Relatorio - Projeto3 SuperComp.md

# Projeto 3 – Parsing web pages em cluster

## Super Computação - 2018/2

### Gabriela Almeida

Para o terceiro projeto de super computação expandiu-se o trabalho feito no projeto 2 para rodar em um cluster de máquinas na AWS. O segundo projeto de super computação trabalhou-se com um problema em que concorrência tinha um papel fundamental na obtenção de bom desempenho: download e análise de páginas web. Criou-se um crawler que identifica páginas de produtos em um site de e-commerce e extrai as informações básicas dos produtos. Alguns exemplos são as categorias do site da Magazine Luiza como as seguinte: DVD Player e Controle Remoto.

Para demonstra como a concorrência tem um papel fundamental nesse tipo de problema, esse relatório tem como objetivo demostrar o desempenho de um crawler distribuido, organizado em processos, implementado em MPI em um cluster de máquinas na AWS. Esse relatório também apontará a diferença de desempenho entre um crawler sequêncial, um paralelo organizado em threads e esse distribuído.

## Descrição e Implementação do problema

### Modelo sequencial

A implementação do modelo sequencial foi feita da seguinte forma: dada uma página de exibição de produto, o web crawler extrai as seguintes informações

```
1. nome do produto
2. descrição do produto
3. url da foto do produto
4. preço à vista
5. preço parcelado
6. categoria do produto
7. url da página de axibição
```

A identificação de páginas de produto é feita a partir de sua categoria, ou seja, o crawler desenvolvido é apontado para uma página com os produtos de uma categoria, como os exemplificados anteriormente, e ele consegue obter as páginas de produto, lindando com possíveis paginação da listagem.

Após a compilação do programa, explicado no README, o programa é executado na linha de comando como:

```
./crawlerSEQ url_da_listagem_por_categoria
```

A partir dessa url disponibilizada, o programa faz o download do conteúdo html dessa página, usando a biblioteca *curl*:

```
#include <curl/curl.h>
#include <curl/easy.h>
#include <string>

using namespace std;

//função auxiliar para a funcao "download"
size_t WriteCallback(void *contents, size_t size, size_t nmemb, void *userp)
{
    ((std::string*)userp)->append((char*)contents, size * nmemb);
    return size * nmemb;
}

//Faz o download de uma pagina web a partir de sua url e retorna o seu conteúdo html em formato de string
string download(string url) {
```

```
    CURL *curl;
    string readBuffer;

    curl = curl_easy_init();
    if(curl) {
        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curl, CURLOPT_FOLLOWLOCATION, 1L);
        curl_easy_setopt(curl, CURLOPT_TRANSFERTEXT, 1L);
        curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, WriteCallback);
        curl_easy_setopt(curl, CURLOPT_WRITEDATA, &readBuffer);
        curl_easy_perform(curl);
        curl_easy_cleanup(curl);
    }
    return readBuffer;
}
```

Com o html da página de categoria disponível, é possível adquirir as urls dos produtos presentes nela e a da página seguinte a ela. Para fazer isso, usou-se o regex da biblioteca boost. Essa ferramenta se baseia em expressões regulares para identificação de strings.

```
#include <boost/regex.hpp>

using namespace std;
using namespace boost;

//Procura em uma string de entrada str uma string ditada pelo regex reg e retorna os matches, de acordo com o index de
void findMatches(string str, regex reg, vector<string>& result, int index){
    smatch matches;
    while(regex_search(str, matches, reg)){
        //caso exista um match
        if(matches.size()> 0){
            result.push_back(matches[index]);
            str = matches.suffix().str();
        }
    }
}

//Busca o total de páginas de produtos daquela categoria a partir da primeira página page
int totalPages(string page){
    vector<string> lastPage;
    regex numPages("\"lastPage\":([^,]+)");
    findMatches(page, numPages, lastPage, 1);
    int totalPages = stoi(lastPage[0]);
    return totalPages;
}

//Dado uma página de produtos page, o total de página e o número da página atual, procura as urls dos produtos present
vector<string> findMatchesPages(string page, int totalPages, int numPag){
    vector<string> urlsProducts;
    vector<string> lastPage;

    regex href("name=\"linkToProduct\" href=\"([^\"]+)");
    findMatches(page, href, urlsProducts, 1);

    if(numPag != totalPages){
        regex nextPage("<link rel=\"next\" href=\"([^\"]+)");
        findMatches(page, nextPage, lastPage, 1);
        if(lastPage.size()>0){
            lastPage[0] = "https://www.magazineluiza.com.br"+lastPage[0];
            urlsProducts.push_back(lastPage[0]);
        }
        return urlsProducts;
    }
    //Caso seja a última página, não tem próxima página
    else{
        urlsProducts.push_back("none");
        return urlsProducts;
    }
}
```

Sendo possível adquirir todas as urls das páginas de produtos da categoria ,além das url dos produtos dessas páginas, então é possível fazer o download das páginas de todos os produtos e obter as informações desejadas.

O resultado do programa é escrito na saída padrão no formato json. Cada produto é um objeto com os seguintes campos:

```
{
    "nome": "",
    "descricao": "",
    "foto": "",
    "preco: 0,
    "preco_parcelado": 0,
    "preco_num_parcelas": 0,
    "categoria": "",
    "url": """
}
```

A partir da página de produto page é possivel extrair as informações necessárias, a url do produto já é informação adquirida anteriormente

```
string collectProduct(string page, string url){
    vector<string> buffer;
    string productName;
    string productDescription;
    string productImage;
    string productPrice;
    string precoParcelado;
    string numeroParcelas;
    string productCategory;

    regex name ("<h1 class=\"header-product__title\" itemprop=\"name\">([^<]+)");
    findMatches(page, name, buffer, 1);
    if(buffer.size()>0){
        productName = buffer[0];
    }
    else{
        productName = " ";
    }

    buffer.clear();
    regex desc("<h2 class=\"description__product-title\">([^<]+)</h2>    <p class=\"description__text\"></p>([^<]+)");
    findMatches(page, desc, buffer, 2);
    if(buffer.size()>0){
        productDescription = buffer[0];
    }
    else{
        productDescription = " ";
    }

    buffer.clear();
    regex image("showcase-product__big-img js-showcase-big-img\" src=\"(https[^\"]+)");
    findMatches(page, image, buffer, 1);
    if(buffer.size()>0){
        productImage = buffer[0];
    }
    else{
        productImage = " ";
    }

    buffer.clear();
    regex price("price-template__text[^>]+>([^<]+)</span>");
    findMatches(page, price, buffer, 1);
    if(buffer.size()>0){
        productPrice = buffer[0];
    }
    else{
        productPrice = "0";
    }

    buffer.clear();
    regex parcelado("installmentAmount\": \" ([^\"]+)");
    findMatches(page, parcelado, buffer, 1);
    if(buffer.size()>0){
        precoParcelado = buffer[0];
    }
    else{
```

```
        precoParcelado = "0";
    }

    buffer.clear();
    regex numparcelas("installmentQuantity\": \"([^\"]+)");
    findMatches(page, numparcelas, buffer, 1);
    if(buffer.size()>0){
        numeroParcelas = buffer[0];
    }
    else{
        numeroParcelas = "0";
    }

    buffer.clear();
    regex category("itemprop=\"item\"> ([^>]+)</a>  </li>  </ul>");
    findMatches(page, category, buffer, 1);
    if(buffer.size()>0){
        productCategory = buffer[0];
    }
    else{
        productCategory = " ";
    }

    string out =
    "  {\n"
    "    \"nome\" : \"" + productName +"\",\n"
    "    \"descricao\" : \"" + productDescription +"\",\n"
    "    \"foto\" : \"" + productImage +"\",\n"
    "    \"preco\" : \"" + productPrice +"\",\n"
    "    \"preco_parcelado\" : \"" + precoParcelado +"\",\n"
    "    \"preco_num_parcelas\" : \"" + numeroParcelas +"\",\n"
    "    \"categoria\" : \"" + productCategory +"\",\n"
    "    \"url\" : \"" + url +"\",\n"
    "  },\n";

    return out;
}
```

## Modeleo distribuído

A implementação do modelo paralelo possui uma lógica parecida com a do sequencial. Assim como o modelo anterior, é feito download da página com os produtos, são identificadas as urls dos produtos presentes nela e a url da página seguinte, é feito o download das paginas dos produtos, extraídas as informações necessárias e após a análise de todos os produtos faz-se o mesmo processo com a página seguinte.

No entanto essa implementação é feita a partir de sistemas com memória distribuída, ou seja, são disparados vários processos em máquinas de um cluster e esses são sincronizados a partide de troca de mensagens entre eles.

