Detecção automática e classificação de anomalias em aplicações web

Fernando Lemes da Silva

Relatório apresentado ao
Instituto de Matemática e Estatística
da Universidade de São Paulo
para exame de qualificação de
Mestrado em Ciências

Programa: Mestrado em Ciência da Computação
Orientador: Prof. Dr. Alfredo Goldman

São Paulo
7 de janeiro de 2020

Detecção automática e classificação de anomalias em aplicações web

Fernando Lemes da Silva

Esta é a versão original do texto de
qualificação elaborado pelo candidato
Fernando Lemes da Silva, tal como
submetido à Comissão Julgadora.

Resumo

Fernando Lemes da Silva. **Detecção automática e classificação de anomalias em aplicações web**. Exame de Qualificação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

A proposta desta pesquisa é de investigar a detecção de anomalias em uma aplicação web de maneira automática, classificando-as de forma que ações corretivas possam ser programadas para manter a aplicação funcionando adequadamente.

Serão consideradas anomalias as requisições a uma aplicação web que demoraram mais tempo que o usual, ou que apresentaram erros em suas respostas. Este tipo de ocorrência pode indicar um problema na aplicação, na infraestrutura em que ela está sendo executada ou em alguma outra aplicação da qual esta depende.

O sistema de detecção de anomalias utilizará como dados de entrada os registros das requisições enviadas a uma aplicação web, os quais podem ser extraídos de um balanceador de carga ou servidor proxy, sem a necessidade de instrumentar ou alterar o código da aplicação.

Para a detecção das anomalias será empregado um mecanismo de aprendizado de máquina autônomo baseado em estatísticas. Para classificação das anomalias será utilizado um algoritmo de regressão logística que será pré-treinado com alguns cenários já conhecidos.

A importância deste estudo é de que, em um contexto de micro-serviços, este tipo de monitoramento é essencial para evitar que problemas em um componente possam eventualmente comprometer outros e colocar todo o sistema de software em uma situação degradada ou comprometida.

Palavras-chave: Micro-serviços. Monitoramento. Detecção de anomalias. Aprendizado de máquina.

Lista de Abreviaturas

ALB	Balanceador de carga de aplicação da AWS (*Application Load Balancer*)
AWS	Amazon Web Services
CPU	Unidade de Processamento Central (*Central Processing Unit*)
EC2	Nuvem Computacional Elástica da AWS (*Elastic Compute Cloud*)
HTTP	Protocolo de Transferência de Hipertexto (*HyperText Transfer Protocol*)
HTTPS	Protocolo Seguro de Transferência de Hipertexto (*HyperText Transfer Protocol Secure*)
I/O	Entrada e Saída (de Dados) (*Input and Output*)
RAM	Memória de Acesso Randômico (*Random Access Memory*)
S3	Sistema Simples de Armazenamento da AWS (*Simple Storage Service*)
SOA	Arquitetura Orientada a Serviços (*Service Oriented Architecture*)
URL	Localizador Uniforme de Recursos (*Uniform Resource Locator*)
vCPU	Unidade de Processamento Central Virtual (*Virtual CPU*)

Lista de Figuras

Lista de Tabelas

Lista de Programas

Sumário

Capítulo 1

Introdução

Com a popularização dos computadores e de diversos equipamentos "*smart*", como telefones celulares e televisores, houve um aumento significativo do contato das pessoas com sistemas de software. Os sistemas de software utilizados por estes equipamentos são os mais diversos, variando de programas para troca de mensagens, passando por comércio de bens e serviços, até compartilhamento de conteúdo.

Em virtude do grande número de usuários que estes sistemas possuem, eles precisam ser desenvolvidos visando a alta disponibilidade, garantindo a satisfação do usuário e assegurando seu sucesso. A alta disponibilidade de um sistema é definida pelo número de noves na porcentagem da garantia que a aplicação estará em funcionamento. Por exemplo, uma aplicação com quatro noves deve estar 99,99% do tempo disponível para ser acessada.

Apesar de quatro noves parecer uma disponibilidade bastante alta, o 0,01% do tempo que o sistema pode ficar indisponível representa 52 minutos e 33 segundos em um ano, o que é uma quantidade de tempo relativamente alta.

Independente do número de noves, é certo que falhas irão ocorrer e sistemas estarão sujeitos a degradação ou indisponibilidade, afetando uma grande quantidade de usuários.

Um cenário comum de degradação é de um sistema que passa a receber um número grande de acessos repentinamente e, por não estar dimensionado para esta carga, passa a demorar mais tempo para executar as ações solicitadas. Outro cenário possível é a indisponibilidade do sistema, onde são retornados erros para as ações solicitadas, por conta de uma falha ao acessar algum recurso como um banco de dados, ou outro sistema, do qual ele depende.

Alguns cenários de falha não ocorrem imediatamente, mas dão sinais de que algo está errado com alguma antecedência, e isto pode ser observado através do comportamento do sistema.

Técnicas de detecção de anomalias são utilizadas para modelar um comportamento, dado um conjunto de dados, e após este modelo ter sido criado, é possível avaliar o

quanto um novo conjunto de dados se parece com o que já foi visto durante a etapa de modelagem.

Utilizando dados de requisições que foram processadas por um sistema para modelar o seu comportamento, e determinando um valor de probabilidade limite para o que deve ser considerado normal ou não, é possível identificar requisições que foram processadas pelo sistema que sofreram alguma anomalia.

A partir de casos anômalos identificados em um sistema e de suas causas, é possível realizar o treinamento de um algoritmo de aprendizado de máquina para dado um caso anômalo determinar a causa mais provável desta anomalia. Com este algoritmo treinado e diante de novas ocorrências de anomalia, podemos identificar com uma boa precisão o comportamento anormal de um sistema e a causa dele.

Existem diversos sistemas comerciais que fazem monitoramento de sistemas, em diversas profundidades e com diferentes funcionalidades, como NewRelic[1], Dynatrace[2] e Datadog[3], entretanto esta pesquisa irá focar somente no monitoramento de requisições HTTP/HTTPS enviadas a uma aplicação, com o objetivo de limitar o amplo escopo existente na área de monitoramento de aplicações.

Dadas estas informações, o que esta pesquisa propõe é avaliar a utilização de algoritmos de aprendizado de máquina para detecção e classificação de anomalias, auxiliando a diagnosticar problemas em um sistema de software, melhorando a garantia de disponibilidade e por consequência a satisfação dos usuários.

Este documento irá apresentar a proposta de pesquisa iniciando por conceitos básicos necessários para o entendimento do problema, passará pela metodologia e proposta da solução, irá expor um experimento preliminar e finalizará com comparações a trabalhos relacionados similares.

[1] http://www.newrelic.com
[2] http://www.dynatrace.com
[3] http://www.datadoghq.com

Capítulo 2

Conceitos básicos

Neste capítulo serão detalhados alguns conceitos básicos relacionados ao problema abordado e a solução proposta para endereçá-lo.

Os sistemas de software alvos desta pesquisa são sistemas de médio ou grande porte que possuem uma arquitetura segmentada em vários serviços, visando garantir um baixo acoplamento entre várias partes que compõe o software.

Este tipo de arquitetura inicialmente foi conhecido como SOA (mais informações em *Service-Oriented Architecture: Concepts, Technology, and Design* (Erl, 2009)), sobre a qual serão dados mais detalhes a seguir.

2.1 Sistemas com arquitetura SOA

Um sistema criado com a arquitetura SOA é um sistema modularizado em vários serviços, em geral independentes, que em conjunto podem oferecer várias funcionalidades a um usuário.

