WebCrawler - MagazineLuiza

Desempenhos variados conforme execução em números diferentes de máquinas em um cluster

Gabriel do Vale Rios, Engenharia de Computação. Insper-2018.

Introdução

Hoje em dia é muito comum a pesquisa de diversos produtos em websites. Diversas lojas e empresas possuem uma boa logística de *e-commerce,* o que torna rápido e fácil o acesso e compra de itens variados. Porém, muitas vezes os sites apresentam diversos produtos, e a comparação de preços se torna lenta e tediosa, fazendo com que o consumidor, às vezes, não opte pela melhor opção possível de acordo com suas preferências. Assim surgiram os sites comparadores de preços, onde o consumidor não mais precisa manualmente fazer a comparação de diversos preços, checar informações e etc. Tais comparadores utilizam de uma técnica chamada "*Crawling"*, onde um ou diversos computadores pesquisam e analisam os produtos de acordo com um filtro, este fornecido pelo consumidor.

O trabalho citado neste relatório se inspirou em um destes analisadores de produtos automático, ou ainda "*WebCrawlers*". O objetivo é programar um algoritmo em cluster, usando a ferramenta MPI para dividir o trabalho em um certo número de máquinas, e analisar o desempenho de cada simulação baseado no número de máquinas que foram utilizadas.

Implementação

Todo código citado neste documento pode ser baixado e executado a partir do endereço abaixo:

https://github.com/GabrielValeRios/superComp2018/tree/master/projeto3

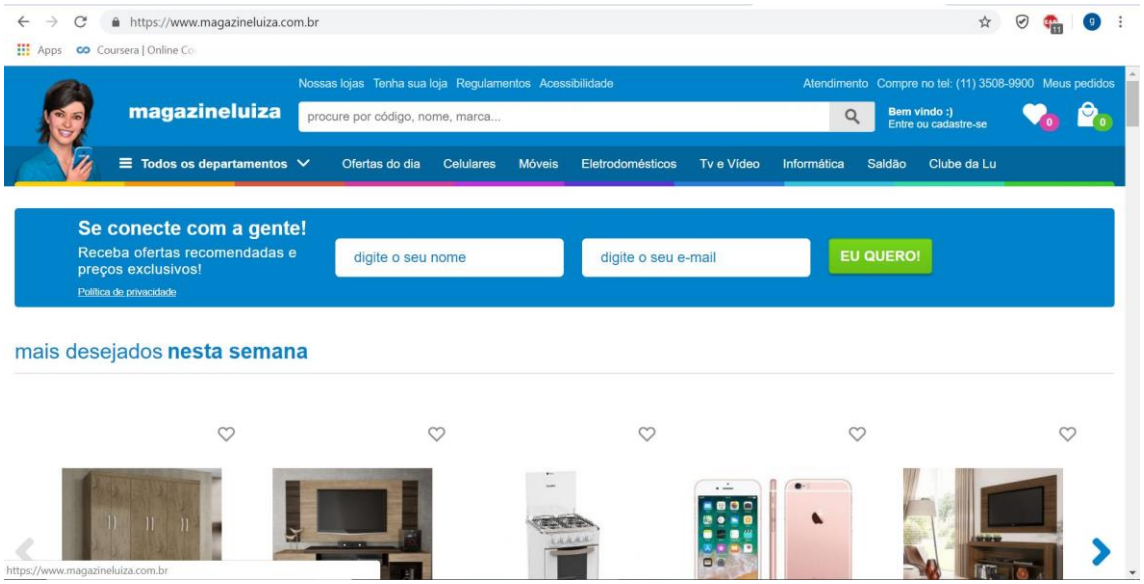
Para a implementação e gestão de todo o código feito para esta simulação, se utilizou a linguagem C++, python e Cmake. O hardware utilizado para compilação e execução das simulações segue abaixo, junto de suas informações:

- Processador: Intel® Core™ i7-4500U CPU @ 1.80GHz 2.40GHz
- Memória (RAM): 16.0 GB
- Tipo de Sistema: Sistema Operacional de 64 bits, processador com base em x64, 4 cores
- Sistema Operacional utilizado: Distribuição Linux (Ubuntu 18.04 LTS)

