/subscribe),* ou seja, é baseado na troca assíncrona de

mensagens. Os *event captors* são os publicadores de evento e os monitores são assinantes e consumidores destes eventos, embora os monitores também façam um tipo especial de publicação, “resultados de eventos de monitoração”. Como *time stamping* é crítico para o monitoramento de SLA, é necessário sincronizar os *event captors* com o relógio de um monitor de referência.

- **Módulo de Conversão de Termos de Monitoramento:** Este módulo traduz os termos de SLA acordados em padrões de eventos computacionais que serão verificados em tempo de execução. A linguagem utilizada para expressar padrões de eventos é EC-Assertion, uma linguagem XML baseada em Event-Calculus, que abstrai e representa ações e efeitos.
- **Monitor:** O monitor utilizado pelo *framework* é baseado no EVEnt REaSoning Toolkit (EVEREST). O monitor utiliza os padrões traduzidos pelo Módulo de Conversão de Termos de Monitoramento. As violações de SLA são registradas no SLA *Violations* DB que pode ser utilizado em negociações futuras.
- **Módulo de Verificação de Termos:** Este módulo é responsável por verificar a monitorabilidade dos termos traduzidos pelo Módulo de Conversão de Termos de Monitoramento, acionando o *Capability Manager*. A verificação dos termos de SLA que podem ser monitorados é obtida através da capacidade de monitoramento de cada serviço básico que compõe a hierarquia de SLA. Esta hierarquia é composta pelo monitor de eventos e pelo conjunto de eventos gerados pelos *event captors* de cada serviço. Podem ocorrer trocas de características que possibilitam *(capabilities)* o monitoramento (baseadas em XML) entre dois serviços hierarquicamente relacionados (de um *Atomic Service* para um *Composite Service*).
- **Módulo de Verificação de Termos:** é este módulo que irá recuperar as *capabilities* do serviço a ser monitorado ou recursivamente irá recuperar as *capabilities* dos serviços que hierarquicamente participam de sua composição. Caso as *capabilities* não estejam disponíveis, irá notificar o módulo de negociação de SLA que aquele item não pode ser monitorado, impedindo a negociação de ser efetivada.

3.4 Accountable Cloud

Conforme descrito na sessão 2.2, a*ccountability* é a capacidade de atribuir responsabilidades por ações dentro de um sistema e, adicionalmente, a capacidade de registrar e auditar as ações de cada sujeito dentro deste sistema. Prover serviços em nuvem com capacidade de serem *accountable* é proposta no artigo [11].

Para que os serviços em computação em nuvem sejam *accountable*, é necessário que estes apresentem as seguintes características:

- **Identidade:** Cada ação na nuvem seja atribuída a uma entidade.
- **Registro:** Todas ações devem ser registradas de forma a permanecerem íntegras.
- **Auditoria:** Os registros devem ser posteriormente acessíveis para fins de auditoria.

A adição dos seguintes mecanismos é proposta pelo artigo para tornar os serviços fornecidos em nuvem *accountable*:

- ***Logs* invioláveis:** *Logs* que sejam capazes de registrar todas as mensagens trocadas dentro de uma nuvem e que não possam ser violados para falsificação. Estes *logs* são base para auditabilidade do serviço prestado em uma nuvem, pois registram o comportamento do provedor e o cliente durante o uso do serviço.
- **Capacidade de reproduzir eventos, baseado em virtualização:** Para reproduzir o ambiente onde determinado evento ocorreu, pode-se armazenar imagens das máquinas virtuais utilizadas. Um dos desafios desta proposta é a capacidade de armazenamento necessária para estas imagens e a utilização de recursos para reprodução numa segunda nuvem idêntica para simulação do desempenho obtido na primeira.
- **Definição da taxa de amostragem:** A medição de SLA é um processo estocástico, portanto é necessário que sejam definidos intervalos de tempo suficientes para uma garantia probabilística de sua correta medição. Além da definição adequada de intervalos de *checkpoint*, podem ser realizadas

medições entre os *checkpoints* de um subconjunto de itens, permitindo a detecção mais ampla de eventuais anomalias na prestação do serviço.

- ***Timestamping* confiável:** *Timestamping* é essencial para detecção da sequência de eventos. Deve-se utilizar um serviço de *timestamping* confiável para as partes cujo SLA é monitorado.

Os requisitos sugeridos em [11] são implantados nesta dissertação, com exceção da capacidade de reproduzir eventos, uma vez que o protótipo utilizará medições obtidas em diversos ambientes, não se restringindo às medições em máquinas virtuais.

3.5 Auditabilidade Mútua

Em meio aos desafios de segurança impostos por computação em nuvem, o artigo [29] procura distinguir ameaças efetivamente introduzidas por computação em nuvem dos problemas de segurança tradicionais, comuns a outros ambientes distribuídos. Como contribuição do artigo, duas novas questões são introduzidas como desafios de computação em nuvem: a complexidade de confiança em várias entidades (*multi-party trust*) e a consequente necessidade de auditabilidade mútua.

Muitas das ameaças de segurança já existem para aplicações web e hospedagem externa de dados, caracterizando-se, portanto, como ameaças para computação em nuvem. Entre estas ameaças pode-se citar períodos de indisponibilidade, perda de dados, senhas vulneráveis e *hosts* comprometidos rodando *malwares.*

Algumas ameaças são efetivamente relacionadas ao advento de computação em nuvem como o ataque baseado na criação de uma máquina virtual atacante na mesma máquina física onde está a máquina virtual alvo, capaz de coletar informações desta. Este tipo de ataque explora a utilização de virtualização e o ambiente multi-locatário, características do ambiente de computação em nuvem. Questões de reputação também são influenciadas pelo ambiente multi-locatário, dado que ações de um usuário dentro de uma nuvem podem impactar negativamente na reputação do provedor de serviços, afetando outros clientes.

A utilização de serviços em nuvem pressupõe uma longa cadeia de confiança. O usuário pode utilizar uma aplicação provida por um SaaS que utiliza uma plataforma de um PaaS que roda na infraestrutura de um provedor de IaaS, tornando a relação de confiança mais complexa que a tradicional usuário-provedor.

O artigo propõe auditabilidade mútua, para assegurar que usuários e provedores comportam-se de maneira benigna e correta, tanto do ponto de vista de consumo como no faturamento dos serviços. Auditabilidade mútua também pode auxiliar o tratamento de incidentes de segurança, uma vez que ambos podem ser origem ou destino de um ataque, permitindo a atribuição de responsabilidade sobre o incidente.

A implantação de um sistema de auditabilidade mútua presume a participação de uma terceira parte independente desempenhando o papel de auditor, consistindo num formato diferente do atual, onde o registro de auditoria e sua manutenção ficam a cargo do provedor. A implantação de sistemas de auditabilidade mútua ainda é um desafio em aberto, não existindo propostas que implantam este conceito. A proposta do artigo de implementar nuvens que possam medir mutuamente o comportamento do provedor de serviços e do cliente norteou o desenvolvimento do protótipo a ser apresentado a seguir neste trabalho.

3.6 Monitorabilidade de SLA na oferta de serviços de aplicações

O artigo [15] modela a relação básica de provimento de serviços sobre a Internet.

Todo SLA deve ter como parte de suas cláusulas a quantidade de requisições esperadas para determinado serviço (*throughput*), uma vez que a quantidade de requisições, se for superior à quantidade prevista, pode causar degradação do nível de serviço prestado.

A definição das entidades, dos eventos e das observações (intervalos de tempo) no fornecimento de um serviço é utilizada para determinar formalmente o grau de monitorabilidade de um SLA.

O protótipo a ser apresentado adiante neste trabalho adapta, para a arquitetura de computação em nuvem, o cenário básico de oferta de serviços pela Internet apresentado no artigo. O artigo relaciona grau de monitorabilidade de um serviço com a capacidade de gestão de seu SLA, assim como proposto em [29], utilizando por sua vez o termo auditabilidade.

A monitorabilidade de um SLA é classificada como não monitoravel, monitorável por uma das partes, mutuamente monitorável por ambas as partes, ou arbitrariável. A monitorabilidade arbitrariável ocorre quando um terceiro de confiança de ambas as partes pode observar todos os eventos relevantes e, portanto, arbitrar no caso de qualquer disputa de SLA. Todo o sistema de SLAs é categorizado com base na monitorabilidade do membro menos controlável.

Tradicionalmente, o cliente irá acordar um SLA juntamente com o provedor para definir suas expectativas quanto ao fornecimento do serviço. Entretanto, a redação do SLA e seu respectivo monitoramento conduzido por cada uma das partes pode ser insuficiente, dada a existência das seguintes barreiras:

- **Dissociação entre os eventos de tentativa de consumo do serviço e o tempo total de disponibilidade do serviço:** A percepção do cliente quanto à disponibilidade do serviço é baseada nas vezes que tentou utilizá-lo; já a métrica adotada pelo provedor é o monitoramento estatístico do serviço, verificando-o em intervalos pré-definidos dentro da janela de uso pré-estabelecida.
- **Envolvimento de outras partes no provimento do serviço:** A causa de uma falha pode estar associada a quaisquer entidades que participam do serviço. Baseando-se no exemplo do cenário apresentado acima, a responsabilidade sobre a falha pode estar localizada no provedor de conexão à Internet. Muitas vezes ambos podem apresentar medições inconsistentes entre si: cliente, provedor de serviço e provedor de Internet podem obter medições diferentes sobre o mesmo serviço prestado, simplesmente por terem realizado medições em intervalos diferentes, e não raro, o intervalo em que uma falha ocorreu pode não ter sido registrado. Desta forma torna-se improvável a atribuição de responsabilidade pelo evento de falha.

O monitoramento de SLA deve contemplar, portanto, a observação de eventos como forma de registrar ocorrências relevantes na entrega de serviços. Um cenário simples de provimento de serviços é demonstrado na Figura 3.5:

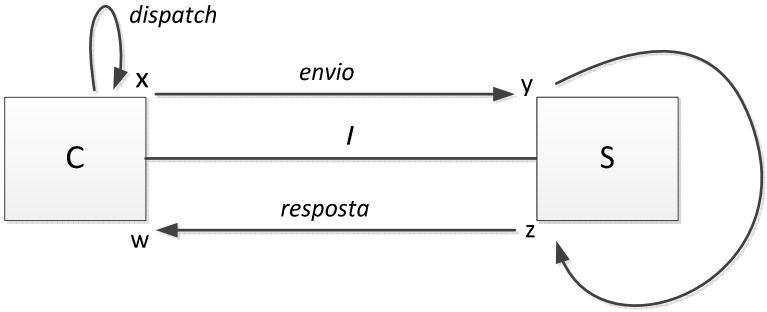

Figura 3.5 – Modelo do provimento de um serviço pela Internet
Adaptada de [15]

O cenário pode ser modelado com os seguintes componentes:

- **Entidades:** Entidades que compõem a relação de serviço. No modelo acima temos: Cliente (C), Provedor de Serviço (S) e Provedor de Acesso à Internet (I);
- **Eventos:** Representam marcos dentro de uma transação. Cada marco é associado a uma sub-tarefa dentro da execução de um processo. Nesta abstração são representados: *Dispatch* (x), *Send* (y), *Process* (z), *Respond* (w);
- **Observações:** Cada observação contém a quantidade de tempo utilizada pelo processo para a transição entre os estados, ou seja, para conclusão de cada sub-tarefa. As observações são:
  - Tempo para envio: y-x
  - Tempo para processamento: z-y
  - Tempo para resposta: w-z
  - Tempo de conclusão da transação, na percepção do usuário: w-x

Sendo o processo submetido a um SLI de tempo máximo para conclusão da transação (t), para evitar uma violação, necessariamente, w-x < t deve ser verdadeiro. Cada observação pode ser medida individualmente, representando parte

do processo; por isso também podemos atribuir SLIs a cada uma das observações: $y-x < t_1$; $z-y < t_2$; $w-z < t_3$. Por fim, os SLIs das observações intermediárias, se somados, devem ser inferiores ao SLI para toda transação, $t_1+t_2+t_3 < t$.