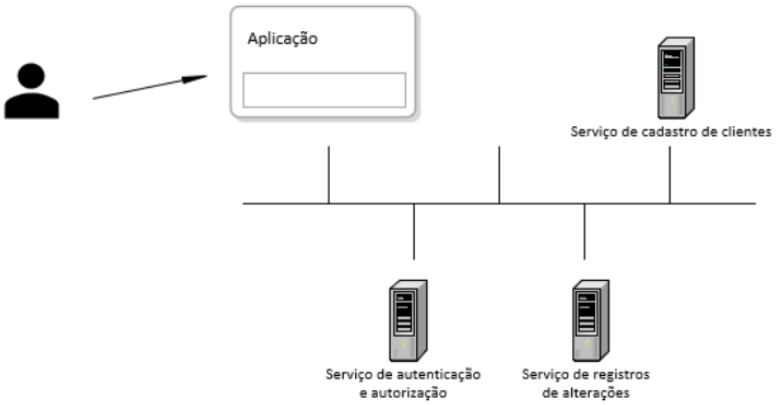

Figura 2.1: *Barramento de uma aplicação SOA.*

Na figura 2.1 encontra-se um diagrama simplificado de uma aplicação hipotética onde a interface com o usuário se comunica com vários serviços como: autenticação e autorização, cadastro de clientes e registros de alterações.

Para um usuário realizar alterações nos dados de um cliente ele irá utilizar a interface da aplicação, que por sua vez utilizará o serviço de autenticação e autorização para validar a identidade do usuário e se este possui as permissões necessárias para a alteração.

Em seguida, uma chamada ao serviço de cadastro de clientes será feita para realizar as alterações necessárias e por fim o serviço de registro de alterações será acionado para registrar que o usuário autenticado realizou uma alteração no cadastro de um cliente.

Com este exemplo é possível mostrar como vários serviços independentes podem ser utilizados para compor um sistema de software modular e de baixo grau de acoplamento.

2.2 Noções em escalabilidade e resiliência

Para que um sistema de software obtenha sucesso, ele precisa ter um grande número de usuários, e estes precisam estar satisfeitos com seu desempenho e disponibilidade. Estes dois requisitos não funcionais dependem de duas características do software: escalabilidade e resiliência.

A escalabilidade se traduz na capacidade de conseguir atender desde um pequeno número de usuários até centenas de milhares deles. Já a resiliência é caracterizada pela capacidade do sistema de se manter operante mesmo diante de falhas, sejam estas dentro ou fora de sua infraestrutura, que em algum momento irão ocorrer.

A escalabilidade de um sistema de software é habitualmente endereçada com a adição de novos recursos computacionais. A adição destes recursos pode ser tanto na forma vertical, com o aumento de recursos como processador e memória para uma instância do sistema em execução, como horizontal, adicionando mais máquinas ao sistema.

Já a resiliência por sua vez está atrelada com a arquitetura do sistema, a qual deverá prever que em algum momento as máquinas, conexões de rede, ou até mesmo um *data-center* inteiro poderão falhar. Com a arquitetura adequada, é possível criar um sistema de software que atenda ambas as necessidades de escalabilidade e resiliência.

No diagrama abaixo (figura 2.2) encontra-se um exemplo simplificado de arquitetura de um sistema de software executado em vários servidores, os quais estão localizados em dois *data-centers* fisicamente separados, atendendo diversos usuários que irão se conectar a estes servidores através de balanceadores de carga.

Alguns detalhes, como replicação de bancos de dados e mecanismos de direcionamento dos usuários aos balanceadores de carga, estão ocultos nesta visualização, mas de forma simplificada é desta maneira que um sistema de software deve ser arquitetado para que os requisitos de escalabilidade e resiliência sejam atendidos.

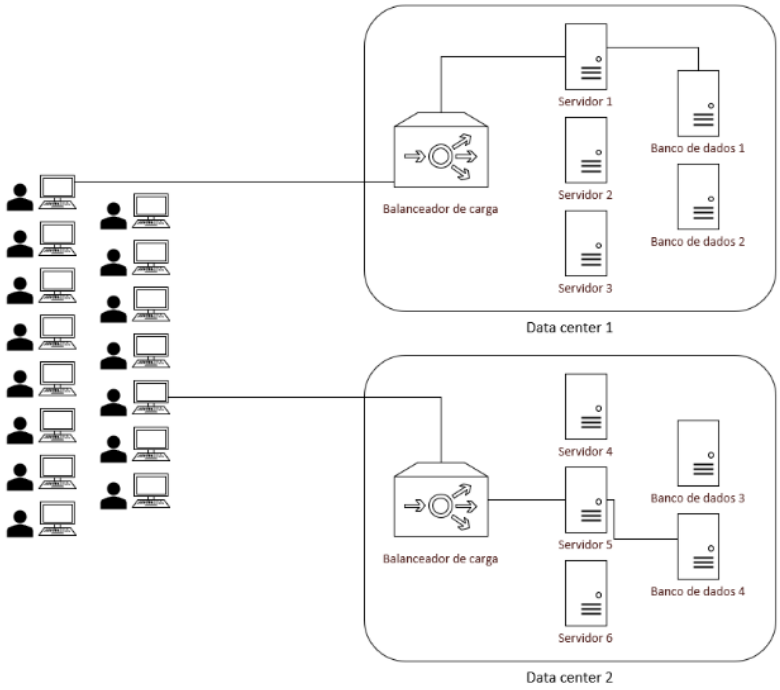

Figura 2.2: *Diagrama de aplicação resiliente.*

2.3 Balanceadores de carga

Os balanceadores de carga são peças chave para que um sistema de software possa ser escalável e resiliente (mais informações em *Availability and Load Balancing in Cloud Computing* (Zenon Chaczko e Mcdermid, 2011)). Estes componentes atuam intermediando as requisições dos usuários para os servidores, direcionando as requisições somente para servidores que estejam em pleno funcionamento.

A resiliência é garantida pois, se um destes servidores deixar de funcionar, ele não receberá mais requisições do balanceador de carga e também, se um dos balanceadores de carga deixar de funcionar, um outro balanceador poderá continuar fazendo o redirecionamento de requisições para todos os servidores ativos.

É importante notar que nunca deve haver somente um balanceador de carga, pois ele iria comprometer totalmente a disponibilidade da aplicação em caso de falha.

2.4 Micro-serviços, containers e Kubernetes

Com a computação em nuvem e o conceito de infraestrutura como código, o modelo de arquitetura SOA evoluiu para o que atualmente conhecemos como micro-serviços, o qual se diferencia principalmente pela independência na implantação de cada serviço.

O principal motivo para sua popularidade é a facilidade que cada micro-serviço passou a ter para definição suas necessidades de servidores, balanceadores de carga e bancos de

dados, através de arquivos texto que são processados por mecanismo de automação na sua implantação.

Na figura 2.3 é possível ver um exemplo de como uma aplicação pode ser composta de vários micro-serviços, executados de forma independente, mas com dependências entre si.

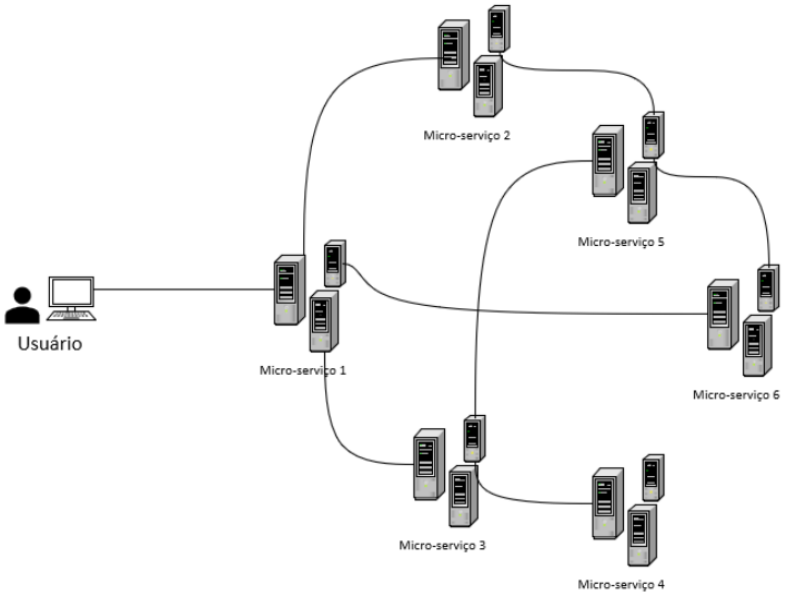

Figura 2.3: *Diagrama de micro-serviços.*

Como os micro-serviços que tratamos neste documento são aplicações que se comunicam por uma rede utilizando geralmente o protocolo HTTP, estes também são conhecidos como aplicações web.