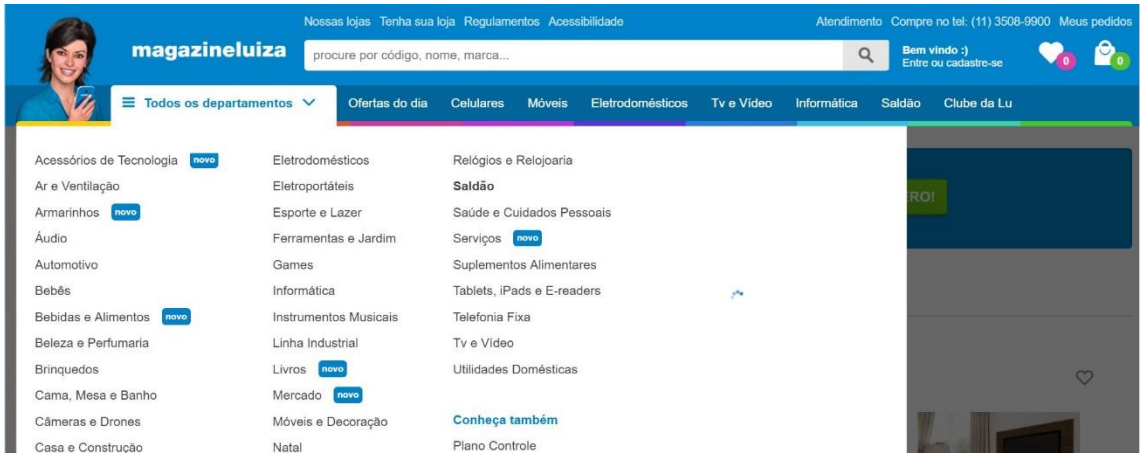
O código busca simular um "*WebCrawler"*, onde informações de produtos de uma certa categoria serão baixadas e processadas. Para a realização dos testes, a empresa escolhida foi a MagazineLuiza, e abaixo segue um passo-a-

passo de como navegar pelo seu site em busca de uma URL válida, ou seja, uma URL que mostra os produtos de uma certa categoria:

