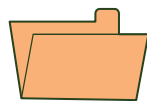
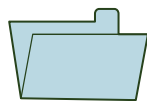
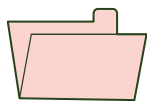


Otimizando a importação de candidatos do SISU no processo de matrícula da UFRJ



Luiz Carlos F. Carvalho
Vitória M. C. Chaves





Contextualizando o problema

Em todas as edições do SISU, a UFRJ recebe um CSV com os candidatos aprovados na chamada regular e os da lista de espera. Este arquivo sempre possui entre 8 e 20 mil linhas com possíveis futuros alunos da UFRJ.

<u>Arquivo CSV</u>	O arquivo geralmente possui informações dos candidatos, que são necessárias para dar início ao processo de matrícula na universidade ou participarem da lista de espera.
<u>Importar Candidatos</u>	Após o recebimento do arquivo, iniciamos a etapa de importar esses candidatos para o banco de dados, mas ainda sem convertê-los nas entidades do banco.
<u>Converter Candidatos</u>	<i>Após a importação ser efetuada, convertemos esses candidatos nas entidades do banco e verificamos se já existem dados existentes ou se são novos e assim, os convertemos para as entidades do banco de dados.</i>
<u>Implantar Candidatos</u>	<i>Após a etapa de conversão ser realizada, iniciamos o processo de implantação, que consiste na persistência dos dados no banco, onde os candidatos já estarão prontos para dar início ao processo de matrícula na UFRJ.</i>

Etapas da solução concorrente

01

Pré-processamento

Carrega um cache de dados já existentes de candidatos que já participaram de processos anteriores.

02

Processamento

Etapa que realiza a conversão dos dados para as entidades do banco de dados.

03

Persistência

Etapa que armazena os dados no banco de dados.

Implementação da solução concorrente

01

ThreadPoolExecutor

Cria um pool de threads, onde utilizamos o método `newFixedThreadPool` para fixar quantas seriam usadas.

```
executorPool.submit() ->
{
    ...
});
```

02

CountDownLatch

Funciona como uma barreira com contador N.

```
latch.countDown();

latch.await();
```

03

CompletableFuture

É uma task que pode ser executada de forma assíncrona com seu resultado sendo retornado futuramente.

```
futures.stream()
    .map(CompletableFuture::join)
    .toList()
```

04

Semaphore

Semaforo do Java

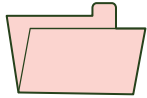
```
new Semaphore(1);

sem.acquire();

sem.release();
```

Testes de Corretude

SQL - Except
SQL - Count



Utilizamos tabelas temporárias para verificar se o resultado concorrente efetuou o processamento dos dados de forma correta. Ou seja, o comando **except** do SQL retorna linhas distintas da consulta de entrada à esquerda que não são produzidas pela consulta de entrada à direita. Dessa forma, conseguimos verificar a diferença dos resultados concorrentes para o sequencial nos diversos conjuntos de dados utilizados. Já o **count**, serve para verificarmos se todos os dados que foram processados foram inseridos no banco de dados.

Resultados dos testes de desempenho

Etapa 1 - Pré-Processamento (maior gargalo)

Tamanho	Threads	Pre-Processamento (s)
1000	1	52
	4	50,6
	8	50,6
	16	50,8
4000	1	208,2
	4	203
	8	202
	16	202
8842	1	460,2
	4	447,6
	8	447,2
	16	447,4