Outro fator que posteriormente contribuiu ainda mais para a maior adoção desta arquitetura foram os containers Linux e sistemas de orquestração e gerenciamento de containers.

Os containers Linux permitem que um processo seja executado como se ele estivesse em uma máquina exclusiva, isolando bibliotecas dos quais ele depende e também oferecendo uma inicialização muito mais rápida que uma máquina virtual visto que o núcleo do sistema operacional é compartilhado pelos vários containers e este já está pronto para ser utilizado quando se inicia um container novo.

Para facilitar a gerência na execução de containers Linux foram criados diversos projetos como o Kubernetes[4], desenvolvido pela Google, e o Docker Swarm[5], desenvolvido pela Docker Inc., também responsável pela implementação de containers Linux denominada Docker[6].

Com a utilização destes projetos, é possível gerenciar automaticamente um conjunto

[4] http://kubernetes.io
[5] docs.docker.com/engine/swarm
[6] www.docker.com/resources/what-container

de máquinas para a execução de diversos containers de micro-serviços, garantindo a reinicialização da aplicação em caso de falha, e escalabilidade, iniciando novos containers nas máquinas conforme a necessidade.

2.5 Detecção de falhas

Para realizar uma análise de um sistema de software implantado utilizando a arquitetura de micro-serviços, independente de sua infraestrutura de implantação, esta pesquisa propõe uma análise não invasiva dos serviços utilizando somente os registros dos balanceadores de carga a frente dos micro-serviços.

Os registros dos balanceadores de carga contém informações bastante interessantes como os horários que as requisições iniciaram e foram concluídas, além de quantidades de dados trafegados, o serviço que foi acessado e o tempo que este serviço levou para ser executado. Estes dados devem ser suficientes para se traçar um perfil de comportamento, permitindo detectar casos que fugiram deste perfil e analisá-los com mais detalhes.

2.6 Classificação de falhas

Uma vez que as falhas podem ser detectadas, estas podem ser classificadas conforme sua causa raiz, que pode variar desde um excesso de utilização a uma falha de um banco de dados, por exemplo.

Com experimentos provocando falhas simuladas, é possível realizar o treinamento de um algoritmo de aprendizado de máquina para que novas falhas de aplicações reais possam ser classificadas, permitindo inferir a causa provável de um problema na aplicação.

O objetivo final desta pesquisa é de detectar e classificar uma falha em um serviço o quanto antes pois, dado o cenário complexo de micro-serviços exposto acima, uma falha em uma aplicação pode causar impactos em outras aplicações, o que poderia causar problemas maiores para todo o sistema de software.

Capítulo 3

Metodologia

Neste capítulo será detalhado o método que se deseja utilizar para realizar a detecção de anomalias que esta pesquisa propõe.

Para a análise do comportamento da aplicação serão coletados dados de aplicações web de forma não invasiva. O principal motivo para utilizar estes dados está na facilidade com que eles podem ser obtidos sem a necessidade que as aplicações tenham que ser instrumentadas para dar informações de seu comportamento.

Para obtenção destes dados serão utilizados registros das requisições e respostas intermediadas por balanceadores de carga, os quais contém o mínimo de informação necessária para a avaliação proposta, e tratando assim a aplicação como uma caixa preta (figura 3.1) da qual não precisamos conhecer nenhum detalhe interno.

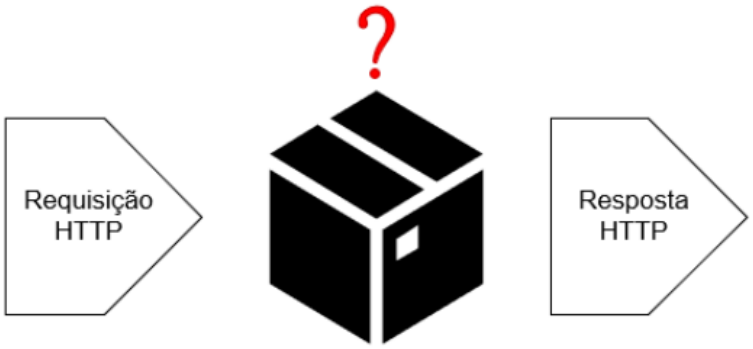

Figura 3.1: *Aplicação tratada como uma caixa preta.*

Outro motivo para esta abordagem é a possibilidade deste mecanismo ser genérico o suficiente para se tornar um serviço de monitoramento de outros serviços, o que é bastante interessante para a empresa HP Inc., da qual sou funcionário, que irá fornecer dados de aplicações em produção para realização de testes.

Na figura 3.2 é descrita a arquitetura de uma aplicação real que realiza o armazenamento de arquivos em diversos provedores.

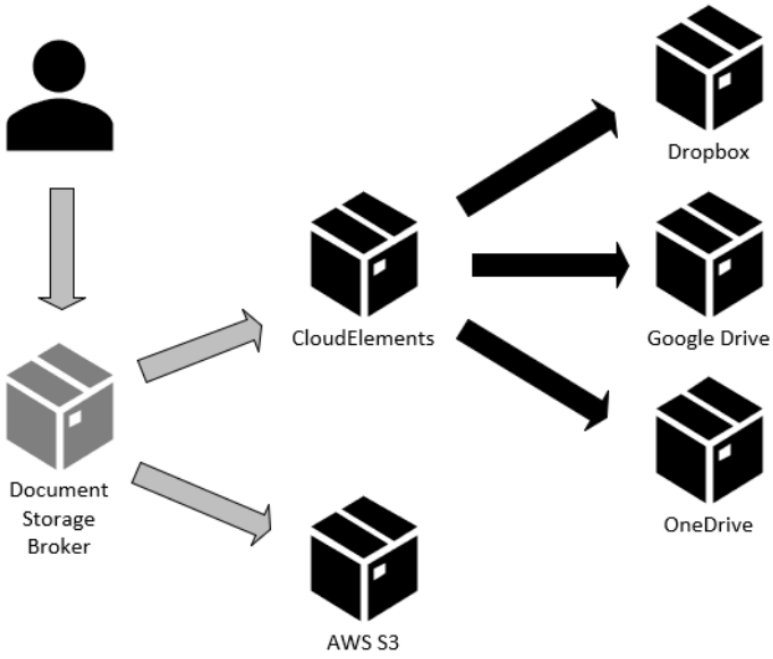

Figura 3.2: *Interação entre serviços do Document Storage Broker.*

O serviço Document Storage Broker recebe requisições dos usuários e encaminha para o serviço de outra empresa chamada CloudElements[7], intermediária de outros serviços como Dropbox, Google Drive ou OneDrive, ou para o serviço de armazenamento de arquivos da AWS, conforme parâmetros enviados pelo usuário.

Nesta arquitetura seria possível utilizar a metodologia apresentada tanto nas requisições recebidas pelo Document Storage Broker quanto nas que são enviadas para as empresas CloudElements e AWS.

De posse destes dados serão utilizadas técnicas de aprendizado de máquina de detecção de anomalias e regressão logística para tentar responder às seguintes questões:

1. A aplicação web está se comportando de forma anômala?
2. Qual a causa provável da anomalia?

Alguns cenários esperados para anomalia e causa são: aumento de tempo de resposta dado um aumento no número de usuários acessando a aplicação, e aumento na quantidade de erros dada uma falha em uma dependência da aplicação.