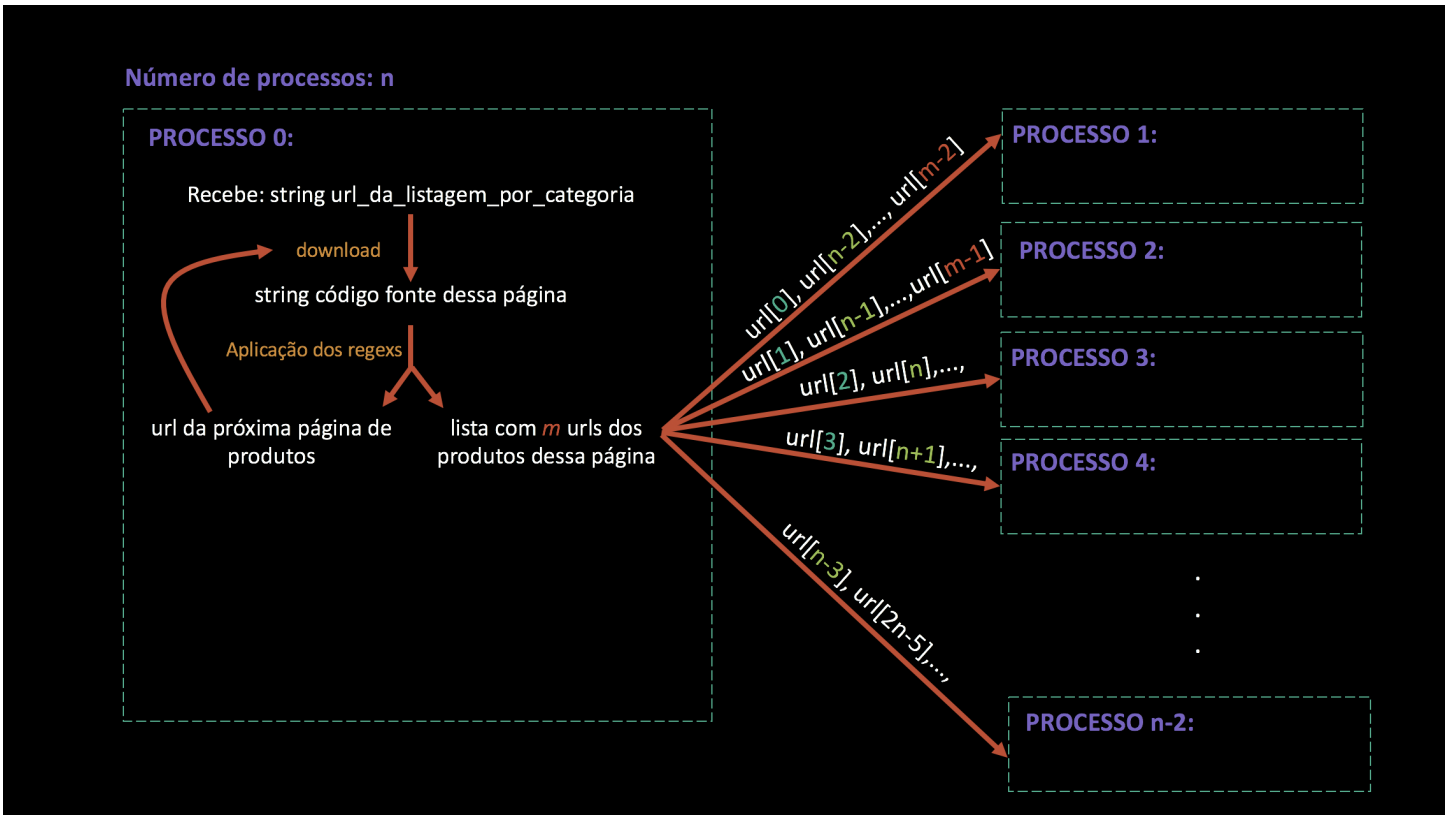
Assim como no sequencial, após a compilação do programa, explicado no README, o programa é executado na linha de comando como:

```
mpiexec –n p –hostfiles hosts ./crawlerDIS url_da_listagem_por_categoria
```

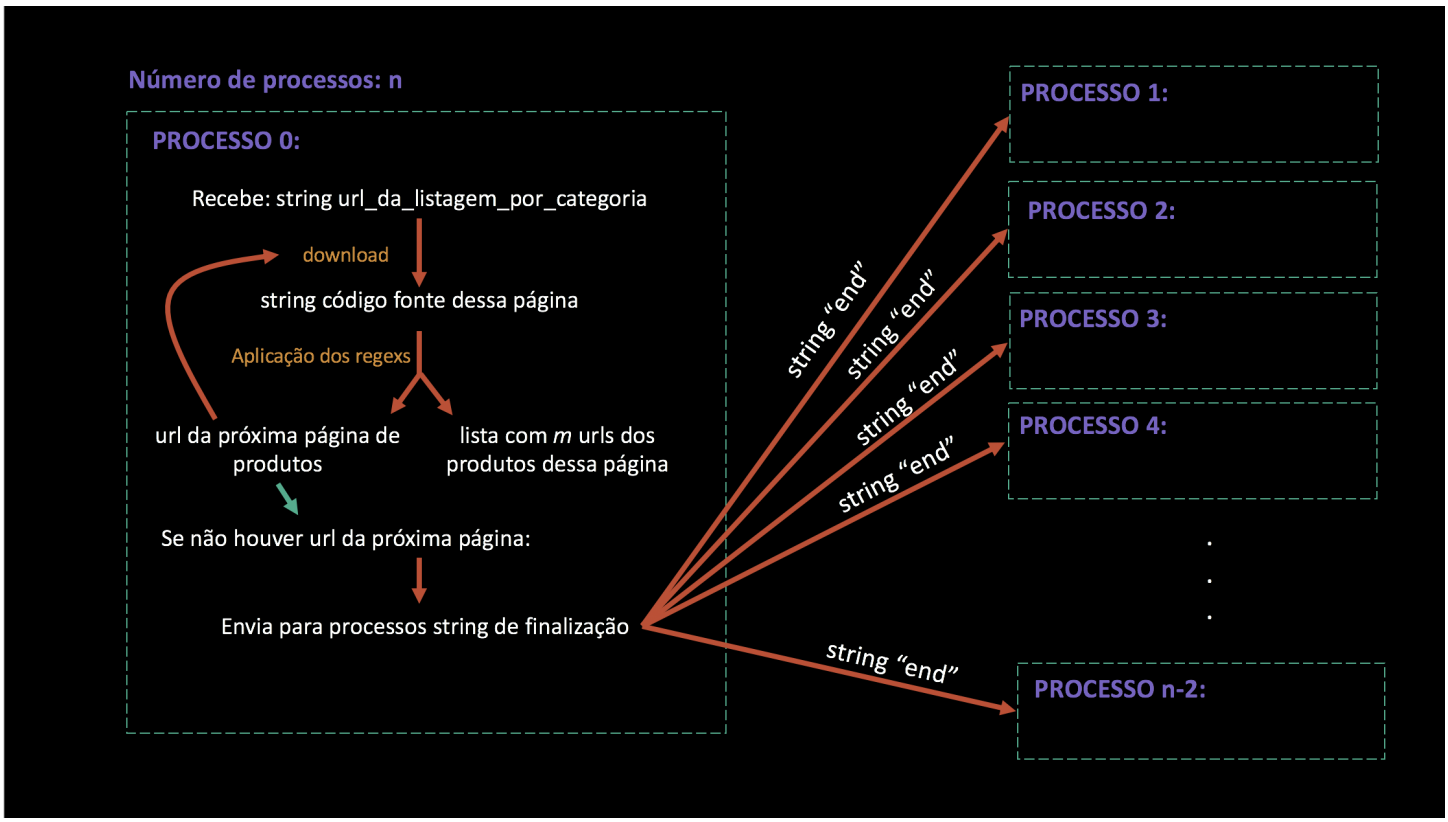
Onde *p* é o número de processos e hosts um arquivo informando os ips das máquinas do cluster que rodaram o programa. Caso queira rodar em somente uma máquina, como em uma local, não coloque essa informação, apenas "mpiexec -n p ./crawlerDIS url_da_listagem_por_categoria"

O modelo distribuido foi estruturado da seguinte forma:

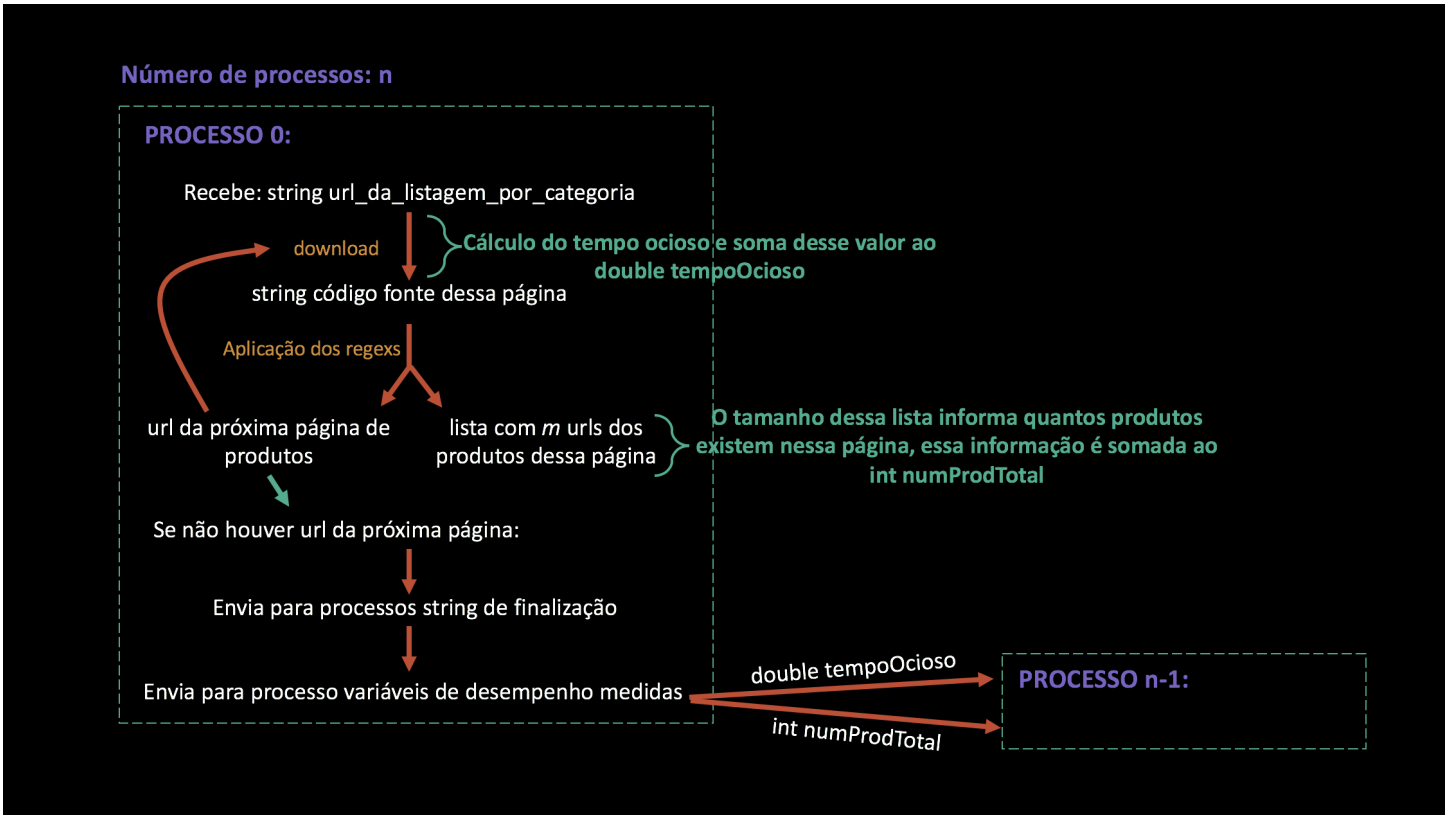
Considerando que existem n processos rodando, o processo de número '0' é o encarregado de receber a url indicada na linha de comando e fazer o download dela. Com a string do código fonte dessa página, resultado desse download, esse processo busca pelas urls desejadas, usando regexs. Com isso ela possui a url da página seguinte e uma lista cujo cada índice é uma string que representa a url de um produto dessa página. Assim o processo 0 distribui as urls dessa lista para os processos que vão de 1 a n-2 em ordem, ou seja, a url[0] vai para o processo 1, a url[1] para o processo 2, a url[2] para o processo[3] e assim por diante. Ao enviar a url[n-3] ao processo n-2 o processo 0 "volta para o início" e entrega a próxima url, url[n-2], para o processo 1, a url[n-1] para o processo 2... Isso acontece até a lista de urls acabar, ou seja, até a url[m-1] for entregue para um processo. Depois disso é feito o download da url da página seguinte e esse processo se repete.




No entanto é preciso adotar também uma estratégia de como informar o fim do processo 0 para esses mesmos processos. Assim, quando não houver mais uma próxima página de produto, o processo 0 enviará para os processos de 1 a n-2 a string de finalização "end".




Como o objetivo desse relatório também é analisar o desempenho do programa é importante apontar como são feitas as trocas de mensagens para que essas medições sejam feitas corretamentes. As medidas levadas em consideração nessa análise serão melhor explicadas na seção "Análise de desempenho", no entanto o que se é preciso saber é que precisa-se calcular o tempo gasto para fazer o download das páginas (tempo ocioso) e o número total de produtos. Esses números são cálculados ao longo do processo 0 e, após enviar a string de finalização aos processos de 1 a n-2, essas variáveis (double tempoOcioso e int numProdTotal) são enviadas ao processo n-1.

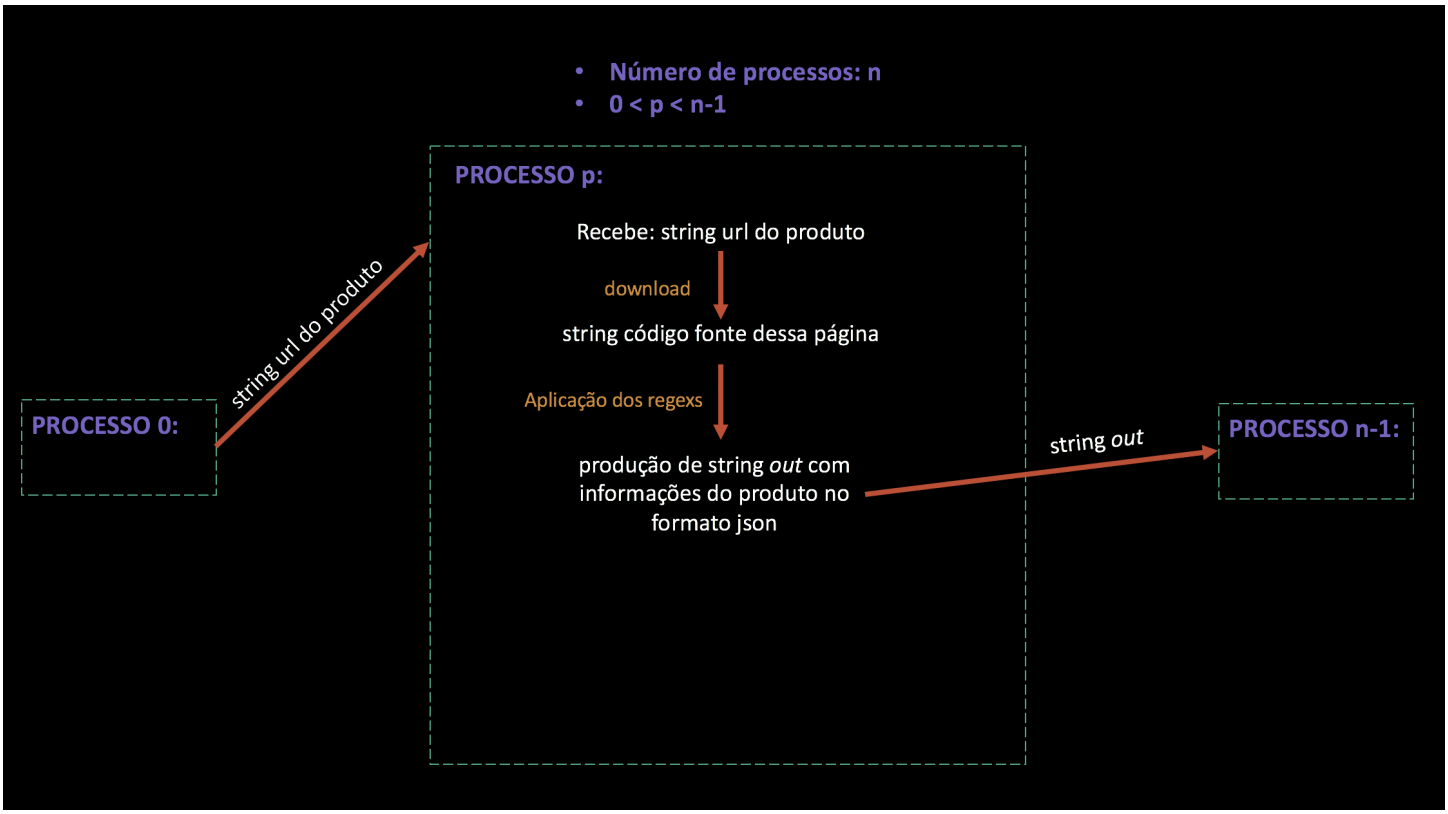



Considerendo a estrutura a cima o código do processo '0' possui o seguinte formato: string url = argv[1];

