UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de São José do Rio Preto

Diego de Lacerda Chiaradia

Acoplamentos fortes em ferramenta de análises estatísticas acoplada a bancos de dados diversos

São José do Rio Preto
2021

Diego de Lacerda Chiaradia

Acoplamentos fortes em ferramenta de análises estatísticas acoplada a bancos de dados diversos

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Conselho de Curso de Bacharelado em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de São José do Rio Preto.

Orientadora: Prof.ª Dr.ª Carina Alexandra Rondini

São José do Rio Preto
2021

C532a

Chiaradia, Diego de Lacerda
Acoplamentos fortes em ferramenta de análises estatísticas acoplada a bancos de dados diversos / Diego de Lacerda Chiaradia. -- São José do Rio Preto, 2021
58 p. : il., tabs.

Trabalho de conclusão de curso (Bacharelado - Ciência da Computação) - Universidade Estadual Paulista (Unesp), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto
Orientadora: Carina Alexandra Rondini

1. Acoplamentos. 2. Estatística. 3. Banco de dados. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca do Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

Diego de Lacerda Chiaradia

Acoplamentos fortes em ferramenta de análises estatísticas acoplada a bancos de dados diversos

Trabalho de Conclusão de Curso (TCC) apresentado como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação, junto ao Conselho de Curso de Bacharelado em Ciência da Computação, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista “Júlio de Mesquita Filho”, Câmpus de São José do Rio Preto.

Comissão Examinadora

Prof.ª Dr.ª Carina Alexandra Rondini
UNESP – Câmpus de São José do Rio Preto
Orientadora

Prof.ª Dr.ª Adriana Barbosa Santos
UNESP – Câmpus de São José do Rio Preto

Prof. Dr. Rodrigo Capobianco Guido
UNESP – Câmpus de São José do Rio Preto

São José do Rio Preto
2021

São as perguntas que não sabemos responder que mais nos ensinam. Elas nos ensinam a pensar. Se você dá uma resposta a um homem, tudo o que ele ganha é um fato qualquer. Mas, se você lhe der uma pergunta, ele procurará suas próprias respostas.

(Patrick Rothfuss)

AGRADECIMENTOS

Agradeço a minha família, que me ajudou a manter a cabeça clara, durante estes anos de pandemia, disponibilizando-me apoio financeiro, quando necessário.

A minha orientadora, Carina Rondini, que teve paciência comigo e com meus horários e tornou a realização do trabalho possível, dando-me todo o apoio, por meio de suas correções.

A minha amiga Daniele Pecorari, a qual me auxiliou de um jeito admirável, durante todo o processo de desenvolvimento do trabalho e tradução do resumo.

Aos meus amigos Felipe Pimenta e Guilherme Gervaes, também formandos, que me motivaram e me deram forças para a continuidade do trabalho.

Agradeço também aos meus amigos Cassio Fogarin, Felipe Ferri, João Ribeiro e Mario Kioshi, os quais me auxiliaram em diversos contratempos, durante o ano, e aos amigos Gustavo Lima, Yuri Seregati e Nícolas Lúcio, por toda a ajuda em corrigir e revisar o trabalho.

RESUMO

Ao buscar os pontos fortes intrínsecos às áreas de análise estatística e de armazenamento de dados, deu-se alicerce ao acoplamento de *softwares* estatísticos aos *softwares* de bancos de dados. Acoplamento de *softwares* significa a medida de dependência entre *softwares* integrados. De forma geral, a meta de acoplamento é ter acoplamentos frouxos, para que não exista nenhuma ou haja pouca dependência entre os *softwares*, de modo que os sistemas possam ter uma reutilização entre tipos diferentes de *softwares*, entretanto, um acoplamento forte tem desempenho superior em relação ao acoplamento frouxo. Assim, este trabalho propôs a análise de um *software* estatístico acoplado a bancos de dados, com níveis de acoplamentos diversos para validar o ganho de desempenho do sistema fortemente acoplado em relação a configurações com grau de acoplamento menor. Para tanto, o sistema foi desenvolvido e testado com diversos graus de acoplamento de componentes, empregando-se uma arquitetura de requisições HTTP para realizar a conexão entre os componentes quando desacoplados. Os resultados obtidos do teste indicaram que o sistema teve uma grande melhoria em seu desempenho, ao acoplar fortemente os componentes, entretanto, ao remover uma média de tempo utilizada pelas requisições HTTP, esse ganho de desempenho se tornou baixo para a quantidade de dados validados.

Palavras-chave: Acoplamento. Estatística. Banco de dados.

ABSTRACT

In seeking the intrinsic strengths of the statistical analysis and data storage areas, the foundation of the coupling of statistical software to database software was built. Software coupling stands for the measurement of dependency between integrated software. In general, the goal of coupling is to have loose couplings, so that there is none or little dependency between the software, so that systems can have a reuse between different types of software. However, a strong coupling has superior performance in relation to a loose coupling. Thus, this work proposes the analysis of a statistical analysis software coupled to database, with different coupling levels to validate the performance gain of the strongly coupled system in relation to configurations with a lower degree of coupling. Therefore, the system was developed and tested with different degrees of component coupling, using an HTTP request architecture to perform the connection between components when decoupled. The results obtained from the test show that the system had a great improvement in its performance tightly coupling the components, however, when removing an average time used by HTTP requests, this performance gain became low for the amount of validated data.

Keywords: Coupling. Statistic. Database.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CSV	*Comma-Separated Values*
HTTP	*Hypertext Transfer Protocol*
IBM	*International Business Machines*
JAR	*Java ARchive*
JSON	*JavaScript Object Notation*
PUBLIC	*PrUning and BuiLding Integrated in Classification*
SGDB	Sistema Gerenciador de Banco de Dados
SPSS	*Statistical Package for the Social Sciences*
SQL	*Structured Query Language*

SUMÁRIO

Capítulo 1

Introdução

A estatística pode ser vista como uma coleção de métodos para planejar experimentos e obter dados, para organizá-los, resumi-los, analisá-los, interpretá-los e, por fim, deles algo concluir (TRIOLA, 2005). O papel da estatística se torna fundamental, a partir do momento em que a estimativa do fluxo de dados na rede mundial de computadores seria aproximadamente de 175 *Zetabytes*, ou 1.099.511.627.776 *Gigabytes* (REINSEL; GANTZ; RYDNING, 2018). Ainda assim, de forma geral, em média, de 60% a 73% desses dados não são utilizados para análises estatísticas (GUALTIERI, 2016).

Considerada como um desafio, a forma de apresentação dos resultados é um ponto principal de uma análise estatística. Uma representação deficitária dos dados pode comprometer o entendimento do leitor, assim como suas conclusões/decisões finais (SEAKOMO, 2014).

Softwares estatísticos dispõem de várias funcionalidades para análise e modelagem de dados, porém, podem apenas lidar com uma quantidade limitada de dados, pois operam somente na memória principal de um sistema; além disso, a quantidade de dados que podem operar simultaneamente é limitada pela disponibilidade de espaço livre na memória principal. Em contrapartida, Sistemas de Gerenciamento de Banco de Dados (SGBD) podem guardar uma grande quantidade de dados, entretanto, dispõem de recursos estatísticos limitados, como a ausência de funções estatísticas (SEAKOMO, 2014). Por exemplo, o *software* estatístico *Statistical Package for the Social Sciences* (SPSS) tem um limite de 2 bilhões de casos, num sistema de 32 bits, e restringe os dados para uma execução estatística baseada na quantidade de memória principal disponível no seu sistema, num sistema de 64 *bits* (IBM, 2021), enquanto o SGDB *MySQL* tem sua execução apoiada na linguagem *Structured Query Language* (SQL), uma linguagem de pesquisa declarativa com poucas funções estatísticas (MELTON, 1996).

Ao buscar os pontos fortes intrínsecos às áreas de análise estatística e de armazenamento de dados, deu-se alicerce ao acoplamento de *softwares* estatísticos aos *softwares* de bancos de dados (SEAKOMO, 2014).

Acoplamento de *softwares* significa a medida de dependência entre *softwares* integrados. A métrica de acoplamento é utilizada como referência para avaliar a integração entre dois ou mais *softwares* interligados (RUH; MAGINNIS; BROWN, 2002).

Um dos mais importantes resultados esperados para a integração de *softwares* é a redução do nível de acoplamento entre eles. O acoplamento é medido visualizando-se o grau de interdependência entre os *softwares* e o impacto que eles têm entre si, definindo-se assim o grau de integração (RUH; MAGINNIS; BROWN, 2002).

Os graus de integração são divididos entre *Loose Coupling*, cuja integração é dependente de algumas interfaces discretas de acoplamento, e *Strong Coupling,* no qual usualmente as dependências de acoplamento são altas (RUH; MAGINNIS; BROWN, 2002).

Um par de conceitos referenciados é o de integração *white-box* e integração *black-box*. A integração *white-box* expõe os componentes internos da aplicação para o integrador, enquanto a integração *black-box* os esconde, embutindo-os no sistema (RUH; MAGINNIS; BROWN, 2002).

De forma geral, a meta de acoplamento é ter acoplamentos frouxos, para que não exista nenhuma ou haja pouca dependência entre os *softwares*, de modo que os sistemas possam ter uma reutilização entre tipos diferentes de *softwares*, resultando em sistemas *plug-and-play* (RUH; MAGINNIS; BROWN, 2002).

Ao analisar métodos de acoplamentos, verifica-se que acoplamentos *frouxos* têm vantagens de portabilidade, de tolerância a falhas e de sobrecarga de comunicação em relação a acoplamentos *fortes*, os quais provêm de sistemas fortemente intrínsecos entre si e com funcionalidades voltadas para funções específicas predeterminadas. Entretanto, essa forte integração tem desempenho superior em relação à integração frouxa (BEZERRA; MATTOSO; XEXÉO, 2000). Em sistemas de acoplamentos *fortes*, a obtenção de informação pode ser um problema, visto que nenhum algoritmo de *data-mining* é ideal para todos os tipos de dados,

tornando inviável a implementação interna ao sistema de banco de dados (ONODA; EBECKEN, 2001).

1.1 OBJETIVO

Como objetivo geral, buscou-se, neste trabalho, a avaliação de desempenho de acoplamentos fortes em sistemas de análises estatísticas acoplados a bancos de dados.

Por outro lado, como objetivo específico, visou-se ao desenvolvimento de uma ferramenta de análises estatísticas frouxamente acoplada em todas as conexões de componentes. A partir da ferramenta frouxamente acoplada, aumentou-se gradativamente o grau de acoplamento do sistema, e este foi avaliado, a fim de gerar melhorias no desenvolvimento de sistemas desse tipo e verificar onde sua utilização pode ser benéfica.