Os dados a serem avaliados inicialmente serão provenientes de uma aplicação de teste com cenário de sobrecarga e posteriormente serão realizados testes utilizando dados anonimizados de uma aplicação em produção da empresa HP Inc., os quais estão pendentes de autorização da empresa para sua publicação.

Caso não seja permitida a publicação dos registros da HP Inc., serão publicados registros novos da aplicação de teste que irão simular os cenários que desejamos detectar.

[7] http://www.cloud-elements.com

Capítulo 4

Proposta de trabalho

Inicialmente partiremos de registros de requisições processadas por uma aplicação de teste para detectar e analisar anomalias que tenham acontecido durante o tempo em que os registros foram coletados. Posteriormente, será implementado um modelo onde esta análise ocorrerá de forma contínua, com um período de aprendizado no início seguido do aprimoramento constante conforme a aplicação é utilizada.

Para a avaliação da aplicação de teste as seguintes etapas serão necessárias:

1. Coleta de registros de requisições;
2. Filtragem dos registros de requisições;
3. Agregação de registros e extração de novas características;
4. Avaliação da URL acessada;
5. Treinamento dos modelos;
6. Detecção de caso anômalo;
7. Avaliação de melhoria contínua do treinamento inicial;
8. Classificação de caso anômalo;
9. Agregação de casos anômalos classificados.

4.1 Etapa 1 - Coleta de registros de requisições

Os registros das requisições processadas serão extraídos a partir de balanceadores de carga conforme o diagrama abaixo sem adicionar complexidade ou atraso na resposta do serviço uma vez que os balanceadores de carga já são elementos necessários para garantir a alta disponibilidade de uma aplicação web.

Existem muitas opções de balanceadores de carga possíveis, entretanto as informações abaixo relacionadas estão presentes nas principais aplicações ou serviços utilizados

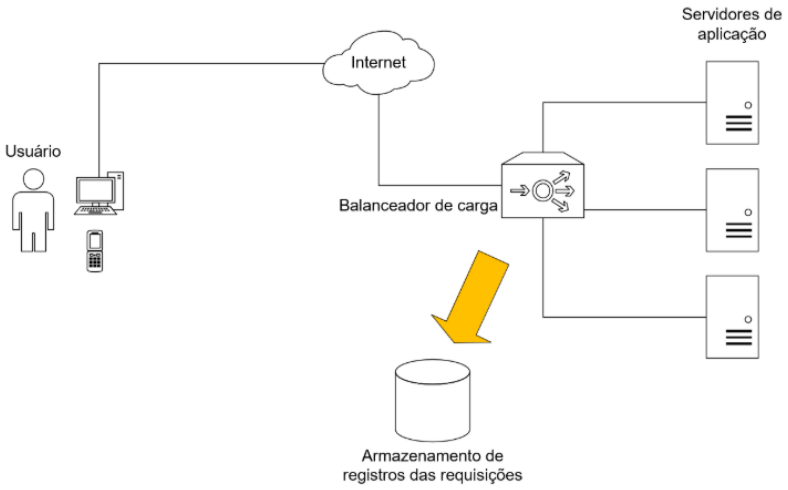

Figura 4.1: *Extração de dados de balanceadores de carga.*

como balanceadores de carga (por exemplo: Nginx[8], HA-Proxy[9], AWS Application Load Balancer[10]):

- Data e hora da solicitação;
- URL solicitada;
- Verbo HTTP utilizado;
- Instância/máquina que realizou o processamento;
- Quantidade de bytes recebidos;
- Quantidade de bytes enviados;
- Tempo de processamento;
- Código de resposta HTTP.

Como cada balanceador de carga tem seu formato de registro de uma requisição intermediada, estes registros serão convertidos em um formato padrão que será utilizado pelas etapas seguintes.

4.2 Etapa 2 - Filtragem dos registros de requisições

Esta etapa é necessária para filtrar eventuais registros indesejados dos balanceadores de carga pois eventualmente o balanceador de carga pode ser responsável por mais de uma aplicação web, e seus registros podem conter dados que não desejamos tratar.

Também existem registros de falhas indeterminadas, ou com ausência das informações apontadas na etapa 1, que desejamos remover para evitar ruídos nos dados.

[8] http://www.nginx.com
[9] http://www.haproxy.org
[10] http://aws.amazon.com/elasticloadbalancing

O caso específico de falha na comunicação entre o balanceador de carga e a aplicação deverá ter o código de resposta HTTP mapeado para zero para que este cenário importante possa ser detectado, pois pode sinalizar falhas em uma instância.

4.3 Etapa 3 - Agregação e extração de características dos registros para o aprendizado de máquina

Nesta etapa será extraída a informação de taxa de requisições concorrentes, que não fazem parte de um registro isolado, mas dependem de um conjunto de registros que estavam sendo processados no mesmo espaço de tempo.

Tomando uma janela de tempo fixa, ou de acordo com o tempo que a requisição levou para ser processada, podemos recuperar a taxa de requisições a qual a aplicação web estava sendo submetida.

Para o cálculo deste valor será utilizado como identificador somente a identificação da instância que processou a requisição, independente de verbo HTTP e URL, uma vez que desejamos saber a taxa de requisições concorrentes da instância e não de uma requisição específica.

A mesma abordagem pode ser utilizada para o tráfego de dados para tentar identificar que o gargalo na execução de uma requisição são as interfaces de comunicação com a rede, porém por não ser certo que os bytes trafegados foram distribuídos uniformemente durante o intervalo de tempo da requisição esta métrica será deixada de lado neste momento.

Para a recuperação da taxa de requisições concorrentes será implementado um contador de requisições ativas para cada janela de 1 segundo, e durante o intervalo de tempo em que a requisição esteve ativa os contadores serão incrementados. O intervalo de 1 segundo é o que se espera ser o mais adequado, entretanto durante a execução da pesquisa será avaliado se este valor é de fato o mais adequado.

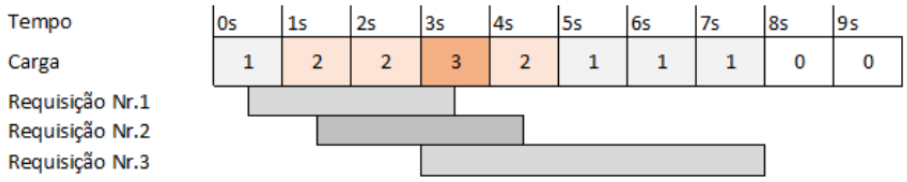

Figura 4.2: *Carga de requisições ao longo do tempo.*

Adicionalmente serão extraídas novas características para o valor de resposta HTTP dado que um modelo de aprendizado de máquina não conseguiria estabelecer uma relação linear para os códigos de retorno. Para isto serão isolados em características distintas e mapeados de acordo com a tabela abaixo:

Resposta HTTP	Vetor de características
0	[0, 0, 0, 0, 0]
100 a 199	[1000, 0, 0, 0, 0] a [1099, 0, 0, 0, 0]
200 a 299	[0, 1000, 0, 0, 0] a [0, 1099, 0, 0, 0]
300 a 399	[0, 0, 1000, 0, 0] a [0, 0, 1099, 0, 0]
400 a 499	[0, 0, 0, 1000, 0] a [0, 0, 0, 1099, 0]
500 a 599	[0, 0, 0, 0, 1000] a [0, 0, 0, 0, 1099]

Tabela 4.1: *Mapeamento de respostas HTTP para um vetor.*

Acima estão sendo utilizados os valores de dois parâmetros que serão ajustados posteriormente de acordo com os resultados dos testes. São estes:

Parâmetro	H_{min}
Valor proposto	1000
Descrição	Valor mínimo caso a categoria de respostas HTTP esteja ativa.

Tabela 4.2: *Parâmetro H_{min}.*

Parâmetro	H_{range}
Valor proposto	100
Descrição	Valor para o qual os valores da categoria X, de X00 a X99 deverão ter sua escala ajustada.