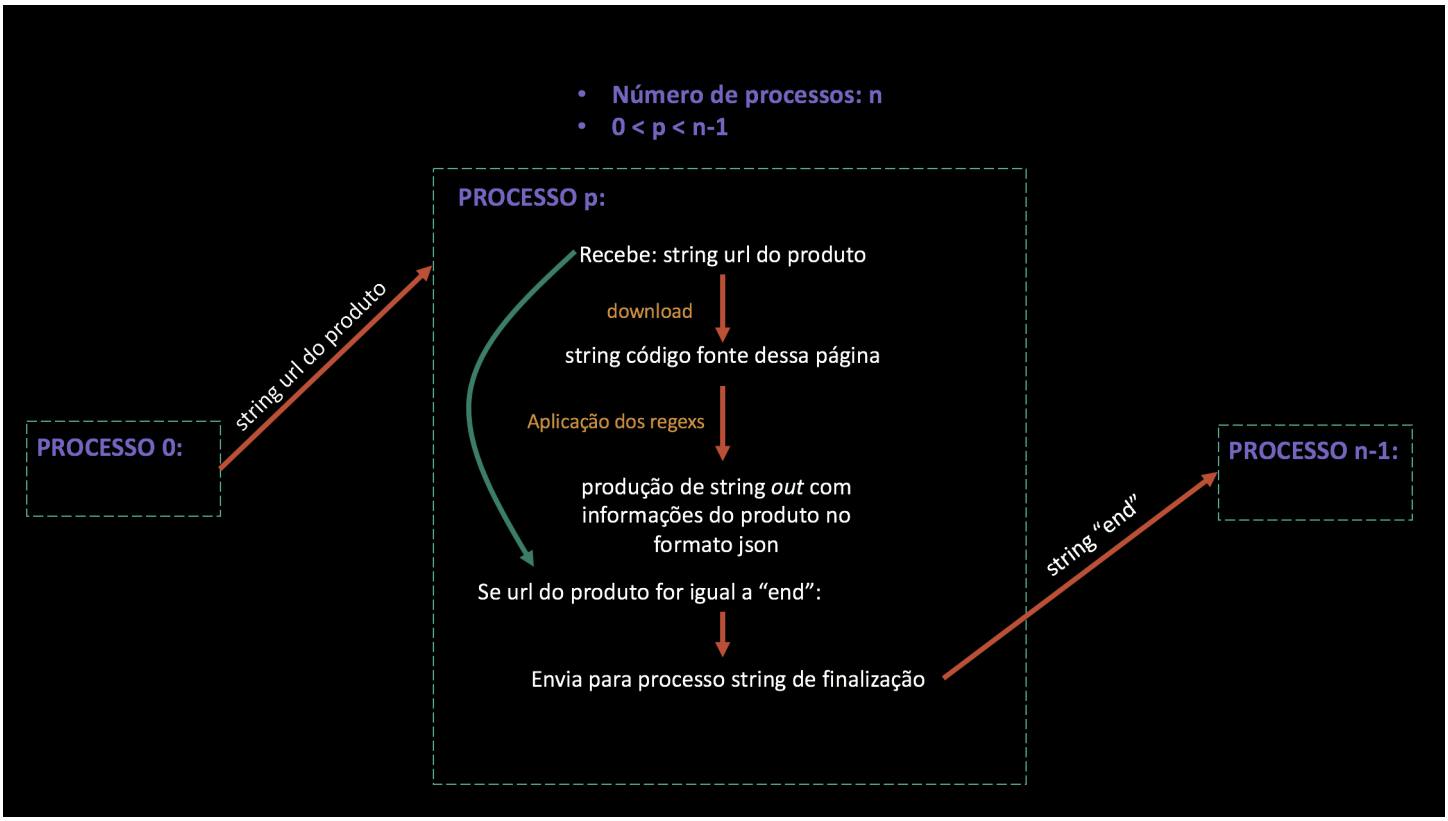
```
boost::mpi::environment env(argc, argv);
boost::mpi::communicator world;

int n = world.size();

if(world.rank()==0){//processo 0

    vector<string> urls;
    string productPage;
    string nextPageUrl;
    int numProdTotal=0;
    int processoAtual = 1; //primeiro processo a enviar é o processo 1

    high_resolution_clock::time_point t1, t2, t3;
    duration<double> ocioso;
    duration<double> tempoProd;
    double tempoOcioso=0;

    t1 = high_resolution_clock::now();
        string currentPage = download(url);
    t2 = high_resolution_clock::now();
    ocioso = duration_cast<duration<double> >(t2 - t1);
    if(ocioso.count()>0)tempoOcioso += ocioso.count();

    int total = totalPages(currentPage);
    for(int p=1; p<=total; p++){
        cout << "pagina " << p << "/" << total << '\n';
        urls = findMatchesPages(currentPage, total, p);
        if(urls.size()>0){
            nextPageUrl = urls[urls.size()-1];
            urls.pop_back();
        }
        else{
            cerr << "Erro ao carregar pagina " << p << "\n"; //Erro no site da magazineLuiza - as vezes a página sai d
            break;
        }

        for(unsigned int u=0; u<urls.size(); u++){
            world.send(processoAtual, 0, urls[u]); //envia para o processoAtual a url[u]
            numProdTotal+=1;
            processoAtual+=1;
```

```
            if(processoAtual == n-1){ //caso o processo n-2 tenha sido o último a receber a url
                processoAtual = 1;//o próximo a receber é o processo 1
            }
        }

        t1 = high_resolution_clock::now();
            currentPage = download(nextPageUrl);
        t2 = high_resolution_clock::now();
        ocioso = duration_cast<duration<double> >(t2 - t1);
        if(ocioso.count()>0)tempoOcioso += ocioso.count();
    }//acabou as páginas contendo produtos

    for (int p = 1; p < n-1; p++){
        string end = ("end")
        world.send(p, 0, end);//envia para os processos de 1 a n-2 a string "end"
    }

    world.send(n-1, 2, tempoOcioso);//envia para o processo n-1 medidas de desempenho
    world.send(n-1, 1, numProdTotal);
}
```

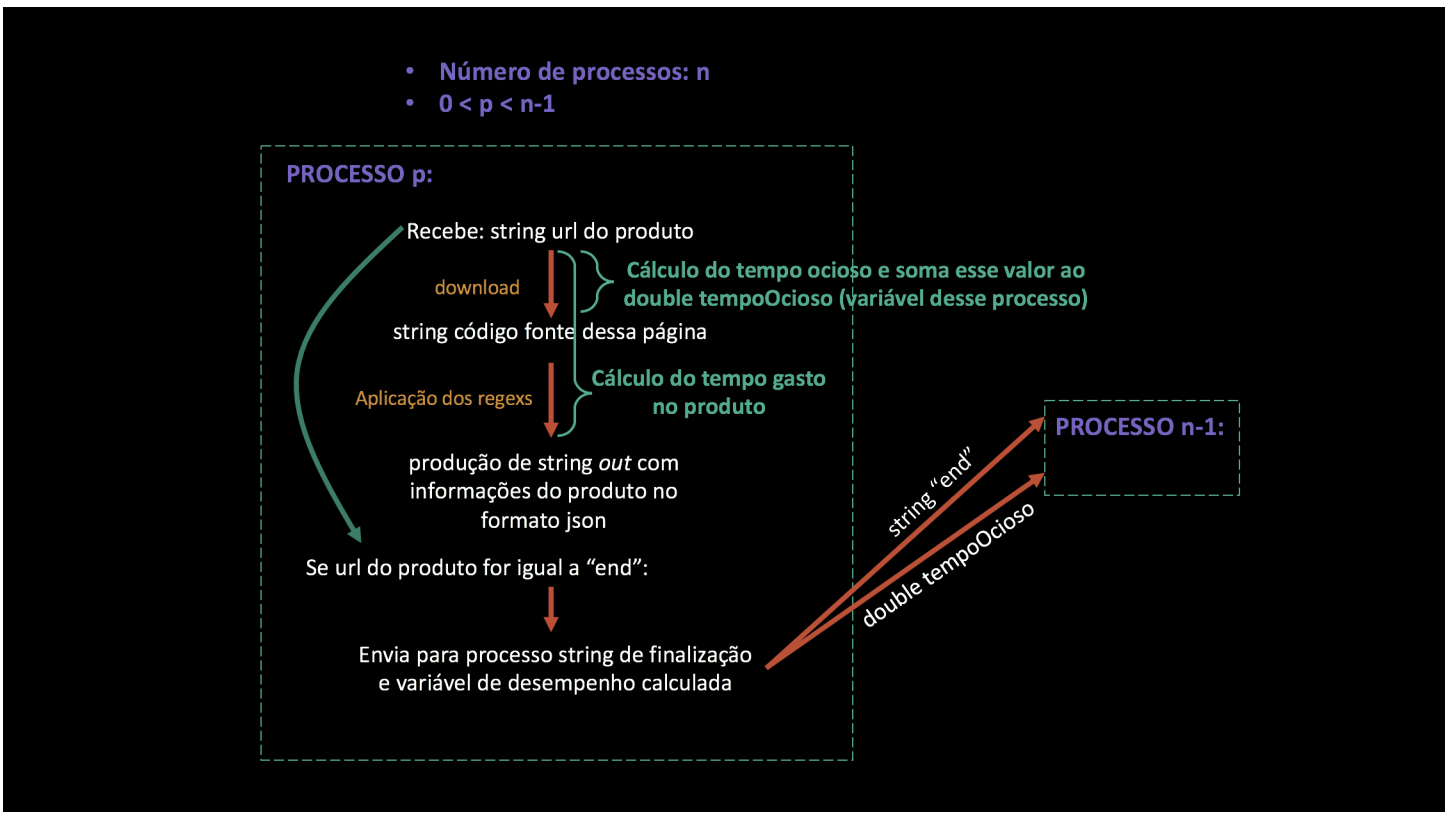
Os processos de 1 a n-2 são encarregados de receber a url do produto advinda do processo 0 e fazer o download dessa página. Com a string do código fonte dessa página, resultado desse download, esse processo busca pelas informações desejadas, usando regexs. Com isso se é produzido a strign 'out' com as informações do produto no formato json. Essa string é então enviada para o processo n-1.




