UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

VINICIUS BITTENCOURT GARCIA

Relato de Experiência com Testes de Performance em Aplicações de Entrega Contínua

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Érika Fernandes Cota

Porto Alegre
2018

AGRADECIMENTOS

À orientação da professora Érika Cota que contribuiu significativamente com seu conhecimento durante a realização deste trabalho.

Aos professores do Instituto de Informática da UFRGS que colaboraram muito para a minha formação profissional.

Aos colegas de trabalho que auxiliaram muito tanto nas discussões sobre os assuntos discorridos no trabalho como com informações importantes para a completude do mesmo.

Em especial a minha Mãe e Minha Vó pelo apoio, amor, carinho e dedicação em todos esses anos e pela oportunidade de me fornecer alicerces para atingir este objetivo que finaliza com a realização deste trabalho.

RESUMO

A verificação e a validação de sistemas de software são fundamentais para a diminuição de seus custos de manutenção. Dentro da área de testes de software existem inúmeras técnicas com os mais diferentes objetivos e abordagens. Dentre elas, uma das mais importantes é o teste de desempenho, também chamado de teste de performance. Principalmente para sistemas de médio e grande porte, com larga escalabilidade e entrega contínua, testes de desempenho são de extrema importância para garantir o nível de satisfabilidade de um sistema sob alta carga de dados e cenários que podem ocorrer na vida real e que se tornam inviáveis de reproduzir com testes manuais e unitários. Devido à complexidade no início das atividade de um profissional de engenharia de performance, o crescimento constante na área e a alta rotatividade de funcionários em empresas de grande porte, é necessário um modelo de teste de performance para profissionais da TI com pouca experiência em análise de desempenho. O objetivo do trabalho propõe a definição de um modelo abstrato, independente de tecnologia para esses profissionais. O modelo proposto é baseado em outros modelos encontrados na literatura desde abordagens sistemáticas e teóricas até modelos mais práticos e técnicos mas acrescido de informações de profissionais atuando há anos na indústria como engenheiros de performance e as principais informações dos modelos na literatura. Para estudo de caso o modelo foi aplicado em um sistema de análise de crédito de uma empresa globalmente distribuída e de larga escala há 15 anos no mercado. Com a previsão de aumento do volume e uma nova configuração dos servidores físicos uma série de testes de desempenho é definida para avaliar o comportamento e disponibilidade do sistema.

Palavras-chave: Teste de performance. tuning. desempenho.

ABSTRACT

The verification and validation of software systems are fundamental for their decrease in maintenance costs. On software testing area there are countless techniques with the most different objectives and approaches. Among them, one of the most important is the performance testing. Mainly for medium and large systems, with large scalability and continuous delivery, performance testing is highly important to ensure the satisfactory level of a system under high load of data and scenarios that might occur in real life but are infeasible to reproduce with manual or unit tests. Due to the complexity of a performance engineering first activities, the constant growth on the area and high turnover of employees from large size companies, it is necessary a model of performance testing for IT professionals with less experience on the area. The goal of this work proposes the definition of an abstract and technology independent model for those professionals. The proposed model is based on other models found in the literature since systematic and theorical approaches to practical and technical models but added with information from professionals acting for years on industry as performance engineers and with the main information from the models on literature. For a case study the model was applied on a credit analysis system from a large scale and globally distributed company acting for 15 years in the market. With a perspective of a volume increasing and a new physical servers configuration a series of performance testing is defined to evaluate the system behavior and availability.

Keywords: performance testing. tuning. performance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SDLC	Software Development Life Cycle
PM	Project Manager
TI	Tecnologia da Informação
KPI	Key Performance Indicators
SLA	Service Level Agreement
SUT	System Under Test

SUMÁRIO

1 INTRODUÇÃO

Sistemas de software são submetidos a mudanças de forma contínua em praticamente todos os ramos de negócio. Com a expansão de mercado seja geográfica ou economica, torna-se cada vez mais necessária a evolução dos sistemas atrelados aos mercados envolvidos. Além da expansão econômica, avanços de tecnologias, mudanças na demanda e no objetivo de negócio e o aparecimento de falhas exigem a manutenção contínua de sistemas de software. Essas manutenções causam um gasto considerável ao se comparar com o custo total de um sistema e a aplicação de testes de software se torna necessária nos estágios mais iniciais de um ciclo de desenvolvimento para que esse custo seja o menor possível (TASSEY, 2002).

Técnicas de teste de software existem com os mais variados objetivos e métodos de aplicação, desde testes funcionais e manuais para verificação de funcionalidades do ponto de vista do usuário, até testes unitários realizados pelos próprios desenvolvedores na fase de desenvolvimento. Em se tratando de sistemas com larga escalabilidade, entrega e manutenções contínuas e, consequentemente, economicamente expansivos, uma das validações mais importantes que deve estar presente é o teste de performance. Os gastos com a parte física de um sistema de *software* incluindo servidores, rede e sua configuração trazem a exigência de se obter o menor custo aliado ao melhor desempenho possível. Descobrir a configuração ideal de infraestrutura sob as circunstâncias específicas de cada sistema só é possível com a execução e análise do teste de desempenho, simulando condições reais de carga e comportamento na infraestrutura disponível, e obtenção de resultados que demonstram as consequências desses comportamentos.

A área de teste de performance está em constante crescimento e o número de profissionais nesse segmento não é tão grande visto que a demanda aumenta mais que a qualificação de especialistas na área. A integração de conhecimento prévio de estratégias de teste de software, soluções de integração contínua e linguagens de programação tanto codificadas no sistema a ser testado quanto as suportadas pelas ferramentas de automação são os principais desafios na construção do perfil de um engenheiro de performance. Outro desafio de maior importância na engenharia de performance está na definição e criação de estratégias de teste. Por se tratar de teste não funcional, critérios de aceitação e a cobertura do sistema no planejamento de teste tornam-se abstratos. Além disso, o tipo de informação necessária para a definição dos testes nem sempre está explicita em artefatos de projeto. Por fim, a definição e implementação do teste exige conhecimento que

vai além da programação. Assim, da mesma forma que estruturas como grafos auxiliam um testador a pensar em testes para verificação dos requisitos funcionais, necessita-se um modelo onde profissionais sem um conhecimento profundo na área sejam capazes de definir cenários de teste a serem criados, critérios de aceitação para testes de desempenho executados e a escolha correta de métricas para uma avaliação correta do desempenho dos sistemas a serem testados. O tempo de aprendizado devido ao grande número de técnicas de teste de performance junto do conhecimento necessário de linguagens de programação e do entendimento de componentes *hardware* envolvidos no recebimento de grandes volumes de dados torna-se demasiadamente longo ocasionando perda de efetividade nos projetos. Por mais essa razão tem-se a necessidade do modelo de performance.

Autores já propuseram modelos de performance na literatura dos mais variados níveis de abstração e detalhismo. Desde os níveis mais teóricos contendo fórmulas matemáticas e embasamentos de distribuição de carga relacionados a dados estatísticos até modelos mais técnicos que listam passos básicos do processo de teste de performance porém que são compreensíveis apenas por quem tem vivência na área. O problema dos modelos apresentados até então é justamente a falta de informação em apenas um guia considerando profissionais sem conhecimento de performance que desejam ingressar em um projeto mas não dispõem do tempo necessário para o acúmulo de conhecimento hábil para realização do processo todo de análise de desempenho.

1.1 Objetivo

Este trabalho propõe uma solução para um dos maiores desafios de um engenheiro de performance iniciante que é a definição de uma campanha de teste de desempenho para um profissional de TI sem vivência prévia na área de performance. O modelo proposto permite a definição e execução do teste de performance desde a primeira etapa de levantamento de requisitos de teste de *software* passando pela montagem de uma estratégia de teste e sua execução onde todas as variáveis necessárias são providas pelo modelo. O modelo de teste visa ser independente de tecnologia pela possibilidade de ser utilizado na maior quantidade de projetos possíveis.

A aplicação do modelo em um sistema real visa garantir que é possível seguir os passos propostos e obter resultados sobre previsibilidade do comportamento do sistema em um ambiente de produção dadas as informações de carga e os comportamentos esperados durante o uso do sistema. Com esses resultados há a possibilidade de discussão

sobre otimizações e mudanças a serem feitas em diversas partes do sistema, trazendo um ganho de qualidade para o produto final.

1.2 Estrutura do Trabalho

No Capítulo 2 discutem-se referências e conceitos sobre teste de desempenho encontradas na literatura e na indústria de teste de performance. Duas das principais fundamentações que serão utilizadas como base do modelo e seus principais pontos para a abstração do processo de teste de performance também são explicadas no Capítulo 2. Além disso, é apresentado os prós e contras de cada uma fundamentação justificando a necessidade da proposta do trabalho.

No Capítulo 3 o modelo de teste de desempenho é proposto com as mudanças definidas para se adaptar de melhor forma à independência de qualquer tecnologia e para o mais fácil entendimento por parte de um profissional sem experiência nessa área de qualidade de software. A importância de cada passo proposto e exemplos de uso também são encontrados neste capítulo.

No Capítulo 4 será explicitado o sistema utilizado como estudo de caso incluindo todas as tecnologias e plataformas, sua disposição física e lógica dentro do ecossistema onde está inserido na empresa que solicitou o teste. A aplicação do modelo para este sistema do início ao fim será descrita no mesmo capítulo seguindo os passos propostos, desta vez aplicando diretamente na tecnologia necessária para a realização do teste.

Finalizando o trabalho, o Capítulo 5 traz a discussão do que se obteve e eventuais oportunidades para trabalhos futuros decorrentes dos resultados alcançados.

2 FUNDAMENTAÇÃO TEÓRICA

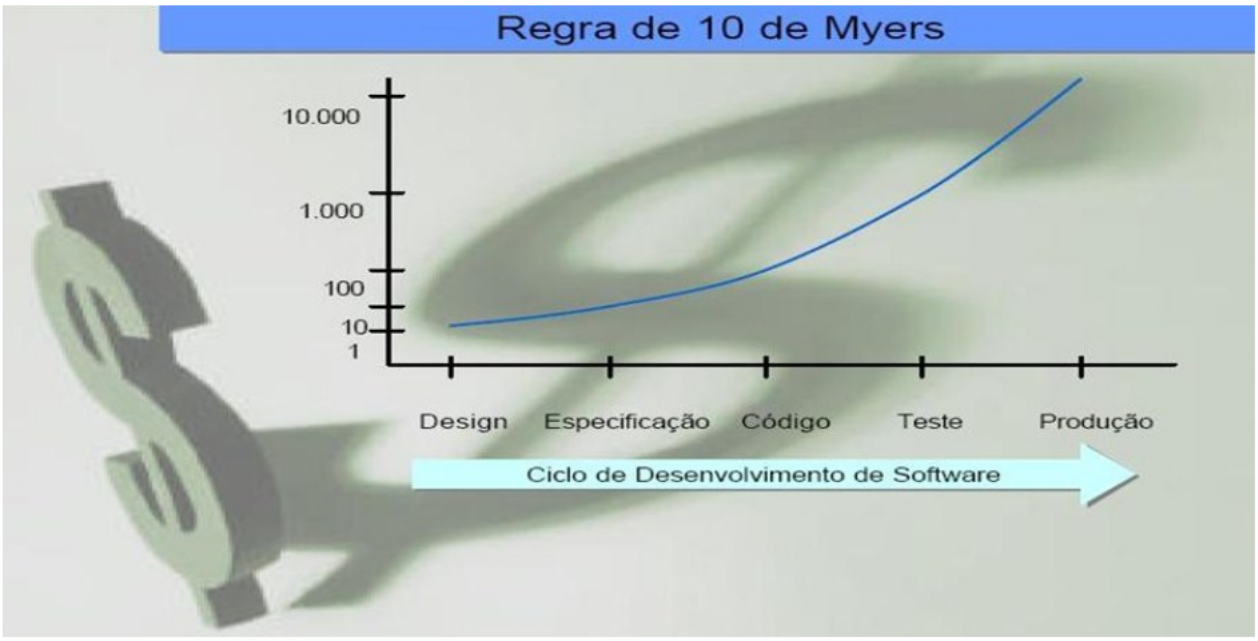
2.1 Teste de Software e Manutenção

Teste de *software* é a técnica de verificação que consiste em avaliar o comportamento do sistema através da observação de sua execução (PEZZE; YOUNG, 2008). Verificação é o processo que determina quando o produto em determinada fase no processo de desenvolvimento de *software* preenche os requisitos estabelecidos durante a fase anterior (AMMANN; OFFUTT, 2016). Visto que um sistema de computador deve ser previsível e consistente para com suas funcionalidades, é necessário o estabelecimento de processos que garantam seu correto comportamento e excluam a possibilidade de funcionalidades não pretendidas por desenvolvedores e clientes finais. Com a expansão do mercado de Tecnologia da Informação (TI) e a crescente necessidade de sistemas cada vez maiores e mais complexos, aliado ao avanço constante das tecnologias necessárias para o desenvolvimento desses sistemas, aplicações com entrega contínua, ou seja, sistemas de *software* com tempo reduzido entre entregas, e, por consequência, em constante atualização são cada vez mais comuns no mercado atual. A constante necessidade de atualização e adaptação ao mercado e aos clientes por parte das empresas também são fatores que impulsionam as atualizações em sistemas já existentes, o que por outro lado aumenta o risco de inserção de comportamentos não desejados a cada modificação no programa.

Visto esse aumento na complexidade e manutenção de sistemas de *software*, mostra-se necessário o investimento nas etapas de verificação e validação do mesmo. De uma maneira geral os requisitos funcionais mudam, devido a diversos motivos como a mudança da necessidade do *software* e quando o usuário demanda o atendimento de outras necessidades. Essas mudanças causam a necessidade dos sistemas passarem por manutenção em parte deles ou até mesmo nos sistemas inteiros. Por esses motivos é comum que os custos de manutenção de um sistema superem os custos de seu desenvolvimento inicial (PEZZE; YOUNG, 2008). Todas essas manutenções constantes abrem grandes possibilidades de inserção de comportamentos não desejados pelo usuário final. Processos de teste de software não garantem um sistema livre de falhas, porém ajudam a reduzir consideravelmente *bugs* no produto final. O custo de correção de falhas aumenta conforme as fases do projeto vão passando. Segundo a Regra 10 de Myers quanto mais cedo descobre-se o erro e se corrige, menor o custo do projeto (MYERS; SANDLER; BADGETT, 2011). Esse custo de correção em *bugs* cresce 10 vezes para cada estágio em que o projeto do software

avança (MYERS; SANDLER; BADGETT, 2011). conforme mostrado na Figura 2.1 é possível ver a proporção do aumento no custo final quando um *bug* é descoberto em fases avançadas. Logo, a falta de um processo correto de teste desde as fases mais iniciais ocasiona um aumento considerável do custo final do sistema.

Figura 2.1: *Regra 10 de Meyers*



Fonte: (CRISTALLI, 2007)

Para entendermos a importância da presença do teste de *software* em todas as fases do sistema vamos considerar o *framework* mais utilizado no desenvolvimento de sistemas, o SDLC (*Software Development Life Cycle*). SDLC consiste em uma série de fases e fornece um entendimento comum para o processo de desenvolvimento de *software* (MYERS; SANDLER; BADGETT, 2011). O SDLC fornece passos para que o time de projeto seja capaz de organizar a entrega de um sistema desde sua coleta de requisitos até as fases finais de teste.

As fases definidos no SDLC consistem na análise de requisitos, onde há a discussão com o cliente sobre suas necessidades e onde buscam-se todos os detalhes do projetos para que cada time do projeto esteja ciente de suas responsabilidades desde testadores, desenvolvedores e analistas de banco de dados. Na etapa de projeto temos o trabalho dos desenvolvedores e arquitetos desenhando o sistema em alto nível com o objetivo de transformar a necessidade do cliente em módulos do sistema a ser desenvolvido. Discussões sobre a tecnologia a ser utilizada, orçamento e riscos de projeto também são discutidos nessa fase. A implementação, onde o código começa a ser escrito levando em conta

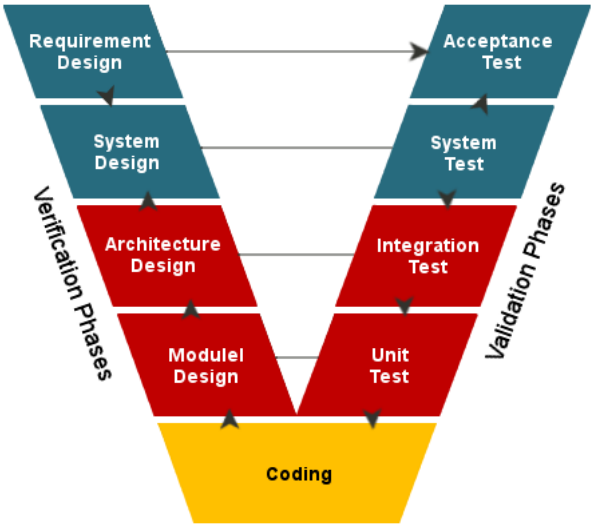
os requerimentos levantados pelos clientes é feita na sequência. Diferentes áreas antes mencionadas como teste e desenvolvimento começam suas atividades nessa fase, com os testadores implementando os casos de teste e preparando a estratégia para sua execução na próxima fase que justamente é a de validação, última fase antes da entrega para o cliente. A fase que completa o ciclo é a de manutenção onde possíveis problemas decorrentes em ambiente de produção podem aparecer e a responsabilidade do time é sanar possíveis problemas ou até mesmo modificar o software de acordo com a necessidade do cliente.