Tabela 4.3: *Parâmetro H_{range}.*

Ou seja, o vetor de características R_{vector} para um código de resposta R poderia ser gerado com:

Programa 4.1 Código para cálculo do vetor R_{vector}.

```
1   int[] getRVector(int R, int hMin, int hRange) {
2      int[] rVector = new int[] { 0, 0, 0, 0, 0 };
3      if (R > 100) {
4         rVector[ (int) Math.floor( R / 100 ) - 1 ] = hMin + (int) ((float) (R \%
              100) / 100 * hRange);
5      }
6      return rVector;
7   }
```

Os valores 1000 e 100 estão sendo utilizados para que o valor 0, que representa a não ocorrência daquela característica, esteja suficientemente distante e que permita ainda as variações dentre a categoria de resposta serem detectadas. O valor 0 que representa ausência de resposta HTTP está sendo mapeado para um vetor zero que difere bastante dos outros valores em caso de resposta.

Posteriormente será avaliada a necessidade de normalizar as características coletadas para que a detecção seja mais eficiente, porém neste momento da pesquisa esta etapa aparenta não ser necessária.

Também será analisada a ideia de se isolar códigos de erro específicos que indiquem algum cenário, como por exemplo:

- 401 (Unauthorized) e 403 (Forbidden): Indicam falhas relacionadas a permissão de acesso;
- 502 (Bad Gateway) e 504 (Gateway timeout): Indicam possíveis falhas com serviços do qual a aplicação depende.

4.4 Etapa 4 - Avaliação da URL acessada

Esta etapa determina como a URL será utilizada para ter seus registros agrupados em uma forma que seja possível determinar quais URLs correspondem a um determinado serviço da aplicação web.

Existe um problema crítico na utilização da uma URL para identificar um tipo de serviço, visto que identificadores de recursos são bastante comuns em parte do caminho acessado. Por exemplo, em uma aplicação de cuida de cadastros de clientes pode ter requisições como as abaixo:

Verbo	Caminho do recurso
POST	/clientes
GET	/clientes/42
PUT	/clientes/42
GET	/clientes/99
GET	/clientes/999
GET	/health

Tabela 4.4: *Exemplos de requisições HTTP.*

Das requisições acima relacionadas, as de verbo “GET” para o caminho “/clientes/{número}” são bastante parecidas e provavelmente devem executar um código de recuperação de dados do cliente identificado pelo número.

Para agrupar estes dados será utilizada uma estrutura de árvore com os caminhos acessados e os nós mais “comuns” terão uma quantidade de registros maior que os demais.

Ao receber um registro de uma requisição a árvore será percorrida e os nós que fazem parte do caminho deste registo terão uma referência para este registro, desta forma será possível avaliar quais são os nós que tem relevância para agrupar registros por um número suficientemente grande de referências, porém com nós filhos com poucas referências.

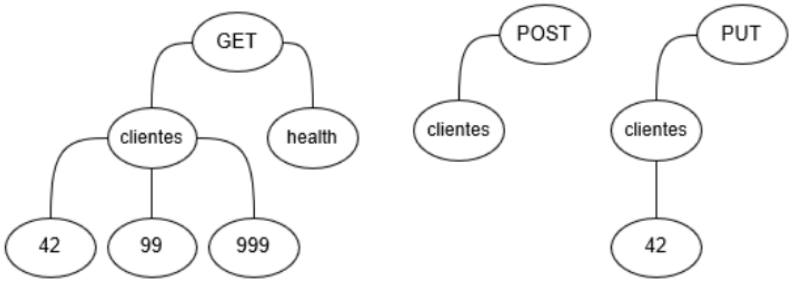

Figura 4.3: *Caminho da URL em uma árvore.*

Será avaliada também a ideia de utilizar expressões regulares para definir os serviços monitorados, pois em alguns cenários a utilização do caminho inicial comum pode não ser suficiente para identificar o grupo de requisições.

4.5 Etapa 5 - Treinamento dos modelos

Uma vez que as etapas anteriores já coletaram e processaram os dados das requisições, o treinamento dos modelos poderá ser iniciado. Caso existam dados salvos de treinamentos anteriores, eles poderão ser carregados para serem aprimorados com novos registros.

O treinamento será independente para cada conjunto de verbo HTTP, caminho parcial da URL e instância monitorada, conforme descrito na etapa 4, desta forma estaremos monitorando sempre o mesmo código executado da aplicação web apesar de que sua execução poderá ser diferente de acordo com os parâmetros.

Dado que estamos considerando a instância monitorada, também será possível na etapa de classificação identificar se as anomalias acontecem em algumas instâncias somente.

Durante o desenvolvimento desta pesquisa serão avaliados diversos parâmetros que poderão ser alterados conforme a necessidade. Inclusive descobrir quais parâmetros deverão ser alterados e para que valores é parte do desafio de pesquisa. São estes:

Parâmetro	N_{min}
Valor proposto	1000
Descrição	Número mínimo de registros que deverão ser utilizados no treinamento de um modelo.

Tabela 4.5: *Parâmetro N_{min}.*

Parâmetro	T_{min}
Valor proposto	86400
Descrição	Tempo mínimo em segundos que as requisições devem ser coletadas para um treinamento sem dados prévios.

Tabela 4.6: *Parâmetro T_{min}.*

A etapa de detecção de anomalias irá se basear em avaliação de probabilidade em uma distribuição multivalorada gaussiana (mais informações em (Ng, 2014)), que será treinada uma vez que ambos os critérios de N_{min} e T_{min} sejam obedecidos.

Assim que houver a quantidade de amostras necessária, as fórmulas abaixo serão aplicadas para m registros x_i e os valores de μ e Σ corresponderão ao treinamento para estes registros.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x_i, \mu \in \mathbb{R}^d \tag{4.1}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m} \left[(x_i - \mu)(x_i - \mu)^T\right], \Sigma \in \mathbb{R}^{d\times d} \tag{4.2}$$

4.6 Etapa 6 - Detecção de caso anômalo

Para a detecção de caso anômalo será calculada a probabilidade $p(x_i)$ de um caso x_i ser semelhante aos casos observados no cálculo de μ e Σ, e o valor limite será definido através do parâmetro P_{min}.

Parâmetro	P_{min}
Valor proposto	0.25
Descrição	Probabilidade mínima requerida para que o registro sob análise não seja considerado anômalo.

Tabela 4.7: *Parâmetro P_{min}.*

Caso a probabilidade $p(x_i)$ seja menor que P_{min} a ocorrência x_i será classificada como anômala e passará pela etapa de classificação (etapa 8).

$$p(x_i; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} exp\left(-\frac{1}{2}(x_i - \mu)^T\Sigma^{-1}(x_i - \mu)\right) \tag{4.3}$$

4.7 Etapa 7 - Melhoria contínua do treinamento inicial

O treinamento inicial será realizado assim que houver a quantidade mínima de registros necessários, entretanto após este treinamento inicial ser realizado, o modelo definido por μ e Σ deverá ser atualizado de regularmente com todos os novos registros observados, tenham estes sido classificados como anômalos ou não.

Parâmetro	T_{update}
Valor proposto	1800
Descrição	Tempo mínimo em segundos para que novos registros sejam considerados no novo treinamento do modelo.

Tabela 4.8: *Parâmetro T_{update}.*

Parâmetro	P_{update}
Valor proposto	0.02
Descrição	Percentual em que os novos registros irão influenciar um modelo treinado.