A estratégia para informar ao processo n-1 que os processos ps acabaram é a mesma adotada anteriormente, caso esse processo receba uma url do produto = "end", significando que não existem mais produtos, o processo p enviará ao processo n-1 a string de finalização "end"




Como no processo anterior, esses processos também calculam o tempo ocioso e, além dessa medida, calculam o tempo gasto em cada produto. No entanto, essa última informação não precisa ser passada para outros processos, somente a primeira (double tempoOcioso) é passada para o processo n-1 após o envio da string de finalização.

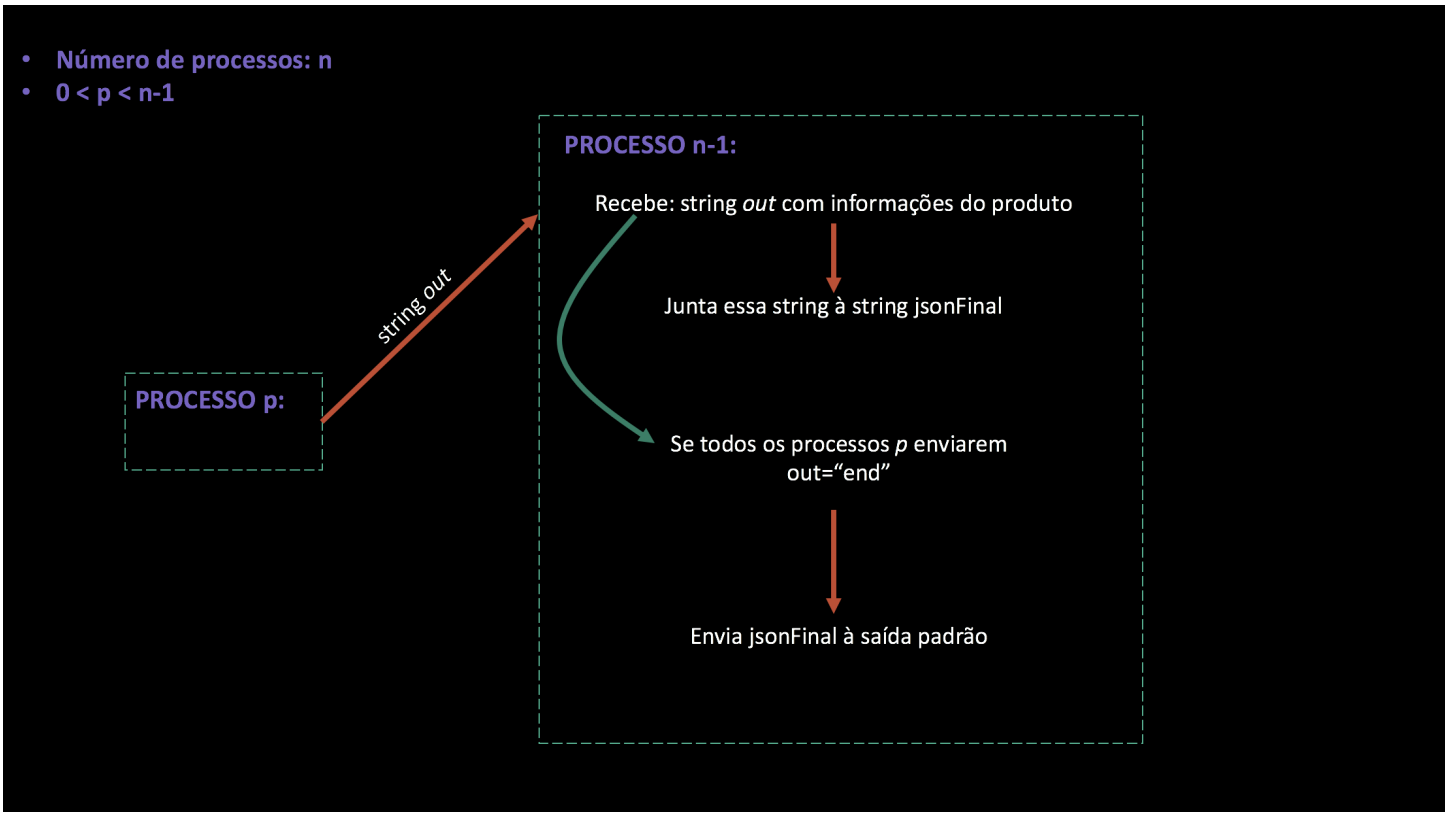



Desse modo a estruturação dos processo *p*s (0<p<n-1) tem o seguinte formato:

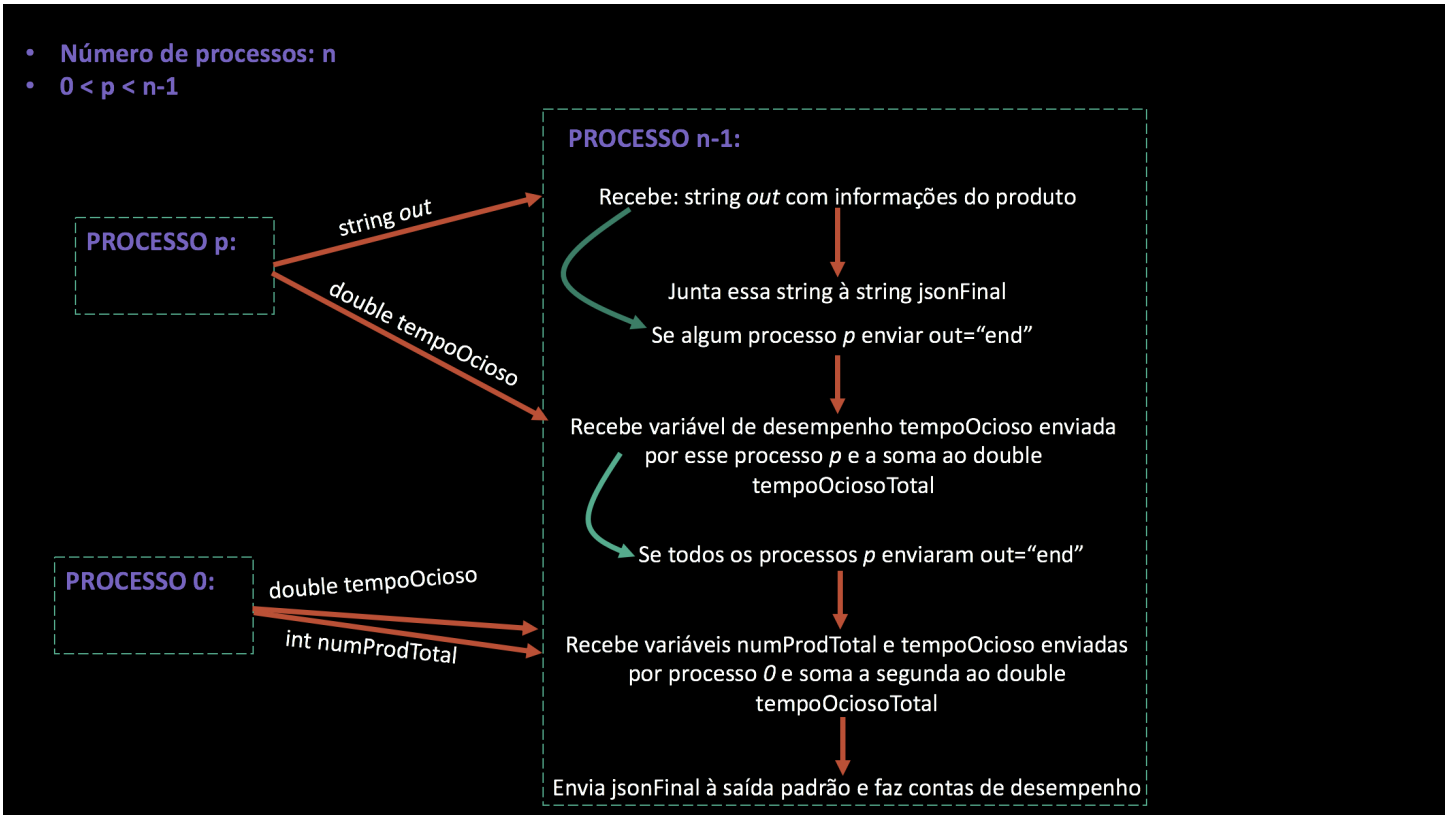
```
else if (world.rank()!= n-1){ //processos de 1 a n-2

    high_resolution_clock::time_point t1, t2, t3;

    duration<double> ocioso;
    duration<double> tempoProd;
    double tempoOcioso=0;
```

```
    while(true){//loop para receber mensagens
        string productPage;
        string url;
        world.recv(0,0,url);//recebe de processo 0 url do produto

        if(url.compare("end")==0){//se url for string de finalização
            string out = "end";
            world.send(n-1, 0, out);//envia para processo n-1 string "end"
            world.send(n-1, 2, tempoOcioso);//envia para processo n-1 medida de desempenho tempoOcioso
            break; // termina loop, não irá receber mais mensagens
        }

        t1 = high_resolution_clock::now();
            productPage = download(url);
        t2 = high_resolution_clock::now();
        ocioso = duration_cast<duration<double> >(t2 - t1);
        if(ocioso.count()>0)tempoOcioso += ocioso.count();

        if(productPage.size()>0){
            string out = collectProduct(productPage, url);
            t3 = high_resolution_clock::now();
            tempoProd = duration_cast<duration<double> >(t3 - t1);
            cerr << "Tempo gasto no produto: " << to_string(tempoProd.count())<<'\n';
            world.send(n-1, 0, out);//envia string out com informações do produto em formato json para processo n-1
        }
    }
}
```