A modelagem acima é utilizada posteriormente nesta dissertação para estabelecer os princípios que são aplicáveis para que SLAs deste modelo de serviço estejam dentro da escala de monitorabilidade, no mínimo, como mutuamente monitoráveis. Posteriormente, o artigo apresenta um algoritmo que identifica o grau de monitorabilidade baseado nos princípios apresentados, com o objetivo de qualificar um SLA previamente à contratação do serviço.

A avaliação do grau de monitorabilidade não será utilizada neste trabalho, uma vez que o protótipo irá prever a auditoria de SLA para serviços já contratados, não analisando a auditabilidade de um SLA antes de implantá-lo. Isto contrasta com nossa proposta para auditar um SLA corrente, previamente analisado. Adicionalmente, a entrega de serviços em nuvem computacional difere do cenário modelado no artigo, aumentando a cadeia de fornecedores de diferentes serviços em nuvem, ao invés de um único provedor de serviço representado por (S).

As observações quanto à responsabilidade de entidades distintas na entrega do serviço e a decomposição do tempo de resposta percebido pelo usuário serão utilizadas na formulação do protótipo deste trabalho. O cenário de entrega de serviços sobre a Internet (*on premise*) utilizado no artigo será também adaptado para contemplar a entrega de serviços em nuvem computacional.

3.7 Mapeamento de Métricas de Recursos em baixo nível para parâmetros de SLA em alto nível

Serviços ofertados em nuvem sob a forma de SaaS possuem negociação de SLA baseada em SLIs de alto nível, como volume de registros, quantidade de transações e tempo de resposta, por exemplo. Para que o SLA definido seja sustentado, os recursos na camada IaaS devem ser corretamente dimensionados, de modo a haver recursos suficientes. Em cenários *on-premise*, o dimensionamento é feito de acordo com estimativas de *workload* e *throughput.*

Em contrapartida, a computação em nuvem oferece a vantagem de dinamicamente alocar novos recursos. O artigo [30] explora esta capacidade por uma interface de gerenciamento que seja capaz de prevenir violações de SLA, ajustando consumo de recursos de infraestrutura sempre que necessário. A capacidade de autogerenciamento de uma nuvem é um objetivo da FoSII (*Foundations of Self-governing ICT Infrastructures*), uma arquitetura proposta pela Universidade de Viena para gerenciamento e garantia de SLA em nuvem computacional.

O funcionamento da interface de autogerenciamento tem como base o componente Enactor e o *framework* LoM2HiS. O LoM2HiS é responsável por enviar informações de *status* sobre a infraestrutura utilizada e indicar ameaças de desvio de SLA, enquanto o Enactor, conforme as informações obtidas, toma as decisões de alocação de mais ou menos recursos de hardware para o serviço, se necessário.

A interação entre o Enactor e o LoM2HiS ocorre da seguinte forma: na oferta do serviço, o provedor entrega o SLA proposto convertido em métricas legíveis ao LoM2HiS (*row metrics*), que são mapeamentos de SLA de nível mais alto para recursos de infraestrutura.

- Passo 1: Os mapeamentos ficam armazenados no repositório de métricas mapeadas;
- Passo 2: O cliente adquire o serviço;
- Passo 3: O SLA acordado é carregado no repositório de SLA contratado, que serve de base para a alocação dos serviços de infraestrutura;
- Passo 4: As métricas sobre a infraestrutura alocada são obtidas pelo *Host Monitor*;
- Passo 5: *Host Monitor* encaminha as métricas de infraestrutura periodicamente para o *Run-Time Monitor;*
- Passo 6: *Host Monitor* encaminha as métricas de infraestrutura periodicamente para o *Enactor*;
- Passo 7: Ao receber as métricas de infraestrutura, o *Run-Time Monitor* as relaciona ao SLA, cria mapeamentos e os registra no repositório de métricas mapeadas Utilizando dados deste repositório, o *Run-Time Monitor* pode então comparar os mapeamentos correspondentes ao nível de serviço

ofertado relacionado ao obtido no monitoramento, o que possibilita a detecção de ameaças de violação de SLA;

- Passo 8: No caso de detecção de ameaças de violação de SLA, o *Enactor* é avisado de forma imediatamente, requisitando os recursos adicionais de infraestrutura necessários para mitigar a ameaça identificada.

O *Enactor* também pode decidir sozinho sobre requisitar mais recursos de infraestrutura, caso ao receber as informações do *Host Monitor* (Passo 6) encontre coincidência das medições recebidas com medições anteriores que geraram ameaças de violação de SLA.

A Figura 3.6, extraída deste artigo, ilustra o funcionamento acima descrito. A criação de uma interface de autogerenciamento para prevenção de ameaças à manutenção do SLA maximiza os benefícios da escalabilidade da computação em nuvem ofertada em IaaS, garantindo a manutenção do SLA mesmo com grandes variações de demanda computacional.

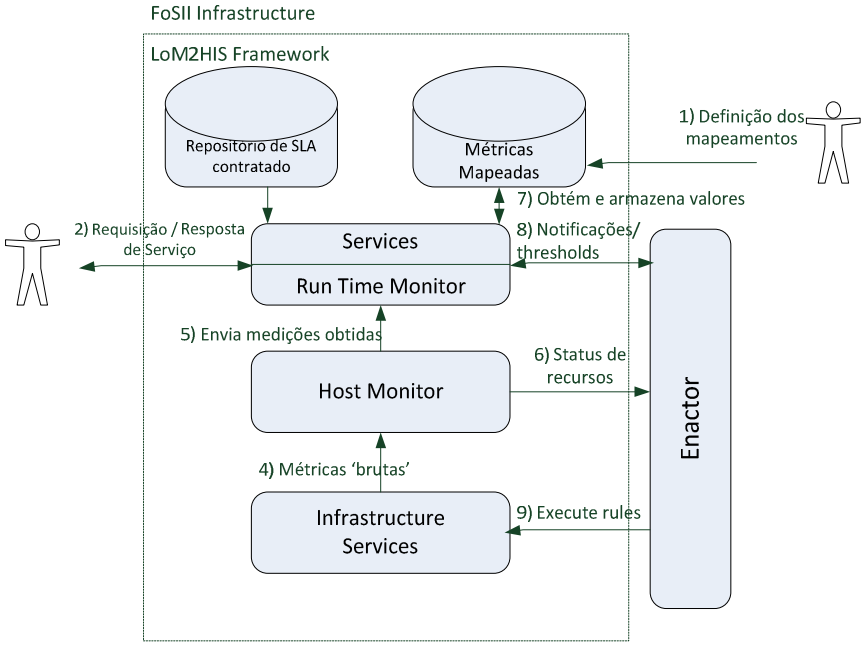

Figura 3.6 – Funcionamento do *framework* LoM2HiS. Adaptada de [30].

É importante observar que o *framework* destina-se a utilizar as métricas obtidas para evitar violações de SLA, não tendo como objetivo a auditoria do serviço para atribuição de responsabilidades. Esta abordagem também não trata a participação de diversas entidades para entrega do serviço, deixando de capturar a

experiência do usuário e problemas em outros níveis de fornecimento do serviço, como, por exemplo, problemas na própria aplicação ou na conexão do usuário à Internet. Para este trabalho, o artigo contribuiu por apresentar uma abordagem para auditoria de serviços em nuvem.

3.8 Agentes distribuídos em nuvem IaaS para auditoria de nível de serviço

Duas propostas de auditoria de nível de serviço em nuvem IaaS, com objetivo de auditar indicadores específicos para segurança, são apresentadas em [31] e [32].

Em [31] é apresentada uma arquitetura de auditoria de segurança para a nuvem, baseada na utilização de agentes assim distribuídos:

- Auditoria na camada de serviço: A auditoria é feita sobre o próprio serviço hospedado, registrando eventos locais, como troca de mensagens ou falhas no serviço.
- Auditoria em nível de nuvem: Todas as requisições de serviço de fora para dentro da nuvem são auditadas. Para isso, insere-se um gerenciador de conexões, que irá funcionar como proxy, ao abrir, registrar e avaliar as requisições para os serviços a serem consumidos. Por funcionar como um ponto único de falha, a inserção de um serviço de proxy, poderia também detectar problemas de performance ou indisponibilidade, pois pode registrar todas as solicitações para o consumo de determinado serviço e todas as respostas fornecidas, indicando se houve resposta àquela solicitação e em que momento ela ocorreu.
- Nível inter-nuvem: Composta pelos mecanismos que compartilham mensagens e recursos entre nuvens distintas. A troca de mensagens entre nuvem é processada pelo gerenciador de conexões. O histórico de federação de serviços pode ser registrado e avaliado também em tempo real.

A contribuição deste artigo está na proposição de dispor os agentes inspetores em diferentes camadas da nuvem para obter medições complementares e correlaciona-las. Entretanto não é abordada a questão de conflito de interesses, deixando-se de mencionar explicitamente a participação de um agente terceiro e se

as informações obtidas são tratadas de forma isolada do ambiente de onde as mesmas são coletadas.

Em [32], introduz-se Auditoria de Segurança como Serviço (Security Audit as a Service, SAaaS). A arquitetura apresentada provê auditoria para nuvem na camada IaaS e compara informações de configuração obtidas no ambiente com as configurações contratadas, como forma de identificar se o ambiente computacional contratado não sofreu alterações não autorizadas.

O modelo baseia-se na realização da auditoria por uma parte terceira que interpreta dados coletados dos agentes distribuídos na infraestrutura da nuvem. Os agentes são assim definidos:

- VM Agent: Agente inspetor que é instalado dentro de cada VM e obtém informações de configuração apresentadas pelo sistema operacional;
- Host Agent: Agente inspetor que é instalado em cada host e obtém informações do hypervisor;
- Cloud Management Agent: Agente inspetor que obtém informações do sistema de gerenciamento da nuvem.

O artigo sugere extensões destes agentes também para antivírus, firewalls, etc, a fim de se tornar capaz de correlacionar eventos medidos por estas aplicações.

Agentes conectam-se ao hypervisor e atualizam-se sobre alterações no estado das VMs e correlacionam as alterações com informações coletadas por outros agentes responsáveis por obter informações do antivírus, IPS e outros anti-malwares. Cada agente recebe um conjunto de métricas e respectivos thresholds. A cada desvio dos thresholds preestabelecidos, os agentes enviam os eventos para um barramento denominado serviço de eventos.

A camada de processamento é responsável por detectar anomalias e correlacionar eventos. Esta camada pode reagir autonomamente a desvios, da mesma forma que proposto em [30].

Quando as medições fora dos thresholds estabelecidos indicarem um desvio de SLA, uma camada externa ao processamento, responsável pela governança do serviço de auditoria e denominada camada de apresentação, registra o comportamento do provedor e gera relatórios sobre sua eficiência.

Da mesma forma que em [31] agentes inspetores distribuem-se em posições distintas da nuvem, para auditar o nível de serviço prestado. A contribuição deste artigo é a distinção explícita entre agentes inspetores, processamento e correlação de eventos e camada de governança e apresentação. Apesar das arquiteturas de auditoria serem restritas às nuvens em IaaS, a contribuição obtida dos dois artigos é a correlação de informações obtidas por agentes inspetores distribuídos dentro da nuvem computacional.

3.9 Considerações

Arquiteturas de auditoria para nuvem em IaaS como em [31] e [32] são capazes de obter informações através de agentes inspetores e correlacioná-las por entidades externas auditando eventuais desvios de nível de serviço dentro de uma nuvem, na camada de provimento IaaS.

Numa arquitetura em nuvem vários provedores podem participar da entrega do serviço para o usuário final, podendo haver provedores IaaS e SaaS distintos[15]. Entretanto, as arquiteturas de auditoria pesquisadas não medem desvios de SLA em camadas de provimento distintas na nuvem, deixando portanto de ser capazes de responsabilizar um provedor faltoso dentro de uma cadeia de fornecimento do serviço de computação em nuvem.

Além de ser entregue por provedores distintos, a qualidade do serviço em nuvem também é afetado por provedores externos até que seja consumido pelo usuário final, momento onde se obtém a experiência do usuário [25]. Nenhuma arquitetura de auditoria analisada é capaz, porém, de obtê-la. Assim, as propostas estudadas não identificam causas de desvio de SLA que sejam externas à nuvem.

A seguir estão sumarizados os aspectos relevantes para a proposta deste trabalho obtidos de cada abordagem.