Por este motivo de termos a necessidade da etapa de teste iniciada juntamente com a implementação é que há o Modelo V visto na Figura 2.2. O modelo V particiona as etapas de teste durante as fases de desenvolvimento do sistema. Durante a criação de módulos independentes do sistema ocorrem os testes unitários geralmente implementados e executados pelos desenvolvedores. No processo de projeto de arquitetura do sistema, com a integração dos módulos há o início do teste de integração onde o fluxo de informação começa a ser distribuído pelo sistema e a ser validado. Na implementação do sistema em si é iniciado o teste de sistema, onde testadores iniciam a execução dos casos de teste e as funcionalidades do sistema são validadas. Por fim o teste de aceitação, usualmente executado por usuários finais do produto, é realizado após todas as etapas anteriores de teste menciondas, com o sistema já tendo passado por todo o processo de verificação funcional. Esse teste leva em conta os requisitos levantados nas fases iniciais de levantamento de requisitos. Dessa forma o Modelo V torna o teste de software presente em todo o projeto adicionando valor a cada etapa e auxiliando num produto final de qualidade (RANI, 2017).

2.2 Tipos de Teste de Software

O processo de analisar um *software* com o objetivo de identificar as diferenças entre as condições exigidas e as existentes para esse software (defeitos/erros/*bugs*) e avaliar os componentes deste sistema é conhecido como teste (., 1998). Os engenheiros de teste são os profissionais da área de TI responsáveis por uma ou mais atividades técnicas de teste, incluindo o projeto de casos de teste, execução e análise dos resultados da aplicação dos mesmos. Os gerentes de teste são responsáveis por um ou mais engenheiros de teste e, mais importante que isso, possuem a responsabilidade de entrar em contato com outros gerentes de áreas diferentes do mesmo projeto e de definir as políticas de teste, processos e estratégias de quais tipos de teste utilizar em um determinado projeto. Dentre as aborda-

Figura 2.2: Modelo V



Fonte: (RANI, 2017)

gens de teste e técnicas utilizadas podemos enumerar duas principais categorias de teste de *software*: Testes Funcionais e Testes Estruturais.

Testes funcionais são baseados em funções descritas em documentos de especificação ou compreendidas pelos testadores e devem ser realizadas em todos os níveis de teste. Os níveis de teste vão desde um simples componente de uma aplicação até um fluxo onde podem haver dois ou mais sistemas diferentes que englobem a mesma funcionalidade. O teste funcional considera o comportamento externo do *software*, então a utilização de técnicas baseadas em especificação podem ser utilizadas para derivar as condições de teste e seus critérios de aceitação (MYERS; SANDLER; BADGETT, 2011).

Testes estruturais por outro lado são responsáveis pela verificação dos aspectos não funcionais do sistema, ou seja, parâmetros que não são endereçados pelos testes funcionais como eficiência, segurança, disponibilidade, usabilidade, etc. Uma característica importante dos testes não funcionais que os difere dos testes funcionais é o maior conhecimento das tecnologias em uso do produto e métricas não tão simples de serem obtidas apenas com a execução funcional do *software* como tempos de resposta e utilização de recursos físicos como memória ou disco. Outra diferença significativa é a abstração dos objetivos e da significância dos comportamentos do sistema em termos de requisitos não funionais pois não se tratam de comportamentos certos ou errados como nos testes funcionais onde os requisitos não são subjetivos em sua definição. Testes não funcionais possuem a complexidade no levantamento de requisitos para que a satisfabilidade do pro-

duto final seja alcançada onde testadores devem determinar com os clientes quando o sistema atinge níveis confiáveis de segurança e usabilidade por exemplo. Dentre os testes não funcionais, tem-se o teste de performance responsável pela garantia de boa parte dos requisitos não funcionais como disponibilidade, eficiência e confiabilidade.

2.3 Teste de Performance

Teste de performance é o processo de exercitar uma aplicação emulando situações de uso reais com o propósito de achar gargalos em um sistema (SAROJADEVI, 2011). Gargalos são definidos como elementos de um sistema de *software* cujos recursos estão sendo utilizados na sua máxima capacidade. Usuários finais, *stakeholders* e times de projeto estão todos interessados nos resultados do teste de performance pelos seus objetivos em obter ou prover o melhor desempenho possível com o menor custo (JAIN, 1990).

De uma maneira geral, o teste de desempenho objetiva uma avaliação quantitativa e qualitativa de um sistema sob condições reais de utilização levando em conta seus requisitos de performance levantados nas fases mais iniciais do projeto de uma aplicação ou a cada mudança no sistema durante seus processos de manutenção. Como discutido anteriormente, as necessidades dos clientes mudam constantemente, e conforme áreas de negócio crescem, é comum a necessidade de sistemas com melhor desempenho, e capazes de suportar altas cargas de dados que fluem por eles e quantidades cada vez maiores de acessos simultâneos, não perdendo sua capacidade de suprir esse aumento de carga.

Teste de performance possui três principais objetivos a serem alcançados através de seus processos. O primeiro objetivo é validar o desempenho do sistema em relação aos seus requisitos. O segundo objetivo é encontrar informações no que diz respeito à capacidade e ao limite desse sistema. Tais informações são essenciais para ajudar clientes a comparar diferentes soluções e fazer uma escolha que melhor se encaixe à sua necessidade de negócio. Por último, teste de performance tem por objetivo ajudar arquitetos de sistema e desenvolvedores a encontrar problemas de desempenho tais como gargalos entre aplicações, gerados pela sobrecarga na utilização de CPU de um servidor de banco de dados que possui menor capacidade que o servidor de serviços no momento que os dados estão fluindo entre as camadas de sistema. (FERRARI, 1978).

Jain Raj (1991) reforça a importância e obrigatoriedade da avaliação de desempenho em todos os estágios do ciclo de vida de um sistema, incluindo seu desenvolvimento, uso em produção, etc. Exemplos da necessidade de avaliações de desempenho nas primei-

ras etapas de um sistema são a possibilidade do arquiteto comparar diversas alternativas de implementação e a opção de escolher uma dentre diversas plataformas para desenvolver uma aplicação e, mesmo quando não há escolha, prever o quão bem o software irá desempenhar em uma determinada arquitetura.

O entendimento e a seleção de métricas são fundamentais para o processo de avaliação de desempenho em qualquer sistema. No modelo de processo proposto vamos definir com mais detalhes as métricas a serem utilizadas, porém as mais utilizadas são tempo de resposta e vazão (JAIN, 1990). Tempo de resposta é definido pelo intervalo entre a requisição do usuário e a resposta do sistema. Vazão, por outro lado, é definido pela taxa de requisições por unidade de tempo, a ser definida pelo profissional que irá coletar a métrica. Geralmente, dependendo do componente a ser avaliado, pode-se obter um resultado referente a *jobs* por segundo, milhões de instruções por segundo (MIPS) ou *requests* por segundo (JAIN, 1990). Samuel Kounev (2012) afirma, como razão para o teste de desempenho, a necessidade de validar não apenas o desempenho mas também a escalabilidade e o grau de confiança na eficiência e disponibilidade de um sistema. Disponibilidade é definida como a capacidade do sistema de se mostrar disponível para o usuário final mesmo sob situações adversas tais como picos de cargas não esperados ou problemas possíveis de hardware em suas plataformas. Já sua escalabilidade é definida como a habilidade do sistema de continuar respondendo aos requisitos previamente estabelecidos em termos de tempos de resposta e vazão à medida que ambos, demanda e recursos físicos (normalmente *Hardware*), vão aumentando (KOUNEV et al., 2012). É importante analizar o desempenho esperado e as características de escalabilidade dos sistemas durante todas as fases de seu ciclo de vida. A avaliação de desempenho implementada ajuda a prover recomendações e atingir um nível de desempenho mais otimizado ao longo do tempo de vida de uma aplicação (KOUNEV et al., 2012).

Um dos principais desafios na área de engenharia de performance é a grande quantidade de aplicações, plataformas e tecnologias presentes tornando inviável obter um padrão de medição de desempenho, ou uma técnica e conjunto de processos padrão para todos os casos possíveis onde esse tipo de teste é aplicado. Diversos são os objetivos e recursos para aplicar a engenharia de desempenho e a maneira mais próxima de padronizar o fluxo completo é com o estabelecimento de uma modelagem para o processo de avaliação de desempenho, onde são estabelecidas técnicas de avaliação, critérios de medição e diversas outras partes onde se estabelece um guia passo a passo para o processo de teste (KOUNEV et al., 2012). O segundo desafio em realizar o teste de desempenho é o

conhecimento específico de técnicas de modelagem e análise de performance bem como a solução de gargalos por parte do engenheiro de performance. A necessidade do conhecimento prévio do maior número possível de tecnologias e plataformas, além de técnicas de avaliação de desempenho como critérios para a correta análise de resultados tornam-se um outro desafio presente. A escolha correta de métricas e definição de carga de dados a ser utilizada também possui extrema importância e possui considerável complexidade quando não há vivência na área de performance.

2.4 Modelagem de Teste de Performance

Como vimos, a principal solução para o problema de guiar o processo de teste de performance é com o estabelecimento de passos a serem seguidos desde a fase de análise de requisitos até a execução dos casos de teste definidos pelo engenheiro de performance. Esse conjunto de passos com o objetivo de analizar a escalabilidade de desempenho e analisar as características do sistema através da criação de um modelo se chama modelagem de performance. (KOUNEV et al., 2012)

2.4.1 Modelo Sistemático

Jain propõe uma abordagem sistemática ao classificar os problemas mais comuns na avaliação de desempenho de sistemas. Apesar de métricas e técnicas de avaliação diferirem dependendo do projeto que será avaliado, há passos similares para todo o processo de análise de performance (JAIN, 1990).

1. **Estabelecer Objetivos e Definir o Sistema**
   O primeiro passo no processo de análise de performance é a definição dos objetivos de teste assim como definir o sistema a ser testado e seus limites. Saber o objetivo esclarece em quais aspectos físicos do sistema os testes devem se direcionar. A comparação de duas CPUs por exemplo pode variar caso o objetivo seja avaliar dois tipos de configurações diferentes entre si, como a Unidade Lógica-Aritimética vs (ALUs). Neste caso ambas as CPUs devem ser analisadas entre si. Se o objetivo do teste for a medição dos tempos de resposta em função das duas CPUs, componentes externos devem ser avaliados também. Isso significa que o objetivo de teste deve ser definido assim como o sistema para que as futuras etapas de avaliação sejam

realizadas corretamente.

2. **Listar os serviços e seus resultados**

   Cada sistema lida com tipos diferentes de serviço, sejam pacotes de internet, consultas em banco de dados ou *webservices*. Cada tipo de serviço requisitado pode oferecer uma resposta desejável ou não. Respostas positivas, negativas ou inconclusivas podem ser recebidas pelo usuário como uma mensagem de sucesso, erro ou o não recebimento por atrasos ou *deadlocks*. Listar os serviços que irão fluir pelo sistema e suas possíveis respostas ajudam na seleção de métricas a serem coletadas nos passos futuros.

3. **Selecionar Métricas**

   Neste passo, deve-se selecionar um critério para a comparação de desempenho, que chamamos de métricas. As métricas coletadas são relacionadas geralmente à velocidade, disponibilidade e precisão. Desempenho de rede é medido através de sua velocidade, taxa de erros relaciona-se à precisão e disponibilidade é verificada com os pacotes enviados. São alguns exemplos de métricas que agregam valor na análise de performance de um sistema.

4. **Listar Parâmetros**

   Estabelecer os parâmetros que podem afetar a performance do sistema é a próxima tarefa. Uma lista particionada entre parâmetros de sistema e de carga deve ser criada. O primeiro inclui parâmetros de *hardware* e *software* como seus componentes que receberão o fluxo de dados. Já os parâmetros de carga são obtidos pela análise de tamanho e quantidade de dados por exemplo que pode afetar a performance do sistema. Essa separação de componentes é importante para ajudar na análise dos resultados separando o que é parte do sistema em termos de recursos e em termos de fluxo de informações.

5. **Selecionar Técnicas de Avaliação**

   As três principais técnicas de avaliação de performance são as de simulação, analítica e de medição. Existem diversos fatores ao se considerar qual técnica utilizar, esses fatores estão listados na Tabela 2.1 O principal fator é a fase atual do estágio de desenvolvimento do software, nesse caso a técnica de medição só é possível quando uma versão do sistema já existe pela necessidade de haver uma base de resultados de desempenho prévia.

   Avaliações de simulação são programas de software que simulam o comportamento de um sistema conforme as requisições são disparadas e vão sendo processadas atra-

Tabela 2.1: Critério para seleção de técnica de avaliação

Critério	*Modelagem Analítica*	*Simulação*	*Mensuração*
1. Fase do Projeto	Qualquer	qualquer	Prototipação
2. Tempo Requerido	Pouco	Médio	Variável
3. Ferramentas Necessárias	analistas	Scripts	Instrumentação
4. Custo	Baixo	Médio	Alto
5. Manutenção	Baixo	Médio	Alto

Fonte: (JAIN, 1990)

vés dos recursos do sistema. Esses modelos simulam trocas de estados do sistema, monitoram tempos de resposta e provêm respostas com alta precisão devido ao auxílio de tecnologias de ferramentas de performance para simulação de cenários e monitoramento de recursos, entretanto são de elevado custo por necessidade da obtenção de licenças para sua utilização. Por outro lado o custo de modelos analíticos é mais baixo visto que ferramentas de performance não são necessárias mas sim um engenheiro de performance com conhecimento do sistema a ser testado, informações de carga de dados e técnicas de teste.

Outro fator importante a ser considerado é o tempo necessário para implementação da técnica. Normalmente empresas trabalham com prazos curtos e a implementação de todos os scripts, configuração de monitoramento e execução dos testes acabam tomando mais tempo que a modelagem analítica, sendo está última a mais vantajosa para curtos prazos.

A manutenção se torna mais fácil em modelos analíticos por não envolver o desenvolvimento de *scripts* e codificação em diversas linguagens de programação como é o caso de modelos de simulação e mensuração. Modelos analíticos não envolvem tecnologias e a complexidade de plataformas para realizar o teste. Como dito, o conhecimento do engenheiro de performance e papel e caneta é o suficiente para o planejamento do teste de performance (JAIN, 1990).

As particularidades de cada projeto são responsáveis por direcionar a técnica escolhida. Fatores importantes como tempo de dedicação nas atividades de performance e orçamento são muito relevantes fazendo com que a técnica dependa disso também. É importante salientar que a combinação de duas ou mais técnicas fornecem um nível maior de precisão ao se complementarem (JAIN, 1990).

6. **Selecionar a carga de dados**

   A carga de dados, também chamada de *workload*, basicamente é a lista de serviços que disparam requisições no sistema. A técnica de modelagem é responsável pela seleção de carga do sistema, entretanto, o que é comum para todos os processos

é a representatividade da carga sendo o mais próximo da vida real. Para alcançar essa precisão o *workload* deve ser caracterizado para cada sistema a ser avaliado e reproduzido nos cenários a serem testados.

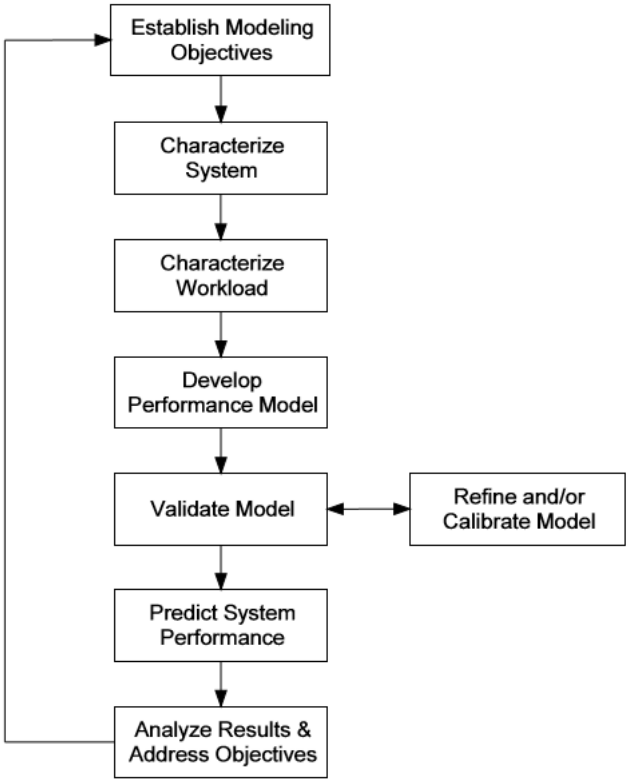
7. **Desenhar os experimentos**
   Com a lista de serviços que o sistema dispõe para operar em produção, quantidade de dados e caracterização do sistema físico, juntamente com os requisitos levantados devem-se montar os cenários de teste. A representatividade da vida real em ambiente de teste deve ser alcançada nessa fase. Forçar os componentes do sistema que mais podem afetar a performance, também já listados nos passos anteriores, é o objetivo também incluído nessa fase.
8. **Analisar e interpretar os dados**
   Os resultados obtidos podem variar com a mínima mudança de parâmetros de teste. Não se devem comparar apenas as médias obtidas. Os resultados devem ser observados e entendidos levando em consideração todas as variáveis envolvidas no sistema e as possíveis mudanças para que a avaliação tenha o objetivo alcançado de encontrar a melhor configuração possível para satisfazer as necessidades expressas na fase de definição de objetivos da avaliação de desempenho.
9. **Apresentar os resultados e iniciar novamente se necessário**
   O último passo de todos os projetos de teste de performance é comunicar os resultados para todos os membros do time de projeto. Apresentar os resultados de maneira fácil de entender para pessoas não tão familiarizadas com tecnologias e termos técnicos é de extrema importância. Neste passo é possível que seja necessária uma ou mais mudanças nos passos anteriores a fim de ajustar os testes e obter resultados melhores.