O processo n-1 é o encarregado de receber as informações vindas dos outros processos e juntá-las tornando-as uma informação completa. Esse processo recebe strings 'out' dos processos de 1 a n-1 e as juntam a uma string jsonFinal. Quando todos os processos enviam a string "end" a string jsonFinal é enviada à saída padrão.




No entanto, é preciso lembrar que esse processo não recebe somente a string 'out', ele também recebe o double tempoOcioso de todos os outros processos e também receve o int numProdTotal do processo 0. Assim, são feitos algumas mudanças no esquema acima: quando algum processo enviar a string 'end' é preciso receber o tempoOcioso advindo dele e somar esse valor ao double tempoOciosoTotal. Outra mudança é que quando todos os processos *p* tiverem informado o seu fim é preciso receber o int numProdTotal enviado pelo processo 0 e o tempoOcioso enviado pelo mesmo e somar esse valor ao tempoOciosoTotal.




A estrutura do processo n-1 tem então o seguinte formato:

```
else{
    int countEnd = 0;//contador de quantos processos ja acabaram
    int numProdTotal;
    int processoAtual = 1;
    double tempoOcioso;
    double tempoOciosoTotal=0;
    string out;
    string jsonFinal ="";

    while(true){//loop para receber mensagens

        world.recv(processoAtual, 0, out);//recebe string out de processos 1 a n-2

        if(out.compare("end")==0){//se out for string de finalização
            countEnd+=1;//aumenta contador de processos finalizados
            world.recv(processoAtual, 2, tempoOcioso);//recebe tempoOcioso desse processo que finalizou
            tempoOciosoTotal+=tempoOcioso;//adiciona valor recebido a tempoOciosoTotal
            if(countEnd>=n-2){//se todos os processos finalizaram
                world.recv(0, 1, numProdTotal);//recebe de processo 0 numProdTotal
                world.recv(0, 2, tempoOcioso);//recebe de processo 0 tempoOcioso
                tempoOciosoTotal+=tempoOcioso;//adiciona valor recebido a tempoOciosoTotal
                break;// termina loop, não irá receber mais mensagens
            }
        }

        else{
            jsonFinal +=out;//adiciona string out recebida a string jsonFinal
        }

        processoAtual+=1;
        if(processoAtual == n-1){//caso o processo n-2 tenha sido o último a enviar a string out
            processoAtual = 1;//o próximo a enviar é o processo 1
        }
    }

    cout << jsonFinal << '\n'; //print de jsonFinal com as informações de todos os produtos

    ofstream myfile;
    myfile.open ("../out.txt");//criação de arquivo com medidas de desempenho
    myfile << tempoOciosoTotal << '\n';
    myfile << numProdTotal << '\n';
```

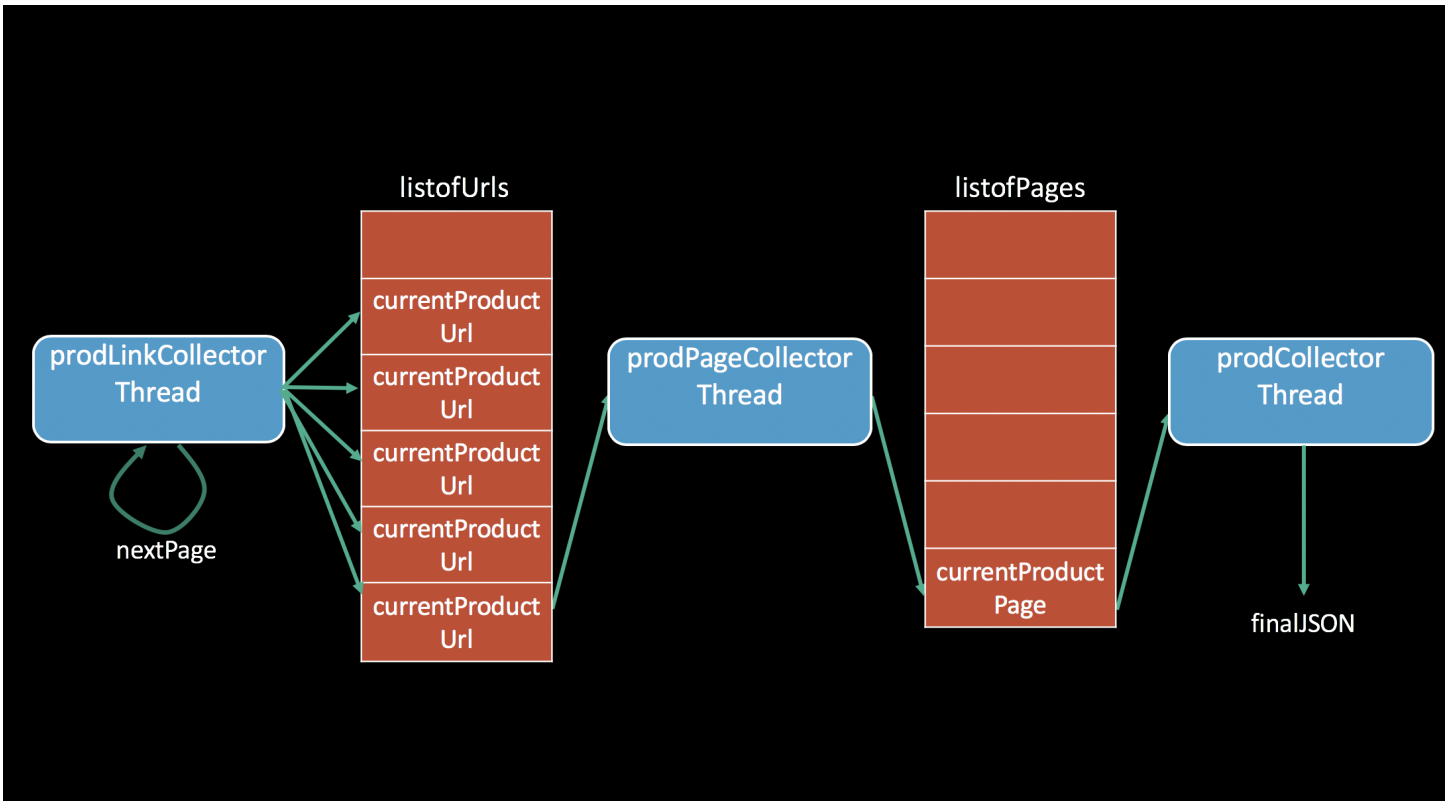
```
    total2 = high_resolution_clock::now();
    tempoTotal = duration_cast<duration<double> >(total2 - total1);

    myfile << tempoTotal.count()/numProdTotal << '\n';
    myfile << tempoTotal.count() << '\n';
    myfile.close();
}
```

## Modelo paralelo

A implementação do modelo paralelo possui uma lógica parecida com a do sequencial. Assim como o modelo anterior, é feito download da página com os produtos, são identificadas as urls dos produtos presentes nela e a url da página seguinte, é feito o download das paginas dos produtos, extraídas as informações necessárias e após a análise de todos os produtos faz-se o mesmo processo com a página seguinte.

No entanto essa implementação é dada em paralelismo por tarefas, ou seja, é preciso sincronizar tarefas que dependem parcialmente umas das outras (parte de uma tarefa depende de um resultado de outra quanto o restante é independente). Ao exercutar as partes independente de maneira paralela pode-se obter ganhos de desempenho consideráveis em processos longos.