1.2 JUSTIFICATIVA

Conforme Bezerra, Mattoso e Xexéo (2000), apesar de acoplamentos frouxos terem vantagens de portabilidade e de tolerância a falhas, acoplamentos fortes têm desempenho superior. Além disso, segundo Ruh, Maginnis e Brown (2002), acoplamentos fortes podem aumentar o desempenho de um sistema, de forma geral. Assim, buscou-se, neste trabalho, unir o que foi apresentado por Seakomo (2014), Ruh, Maginnis e Brown (2002) e Bezerra, Mattoso e Xexéo (2000), em uma análise dos graus de acoplamento em sistemas de análise estatística, a fim de definir se acoplamentos fortes, em sistema de análise estatística, resultam em uma performance superior em sistemas com escopo bem definido e com pouca necessidade de alteração.

Seguindo essa concepção descrita acima, a principal justificativa neste trabalho se refere, portanto, à necessidade de análise de acoplamentos fortes em sistemas de análise estatística, de sorte a definir se essa forma de acoplamento resulta em uma performance superior, em sistemas com escopo bem definido e com pouca necessidade de alteração.

1.3 METODOLOGIA

O trabalho está dividido em duas partes: teórica e prática. O desenvolvimento teórico está pautado na revisão narrativa, concebida como um tipo de descrição de artigos de forma ampla, permitindo ao leitor adquirir conhecimentos sobre um determinado assunto, em um curto período de tempo (ROTHER, 2007).

A revisão narrativa é considerada a revisão tradicional ou exploratória, sem definição de critérios explícitos e seleção arbitrária das fontes de dados utilizados (FERENHOF; FERNANDES, 2016).

A parte empírica do trabalho está ilustrada na Figura 1.

O primeiro passo é a seleção dos tipos de bancos que são usados para o desenvolvimento deste trabalho.

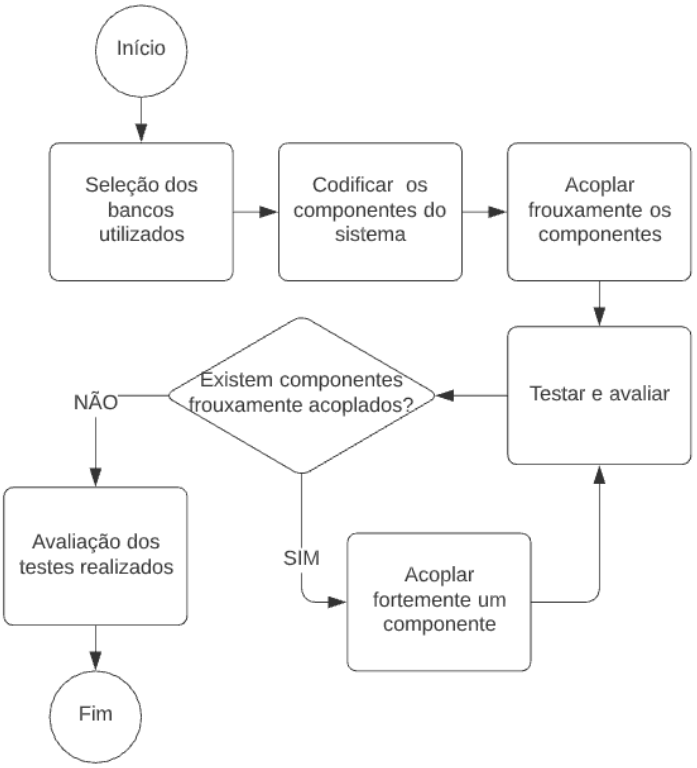
Após feita essa seleção, os componentes de entrada dos bancos para a leitura dos mesmos, assim como os componentes de controle e os de análise estatística, são codificados.

Ao final da codificação dos componentes, estes são incluídos em um sistema em configuração de acoplamento frouxo.

Concluída a implementação, são empregadas métricas de medição de eficiência, para medição de performance, e um componente, antes frouxamente acoplado, é fortemente acoplado.

A performance é novamente medida e, caso existam componentes frouxamente acoplados ao sistema, o processo é refeito, até que o sistema esteja integrado apenas a acoplamentos fortes.

Figura 1: Fluxograma do desenvolvimento empírico da pesquisa.



Fonte: Autoria própria.

Por fim, os dados de cada configuração de componentes são comparados entre si, a fim de que sejam realizadas conclusões e avaliações.

Dessa forma, o texto está organizado da seguinte maneira:

- Capítulo 2 - Trabalhos correlatos e estado da arte
- Capítulo 3 - Desenvolvimento
- Capítulo 4 - Avaliação de Resultados
- Capítulo 5 - Conclusões finais

Capítulo 2

Trabalhos Correlatos e Estado da Arte

No segundo capítulo deste trabalho, relata-se a fundamentação teórica. Na seção 2.1, aborda-se o estado da arte, para entendimento de conceitos relacionados a este texto. Em seguida, na seção 2.2, são focalizados trabalhos correlatos a este.

2.1 ESTADO DA ARTE

De modo a desenvolver este trabalho, é necessário entender conceitos ligados a acoplamentos de *software*, para compreensão das características avaliativas propostas.

Nos próximos tópicos, é apresentada uma revisão narrativa, dividida da seguinte forma: seção 2.1.1: Acoplamentos de *softwares*; seção 2.1.2: Tipos gerais de acoplamentos frouxo e forte em que também aparecem especificações de acoplamentos; seção 2.1.3: Métricas para medidas de acoplamentos; seção 2.1.4: Análises estatísticas; por fim, na seção 2.1.5, são introduzidos a linguagem *Java* e o *Spring Framework*.

2.1.1 ACOPLAMENTO

Acoplamento é uma métrica de *software* que significa o quão dois *softwares* são conectados e independentes entre si. O conceito de acoplamento foi introduzido pela IBM, em artigo de 1974 (FAGARASAN, 2016).

O fato de um componente qualquer precisar ser inspecionado, para entender outro, condiz perante um grau de conexão entre componentes. Por exemplo, se dois componentes A e B quaisquer são definidos, quanto mais conhecimento de B for necessário para entender A, mais A é fortemente conectado a B. Acoplamento pode ser definido como uma medida de quanto um componente interligado conhece de outro (FAGARASAN, 2016).

Em contraste com o grau de acoplamento, existe a métrica de coesão. A coesão é medida de maneira inversamente proporcional ao acoplamento: se um sistema tem um acoplamento frouxo, ele tem uma coesão forte (FAGARASAN, 2016). A ideia de coesão foi originalmente introduzida juntamente com a ideia de acoplamento, em 1974. A coesão foi redefinida para o conceito de orientação a objetos, em 1992, passando a significar o grau no qual os métodos e atributos de uma certa classe pertencem entre si (PEREPLETCHIKOV; RYAN; FRAMPTON, 2007).

2.1.2 TIPOS DE ACOPLAMENTO

Métodos de acoplamentos são descritos em dois tipos: frouxos (*low*/*loose*/*weak)* e fortes (*high*/*tight*/*strong*). Pode-se identificar acoplamentos fortes entre dois componentes, quando a mudança de um requer também a modificação de outro, para que o sistema não perca sua funcionalidade e mantenha a resiliência, evitando que o sistema pare de funcionar, quando uma modificação ocorre. Em contrapartida, acoplamentos frouxos podem ser definidos como uma ligação entre dois componentes que são independentes entre si. Caso um componente seja modificado em uma ligação frouxa, o componente ao qual ele está interligado terá de ter resiliência e segurança, para que não precise ser modificado, ou seja, caso uma modificação ocorra, outro componente que se interligava continuará funcionando (DAS, 2020).

Existem seis categorias de acoplamentos, as quais estão divididas entre acoplamentos frouxos e fortes (KAR, 2021):

- *Data Coupling*: Ligação entre dois componentes em que a comunicação ocorre apenas por transferência de dados. É definida como o acoplamento mais frouxo.
- *Stamp Coupling*: Ligação de transferência de dados na qual uma estrutura de dados completa é transferida entre dois componentes. É semelhante ao *Data Coupling* e se dá quando existe a necessidade de performance e eficiência

em uma transferência de dados; é comumente definida como um acoplamento frouxo.

- *Control Coupling*: Se dois componentes se comunicam para troca de informações de controle, a ligação é caracterizada como *Control Coupling.* Pode ser um acoplamento frouxo, caso os parâmetros possam ser reusados em conexão com outros sistemas, ou um acoplamento forte, caso os parâmetros sejam específicos.
- *External Coupling*: Sinaliza uma ligação entre um componente interno de um sistema e um componente externo a ele. Definida como uma ligação frouxa, caso aceite diversos componentes externos para realizar a ligação, ou como um acoplamento forte, se o sistema externo for específico para a função.
- *Common Coupling*: Os componentes em um *Common Coupling* têm estrutura de dados compartilhados, na qual uma modificação no componente que transfere os dados faz com que o funcionamento do sistema seja quebrado e torne necessária a modificação do componente que recebe os dados. É concebida como um acoplamento forte.
- *Content Coupling*: *Content Coupling* ocorre quando um componente tem o poder de modificar os dados e o funcionamento de um outro componente. É caracterizado como o tipo mais forte de acoplamento.

Em *design* de *softwares*, é recomendado que os acoplamentos tenham um grau frouxo, para que o sistema seja mais resiliente e seguro contra falhas causadas por modificações em componentes (KAR, 2021).

2.1.3 MÉTRICAS DE MEDIDA DE ACOPLAMENTO

Métricas de acoplamento de *software* determinam o grau de complexidade da arquitetura requerida; baseiam-se em dependências entre componentes e auxiliam desenvolvedores no desenvolvimento de *softwares* resilientes (KANJILAL, 2020).

A métrica de Fenton e Melton (FENTON; MELTON, 1990), Equação (1), é usada para medir o grau de acoplamento de uma arquitetura.

$$C(a,b) = i + n / (n + 1) \quad (1)$$

Nessa equação, “a” e “b” representam dois componentes, “n” representa o número de dependências entre os componentes e “i” representa o valor de nível de acoplamento que existe entre eles e varia entre 0, sem acoplamento, e 5, um acoplamento forte (ALGHAMDI, 2007).

Efferent Coupling (Ce) e *Afferent Coupling* (Ca) são métricas utilizadas para medir a dependência de um componente, em específico, assim como a métrica de *Instability* (MARTIN, 2002). Pode-se exemplificar, considerando-se um componente “A” qualquer que seja, dependente de outro componente, caso em que se define o valor de Ce de “A” como 1. Se outros quatro componentes forem dependentes desse componente “A”, o valor de Ca de “A” é 4 (KANJILAL, 2020).