No modelo descrito previamente obtemos uma série de etapas a serem executadas durante um projeto em uma abordagem mais teórica do que prática. Os passos são bem descritos e possuem boa fundamentação teórica porém informações técnicas de como estabelecer *workload*, escolher tipos de teste a serem executados não são explicados com muitos detalhes. Modelos de performance adicionais mais técnicos são encontrados na literatura com etapas parecidas porém divergentes em alguns aspectos quando comparados ao modelo apresentado.

2.4.2 Modelo para Sistemas de Informação de Negócios

Um modelo de performance é uma apresentação abstrata do sistema que relaciona a carga de dados parametrizados com a configuração do sistema resultando na captura de métricas que vão determinar o desempenho do sistema em condições reais de uso (KOUNEV et al., 2012). Samuel Kounev propõe um modelo para sistemas integrados, que caracterizam grande parte das aplicações que necessitam de estimativas para seus níveis de performance e recomendações de otimização para atingir o melhor nível de qualidade possível.

Figura 2.3: Técnica baseada em Modelagem



Fonte: (KOUNEV et al., 2012)

Na Figura 2.3, temos um modelo em formato de fluxograma. Cada passo do modelo determina uma etapa do teste de performance desde a identificação dos objetivos até os possíveis refinamentos após a análise dos resultados coletados.

1. **Estabelecer Objetivos de teste**

   Deve-se especificar os objetivos de teste através da motivação do cliente ao exigir o teste de desempenho, essa informação será útil na escolha do tipo de teste a ser executado

2. **Caracterizar o sistema**

   O sistema deve ser descrito em termos de *hardware* e *software* bem como seus principais componentes.

3. **Caracterizar Carga de Dados**

   A carga de dados deve ser caracterizada e modelada a fim de projetá-la no sistema a ser verificado. A caracterização de *workload* é a etapa mais importante do fluxo, onde os dados que passam pela aplicação devem ser descritos de maneira qualitativa e quantitativa. (KOUNEV et al., 2012).

4. **Desenvolver Modelo de Performance em Casos de Teste**

   Os cenários são criados fazendo a escolha de técnicas de teste que combinam carga coletada e caracterização do sistema juntamente com os objetivos de teste previamente coletado.

5. **Validar o Modelo e Refinar se Necessário**

   Após a criação do modelo temos a etapa de validação, onde comparam-se as métricas obtidas com a criação do modelo com os resultados de medições obtidos em um sistema real podendo ser hospedado em um pequeno ambiente de teste. Além disso é possível realizar o ajuste do modelo caso os números obtidos na modelagem não atinjam um nível de precisão aceitável, levando, então, o modelo a ser calibrado e ajustado para uma nova validação.

6. **Prever Desempenho do Sistema**

   Comparam-se os valores encontrados aplicando a carga modelada no sistema descrito com os resultados obtidos em produção ou com ferramentas de monitoramento

7. **Analisar Resultados e Endereçar Objetivos**

   Por fim, os resultados são analisados e os passos posteriores envolvem discussões com outros times envolvidos no projeto a fim de endereçar possíveis mudanças e otimizações a serem implementadas visando ganho de desempenho.

Com o modelo descrito questões como o máximo nível de carga o sistema conseguirá aguentar em produção, o tempo médio de resposta, vazão e utilização de recursos são respondidos (KOUNEV; BUCHMANN, 2003). Ainda segundo Buchmann e Kounev, a possibilidade de definir a quantidade de recursos de *hardware* também é alcançada pelo modelo. Os problemas no modelo são pela falta de detalhismo dos passos. Eles são explicativos e com níveis de entendimento fáceis para profissionais da área, entretanto falta a parte teórica ja encontrada nos passos descritos na Seção 2.4.1. Todavia, o modelo citado

na Figura 2.3 é composto por passos muito próximos da realidade de grandes empresas que lidam com sistemas distribuídos e necessitam desse tipo de teste de desempenho.

2.4.3 Modelo Técnico - Neotys

O último modelo apresentado é feito pela empresa Neotys, fabricante da ferramenta para testes de performance Neoload, a qual usaremos para validar o modelo proposto neste trabalho. Na Figura 2.4 encontra-se o fluxo do modelo da Neotys cuja abordagem é de alto nível, visando a prática na indústria e não oferecendo tantos detalhes teóricos.

Figura 2.4: Modelo Técnico de Teste de Performance

Fonte: (ALFANO, 2018)

Os passos descritos acima também são facilmente explicativos para profissionais com experiência em testes de performance. Alguns passos serão detalhados e utilizados no modelo proposto de trabalho, mas de forma resumida eles são explicitados da seguinte maneira:

1. **Identificar Ambiente de Teste**

   O engenheiro de performance deve identificar os componentes de *hardware* e a escalabilidade dos mesmos em relação aos componentes do ambiente de produção a fim de saber a representatividade do ambiente de performance e realizar os ajustes de carga com o objetivo de obter resultados mais próximos possíveis do que os encontrados em produção.

2. **Definir Critérios de Performance**

   Discussões com os desenvolvedores e clientes acerca das funcionalidades mais críticas em termos de desempenho, os fluxos mais utilizados pelos usuários finais e os aspectos mais importantes a serem verificados em termos de desempenho a fim de se obter objetivos para os resultados serem reportados como satisfeitos ou não satisfeitos de maneira mais precisa.

3. **Planejamento de Teste**

   Ao verificar com as partes interessadas do projeto os principais aspectos a serem

testados e comportamentos esperado em produção deve-se pensar em tipos de teste e técnicas de verificação de desempenho a serem utilizadas. As técnicas mais importantes serão aplicadas no modelo proposto e são definidas nesse momento do teste de performance.

4. **Configurar ambiente de teste**

   Nessa fase o time de performance trabalha junto ao time de desempenho para garantir que o mesmo código que existe em produção ou que será entregue em ambiente de produção esteja no ambiente de performance. Isso assegura que, guardas as proporções entre ambientes, o que está sendo verificado no ambiente de performance será replicado aos usuários finais após todo o processo de desenvolvimento do sistema.

5. **Implementar design de teste**

   Os testes planejados são implementados em formato de *scripts* em ferramentas próprias de teste de performance em suas respectivas linguagens de programação

6. **Executar testes**

   Os *scripts* desenvolvidos são executados utilizando os dados e configurações do ambiente de performance com as cargas e cenários discutidos anteriormente com os times de regra de negócio e desenvolvimento.

7. **Analisar, otimizar e retestar**

   Com o auxilio da ferramenta de teste de performance, as métricas coletadas em termos de tempo de resposta e utilização de recursos são analizadas e discutidas a fim de encontrar oportunidades de melhorias ou evidências de que o sistema sob verificação está cumprindo todos os requisitos de performance solicitados pelo cliente.

Apresentados os modelos de teste de performance encontrados na literatura, observam-se prós e contras em seus usos na indústria. O modelo de Jain como já foi discutido é bastante embasado na teoria e sistemático, possui um nível de detalhe muito importante nos passos a serem seguidos, porém nem todos os passos são implementáveis no ambiente profissional por se tratar de processos mais abstratos e sua implementação por desenvolvedores necessitar de processos mais técnicos. O modelo de Kounev já trata de uma abordagem mais próxima do mercado de trabalho porém nota-se a falta de mais detalhes referentes às técnicas de teste e construção de artefatos de teste como cenários e *test cases*. No último modelo apresentado, da Neotys, é percebido um nível mais técnico, principalmente por se tratar da fabricante de uma ferramenta de teste de performance,

porém, ao contrário do primeiro modelo, falta a parte teórica e com mais detalhes no que se refere ao processo de traçar objetivos de teste e seleção de métricas. Dessa forma, para um profissional novo no ramo da engenharia de performance, existe a necessidade de uma metodologia que englobe tanto aspectos técnicos como projeto de cenários de performance e caracterização de sistemas a serem testados como aspectos teóricos e estratégicos que envolvem a definição de objetivos de teste de performance e análise de requisitos em um único processo modelado. A tabela 2.2 mostra a comparação entre os modelos analisados e os passos propostos no modelo construído para a aplicação do teste de performance. Lembrando que todos os passos foram acrescidos de experiência de profissionais com vivência na área de teste de performance.

Dados os prós e contras de todos os modelos vistos propõe-se a construção de um modelo que integre todos os passos necessários para que o processo completo de um profissional sem experiência na área de teste de performance consiga exercer todas as atividades necessárias. Os principais passos de cada modelo e mais a experiência de profissionais na engenharia de performance dão origem ao modelo que é proposto neste trabalho.

Tabela 2.2: Comparativo entre modelos

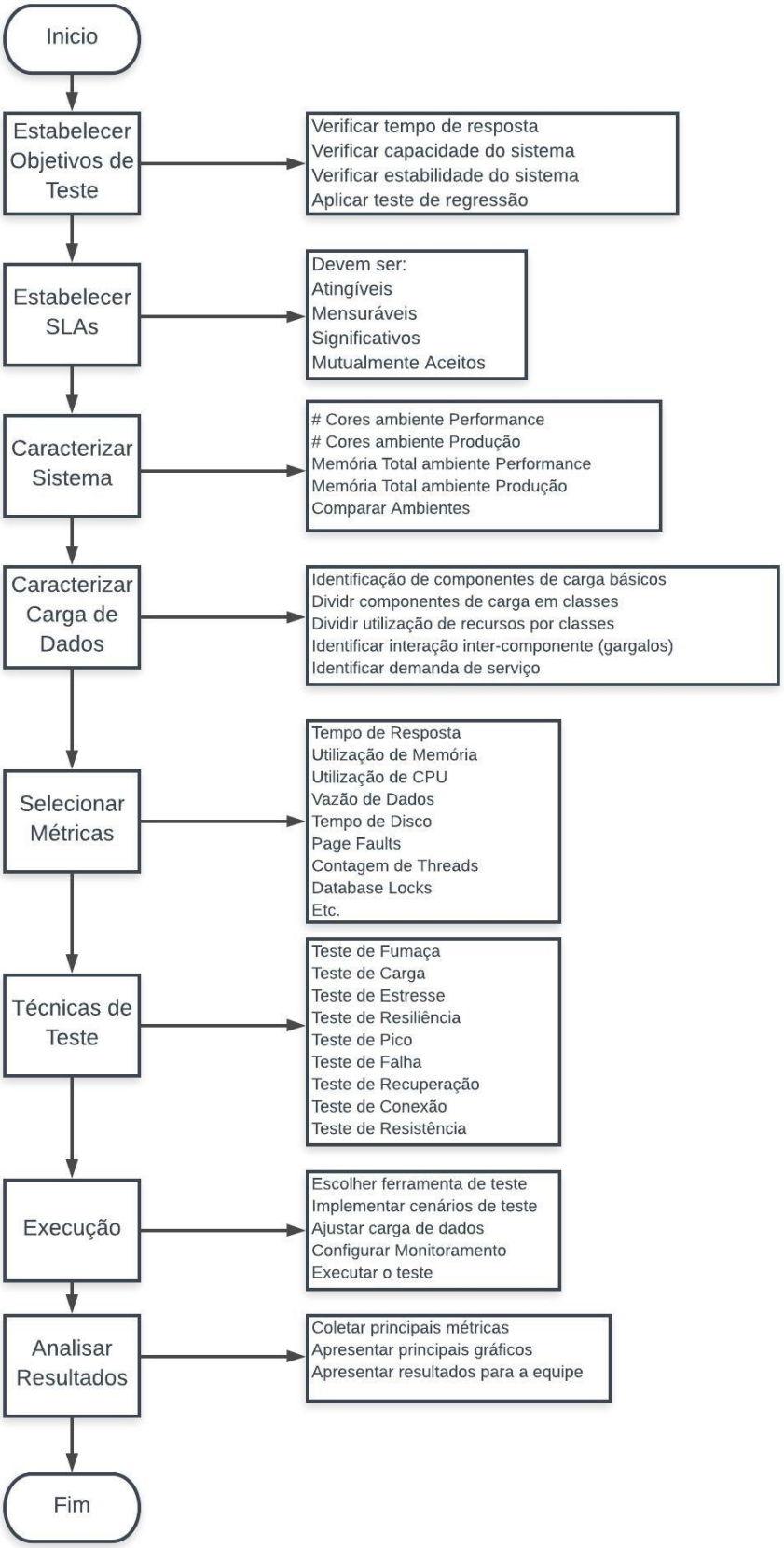
Passos	*Jain*	*Kounev*	*Neotys*
Estabelecer Objetivos	Não	Sim	Não
Estabelecer SLA's	Não	Não	Sim
Caracterizar Sistemas	Não	Sim	Sim
Caracterizar Carga de Dados	Não	Sim	Sim
Selecionar Métricas	Sim	Não	Sim
Escolher Técnicas de Teste	Sim	Não	Não
Execução	Não	Não	Sim
Apresentação de Resultados	Não	Sim	Sim

3 MODELO PARA PLANEJAMENTO DE TESTE DE PERFORMANCE

Na indústria, considerando as empresas de médio e grande porte que possuem em seus projetos de desenvolvimento de sistemas o teste de performance, existe a fase de planejamento e estimativa de tamanho dos projetos em termos de *timeline* onde é solicitada a alocação dos times, também chamado de engajamento. Na empresa abordada durante o estudo de caso a ser discutido no Capítulo 4, devido o fato do time de performance no sentido de recursos humanos ser proporcionalmente muito menor que os times de desenvolvimento e teste funcional, nem todas as aplicações desenvolvidas dentro da empresa são cobertas por teste de performance. Desta forma, para aplicações críticas em relação ao alto volume ou necessidade alta de garantia na estabilidade, é realizado o engajamento formal do time de performance para que o planejamento e execução seja realizado para o projeto em questão.

A proposta desse trabalho é o modelo representado na Figura 3.1 com todos os passos previamente determinados.

Figura 3.1: Modelo Proposto



Modelagem Teórico-Técnica para testes de performance

Com a aplicação de todo o processo e a execução de todos os passos detalhados é possível que todas as atividades de análise de desempenho sejam cumpridas. Embora os passos sejam similares aos passos dos modelos descritos na Seção 2.4 o modelo proposto incorpora métodos técnicos para a escolha de cenários adequados para execução de teste de performance baseados em objetivos de performance e análise de requisitos que são obtidos de outras abordagens de modelos teóricos e sistemáticos. Tendo em vista o crescimento da necessidade de teste de performance, falta de profissionais com experiência no ramo e alta rotatividade de funcionários em empresas de grande escala, a modelagem construída contém os principais aspectos necessários que não são encontradas em um único processo para novos engenheiros de performance, mas sim com a integração de metodologias teóricas e técnicas além da experiência de profissionais da área de engenharia de performance com suas viviências em projetos na indústria.

As atividades mostradas na Figura 3.1 são detalhadas a seguir:

3.1 Estabelecer Objetivos de Teste

Nesta fase o engenheiro de performance deve identificar a necessidade do cliente para realizar o teste de performance. Devido ao grande número de aplicações existentes no mercado e às inúmeras possibilidades de ciclos de vida dos sistemas de *software*, diversas podem ser as razões que motivam o engajamento de times de análise de desempenho. E dependendo do objetivo do teste, o engenheiro de performance pode guiar os próximos passos com mais facilidades. As técnicas de avaliação de desempenho, métricas a serem coletadas e carga de dados a serem inseridas no sistema vão depender do objetivo proposto para o teste de performance. As principais motivações para o teste de desempenho são as seguintes:

1. **Verificar Tempo de Resposta da Aplicação:** Normalmente se trata do principal objetivo para qualquer teste de performance. Porém a definição deste objetivo torna clara para o engenheiro a necessidade de estabelecer futuramente cenários comuns de utilização dos usuários finais e coletar tempos de resposta esperados pelo cliente, também chamados de SLA *(Service Level Agreement)* onde darão respostas claras sobre a satisfabilidade dos serviços oferecidos pela aplicação em termos de unidade de tempo.
2. **Verificar Capacidade do Sistema:** Neste caso o objetivo é a verificação de capa-

cidade máxima de carga que a aplicação é capaz de resistir. É fundamental nessa abordagem a futura identificação dos principais componentes de hardware do sistema e a correta inserção de carga de dados que serão utilizados em produção a fim de que situações normais de uso sejam simuladas em larga escala, aumentando cada vez mais até que a aplicação pare de responder, encontrando assim sua capacidade máxima para uma determinada quantidade de carga por unidade de tempo.

3. **Verificar a estabilidade da aplicação:** Nesta abordagem tanto os cenários de teste quanto o volume a ser testado já é previamente conhecido. É o caso de simular um determinado fluxo na aplicação sob também um volume de dados previamente determinado. Avaliar o tempo de resposta e também métricas de *hardware* são comuns nesta abordagem visto que as condições do ambiente já são previamente conhecidas.
4. **Teste de Regressão:** Usualmente solicitado durante a inserção de novas funcionalidades ou atualizações de aplicações. Trata-se de um teste onde previamente já houve a verificação de desempenho e deseja-se confirmar que quaisquer atualizações ocorridas não inseriram problemas de desempenho no sistema sob verificação. Neste caso também cenários e cargas são previamente conhecidos visto que trata-se da replicação de um teste de desempenho já realizado.