Tabela 4.9: *Parâmetro P_{update}.*

De forma paralela a detecção, um algoritmo de atualização será executado a cada T_{update} segundos para atualização dos valores μ e Σ com n novas amostras coletadas. As fórmulas utilizadas na etapa 5 serão alteradas de forma que os novos registros tenham um peso limitado sobre os valores de μ e Σ atuais, conforme abaixo:

$$\mu' = (1 - P_{update})\mu + P_{update}\frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.4}$$

$$\Sigma' = (1 - P_{update})\Sigma + P_{update}\frac{1}{n}\sum_{i=1}^{n} \left[(x_i - \mu')(x_i - \mu')^T\right] \tag{4.5}$$

A importância de se limitar o quanto os novos registros irão afetar o treinamento é de impedir que uma falha que dure vários minutos possa influenciar no aprendizado de forma que estas falhas deixem de ser detectadas. Adicionalmente também é importante limitar a influência de registros antigos para que mudanças na aplicação possam influenciar o treinamento existente.

4.8 Etapa 8 - Classificação de caso anômalo

Para a classificação do caso anômalo será utilizado o valor de $\delta_i = (x_i - \mu)$ como dado de entrada pois ele representa o quanto cada característica diferiu do valor médio esperado para ela.

Os vetores δ_i serão eventualmente pré-processados para normalizar os valores e utilizando um algoritmo de regressão logística (mais informações em (Yaser S. Abu-Mostafa e Lin, 2012)), ou outro mecanismo de aprendizado de máquina supervisionado, serão classificados como algum dos tipos de falha abaixo:

- Aplicação sobrecarregada;
- Aplicação sobrecarregada e irresponsiva;
- Falha em dependência da aplicação.

Dado que a ocorrência de falhas nas aplicações web não é algo frequente, serão simulados alguns cenários com uma aplicação de teste para realizar o treinamento deste modelo de aprendizado de máquina. A aplicação de teste será submetida aos seguintes cenários:

1. Aumento de carga além do esperado provocando um aumento no tempo de resposta mas sem gerar erros. Os resultados anômalos detectados serão classificados como "Aplicação sobrecarregada";
2. Aumento de carga além do esperado provocando um aumento no tempo de resposta gerando erros, ou vindos da aplicação ou do balanceador de carga. Os resultados anômalos detectados serão classificados como "Aplicação sobrecarregada e irresponsiva";
3. Remoção de dependência, como um banco de dados, provocando erros na resposta da aplicação. Os resultados anômalos serão classificados como "Falha em dependência da aplicação".

4.9 Etapa 9 - Agregação de casos anômalos classificados

Uma vez que as anomalias forem classificadas, será necessário avaliar a quantidade de anomalias, para avaliar casos isolados, se elas foram classificadas da mesma forma ou se são classificações variadas e ainda se é uma única instância que está sofrendo desta anomalia ou várias delas.

Os resultados previstos a serem reportados são:

- Nenhuma anomalia
- Pequenas anomalias detectadas
- Instância defeituosa

- Aplicação sobrecarregada
- Aplicação sobrecarregada e irresponsiva
- Falha em dependência da aplicação
- Falha geral na aplicação.

O parâmetro A_{min} será utilizado para definir se o resultado final da análise será de "Pequenas anomalias detectadas", indicando que algo anormal foi detectado mas não tem relevância suficiente para disparar algum alarme ou ação automática. Abaixo segue a definição deste parâmetro:

Parâmetro	A_{min}
Valor proposto	0.01
Descrição	Percentual mínimo de requisições anômalas para que estas sejam apontadas.

Tabela 4.10: *Parâmetro A_{min}.*

Uma vez que a quantidade de anomalias sejam significantes deverá ser avaliado se somente uma instância está sendo afetada. Em caso positivo o resultado final da análise será de "Instância defeituosa".

Caso múltiplas instâncias sejam afetadas o resultado final da análise será dado pela classificação que tenha mais de 50% das ocorrências e caso não exista uma que possua esta quantidade o resultado reportado deverá ser de "Falha geral na aplicação". Este percentual inicialmente não será parametrizado, porém conforme resultados da pesquisa ele poderá ser alterado.

Capítulo 5

Experimento preliminar

Foi realizado um experimento de teste utilizando a nuvem pública de computação da Amazon Web Services usando uma máquina virtual EC2 com 1 vCPU e 1GB de memória RAM. Para ter o cenário de um balanceador de carga na frente da aplicação foi criado um Application Load Balancer direcionando o tráfego HTTP para esta máquina e coletando os registros das requisições no serviço de armazenamento S3.

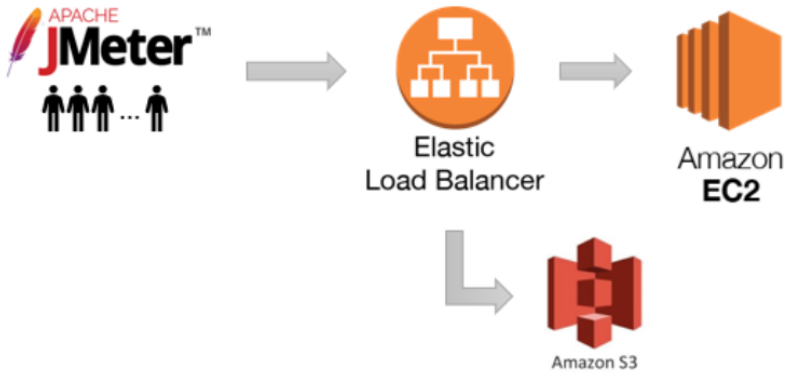

Figura 5.1: *Diagrama do experimento preliminar.*

A aplicação web foi criada utilizando SpringBoot atendendo requisições GET que executavam o código Java abaixo, consumindo tanto CPU quanto memória:

Programa 5.1 Código de teste da aplicação web.

```
1   List<Integer> array = new ArrayList<>();
2   for (int index = 0; index < 1000000; index++) {
3     array.add((int) (Math.random() * Integer.MAX_VALUE));
4   }
5   Collections.sort(array);
```

O teste da aplicação foi feito utilizando JMeter para com o número de usuários concorrentes variando ao longo do tempo chegando a 10 usuários concorrentes.

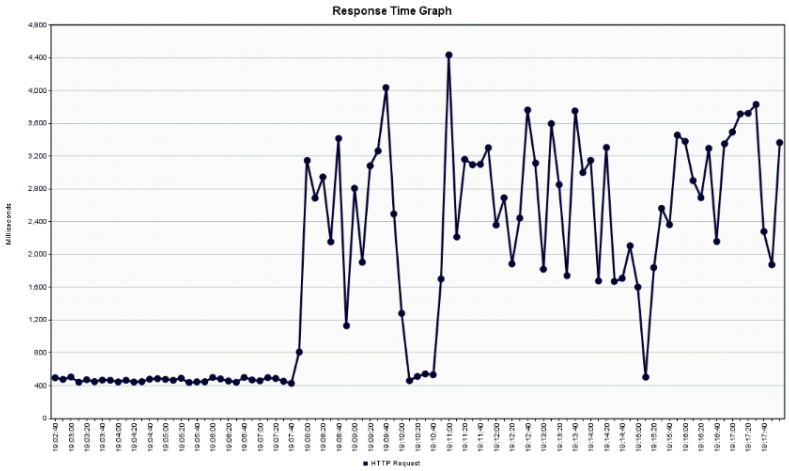

Figura 5.2: *Tempo de resposta da aplicação ao longo do tempo.*

A figura acima mostra o tempo de resposta da requisição ao longo da execução do teste. A aplicação se comporta bem no início, porém após os créditos de CPU da instância se esgotarem ela começa a ter tempos de resposta bastante altos, e será justamente esta anomalia que desejamos identificar neste experimento.

Os créditos de CPU mencionados são uma forma da AWS permitir que a máquina utilize um pouco mais de CPU durante picos de uso, pois créditos de CPU são incrementados até um certo limite caso a máquina esteja ociosa, ou consumidos em caso de alta carga.

Foram coletados 565 registros do ALB correspondentes a este teste e algumas amostras estão nas tabelas abaixo, onde os valores das colunas correspondem a:

1. Data e hora do início da requisição no ALB no formato ISO-8601;
2. Data e hora do término da requisição no ALB no formato ISO-8601;
3. Instância de destino que executou a requisição;
4. Tempo de processamento da requisição no destino;
5. Código de resposta HTTP;
6. Quantidade de bytes enviados a aplicação;
7. Quantidade de bytes recebidos da aplicação;
8. Verbo HTTP utilizado pela requisição;
9. Caminho da URL utilizado na requisição.