Instability (MARTIN, 2002) é uma métrica, Equação (2), a qual mede a susceptibilidade de um componente deixar de funcionar e afetar o resto do sistema, durante sua execução. Seu valor varia entre 0 e 1, em que 0 indica um componente estável e 1 indica um componente suscetível a parar o funcionamento do sistema (KANJILAL, 2020).

$$I = Ce / (Ce + Ca) \quad (2)$$

Utilizando o exemplo anterior, pode-se calcular o grau de *Instability* do componente “A”, conforme apresentado na Equação (3):

$$I = 1/ (1 + 4) \quad (3)$$

Assim, o grau de *Instability* do componente “A” tem o valor de 0,20, indicando um componente pouco suscetível a quebrar.

Abstractness (MARTIN, 2002), Equação (4), é uma métrica geral que mede a abstração da arquitetura de um sistema. Seu valor também varia entre 0 e 1, sendo

que 0 indica um sistema completamente não abstrato e 1 indica um sistema completamente abstrato (KANJILAL, 2020).

$$A = Ta / (Ta + Tc) \quad (4)$$

Nessa equação, “Ta” significa o número total de componentes abstratos em um sistema e “Tc” representa o número total de componentes não abstratos, ou concretos (KANJILAL, 2020).

2.1.4 ANÁLISE ESTATÍSTICA DESCRITIVA

A análise estatística descritiva é a fase inicial para o estudo de casos coletados. Ela tem a função de organizar, resumir e descrever aspectos importantes de um conjunto de dados ou compará-los com outros conjuntos (REIS; REIS, 2002).

A descrição e a interpretação dos dados têm um papel importante na identificação de anomalias, como registros incorretos, as quais não seguem a tendência geral de um conjunto de dados (REIS; REIS, 2002).

As variáveis de uma análise estatística são divididas em quantitativas e qualitativas. Variáveis quantitativas são valores numéricos que podem ser medidos em uma escala quantitativa, sendo divididas em contínuas e discretas. Variáveis quantitativas contínuas são caracterizadas por serem mensuráveis em uma escala real, enquanto variáveis quantitativas discretas constituem resultado de alguma contagem (REIS; REIS, 2002).

Variáveis qualitativas são características que não podem ser medidas. Representam valores atinentes a uma classificação de indivíduos, sendo estas divididas em nominais e ordinais. Variáveis qualitativas ordinais são definidas por uma categorização de ordem entre seus membros, enquanto variáveis qualitativas nominais se organizam em categorias sem ordenação (REIS; REIS, 2002).

As medidas descritivas podem ser divididas entre medidas de tendência central e medidas de dispersão. As principais medidas de tendência central são a média, a mediana e a moda, enquanto as principais medidas de dispersão são as

medidas de variância, amplitude total e desvio padrão (RODRIGUES; LIMA; BARBOSA, 2017).

As medidas de tendência central são caracterizadas por valores comuns para um conjunto e representam todos os valores internos dele:

- A média é uma medida que incorpora o valor de todo membro do grupo e é definida pela soma deles dividida pelo número total de membros.
- A mediana é a medida que define o valor central de todos os valores do grupo, em ordem crescente.
- A medida de moda permite entender o valor que mais se repete em um conjunto.

As medidas de dispersão indicam como os dados variam em torno dos valores típicos:

- As medidas de variância e desvio-padrão estimam o grau em que a variável se desvia da média; o desvio-padrão é a raiz quadrada da variância.
- A medida de amplitude total é definida pela distância total entre os valores mais alto e mais baixo, do conjunto de dados. Empregada juntamente a mediana, faz a separação dos valores em quartis: o primeiro quartil representa os primeiros 25% dos valores e o terceiro quartil define os 75% valores acima dessa posição.

A utilização correta da estatística descritiva permite ao analista ter confiança nos resultados obtidos em sua pesquisa, enquanto a má utilização acarreta descrédito da mesma (RODRIGUES; LIMA; BARBOSA, 2017).

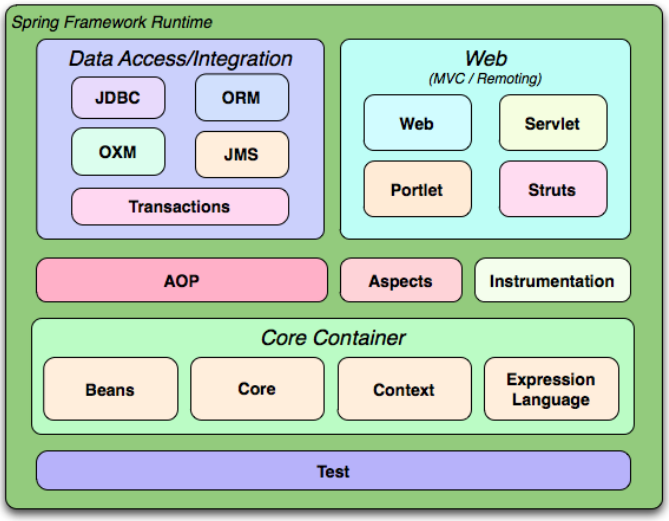
2.1.5 - JAVA E *SPRING FRAMEWORK*

A linguagem de programação *Java* é uma linguagem orientada a objetos baseada na utilização de classes. É uma linguagem de propósito geral, podendo ser usada para sistemas de diversas funções (GOSLING *et al*., 2000).

Ela foi desenvolvida, em 1991, pela *Sun Microsystems*, com o objetivo de ser simples e eficiente, tendo o propósito inicial de ser adotada em sistemas de eletrodomésticos. Porém, com o avanço da internet, o código foi adaptado para emprego em microcomputadores ligados à rede. Assim, a linguagem foi popularizada e utilizada amplamente (GOSLING *et al*., 2000).

Spring é um *framework* para desenvolvimento de aplicações em *Java*, considerado o mais empregado atualmente. Foi lançado em 2002 e se baseia no ideal de injeção de dependências, o qual elimina a necessidade de instanciar objetos, removendo a necessidade de declarar algo já existente no sistema novamente, apenas declarando o uso do módulo já utilizado (PICHETTI, 2015). O *framework* possui aproximadamente vinte módulos, que podem ser facilmente integrados a qualquer tipo de aplicação, com a inserção de suas dependências em seu arquivo de configuração, instalando bibliotecas externas facilmente e possibilitando, dessa forma, o desenvolvimento de sistemas robustos, como um sistema de requisições de várias camadas, de forma simples (JOVANOVIC *et al*., 2017). A arquitetura do *framework* com seus componentes é focalizada na Figura 2.

Figura 2: Arquitetura do *Spring Framework*



Fonte: (WEBB, 2021)

A camada *Core* contém as funcionalidades fundamentais para a execução do *framework*, como a injeção de dependências via arquivo de configuração. Já da camada *Data Access/Integration* provém a abstração de dados necessária para identificar o tipo e a propriedade dos dados com o sistema, assim permitindo a fácil integração com diversos tipos de *software* com a linguagem *Java*. A camada *Web* contém funcionalidades voltadas para o desenvolvimento de *Application Programming Interfaces* (APIs) e servidores, facilitando a criação de microsserviços e requisições HTTP. As camadas AOP, *Aspects* e *Instrumentation,* fornecem funcionalidades de interceptação de métodos e desacoplamentos, a fim de utilizar módulos existentes em diferentes partes do sistema. Por fim, a camada *Test* fornece funcionalidades voltadas para testes unitários e de integração (WEBB, 2021).

2.2 TRABALHOS CORRELATOS

Para melhor entender o assunto abordado neste trabalho, foram analisadas publicações que discorrem a respeito de acoplamentos de *softwares* e uso de dados. Para tanto, empreendeu-se busca nas bases de dados *Google Schoolar e Research Gate,* com descritores de “acoplamentos” e “estatística acoplada a bancos de dados”, encontrando-se, assim os dois trabalhos apresentados a seguir.

2.2.1 Implementação de um Algoritmo de Árvore de Decisão Acoplada a um SGBD Relacional

Onoda e Ebecken (2001) apresentam a implementação de um algoritmo de árvore de decisão acoplada a um SGBD relacional. A implementação foi feita em PUBLIC, um sistema de classificação de dados, o qual, diferentemente de outros sistemas de classificação, realiza a geração da árvore em largura. O sistema é frouxamente acoplado com microcomputadores atuando com o algoritmo de árvore de decisão, em *Java*, e faz a análise de desempenho do sistema, com três bancos de dados diferentes, verificando o tempo de execução, a quantidade de nós que a árvore gerou e a porcentagem de erros que o algoritmo teve, durante seu andamento. Ao final do processamento, também se compara a velocidade do algoritmo com uma implementação da mesma função, em SQL.

A Tabela 1 representa os resultados obtidos pela implementação do sistema PUBLIC em *Java*.

Tabela 1: Resultados obtidos pela implementação do algoritmo PUBLIC

Base de dados	**Tamanho da árvore (nós)**	**Tempo de execução (s) em *Java***	**Erro (%)**
Adult	105	717	18
Meteorologia	183	379	36
Seguro	267	39 006	30

Fonte: (ONODA; EBECKEN, 2001, p. 9).

A Tabela 2 detalha a comparação entre o tempo de execução obtido pela implementação do sistema PUBLIC, em *Java*, e por uma implementação da mesma função, em SQL.

Tabela 2: Comparação de tempo de execução da implementação do algoritmo PUBLIC e implementação SQL de mesma função

Base de dados	**Tempo de execução (s)** ***Java***	**Tempo de execução (s)** **PL/SQL**
Adult	717	797
Meteorologia	379	398
Seguro	39 006	38 413

Fonte: (ONODA; EBECKEN, 2001, p. 10).

Os autores concluíram que microcomputadores frouxamente acoplados com um SGBD têm desempenho semelhante ao de uma implementação SQL comum para a mesma função, cobrindo certos problemas, como escalabilidade e portabilidade, de modo a abrir portas para a área de *data mining* incremental (ONODA; EBECKEN, 2001).

Os autores também verificaram que o sistema de acoplamento frouxo pode ter um desempenho melhor do que o de uma função interna de um componente, dependendo do tipo de sistema adotado.