- **Auditoria de Acordo de Nível de Serviço em *grid computing***

O artigo **[24]** introduziu a participação de uma entidade terceira como auditora de serviços distribuídos na Internet. As arquiteturas avaliadas no artigo foram testadas

para *grid computing* e não são aplicáveis para computação em nuvem. Por isso, as recomendações do artigo quanto à arquitetura mais adequada para este ambiente computacional não são utilizadas no protótipo desenvolvido. Em contrapartida, a minimização do impacto da participação do auditor na solução foi discutida para todas as arquiteturas, e esta avaliação inspirou uma das principais características do protótipo.

- **Qualidade da Experiência: Obtendo a Qualidade de Serviço percebida pelo usuário**

No artigo [25] é apresentado o conceito de QoE, que foi utilizado na concepção do protótipo. Ao medir como o serviço é apresentado ao usuário final, estamos utilizando métricas de QoE. A métrica adotada é baseada em um dos exemplos citados pelo artigo como "tempo de resposta fim-a-fim". Em outro artigo [26], citado na mesma seção, obtivemos o tempo de resposta máximo fim-a-fim que será parâmetro de SLI para os testes (3 segundos).

- **Capacidade de monitoramento (monitorabilidade) de SLA como requisito para aquisição de serviços em computação em nuvem**

O artigo [28] apresenta um revendedor de serviços de computação em nuvem capaz de avaliar a monitorabilidade de cada item de SLA. O modelo decompõe cada item de SLA sucessivamente até o nível de recurso de infraestrutura. Ao obter a composição do SLA, o revendedor identifica se cada um dos componentes é monitorável antes de permitir a negociação. A decomposição do SLA também foi implementada no protótipo , mas de forma distinta - as informações sobre o cumprimento do SLA são determinadas pela camada de software e, somente no caso de violações, é que são avalidas as informações de infraestrutura disponíveis, mencionadas no artigo como IS.

- ***Accountable cloud***

O artigo [11] discorre sobre os critérios de fornecimento de computação em nuvem para permitir a adoção de medidas de monitoramento. Entre os mecanismos apresentados no artigo, foram utilizados:

- A adoção de *timestamping* confiável para o correto sequenciamento de eventos. Neste caso, o ajuste de relógios foi implementado externamente ao protótipo, no ambiente de testes.
- A geração e manutenção de *logs* de eventos em repositórios invioláveis. O protótipo não implantou mecanismo para salvaguarda protegida dos registros, embora permita a utilização de base de dados em ambiente segregado com a utilização de mecanismos de proteção que podem ser implementados *a posteriori*.
- A definição de taxa de amostragem adequada. Uma vez que o protótipo pretende avaliar a percepção do usuário sobre o serviço provido fim-a-fim, a taxa de amostragem é de 100% das transações, não se baseando no tempo em que o serviço está disponível, mas sim em sua utilização quando requisitado.

- **Auditabilidade mútua**

O artigo [29] discorre sobre os riscos introduzidos pela existência de diversas entidades em ambiente compartilhado. Ao mesmo tempo em que ações do provedor têm impacto no serviço de diversos usuários que compartilham o mesmo serviço, um único usuário pode comprometer o funcionamento do serviço para os demais.

Da mesma forma, um usuário pode reclamar problemas de desempenho supostamente gerados pelo provedor, sendo a fonte do problema, na verdade, uma situação gerada por suas próprias aplicações ou configurações.

O protótipo desenvolvido procura identificar a responsabilidade por violações de SLA sugerindo os casos nos quais o problema é gerado pelo próprio cliente, em decorrência de consumo excessivo de recursos *vis-à-vis* o contratado, ou dos provedores de serviço, de acordo com os dados coletados nas transações cujo SLA foi violado.

- **Monitorabilidade de SLA na oferta de serviços de aplicações**

O artigo [15] utiliza um cenário clássico de serviços providos sobre a Internet (sem abordar uma arquitetura específica), elucidando questões relevantes sobre

monitoramento de SLA que foram implantadas no protótipo da seguinte forma:

- Participação de vários provedores na entrega do serviço: Da mesma forma que o protótipo é capaz de identificar ações de clientes e provedores dentro da nuvem (especificamente avaliando impacto no tempo de resposta fim-a-fim), ele também é capaz de identificar ou sugerir qual provedor foi responsável pela degradação deste tempo. A modelagem do tempo de resposta fim-a-fim também foi adotada no protótipo para a medição de diversos tempos intermediários obtidos entre os eventos de envio e recebimento de mensagens.
- Avaliação do SLA pelas tentativas de uso do serviço, e não pelo tempo pelo qual o serviço permaneceu disponível: Ao utilizar as métricas obtidas em cada transação, o protótipo obtém medições de SLA do serviço efetivamente consumido, e não do serviço disponível para consumo, que é a abordagem de medição estatística em intervalos regulares.

- **Mapeamento de métricas de recursos de baixo nível para parâmetros de SLA de baixo nível**

O artigo [30] utiliza um mecanismo de monitoramento integrado a um revendedor de serviços em nuvem. Ao identificar tendência de desvio de SLA, o revendedor aloca mais recursos de infraestrutura, prevenindo problemas de desempenho causados por exaustão de recursos. Embora o protótipo tenha objetivo distinto (auditar o comportamento das entidades participantes da nuvem computacional), o artigo contribui para uma sugestão de melhoria futura, em que poderia armazenar todas as transações utilizando-as também para identificar tendências de violação de SLA e enviar alertas aos provedores de serviço.

- **Agentes distribuídos em nuvem IaaS para auditoria de nível de serviço**

Em [31] e [32] são propostas duas arquiteturas de auditoria de nível de serviço em IaaS. Em comum, as arquiteturas possuem agentes inspetores distribuídos na nuvem computacional, que obtém informações diretamente no provimento do serviço e que as enviam para correlação entre si, a fim de compreender o serviço efetivamente entregue pelo provedor IaaS

4 Desenvolvimento da Proposta

Este capítulo apresenta a arquitetura, a visão geral dos cenários simulados, o papel do auditor e do módulo de governança e a especificação do protótipo. Por fim, são apresentados os detalhes de sua implementação.

Para implantar os objetivos definidos a seguir, a proposta considera a participação das três entidades: provedor, contratante e cliente. O contratante do serviço de nuvem pode ser uma organização pública ou privada que ofereça serviços desenvolvidos em nível *SaaS* a um conjunto de clientes. São os clientes que efetivamente realizam operações nos serviços oferecidos na camada *SaaS* da nuvem computacional, ou seja, os clientes são os usuários finais da aplicação. O provedor de IaaS é denominado apenas como provedor.

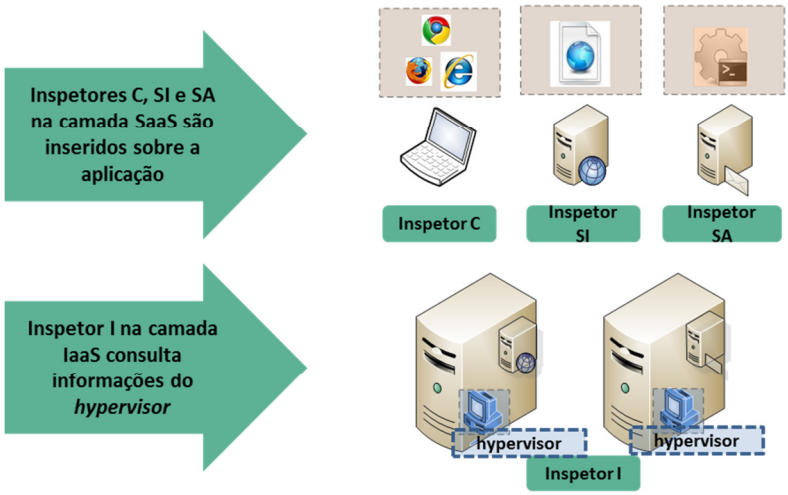

Figura 4.1 – Posição dos Inspetores no protótipo

4.1 Arquitetura do Protótipo

O protótipo é composto por módulos de software denominados Inspetores e Auditores. Os inspetores executam o mecanismo de coleta e são desenvolvidos nas camadas SaaS e IaaS. A Figura 4.1 apresenta a visão geral do protótipo, demonstrando as camadas e ambientes onde os inspetores são executados. O inspetor C é executado no navegador do cliente, o inspetor SI é executado junto à interface da aplicação e o inspetor SA no motor da aplicação.

A arquitetura proposta para auditoria de SLA em computação em nuvem considerou os seguintes aspectos em sua proposição:

- As informações de auditoria devem ser obtidas por um terceiro, entidade independente em relação ao cliente, contratante e provedor, porém confiável a todos;
- Os inspetores coletam os dados do cliente, do contratante e do provedor continuamente e os fornecem ao auditor, que cria o SLI e o compara ao respectivo SLO;
- As informações obtidas no ambiente do cliente e do contratante devem auxiliar na identificação de falhas externas, não associadas ao funcionamento da nuvem;
- As informações obtidas no ambiente do contratante e do cliente podem ser confrontadas com as informações do provedor para avaliar não-conformidades de SLA causadas por falhas internas à nuvem, sejam de responsabilidade do contratante ou do provedor;
- O inspetor não deve conseguir associar o consumo de serviços ao cliente ou contratante dentro da nuvem, mas o rastreamento das ações de clientes e contratantes dentro da nuvem deve ser possível para o pessoal de governança de TI;
- Através do comportamento de contratante e provedor é possível responsabilizar as partes por suas faltas dentro do ambiente de nuvem; e
- A coleta de informações do contratante permite que o nível de serviço seja auditado, considerando-se também a experiência do cliente.

4.2 Visão geral do cenário da proposta

Conforme analisado em [15], a percepção de desempenho sobre um serviço pelo usuário é definida pela somatória do desempenho de vários provedores. O mecanismo de monitoramento e coleta baseia-se na utilização de agentes inspetores inseridos na infraestrutura do provedor e na camada de aplicação do

contratante. Há um agente auditor, lógica e fisicamente, fora da nuvem auditada e uma camada de governança do ambiente. A Figura 4.2 ilustra as correlações entre provedor, contratante e cliente, caracterizadas pelas flechas vermelhas e demonstra a comunicação destas entidades com o Auditor.

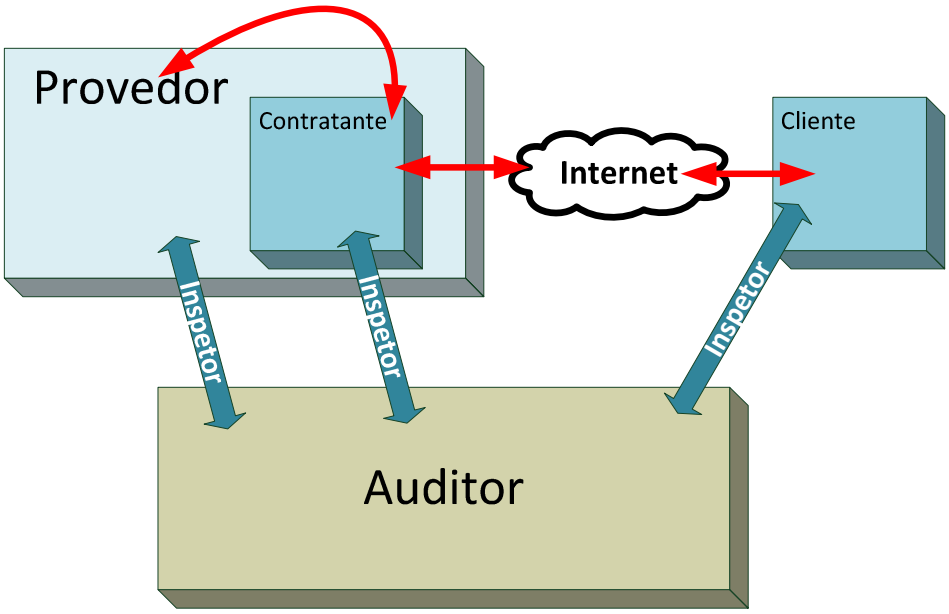

Figura 4.2 – Comunicação entre provedor, cliente, contratante e auditor

Ao auditor cabe correlacionar as medições de SLI, confrontando-os com o SLO para identificar eventual não-conformidade com o SLA. Já a camada de governança tem acesso às informações que permitem uma visão geral consolidada de acompanhamento de cada SLA estabelecido entre um contratante e um provedor, sendo capaz de isolar causas de problemas através da correlação entre medições de SLIs, reunindo as informações que podem ser utilizadas no caso de questionamentos de uma das partes.