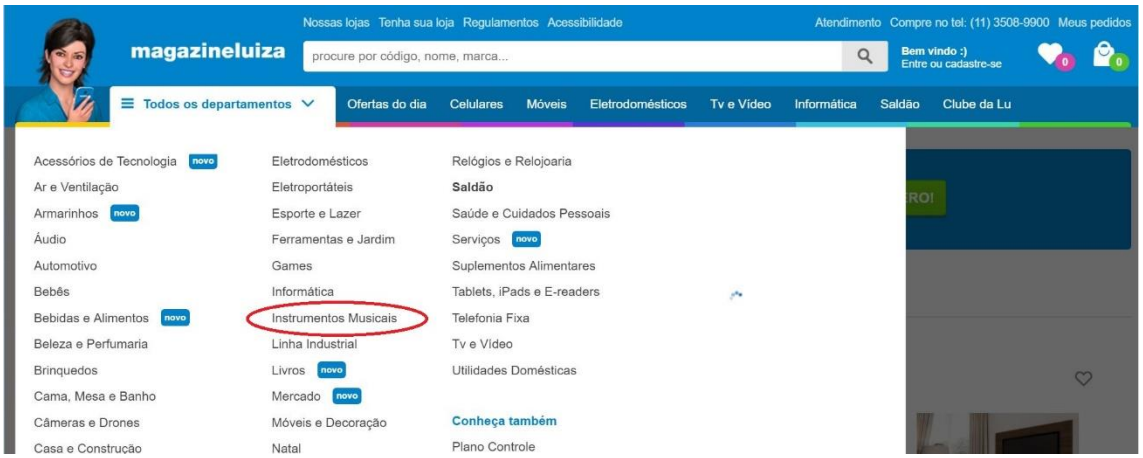
1. Entrar em www.magazineluiza.com.br



2. Nesta página, selecionar a aba “Todos os departamentos”

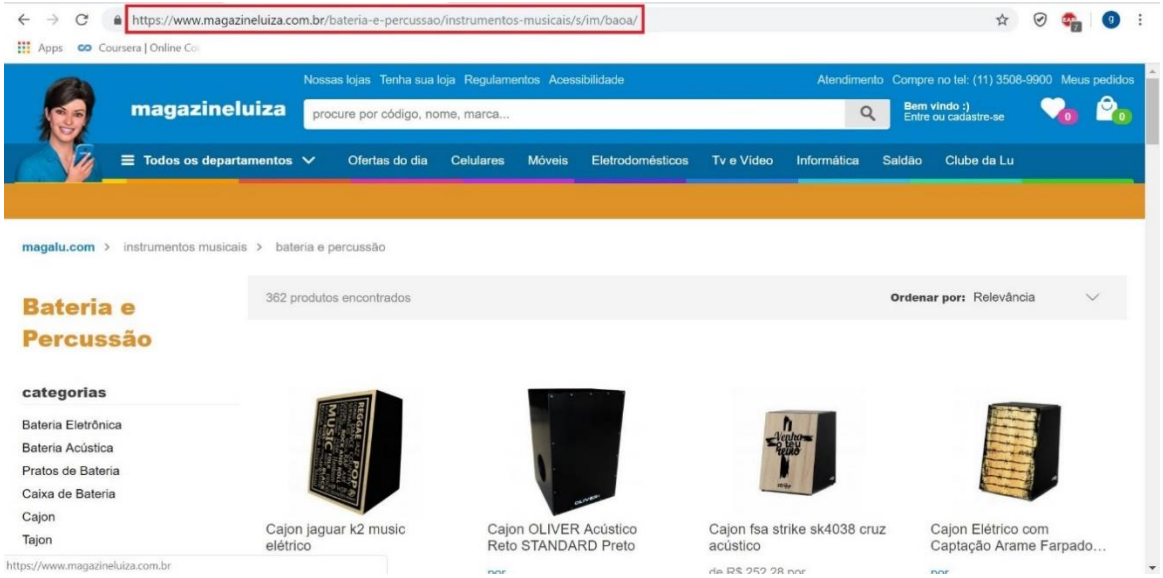


3. Na aba selecionada, para fins de replicar os testes feitos, selecione a aba “instrumentos musicais”.



4. Uma vez que se selecionou esta aba, selecione “Bateria e Percussão”

5. Por fim, o link a ser utilizado para o “crawling” é o indicado na região abaixo



Link: https://www.magazineluiza.com.br/bateria-e-percussao/instrumentos-musicais/s/im/baoa/

- Arquivos em C++

Os arquivos deste projeto para gerar o executável são:

- Controller.cpp, responsável pelo crawling e divisão de tarefas no cluster
- Produto.cpp, responsável por mostrar no bash os resultados do crawling (produtos)

O arquivo produto.cpp é responsável por formatar as informações de cada produto em JSON. Para isso, cada produto analisado é um objeto e existe o método “to.Json()” que realiza essa tarefa.

Controller.cpp é responsável por realizar o crawling e dividir as tarefas no Cluster, como dividir quem seria o nó mestre e seus subordinados. Para identificar o nó mestre, foi utilizado a máquina cujo ID é igual a 0. As demais seriam seus subordinados, e esta identificação do mestre é importante para que os resultados obtidos em cada máquina pudessem ser compilados em um só lugar.

Para realizar a divisão de trabalho, cada máquina possui uma lista de URL’S de produto, que precisa ser baixada e processada. Para isso, foi utilizado um método da biblioteca MPI, chamada *scatter*, que é responsável por separar a lista original de URL’S em listas menores, cada uma designada a uma máquina. Assim, cada máquina separadamente processa uma quantidade menor de URL’S, e os resultados dos produtos de cada máquina são compilados no nó master usando um outro método da biblioteca MPI, chamado *gather*. Tais métodos foram utilizados por simplificar o código do projeto, tornando mais fácil, rápida e simples a implementação. Mais informações sobre os métodos utilizados no link abaixo:

https://www.boost.org/doc/libs/1_41_0/doc/html/boost/mpi/scatter.html

https://www.boost.org/doc/libs/1_41_0/doc/html/boost/mpi/gather.html

- Arquivos em Python

Para que a simulação dos testes fosse feita de maneira menos tediosa, existem 2 arquivos em python que ajudam a realizar os testes:

- buildFiles.py, responsável por gerar o MakeFile através do CMake, e executá-lo, também gerando o executável.
- runMPI.py, responsável por executar o crawler em cluster.

NOTA: Mais informações de como realizar os testes e usar seus resultados para interpretação podem ser encontradas em README, no link:

https://github.com/GabrielValeRios/superComp2018/tree/master/projeto3

- CMake

Para satisfazer todas as exigências para compilação, o arquivo CMakeLists.txt possui todas as configurações necessárias.