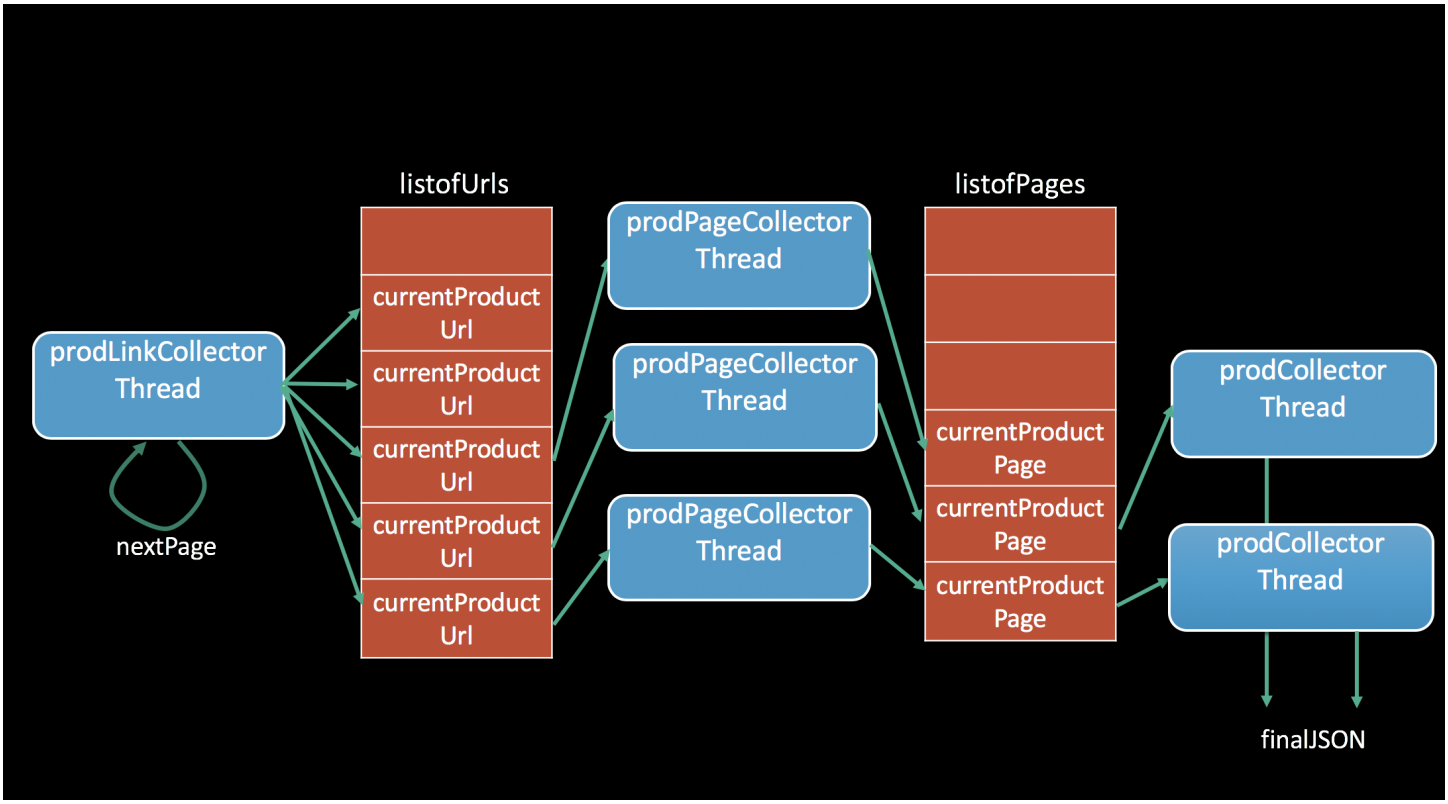
O modelo paralelo por tarefas foi estruturado da seguinte forma:




É possível perceber no esquema a cima que as threads compartilham de alguns recursos fazendo com que elas sejam interdependentes.

A thread prodLinkCollectorThread é encarregada de fazer o download das páginas com os produtos, identificar as urls dos produtos presentes nelas e colocá-las na listofUrls. Além disso essa thread é encarregada de adquirir a url da pagina com produtos seguinte e fazer o mesmo processo, sucessivamente, até a última página dessa categoria. Desse modo, essa thread terá adicionado à listofUrls as urls de todos os produtos disponíveis naquela categoria. A segunda thread, prodPageCollectorThread, precisa recolher uma url da listofUrls, fazer o download dela e adicionar-la à listofPages. Já prodCollectorThread coleta uma pagina de listofPages, coleta as informações desejadas e as adiciona a uma string global "finalJSON".

Levando em consideração essa estruturação, é possível definir multiplas threads do tipo prodPageCollectorThread fazendo com que elas rodem em paralelo.O mesmo pode acontecer com prodCollectorThread.




Como essas threads compartilham algumas estruturas, o uso de semáforos é fundamental para coordenar o acesso de cada thread a elas. Desse modo foram definidos os seguintes semáforos:

```
Semaphore accessListofUrls(1);
Semaphore accessListofPages(1);
Semaphore accessJSON(1);
Semaphore listofUrlCount(0);
Semaphore listofPagesCount(0);
```

O primeiro controla o acesso à listofUrls, o segundo o acesso à listofPages e o tercero o acesso à string global "finalJSON". Já os dois últimos valem como contadores de quantidade de elementos em listofUrls e listofPages respectivamente.

Assim como no sequencial, após a compilação do programa, explicado no README, o programa é executado na linha de comando como:

```
./crawlerPAR url_da_listagem_por_categoria numProducers numConsumers
```

No qual numProducers é o número de threads do tipo prodPageCollectorThread e numConsumers é o numero de threads do tipo prodCollectorThread.

As threads são definidas da seguinte forma:

```
string url = argv[1];
int numProducers = atoi(argv[2]);
int numConsumers = atoi(argv[3]);

thread prodPageCollectorThread[numProducers];
thread prodCollectorThread[numConsumers];

thread prodLinkCollectorThread(produceUrls, std::ref(listofUrls), std::ref(accessListofUrls),std::ref(listofUrlCount),

for(int p=0; p<numProducers; p++){
    prodPageCollectorThread[p] = thread(producePages, std::ref(listofUrls), std::ref(accessListofUrls), std::ref(listo
}

for(int c=0; c<numConsumers; c++){
    prodCollectorThread[c] = thread(consumePages,  std::ref(listofPages), std::ref(accessListofPages), std::ref(listof
    }
```

```
prodLinkCollectorThread.join();

for(int p=0; p<numProducers; p++){
    prodPageCollectorThread[p].join();
}

for(int c=0; c<numConsumers; c++){
    prodCollectorThread[c].join();
}


/*FUNÇÃO de prodLinkCollectorThread - Faz o download das páginas com os produtos, identifica as urls dos produtos pres
void produceUrls(list<string>& listofUrls, Semaphore& accessListofUrls, Semaphore& listofUrlCount, string url, bool& n
    list<string> urlsPage;
    string nextPageUrl;

    string page = download(url);
    int total = totalPages(page);

    for(int p=1; p<=total; p++){
        urlsPage = findMatchesPages(page, total, p);
        nextPageUrl = urlsPage.back();
        urlsPage.pop_back();
        accessListofUrls.acquire();
            for(auto u = urlsPage.begin(); u != urlsPage.end(); ++u){
                listofUrls.push_back(*u);
                listofUrlCount.release();
            }
        accessListofUrls.release();

        page = download(nextPageUrl);
    }
    noMoreUrls = true;
}

/*FUNÇÃO de prodPageCollectorThread - Recolhe uma url da listofUrls, faz o download dela e a adicionar à listofPages*/
void producePages(list<string>&listofUrls, Semaphore& accessListofUrls, Semaphore& listofUrlCount, list<string>&listof
    string currentProductUrl;
    string currentProductPage;
    bool end = false;
    while(!end){
        listofUrlCount.acquire();
        accessListofUrls.acquire();
            if(noMoreUrls && listofUrls.empty()){
                end = true;
            }
            else{
                currentProductUrl = listofUrls.front();
                listofUrls.pop_front();
                if(noMoreUrls && listofUrls.empty()){
                    for(int pt=0; pt< numProducers; pt++){
                        listofUrlCount.release();
                    }
                }
            }
        accessListofUrls.release();

        urrentProductPage = download(currentProductUrl);

        accessListofPages.acquire();
            listofPages.push_back(currentProductPage);
            listofPagesCount.release();
        accessListofPages.release();
    }
    noMorePages = true;
}


/*FUNÇÃO de prodCollectorThread - coleta uma pagina de listofPages, coleta as informações desejadas e as adiciona a st
void consumePages(list<string>& listofPages, Semaphore& accessListofPages, Semaphore& listofPagesCount, Semaphore& acc
    string currentProductPage;
    string jsonProduct;
    string* tempJson = const_cast<string*>(&finalJSON);
    bool end = false;
```

```
    while(!end){
        listofPagesCount.acquire();
        accessListofPages.acquire();
            if(noMorePages && listofPages.empty()){
                end = true;
            }
            else{
                currentProductPage = listofPages.front();
                listofPages.pop_front();
                if(noMorePages && listofPages.empty()){
                    for(int ct=0; ct< numConsumers; ct++){
                        listofPagesCount.release();
                    }
                }
            };
        accessListofPages.release();

        jsonProduct = collectProduct(currentProductPage);

        accessJSON.acquire();
            *tempJson = finalJSON + jsonProduct;
        accessJSON.release();
    }
}
```