Os inspetores são inseridos no ambiente da nuvem computacional, nas camadas IaaS e SaaS. Em ambiente externo à nuvem, o auditor recebe e consolida as informações de cada SLI monitorado.

Este protótipo implanta apenas auditoria para um tipo de SLI para IaaS e um SLI para SaaS, mas seria possível monitorar mais SLIs através da utilização em paralelo de outros inspetores e auditores. Por fim, uma camada de governança pode gerar dados estatísticos a partir dos dados coletados, avaliar o serviço de

determinado provedor ou mesmo as causas do desvio de SLA para determinado contratante. No protótipo, a camada de governança não foi implementada em software, assim como a avaliação de desvios por parte dos auditores, foi manualmente construída. Os dados processados pelos auditores são segregados por SLI, de forma que estes não tenham acesso à visão geral do fornecimento do serviço, evitando conflito de interesses pelos responsáveis pelas medições.

Na proposta, vários inspetores são posicionados em diferentes camadas permitindo correlacionar medições obtidas, a fim de identificar causas de eventos de desvio do SLA. A distribuição dos inspetores dentro das diferentes camadas da nuvem computacional e a posição dos Auditores e camada de governança está ilustrada na Figura 4.3.

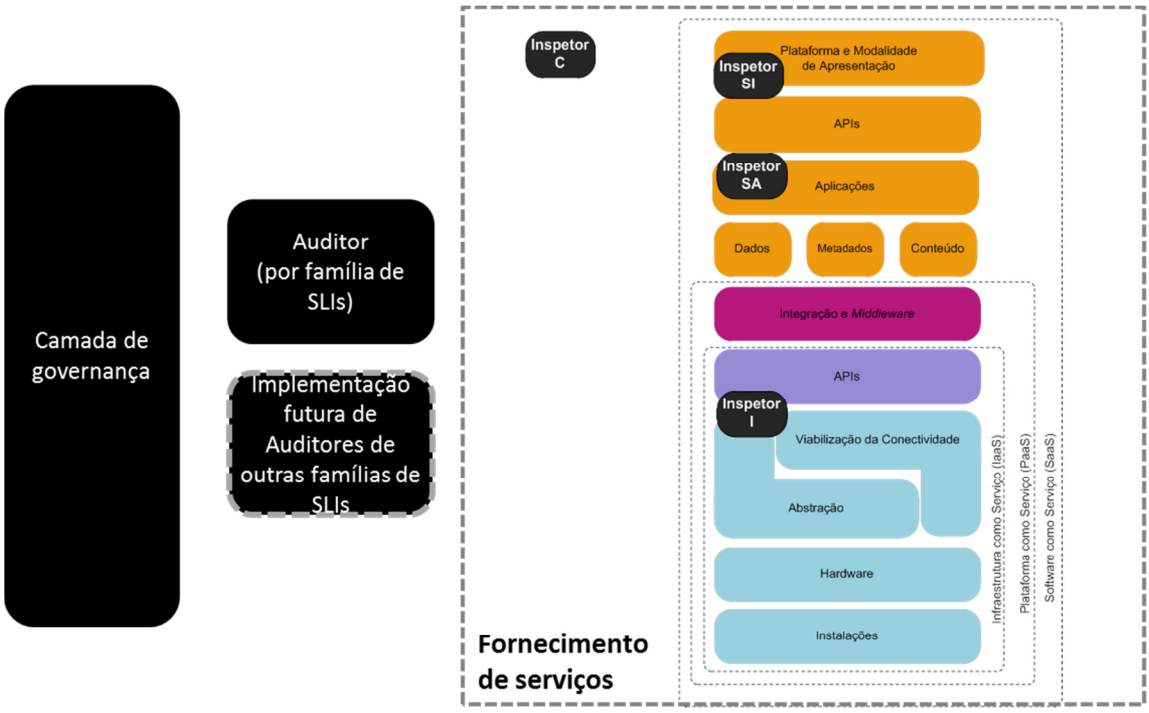

Figura 4.3 – Distribuição dos componentes nas diversas camadas da nuvem computacional

Embora o ambiente computacional do usuário final possa influenciar o desempenho do serviço, para fins de protótipo, os inspetores não obtêm medições de infraestrutura do ambiente do cliente, uma vez que esta premissa pode ser gerenciada na contratação do serviço através do fornecimento das especificações mínimas da estação para funcionamento adequado do serviço em nuvem. Com o intuito de manter coerência com esta premissa, nos testes, os

microcomputadores utilizados para acesso ao serviço têm configuração superior à necessária para execução da aplicação.

A especialização dos inspetores permite que sua execução dentro da nuvem tenha impacto em desempenho minimizado, uma vez que as tarefas executadas pelo inspetor são restritas à consulta de APIs ou à obtenção de pequenos registros de tempo a partir de eventos pré-definidos.

As funções dos auditores e a camada de governança do protótipo são implantadas manualmente e um exemplo das análises obtidas será demonstrado na etapa de testes. Neste protótipo, um inspetor só lê o valor de um indicador relacionado a um pseudônimo. O mesmo acontece com o auditor que cria SLIs do mesmo tipo para o mesmo pseudônimo e os confronta com atributos de SLO para este pseudônimo, disponibilizando-os a camada de governança de modo consolidado. Porém, nem o inspetor, nem o auditor, conseguem relacionar o pseudônimo a uma identidade do mundo real - só a camada de governança tem esta habilidade. A especialização dos inspetores por SLA é representada na Figura 4.4.

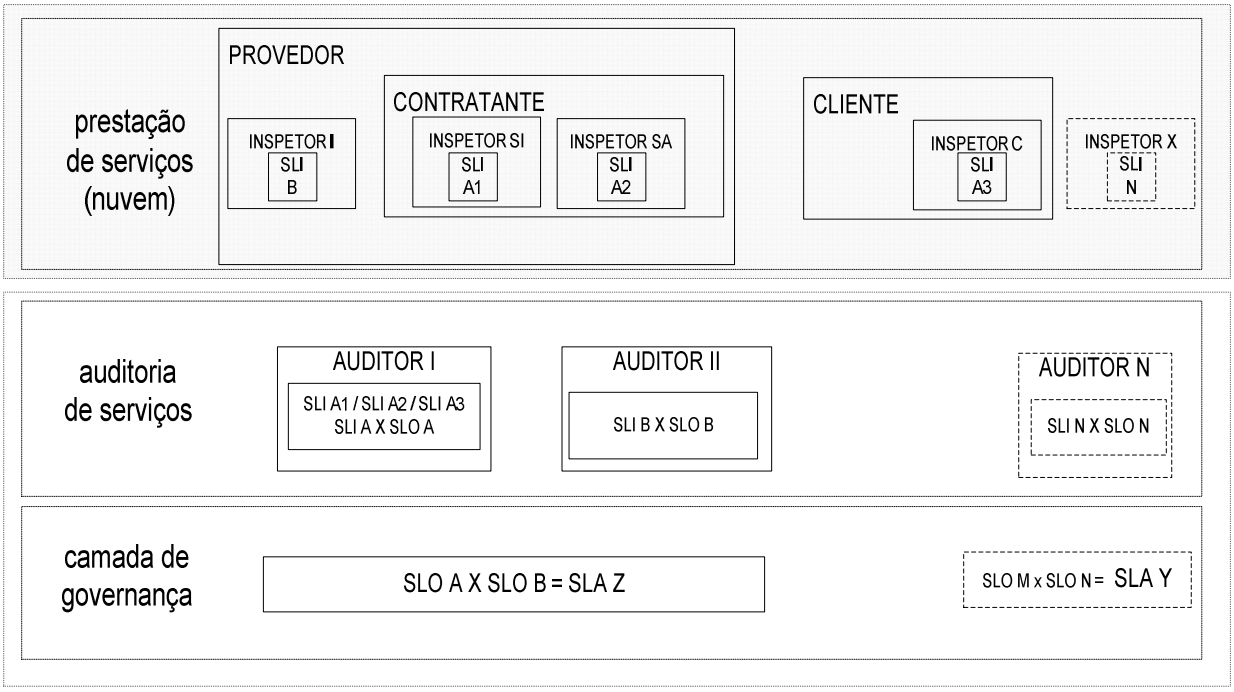

Figura 4.4 – Arquitetura de Controle *online* da Prestação de Serviços em nuvem computacional

Assumimos que a homologação dos códigos que coletam os indicadores deve ser feita antes que os mesmos sejam colocados nos pontos de coleta do inspetor.

Estes códigos podem ser inspecionados a qualquer momento pelo auditor através de geração de *hash code* criptográfico. Neste caso, é sugerido que os códigos coletores sejam desenvolvidos usando a técnica de reentrância [33].

Uma vez que os inspetores consultam as camadas de infraestrutura e aplicação, é necessário definir as premissas iniciais de ambiente para o qual a proposta será prototipada. No âmbito de infraestrutura, a arquitetura presume a existência de virtualização. Para a camada de aplicação, a arquitetura está preparada para funcionar em aplicações disponibilizadas em servidores web, nos papéis de servidor de interface para o cliente e motor de aplicação.

Na camada de infraestrutura, o inspetor (Inspetor I) obtém informações de virtualização sobre recursos disponibilizados e consumidos, controlados pelo *hypervisor*, que extrai as medições de SLI da infraestrutura. Em execução no *hypervisor*, o Inspetor I abre uma conexão com o Xenstat e obtém as informações sobre todo o *cluster* (soma do *hypervisor* e das VMs alocadas) e sobre a utilização de recursos em cada VM.

As informações obtidas pelo inspetor sobre o *cluster* englobam tempo do *cluster* em execução, quantidade total de VMs alocadas, quantidade de VMs por *status* (em execução, bloqueadas, desligadas ou pausadas) e outras informações sobre o servidor físico, tais como memória total disponível e consumida pelo *cluster* e quantidade total de processadores (núcleos) disponíveis.

Por domínio, é possível obter indicadores como a quantidade de CPUs (núcleos virtuais) alocados e o percentual de consumo por CPU, consumo de rede (MB enviados e recebidos), quantidade de requisições que a VM fez para leitura e escrita de blocos em memória ou disco. Este último indicador não separa as informações, uma vez que no Xen o *hypervisor* gerencia o armazenamento e o carregamento de páginas em memória de forma transparente para cada VM.

Na camada de aplicação, os inspetores são instalados no servidor de interface, no cliente e no servidor do motor da aplicação. As informações coletadas pelos inspetores constroem o SLI “tempo de resposta para o cliente *vs.* operação” e outros SLIs intermediários, que medem o desempenho de funções dentro de uma operação:

- Um inspetor de aplicação é executado no cliente (Inspetor C). O objetivo do inspetor C é coletar *timestamps* das transações que ocorrem dentro da estação do cliente (T1 e T8, Figura 4.5).
- Os inspetores de aplicação que são executados no ambiente de aplicação do contratante estão assim divididos e ilustrados na Figura 4.5:

a) O Inspetor SI é executado no servidor de interface registra *timestamps* na interação com o cliente e com o servidor de motor da aplicação (T3, T6 e T7).

b) O Inspetor SA é executado no servidor de motor de aplicação e registra um *timestamp* (T4) ao receber a requisição para execução de operação e outro ao devolver a resposta para o solicitante (T5).