Assim, as flags a serem usadas foram:

- Para gerar o executável - "-lboost_regex -lboost_mpi"

Os executável gerado é:

- controller

Simulação e Comparação

Para a simulação e análise de resultados, foi escolhida a categoria “Bateria e Percussão”, de instrumentos musicais. O link para acesso é:

https://www.magazineluiza.com.br/bateria-e-percussao/instrumentos-musicais/s/im/baoa/

Ao fim de cada simulação, é gerado um arquivo de texto chamado “testes.txt”, e nele estão contidos:

- Tempo de cada produto
- Tempo total de simulação (TotalTime)
- Tempo total ocioso (WaitTime)

Para realizar as simulações, foi fixado um número de 2 processos por máquina. Este número foi escolhido baseado no número de produtos no link de exemplo usado, para evitar erros de execução. Para outros links, é possível que o número de processos por máquina seja maior.

O tempo total de simulação foi reduzido ao processo mais demorado dentre os existentes, pois, por estar rodando em um cluster, a latência entre mensagens é desprezível. Mais informações sobre latência podem ser conferidas no link abaixo.

https://computing.llnl.gov/mpi/mpi_benchmarks.html

Assim, para um número constante de 2 processos por máquina, com um número crescente de máquinas (até 7 máquinas rodando simultaneamente no cluster), foi feito uma análise de desempenho do algoritmo. Abaixo, segue o resultado de desempenho (tempo total de execução) conforme o número de máquinas rodando no cluster.

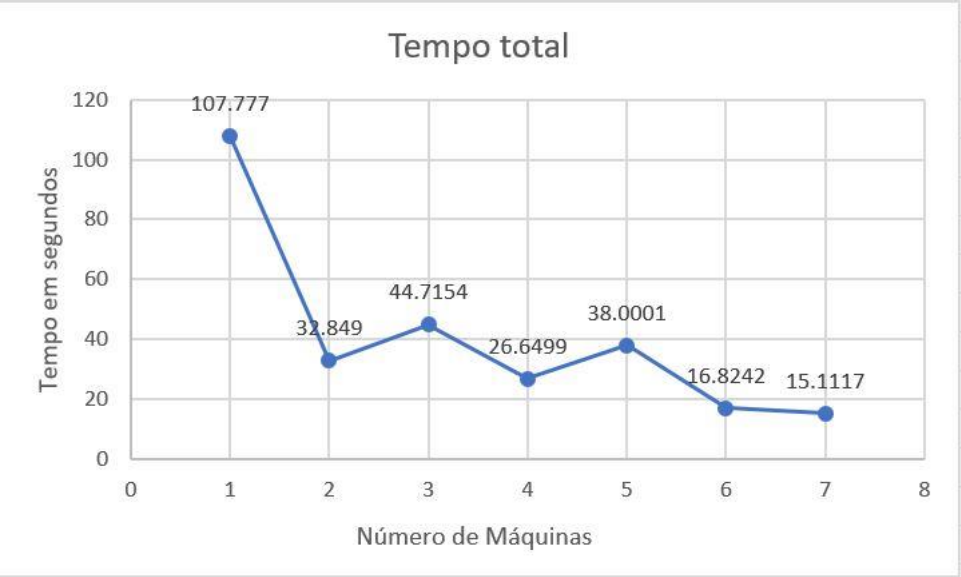


Figura 1 Desempenho por máquina no cluster

Análise e Conclusão

A partir do gráfico analisado, pode-se perceber um grande aumento de desempenho apenas ao se adicionar 2 máquinas ao cluster ao invés de 1 só. Isso era esperado, pois com 1 máquina não existe divisão de trabalho, e com 2 já é possível distribuir a carga de trabalho. Conforme se aumenta o número de máquinas, é visto que existe um aumento de desempenho em relação a 1 máquina só. Porém, para os casos onde o número de máquinas é igual a 3 e 5, existe um pequeno aumento no tempo de execução. Porém no geral, houve uma grande baixa no tempo de execução do crawler, de até 7,12 vezes no caso em que o número de máquinas é igual a 7, por exemplo.

Conclui-se, portanto, que, conforme o cluster possui mais máquinas, o desempenho aumenta consideravelmente. O tempo de execução, para esse teste, mostrou uma melhora de até 7,12 vezes. Existiram alguns casos onde o aumento de máquinas aumentou um pouco o tempo da simulação anterior, porém o aumento de tempo não gerou um desempenho tão destoante dos demais.