Lotes

Utilizamos lotes de
1000, 4000 e 8842
dados

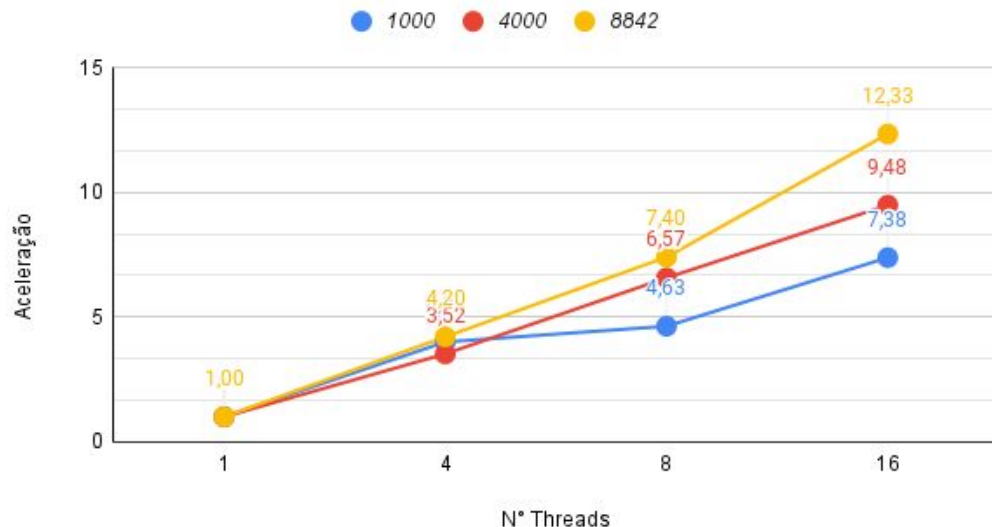
Problemas

Usamos um banco de
dados SQL Server 2005 e
após muitas tentativas
de executar queries
paralelas, ainda assim,
cada execução foi
realizada de forma
sequencial.

Resultados dos testes de desempenho

Etapa 2 - Processamento

Aceleração - Processamento de Candidatos



Lotes

Foram usados lotes de 1000, 4000 e 8842 dados.

Maior aceleração: 12,33 para 8842 dados e 16 threads.

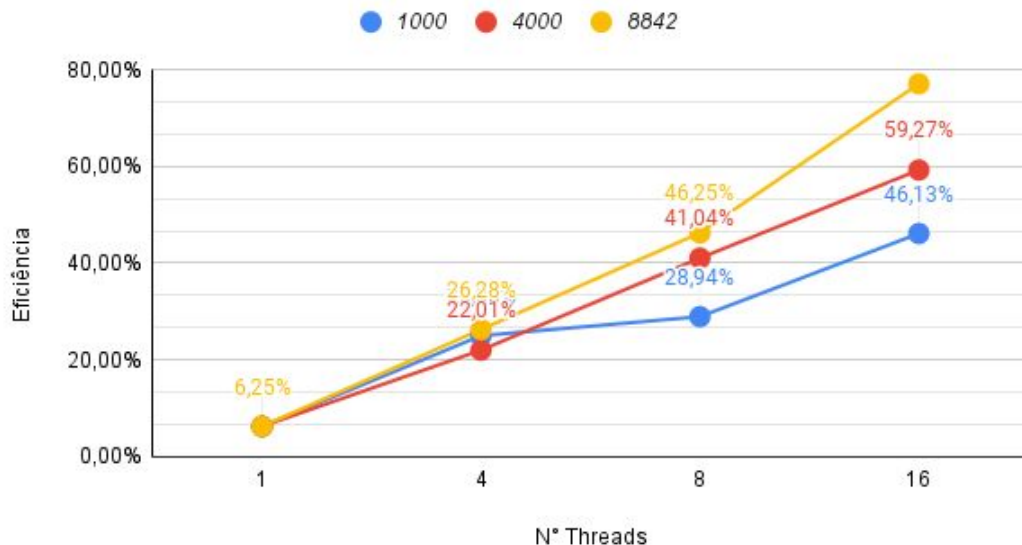
Testes

Realizamos 5 execuções para cada conjunto de testes e estimamos a média dos tempos sequencial (com 1 thread) e concorrente.

Resultados dos testes de desempenho

Etapa 2 - Processamento

Eficiência - Processamento de Candidatos



Lotes

Foram usados lotes de 1000, 4000 e 8842 dados. Maior eficiência: 77,08% para 8842 dados e 16 threads.