3.2 Estabelecer SLA's

SLA *(Service Level Agreement)* é um acordo contratual entre uma empresa que fornece um serviço e o cliente, compromissando que o cliente receberá o solicitado no acordo. Em TI, e mais precisamente na engenharia de performance, SLA é o conjunto de medidas aceitáveis em termos de tempo de resposta ou utilização de recursos de um sistema de *software* relativos a serviços oferecidos por uma aplicação, transições entre páginas e todos os comportamentos de um sistema que devem ser medidos quantitativamente em um teste de performance. SLA beneficia o cliente provendo um critério de classificação e o protegendo de um serviço ruim. Por outro lado também beneficia o provedor do serviço, no caso o engenheiro de performance que terá por objetivo verificar o cumprimento do SLA, possibilitando um caminho para garantir que as expectativas estão sendo cumpridas. Desta maneira, o teste de performance julgará corretamente os resultados obtidos e possibilitará ao provedor de serviços, neste caso o time de desenvolvimento,

a melhorar a qualidade dos serviços se necessário.

Para aplicações já existentes no mercado em produção e com SLA's previamente especificados, ao engenheiro de performance é necessária a informação destes tempos de resposta a fim de que os próximos testes de performance solicitados por qualquer modificação no sistema sejam guiados através destes números tendo como base a melhora, piora ou estabilidade de tais métricas nos resultados das execuções.

SLAs devem cumprir alguns princípios:

- Atingíveis
- Mensuráveis
- Significativos
- Mutualmente aceitos

Níveis de serviço devem ser atingíveis em termos de recursos devem possuir um nivel de satisfabilidade onde respostas sejam por utilização de recurso ou tempos de resposta sejam capazes de responder se tal nível de serviço foi atingido ou não. Devem ser quantificados. Um desafio grande na análide de desempenho é o levantamento de SLAs pois termos como o serviço ser rápido ou instantâneo não são atingíveis em termos de teste de performance, o que leva ao segundo caso, devem ser capazes de ser mensurados através de métricas a serem selecionadas nas próximas fases do modelo proposto. Unidades de tempo ou de *hardware* são exemplos de métricas a serem coletadas a fim de estabelecer a satisfabilidade ou não dos SLAs levantados. Devem possuir significância para o sistema a ser testado onde obrigatoriamente devem fazer parte dos processos principais de casos de teste com o objetivo de prover maior confiabilidade nos resultados do teste de performance. Por último mas não menos importante devem ser aceitos tanto pelos desenvolvedores e engenheiros de performance como pelos clientes.

3.3 Caracterizar o Sistema

Em boa parte das empresas que realizam teste de performance, por questões financeiras, não se dispõe de um ambiente de testes exatamente igual ao ambiente de produção. Embora existam situações onde testes de carga são realizados no próprio ambiente de produção durante horários alternativos aos horários onde usuários finais estão utilizando o sistema, não é considerada uma boa prática realizar testes onde é possível quebrar o sistema no próprio ambiente onde a aplicação está hospedada. Por essas razões comu-

mente os times de teste se utilizam de ambientes secundários que simulam a configuração utilizada em produção porém em menor escala. Por razões da necessidade do teste de desempenho validar comportamentos diretamente relacionados às configurações de *hardware* é muito importante que o engenheiro de performance faça uma análise criteriosa de ambos os ambientes, de performance e produção, para que se haja a correta proporção de todas as principais características entre os recursos físicos que serão submetidos a altas cargas de dados no ambiente de teste para a precisa avaliação de como se dará o comportamento do sistema no ambiente com as configurações de produção. Tal proporção é feita medindo as quantidades disponíveis de memória e CPU em ambos os ambientes. Devido a possibilidade de muitos detalhes como *threadpool*, tamanho de fila, etc serem diferentes, abstrai-se aos componentes mais importantes no recebimento de carga para se realizar a proporção, ou seja, soma-se os valores totais de ambos os ambientes para a determinação de equivalência entre performance e produção.

A caracterização do sistema será realizada buscando os principais componentes físicos dos quais a infraestrutura será composta. Detalhes mais específicos e que dependam muito de determinados sistemas operacionais não serão considerados neste modelo. O objetivo desta fase é buscar o maior número de componentes em comum independente da plataforma onde o sistema estará hospedado e a quantidade unida à capacidade de cada componente. Os principais componentes a serem caracterizados e detalhados quantitativamente são:

- **Utilização de Processador:** Quantidade de recursos que o processador gasta executando processos
- **Utilização de Memória:** Quantidade de memória física disponível para processamento
- **Tamanho de de Disco:** Quantidade espaco em disco que se passará ocupado executando requisições de escrita ou leitura
- **Largura de Banda:** Quantidade de bits por segundo utilizada por uma interface de rede
- **Memória Virtual:** Quantidade de memória virtual utilizada
- ***Connection Pooling*:** Cache de coneções de banco de dados para execuções de comandos de outras camadas com a camada de dados

Sabendo a configuração de recursos do sistema em aspectos físicos, o engenheiro de performance conseguirá mapear a previsibilidade do comportamento do sistema ao

aplicar a carga de dados que é discutida no próximo passo.

3.4 Caracterizar Carga de Dados

A caracterização de carga de dados é o processo de descrever essa carga de maneira quantitativa e qualitativa (KOUNEV et al., 2012). Um dos principais erros na avaliação de desempenho de sistemas de *software* é a utilização de carga não representativa, diferindo da utilização real do sistema em produção (JAIN, 1990). O processo de caracterização normalmente envolve uma série de passos que irão prover parâmetros de entrada para o modelo de performance ser executado.

- **Identificação de componentes de carga básicos:** Componentes básicos são referenciados por unidades de carga que entram em um sistema através de uma fonte externa, como por exemplo, *requests* HTTP, chamadas de *webservices*, transações de banco de dados, etc. O sistema que será testado é que informará o nível de detalhe dos componentes de carga
- **Componentes de carga básicos divididos em classes:** Componentes com características similares são subdivididos em classes de acordo com sua natureza. A divisão de classes fica a critério dos objetivos de teste e da facilidade como os dados serão coletados. Por exemplo, componentes de carga de bancos de dados como consultas, e transações internas dentro de uma base de dados podem ser agrupados em uma classe para melhor representatividade das métricas de banco ao final de uma análise de desempenho.
- **Componentes de sistema e seus recursos utilizados por classe de carga:** Identificam-se quais componentes físicos são utilizados para cada classe de dados. Utilizando o exemplo de classe de dados que fluem por uma base de dados do sistema sob avaliação, são identificados aspectos físicos do banco de dados como espaço de tabelas, disco e memória do servidor de banco de dados para análise quando a carga estiver fluindo para essa classe.
- **Interação inter-componentes identificadas:** Quando mais de uma aplicação estiver sendo verificada no teste devem-se identificar os componentes e tipos de dados que fluirão de um sistema para outro a fim de identificar possíveis gargalos e componentes físicos responsáveis por essas situações.
- **Identificar demanda de serviço :** O objetivo principal é especificar a média total

da quantidade de carga de dados por classes. Dessa maneira, descobrindo o conjunto total de componentes geradores de carga e seus recursos utilizados, podemos estimar, com o fluxo total de dados separados por classes, a quantidade e intensidade que irá onerar cada tipo de recurso.

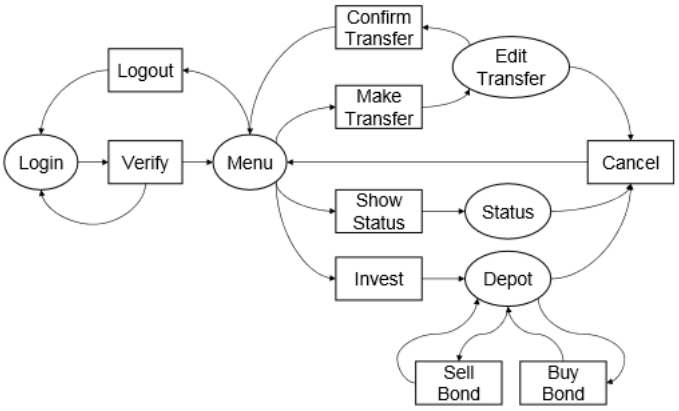
- **Ajustar volume em relação à equivalência do ambiente de performance:** Considerando o sistema caracterizado em termos de capacidade do ambiente de performance, deve-se ajustar a quantidade total de volume definido e classificado para que a correta carga de dados seja implementado na infraestrutura dedicada de performance com o objetivo de verificar o comportamento correto com os dados e ambiente de produção.

Um dos principais desafios na fase de caracterização de carga é a obtençao de valores para os serviços. Em sistemas que estão sendo criados pela primeira vez essas informações dependem de previsibilidade do cliente ou de análises de mercado por exemplo, o que causa uma confiabilidade menor nos dados a serem obtidos. De uma forma geral todas as técnicas para obtenção de dados em produção a fim de atingir o maior nível de precisão possível requerem a utilização de ferramentas de monitormanto para a coleta deste tipo de dados, o que também requer recursos adicionais aumentando o custo.

Para caracterização de carga de dados em aplicações *web* é também muito utilizada a metodologia orientada a formulários, onde a aplicação é descrita como uma máquina de estados tipada e bipartida que consiste de páginas, ações e transições entre as páginas (LUTTEROTH; WEBER, 2008). Neste contexto, páginas são entendidas como uma série de telas apresentadas ao usuário final. As informações contindas nessas telas variam de acordo com sua funcionalidade, onde cada tela possui uma série de formulários onde tais formulários podem possuir uma série de campos onde informações são inseridas para posteriores transições entre páginas. Essas transições invocam ações no lado do servidor que por sua vez processa a informação submetida e retorna uma nova tela para o lado do cliente em formato de resposta.

Modelos orientados a formulários podem ser visualizados usando grafos direcionados onde as páginas web são representadas por bolhas e as ações são representadas por caixas enquanto as transições entre páginas através de ações são representadas pelas arestas direcionadas. Formulários são grafos bipartidos direcionados, ou seja, em cada caminho páginas e ações ocorrem alternadamente. Esse particionamento cria uma distinção entre sistema e usuários onde cada transição ação-página representa um comportamento de sistema e cada página-ação representa um comportamento de usuário como mostrado

Figura 3.2: Exemplo de Modelo Orientado a Formulário

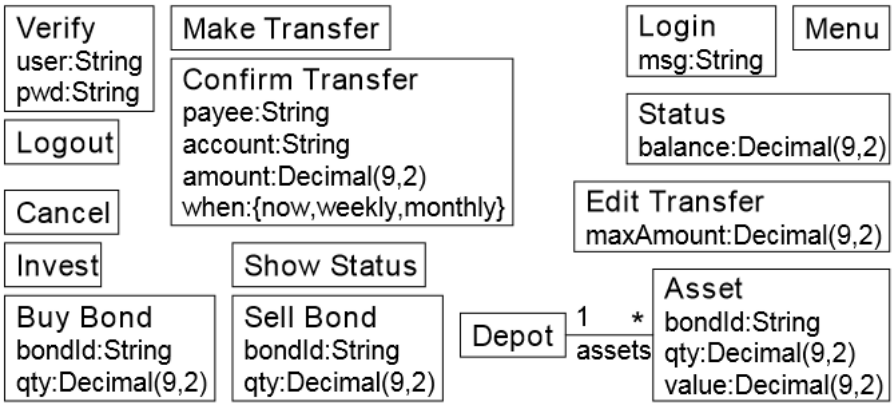


Fonte: (LUTTEROTH; WEBER, 2008)

na Figura 3.2 representando um formulário de um sistema simplificado de banco *online*.

Utilizando a metodologia de formulário aplicado no processo de caracterização de carga pode-se especificar também tipos de mensagem para todas as páginas e ações. Essa abordagem é muito útil na definição de carga por especificar a estrutura de dados que pode ser submetida através do formulário modelado. A representação dinâmica das informações que fluem no sistema são de grande importância para identificar partes específicas do sistema onde o fluxo de dados tende a aumentar através das mensagens enviadas (LUTTEROTH; WEBER, 2008).

Figura 3.3: Exemplo de Modelo de Mensagens



Fonte: (LUTTEROTH; WEBER, 2008)

Na Figura 3.3 o modelo bancário é representado por um modelo de mensagens. As ações representadas na Figura 3.2 por retângulos estão do lado esquerdo e as páginas do sistema, representadas por elipses, também na Figura 3.2, estão no lado direito. Cada elemento no modelo de mensagens, quando invocados apenas por *links*, não possuem ne-

nhum dado a ser trafegado entre páginas através de ações, porém quando há formulários nas páginas ou qualquer método de inserção de informações, todos os dados são informados nas respectivas caixas. Essa abordagem é de grande utilidade para o engenheiro de performance por prover visualmente os dados quantificados em cada parte do sistema, ajudando a caracterizar o fluxo de informações trocados entre componentes da aplicação.

3.5 Selecionar Métricas

Para cada objetivo de análise de performance um grupo de métricas deve ser escolhido e avaliado. A lista de métricas a serem estudadas vão variar com o tipo de serviço e o tipo de resposta que os serviços implementados tendem a oferecer. Sistemas possuem serviços que podem ser respondidos corretamente, incorretamente ou não serem respondidos. Para os serviços que possuem respostas corretas selecionamos três tipos de métricas: Métricas de Tempo, Taxa e Utilização de recurso. Para os serviços que respondem incorretamente é importante para o engenheiro de performance descobrir a taxa de erros por intervalo de tempo e para serviços que não respondem a duração de tempo entre o envio da informação e a resposta do sistema da falha da requisição além de também coletar as taxas de falhas por intervalo de tempo.

3.5.1 Tempo de Resposta

Tempo de resposta é definido como o intervalo entre o início da requisição do usuário e o final da resposta submetida pelo sistema (JAIN, 1990). Tempos de resposta são usualmente medidos em segundos mas podem variar de acordo com o perfil da aplicação podendo ser essa aplicação composta de serviços extremamente rápidos com tempo de resposta em milissegundos (ms). Tempos de resposta normalmente não mudam com cargas baixas mas crescem linearmente com o correspondente crescimento de volume de dados no sistema.

3.5.2 Vazão

Vazão é definida pela taxa de requisições por unidade de tempo que está sendo implementada em um sistema sob teste. Cada recurso a ser testado e seus respectivos

volumes vão indicar a unidade de medida na qual será avaliada no teste. *batch jobs* são medidos por *jobs* por segundo, utilização de CPU é medida em MIPS *(Millions of Instructions per Second)*, etc. Assim como tempo de resposta, a vazão de um sistema cresce com o aumento do volume inserido, porém em determinado instante essa vazão interrompe o crescimento fazendo com que o sistema atinja sua **capacidade nominal**, indicando ser o máximo de quantiddade de informação por unidade de tempo que a aplicação é capaz de processar. Pelo motivo de sistemas necessitarem quantidades muito acima das aceitáveis de informação para chegarem em sua capacidade nominal, na engenharia de performance utiliza-se a **capacidade utilizável** onde cargas altas porém aceitáveis em produção são inseridas fazendo com que a vazão torne-se estável. Tais vazões são importantes de serem medidas pois a taxa da capcidade utilizável pela capacidade nominal provê a **eficiência** do sistema.

3.5.3 Utilização de Recursos

A utilização de recursos é medida pela fração de tempo que o recurso está ocupado processando requisições e dividida entre tempo ocupado e tempo ocioso de recurso, onde não há um período que um dado recurso não está operando nenhum tipo de requisição

- **Utilização de CPU:** Quantidade de CPU utilizada durante o processamento de informações. Pode ser medida em função da capacidade de CPU ou em porcentagem como a maioria das ferramentas de performance realiza a medição
- **Utilização de Memória** Quantidade de memória utilizada durante o processamento de informações. Assim como a utilização de CPU é expressa usualmente em porcentagem de utilização
- **Contagem de Threads** A saúde da aplicação pode ser medida pelo número de threads que estão rodando em determinado momento e permanecendo com o estado ativo. Altos índices de carga ocasionam threads de ficarem presas não sendo capazes de processar as informações, o que ocasiona diminuição no número total de *threads* disponíveis e piores índices de performance nos testes.
- ***Page Faults*** A taxa geral de *page faults* que são processadas pelo processador, esta taxa gera degradação de performance quando o disco deve ser acessado para a coleta de informação e causa o consequente aumento no tempo de resposta para a requisição de origem.

Tempo de Disco Quantidade de tempo gasto em disco para a escrita e leitura de informações. É medido em função do tempo que podee ser em milisegundos e segundos, variando conforme o tipo de aplicação sob teste.

Database Locks É importante o monitoramento do tempo entre as tabelas de bases de dados serem trancadas para possibilidades de otimizações em casos de má performance nessa métrica e uma aplicação com grande movimentação em tabelas de bases de dados como edições e inserções.