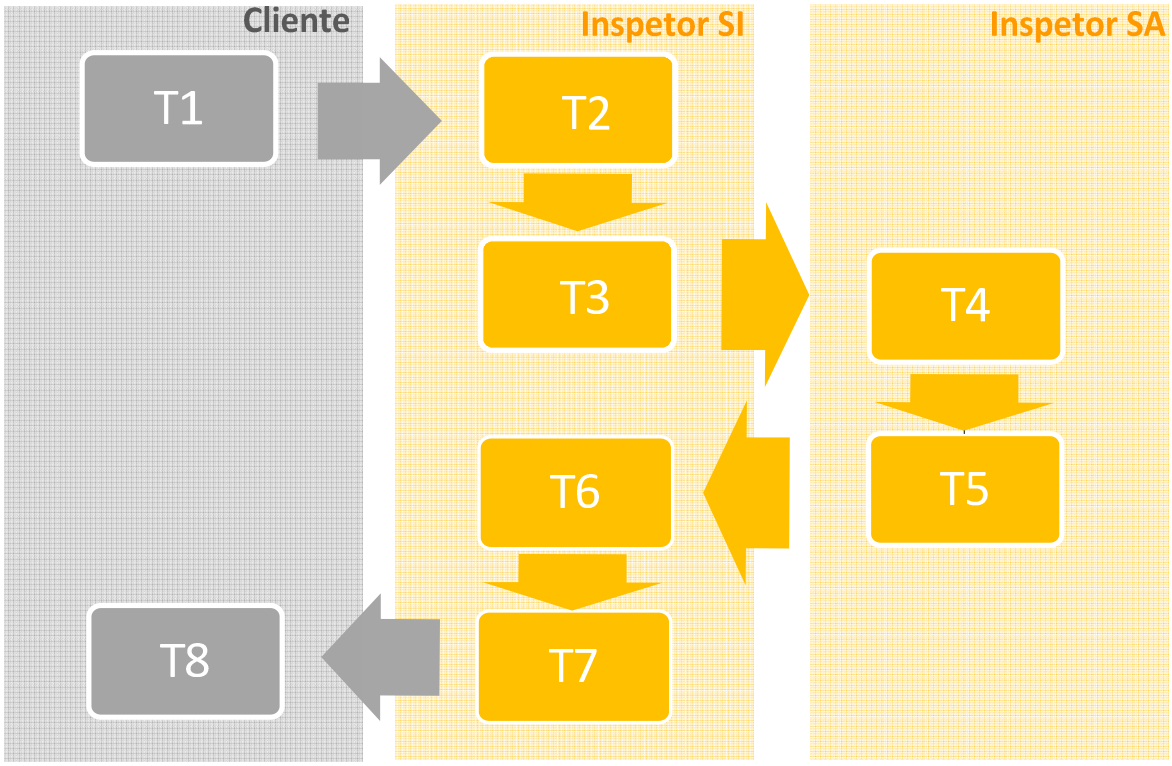

Figura 4.5 – Coleta de *timestamps* pelos inspetores do módulo de software

A partir dos *timestamps* coletados pelos inspetores, o auditor pode calcular os tempos de cada função interna à operação de modo a identificar possíveis inconsistências entre o SLO para a função e os SLIs. A descrição completa de todos os *timestamps* coletados pelos inspetores é mostrada na Tabela 4.1.

Tabela 4.1 – Descrição dos tempos coletados pelos inspetores

Inspetor	Onde	ID Registro	Função interna à operação
Inspetor C	Cliente (estação do usuário)	Tempo 1 (T1)	Registra o *timestamp* do envio de página web pelo cliente, i.e, no momento da requisição de operação para a aplicação.
Inspetor SI	Servidor de interface	Tempo 2 (T2)	Registra o *timestamp* do recebimento do pedido de operação pelo servidor.
Inspetor SI	Servidor de interface	Tempo 3 (T3)	Registra o *timestamp* imediatamente anterior ao envio da solicitação para o servidor de aplicação.
Inspetor SA	Servidor de motor da aplicação	Tempo 4 (T4)	Registra o *timestamp* imediatamente posterior ao recebimento da requisição de operação.
Inspetor SA	Servidor de motor da aplicação	Tempo 5 (T5)	Registra o *timestamp* imediatamente anterior ao envio do resultado da operação.
Inspetor SI	Servidor de interface	Tempo 6 (T6)	Registra o *timestamp* imediatamente posterior ao recebimento da resposta do motor da aplicação.
Inspetor SI	Servidor de interface	Tempo 7 (T7)	Registra o *timestamp* imediatamente anterior ao envio do resultado para o cliente.
Inspetor C	Cliente (estação do usuário)	Tempo 8 (T8)	Registra o *timestamp* imediatamente posterior ao recebimento do resultado da operação.

4.3 O papel do Auditor e do Módulo de Governança

Nos auditores, os *timestamps* geram SLIs que são comparados aos SLOs de referência, identificando desvio para cada sub-operação. A partir da identificação de desvios,.a camada de governança solicita informações adicionais ao Inspetor I. A correlação destas informações pode fornecer as seguintes conclusões:

- Problemas de desempenho externos à nuvem, como no caso dos serviços de conexão à Internet. A existência de altas taxas de latência nas conexões do cliente, do servidor de interface ou do servidor de aplicação pode ser detectada através de SLIs calculados a partir dos seguintes *timestamps*:
    a) No lado do cliente ou servidor de interface:
       se observados valores superiores aos limites definidos para T8 – T7, T2 – T1, respectivamente.
    b) No lado do servidor de interface ou servidor de motor de aplicação: se observados valores superiores aos limites definidos para T4 – T3 ou T6 – T5, respectivamente.
- Problemas de desempenho dentro da nuvem são evidenciados por diferença entre *timestamps* coletados dentro do mesmo servidor, caracterizando maior tempo para conclusão de determinada operação, que pode ocorrer:
    a) No servidor de interface, evidenciado por T3 – T2 ou T7 – T6.
    b) No servidor de motor da aplicação, evidenciado por T5 – T4.
- A identificação de problemas dentro da nuvem, em si, não caracteriza falta do provedor de acesso ou problemas no ambiente do cliente. Esta medição precisa ser combinada às informações obtidas pelo Inspetor I. Para as operações que levaram mais tempo para serem concluídas, o auditor busca evidências de maior consumo ou até mesmo de esgotamento de recursos de hardware, como CPU e memória.
- As correlações entre *timestamps* obtidos pelos inspetores podem revelar anomalias em funções dentro de uma operação. A Tabela 4.2 descreve os SLI e as respectivas análises que podem ser geradas pelo módulo de governança.

Os indicadores Tsf e Tse refletem o funcionamento da própria nuvem e são analisados conjuntamente com informações do Inspetor I. Ao se detectar desvio de SLO em Tsf e Tse, procura-se registros de esgotamento de recursos no respectivo servidor, através de informações do Inspetor I. O recurso analisado no protótipo foi CPU, com o respectivo percentual de consumo. Estes indicadores são circulados hachurado na figura 19.

Tabela 4.2 – SLIs gerados pelo auditor

SLI	Função interna a uma operação	Cálculo	Análise do módulo de governança
Tru	Identifica o tempo de resposta para o usuário final (execução toda de uma operação).	Tru = T8 – T1	Caso ultrapasse o SLO, sugere a análise de mais indicadores para identificar a operação gargalo.
Trd1	Identifica o tempo para o envio da página do cliente para o servidor de *interface* (aplicação).	Trd1 = T2 – T1	Tempo elevado entre o envio e o da página indica gargalos na cor em rede entre o cliente e o servidor.
Tsf	Identifica o tempo de processamento do servidor de interface para executar a operação de recebimento da página e empacotamento da mensagem.	Tsf = T3 – T2	Tempo elevado para receber e transmitir a pagina indica gargalo no servidor de interface, mas não identifica se há problema na disponibilização do hardware contratado ou sobrecarga de recursos. No caso de medições mais altas são necessárias informações adicionais do Inspetor I.
Tse	Identifica o tempo de processamento do servidor de motor de aplicação para executar o envio da mensagem.	Tse = T5 – T4	Tempo elevado para receber e transmitir a página indica gargalo no servidor de motor de aplicação, mas não identifica se há problema na disponibilização do hardware contratado ou sobrecarga de recursos. No caso de medições mais altas são necessárias informações adicionais do Inspetor I.
Trd2	Identifica o tempo para o envio do formulário do servidor de interface para o cliente.	Trd2 = T8 – T7	Tempo elevado entre o envio e o recebimento do formulário indica gargalos na comunicação em rede entre o servidor de interface e o cliente.

continua

Os indicadores Trd1 e Trd2 são predominantemente influenciados por fatores externos, exprimindo o tempo consumido pela comunicação entre o servidor de interface e o cliente. O tempo total consumido é altamente dependente da qualidade

da conexão entre estes, em se tratando de Internet nem sempre pode ser garantida. Estes indicadores são circulados em pontilhado na Figura 4.6.

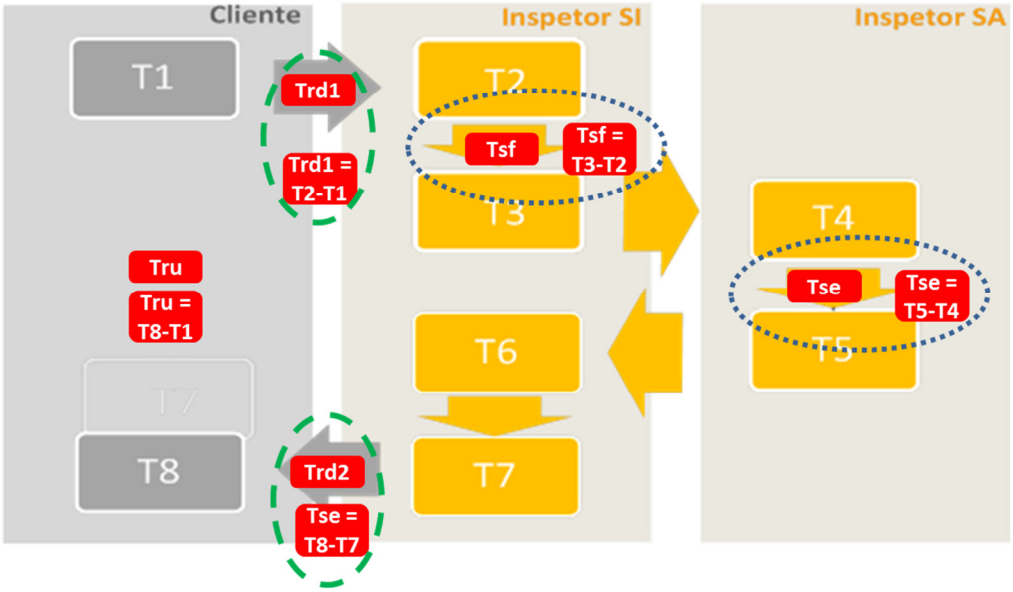

Figura 4.6 – Correlação entre *timestamps* e indicadores

O auditor também foi desenvolvido de forma a preservar a identidade de contratantes e clientes dos inspetores, evitando assim possível conflito de interesse do auditor em favor ou prejuízo de cliente, contratante ou provedor. A arquitetura do protótipo utiliza o mascaramento de identidade, conforme explicado abaixo:

- O contratante do serviço de nuvem pode ser uma organização pública ou privada que ofereça serviços desenvolvidos em nível *SaaS* a um conjunto de clientes. São os clientes que efetivamente realizam operações nos serviços oferecidos na camada *SaaS* da nuvem computacional. Assim, um provedor pode atender a vários contratantes (locatários), que por sua vez podem representar vários clientes.
- Os *timestamps* obtidos pelos inspetores nas operações realizadas por clientes são transmitidos ao repositório do auditor. Naturalmente, cada cliente é identificado por um pseudônimo para evitar conflito de interesse de auditores e inspetores.
- Um servidor de pseudônimos externo ao ambiente do mecanismo de monitoração e coleta (i.e., no domínio de governança) fornece os

pseudônimos ao cliente. O inspetor observa este pseudônimo a cada operação feita nos serviços da nuvem e o registra para fins de rastreamento, porém não consegue obter outras informações a ponto de identificar o cliente.

- O registro da operação é armazenado associado ao pseudônimo, e somente quando necessário ele pode ser associado à verdadeira identidade do cliente. Situações em que o cliente precisa ser identificado estão na esfera legal e geralmente surgem quando há impasse quanto à conformidade de SLAs. O servidor substitui os pseudônimos com certa periodicidade, evitando assim o vazamento de informações de clientes.