2.2.2 Análise de um sistema acoplado a um componente de entrada utilizando duas configurações diferentes

Bora e Bezboruah (2020) apontam que o emprego de acoplamentos frouxos cria *frameworks* de soluções de integrações com confiabilidade alta. *Software as a Service* (SaaS) é um exemplo de *framework*. Nesse *framework*, é promovida a avaliação de uma plataforma SaaS, visando a medir sua confiabilidade, mediante a análise de requisições HTTP em um banco de dados de 15000 registros de dados médicos, com seus tipos já mapeados no sistema. A plataforma foi separada em quatro camadas: *consumer*, *parent*, *service* e *database*. A camada *consumer* representa a interface gráfica que o usuário do sistema recebe; a *parent* faz a mediação entre as camadas *consumer* e *service*, a qual, por sua vez, realiza toda a

execução das regras de negócio. Por fim, a camada *database* é responsável por todas as transações envolvendo o banco de dados da plataforma.

Essas quatro camadas – independentes uma da outra – trabalham juntas, a fim de receber diversos acessos de múltiplos usuários simulados, com números variando entre 50, 100, 500, 800, 900, 1.000 e 1.500. O ambiente de teste contém duas configurações separadas para avaliação: uma, com separação por *cluster*, com dois *clusters* acoplados frouxamente com a plataforma, para balanço de carga de trabalho para o envio de requisições, controlando e balanceando a entrada do usuário no sistema e os envios de requisições; e outra, com apenas uma interface de envio direto, com a qual o usuário simulado envia diretamente suas requisições, sem controle de entrada ou balanceamento de carga de envio.

Os testes são realizados verificando-se as requisições HTTP totais feitas pelos usuários simulados e observando-se as requisições falhas, durante o período de 300 segundos após todos os usuários simulados adentrarem o sistema. A Tabela 2.3 representa os resultados dos testes. THR (*Total HTTP Request)* significa a quantidade total de requisições apresentadas pelos usuários simulados, FHR (*Failed HTTP Request*) significa a quantidade de requisições falhas e FR (*Failure Rate*) significa a taxa de requisições falhas pelas requisições totais executadas em porcentagem: FHR/THR = FR. LBC se refere ao ambiente com configuração de *clusters* (*Load Balancing Clustered)*, enquanto Non LBC concerne ao sistema sem *clusters*.

Tabela 3: Resultados dos testes realizados

Caso de Teste	Nível de estresse do sistema	Plataforma com balanceamento de carga			Plataforma sem balanceamento de carga		
		THR	FHR	FR(%)	THR	FHR	FR(%)
Recuperação de dados	50	366	0	0	284	0	0
	100	978	0	0	656	0	0
	500	14 863	0	0	12 598	0	0
	800	35 924	0	0	19 546	0	0
	900	48 525	15 251	31	47 659	0	0
	1000	55 082	17 877	32	53 275	0	0
	1500	121 226	78 650	65	62 975	0	0
	1800	211 675	170 862	81	112 616	42 136	34

Fonte: (BORA; BEZBORUAH, 2020, p. 3).

Pode-se observar que uma plataforma SaaS, em um sistema com *clusters*, tem uma confiabilidade superior à do mesmo sistema com uma configuração sem a utilização dos *clusters*: apenas com 1 800 usuários simulados, o sistema teve falhas nas requisições, com uma taxa de 34% de falha, enquanto o sistema sem clusterização teve falhas apontadas com 900 usuários simulados e taxa inicial de 31% de falhas, as quais cresceram até 81% com o emprego de 1 800 usuários.

Entretanto, o sistema sem clusterização teve um desempenho superior no número de requisições, em todos os pontos do sistema, ocasião em que as requisições eram feitas de forma fortemente acoplada, logo, a configuração com clusterização teve melhor resiliência e resistência a falhas, enquanto o sistema sem clusterização teve melhor desempenho e quantidade de usuários, dentro do sistema, mas sem controle de falhas (BORA; BEZBORUAH, 2020).

Isso significa, comparativamente, que um acoplamento forte tem desempenho elevado em situações nas quais a possibilidade de falha e a resiliência a erros são baixas, porém, numa situação em que a resiliência a erros é um fator crucial, é concebível que um sistema frouxamente acoplado apresente melhor desempenho.

Capítulo 3

Desenvolvimento

No terceiro capítulo deste texto, descreve-se a metodologia de desenvolvimento e avaliação do trabalho. Na seção 3.1, discute-se a forma adotada para o desenvolvimento do sistema descrito nesta pesquisa. Na subsequente seção 3.2, são descritos os métodos de avaliação utilizados para comprovar os resultados e apresentam-se as conclusões aferidas durante os testes. Na seção 3.3, os resultados do desenvolvimento esperado são explicados. Por fim, na seção 3.4, é abordado o desenvolvimento funcional do trabalho.

3.1 METODOLOGIA

A metodologia para o desenvolvimento foi iniciada pela escolha de um banco de dados existente em três formas distintas, com uma quantidade mínima de 1.000 registros e pelo menos três atributos numéricos. O registro desse banco foi modelado para orientação a objetos, de sorte a ser introduzido no sistema, utilizando a linguagem *Java*, adotada no *Spring Framework*. A seguir, codificou-se um componente de controle, no qual se iniciou o fluxo do sistema. Após, foram codificados os componentes de aceitação dos sistemas de bancos de dados que se introduzem no sistema. Empregaram-se JSON, CSV e SQL, para realizar a leitura desses bancos.

Depois da codificação dos sistemas de bancos de dados para leitura dos bancos, foram codificados os componentes de análises estatísticas que usam os dados provenientes dos bancos de dados, para análise e geração de resultados. Cada sistema tem seu retorno modelado em uma classe e, na sequência, foram introduzidos os componentes no sistema de forma frouxamente acoplada, com utilização de uma arquitetura que se vale de requisições HTTP, dentro do componente de controle. Cada componente funciona independente de outro. Na

sequência, o grau de acoplamento do sistema foi aumentado, inserindo-se funções de componentes, antes desacoplados, dentro do componente de controle, de sorte a eliminar a independência do componente. Assim, o sistema foi novamente avaliado, repetindo-se essa rotina até que ele reunisse somente acoplamentos fortes e todos os componentes tivessem sua funcionalidade inserida dentro da função do componente de controle. Ao final desse processo, os resultados de desempenho obtidos em cada configuração foram comparados.

O desenvolvimento completo foi implementado no computador pessoal do aluno, com um sistema operacional *Windows 10* de 64 *bits,* a partir de um sistema de armazenamento em disco, de oito *gigabytes* de memória RAM, e de um processador de 4 núcleos e 8 *threads* de 1,8 *gigahertz* de velocidade-base, podendo-se acelerar para 3,9 *gigahertz*.

3.2 MÉTODO DE AVALIAÇÃO

As avaliações foram implementadas testando-se o funcionamento do sistema com cada um dos tipos de bancos de dados inseridos. Ao final, foi promovida a metrificação de desempenho das diferentes configurações.

Durante os testes, mediu-se a velocidade das ações no sistema e uso dos recursos do sistema, para a execução da análise, avaliando-se cada uma das configurações.

Tais testes foram feitos, a fim de se constatar similaridades e discrepâncias nesses quesitos e gerar conclusões a respeito deles mesmos.

3.3 DESENVOLVIMENTO FUNCIONAL DO TRABALHO

Na seção 3.3, adentramos especificamente no desenvolvimento funcional da investigação, explicando ao leitor o trabalho desenvolvido para validação da teoria.

3.3.1 ESPECIFICAÇÕES DO DESENVOLVIMENTO DOS COMPONENTES

Para o desenvolvimento na linguagem *Java* e *Framework Spring*, devemos atentar a configurações necessárias que são comuns para o emprego do *framework*, como as configurações de *application.properties* e dependências adotadas.

A linguagem de programação *Java* foi utilizada, por ser uma linguagem de controle com diversas funções de adequação para uso de bancos de dados de diversos tipos e permitir várias formas de conexão entre os componentes de um sistema.

A fim de facilitar o desenvolvimento, podemos usar a ferramenta *Spring Initializr*, uma interface *web* de fácil manuseio, feita para facilitar a geração de projetos que empregam *Spring*. A ferramenta disponibiliza uma escolha simples de dependências e configurações da aplicação, ou componente, e retorna um projeto do *Spring* previamente configurado em quesito de versão e dependências.

Comumente, o projeto é concebido como um projeto *Maven*, a versão do *Spring* para 2.5.6, a definição do tipo de pacotes para JAR, a linguagem como *Java* e sua versão aplicada para 11. A identificação de pacote não tem uma pré-definição necessária e pode ser caracterizada a gosto do usuário. Como dependências, definimos para todos a dependência *Spring Starter Web.*

Ao abrir o componente, é visto em seus recursos um arquivo de texto denominado *application.properties*, o qual define as configurações adotadas para a aplicação *Spring*. A princípio, é necessário definir uma porta diferente para cada componente, por intermédio da configuração *server.port*.

Cada componente é descrito por dois pacotes comuns a todos: *controller* e *model*. Na programação orientada a objetos, pacotes são utilizados para agrupar códigos de funções semelhantes. O *controller* remete a porções do código as quais executam operações sobre os dados, enquanto o *model* é empregado para definir os dados em uma classe e uso em escopo orientado a objetos.

3.3.1.1 SELEÇÃO DE BANCO DE DADOS

Para melhor validação do projeto, o conjunto de dados escolhido foi o *Appalachian Play Fairway Analysis Seismic Hazards Supporting Data*, uma base de dados geográficos a respeito da Bacia do Apalache, situada nos Estados Unidos da América. Esse banco contém dados de monitoração de abalos sísmicos, naturais ou induzidos. Ele possui 1 647 entradas com 12 colunas, as quais estão divididas entre quatro valores numéricos e oito valores nominais. O banco foi obtido via disponibilização pública do sistema *data.world* e foi convertido em JSON, pois o mesmo não está disponível nesse formato, diferentemente dos dados em CSV e SQL.

3.3.1.2 ESPECIFICAÇÃO DA ESTRUTURA *CONTROLLER*

A estrutura dos pacotes *controller* dos componentes é dependente da função daquele componente. Por exemplo, um componente de média tem uma função de *controller* definida pelo cálculo da média dos dados recebidos e o retorno dos resultados.

A definição dos códigos de *controllers* são especificados pela anotação de *Java RestController*, a qual determina que tudo o que está presente na classe atual faz parte de um controlador para requisições e um mapeamento, definindo qual o tipo de requisição que esse *controller* recebe.

3.3.1.3 ESPECIFICAÇÃO DA ESTRUTURA *MODEL*

A estrutura dos códigos presentes no pacote de *model* define a tipificação de cada dado utilizado dentro dos componentes e suas funções de inicialização. Um modelo é composto pelos atributos presentes em um tipo de dados e seus tipos, além da forma de inicialização desses dados e modelos de recuperação e sua definição.