Existem uma série de outras métricas a serem consideradas nas plataformas dispiníveis no mercado. Cada ferramenta de performance é capaz de coletar um determinado número de métricas dependendo de sua capacidade. O Neoload, ferramenta utilizada neste trabalho, oferece a possibilidade de se buscar uma série de métricas para Windows e Linux principalmente, o detalhamento dessas métricas junto com descrições de possíveis soluções e algumas indicações de gargalos podem ser encontrados no Apêncie A e Apêndice B

3.6 Selecionar Técnicas de Teste

A etapa de seleção de técnicas de teste é uma das mais importantes de todo o processo de análise de desempenho. Neste momento o engenheiro de performance tendo em mente os objetivos de teste, identificação do ambiente de performance e volume de dados inicia a etapa de selecionar os tipos de teste que serão executados. As estratégias apresentadas no modelo podem ser testadas em diferentes ciclos de vida do processo de teste de performance, e para testes de execução de *scripts* são necessárias ferramentas específicas de teste de desempenho. As principais técnicas de teste e suas motivações e explicações são as seguintes:

3.6.1 Teste de Fumaça

Geralmente é o primeiro cenário a ser projetado e executado pelo engenheiro de performance. Trata-se de um cenário com uma carga muito abaixo da estabelecida para volumes normais com a finalidade de uma execução simples garantindo que o sistema a ser testado está funcionando bem em termos de *scripts* desenvolvidos e funcionalidades do sistema sendo cumpridas com sucesso. A prática de realizar um teste de fumaça,

também chamado de *smoke test*, evita que sejam realizadas cargas complexas no sistema ou variadas execuções para somente depois descobrir que o fluxo básico do sistema pode não estar sendo cumprido. Por padrão um sistema sob teste (SUT - *System Under Test*) nesse tipo de cenário deve cubrir todos os componentes de *hardware* que serão utilizados em execuções futuras.

Usualmente é utilizado 10% do volume total esperado em produção para uma execução desse cenário. Os resultados obtidos na execução do *smoke test* oferecem uma visão estrutural em termos de utilização de *hardware* e tempos de resposta. Além disso, consegue validar o correto funcionamento dos *scripts* desenvolvidos pelo engenheiro de performance.

3.6.2 Teste de Carga

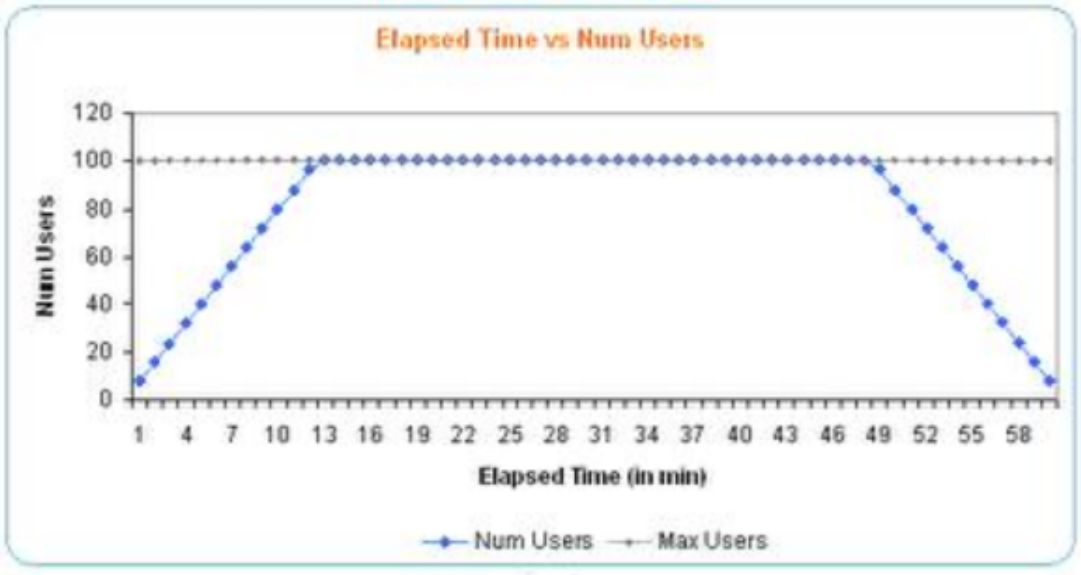
Nesta fase o volume de dados é ajustado o mais próximo possível do volume previsto em produção com o sistema disponível para utilização dos usuários finais. As cargas de dados identificadas devem ser atribuídas aos corretos serviços ou requisições que o sistema vai executar e seus recursos físicos devem ser monitorados durante o teste. Um ponto adicional a ser considerado nesse tipo de teste é o tempo de execução, que vai levar em conta a utilização do sistema em produção, o teste pode ser realizado simulando apenas uma hora de um comrpotamento normal ou também um dia útil inteiro caracterizando um teste de cerca de 8 horas de duração. Testes de carga com curta duração não são efetivos pelo fato do sistema não conseguir refletir as consequências de um volume considerável de dados em questão de minutos.

Outra consideração a ser levada no teste de carga é a rampa de usuários crescendo e decrescendo. Nesse cenário o engenheiro de performance deve considerar usuários iniciando suas atividades na aplicação a ser testada de maneira não simultânea, com o sistema sendo exigido aos poucos até completar o volume desejado e se manter pelo período de tempo considerado no levantamento de requisitos. Para a rampa de usuários decrescente, é importante ser incluída no teste de carga para o correto monitoramento de liberação de recursos do sistema. O mínimo de tempo aceitável para a carga máxima em um teste de carga é de 30 minutos com 30 minutos adicionais de rampa crescente e decrescente. Na Figura 3.4 é possível ver um exemplo de teste de carga com a rampa crescente e decrescente. Mesmo com os tempos de rampa não correspondente aos utilizados normalmente em produção o comportamento da carga de dados é mostrado na imagem.

Existem aspectos técnicos a serem avaliados e que motivam a inclusão dessa técnica em qualquer avaliação de desempenho. Estabelecer uma base de teste para futuras execuções está entre elas. Além disso a detecção de problemas na concorrência de usuários executando atividades semelhantes, erros de funcionalidade sob volume elevado e distribuição de recursos do sistema com múltiplos acessos por período de tempo elevado.

As entregas para o cliente com teste de carga são as validações dos SLAs previamente estabelecidos, validar possibilidades de melhoria em tempos de resposta no sistema considerando todo o fluxo de dados e a avaliação da capacidade corrente do sistema visando a necessidade ou não de mudanças nas plataformas utilizadas.

Figura 3.4: Exemplo de teste de carga



Fonte: (CIGNITI, 2011)

3.6.3 Teste de Estresse

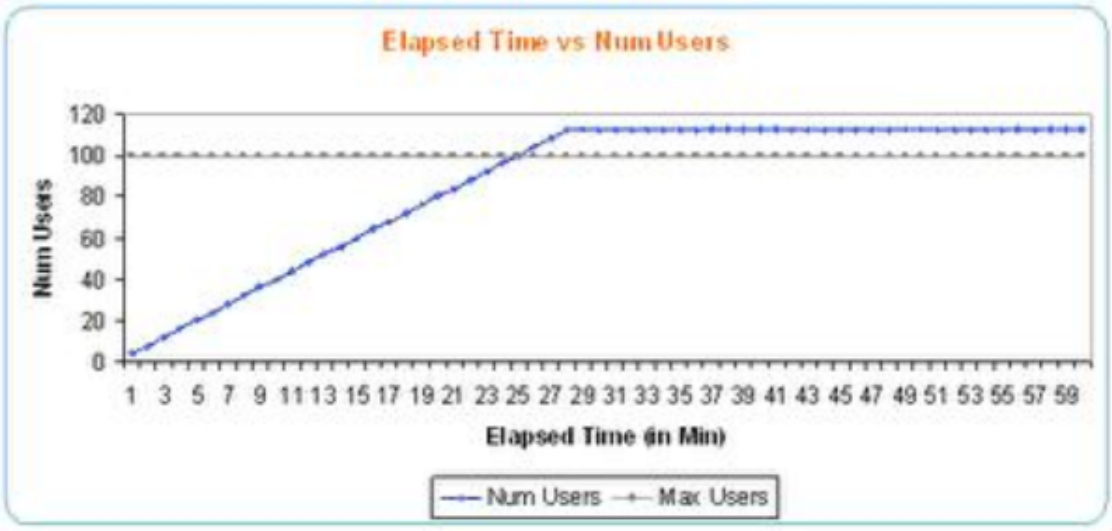
O teste de estresse é utilizado para avaliar a capacidade do sistema operar sob um volume maior do que o esperado em produção. O volume acima pode ser considerado tanto em termos de usuários concorrentes quanto de aumento de volume em transações normais. É boa prática separar os cenários e durante o teste de estresse aumentar apenas o número de usuários concorrentes operando normalmente as transações planejadas e em um teste adicional aumentar o volume de dados operado pelos usuários considerando os mesmos em quantidade normal para que os possíveis problemas no aumento do volume sejam mais facilmente identificados. Na Figura 3.5 há um exemplo de teste de estresse

com o número de usuários sendo aumentado em relação ao volume usado em um teste de carga. A rampa de usuários concorrentes cresce constante e linearmente atingindo uma taxa de usuários acima do planejado em volumes de produção.

É recomendado para o engenheiro de performance estressar o sistema inicialmente com um aumento de 10% no volume de dados esperado e nas execuções seguintes aumentar na mesma taxa até alançar uma carga alta porém viável em produção. O objetivo deste teste é validar o comportamento dos recursos em situações atípicas como por exemplo um site de vendas em datas de alto movimento como a *Black Friday*.

Para o engenheiro de performance é importante observar possíveis degradações em tempos de resposta dos serviços oferecidos pela aplicação. É necessário também validar a funcionalidade da aplicação sob alto volume e também se vulnerabilidades de segurança não são oferecidas pelo sistema em situações de alta quantidade de volume de dados.

Figura 3.5: Exemplo de teste de estresse



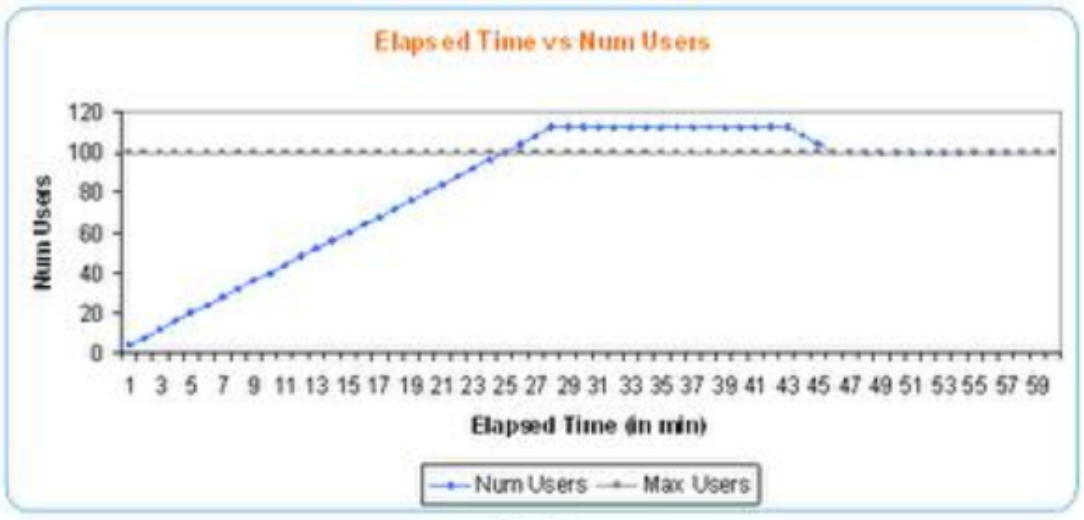
Fonte: (CIGNITI, 2011)

3.6.4 Teste de Resiliência

O teste de resiliência é executado a fim de determinar a capacidade do sistema de voltar ao estado inicial de volume normal após um período de stress. Neste cenário juntamos os dois primeiros testes listados porém com o volume normal esperado de dados implementado após um período de stress. É recomendado no mínimo 30 a 60 minutos para cada estado de carga a fim de se monitorar de forma mais efetiva os recursos utilizados e as métricas coletadas.

O objetivo principal é a confirmação de que diversos aspectos como tempo de resposta, vazão de dados, alocação de *threads* são capazes de voltar ao estado normal e operar com regularidade. A Figura 3.6 ilustra o comportamento de usuários concorrentes em função do tempo de teste para o cenário de teste resiliência com o número de usuários concorrentes crescendo até cerca de 10% acima do volume normal de dados causando um estado de estresse no sistema com o volume de usuários voltando ao estado normal utilizado no teste de carga. Assim como o Teste de Estresse, o aumento de dados pode ser realizado em relação à vazão sendo enviada ao sistema.

Figura 3.6: Exemplo de teste de resiliência



Fonte: (CIGNITI, 2011)

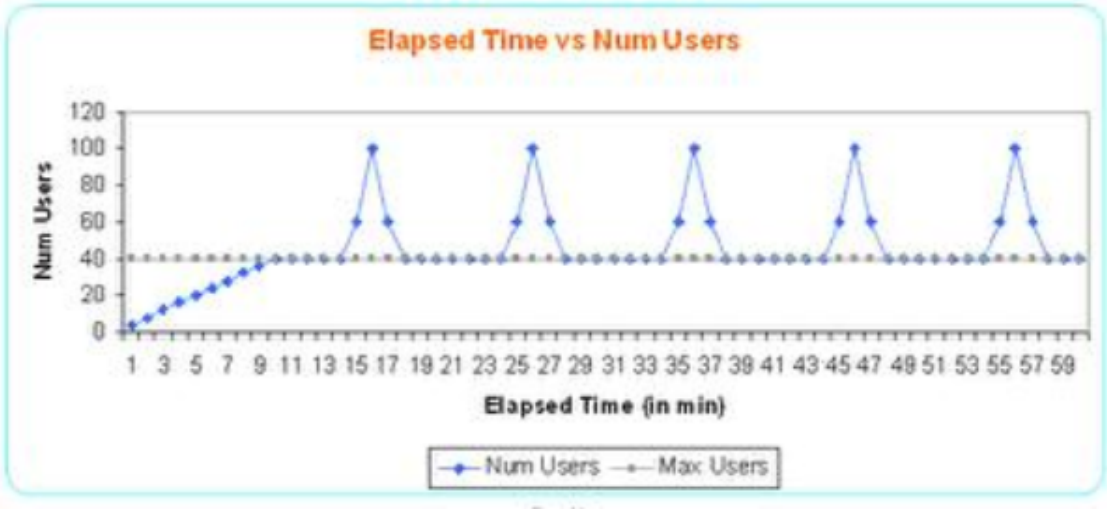
3.6.5 Teste de Pico

O teste de pico é executado em cenários onde a aplicação espera aumentos subtos de carga em intervalos muito curtos de tempo voltando ao seu estado natural. A carga pode ser aumentada em cerca de 20% a 30% por alguns segundos e depois voltar ao volume anterior por diversas vezes como mostrado na Figura 3.7. Com o objetivo de mitigar a investigação das causas para uma possível quebra no sistema, recomenda-se que a carga inicial se mantenha em cerca de 50% antes de iniciar os picos instantâneos, ou seja, aumento do volume por poucos segundos antes de baixá-lo novamente. Após as primeiras execuções realizadas com sucesso é possível aumentar o volume médio de dados da aplicação até o volume estabelecido para o teste de carga.

É o objetivo do engenheiro de performance avaliar a capacidade da aplicação em

controlar os recursos com alto volume em curto período de tempo ao invés de um volume de estresse sendo acrescido aos poucos e por muito tempo com o alto volume sendo mantido. Aplicações que itlizam memória virtual devem ser observadas pois em situações de picos instantâneos estouro de memória virtual é muito recorrente em aplicações que possuem essa infraestrutura.

Figura 3.7: Exemplo de teste de pico



Fonte: (CIGNITI, 2011)

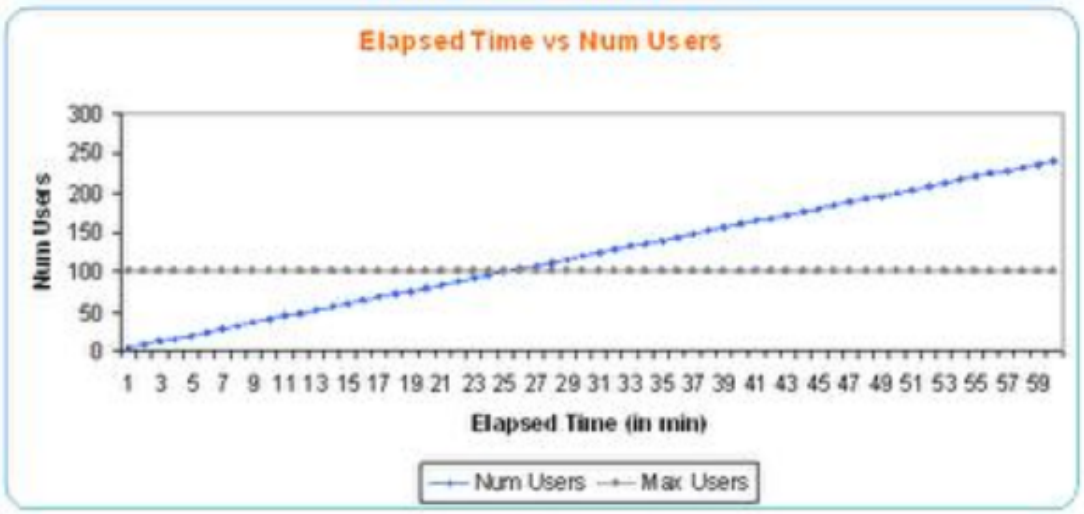
3.6.6 Teste de Falha

No teste de falha o volume é aumentado linearmente até o momento em que a aplicação apresenta falhas do ponto de vista funcional. Isso pode ocorrer de diversas maneiras como a utilização excessiva de CPU ou Memória. *threads* presas por excesso de processos. Multiplas execuções desse tipo de teste são planejadas para a determinação de capacidade do sistema e possíveis otimizações como aumento de *hardware*. A Figura 3.8 mostra o aumento linear no número de usuários concorrentes executando operações no sistema até o momento em que o sistema inicia a apresentação de falhas indicando o ponto de quebra em termos de usuários concorrentes. O mesmo pode ser realizado em função do volume total de dados passando simultanemanete pela aplicação.