4.4 Especificação do Protótipo

O protótipo foi desenvolvido sobre uma aplicação Web, baseada no envio de *e-mails* (mensagens e anexos). A aplicação está subdividida em (i) camada de interface, que é responsável por receber e enviar os formulários de interação (página web) com o cliente, tratando as respostas recebidas e encaminhando as solicitações de processamento ao (ii) servidor de aplicação (um servidor SMTP simples), que por sua vez recebe o formulário e seus anexos e realiza o envio das mensagens via SMTP (*Simple Mail Transfer Protocol*).

Esta aplicação foi escolhida por existirem implementações abundantes e de fácil adaptação para o protótipo. O serviço de *e-mail* do protótipo oferece um formulário simples de envio de *e-mail* contendo destinatário, assunto e mensagem, com a opção de inserir anexos. Cada anexo é criptografado. A Figura 4.7 representa as tarefas que são executadas em cada um dos servidores e no cliente.

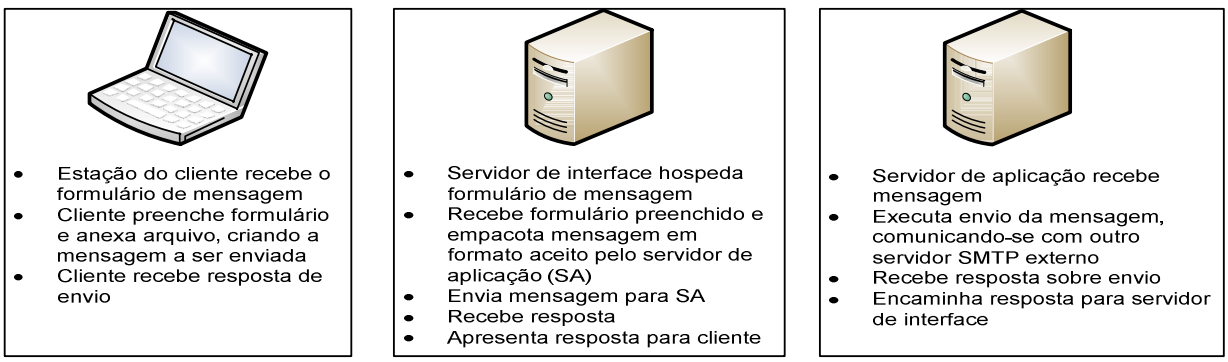

Figura 4.7 – Operações distribuídas entre servidores e cliente

Os servidores são instanciados numa nuvem computacional e devem possuir configuração mínima de 01 núcleo virtual e 256 MB RAM. O servidor de interface usa o Apache TomCat e acesso a um SGBD (Sistema de Gerenciamento de Banco de Dados) externo, nos testes o MySQL. Todos os servidores precisam ter seus relógios sincronizados a partir de um mesmo servidor NTP (*Network Time Protocol*).

Mensagens trocadas entre o servidor de interface e o servidor de motor de aplicação estão no formato JSON (formato de troca de mensagens alternativo ao XML).

O Inspetor I conecta-se no *hypervisor* obtendo informações diretamente do *Xentrace*, que é um gerador de eventos disponível nativamente no virtualizador *Xen*. Este inspetor coleta várias métricas de recursos, mas para fins de teste do protótipo foi considerado apenas o SLI "consumo percentual de CPU", medido por máquina virtual para identificar esgotamento de recursos de processamento. Outras métricas poderiam ser adicionadas, não sendo consideradas para o escopo proposto. O Inspetor I, desenvolvido em 'C', salva as informações coletadas em arquivos texto para posterior consolidação no auditor.

Os SLIs utilizados pelo Inspetor I são limitados às informações que são fornecidas pelo Xen. Anteriormente a implementação do protótipo, pretendíamos analisar também o uso de memória virtual, como um indicador de esgotamento da memória RAM. O *Xentrace*, entretanto, apresenta a quantidade de bytes escritos em memória RAM e disco rígido em um único indicador, uma vez que para o hypervisor oculta as informações de escrita e leitura das VMs hospedadas. Para diferenciar a escrita em memória e disco seria necessário acessar diretamente as informações do sistema operacional de cada VM, distanciando-se do objetivo deste trabalho. Uma vez que o Inspetor I depende das informações geradas pelo hypervisor, sua implementação pode ser distinta se adaptado a outros virtualizadores.

Os inspetores executados na camada de aplicação, no lado cliente, foram implementados da seguinte forma:

- O Inspetor C é construído dentro de um *applet,* codificado em JavaScript, sendo executado pelo navegador do usuário. Ao clicar no botão "enviar", o *applet* obtém o horário da estação, registra o *timestamp* T1 e o adiciona ao

formulário, que será recebido pelo Inspetor SI, hospedado no servidor de interface. O Inspetor C também registra o *timestamp* T8, obtido no momento do *download* do formulário de resposta da solicitação.

```
    public static void enviarEmail(Mensagem mensagem, File anexo, Timestamp horarioDeEnvioDoCliente)
throws Exception {

        Timestamp horarioDeRecebimentoDaMensagem = new Timestamp();
        Operacao operacao = Operacao.nova("Envio de mensagem");

    operacao.registrarEvento("Cliente enviou mensagem para servidor",
    horarioDeEnvioDoCliente);                                                    Obteve registro T1

    operacao.registrarEvento("Servidor recebeu mensagem do cliente",
    horarioDeRecebimentoDaMensagem);                                             Criou registro T2

        String url = AppProperties.mailServiceUrl();
        Logger.info("Convertendo anexo para base64");
        String anexoEmBase64 = toBase64(anexo);
        Logger.info("Preparando request http para mail-service.");
        WS.WSRequest httpRequest = WS.url(url)
                .setParameter("destinatario", mensagem.destinatario)
                .setParameter("assunto", mensagem.assunto)
                .setParameter("conteudo", mensagem.conteudo)
                .setParameter("anexo", anexoEmBase64)
                .setHeader("Accept", "application/json");

        Logger.info("Enviando mensagem para %s", url);

    operacao.registrarEvento("Servidor requisitou envio de mensagem ao mail-
    service");                                                                   Criou registro T3

        WS.HttpResponse response = httpRequest.post();
        Logger.info("Concluiu envio da mensagem.");
        JsonObject data = response.getJson().getAsJsonObject();

    operacao.registrarEvento("Mail-service recebeu requisição de envio", new
    Timestamp(data.getAsJsonObject("recebido")));                                Obteve registro T4

    operacao.registrarEvento("Mail-service concluiu envio", new
    Timestamp(data.getAsJsonObject("concluido")));                               Obteve registro T5

    operacao.registrarEvento("Resposta recebida do mail-service");               Criou registro T6

        Logger.info("Gravando operacao e eventos em banco");
        operacao.salvarTudo();

    operacao.registrarEvento("Feedback enviado para o cliente");                 Criou registro T7

        resultado(operacao.id);
    }
    public static void resultadoRecebidoPeloCliente(long idOperacao, Timestamp timestamp) throws
Exception {
        Operacao operacao = Operacao.findById(idOperacao);

    operacao.concluir("Cliente recebeu confirmação de envio", timestamp)         Obteve registro T8

       ok();
    }
```

Figura 4.8 – Codificação do Inspetor SI

- O Inspetor SI, desenvolvido em Java, registra os *timestamps* relacionados às operações executadas dentro do servidor de interface (T2, T3, T6 e T7), onde

está hospedado (vide seção 4.2). O Inspetor SI tem uma função adicional e específica, obtendo os *timestamps* do Inspetor C e do Inspetor SA e os registrando ordenadamente por operação em banco de dados (MySQL) acessível ao auditor.

- O Inspetor SA, desenvolvido em Java, registra os *timestamps* relacionados às operações executadas dentro do servidor de motor de aplicação (T4 e T5), onde está hospedado (vide seção 4.2).

Além de registrar os próprios *timestamps*, fica a cargo do inspetor SI o envio dos registros obtidos pelos demais inspetores para o auditor, conforme ilustrado na Figura 4.8. Já a codificação do Inspetor C é mostrada abaixo, na Figura 4.9:

```
$ ->
  $("button").click ->
    $(@)
      .attr("disabled", "disabled")
      .html("Enviando mensagem...")
    time = formatDate new Date()                          | Criou registro T1
    $("#dataHora").val time
    $("form").submit()

$ ->
  now = new Date()
  $.ajax
    url: urlResultadoRecebidoPeloCliente
    dataType: "json"
    data:
      "idOperacao": idOperacao
      "timestamp.millis": now.getTime()                   | Criou registro T8
      "timestamp.timezoneOffset": now.getTimezoneOffset()
```

Figura 4.9 – Codificação do Inspetor C

4.5 Considerações

Nesta seção foi apresentada a proposta e a seguir serão mostrado os testes executados para avaliar a proposta deste trabalho.

5 Teste do protótipo

O protótipo foi testado em uma nuvem computacional instalada nas dependências da PUC-PR. Neste ambiente foi possível implantar o protótipo completamente devido à inexistência de restrições do lado provedor, sendo também possível gerar cenários com restrição de banda e com alteração dos recursos de hardware fornecido pelo *hypervisor* - o Inspetor I teve acesso livre às informações do *hypervisor*.

Neste cenário foram utilizados dois servidores rodando o Eucalyptus versão 2.0.3, utilizando o virtualizador Xen versão 4.1.2. São necessários no mínimo dois servidores físicos para que uma nuvem seja criada pelo Eucalyptus. O servidor 01 hospeda o módulo de gerenciamento do Eucalyptus (*Cloud Controller*), onde são definidas e gerenciadas as máquinas virtuais. O servidor 02 hospeda o *hypervisor* Xen e os servidores virtuais (servidor de interface e de aplicação). Uma vez que o cliente e os servidores precisam ter um servidor NTP comum, eles foram sincronizados com o servidor de NTP da PUC-PR.

O servidor virtual que hospeda a interface da aplicação possui configuração de 01 núcleo virtual e 256 MB RAM. O sistema operacional utilizado é o Linux Ubuntu 10.04, onde foi instalado o Apache TomCat versão 7.0.27 para disponibilizar o *servlet* de interface. O servidor virtual que hospeda a aplicação também possui configuração de 01 núcleo virtual e 256 MB RAM; o Linux Ubuntu 10.04 foi o sistema operacional utilizado.

Por estarem no mesmo servidor físico, os servidores virtuais se comunicam através de uma interface virtual (*bridge*) gerenciada pelo próprio virtualizador. Nos testes, não limitamos a banda entre os servidores virtuais. Já o cliente (uma estação Linux) é conectado através da rede local, cuja conexão com o servidor de interface foi limitada em um dos cenários simulados (Cenário D).

A Tabela 5.1 sintetiza os cenários de teste utilizados.

Tabela 5.1 – Descrição dos cenários testados

Cenário	Descrição	Implantação
A	Recursos de processamento de ambos os servidores dedicados exclusivamente ao processamento da aplicação. Utiliza-se a banda de rede disponível ao cenário.	Execução de simulações no ambiente, sem implantação de restrição de banda ou adição de carga de processamento nas VMs dos servidores interface e aplicação.
B	Recursos de processamento de ambos os servidores comprometidos com aplicações concorrentes. Utiliza-se a banda de rede disponível ao cenário.	Utilização de aplicação Java, com algoritmo de criptografia (chave de 168 bits) e o algoritmo DES gerando consumo adicional de CPU, dentro das VMs dos servidores interface e aplicação. A execução desta aplicação ocorre em paralelo à execução da interface e aplicação.
C	Recursos de processamento de apenas um dos servidores comprometido com aplicações concorrentes. Utiliza-se a banda de rede disponível ao cenário.	Utilização de aplicação Java, com algoritmo de criptografia (chave de 168 bits) e o algoritmo DES gerando consumo adicional de CPU, especificamente na VM do servidor de interface. A execução desta aplicação ocorre em paralelo à execução da interface da aplicação. No servidor de aplicação não é adicionado carga artificial.
D	Recursos de processamento de ambos os servidores dedicados exclusivamente ao processamento da aplicação. Restringe-se a banda de rede.	Execução de simulações no ambiente, sem adição de carga de processamento nas VMs dos servidores interface e aplicação. Foi implantado a restrição de banda através do uso do CBQ (*Class Based Queue*), utilitário Linux para implantação de QoS. A velocidade medida entre o cliente e o servidor foi de 100 Kbps/s.

