Explorando Padrões de Comunicação para Servir Sistemas Inteligentes

Washington Luiz Meireles de Lima [1], Ygor Tavela Alves da Silva[1], Prof. Dr. Alfredo Goldman[1], e Me. Renato Cordeiro Ferreira[1]

[1] Instituto de Matemática e Estatística - Universidade de São Paulo

Introdução

Sistemas Inteligentes conectam usuários à inteligência artificial (*Machine Learning*, ML) para alcançar objetivos significativos [1]. Diferentemente de sistemas de software tradicionais, cujo ciclo de vida se baseia em mudanças apenas no código, sistemas inteligentes possuem três eixos de mudanças: código, dados e modelo [2]. Esses eixos tornam o desenvolvimento de sistemas inteligentes mais complexo e trazem à tona diversos novos desafios quando comparamos com o desenvolvimento de sistemas tradicionais.

Motivação

A escabilidade é um dos desafios do *workflow* de ML [3] no desenvolvimento de sistemas inteligentes, na qual temos: etapas de coleta de dados, pré-processamento de *features*, treinamento do modelo de ML e, foco desta pesquisa, **como servir o modelo em produção para os clientes**.

Neste último ponto, uma decisão arquitetural importante é a escolha do padrão de comunicação, que depende da interação com o cliente. Os padrões de comunicação podem ser classificados pela responsividade: síncrono ou assíncrono. Para ponderar a escolha de um padrão ou outro, consideramos alguns questionamentos como importantes:

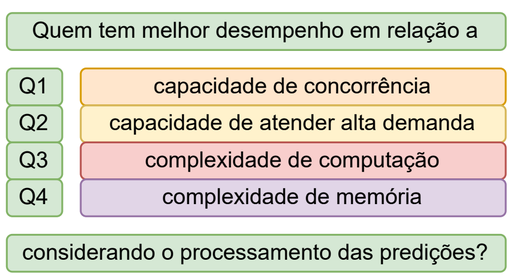

Diagrama 1

Objetivo

No contexto de sistemas inteligentes, a escolha de um padrão de comunicação – e o tipo de responsividade – para servir um modelo de predição podem não se basear apenas em premissas já conhecidas do desenvolvimento de sistemas tradicionais. Por isso, o objetivo desta pesquisa é explorar alguns cenários e *trade-offs* de diferentes padrões de comunicação em um sistema inteligente, como os descritos no Diagrama 1. Para isso, foram desenvolvidas um sistema preditor e um *benchmark* para realização de experimentos.

Arquitetura

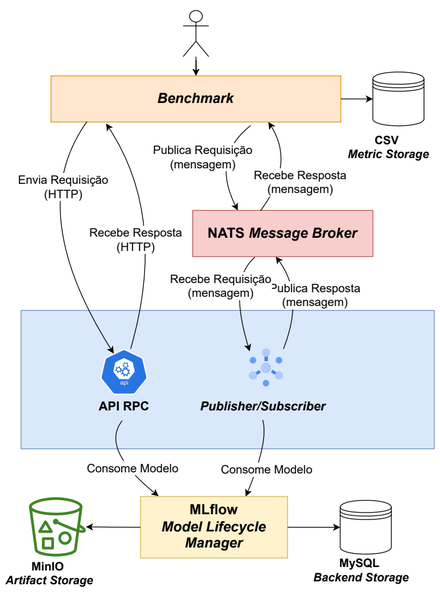

Diagrama 2: Integração dos sistemas: Apresenta a abstração de contêiner no padrão C4 [4], mostrando as relacões entre o *Benchmark* e o Sistema Inteligente via **Mensageria** [5] e **API RPC** [6] cujos tipos de responsividade são, respectivamente, assíncrono e síncrono.

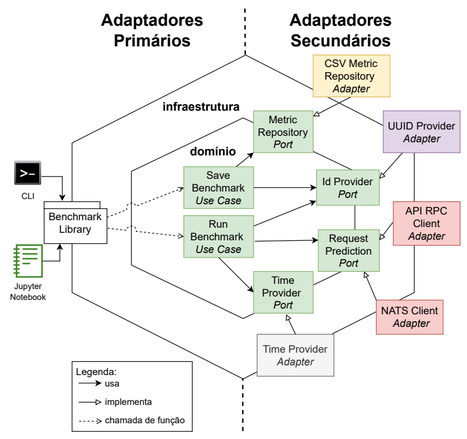

Diagrama 3: *Benchmark*: A abstração de componente no padrão C4 [4], o sistema realiza testes e armazena métricas relacionadas as requisições de predição que são disparadas por de uma API RPC e Mensageria. O sistema adota o estilo arquitetural Hexagonal [7] (Portas e Adaptadores), explicando a forma "hexagonal" do diagrama.