Resultados obtidos nesse tipo de execução são a quantidade de carga ou de usuários simultâneos que podem fazer com que o sistema pare de funcionar. O resultado do monitoramento e observação do comportamento de diferentes componentes do sistema antes de ocasionar a falha geral pode servir de indicador para futuros monitoramentos

onde o mesmo comportamento pode ocorrer, indicando não por motivos de excesso de carga mas por quaisquer outras situações uma iminente falha na aplicação. Além disso é possível determinar as possíveis falhas com excesso de dados, ajudando o mapeamento de vulnerabilidades do sistema. Na maioria dos casos os gargalos de um sistema são descobertos nesses tipos de teste onde o primeiro componente a falhar indica sua menor capacidade em relação aos demais.

Figura 3.8: Exemplo de teste de falha



Fonte: (CIGNITI, 2011)

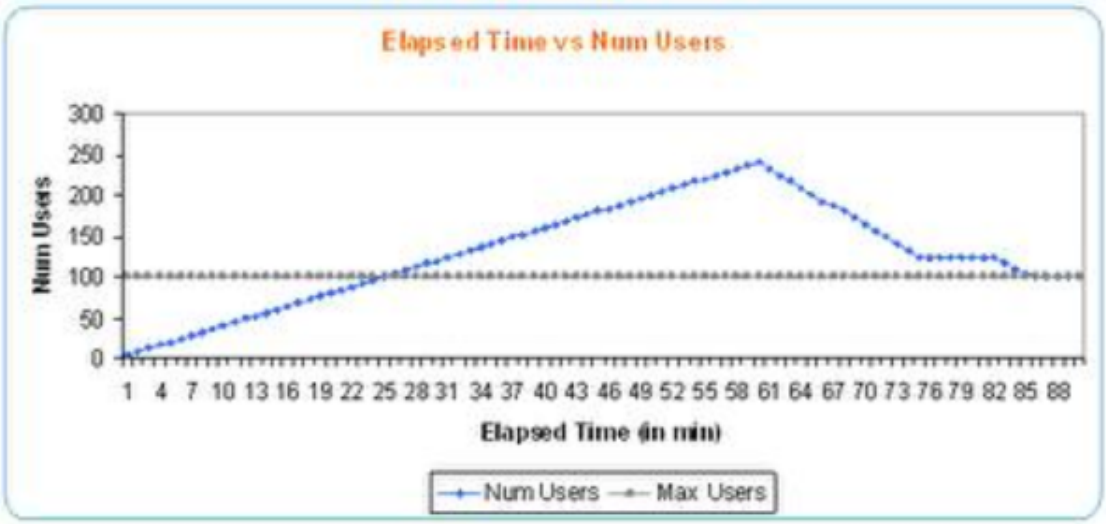
3.6.7 Teste de Recuperação

Ao ser descoberto o ponto de falha de uma aplicação, ou seja, o momento exato onde uma aplicação deixa de estar operando funcionalmente, utilizamos o mesmo para realizar o teste de recuperação. Nesse cenário o engenheiro de performance deve projetar uma execução onde o volume de dados ou de usuários vai crescendo até o ponto onde foi descoberta a quebra do sistema. Neste exato momento a mesma carga deve ser diminuída até o nível normal de carga estabelecido no levantamento de requisitos. É similar ao teste de resiliência demonstrado na Seção 3.6.4 porém com a carga máxima focada no ponto de quebra. Na Figura 3.9 é ilustrado o crescimento linear de usuários até o ponto de quebra registrado na Figura 3.8 porém com o volume de usuários concorrentes imediatamente decrescido até o volume utilizado no teste de carga.

Garantir que a aplicação consegue se recuperar sozinha ao chegar em um pico de estresse idêntico ao ponto de falha é o principal objetivo de teste. Do ponto de vista do

cliente é necessário garantir que em determinada situação inesperada com pico alto de carga superior ao esperado, o sistema é capaz de se manter funcionando.

Figura 3.9: Exemplo de teste de recuperação

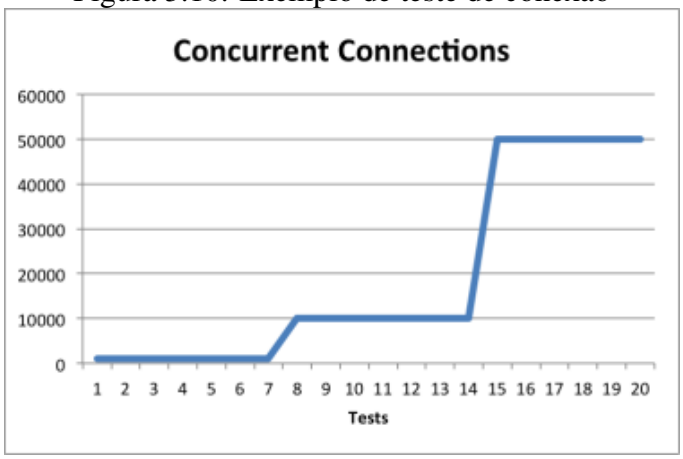


Fonte: (CIGNITI, 2011)

3.6.8 Teste de Conexão

Em sistemas que enfrentam situações de múltiplos usuários se conectarem ao mesmo tempo o teste de conexão é necessário. Por exemplo, portais de vendas de ingressos onde um grande número de *logins* são efetuados simultanemente necessitam a garantia de que o sistema consegue se manter operando sem degradação de tempos de resposta e utilização sobrecarregada de recursos do sistema. O teste é planejado pelo engenheiro de performance de duas maneiras: Utilizam-se ambas as cargas normais e de estresse no sistema porém sem a rampa crescente de usuários. Todas as conexões devem ser realizadas simultâneamente ou com um período muito curto de tempo para a correta avaliação da capacidade de conectividade no desempenho do sistema. A Figura 3.10 exemplifica um cenário com o tipo de teste onde o número de conexões, ou seja, novos usuários virtuais concorrentes começam sendo inseridos simultanemanete no sistema até determinado volume e os recursos físicos devem ser monitorados com maior foco no último período de maior aumento no volume total.

Figura 3.10: Exemplo de teste de conexão

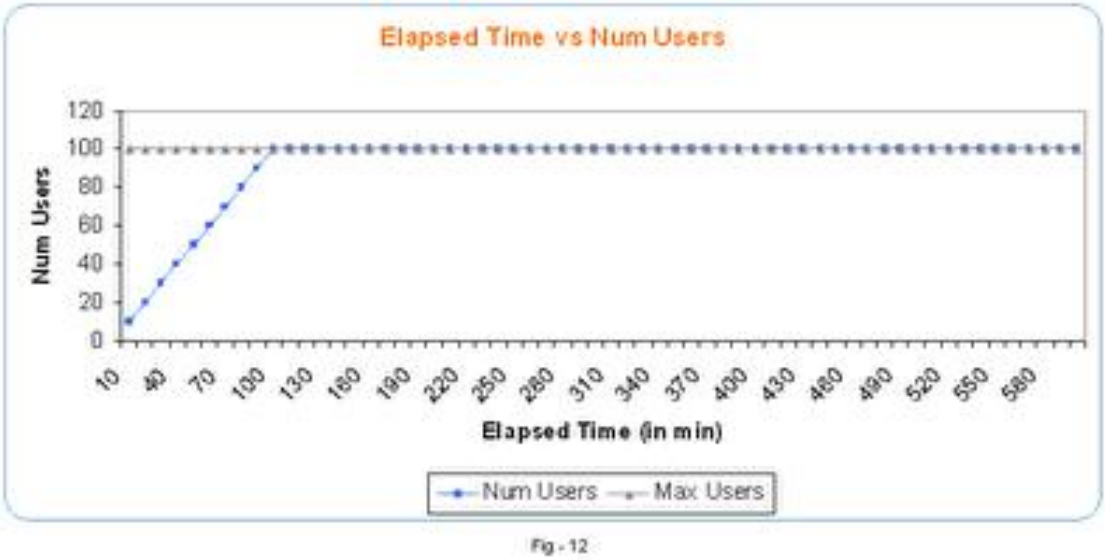


Fonte: (NGINX. . . , 2014)

3.6.9 Teste de Resistência

O Teste de resistência é realizado com o objetivo de verificar se tempos de resposta e utilização de recurso são prejudicados ou sofdrem algum tipo de utilização não esperada ou degradação em função do longo tempo da aplicação sendo utilizada ininterruptamente. Normalmente esse tipo de teste é realizado por último pois quaisquer problemas encontrados no teste de resistência podem estar também associados com problemas encontrados nos testes anteriores em função da alta vazão por exemplo. A Figura 3.11 mostra o volume de usuários na mesma proporção que o teste de carga porém com o tempo total de teste muito maior que os tempos realizados nos testes anteriores. O tempo de operação dos testes deve ser definido com o cliente estimando o tempo que os servidores em produção devem ficar ligados ininterruptamente. O tempo de engajamento do time de performance também é importante para avaliar o tempo de execução, entretando, é uma boa prática testes de resistência de no mínimo 24 horas para uma análise mais precisa da utilização de recursos e tempos de resposta com longos períodos de funcionamento da aplicação.

Figura 3.11: Exemplo de teste de resistência



Fonte: (CIGNITI, 2011)

3.7 Implementar Execução

O objetivo desta fase é utilizar todas as informações obtidas nos passos discutidos nas seções anteriores e colocar o plano de teste de performance em prática. Como o objetivo do trabalho é o foco no processo de modelagem de teste de performance com a menor dependência do conhecimento de tecnologia possível, incluindo a utilização de ferramentas para apoio em teste de performance, esta seção explica sem muitos detalhes o processo de execução dos cenários planejados de teste de desempenho.

No mercado há uma quantidade considerável de ferramentas de apoio de teste de performance como *Neoload, Load Runner, JMeter, etc.* Cada ferramenta possui suas particularidades como diferentes linguagens de programação para construção dos *scripts* a serem executados pelo engenheiro de performance e variadas opções de monitoramento de recursos por parte de quem irá executar o teste. As tarefas em comum e que são relevantes para o processo modelado no trabalho são as aplicações na ferramenta dos cenários escolhidos em função de volume de dados, usuários concorrentes e tempo de execução do teste. As métricas selecionadas tanto em tempo de resposta como utilização de recursos por parte do sistema devem ser corretamente configuradas e para grandes empresas com times distribuídos, como é o conceito para o qual o modelo proposto foi desenvolvido, a correta permissão para captura das métricas deve ser provida pelos times

responsáveis pela manutenção dos ambientes de performance oferecidos no projeto.

Com todos os passos aplicados corretamente na ferramenta escolhida, os comportamentos esperados em produção devem ter sido identificados e todas as possibilidades de melhoria identificadas a partir do momento em que as execuções são concluídas, restando ao engenheiro de performance compilar os resultados obtidos para a apresentação aos times envolvidos e *stakeholders*.

3.8 Apresentação de Resultados

O último passo do processo de teste de desempenho é a análise de resultado e apresentação para o time de projeto e todas as demais partes interessadas. Similar ao processo da Seção 3.7 onde o teste é implementado, todas as principais informações que se referem aos cenários, volume e métricas coletadas são apresentados. É importante reforçar que neste ponto a apresentação é feita para um público menos técnico que todas as fases iniciais. É importante que o engenheiro de performance organize as informações e métricas apresentadas de uma forma que pessoas não familiarizadas com a TI sejam capazes de entender. Agrupar as informações a um nível de entendimento de usuário é mandatório para que haja a maior clareza possível nos resultados obtidos e possíveis melhoras alcançadas por parte do cliente que receberá o projeto e conseguirá ver o valor do teste de performance agregado através dos conhecimentos obtidos pelo engenheiro de performance ao aplicar o processo descrito durante este trabalho.

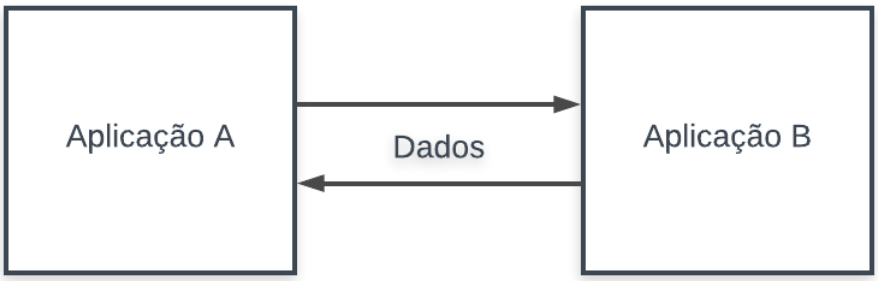
O modelo possui todos os seus passos incluído no escopo de aplicações que possuem orientação de entrega contínua. Tal método de entrega inclui em cada inclusão de funcionalidades em aplicações existentes o engajamento de teste de performance desde a análise de requisitos para uma nova funcionalidade até definição de sistema e carga mesmo quando entregas são realizadas em ritmo constante. A definição de SLA devido a inclusão recorrente de funcionalidades deve ser revisada pela possibilidade da inclusão de componentes que podem degradar o desempenho de determinado escopo. O mesmo com a definição de carga e na estratégia de testes pois cada funcionalidade entregue pelo time de desenvolvimento pode impactar a aplicação de uma forma diferente e a repetição dos passos propostos até aqui é de grande importância para a correta análise de desempenho do sistema.

4 ESTUDO DE CASO

Para o estudo de caso da metodologia de testes de performance modelada para profissionais com pouca ou nenhuma experiência na área foi utilizado um engajamento de duas aplicações, aqui referidas como Aplicação A e Aplicação B devido à confidencialidade, de uma grande empresa, responsáveis pela análise de crédito para financiamento de computadores. As duas aplicações estão em um ecossistema composto por diversas outras aplicações, entretanto os dois sistemas estudados comunicam-se diretamente entre si como ilustra a Figura 4.1. O modelo proposto pode ser aplicado em diversos tipos de empresas e para uma grande quantidade de aplicações. Todavia, o foco principal é para aplicações já existentes no mercado e que possuem como padrão a entrega contínua e incremento de funcionalidades ou até mesmo atualizações constantes seja no código do sistema ou mudança de *hardware* ou plataforma.

Os passos apresentados no Capítulo 3 são aplicados no sistema sob teste com o objetivo de traçar o melhor plano de performance possível sem utilização de conhecimentos prévios ou baseado em experiência na área de engenharia de performance. Apenas informações extraídas do modelo serão discutidas nas próximas seções.

Figura 4.1: Aplicações do Estudo de Caso



Fonte: Empresa que solicitou teste de performance

4.1 Estabelecer Objetivos de Teste

O principal objetivo de teste do estudo de caso apresentado é a certificação da capacidade e estabilidade das duas aplicações para o período de *Black Friday* onde o volume de acessos previsto é muito maior do que o volume enfrentado usualmente. Recursos físicos do sistema e tempo de resposta devem ser devidamente verificados no teste de per-

formance. Além do objetivo de validar o comportamento de ambas as aplicações sob alto volume uma das aplicações sofreu *downgrade* de máquinas físicas devido a baixa utilização de CPU no ambiente de produção. Com a nova configuração em *hardware* diminuída em relação ao sistema antes da mudança os mesmos aspectos devem ser verificados em termos de utilização de recursos, tempos de resposta e comportamento da aplicação sob alto volume de dados. Em resumo temos os seguintes objetivos de teste para o estudo de caso:

- Verificar tempos de resposta após nova configuração;
- Verificar utilização de recursos após nova configuração;
- Verificar tempos de resposta para alto volume esperado;
- Verificar utilização de recursos para alto volume esperado;

Com os objetivos traçados para o engajamento tendo em vista que a aplicação além de esperar um alto volume de dados após a última atualização, terá seu hardware diminuído, o engenheiro de performance poderá seguir os próximos passos selecionando os melhores tipos de teste e caracterizando o sistema e carga de dados para que na etapa de execução com auxílio de ferramentas específicas para teste de performance todo o projeto de performance esteja correto.

4.2 Estabelecer SLA's

As aplicações testadas no estudo de caso deste trabalho já estão operando no mercado há mais de 10 anos. Por este motivo, como mencionado na Seção 3.2, não é necessária a obtenção de SLA's pois já existem valores definidos para os tempos de resposta deste estudo de caso. O que é necessário ao engenheiro de performance nesta fase é saber quais SLA's já foram obtidos e os tempos de resposta correntes em produção. Neste caso basta a obtenção dos resultados da última execução de teste de performance caso tenha sido feita. Caso os SLA's não estejam disponíveis, é necessário que o time de desenvolvimento ou o próprio cliente forneçam uma estimativa dos tempos de resposta. O estabelecimento de SLA's serve como guia para a garantia do comportamento esperado dos serviços ou requisições realizados pelo sistema, os resultados serão apresentados para o time de projeto na última etapa e estabelecer os números correntes em produção é importante para o engenheiro de performance usar como base na avaliação de suas execuções sendo possível concluír a melhora ou piora dos serviços a serem testados.