O objetivo dos testes foi identificar a capacidade do protótipo para:

- Identificar falhas externas percebidas pelo cliente e não associadas ao desempenho da nuvem;

- Identificar ocorrências de não-conformidade de SLA através das informações obtidas no ambiente do cliente, do contratante e do provedor;
- Obter informações que permitam identificar o responsável pela não-conformidade de SLA, através de análises dos SLIs.

O quadro abaixo resume o plano de testes e os indicadores cujas medições poderiam evidenciar a eficácia do protótipo.

Quadro 5.1 – Resumo dos testes aplicados no protótipo

Cenário	Distúrbios internos à nuvem		Distúrbios externos à nuvem	Indicadores relevantes
	Servidor Interface	Servidor Aplicação	Rede	
A	Sem distúrbios gerados, por ser o cenário de referência			Todos (medição inicial)
B	X	X		Tsf, Tse
C	X			Tsf
D			X	Trd1, Tr2

5.1 Resultado dos testes e análises

Os resultados dos testes mostram a capacidade do auditor em identificar ocorrências de violação de SLA, através da formulação dos SLIs e sua comparação com os respectivos SLOs, para cada função interna às operações executadas pelo cliente. O módulo de governança mostrou-se apto a identificar faltas externas ou internas à nuvem com sucesso, relacionando-as ao responsável pelo fato.

As funções são representadas pelos SLIs já comentados na Tabela III. Os indicadores extraídos em cada cenário são mostrados nas Tabelas V, VI e VII. Cada conjunto de testes foi repetido 40 vezes por cenário, onde mediu-se o tempo de uma operação a partir do envio do formulário de mensagem pelo usuário final até o

processamento da mensagem pelo servidor SMTP. Para todos os casos, o coeficiente de variabilidade foi inferior a 5%.

O objetivo do Cenário A foi capturar os indicadores em cenário neutro, sem intervenções internas ou externas à nuvem. Seus resultados são demonstrados na Tabela 5.2 .

Tabela 5.2 – Resultado dos testes para o Cenário A

SLI	Tru	Trd1	Tsf	Tse	Trd2
Média (em mSeg)	4.354	641	427	1.130	519

Os valores de referência para os SLIs medidos nos próximos cenários foram estabelecidos com base no Cenário A, constituindo assim os SLOs para cada operação. Neste cenário os indicadores de consumo de CPU não apontaram esgotamento ou consumo excessivo de recursos; o consumo médio de CPU foi de 12,6% para o servidor de interface e de 7,5% para o servidor de motor de aplicação.

No Cenário B foram iniciadas outras aplicações para geração de carga de processamento adicional nos servidores de interface e motor de aplicação. Com esta carga adicional, esperava-se um maior tempo total das operações na aplicação, além do fato de que o Auditor A fosse capaz de identificar quais as operações foram impactadas pela alteração no ambiente. Seus resultados foram compilados na Tabela 5.3.

Tabela 5.3 – Resultado dos testes para o Cenário B

SLI	**Tru**	**Trd1**	**Tsf**	**Tse**	**Trd2**
Média (em mSeg)	18.466	775	1.162	10.182	579

Como esperado, através das medições dos inspetores C, SI e AS, o Auditor A identificou com sucesso o aumento nos tempos de funções executadas no ambiente servidor, representadas pelos SLIs Tsf e Tse. Quando comparados ao SLO estabelecido com as medições do Cenário A (vide Tabela V), os indicadores alcançaram 270% e 900% do valor de referência, respectivamente. O maior tempo

para execução das funções processadas nos servidores fez com que o indicador SLI Tru, que exprime o tempo total da operação, atingisse 420% do valor do SLO.

Uma vez que as operações identificadas com a maior latência são realizadas dentro da nuvem – pois dependem exclusivamente de seu processamento pelo servidor, o módulo de governança requisita que o Auditor B busque as medições do Inspetor I para obter o consumo de CPU dos servidores. Neste cenário foram identificados 86% de consumo de CPU no servidor de interface e 93% no servidor de motor de aplicação, evidenciando para o módulo de governança que a causa de desvio do SLO para esta operação foi causado por consumo intensivo de CPU pelo contratante. A Figura 5.1 contém o gráfico de ocupação de CPU nos servidores de interface e de aplicação.

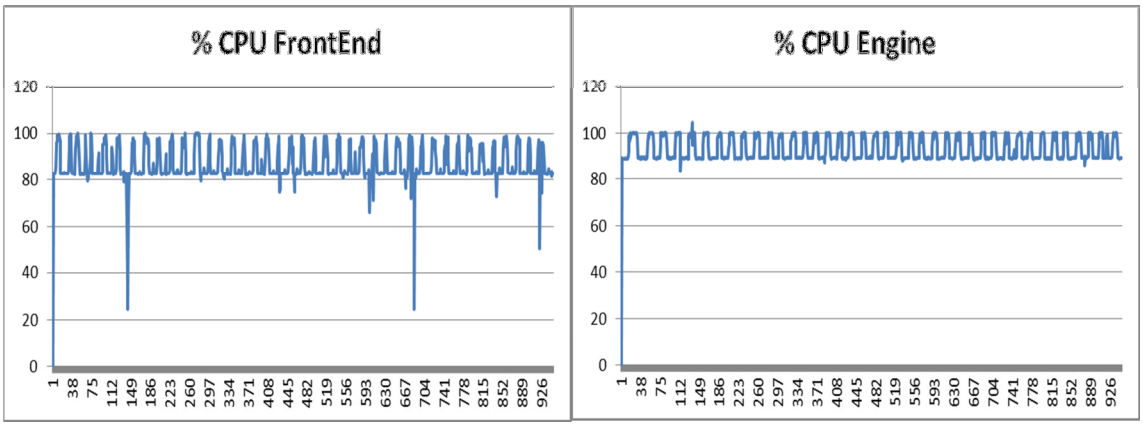

Figura 5.1 – Consumo de CPU dos servidores de interface e aplicação no Cenário B

No Cenário C foram iniciadas outras aplicações para geração de carga de processamento adicional apenas no servidor de interface. Com esta carga adicional, esperava-se um maior tempo total das operações na aplicação, cuja causa seria identificada pelo Auditor A através da análise das funções impactadas pela alteração no ambiente. Os resultados deste cenário foram compilados na Tabela 5.4.

Tabela 5.4 – Resultado dos testes para o Cenário B

SLI	Tru	Trd1	Tsf	Tse	Trd2
Média (em mSeg)	10.396	769	1.019	1.245	1.045

O Auditor A identificou com sucesso o aumento nos tempos de funções executadas no ambiente do servidor de interface, representada pelo SLI Tsf, através das medições dos inspetores C, SI e SA. Comparado ao SLO estabelecido com as medições do Cenário A (vide Tabela V), este indicador atingiu 240% do valor de referência. O maior tempo para execução das funções processadas nos servidores fez com que o SLI Tru, que exprime o tempo total da operação, atingisse 240% do SLO.

Da mesma forma que analisado no Cenário B, a operação identificada com maior latência em relação ao SLO é realizada dentro da nuvem, pois depende exclusivamente de seu processamento pelo servidor de interface. O módulo de governança requisita que o Auditor B busque as medições do Inspetor I para obter o consumo de CPU dos servidores. Como esperado, neste cenário identificou-se maior consumo de CPU apenas no servidor de interface, alcançando 89% de ocupação, enquanto o servidor de motor de aplicação demonstrou ociosidade com 5% de ocupação. Neste caso ficou evidenciado para o módulo de governança que a causa de desvio do SLO para estas operações foi causado por consumo intensivo de CPU apenas no servidor de interface, pelo contratante. A Figura 5.2 contém o gráfico de ocupação de CPU nos dois servidores.

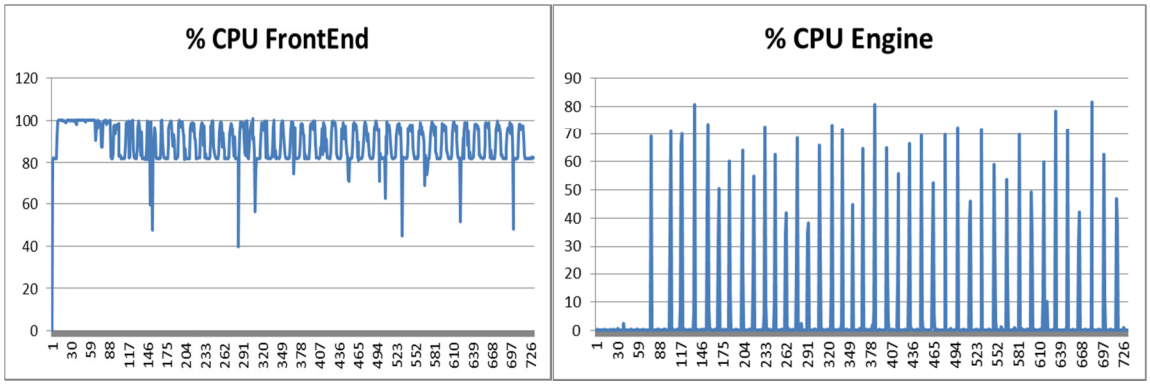

Figura 5.2 – Consumo de CPU dos servidores de interface e aplicação no Cenário C

No Cenário D, a banda disponível para conexão do cliente ao servidor de interface foi reduzida. Com esta alteração, esperava-se um aumento do tempo total da operação, e também que os inspetores C, SI e SA fossem capazes de identificar

quais as funções que foram impactadas pela alteração no ambiente. Os resultados das simulações deste cenário são apresentados na Tabela 8.

Tabela 8 – Resultado dos testes para o Cenário B

SLI	Tru	Trd1	Tsf	Tse	Trd2
Média (em mSeg)	15.557	12.009	189	1.185	654

Através dos dados dos inspetores C, SI e SA, o Auditor A identificou com sucesso o aumento nos tempos de operações que dependem da conexão de rede entre cliente e servidor de interface, representadas pelos SLIs Trd1 e Trd2. As medições obtidas para estes indicadores atingiram 1870% e 130% do SLO. O desvio foi expressivamente maior para o indicador Trd1 quando comparado ao Trd2 porque a operação medida por Trd1 contempla todo o tempo da operação (incluindo a função de envio do formulário e de transmissão do arquivo de 3,5 MB), enquanto em Trd2 apenas um página de confirmação é enviada do servidor de interface para o cliente.

Como esperado, o módulo de governança identificou que as operações com maior desvio do SLO são operações dependentes de conexão de rede, confirmando que a responsabilidade do descumprimento do SLA foi externa à nuvem.

Em todos os cenários em que geramos modificações no ambiente (Cenários B, C e D), observamos que o módulo de governança pôde determinar as ocorrências de não-conformidade com o SLA nas operações com maior tempo de resposta, identificando também as causas do desvio. Em todos estes cenários, as causas foram apuradas através de informações coletadas pelos Inspetores C, SI, AS e I, e pela consequente geração de SLIs e comparação ao SLO realizada pelos Auditores A e B.

5.2 Limitações do protótipo

Os testes em nuvem pública evidenciaram que entidades terceiras não conseguem obter informações de baixo nível sem utilizar as APIs oferecidas por cada uma das nuvens disponíveis. Portanto, a auditoria em nível de hardware não pode ser realizada de forma independente por entidade terceira às nuvens públicas utilizadas no teste. A mesma situação se repetiria com o uso de outras nuvens comerciais e públicas.

Uma vez que o processamento realizado pelo cliente é mínimo e com baixa probabilidade no tempo total da transação, o protótipo não analisa casos de quebra de SLA por problemas de desempenho gerados no cliente.

Para simplificação, a implementação do protótipo assume que o SLO por operação dentro de uma transação é obtido através da simulação da aplicação em ambiente sem interferências na banda disponível ou processamento, que foi gerado através do Cenário A. Em situações reais, os SLOs dentro de um SLA refletem as necessidades de uma pessoa ou organização e precisam ser estabelecidos com base nesta avaliação.