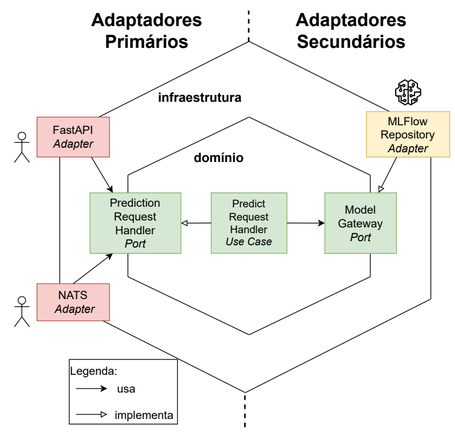

Diagrama 4: Sistema Inteligente: A abstração de componente no padrão C4 [4], o sistema serve um modelo de ML provisionado numa plataforma externa (MLflow). O sistema pode servir o modelo por meio de uma API RPC e Mensageria. Também adota o estilo arquitetural Hexagonal [7].

Metodologia

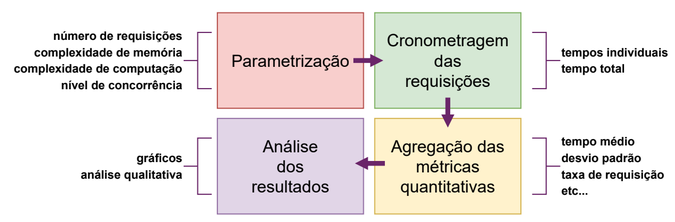

Diagrama 5: Metodologia: O modelo utilizado pelo Preditor é um modelo de simulação que resolve o sistema linear $Ax = b$. O uso de um modelo de simulação possibilitou o controle dos parâmetros de processamento e uso de memória. Assim, não houve a necessidade de servir múltiplos modelos de ML com diferentes comportamentos para entender a influência desses parâmetros sobre diferentes padrões de comunicação.

Resultados

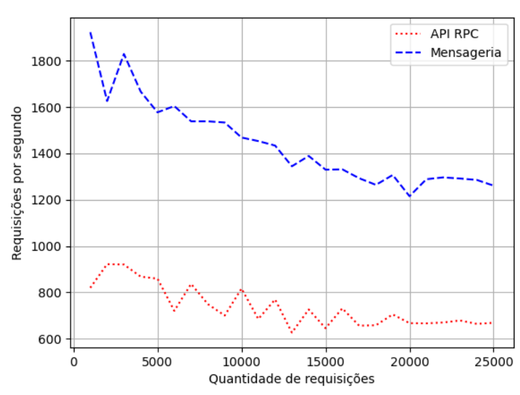

Figura 1: Número de requisições por segundo em função do tamanho uma carga de requisições. A Mensageria teve resultado superior para todas as cargas, mas apresentou uma taxa de decrescimento superior.

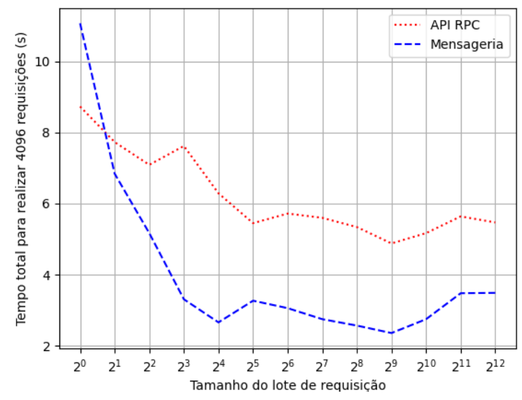

Figura 2: Tempo total para realizar 4096 requisições em função do tamanho do lote de requisições feitas concorrentemente. A Mensageria tem melhor desempenho que API RPC a partir de um lote de tamanho 2.