Estes exemplos são registros regulares:

1	2	3	4	5	6	7	8	9
2019-11-21T21:02:56.898000Z	2019-11-21T21:02:57.250351Z	172.31.7.121:8080	0.351	200	292	175	GET	/test
2019-11-21T21:02:57.886000Z	2019-11-21T21:02:58.233481Z	172.31.7.121:8080	0.347	200	292	175	GET	/test
2019-11-21T21:02:58.877000Z	2019-11-21T21:02:59.220909Z	172.31.7.121:8080	0.343	200	292	175	GET	/test
2019-11-21T21:02:59.868000Z	2019-11-21T21:03:00.348951Z	172.31.7.121:8080	0.480	200	292	175	GET	/test
2019-11-21T21:03:00.984000Z	2019-11-21T21:03:01.337323Z	172.31.7.121:8080	0.352	200	292	175	GET	/test

Estes exemplos foram considerados anômalos:

1	2	3	4	5	6	7	8	9
2019-11-21T21:16:04.811655Z	2019-11-21T21:16:00.847000Z	172.31.7.121:8080	3.964	200	328	174	GET	/test
2019-11-21T21:16:09.250483Z	2019-11-21T21:16:05.195000Z	172.31.7.121:8080	4.054	200	328	174	GET	/test
2019-11-21T21:16:13.374715Z	2019-11-21T21:16:09.344000Z	172.31.7.121:8080	4.030	200	328	174	GET	/test
2019-11-21T21:16:32.603263Z	2019-11-21T21:16:28.658000Z	172.31.7.121:8080	3.945	200	328	174	GET	/test
2019-11-21T21:16:51.868990Z	2019-11-21T21:16:47.794000Z	172.31.7.121:8080	4.074	200	328	174	GET	/test

Tabela 5.1: *Exemplos de registros regulares e anômalos do balanceador de carga.*

Além do aumento no tempo de execução (coluna 3) é possível notar um aumento discreto no tamanho da quantidade de bytes recebidos, talvez por algum cabeçalho HTTP adicional, mas ainda não foi investigado o que provocou esta alteração.

Utilizando os primeiros 300 registros com respostas bem similares para a etapa de treinamento da distribuição gaussiana multivalorada, marcamos os registros com menos de 10% de probabilidade como anômalos e os resultados foram plotados na figura 5.3.

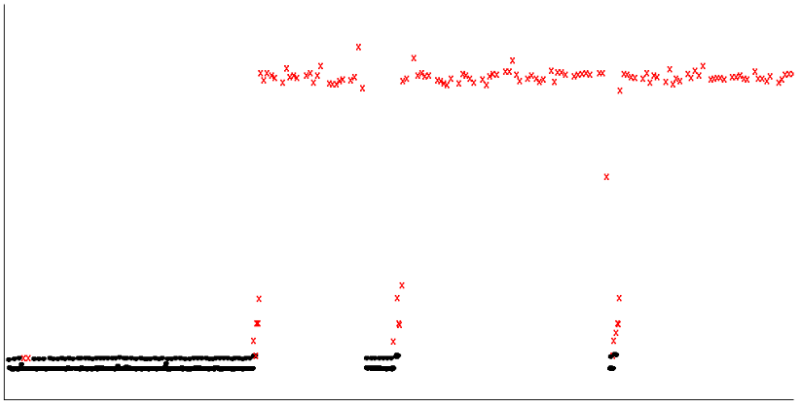

Figura 5.3: *Anomalias detectadas no experimento.*

É possível ver que a classificação de anomalia funciona bem neste cenário controlado e que alguns registros que deveriam ser considerados normais foram marcados como anômalos.

Logo no começo é possível ver dois pontos vermelhos que representam duas amostras que levaram o tempo de 483 e 488 milissegundos enquanto a carga da aplicação percebida era de 1 execução concorrente. O tempo médio de resposta no treinamento era de 375 milissegundos e dada a pouca variação ocorrida nos primeiros 300 registros estes tempos foram marcados como anormais.

Pouco antes de começarem a ocorrer várias falhas com tempos bastante altos foi detectado um registros com 514 milissegundos de execução. Como a informação de carga era de que haviam duas execuções concorrentes neste instante a requisição foi marcada como normal.

Com esta observação é possível afirmar que a informação de carga na aplicação estava de fato sendo considerada na detecção de anomalia, pois a requisição que levou 483 milissegundos no início do teste (sem concorrência) foi identificada como anômala.

Capítulo 6

Trabalhos relacionados

Foi realizada uma pesquisa por trabalhos semelhantes e a maioria dos trabalhos encontrados possui diferenças significantes em relação a proposta desta pesquisa.

A busca foi realizada tanto nas buscas gerais do Google quanto no Google Scholar com as seguintes palavras-chave:

- anomaly detection logs
- anomaly detection log records
- anomaly detection application
- anomaly detection application logs
- anomaly detection distributed application
- anomaly detection http
- anomaly detection load balancer

Os trabalhos similares encontrados serão detalhados a seguir.

6.1 Anomaly Detection for Application Log Data (Grover, 2018)

Este trabalho possui uma proposta similar de detectar falhas em uma aplicação em execução de forma não invasiva, detectando anomalias através da saída de texto da aplicação.

A extração de características é bem diferente e várias abordagens para detecção de anomalia são propostas, como: detecção baseada em estatística, agrupamento por similaridade ("clustering"), distância de uma ocorrência ao grupo similar mais próximo, aprendizado supervisionado e redes neurais.

Esta pesquisa propõe utilizar a detecção baseada em estatística por acreditar que é o método mais adequado ao cenário, entretanto estas opções serão melhor estudadas para serem consideradas na escrita da dissertação.

Em relação à classificação, este trabalho se restringe somente a detecção de uma anomalia na aplicação sem classificá-la, pois a idéia é somente apontar entre as milhares de linhas de saída da aplicação o que está fora do padrão.

6.2 Anomaly Detection in Log Records (Rastogi *et al.*, 2018)

Este outro trabalho também trata da detecção de anomalia baseada em arquivos de logs de um servidor web Apache, porém com uma abordagem bem mais simplificada.

A detecção de anomalia proposta é sobre a quantidade de requisições HTTP que partem de uma mesma origem (endereço IP) ao longo do tempo, e o foco da análise é de predizer estas ocorrências em um futuro próximo.

Apesar da ideia de detecção ser similar, o propósito do artigo é mais focado em prever o uso futuro dos servidores web, o que é bem distinto desta pesquisa.

6.3 PAD: Performance Anomaly Detection in Multi-server Distributed Systems (Peiris *et al.*, 2014)

Esta publicação descreve um software que foi criado para analisar anomalias de desempenho em aplicações distribuídas que possuem alta demanda e pedem uma baixa latência de resposta.

Ela difere bastante por ser mais invasiva e requerer contadores de recursos (CPU, memória e I/O), assim como contadores de mensagens processadas e número de processos em uma máquina. Adicionalmente ela utiliza uma quantidade destas características na casa das centenas, o que é computacionalmente caro.

Também é importante ressaltar que o propósito desta aplicação é de ser utilizada na execução de um teste de desempenho para análise das anomalias e não monitora o desempenho de uma aplicação em produção como a que esta pesquisa pretende fazer.

6.4 Measuring normality in HTTP traffic for anomaly-based intrusion detection (Juan M. Estevez-Tapiador, 2004)

Neste trabalho o foco está na detecção de intrusão através de anomalias nas requisições enviadas a uma aplicação web.

A detecção de anomalia é feita de uma forma bastante diferente utilizando cadeias de Markov para modelar as sequências regulares que a aplicação recebe, marcando uma sequência de requisições de baixa probabilidade de ocorrência como uma anomalia.