Por fim, a identificação de violação de SLA por problemas nos serviços de conexão de rede não é conclusiva sobre o fornecedor a ser responsabilizado pelo problema. Isso ocorre porque a qualidade da conexão entre o cliente e os servidores, ou entre os próprios servidores, pode ser impactada pela quantidade de *hops*, em especial se estiverem em nuvens distintas, utilizando ISPs diferentes, situação esta que não pode ser controlada em ambientes reais. O protótipo limita-se a identificar o problema de conectividade como a causa do desvio de SLA.

6 Conclusão

A definição de um SLA exprime a expectativa de clientes e contratantes e os compromissos dos provedores de serviços em nuvem. Entretanto, a qualidade dos serviços percebida pelo cliente não é afetada unicamente pelo provedor. Fatores externos à nuvem e o próprio comportamento dos contratantes e clientes podem impactar em seu funcionamento.

A simples extração de informações de consumo de recursos de infraestrutura por parte do provedor entrega uma visão parcial e não o isenta de conflitos de interesse sobre a entrega do serviço. Adicionalmente, as medições de SLIs extraídas pelo provedor podem não corresponder à forma como o usuário percebe e avalia o serviço. A medição do serviço percebido pelo usuário, definida em[25] como QoE, é uma forma de identificar o impacto dos fatores internos ou externos. O usuário avalia o serviço através da sua experiência de uso, QoE, que é afetada pelo desempenho de uma cadeia de provedores, incluindo o próprio provedor da nuvem, e as operadoras que fornecem conexão do usuário à Internet.

A utilização de SLIs baseados em QoE podem auxiliar na identificação do componente onde há degradação de nível serviço e que seja causador de descumprimento de SLA, mas esta informação é visível parcialmente ao cliente e parcialmente ao contratante. Desta forma, informações que eventualmente podem identificar gargalos em operações fora da nuvem, retirando a responsabilidade sobre as faltas do provedor, também não são visíveis a este. Paradoxalmente, as informações disponíveis sobre a entrega de serviços por parte do provedor são visíveis apenas ao mesmo. Combinando as informações disponíveis do cliente e provedor, torna-se possível obter uma única visão sobre a entrega de serviços e eventual causa sobre desvios de SLA.

Nesta dissertação, a auditoria de SLA teve como objetivo identificar as eventuais causas de não conformidade do SLA contratado versus o SLA efetivamente entregue ao usuário final. Entretanto, em outras propostas, o monitoramento de SLA relaciona-se à comercialização e consumo de serviços de nuvem. Em [28] a monitorabilidade do nível de serviço para a comercialização de IaaS deve ser confirmada antes do provisionamento do serviço, tornando o

revendedor uma espécie de entidade mediadora entre o provedor e o cliente. Já em [30] o consumo de recursos é monitorado para avaliar tendência de esgotamento e, quando necessário, mais recursos são provisionados pelo provedor automaticamente para a nuvem, explorando sua característica de elasticidade.

A utilização de um auditor independente para identificar a ocorrência de não conformidade com SLA, observa-se em **[24],** onde foi introduzida a participação de uma entidade neutra, confiável e independente para coletar e relacionar as informações obtidas no provimento de serviços em *grid computing*. Em [13] e [14] um auditor independente também é utilizado no provimento de serviços, mas, neste caso é destinado a auditar o acesso às informações salvaguardadas em nuvem computacional. Nesta dissertação também foi feito uso desta entidade neutra para evitar conflitos de interesse entre cliente, contratante e provedor, confirmando a viabilidade da auditoria multipartes. Uma vez que o serviço em nuvem SaaS (cenário utilizado como base para esta dissertação) compreende a participação de três entidades distintas, cliente, contratante e provedor, conforme caracterizado acima, passamos a utilizar o termo auditoria multipartes, ampliando o significado do termo auditoria mútua, que em [29] considerava apenas a participação de duas entidades cliente e provedor no provimento do serviço em nuvem.

Para ser possível a auditoria em nuvem, a entidade de auditoria deve ter acesso às informações diretamente dos ambientes do provedor, do contratante e do cliente, tornando possível consolidá-las em ambiente externo e livre de interferências. A arquitetura de auditoria considera que cliente, contratante e provedor estejam cientes de sua execução, uma vez que é necessária a adição de *plug-in* no lado do cliente e do contratante, bem como a disponibilização do código coletor no provedor, especificamente na camada de virtualização. Este foi o motivo que inviabilizou a execução de testes do protótipo em nuvem pública. Em [12], chama-se atenção para a ausência de recursos e padrão de auditabilidade em nuvem, sendo que atualmente, as auditorias manuais são as únicas efetivamente executadas por entidades terceiras em nuvens públicas.

Os testes realizados em nuvem privada evidenciaram a capacidade do protótipo em sugerir o componente responsável pelo desvio, de acordo com a operação afetada, possibilitando a atribuição de responsabilidade sobre o

descumprimento de SLA. Adicionalmente ao auditor de infraestrutura e ao auditor de aplicação, a arquitetura permite o desenvolvimento de novos inspetores e auditores associados a outros SLIs.

Sugere-se para trabalhos futuros, novas propostas de SLIs auditados em nuvem e a respectiva criação de novos indicadores correlacionados. Sugere-se também a formulação de padrões de interoperabilidade entre provedores de serviço em nuvem e auditores independentes, que sejam independentes da plataforma de tecnologia utilizada em contratante e provedor.

Por fim, conclui-se que a criação de serviços auditáveis multipartes com a existência de entidades de auditoria reconhecidas e de boa reputação pode aumentar o nível de confiança dos futuros contratantes na utilização de computação em nuvem, em especial no provimento de aplicações e infraestrutura de missão crítica.

Referências

[1] JANSEN, W; GRANCE, T. Guidelines on Security and Privacy in Public Cloud Computing" disponível no site do National Institute of Standards and Technology <nist.gov.br>. Acesso em: 04.Nov.2011

[2] BUYYA, R.; CHEE, S.Y.; VENUGOPAL, S.; BROBERG, J.; BRANDIC, I.. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computer Systems – Elsevier. p 599-616, 2009.

[3] BERL, A.; GELENBE, E.; DI GIROLAMO, M.; GIULIANI, G.; DE MEER, H.; DANG, M.Q.; PENTIKOUSIS, K.. Energy-Efficient Cloud Computing, The Computer Journal, Oxford Press, Vol. 53 No. 7, 2010

[4] HUBBARD, D; SUTTON, M.. Top Threats to Cloud Computing, Cloud Security Alliance, disponível em <cloudsecurityalliance.org/topthreats>. Acesso em: 28.Fev.2011

[5] PETERSON, G.. Don't Trust. And Verify. A Security Architecture Stack for the Cloud. IEEE SECURITY & PRIVACY. v.8, ed.5. p.83-86, 2010

[6] KLEINROCK L. A vision for the Internet, ST Journal of Research 2005.

[7] MELL, P.; GRANCE T., The NIST Definition of Cloud Computing, disponível no site do National Institute of Standards and Technology <nist.gov>. Acesso em: 01.Out.2011

[8] IDC, "It cloud services user survey, pt.2: Top benefits & challenges," disponível <blogs.idc.com/ie/?p=210>. Acesso em Dez.2010.

[9] CSA GRC Stack Research Group, Cloud Security Alliance GRC Stack, disponível em: <cloudsecurityalliance.org/research/projects/grc-stack>. Acesso em: 10.Jan.2010

[10] Open Cloud Standards Incubator – DMTF. Architecture for Managing Clouds, disponível em <www.dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf>. Acesso em: 15.Jan.2011

[11] HAEBERLEN, A. A Case for the Accountable Cloud, ACM SIGOPS Operating Systems Review. p.52-57, 2010

[12] CHOW, R.; GOLLE, P.; JAKOBSSON, M.; SHI,E.; STADDON, J.; MASUOKA, R.; MOLINA, J.."Controlling Data in the Cloud: Outsourcing Computation without Outsourcing Control", CCSW '09 Proceedings of the 2009 ACM workshop on Cloud computing security. p. 85-90, 2009.

[13] WANG,Q.; REN,K.; LOU, W.. Privacy-Preserving Public Auditing for Data Storage Security in Cloud Computing, IEEE INFOCOM. p. 1-9, 2010

[14] WANG, C.; Ren, K.; Lou, W.; Li, J.. Toward Publicly Auditable Secure Cloud Data Storage Services; IEEE Network. p 19-24, 2010

[15] SKENE, J.; RAIMONDI, F.; EMMERICH, W.. Service-Level Agreements for Electronic Services. Software Engineering, IEEE Transactions, v.36, n.2, p.288-304, 2010

[16] CSA Research Group. Guia de Segurança para Áreas Críticas Focado em Computação em Nuvem, disponível em <cloudsecurityalliance.org/guidance>. Acesso em: 11.Dez.2010

[17] LANDWEHR, C.E. International Journal of Information Security, Springer-Verlag. p. 3-13, 2001

[18] GRAY, J.; SIEWIOREK, D.P.. "High-availability computer systems," Computer, vol.24, no.9, pp.39-48, 1991

[19] IETF – Internet Engineering Task Force. An Architecture for Differentiated Services. Disponível em <http://www.ietf.org/rfc/rfc2475.txt>. Acesso em 10.Mai.2012

[20] SAUVE, J.; MARQUES, J.; MOURA, A.; SAMPAIO, M.; JORNADA,J.; RADZIUK, E.. SLA design from a business perspective- Ambient Networks, Springer, p.72-83, 2005

[21] DOBSON, G.; SANCHEZ-MACIAN, A.. Towards Unified QoS/SLA Ontologies. Services Computing Workshops, 2006. SCW '06. IEEE. p.169-174, 2006

[22] NUMI, D.; WOLSKI, R.; GRZEGORCZYK, C.; OBERTELLI, G.; SOMAN, S.; Youseff, L.; Zagorodnov, D.. The Eucalyptus Open-source Cloud-Computing System, Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International. p. 124-131, 2009

[23] Xen.org Team. How does Xen work. Disponível em <http:// wiki.xen.org>. Acesso em 11.Fev.2012

[24] BARBOSA, A.C; SAUVÉ, J.; CIRNE, W.; CARELLI, M.. Evaluating architectures for independently auditing service level agreements; Future Generation Computer Systems 22. p. 721–731, 2006

[25] VAN MOORSEL,A.. Metrics for the Internet Age: Quality of Experience and Quality of Business. Disponível em: <http://www.hpl.hp.com/techreports/2001/ HPL-2001-179.pdf>. Acesso em: 20.Jan.2012.

[26] KHIRMAN, S.; HENRIKSEN, P.. Relationship between Quality-of-Service and Quality-of-Experience for Public Internet Service. Disponível em: http://www-v1.icir.org/2002/Relationship_Between_QoS_and_QoE.pdf. Acesso em: 20.Set.2011

[27] FIEDLER, M.; HOSSFELD, T.; PHUOC, T.. A generic quantitative relationship between quality of experience and quality of service. Network, IEEE, v.24, n.2, p.36-41, 2010

[28] COMUZZIA, M.; KOTSOKALIS, C.; SPANOUDAKIS, G.; YAHYAPOUR, R.. Establishing and Monitoring SLAs in complex Service Based Systems. IEEE International Conference on Web Services. P. 783-790, 2009

[29] CHEN, Y.; PAXSON; VERN, K.; RANDY, H.. What's New About Cloud Computing Security?, University of California at Berkeley, disponível em

<http://techreports.lib.berkeley.edu/accessPages/EECS-2010-5.html>. Acesso em: 14.Fev.2011

[30] EMEAKAROHA, V. C.; BRANDIC, I.; MAURER, M.; DUSTDAR, S.. Low level Metrics to High level SLAs – LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. High Performance Computing and Simulation (HPCS), 2010 International Conference on, p.48-54, 2010

[31] XIE, R.; GAMBLE, R.. A Tiered Strategy for Auditing in the Cloud. IEEE Fifth International Conference on Cloud Computing. p. 945-946, 2012

[32] DOELITZSCHER, F.; FISCHER, C.; MOSKAL,D.; REICH, C.; KNAHL, M.; CLARKE, N.. Validating Cloud Infrastructure Changes By Cloud Audits. IEEE Eighth World Congress on Services. p. 377-384, 2012

[33] WLOKA, J.; SRIDHARAN, M.; TIP, F.. Refactoring for reentrancy. ACM SIGSOFT symposium on The foundations of software engineering (ESEC/FSE'09). p.173-182, 2009