Na linguagem *Java*, os dados de um modelo são inicializados por uma função de *builder*, que retorna uma nova instância desse dado com os atributos nele definidos. As funções de recuperação e definição de atributos são denominadas *Getters* e *Setters*, respectivamente. Um modelo tem uma função de *Getter* e uma função de *Setter* para cada atributo, as quais são utilizadas para recuperar e definir o valor desse atributo.

São especificados sete modelos de dados usados dentro do sistema:

- *Earthquakes*
- Média
- Mediana
- Moda
- Variância
- Desvio-Padrão
- Amplitude

O modelo denominado *Earthquakes* é adotado para definir os dados recebidos dos bancos, enquanto os outros modelos são usados para guardar os dados do cálculo estatístico referentes a este.

3.3.2 DESENVOLVIMENTO DO COMPONENTE DE CONTROLE

O componente de controle é descrito como o controlador principal de funcionamento do sistema e a única comunicação entre o usuário e ele. Conforme ressaltado anteriormente, nele são definidos os pacotes *controller* e *model*, com a adição do pacote *httpRequests*. O pacote *controller* do componente contém os códigos com as chamadas para cada tipo de banco utilizado e define o fluxo pelo qual o componente chama os outros, além de metrificar o tempo de execução do sistema, enquanto o pacote *model* contém todos os modelos de dados definidos.

O pacote *httpRequests* engloba os códigos de chamadas adotados durante o fluxo para comunicação entre o componente de controle e os outros componentes

desacoplados. Essa comunicação é feita via chamadas HTTP, respectivamente iniciadas dentro do contexto dos *controllers*.

Os códigos do pacote *controller* recebem chamadas provindas de requisições HTTP do usuário e determinam qual fluxo seguir, baseando-se no tipo de banco escolhido.

A partir dessa ação, o programa controlador segue o roteiro definido no *controller* daquele tipo de banco, fazendo requisições HTTP internas para se comunicar com os outros componentes do sistema, a fim de obter os dados do banco e os utilizar para os componentes de análise estatística, retornando ao final os resultados dos cálculos para o usuário.

3.3.2.1 ESPECIFICAÇÃO DA ESTRUTURA *HTTPREQUESTS*

A estrutura dos códigos presentes no pacote *httpRequests* define qual a conexão que é feita pelo sistema, fixando o método empregado para a conexão, como *post* e *get*, e o endereço utilizado.

Para realização da conexão, é necessário definir um *RestTemplate*, uma classe de integração gerada unicamente para um tipo de requisição HTTP. Este permite ao usuário instaurar rapidamente uma forma de conexão, adotando os parâmetros mencionados acima.

3.3.3 DESENVOLVIMENTO DOS COMPONENTES ADAPTADORES

Os componentes adaptadores têm a função de se integrar ao banco de dados e retornar ao componente de controle os dados recebidos, valendo-se do modelo de *Earthquakes*, com cada componente tendo sua própria especificação para conexão entre cada banco de dados, entretanto, seu funcionamento se dá, utilizando o modelo de *Earthquakes*, no qual os dados que recebidos do banco são inseridos. A seguir, é contemplada a especificação de cada um deles.

3.3.3.1 DESENVOLVIMENTO DO COMPONENTE ADAPTADOR DE BANCO JSON

O primeiro componente adaptador é o componente de leitura do banco JSON. Ele realiza a leitura em bancos de dados codificados nessa linguagem e retorna ao componente de controle.

O componente necessita que os dados do banco em formato JSON sejam inseridos em seus recursos e de uma dependência denominada *Gson*, além do modelo *Earthquake*. A dependência *Gson*, criada pelo time de tecnologia do Google, permite a leitura de arquivos JSON e retorna os dados de diversas formas, como listas de modelos pré-definidos e texto puro, usando a função de Gson para adaptar os dados aos modelos, e a função de *JsonReader*, para ler os dados requisitados do arquivo de banco.

O componente tem seu funcionamento iniciado, ao ser requisitado pelo sistema principal. Ao ser chamado, o fluxo do *controller* inicia uma nova lista vazia de modelos de *Earthquake*. A seguir, o sistema recupera o arquivo que está em seus recursos para dentro do código e cria uma nova instância do adaptador *Gson* e de um leitor *JsonReader*, este se interligando ao arquivo recebido dos recursos. Após, o componente utiliza o adaptador *Gson*, interligando ao leitor *JsonReader* para ler os dados do banco e os inserir diretamente na lista adaptada ao modelo *Earthquake*. Por fim, temos o retorno da lista ao componente de controle.

3.3.3.2 DESENVOLVIMENTO DO COMPONENTE ADAPTADOR DE BANCO CSV

O segundo componente adaptador definido é o componente adaptador de banco CSV, com a função de retornar os dados recebidos de um banco desse tipo para o programa principal, ao ser solicitado.

Para o funcionamento do componente, é necessário o modelo *Earthquake* e que o banco de dados CSV esteja inserido nos recursos do componente. Além disso, o componente requer uma dependência de leitura de arquivos em CSV denominada *OpenCSV*. Essa dependência permite a abertura e a adaptação de arquivos CSV para um tipo de modelo específico, como uma lista, caso esse modelo se adapte aos dados inseridos.

O funcionamento do componente se inicia de modo igual aos de seus componentes irmãos, ao ser chamado pelo componente de controle. Ao ser chamado, o fluxo do *controller* cria uma lista vazia de modelos de *Earthquake* e procura o arquivo do banco em seus recursos. Na sequência, o arquivo de banco é lido para o formato do modelo requisitado e retorna os dados para a lista anteriormente vazia. Por fim, o fluxo é finalizado, ao retornar para o componente de controle uma lista recebida do banco de dados.

3.3.3.3 DESENVOLVIMENTO DO COMPONENTE ADAPTADOR DE BANCO SQL

O terceiro e último componente adaptador realiza leituras de dados de um banco do tipo SQL. Este faz a leitura de dados diretamente de um banco inicializado, isto é, não abre um arquivo de banco para ler e retornar os dados ao componente de controle.

Para seu funcionamento, o componente precisa do modelo de dados *Earthquake*, com anotações específicas, das dependências *DataJPA* e H2 *Database*, além de configurações diferentes inclusas no seu arquivo *application.properties*. O modelo de *Earthquake* recebe anotações específicas da dependência de *DataJPA* para ser utilizada, simulando a estrutura de dados de um banco SQL, com a definição de uma chave primária e nomenclatura das colunas usadas nos atributos do modelo. A dependência *DataJPA* insere métodos de leitura de bancos internamente da linguagem *Java*, possibilitando a leitura e escrita desse banco. Enquanto isso, a dependência H2 *Database* possibilita ao componente a virtualização do banco, removendo a necessidade de inserir um ambiente externo de SQL, de sorte a iniciar um ambiente local de SQL, durante a inicialização da aplicação.

Para utilização dessas dependências, o componente necessita de configurações que são aplicadas dentro do arquivo *application.properties* e de códigos da estrutura *repository*, contendo o código de *earthquakeRepository*. As configurações específicas para o componente adaptador de banco SQL são definições de acesso ao banco (*spring.datasource.username = sa* e *spring.datasource.password = ””*, esta última, nula), a definição do motor do banco a

ser usado, neste caso, o H2 *Database* iniciado durante a inicialização do componente (*spring.datasource.driver-class-name = org.h2.Driver*, *spring.datasource.platform = h2 e spring.datasource.url=jdbc:h2:mem:data.sql; DB_CLOSE_ON_EXIT=FALSE*) e especificações de inicialização do banco (*spring.datasource.initialization-mode = always* e *spring.jpa.hibernate.ddl-auto = update*).

O funcionamento do componente começa durante sua inicialização, quando a dependência H2 *Database* inicia o banco e insere os dados dentro do *earthquakeRepository*. Após, o componente fica em espera até que o componente de controle o requisite. Ao ser chamado, o fluxo do *controller* cria uma lista vazia de modelos de *Earthquake* e, logo depois, insere nele os dados, a fim de retornar essa lista para o componente de controle.

A estrutura dos códigos presentes no pacote *repository* determina interfaces para os repositórios do banco de dados SQL, permitindo seu manuseio por funções.

Para utilização do mesmo, não são necessárias funções ou declarações, apenas a extensão da interface para o modelo de dados *JpaRepository*, obtido da dependência *DataJPA*, com definição do modelo de dados adotado (neste caso, *Earthquakes*) e o tipo da chave primária que o modelo utiliza.

3.3.4 DESENVOLVIMENTO DOS COMPONENTES DE ANÁLISES ESTATÍSTICAS

Durante o fluxo de execução do sistema, temos seis componentes que realizam análises estatísticas, a partir dos dados recebidos do banco. Esses componentes empregam os dados recebidos e realizam suas operações específicas, a fim de retornar para o componente de controle seus resultados, sem a necessidade de dependências ou configurações especiais.

Esses componentes são média, mediana, moda, variância, amplitude e desvio-padrão; nas seções seguintes, relataremos mais sobre os mesmos.

3.3.4.1 DESENVOLVIMENTO DO COMPONENTE DE MÉDIA

O componente de Média é o primeiro componente de análises estatísticas a ser utilizado, com a função de calcular a média dos atributos numéricos do modelo *Earthquake* e retornar os resultados obtidos ao componente de controle.

Para o funcionamento do componente, são necessárias as inclusões dos modelos de *Earthquake* e *Media*. O modelo de média é composto pelos atributos de *latMedia, longMedia, depthMetersMedia* e *magnitudeMedia*, definidos pelos cálculos ocorridos durante o fluxo do *controller* do componente.

O fluxo do componente de Média se inicia ao ser chamado pelo componente principal via método *post*, com um vetor de modelos de *Earthquake*. O fluxo tem sua continuidade no *controller*, onde são declaradas as quatro variáveis de média com o valor 0 e uma nova instância do modelo de média. Após as declarações, o fluxo, recorrendo a uma estrutura de repetição, percorre o vetor recebido pelo componente de controle até o tamanho total do mesmo, realizando a soma de cada atributo numérico de cada item do vetor, dentro das variáveis de média.

Assim, as variáveis, nesse ponto, têm seu valor definido como a soma de todos os atributos numéricos do vetor de *Earthquakes*. Por fim, o valor das variáveis é dividido pelo tamanho do vetor recebido e estas são inseridas dentro do modelo de *Media* pelo método de modelo *get*, para ser enviado de volta para o componente principal.

3.3.4.2 DESENVOLVIMENTO DO COMPONENTE DE MEDIANA

O componente de Mediana é o segundo componente de análises estatísticas a ser adotado, com a função de calcular a mediana dos atributos numéricos do modelo *Earthquake*, retornando os resultados obtidos ao componente de controle.