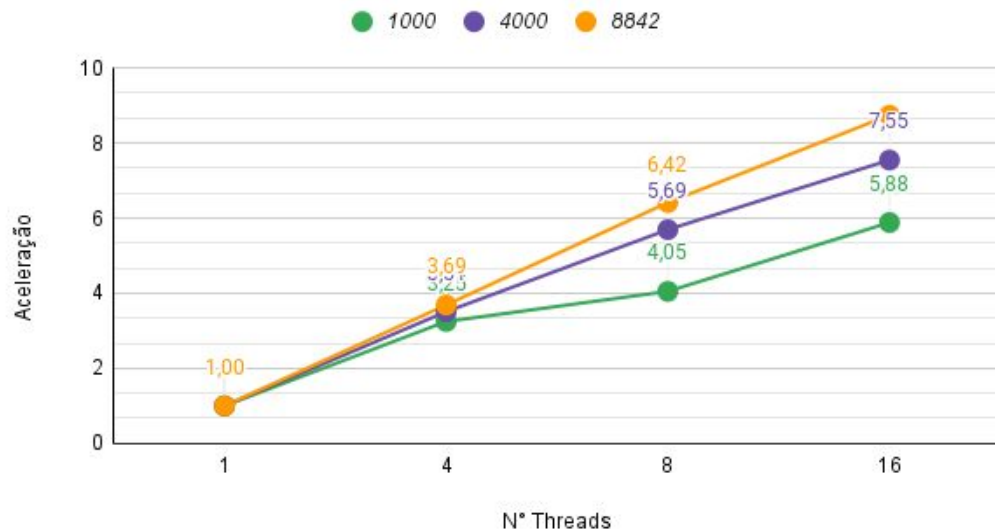
Testes

Realizamos 5 execuções para cada conjunto de testes e estimamos a média dos tempos sequencial (com 1 thread) e concorrente.

Resultados dos testes de desempenho

Etapa 3 - Persistência

Aceleração - Persistência de Candidatos



Lotes

Foram usados lotes de 1000, 4000 e 8842 dados. Maior aceleração: 7,55 para 8842 dados e 16 threads.

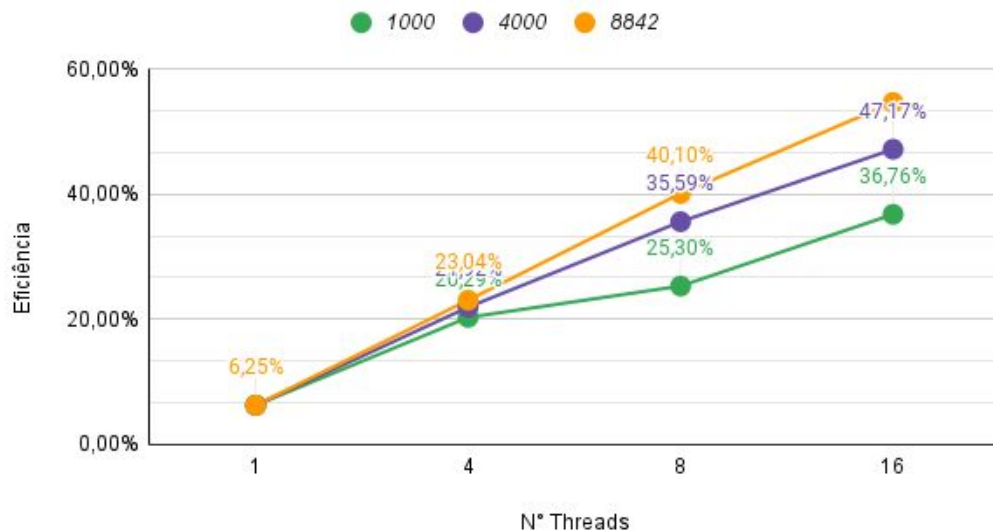
Testes

Realizamos 5 execuções para cada conjunto de testes e estimamos a média dos tempos sequencial (com 1 thread) e concorrente.

Resultados dos testes de desempenho

Etapa 3 - Persistência

Eficiência - Persistência de Candidatos



Lotes

Foram usados lotes de 1000, 4000 e 8842 dados. Maior eficiência: 47,17% para 8842 dados e 16 threads.

Testes

Realizamos 5 execuções para cada conjunto de testes e estimamos a média dos tempos sequencial (com 1 thread) e concorrente.



Obrigado

Accept

Cancel