Universidade Federal de São Carlos
MBA em Machine Learning in Production
Sistemas Distribuídos

Atividade 1

Valter Alberto Melgarejo Martins

Maio de 2023

Atividade:

Escolha um problema de machine learning que permita otimização de hiperparâmetros e implementê-o com uma arquitetura distribuída (utilizando zmq) de sua escolha (sinta-se à vontade para modificá-la). Salve os modelos treinados no minio se julgar pertinente, para acesso futuro. Containerize sua solução com Docker.

Desenvolva um documento explicando suas escolhas de arquitetura, citando suas vantagens e desvantagens/limitações, crie um diagrama que mostre a arquitetura escolhida (https://lucid.app/)

Envie os arquivos de código e o pdf.

Solução:

Para solução do problema achei interessante a abordagem de Pipeline Pattern, utilizando Push/Pull sockets. A ideia desse tipo de arquitetura é a possibilidade de distribuir a mensagem para vários *workers*, e cada worker distribui a mensagem *downstream* para o *Collector*, não possibilitando o produtor saber se a mensagem teve sucesso, ou não.

Assim, a solução do problema se dá:

- Um *Producer* distribui igualitariamente os hiperparâmetros via PUSH pela porta 8000
- Os *Consumers* recebem os hiperparâmetros via PULL pela mesma porta, adicionando em uma fila quando necessário, treinam o modelo com os hiperparâmetros recebidos, e enviam via PUSH novamente, pela porta 8001.
- Um *Collector* recebe todas as mensagens pelos consumidores via PULL pela porta 8001. E após receber todas as mensagens, identifica qual o melhor modelo pela acurácia, e printa os hiperparâmetros treinados.
- Todo o pipeline pode ser rodado utilizando o comando "*docker-compose up –build*". Onde cada para cada Dockerfile foi exposto a porta requerida para cada fase.
- O docker-compose.yml contém um *Collector*, três *Consumer*'s que dependem do *Collector*, e um *Producer* que depende de algum *Consumer*. Isto se dá devido ao docker criar em paralelo, sendo um dos *downsides* dessa arquitetura, a ordem de criação.

Upsides e *downsides* de usar o padrão *Pipeline* utilizando *PUSH/PULL* sockets:
Upsides:

- Comunicação simplificada: possibilita uma comunicação de *one-to-many* e *many-to-one* de uma maneira simples, o que possibilita uma distribuição eficiente e agregação de trabalho.
- Escalabilidade: existe a possibilidade de adicionar 'n' *workers* que forem necessários, sem precisar modificação de código.
- Processamento Assíncrono: cada *worker* executa sua tarefa de maneira independente e assíncrona, o que possibilita o aumento de *throughput*.
- Modular: cada componente do pipeline pode ser desenvolvido e testado de modo independente dos outros.

Downsides:

- *Bottlenecks*: pode acontecer de um componente ser mais lento que outro, onde em tarefas muito complexas é necessário utilizar mecanismos de balanceamento de cargas ou outros tipos de otimização.
- *Single-failure*: caso algum componente falhe, pode destruir todo o pipeline, sendo necessário a criação de mecanismos de tolerância a falhas e tratamento de erros entre os estágios.