Para o funcionamento do componente, são necessários os modelos de *Earthquake* e Mediana. O modelo de Mediana é composto pelos atributos de *latMediana, longMediana, depthMetersMediana* e *magnitudeMediana*.

Seu fluxo se inicia, ao ser chamado pelo componente principal via método *post*, com um vetor de modelos de *Earthquake*. O fluxo tem sua continuidade no

controller, onde são declaradas uma instância vazia do modelo Mediana e quatro listas vazias para cada atributo do modelo. Após as declarações, o fluxo, utilizando uma estrutura de repetição, percorre o vetor recebido pelo componente de controle até o tamanho total do mesmo, adicionando nas listas de atributos os respectivos atributos do vetor recebido.

Logo depois, são empregados os métodos de *Collections* para organizar as listas em ordem crescente. Na sequência, utilizamos variáveis auxiliares para definir o valor da mediana de cada lista, verificando o tamanho da lista e checando para avaliar se o tamanho total delas é par ou ímpar, e as inserimos dentro do modelo de Mediana definido anteriormente, para assim retornarmos esse modelo para o componente principal.

3.3.4.3 DESENVOLVIMENTO DO COMPONENTE DE MODA

O terceiro componente de análises estatísticas a ser definido é o componente de Moda. Esse componente realiza a contagem de todos os atributos presentes dentro do modelo *Earthquake* e indica o que mais se repete para cada um deles.

Para funcionar, o componente necessita do modelo de *Earthquake* e do modelo Moda, de sorte a guardar os valores que mais se repetem entre eles e a quantidade de repetições. O modelo de Moda é definido por 16 atributos, separados em quais valores de atributo mais se repetem e a quantidade de vezes que eles se repetem.

O fluxo se inicia ao ser chamado pelo componente principal via método *post*, com um vetor de modelos de *Earthquake*. O *controller* dá segmento instanciando um novo modelo de Moda vazio e *mappers* para inserção dos valores dos atributos e a quantidade de vezes que eles se repetem. A partir desse ponto, o vetor recebido pelo componente de controle é percorrido e, caso os valores de seus atributos não estejam inseridos dentro do *mapper* referente ao mesmo, ele insere o valor como chave e 1 como quantidade de vezes que esse valor foi repetido. Caso o valor já esteja dentro do *mapper*, o sistema adiciona 1 ao valor dessa chave. A seguir, os *mappers* são percorridos e o maior valor é instanciado dentro de uma variável auxiliar, a qual é usada para receber a chave dela, recebendo os valores com maior

número de repetições dentro e os inserindo dentro do modelo Moda, para que sejam retornados ao componente de controle.

3.3.4.4 DESENVOLVIMENTO DO COMPONENTE DE VARIÂNCIA

O quarto componente de análises estatísticas a ser definido é o componente de Variância. Esse componente calcula a variância de todos os atributos numéricos do modelo *Earthquake*.

Em seu funcionamento, é necessário o modelo de *Earthquake* e o modelo "Variância", composto por quatro atributos resultantes dos cálculos que ocorrem em seu *controller*.

Ao se iniciar, o fluxo recebe do componente principal um *Mapper* com os valores de média, calculados no componente de Média, e a lista de modelos de *Earthquake*, os quais são inseridos em variáveis auxiliares de cálculo, sendo elas *larVar*, *longVar*, *depthMetersVar* e *magnitudeVar*, declaradas inicialmente com o valor 0.

O vetor de modelo *Earthquake* é percorrido e, para cada valor de atributo, é subtraído o valor de média e elevado ao quadrado. O resultado é somado em uma variável auxiliar de cálculo de variância respectiva para cada atributo.

Após o modelo *Earthquake* ser inteiramente percorrido, os valores presentes nas variáveis de cálculo de variância são divididos pelo tamanho do modelo *Earthquake*, resultando na variância de cada um dos atributos.

Por fim, os resultados dos cálculos de variância são inseridos dentro do modelo Variância, que é retornado ao componente principal.

3.3.4.5 DESENVOLVIMENTO DO COMPONENTE DE DESVIO-PADRÃO

O penúltimo componente de análises estatísticas a ser definido é o componente de Desvio-Padrão. Este utiliza os dados do componente de Variância para calcular o desvio-padrão de seus atributos.

Em seu funcionamento, é necessário o modelo de “Variância” e o modelo de “Desvio-Padrão”. Este último é composto por quatro atributos, resultados de cálculos de seu *controller*.

Ao se iniciar, o fluxo recebe do componente principal o modelo “Variância” e cria uma instância vazia de um modelo “Desvio-Padrão”. Usando os dados recebidos do modelo “Variância”, os dados são percorridos e, para cada um, é realizado o cálculo de raiz quadrada, guardados previamente nas variáveis de auxílio de cálculo, sendo elas *latDesvio*, *longDesvio*, *depthMetersDesvio* e *magnitudeDesvio*.

Por fim, seus resultados são inseridos no modelo “Desvio-Padrão”, o qual é retornado ao componente principal.

3.3.4.6 DESENVOLVIMENTO DO COMPONENTE DE AMPLITUDE

Para finalizar os componentes de análises estatísticas, temos o componente de Amplitude. Esse componente efetua os cálculos de amplitude para todos os atributos numéricos do modelo *Earthquake* e os retorna para o componente de controle.

Para seu funcionamento, o componente necessita do modelo Amplitude, a fim de guardar os dados definidos em seus cálculos, além do modelo *Earthquake*. O modelo de Amplitude é definido por quatro atributos que definem a diferença entre máximo e mínimo de cada atributo numérico do modelo *Earthquake*.

O fluxo do componente de amplitude se inicia, ao ser chamado pelo componente principal via método *post*, com um vetor de modelos de *Earthquake*. O *controller* do componente de Amplitude, ao ser chamado, declara uma instância vazia e oito variáveis, sendo duas para cada atributo numérico do modelo *Earthquake*, uma variável de valor máximo e outra de valor mínimo. Os valores das variáveis de valor máximo e mínimo são definidos para os primeiros valores de entrada recebidos pelos atributos do modelo *Earthquake*, para, a partir destes, realizar as comparações necessárias para cálculo de amplitude.

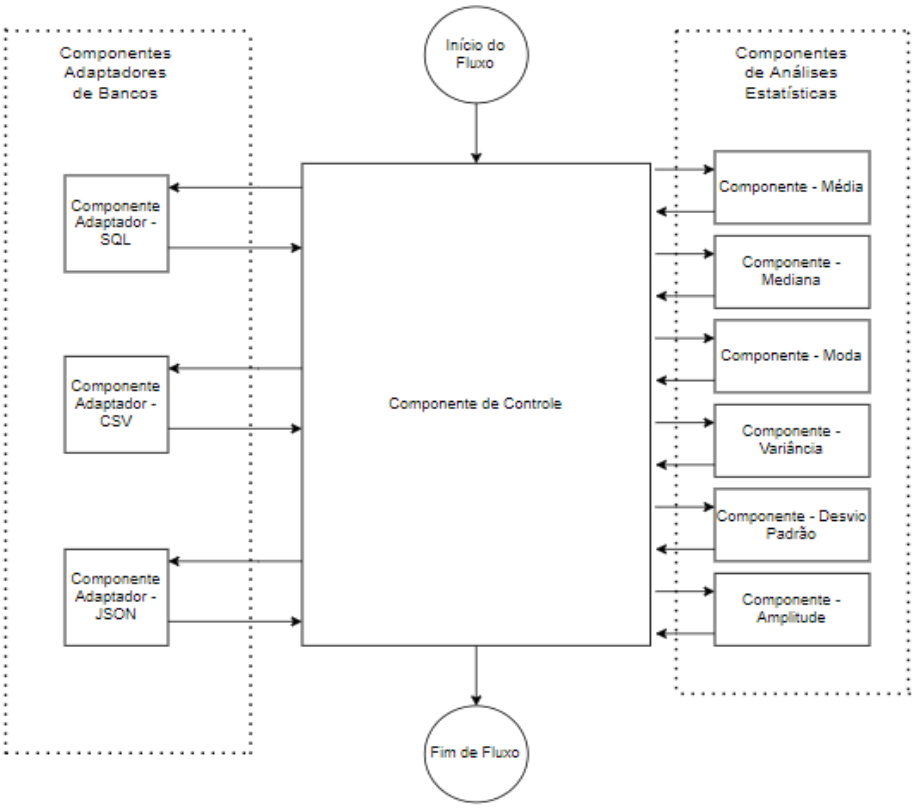
Depois, o vetor de modelos *Earthquake* é percorrido e, para cada modelo, seus dados de cada atributo são validados: caso seu valor seja menor que o valor

mínimo declarado, o valor mínimo recebe esse valor e, se seu valor for maior que o valor máximo, o valor máximo é declarado para esse valor. Com isso, podemos definir o valor máximo e mínimo de cada atributo do banco, os quais são subtraídos e inseridos dentro do modelo de Amplitude instanciado e enviados de volta para o componente principal.

3.3.5 MUDANÇAS DO NÍVEL DE ACOPLAMENTO E DEFINIÇÃO DE CONFIGURAÇÕES

Após o desenvolvimento do sistema ser finalizado, este é acoplado de forma frouxa, ou desacoplado, adotando-se o tipo de acoplamento *Stamp Coupling*, com a utilização de requisições HTTP para comunicações entre si. Essa configuração é denominada “Configuração 1” e sua visão geral está na Figura 3.

Figura 3: Visão geral da “Configuração 1”



Fonte: Autoria própria.

A partir da “Configuração 1”, é iniciado o acoplamento forte de cada componente frouxamente acoplado. Como apenas um componente adaptador de bancos é empregado em cada execução, os três são acoplados fortemente de uma única vez, definindo-se o tipo de acoplamento como *Content Coupling*. Essa configuração, com adaptadores de bancos acoplados fortemente, é denominada “Configuração 2”.

Com a “Configuração 2” acoplando fortemente todos os componentes adaptadores de bancos, principia-se o acoplamento forte de componentes de análises estatísticas, acoplando fortemente o componente de Média e dando origem à “Configuração 3”.

A partir da “Configuração 3”, é realizado o acoplamento da componente de Mediana, gerando a “Configuração 4”, de maneira a dar seguimento ao acoplamento forte de componentes de análise estatística.