Os SLA's para as aplicações testadas foram definidos previamente baseados em seus tempos de resposta unicamente. A aplicação lida com *webservices* e requisições HTTP. Em ambos os casos os SLA's são mensurados em função de seu tempo de resposta, ou seja, para os serviços a partir do momento em que a requisição é enviada e o XML de resposta é recebido pelo sistema. No caso das requisições HTTP a partir do momento em que o usuário preenche as informações na tela e a próxima tela é recebida ao usuário final com as informações referentes ao processo de financiamento no caso da aplicação sendo testada.

Na Tabela 4.1 é possível ver os SLA's coletados com o time de projeto juntamente com os tempos correntes de produção dos serviços a serem testados. Todas as informações contidas na tabela e obtidas nesse passo do modelo vão servir como base para o engenheiro de performance avaliar os resultados encontrados nas execuções dos testes de performance projetados neste modelo.

Tabela 4.1: Serviços a serem testados, SLA's estabelecidos e tempos de resposta correntes

Serviço / Requisição	*SLA Estabelecido*	*Tempo de Resposta*
Serviço 1	< 5s	1.7s
Serviço 2	< 5s	4.87s
Serviço 3	< 5s	3.32s
Serviço 4	< 5s	0.3s
Serviço 5	< 5s	1.85s
Serviço 6	< 5s	4.32s
Serviço 7	< 5s	4.1s
Serviço 8	< 5s	3.1s
Serviço 9	< 5s	1.5s
Serviço 10	< 5s	2.6s
Serviço 11	< 5s	2.1s
Serviço 12	< 5s	6.0s
Serviço 13	< 15s	0.5s
Serviço 14	< 15s	14.2s
Serviço 15	< 30s	17.2s
Serviço 16	< 30s	15.1s
Serviço 17	< 30s	21.7s
Requisição 1	< 6s	5.75s
Requisição 2	< 6s	4.85s
Requisição 3	< 6s	5.1s
Requisição 4	< 6s	5.2s
Requisição 5	< 12s	7.87s
Requisição 6	< 12s	7.98s
Requisição 7	< 12s	8.12s
Requisição 8	< 12s	8.9s

Fonte: Empresa que solicitou teste de performance

4.3 Caracterizar Sistemas

Neste passo foi feita uma avaliação sobre os recursos físicos da Aplicação A e Aplicação B solicitadas para o teste de performance. As informações a serem obtidas segundo o plano são os componentes básicos referentes ao *hardware* de ambos os ambientes de performance e produção pois tratam-se de aplicações que não possuem a mesma configuração em termos de capacidade nos ambientes de performance e produção. Sendo assim é necessário realizar a busca por tamanho de CPU e Memória assim como a quantidade de CPUs em ambos os ambientes para medir a proporção correta e realizar os ajustes de volume de dados conforme as diferenças entre ambos os ambientes. Na Tabela 4.2 podemos ver as principais informações referentes a configuração de capacidade da aplicação A para o ambiente de performance que é em uma escala menor que o ambiente de produção como mostrado na Tabela 4.3.

Tabela 4.2: Aplicação A - Ambiente de Performance

Servidores	*Tecnologia*	*CPU*	*Memória*
Servidor 1	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	62GB RAM
Servidor 2	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	94GB RAM
Servidor 3	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	94GB RAM
Servidor 4	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	94GB RAM
Servidor 5	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	94GB RAM
Servidor 6	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM

Fonte: Empresa que solicitou teste de performance

Tabela 4.3: Aplicação A - Ambiente de Produção

Servidores	*Tecnologia*	*CPU*	*Memória*
Servidor 1	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	62GB RAM
Servidor 2	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM
Servidor 3	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM
Servidor 4	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM
Servidor 5	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM
Servidor 6	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM
Servidor 7	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM
Servidor 8	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM
Servidor 9	Oracle Enterprise Linux 7.3	48 cores @ 2.20GHz	96GB RAM

Fonte: Empresa que solicitou teste de performance

Com ambas as configurações caracterizadas para a aplicação A deve-se ver a proporção entre ambos os ambientes em termos de capacidade. Usualmente ambientes de performance possuem uma capacidade reduzida em relação a produção por questões financeiras. No caso da aplicação A sob teste é mostrado na Tabela 4.4 que a proporção é do ambiente de performance tendo o equivalente a cerca de 65%do ambiente de produção.

Tabela 4.4: Aplicação A - Comparativo entre ambientes

	Performance	**Produção**
CPU	288	432
Memória	534GB	830GB
Nro. Servidores	6 Servidores	9 Servidores
Sistema Operacional	OEL 7.3	OEL 7.3

Fonte: Empresa que solicitou teste de performance

O mesmo comparativo deve ser feito para a Aplicação B levando em conta, também, informações de CPU e memória com o objetivo de determinar a equivalência entre ambientes. As informações dos ambientes de performance e produção estão disponíveis na Tabelaa 4.5 e Tabelaa 4.6 respectivamente.

Tabela 4.5: Aplicação B - Ambiente de Performance

Servidores	*Tecnologia*	*CPU*	*Memória*
Servidor 1	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM
Servidor 2	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	15GB RAM
Servidor 3	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	15GB RAM
Servidor 4	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	15GB RAM

Fonte: Empresa que solicitou teste de performance

Tabela 4.6: Aplicação B - Ambiente de Produção

Servidores	*Tecnologia*	*CPU*	*Memória*
Servidor 1	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM
Servidor 2	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM
Servidor 3	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM
Servidor 4	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM
Servidor 5	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM
Servidor 6	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM
Servidor 7	Oracle Enterprise Linux 7.3	8 cores @ 2.20GHz	16GB RAM

Fonte: Empresa que solicitou teste de performance

Com as informações obtidas, a Tabelaa 4.7 mostra a equivalência do ambiente de performance em relação ao de produção em 55% em termos de sua capacidade. Ambas as plataformas são as mesmas e o número de núcleos e memória iguais, variando basicamente o número de servidores disponíveis entre um ambiente e outro.

Tabela 4.7: Aplicação B - Comparativo entre ambientes

	Performance	**Produção**
CPU	32	56
Memória	61GB	112GB
Nro. Servidores	4 Servidores	7 Servidores
Sistema Operacional	OEL 7.3	OEL 7.3

Fonte: Empresa que solicitou teste de performance

4.4 Caracterizar Carga de Dados

Este é o passo mais importante no planejamento do estudo de caso visto que os volumes são definidos e ajustados em relação ao ambiente de performance e produção nessa etapa. Para todos os tipos de dados obtidos ambas as aplicações A e B são submetidas à mesma carga pois fazem parte do mesmo fluxo para este estudo. Visto que a aplicação já está no mercado e a abordagem do modelo proposto é adaptada a aplicações de entrega contínua e para grandes empresas os números de volumes esperados foram fornecidos pelo cliente. Entretanto os volumes de dados também podem se obtidos, se houver disponibilidade, por ferramentas de *profiling* ou monitoramento capazes de coletar essas informações através do histórico de uso das aplicações. Abordagens como a criação de modelos orientados a formulários como mostrado na Figura 3.2 não serão aprofundados por serem úteis em sistemas onde estão sendo testados pela primeira vez não havendo noção por parte do time de projeto dos dados envolvidos entre páginas e serviços operados, entretanto a utilização dessa técnica é altamente recomendada para aplicações sob teste de performance pela primeira vez.

Como proposto no modelo os componentes básicos são divididos em classes e os recursos físicos utilizados por esses componentes de carga de dados são corretamente atribuídos. O passo seguinte é determinar os números absolutos de carga esperados por uma determinada unidade de tempo, e por consequência, determinar a vazão de dados a fim de se obter a caracterização de carga que irá ser testada. As cargas devem ser descritas para ambas as situações de utilizaçã normal do sistema e para os picos de volume como determinado na etapa onde se definem os objetivos de performance.

- **Identificação de componentes de carga básicos:** As operações a serem realizadas pelas aplicações e validadas nas execuções de performance são constituídas por requisições HTTP e chamadas de *Webservices*. Num total temos 17 serviços diferentes sendo disparados e 8 requisições em HTTP sendo enviadas durante as execuções.
- **Componentes de carga básicos divididos em classes:** Levando em consideração as chamadas realizadas pelas aplicações no teste de performance temos basicamente duas classes. classe dos serviços e classe das requisições, por utilizarem recursos similares é útil separar as chamadas de *webservices* das requisições HTTP.
- **Componentes de sistema e seus recursos utilizados por classe de carga:** Nesta etapa utilizam-se as classes separadas no passo anterior, no caso as requisições e

os *webservices*, e selecionam-se, para cada classe, os recursos físicos utilizados em cada uma delas. É importante que se faça essa correspondência para que a análise dos recursos físicos utilizados seja facilitada com altos volumes de dados ou um maior volume para uma determinada classe onde o correspondente recurso do sistema deve ser analisado com mais atenção. Nas aplicações sob teste deste capítulo temos para o ambiente de performance dois servidores dedicados à base de dados, onde parte dos serviços são disparados diretamente sobre esses servidores além de metade das requisições HTTP. As demais máquinas físicas são dedicadas para a aplicação onde o restande dos serviços e metade das requisições HTTP são enviadas para tais servidores. Na Tabela 4.8 e Tabela 4.9 é possível ver quais servidores são disparados por quais serviços e requisições. No momento em que o engenheiro de software isola as cargas de dados em execuções separadas, a avaliação de utilização dos recursos físicos é mais clara. Para testes de carga também é possível identificar raízes de problemas ao se observar utilização maior que o normal em determinados servidores caso aconteça.

Tabela 4.8: Recursos Utilizados por *Webservices*

	Aplicação A	**Aplicação B**
Serviço 1 a 12	Servidor 1 e 2	Servidor 1 ao 5
Serviço 13 a 17	Servidor 3 e 4	Servidor 6 e 7

Fonte: Empresa que solicitou teste de performance

Tabela 4.9: Recursos Utilizados pelas requisições

	Aplicação A	**Aplicação B**
Requisições 1 a 4	Servidor 1 e 2	Servidor 1 ao 5
Requisições 5 a 8	Servidor 3 e 4	Servidor 6 e 7

Fonte: Empresa que solicitou teste de performance

- **Identificação de interação inter-componentes:** Como se tratam de duas aplicações onde o fluxo passa por ambas de maneira igual essa fase é apenas para a confirmação com o time de projeto. Não há diferenças de cargas de dados que fluem por uma aplicação ou outra, apenas a utilização de recursos é orientada por aplicação porém o fluxo de dados é distribuído de maneira igual entre ambas as aplicações.
- **Identificar demanda de serviço:** Essa informação é coletada com o time de projeto e cliente, onde o volume de dados específico por serviço ou requisição em uma determinada unidade de tempo é especificado. Para o estudo de caso foi coletado o número de transações por hora. Transação é definido pelo número de serviços dis-

parados e requisições enviados por parte dos usuários em uma hora. A Tabela 4.10 mostra a quantidade de informação a ser passada nas aplicações em função do número de transações e volume de dados de cada uma:

- **Ajustar volume em relação à equivalência do ambiente de performance:** Nesse passo as informações obtidas na identificação de demanda de serviço realizada no passo anterior, é ajustado ao correto volume adaptado ao ambiente de performance. Também, na Tabela 4.10 os volumes já estão ajustados ao ambiente de performance previamente calculado em relação à sua equivalência com produção. Todos os números em termos de transações por hora e volume de dados estão ajustados com a proporção de 60% em relação aos volumes fornecidos pelo cliente.

Tabela 4.10: Volumes de dados esperados em dias normais de produção

Serviço / Requisição	*Transações por hora*	*Volume por Hora*
Serviço 1	3170	6.2MB
Serviço 2	40	0.6MB
Serviço 3	25	7.4MB
Serviço 4	17000	41.8MB
Serviço 5	2340	6.8MB
Serviço 6	1250	4,8MB
Serviço 7	1160	7.1MB
Serviço 8	580	6.4MB
Serviço 9	2100	1.5MB
Serviço 10	40	2.5MB
Serviço 11	2100	4.2MB
Serviço 12	10650	10,4MB
Serviço 13	360	8.0MB
Serviço 14	45	3.1MB
Serviço 15	120	3.7MB
Serviço 16	320	2.9MB
Serviço 17	8350	8.5MB
Requisição 1	2680	18.4MB
Requisição 2	2680	19.7MB
Requisição 3	2680	11.7MB
Requisição 4	2680	9.9MB
Requisição 5	4800	11.4MB
Requisição 6	4800	12.2MB
Requisição 7	4800	7.6MB
Requisição 8	4800	5.2MB
Total	79580	222MB

Fonte: Empresa que solicitou teste de performance

4.5 Selecionar Métricas

A etapa de seleção de métricas envolve a escolha das principais informações referentes a ambas as aplicações testadas sobre seus desempenhos sob as execuções de performance. Levando em consideração a aplicação do modelo de performance para grandes empresas com a possibilidade de utilização de ferramentas de performance com uma alta quantidade de métricas disponíveis para coletar foi utilizado o máximo de métricas possíveis dentro das limitações de tecnologia. Usando como base as informações da Seção 3.5, Tabela A.1 do Apêndice **??** e o Apêndice **??**, onde temos as métricas coletadas na literatura além do conhecimento de engenheiros de performance em atuação na indústria as métricas coletadas serão as seguintes:

1. *Linux_System_CPUIdle* (Quantidade de CPU não utilizada)
2. *Memory_% User Memory* (% de memória utilizada)
3. *JMX_MemoryHeap_Used* (Memória alocada pela *Java Virtual Machine*)
4. *%Execute_ThreadIdle_Count* (Quantidade de *threads* ociosas)
5. *Oracle_IndexedQueries_Percentage* (% de *queries* indexadas)
6. *WebLogic_JvmProcessorLoad* (Quantidade de uso do processador da JVM)
7. *Linux_System_ProcessRunnable* (Quantidade de processos aptos a executar)
8. *WebLogic_ConnectionPool_ConnectionDelayTime* (Tempo de atraso nas conexões da *connectionpool*)
9. *ThreadPoolRuntime_StuckThreadCount* (*Threads* presas no momento dentre a *threadpool*)
10. *JMX_ProcessCpuLoad* (Quantidade de uso do processador da JMX)
11. *Thread_HoggingThreadCount* (Quantidade de *threads* presas)
12. *Oracle_Call Rates_Rollback* (Taxa de chamadas não finalizadas)
13. *Oracle_Sessions_%active* (% de sessões ativas no banco)

4.6 Escolher Técnicas de Teste

Na etapa de seleção de técnicas de teste vamos utilizar as técnicas apresentadas na Seção 3.6 e as cargas de dados obtidas para que seja possível caracterizar os cenários que serão executados com o auxilio das ferramentas de performance específicas. Como determinado anteriormente, os objetivos de teste incluem a verificação da capacidade do sistema para picos de volumes ocasionados por uma alta quantidade de usuários em relação ao volume normal de produção que está previsto. Além disso a estabilidade e o consumo de recursos após a diminuição de *hardware* também devem ser avaliadas.

A Tabela 4.11 mostra a lista de testes utilizados no estudo de caso bem como os detalhes referentes aos usuários concorrentes esperados para a execução dos cenários. É importante reforçar que as ferramentas de performance possuem mecanismos para controlar o volume em função dos usuários virtuais concorrentes e vice-e-versa facilitando o trabalho do engenheiro de performance na configuração dos cenários listados na Tabela 4.11. Considerando o volume de dados, sempre é objetivado o valor obtido com o time de projeto especificado na Tabela 4.10 de 222MB por hora de execução em ambiente

de produção visto que há diferenças entre os ambientes de performance e produção como já avaliado. Por padrão todas as rampas de inserção dos usuários virtuais foram de 30 minutos como mencionado na Seção 3.6. O número de usuários virtuais foi dado pela própria ferramenta de execução de teste visto que as execuções são condicionadas em função do volume de transações por hora e pela vazão de dados. Portanto, objetivando o número de transações obtido na caracterização de carga de dados são necessários os usuários virtuais listados na Tabela 4.11. Quando necessário simular um número específico de usuários concorrentes, é possível inserir manualmente este parâmetro na ferramenta de teste de performance.

Tabela 4.11: Técnicas de teste utilizadas

Técnica	**Duração**	**Usuários Carga Normal**	**Usuários Carga Pico**
Teste de Fumaça	30 minutos	10	NA
Teste de Carga	2 horas	250	NA
Teste de Pico 1	2 horas	250	275
Teste de Pico 2	2 horas	250	305
Teste de Pico 3	2 horas	250	400
Teste de Resiliência	4 horas	250	400
Teste de Conexão - Normal	2 horas	250	NA
Teste de Conexão - Pico	2 horas	NA	400
Teste de Resistência 1	24 horas	250	NA
Teste de Resistência 2	48 horas	250	NA
Teste de Estresse	4 horas	NA	400
Teste de Falha - App B	4 horas	NA	1500
Teste de Recuperação - App B	4 horas	NA	Ponto de Quebra

Fonte: Empresa que solicitou teste de performance

Os testes de falha e de recuperação são executado com o foco na aplicação B pelo fato de ter sido modificada a sua capacidade em relação aos recursos físicos. Como a aplicação A já em produção e sem sofrer mudanças nos recursos físicos ja possuia informações relativas ao seu ponto de quebra, a execução do teste de falha e de recuperação foi planejada com o objetivo de descobrir o ponto máximo de carga que a nova configuração da aplicação B é capaz de suportar.