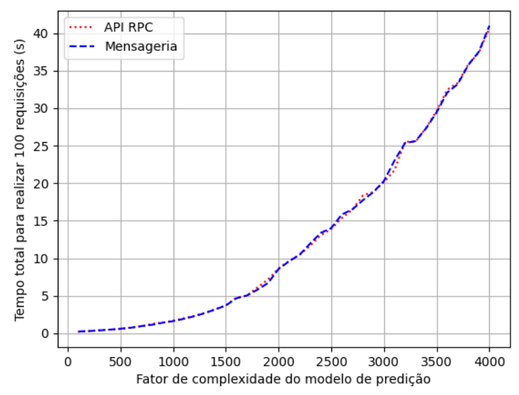

Figura 3: Tempo total para realizar 100 requisições em função da complexidade de processamento (*Complexity Factor*, *CF*). Não há diferenças significativa entre a Mensageria e API RPC.

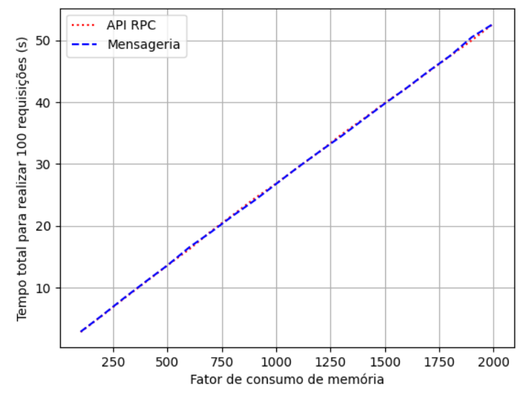

Figura 4: Tempo total para realizar 100 requisições em função do uso de memória (*Memory Overhead*, *MO*). Não há diferenças significativa entre a Mensageria e API RPC.

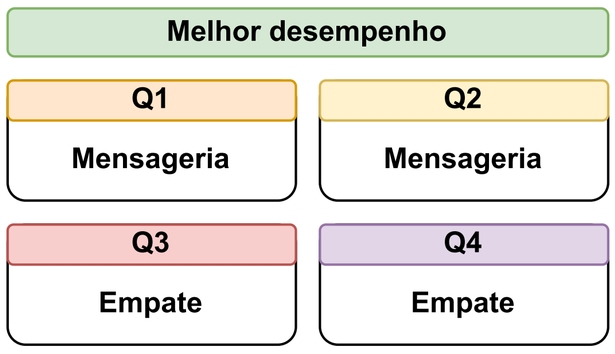

Diagrama 6: Uma comparação qualitativa baseado na análise dos gráficos, em resposta aos questionamentos do Diagrama 1. O termo "Empate" deve ser entendido que não há uma diferença perceptível a nível de gráfico.

Conclusão

Pelos resultados dos experimentos, a Mensageria apresentou um tempo menor de duração das requisições que a API RPC em relação ao volume de requisições e em termos de concorrência. Quanto aos outros cenários, considerando os parâmetros CF e MO, a diferença entre os dois padrões não se mostrou significante com a metodologia adotada. Vale destacar que é possível considerar algumas premissas já conhecidas, como por exemplo, é mais complexo e caro manter sistemas de Mensageria pela necessidade de um *Message Broker*.

Referências

[1] Geoff Hulten. *Building Intelligent Systems*. Berkeley, CA: Apress, 2019. isbn: 978-1-4842-3933-9. doi: 10.1007/978-1-4842-3933-9.
[2] Danilo Sato, Arif Wider e Christoph Windheuser. "Continuous Delivery for Machine Learning". Em: *Martin Fowler* (2019).
[3] Valliappa Lakshmanan, Sara Robinson e Michael Munn. *Machine Learning Design Patterns: Solutions to Common Challenges in Data Preparation, Model Building, and MLOps*. 2020.
[4] Simon Brown. *The C4 Model for Software Architecture*. Jun. de 2018. url: https://www.infoq.com/articles/C4-architecture-model/.
[5] Gregor Hohpe e Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions (Google eBook)*. 2012.
[6] Bruce Jay Nelson. "Remote Procedure Call". Tese de dout. XEROX PARC, 1981.
[7] Alistair Cockburn. *Hexagonal Architecture*. 2005. url: https://alistair.cockburn.us/hexagonal-architecture/.

Para mais informações, consulte https://linux.ime.usp.br/~ygortavela/mac0499/ ou envie um e-mail para luiz.meireles@usp.br ou ygor_tavela@usp.br