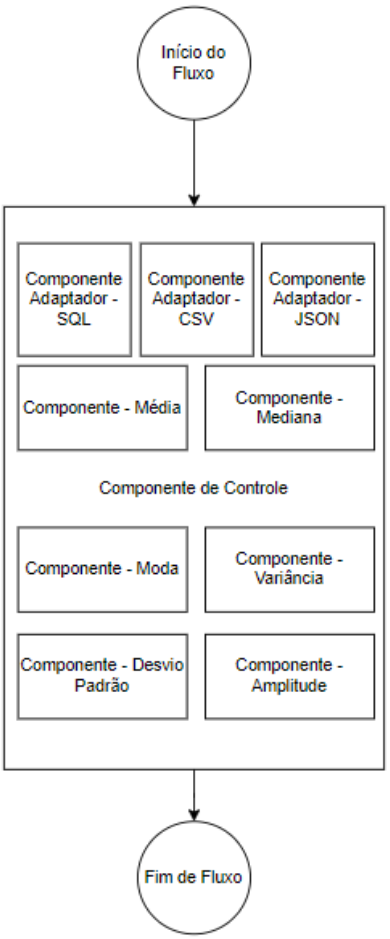
Empregando-se a “Configuração 4”, o componente de Moda é acoplado fortemente, de modo a finalizar o acoplamento forte dos componentes de análise estatística de tendência central e geramos a “Configuração 5”.

Partindo-se da “Configuração 5”, é iniciado o acoplamento forte dos componentes de medidas de dispersão, acoplando fortemente o componente de Variância, gerando a “Configuração 6”.

A penúltima configuração gerada é a “Configuração 7”, criada a partir do acoplamento forte do componente de Desvio-Padrão na “Configuração 6”.

Por fim, com base na “Configuração 7”, é gerada a “Configuração 8”, acoplando fortemente o componente de Amplitude. Essa configuração é fortemente acoplada por completo e pode ser visualizada na Figura 4.

Figura 4: Visão geral da “Configuração 8”



Fonte: Autoria própria.

Capítulo 4

Resultados

No Capítulo 4 deste trabalho, são validados os resultados obtidos por cada configuração, metrificando em quesito nível de acoplamento, desempenho e uso de recursos. Cada teste foi realizado com um reinício total do sistema, a fim de eliminar recursos da memória de testes anteriores e prover um ambiente de testes similar para cada teste efetivado. Na seção 4.1 são verificadas as métricas de acoplamentos pertinentes a cada configuração, na seção 4.2, é focalizada a forma de avaliação e descrição dos testes de desempenho e, na seção 4.3, é descrita a avaliação dos testes de uso de recursos e os resultados são avaliados.

4.1 METRIFICAÇÃO DE ACOPLAMENTO DE CADA CONFIGURAÇÃO

As métricas pertinentes para uma avaliação de acoplamentos são calculadas para cada configuração, adotando-se as métricas de *Instability* (I) e *Abstractness* (A) para os componentes, em cada configuração. A métrica de Fenton e Melton não é utilizada, pois se refere a uma medida entre dois componentes, não a um sistema geral ou a um componente perante o sistema. A Tabela 4 representa os valores de *Instability* para cada componente em cada configuração.

Tabela 4: Valores de *Instability* de cada componente em cada configuração

Configuração	**Valor de *Instability***									
	Controle	**SQL**	**CSV**	**JSON**	**Média**	**Mediana**	**Moda**	**Var.**	**Desvio**	**Amplitude**
Configuração 1	0	0	0	0	0	0	0	0	0	0
Configuração 2	0,50	1	1	1	0	0	0	0	0	0
Configuração 3	0.50	0,50	0,50	0,50	1	0	0	0	0	0
Configuração 4	0,50	0,33	0,33	0,33	0,80	1	0	0	0	0
Configuração 5	0,50	0,25	0,25	0,25	0,66	0,83	1	0	0	0
Configuração 6	0,50	0,20	0,20	0,20	0,57	0,71	0,85	1	0	0
Configuração 7	0,50	0,16	0,16	0,16	0,50	0,62	0,75	0,87	1	0
Configuração 8	0,50	0,14	0,14	0,14	0,44	0,55	0,66	0,77	0,88	1

Fonte: Autoria própria.

Pela Tabela 4, podemos observar que os valores de *Instability* de cada componente do sistema aumenta para cada aumento no grau de acoplamento do sistema. O componente de controle depende do funcionamento de todos os componentes, e todos os componentes dependem dele, determinando uma constante em seu valor de *Instability*.

Também é possível observar que o último componente fortemente acoplado de cada configuração tem um valor de *Instability* máximo, e que os componentes fortemente acoplados anteriormente decaem seu valor de *Instability*, com a adição de novos acoplamentos fortes, entretanto, os valores ainda têm um valor ruim para a estabilidade do sistema.

A Tabela 5 informa os valores da métrica de *Abstractness* para cada configuração do sistema.

Tabela 5: Métrica de *Abstractness* de cada configuração

Configuração	Valor de *Abstractness*
Configuração 1	1
Configuração 2	0,6
Configuração 3	0,5
Configuração 4	0,4
Configuração 5	0,3
Configuração 6	0,2
Configuração 7	0,1
Configuração 8	0

Fonte: Autoria própria.

Na Tabela 5, verifica-se que o valor de *Abstractness* decai a cada configuração, variando entre um sistema completamente abstrato e um sistema completamente concreto.

A cada introdução de acoplamento forte, o sistema tem seu valor decaído por esse componente acabar por ser concreto para esse sistema, impossibilitando seu emprego por outros sistemas, designando um sistema de função única, cujos componentes não são reutilizáveis por outros sistemas.

4.2 METRIFICAÇÃO DE DESEMPENHO DE CADA CONFIGURAÇÃO

O desempenho de cada configuração é calculado com base no tempo de execução do sistema por inteiro. A execução de uma requisição HTTP tem uma volatilidade, em seu tempo de execução, variando entre aproximadamente 0,25 e 0,35 segundos entre a chamada de um componente e seu tempo de resposta. Para isso, foi calculada uma média de 0,30 segundos para cada execução, permitindo, assim, um cálculo de tempo no qual apenas a execução do sistema em si acontece. Cada execução de teste para cada configuração diferente foi metrificada três vezes, a fim de adquirir uma média entre os três valores obtidos. A escolha do valor de três

metrificações para obtenção da média foi feita por ser um número fiel a resultados mais detalhados, visto que, durante o processo de testes iniciais da primeira configuração, testes de dez metrificações mostraram uma média similar comparada com a média retirada de um teste de três metrificações, com uma diferença de apenas aproximadamente 0,01 segundos como margem de erro. A Tabela 6 contém os resultados de desempenho obtidos.

Tabela 6: Resultados dos testes de desempenho

Configurações	**Tempo de cálculo (em segundos)**					
	SQL		**CSV**		**JSON**	
	Com HTTP	**Sem HTTP**	**Com HTTP**	**Sem HTTP**	**Com HTTP**	**Sem HTTP**
Configuração 1	2,88	0,78	2,83	0,73	2,63	0,53
Configuração 2	2,38	0,58	2,15	0,35	2,20	0,40
Configuração 3	1,99	0,49	1,91	0,41	2,10	0,60
Configuração 4	1,67	0,47	1,49	0,29	1,57	0,37
Configuração 5	1,47	0,57	1,24	0,34	1,07	0,17
Configuração 6	0,96	0,36	0,92	0,32	0,75	0,15
Configuração 7	0,79	0,49	0,68	0,38	0,59	0,29
Configuração 8	0,46	-	0,28	-	0,08	-

Fonte: Autoria própria.

Pela Tabela 6, podemos observar o desempenho de cada configuração. Os valores têm a tendência de diminuir, com o aumento do grau de acoplamento, porém, alguns valores acabam por se tornar *outliers*, devido à volatilidade do sistema, em suas execuções e dificuldade de metrificar o tempo médio de uma requisição HTTP. Ao validar a diferença entre a primeira configuração, com um acoplamento mais frouxo, com a última configuração, de acoplamento mais forte, temos uma amplitude máxima de 2,55 segundos, vista na execução do fluxo com um banco JSON. Essa amplitude máxima decai, ao se remover o tempo médio de uma

requisição HTTP, com um máximo de 0,45 segundos, considerando-se o tempo puro para cálculos efetuados.

Esse valor de execução representa a execução de 1 647 entradas. Com ele, podemos estimar que o ganho para cada 1 000 entradas é de aproximadamente 0,27 segundos, para os bancos JSON e de CSV, e de 0,19 segundos, para o banco de SQL. Para obtermos um diferencial maior que 5,00 segundos neste quesito, é necessário um valor de entradas maior que 18 519 e 26 316 respectivamente para cada tipo de banco.

É também possível validar que o banco JSON teve a maior performance entre os três bancos escolhidos, com o tempo de execução utilizando a configuração mais acoplada de 0,08 segundos, mesmo que sua amplitude entre a configuração de acoplamento mais frouxo e a configuração de acoplamento mais forte seja a mesma que o banco CSV. O banco SQL teve o pior desempenho entre os bancos, sendo este 5,75 vezes mais lento que o banco JSON na configuração de acoplamento mais forte

4.3 METRIFICAÇÃO DE USO DE RECURSOS DE CADA CONFIGURAÇÃO

O uso de recursos de cada configuração é metrificado, por meio do gerenciador de tarefas do sistema, verificando-se a memória usada para cada configuração e o pico de uso do processador, ao executar o sistema. De modo a calcular a memória, também é realizado um cálculo para que elimine o uso da IDE utilizada, a fim de que a memória validada seja unicamente do uso da aplicação. A Tabela 7 representa os resultados obtidos nos testes.

Tabela 7: Resultados dos testes de uso de recursos

Configurações	**Uso de CPU em porcentagem**	**Uso de Memória em *Megabytes***
Configuração 1	51,3	1 832,3
Configuração 2	55,1	1 360,2
Configuração 3	53,8	1 136,0
Configuração 4	49,4	1 093,6
Configuração 5	33,6	882,0
Configuração 6	25,4	751,1
Configuração 7	19,7	657,8
Configuração 8	13,7	534,7

Fonte: Autoria própria.

O uso de recursos de memória teve queda constante, à medida que o grau de acoplamento se tornou mais forte. Essa queda é visível pela diminuição de componentes a cada configuração, com menos necessidade de memória para sua execução.

O uso de recursos de processador teve um *outlier*, durante a segunda configuração, causada por um uso maior, devido à leitura de bancos no mesmo sistema, entretanto, é vista uma queda significativa, em função da menor utilização de componentes, resultando em um ambiente de execução de menor uso de recursos com um grau de acoplamento mais forte.

A recorrência a recursos do sistema para uma configuração desacoplada pode tornar-se um peso para sistemas com alto uso e poucos recursos de máquina, dificultando, assim, seu emprego.