4.7 Execução

A Execução dos cenários baseia-se na aplicação em ferramenta específica para teste de performance, de todos os cenários elaborados e a implementação das informações adquiridas no processo de planejamento de teste de performance em formato de *scripts* a fim de simular as condições reais da aplicação no ambiente de produção no ambiente de

performance.

A ferramenta utilizada no estudo de caso foi o *Neoload* pois a empresa que solicitou o engajamento do teste possui licença para sua utilização. Com o objetivo do trabalho de propor uma metodologia de teste modelada para a realidade de grandes empresas com alta rotatividade de funcionários foi utilizada a mesma ferramenta que esse tipo de público com essa realidade também utiliza.

Para a criação dos cenários todos os volumes de carga de dados coletados durante o processo de planejamento foram utilizados. O mesmo foi feito com a criação dos cenários e comportamento dos usuários virtuais para a execução de cada teste.

4.8 Apresentação de Resultados

A última atividade no processo de engenharia de performance em um projeto é a apresentação de resultados para o time e *stakeholders*. Como já mencionado é um processo onde todas as informações apresentadas pelo engenheiro devem conter a menor quantidade de dados técnicos possíveis e focar em resultados claros do ponto de vista de regra de negócio como valores absolutos em tempos de resposta e melhorias alcançadas ou riscos encontrados durante o período de execução. Visto que o alvo para as apresentações é um público menos envolvido com a TI quanto mais claro e simples forem gráficos e tabelas apresentadas, mais claro o valor agregado pelo teste de performance se torna. Métricas simples como utilização de CPU ou Memória e Tempo de resposta de um determinado cenário são suficientes para o entendimento em apresentações de resultado para esse tipo de público.

5 CONCLUSÃO

A importância da engenharia de performance em aplicações de larga escala vem se tornando cada vez mais significativa. Problemas ocasionados pela complexidade de sistemas e constantes mudanças no modo como grandes empresas operam em termos de desempenho se tornam iminentes. Por esse motivo, processos de análise de desempenho corretos são essenciais. Devido a alta rotatividade de funcionários em grandes empresas e o crescimento na área, além das complexidades envolvidas na engenharia de performance como conhecimento em termos de tecnologias e processos de teste, é necessária uma metodologia de teste de performance modelada ao público profissional da TI sem a devida experiência. Tal necessidade pôde ser suprida com o modelo apresentado no decorrer deste trabalho.

Uma ameaça à proposta do modelo é a falta de validação por um profissional sem experiência na área de performance. Os passos foram seguidos, porém as atividades executadas por um engenheiro de performance já experiente. Além disso, há oportunidade para futuros trabalhos incluindo o aprimoramento do modelo agregado por experiências adicionais e mais informações provenientes da literatura. Foi identificada também a possibilidade de automação do modelo com um guia de recomendações e a aplicação de todos os passos descritos diretamente em uma ferramenta de apoio ao teste de performance.

A importância do desenvolvimento do modelo proposto foi enriquecedora no ponto de vista de um profissional da área de performance pois ao pesquisar informações de todas as etapas do processo de planejamento de performance, técnicas como tipos de testes e métricas a serem coletadas assim como indicadores de gargalos não eram conhecidos pelo autor nesse trabalho, enriquecendo não só quem pode aplicar o modelo como quem já possui experiência.

REFERÊNCIAS

. Ieee standard for software test documentation. *IEEE Std 829-1998*, Los Alamitos, IEEE Computer Society, p. 1–64, Dec 1998.

ALFANO, J. **Modeling Performance Tests**. [S.l.], 2018. Acessado em 18/08/2018. Available from Internet: <https://www.neotys.com/blog/modeling-performance-tests/>.

AMMANN, P.; OFFUTT, J. **Introduction to Software Testing**. [S.l.]: Cambridge University Press, 2016. (Introduction to Software Testing). ISBN 9781107172012.

CIGNITI, T. **Types of Performance Testing**. 2011. Acessado em 02/10/2018. Available from Internet: <https://www.cigniti.com/blog/types-of-performance-testing/>.

CRISTALLI, A. B. E. R. R. **Base de conhecimento em teste de software**. 2nd. ed. [S.l.]: Martins Fontes, 2007. P.19. ISBN 978-8580630534.

FERRARI, D. **Computer Systems Performance Evaluation**. [S.l.]: Prentice-Hall, 1978. ISBN 9780131651265.

JAIN, R. **The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling**. [S.l.]: John Wiley & Sons, 1990.

KOUNEV, S.; BUCHMANN, A. P. Performance modeling and evaluation of large-scale j2ee applications. In: **29th Int. CMG Conference**. [S.l.: s.n.], 2003.

KOUNEV, S. et al. Model-based techniques for performance engineering of business information systems. **International Symposium on Business Modeling and Software Design**, v. 0109, p. 19–37, 2012.

LUTTEROTH, C.; WEBER, G. Modeling a realistic workload for performance testing. In: **12th IEEE International Enterprise Distributed Object Computing Conference**. Munich, Germany: IEEE, 2008. p. 149–158. ISSN 1541-7719.

MYERS, G.; SANDLER, C.; BADGETT, T. **The Art of Software Testing**. [S.l.]: Wiley, 2011. (ITPro collection). ISBN 9781118133156.

NGINX Websocket Performance. 2014. Acessado em 18/09/2018. Available from Internet: <https://www.nginx.com/blog/nginx-websockets-performance/>.

PEZZE, M.; YOUNG, M. **Software Testing and Analysis: Process, Principles and Techniques**. [S.l.]: Wiley, 2008. ISBN 9780471455936.

RANI, S. B. U. A detailed study of software development life cycle (sdlc) models. **International Journal of Engineering and Computer Science**, v. 6, n. 7, p. 22097–22100, jul. 2017. ISSN 2319-7242.

SAROJADEVI, H. Performance testing: methodologies and tools. **Journal of Information Engineering and Applications**, v. 1, n. 5, p. 5–13, 2011. ISSN 2224- 5758.

SAS, N. **Neoload 5.2 Documentation**. 2016. Acessado em 27/08/2018. Available from Internet: <https://www.neotys.com/documents/doc/neoload/5.2.x/en/WebHelp/#2983.htm>.

TASSEY, G. **The Economic Impacts of Inadequate Infrastructure for Software Testing**. [S.l.]: Diane Publishing Company, 2002. ISBN 9780756726188.

WINDOWS VM Health. 2017. Acessado em 05/10/2018. Available from Internet: <https://docs.microsoft.com/en-us/azure/monitoring/infrastructure-health/>.

APÊNDICE A — TABELA DE MÉTRICAS PADRÃO

Tabela A.1: Conjunto de métricas padrão e indicadores de gargalos

Métrica	**Regra de Análise**	**Descrição de Análise**
Windows_% Processor Time	Um gargalo é identificado quando: 1. A taxa atingir 85% em qualquer momento ou 2. A taxa atingir 70% durante 60% do tempo ou 3. A média atingir 60% durante 70% do tempo A média for 50% durante 80% do tempo.	Encontrar o processo com alto consumo, realizar um *Upgrade* de processador ou instalar um processador adicional.
Windows_Memory_Available_MB	Um gargalo é encontrado quando a média for <= 1000.	
Linux_System_CPUIdle	Um gargalo é identificado quando a propriedade avg for: 1. <=15% em qualquer momento ou: 2. A média atingir 30% durante 60% do tempo ou 3. A média atingir 40% durante 70% do tempo A média for 50% durante 80% do tempo	Encontrar o processo que está usando alta porcentagem do processador, realizar um *Upgrade* de processador ou instalar um processador adicional
Linux_MemoryFree	Um gargalo é encontrado quando a propriedade média for menor que 10%	
JMX_MemoryHeapUsed	Um gargalo é encontrado quando a média não permanece estável para execuções longas. Utilização de memória heap crescente indica risco de *memory leak*	

Fonte: (SAS, 2016)

APÊNDICE B — TABELA DE MÉTRICAS SECUNDÁRIAS

- ***Windows_System_Processor_Queue_Length***

  Identificação de Gargalo:

  Se a propriedade *average* for:

  1. >= 5 E Utilização de Processador >= 85%.

  Descrição e indicação de melhoria: Caso há mais tarefas prontas para execução do que processadores, as *thread queues* aumentam. A fila do processador é a coleção de *threads* que estão prontas mas não aptas a ser executadas pelo processador pois outra *thread* ativa está executando um processo no momento. Uma fila sustentada por mais de duas *threads* é uma clara indicação de um gargalo. Um throughput maior pode ser alcançado com a diminuição de paralelismo nesses casos.

  Fonte: (WINDOWS. . . , 2017).

- ***Linux_System_ContextSwitchCount***

  Identificação de Gargalo:

  Quando a propriedade *ContextSwitchCount* for:

  >= 0 E a duração for >= 5%.

  Fonte: (WINDOWS. . . , 2017).

- ***Windows_Memory_PagesperSec***

  Identificação de Gargalo:

  Se a propriedade *average* for >=500.

  Descrição e indicação de melhoria: A métrica indica a taxa que as páginas são lidas ou escritas do disco devido ao alto número de *page faults*.

  Fonte: (WINDOWS. . . , 2017).

- ***Windows_Memory_PoolNonpagedFailures***

  Identificação de Gargalo:

  Se a propriedade *average* for != 0.

  Descrição e indicação de melhoria: A métrica indica o número de vezes que que as páginas não são encontradas na memória, indicando a memória estar com pouco tamanho. Também pode indicar a ocorrência de *memory leak*.

  Fonte: (WINDOWS. . . , 2017).

- ***Memory_Cache_Faultssec***

  Identificação de Gargalo:

  Se a propriedade *faults per sec* for > 0 E a duração maior que 10%.

  Descrição e indicação de melhoria: A métrica indica a frequência do sistema operacional procurando por dados na *cache* mas falhando ao encontrar. A métrica em altas taxas podem indicar espaço insuficiente em memória ou má localização dos dados.

  Fonte: (WINDOWS. . . , 2017).

- ***Windows_Disk_DiskTimePerCent***

  Identificação de Gargalo:

  Se a propriedade *average* for de acordo com a prioridade em ordem crescente:

  1. *average* >= 90%

  2. *average* >= 55% e a duração for >= 30%

  Descrição e indicação de melhoria: A métrica indica o número de vezes que que as páginas não são encontradas na memória, indicando a memória estar com pouco tamanho. Também pode indicar a ocorrência de *memory leak*.

  Fonte: (WINDOWS. . . , 2017).

- ***PhysicalDisk_avg_DiskRead_QueueLength***

  Identificação de Gargalo:

  Se a propriedade *average* for < 2.

  Descrição e indicação de melhoria: A métrica indica o número de requisições de leitura que foram enfileirados para o disco durante o período de execução.

  Fonte: (WINDOWS. . . , 2017).

- ***PhysicalDisk_avg_DiskWrite_QueueLength***

  Identificação de Gargalo:

  Se a propriedade *average* for < 2.

  Descrição e indicação de melhoria: A métrica indica o número de requisições de escrita que foram enfileirados para o disco durante o período de execução.

  Fonte: (WINDOWS. . . , 2017).

- ***PhysicalDisk_Avg_Disk_secTransfer***

  Identificação de Gargalo:

  Se a propriedade *average* for < 0.018.

  Descrição e indicação de melhoria: A métrica indica uma alta quantidade de fragmentação no disco, disco lento ou falhas no disco. Valores menores que o indicado identificam necessidade de aumento na memória RAM.

  Fonte: (WINDOWS. . . , 2017).

- ***PhysicalDisk_DiskTransferssec***

  Identificação de Gargalo:

  Se a propriedade *average* for > 25.

  Descrição e indicação de melhoria: Quando a métrica ultrapassa 25 *Disk IO per second* o tempo de resposta de disco se torna degradado, indicando um gargalo.

  Fonte: (WINDOWS. . . , 2017).

- ***PhysicalDisk_DiskReadssec***

  Identificação de Gargalo:

  Se a propriedade *average* for > 70.

  Descrição e indicação de melhoria: Deve-se checar a taxa específica de transferência a fim de verificar que a taxa corrente não superou as especificações. É importante levar em conta que operações I/O geram grande impacto nos valores de leitura e escrita de disco. Em geral métricas aceitáveis estão entre 50 e 70 operações por segundo.

  Fonte: (WINDOWS. . . , 2017).

- ***Processor% Interrupt Time***

  Identificação de Gargalo:

  Se a propriedade *average* for > 15.

  Descrição e indicação de melhoria: Essa métrica é um indicador indireto da atividade de dispositivos que geram interrupções, como adaptadores de rede. Um crescimento significativo nesse valor médio indica degradação potencial no *hardware*.

  Fonte: (WINDOWS. . . , 2017) e Engenheiros de Performance 1, 2, 3, 4, e 5.

- ***Processor_InterruptPerSec***

  Identificação de Gargalo:

  Se a propriedade *average* for >= 1000.

  Descrição e indicação de melhoria: Um crescimento significativo na métrica sem o aumento de atividade no sistema correspondente indica degradação no *hardware*. Deve-se investigar algum dispositivo como adaptador de rede ou disco que possa estar causando o problema.

  Fonte: (WINDOWS. . . , 2017) e Engenheiros de Performance 1, 2, 3, 4, e 5.

- ***Windows_Network_Interface_PacketsPerSec***

  Identificação de Gargalo:

  Se a propriedade *average* for >= 100.

  Fonte: Engenheiros de Performance 6 e 7.

- ***Windows_physical disk idle time***

  Identificação de Gargalo:

  Se a propriedade *average* for < 20.

  Fonte: Engenheiros de Performance 6 e 7.

- ***WebLogic_ConnectionPool_CurrCapacity***

  Identificação de Gargalo:

  Se a propriedade *average* for >= 20%.

  Fonte: Engenheiros de Performance 6, 7 e 8.

- ***Threads_%ExecuteThreadIdleCount***

  Identificação de Gargalo:

  Se a propriedade *average* for <= 10% E a duração for maior que 1%.

  Fonte: Engenheiros de Performance 6, 7 e 8.

- ***Oracle_Sessions_%active***

  Identificação de Gargalo:

  Se a propriedade *average* for >= 80% E a duração for maior que 1%.

  Fonte: (SAS, 2016).

- ***Oracle_IndexedQueries_Percentage***

  Identificação de Gargalo:

  Se a propriedade *average* for <= 80% E a duração for maior que 1%.

  Fonte: (SAS, 2016).

- ***Oracle_Call Rates_Rollback***

  Identificação de Gargalo:

  Se a propriedade *average* for > 30.

  Fonte: (SAS, 2016).

- ***WebLogic_JvmProcessorLoad***

  Identificação de Gargalo:

  Se a propriedade *average* for:

  1. >= 85% em qualquer momento OU

  2. >= 70% durante 60% do tempo de execução OU

  3. >= 60% durante 70% do tempo de execução OU

  4. >= 50% durante 80% do tempo de execução.

  Fonte: Engenheiros de Performance 1, 2, 3 e 4.

- ***JDBCConnectionPoolRuntime_ActiveConnectionsCurrentCount***

  Identificação de Gargalo:

  A propriedade *average* for igual à média da métrica *WebLogic_ConnectionPool_MaxCapacity*.

  Fonte: Engenheiros de Performance 1, 2, 3 e 4.

- ***WebLogic_ConnectionPool_WaitingForConnectionCurrentCount***

  Identificação de Gargalo:

  A propriedade *name* for >= 3 E a duração for maior que 1%.

  Fonte: Engenheiros de Performance 9 e 10.

- ***JDBCConnectionPoolRuntime_ConnectionDelayTime***

  Identificação de Gargalo:

  A propriedade *average* for >= 3.

  Fonte: Engenheiros de Performance 9 e 10.

- ***WebLogic_ConnectionPool_ConnectionDelayTime***

  Identificação de Gargalo:

  A propriedade *average* for >= 3.

  Fonte: Engenheiros de Performance 9 e 10.

- ***ThreadPoolRuntime_StuckThreadCount***

  Identificação de Gargalo:

  A propriedade *average* for >= 0.

  Fonte: Engenheiros de Performance 9 e 10.

- ***ThreadPoolRuntime_HoggingThreadCount***

  Identificação de Gargalo:

  A propriedade *average* for >= 0.

  Fonte: Engenheiros de Performance 9 e 10.

- ***Thread_HoggingThreadCount***

  Identificação de Gargalo:

  A propriedade *average* for >= 0 .

  Fonte: Engenheiros de Performance 9 e 10.

- ***WorkManagerRuntime_PendingRequests***

  Identificação de Gargalo:

  A propriedade *name* for >= 5 E a duração for >= 1%.

  Fonte: Engenheiros de Performance 9 e 10.

- ***Linux_System_ProcessRunnable***

  Identificação de Gargalo:

  A propriedade *average* for igual à média da métrica *productionPool/MaxCapacity*.

  Fonte: Engenheiros de Performance 4.

- ***Linux_System_CPUUsed***

  Identificação de Gargalo:

  A propriedade *average* for >= 80% e a duração for >= 10%.

  Fonte: Engenheiros de Performance 4.

- ***Memory_% User Memory***

  Identificação de Gargalo:

  A propriedade *average* for >= 90%.

  Fonte: Engenheiros de Performance 4.