6.5 An Anomaly-Based Approach for Intrusion Detection in Web Traffic (Torrano-Giménez *et al.*, 2010)

Este outro trabalho propõe a criação de um firewall de requisições a aplicações web, porém ele requer um treinamento manual que descreve o que deve ser considerado anomalia ou não.

Apesar do título indicar alguma relação com o que está sendo estudado, ele difere bastante por não aprender automaticamente o que é anomalia ou não, e pela ação de bloqueio da requisição diante da anomalia percebida.

6.6 Performance Anomaly Detection in Microservice Architectures Under Continuous Change (Düllmann, 2017)

Esta dissertação de mestrado é a que mais se aproxima do que esta pesquisa propõe por abordar o contexto de micro-serviços e se basear na performance da aplicação.

Algumas diferenças podem ser apontadas na etapa de detecção de anomalias, a qual confia em um limiar pré-determinado para apontar quais requisições são anômalas e nenhuma tentativa de aprendizado do comportamento da aplicação é utilizada.

O autor menciona problemas interessantes que esta pesquisa deve se deparar, como atrasos iniciais devido a criação de estruturas de dados, cache ou conexões com bancos de dados na inicialização de uma instância da aplicação. Este cenário será verificado durante a pesquisa para avaliar se é possível classificá-lo ou suprimir alertas gerados por este motivo.

6.7 Comparativo

Na tabela 6.1 serão sumarizadas as diferenças entre os trabalhos relacionados e esta pesquisa:

Uma vez identificados alguns trabalhos relacionados, temos algumas ideias de alternativas para detecção de anomalias e também um cenário novo que deverá ser levado em conta durante os testes do projeto.

Durante o desenvolvimento do projeto e escrita da dissertação, novas pesquisas serão realizadas para avaliar o surgimento de outros trabalhos relacionados.

É importante ressaltar que esta pesquisa se torna relevante por não haver nenhuma outra com a mesma proposta, e também por ser de interesse da indústria de software, uma vez que tenho apoio da empresa HP Inc. na forma de tempo para inovação e acesso a dados.

Trabalho relacionado	Utiliza detecção de anomalia automática	Utiliza logs da aplicação	Dispensa instrumentação do código	Realiza classificação de problemas	Realiza o monitoramento de uma aplicação
6.1(Grover, 2018)	✓	✓	✓		✓
6.2(Rastogi *et al.*, 2018)	✓	✓	✓		✓
6.3(Peiris *et al.*, 2014)	✓	✓			
6.4(Juan M. Estevez-Tapiador, 2004)	✓	✓	✓		✓
6.5(Torrano-Giménez *et al.*, 2010)		✓	✓		✓
6.6(Düllmann, 2017)		✓	✓	✓	✓

Tabela 6.1: *Comparativo entre trabalhos relacionados e esta pesquisa.*

Capítulo 7

Cronograma

O experimento relatado no capítulo 5 indica que a pesquisa proposta é viável, apesar de ser utilizado um cenário bastante simples, onde só foram realizadas requisições de um único tipo e os registros não tiveram variação no tipo de resposta recebida.

Para a escrita da dissertação as seguintes atividades serão necessárias:

1. Coleta de registros de um serviço real em ambiente de produção, homologação ou qualificação de aplicações web da empresa HP Inc Brasil;
2. Realização de testes de detecção de anomalias com os dados do serviço real;
3. Adequação de algoritmos caso necessário;
4. Realização de testes de classificação das anomalias detectadas;
5. Implementação de uma aplicação para fazer o monitoramento de registros de um Application Load Balancer da AWS;
6. Avaliação da aplicação implementada e coleta de resultados.

As atividades elencadas serão executadas nos próximo 12 meses, iniciando em janeiro de 2020, conforme a tabela abaixo, com o objetivo de concluir a dissertação ao final de 2020.

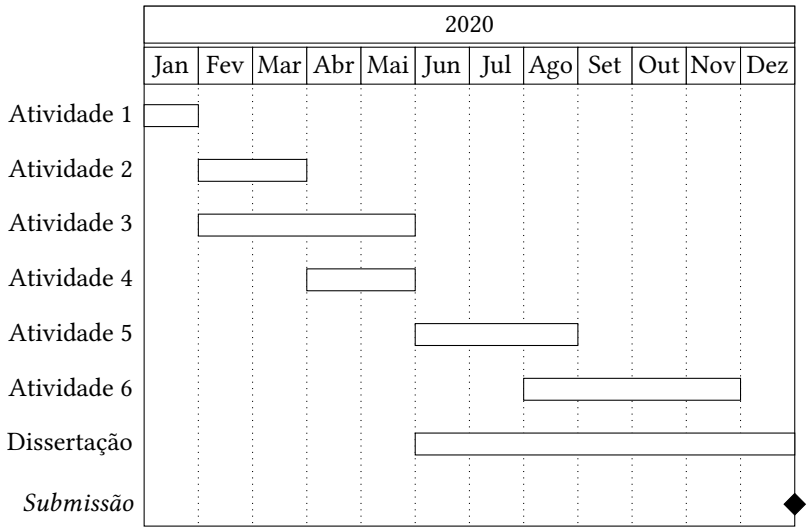

Figura 7.1: *Cronograma para conclusão da pesquisa.*

Referências

[Düllmann 2017] Thomas F. Düllmann. "Performance Anomaly Detection in Micro-service Architectures Under Continuous Change". Diss. de mestr. University of Stuttgart, 2017. URL: http://dx.doi.org/10.18419/opus-9066 (citado nas pgs. 27, 29).

[Erl 2009] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. The Prentice Hall Service Technology Series, 2009 (citado na pg. 3).

[Grover 2018] Aarish Grover. "Anomaly Detection for Application Log Data". Diss. de mestr. San Jose State University, 2018 (citado nas pgs. 25, 29).

[Juan M. Estevez-Tapiador 2004] Jesus E. Diaz-Verdejo Juan M. Estevez-Tapiador Pedro Garcia-Teodoro. "Measuring normality in http traffic for anomaly-based intrusion detection". Em: *Computer Networks* 45 (jun. de 2004), pgs. 175–193 (citado nas pgs. 27, 29).

[Ng 2014] Andrew Ng. *Machine Learning Coursera Course*. 2014. URL: http://coursera.org/learn/machine-learning (acesso em 01/01/2020) (citado na pg. 17).

[Peiris *et al.* 2014] M. Peiris *et al.* "Pad: performance anomaly detection in multi-server distributed systems". Em: *2014 IEEE 7th International Conference on Cloud Computing*. Jun. de 2014, pgs. 769–776. DOI: 10.1109/CLOUD.2014.107 (citado nas pgs. 26, 29).

[Rastogi *et al.* 2018] R. Rastogi, S. Nahata, Poonam Ghuli, D. Pratiba e G. Shobha. "Anomaly detection in log records". Em: *Indonesian Journal of Electrical Engineering and Computer Science* 10 (abr. de 2018), pgs. 343–347. DOI: 10.11591/ijeecs.v10.i1.pp343-347 (citado nas pgs. 26, 29).

[Torrano-Giménez *et al.* 2010] Camen Torrano-Giménez, Alejandro Pérez-Villegas e Gonzalo Álvarez Marañón. "An anomaly-based approach for intrusion detection in web traffic". Em: 2010 (citado nas pgs. 27, 29).

[Yaser S. Abu-Mostafa e Lin 2012] Malik Magdon-Ismail Yaser S. Abu-Mostafa e Hsuan-Tien Lin. *Learning from Data, A short course*. AMLBook, 2012 (citado na pg. 19).

[Zenon Chaczko e Mcdermid 2011] Shahrzad Aslanzadeh Zenon Chaczko V. Mahadevan e Christopher Mcdermid. “Availability and load balancing in cloud computing”. Em: *International Conference on Computer and Software Modeling IPCSIT* (2011) (citado na pg. 5).