Capítulo 5

Conclusões Finais

Com a avaliação das configurações propostas, foi possível constatar que um sistema de análises estatísticas acoplado a bancos de dados, com a arquitetura de desenvolvimento vista neste trabalho, tem tempo de execução e gasto de recursos inferiores, em comparação a um sistema de acoplamento frouxo de mesma função. Além disso, essa diferença pode ser mais evidente em sistemas com menos recursos disponíveis para execução, à medida que a falta de recursos torna o sistema ainda mais lento e define uma escolha dependente dos recursos de sistema disponíveis no ambiente.

O acoplamento forte, em sistemas de análises estatísticas, é vantajoso em questão de tempo e pode trazer melhorias na performance de *softwares* estatísticos, entretanto, para se ter uma melhoria visível no tempo de execução do sistema, ao se remover o tempo médio gerado pela arquitetura de desacoplamento de HTTP, é preciso que o tamanho de entradas para o sistema seja alto, a manutenção necessária e a mudança de código, durante sua vida útil, sejam mínimas.

Isso torna a decisão de utilizar acoplamentos fortes, em sistemas de análises estatísticas com associação a banco de dados, algo determinado pela quantidade de uso que o sistema tem e com a manutenção necessária de código, pois existe uma alta perda de estabilidade e resiliência da aplicação, sendo, por conseguinte, mais viável e recomendado para sistemas legados que possam necessitar de performance e sistemas com baixa possibilidade de falhas.

Por esses motivos, ao se verificar o tempo de resposta e o uso de recursos da máquina, torna-se claro que sistemas estatísticos fortemente acoplados têm desempenho superior ao mesmo sistema desacoplado, usando requisições de HTTP, assim como proposto por este trabalho; porém, novas arquiteturas e modificações de acoplamento podem definir melhores formas de se perder menos resiliência e definir melhores formas de se obter desempenho, por meio de acoplamentos fortes.

O sistema pode ser melhorado em certos pontos referentes a seu desenvolvimento e tecnologias utilizadas. A seleção de um banco de dados único foi exigida pela necessidade de a linguagem orientada a objetos precisar de dados pré-definidos em modelos para leitura dos bancos, tornando inviável o emprego de banco de dados genéricos, sem precisar remodelar o sistema para essa função. Assim, uma melhoria, em trabalhos futuros, seria a avaliação de um sistema semelhante desenvolvido em linguagem aberta e com maior flexibilização da leitura do banco.

Outro ponto de melhoria seria a maior variação do nível de acoplamento ocorrido nas modificações, almejando validar e metrificar outros tipos de modificação no nível de acoplamento do sistema, adotando-se outra arquitetura de sistema.

REFERÊNCIAS

ALGHAMDI, Jarallah S. Measuring sofware coupling. **Arabian Journal for Science & Engineering** (Springer Science & Business Media BV), v. 33, 2008. Disponível em: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136.7351&rep=rep1&type=pdf. Acesso em: 4 jul. 2021.

BEZERRA, E.; MATTOSO, M.; XEXEO, G. An analysis of the integration between data mining applications and database systems. **WIT Transactions on Information and Communication Technologies**, v. 25, 2000. Disponível em: https://www.witpress.com/Secure/elibrary/papers/DATA00/DATA00014FU.pdf. Acesso em: 20 abr. 2021.

BORA, Abhijit; BEZBORUAH, Tulsh. Investigation on reliability estimation of loosely coupled software as a service execution using clustered and non-clustered web server. **International Journal of Engineering**, v. 33, n. 1, p. 75-81, 2020. Disponível em: https://www.ije.ir/article_101149_460b2f3d8270e712af952b39718f7b32.pdf. Acesso em: 14 jun. 2021.

DAS, Manaswini. **Coupling in Java**. [*S. l.*], 22 set. 2020. Disponível em: https://www.geeksforgeeks.org/coupling-in-java/. Acesso em: 16 jun. 2021.

FAGARASAN, Ioan. **Thoughts on Coupling in Software Design**. [*S. l.*], 25 jul. 2016. Disponível em: https://www.codurance.com/publications/software-creation/2016/07/25/thoughts-on-coupling-in-software-design. Acesso em: 16 jun. 2021.

FENTON, Norman; MELTON, Austin. Deriving structurally based software measures. **Journal of Systems and Software**, [*S. l.*], p. 177-187, 1 abr. 1990. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/016412129090038N. Acesso em: 21 jun. 2021.

FERENHOF, Helio Aisenberg; FERNANDES, Roberto Fabiano. Desmistificando a revisão de literatura como base para redação científica: método SSF. **Revista ACB**: Biblioteconomia em Santa Catarina, Florianópolis, SC, v. 21, n. 3, p. 550-563, 11 nov. 2016. Disponível em: https://www.researchgate.net/publication/325070845_DESMISTIFICANDO_A_REVISAO_DE_LITERATURA_COMO_BASE_PARA_REDACAO_CIENTIFICA_METODO_SSF. Acesso em: 17 jul. 2021.

GOSLING, James *et al*. **The Java Language Specification**. [*S. l.*: *s. n.*], 2000. Disponível em: https://books.google.com.br/books?hl=pt-BR&lr=&id=Ww1B9O_yVGsC&oi=fnd&pg=PA1&dq=GOSLING+et+al.,+2000&ots=Sh0JeoRdqI&sig=fXT7OqYVr5LgMWgOUPpyQXZQ2U4#v=onepage&q&f=false. Acesso em: 21 jul. 2021.

GUALTIERI, Mike. **Hadoop Is Data's Darling For A Reason**. [*S. l.*], 21 jan. 2016. Disponível em: https://go.forrester.com/blogs/hadoop-is-datas-darling-for-a-reason/. Acesso em: 9 abr. 2021. Acesso em: 9 abr. 2021.

IBM. **IBM SPSS Statistics 27 Documentation**. 3006603. [*S. l.*], 24 maio 2021. Disponível em: https://www.ibm.com/support/pages/ibm-spss-statistics-27-documentation. Acesso em: 21 jul. 2021.

JOVANOVIC, Zeljko *et al*. **Java Spring Boot Rest WEB Service Integration with Java Artificial Intellgence Weka Framework**. *In:* INTERNATIONAL SCIENTIFIC CONFERENCE, [*S. l.*], 17 nov. 2017, p. 270-274. Disponível em: https://www.researchgate.net/profile/Sinisa-Randic/publication/321757987_Java_Spring_Boot_Rest_WEB_Service_Integration_with_Java_Artificial_Intellgence_Weka_Framework/links/5a305d44aca27271ec8a07f8/Java-Spring-Boot-Rest-WEB-Service-Integration-with-Java-Artificial-Intellgence-Weka-Framework.pdf. Acesso em: 28 jul. 2021.

KANJILAL, Joydip. **The basics of software coupling metrics and concepts**. [*S. l.*], 22 set. 2020. Disponível em: https://searchapparchitecture.techtarget.com/tip/The-basics-of-software-coupling-metrics-and-concepts. Acesso em: 16 jun. 2021.

KAR, Josika. **Software Engineering | Coupling and Cohesion**. [*S. l.*], 30 jul. 2021. Disponível em: https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/. Acesso em: 7 jul. 2021.

MARTIN, Robert C. **Agile software development**: principles, patterns, and practices. [*S. l.*: *s. n.*], 2002. Disponível em: https://archive.org/details/agilesoftwaredev00robe. Acesso em: 18 jun. 2021.

MELTON, Jim. SQL language summary. **ACM Computing Surveys**, [*S. l.*], v. 28, n. 1, p. 141-143, 1 mar. 1996.

ONODA, Mauricio; EBECKEN, Nelson FF. Implementação em Java de um Algoritmo de Árvore de Decisão Acoplado a um SGBD Relacional. *In:* SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, XVI. **Anais** [...]. [*S. l.*]: **SBBD**, 2001. Disponível em: https://www.researchgate.net/profile/Nelson-Ebecken/publication/221535942_Implementacao_em_Java_de_um_Algoritmo_de_Arvore_de_Decisao_Acoplado_a_um_SGBD_Relacional/links/0deec51535493d9b8f000000/Implementacao-em-Java-de-um-Algoritmo-de-Arvore-de-Decisao-Acoplado-a-um-SGBD-Relacional.pdf. Acesso em: 20 abr. 2021.

PEREPLETCHIKOV, Mikhail; RYAN, Caspar; FRAMPTON, Keith. Cohesion Metrics for Predicting Maintainability of Service-Oriented Software. *In:* INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE, 7., 2007. **Anais** [...]. [*S. l.*], p. 328-335, 11 nov. 2007. DOI 10.1109/QSIC.2007.4385516. Disponível em:

https://www.researchgate.net/publication/4292925_Cohesion_Metrics_for_Predicting_Maintainability_of_Service-Oriented_Software. Acesso em: 16 jun. 2021.

REINSEL, David; GANTZ, John; RYDNING, John. **The Digitization of the World**. From Edge to Core, [*S. l.*], 2018. Disponível em: https://resources.moredirect.com/white-papers/idc-report-the-digitization-of-the-world-from-edge-to-core. Acesso em: 9 abr. 2021.

REIS, Edna Afonso; REIS, Ilka Afonso. **Análise Descritiva de Dados**. [*S. l.*: *s. n.*], 2002. Disponível em: http://www.est.ufmg.br/portal/arquivos/rts/rte0202.pdf. Acesso em: 26 jul. 2021.

RODRIGUES, Célio Fernando de Sousa; LIMA, Fernando José Camello de; BARBOSA, Fabiano Timbó. Importância do uso adequado da estatística básica nas pesquisas clínicas. **Brazilian Journal of Anesthesiology**, [*S. l.*], p. 619-625, 1 nov. 2017. Disponível em: https://www.sciencedirect.com/science/article/pii/S0104001417300167. Acesso em: 21 jul. 2021.

ROTHER, Edna Terezinha. Revisão sistemática X revisão narrativa. **Acta Paulista de Enfermagem**, [*S. l.*], p. 5-7, 17 jul. 2007. Disponível em: https://www.scielo.br/j/ape/a/z7zZ4Z4GwYV6FR7S9FHTByr/?lang=pt#. Acesso em: 16 jul. 2021.

RUH, William A.; MAGINNIS, Francis X; BROWN, William J. **Enterprise Application Integration**: A Wiley Tech Brief. [*S. l.*: *s. n.*], 2001.

SEAKOMO, Sedem. Coupling databases and advanced analytical tools (r). 2014. MSc degree in Information Technologies for Business Intelligence (Alberto Abello) - Universitat Politècnica de Catalunya, [*S. l.*], 2014.

TRIOLA, Mario F. **Introdução à Estatística**. 11. ed. [*S. l.: s. n.*], 2005.

WEBB, Phillip. **Spring Boot Reference Documentation**. [*S. l.*], 2021. Disponível em: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/. Acesso em: 21 jul. 2021.