## Análise de desempenho

Como dito anteriormente, um dos objetivos desse projeto é comparar o desempenhos entre o crawler distribuido, paralelo e o sequencial. Para fazer essa comparação foi preciso compilar o programa usando as flags correntas. Para o modelo paralelo as seguintes flags foram usadas:

```
-lpthread -O2
```

Para a sequencial a seguinte flag:

```
-O2
```

Para o distribuido as seguintes flags:

```
-O2 -lboost_mpi -lboost_serialization
```

Para fazer essa análise de desempenho, usou-se um cluster na aws com 7 máquinas. O tipo de instância usada nelas é t3.micro cuja especificações são as seguintes:

| Name | vCPUs | RAM (GiB) | Baseline Performance/vCPU | CPU Credits earned/hr | Network burst bandwidth (Gbps) | EBS burst bandwidth (Gbps) |
|---|---|---|---|---|---|---|
| t3.micro | 2 | 1.0 | 10% | 12 | 5 | 1.50 |

E as especificações do sistema operacional dessas instâncias são as seguintes:

```
Distributor ID: Ubuntu
Description:    Ubuntu 18.04.1 LTS
Release:        18.04
Codename:       bionic
```

Para obter os tempos de simulação é preciso executar os seguintes comandos:

```
mkdir build
cd build
cmake ..
make
cd ..
python3 run.py
```

Esses comandos compilam os arquivos com cmake e geram 3 executaveis diferentes: crawlerSEQ, crawlerPAR e crawlerDIS. Depois eles são executados com os 4 diferentes links de categorias diferentes, sendo eles:

- https://www.magazineluiza.com.br/notebook-lenovo-ideapad/informatica/s/in/leip/
- https://www.magazineluiza.com.br/dvd-player/tv-e-video/s/et/tvdb/
- https://www.magazineluiza.com.br/hd-externo/informatica/s/in/hdex/
- https://www.magazineluiza.com.br/adega/eletrodomesticos/s/ed/adeg/

Levando em consideração as especificações das máquinas utilizadas no cluster, optou-se por rodar o crawler paralelo com 3 threads produtoras e 3 threads consumidoras, pois como visto no projeto anterior essa era a configuração mais eficiente para esse cenário.

Já para o crawler distribuído, optou-se por rodar com 4 processos por máquina. Foram feitos testes rodando de 1 a 7 máquinas no cluster.

Para todas essas situações foram medidos o tempo ocioso, que é o tempo total gasto esperando o download de páginas web, o tempo médio por produto, que é o tempo total de execução do programa dividido pelo total de produtos analisados, e tempo total, que é o tempo total de execução do programa.

Por fim, é escrito um arquivo de saída analysis.txt com os valores dos tempos de cada categoria, para cada executável, nas situações descritas a cima, em ordem.

```
t_num_prod=[33,90,178,216]

t_ocioso_seq=[47.6662,81.5548,305.4,340.358]
t_medProd_seq=[1.51591,0.973464,1.77754,1.63987]
t_total_seq=[50.0249,87.6118,316.402,354.212]

t_ocioso_par_3_3=[58.9774,146.47,281.832,365.421]
t_medProd_par_3_3=[0.684618,0.553316,0.534241,0.566174]
t_total_par_3_3=[22.5924,49.7984,95.0949,122.294]

t_ocioso_dis_1=[60.2152,168.18,333.796,454.519]
t_ocioso_dis_2=[32.0191,90.0839,202.565,269.897]
t_ocioso_dis_3=[23.04,60.1019,159.073,154.469]
t_ocioso_dis_4=[16.9147,49.6238,105.018,124.932]
t_ocioso_dis_5=[11.978,43.2922,77.383,109.053]
t_ocioso_dis_6=[11.0729,37.0383,79.1058,99.3294]
t_ocioso_dis_7=[12.0187,31.753,66.6282,82.9702]

t_medProd_dis_1=[1.09389,1.02553,1.02753,1.10602]
t_medProd_dis_2=[0.375824,0.351161,0.363493,0.393484]
t_medProd_dis_3=[0.24852,0.217793,0.242519,0.209929]
t_medProd_dis_4=[0.172867,0.172247,0.158582,0.151773]
t_medProd_dis_5=[0.102936,0.142198,0.126536,0.115807]
t_medProd_dis_6=[0.115057,0.116644,0.108899,0.110074]
t_medProd_dis_7=[0.112312,0.0887869,0.0827885,0.0850912]

t_total_dis_1=[36.0984,92.2978,182.9,238.899]
t_total_dis_2=[12.4022,31.6045,64.7017,84.9926]
t_total_dis_3=[8.20118,19.6013,43.896,46.6043]
t_total_dis_4=[5.70461,15.5022,28.7034,34.3008]
t_total_dis_5=[3.39688,12.7978,22.903,26.4039]
t_total_dis_6=[3.79689,10.498,19.6018,25.0968]
t_total_dis_7=[3.70629,7.99082,14.9019,19.4008]
```

Com esses vetores é possível fazer uma análise visual do desempenho. A primeira análise feita é a contribuição em desempenho que cada máquina pode dar. No cluster tinha-se disponível 7 máquinas

```
num_maq = [1,2,3,4,5,6,7]
```

```
t_ocioso_dis_1_por_maquina =[]
t_ocioso_dis_2_por_maquina =[]
t_ocioso_dis_3_por_maquina =[]
t_ocioso_dis_4_por_maquina =[]
t_ocioso_dis_5_por_maquina =[]
t_ocioso_dis_6_por_maquina =[]
t_ocioso_dis_7_por_maquina =[]
for n_maq in range (0, 4):
    t_ocioso_dis_1_por_maquina.append(t_ocioso_seq[n_maq]/t_ocioso_dis_1[n_maq]/1)
    t_ocioso_dis_2_por_maquina.append(t_ocioso_seq[n_maq]/t_ocioso_dis_2[n_maq]/2)
    t_ocioso_dis_3_por_maquina.append(t_ocioso_seq[n_maq]/t_ocioso_dis_3[n_maq]/3)
    t_ocioso_dis_4_por_maquina.append(t_ocioso_seq[n_maq]/t_ocioso_dis_4[n_maq]/4)
    t_ocioso_dis_5_por_maquina.append(t_ocioso_seq[n_maq]/t_ocioso_dis_5[n_maq]/5)
    t_ocioso_dis_6_por_maquina.append(t_ocioso_seq[n_maq]/t_ocioso_dis_6[n_maq]/6)
    t_ocioso_dis_7_por_maquina.append(t_ocioso_seq[n_maq]/t_ocioso_dis_7[n_maq]/7)
```

```
import pandas as pd
import numpy as np

data = [t_ocioso_dis_1_por_maquina, t_ocioso_dis_2_por_maquina,t_ocioso_dis_3_por_maquina,t_ocioso_dis_4_por_maquina,t
df = pd.DataFrame(data, columns=["Categoria c/ 33 prod", "Categoria c/ 90 prod", "Categoria c/ 178 prod", "Categoria c
df.index = num_maq
df.index.name = 'Número de máquinas'
print('\n')
print("                                GANHO EM TEMPO OCIOSO POR MAQUINA")
print('\n')
print(df)
```

```
                                GANHO EM TEMPO OCIOSO POR MAQUINA


                    Categoria c/ 33 prod  Categoria c/ 90 prod  \
Número de máquinas
1                               0.791597              0.484926
2                               0.744340              0.452660
3                               0.689615              0.452314
4                               0.704509              0.410865
5                               0.795896              0.376764
6                               0.717460              0.366984
7                               0.566572              0.366916

                    Categoria c/ 178 prod  Categoria c/ 216 prod
Número de máquinas
1                                0.914930               0.748831
2                                0.753832               0.630533
3                                0.639958               0.734469
4                                0.727018               0.681087
5                                0.789321               0.624207
6                                0.643442               0.571093
7                                0.654806               0.586025
```

Com os dado a cima é possível perceber que a contribuição por máquina para o tempo ocioso é muito pequena

```
t_medProd_dis_1_por_maquina =[]
t_medProd_dis_2_por_maquina =[]
t_medProd_dis_3_por_maquina =[]
t_medProd_dis_4_por_maquina =[]
t_medProd_dis_5_por_maquina =[]
t_medProd_dis_6_por_maquina =[]
t_medProd_dis_7_por_maquina =[]
for n_maq in range (0, 4):
    t_medProd_dis_1_por_maquina.append(t_medProd_seq[n_maq]/t_medProd_dis_1[n_maq]/1)
    t_medProd_dis_2_por_maquina.append(t_medProd_seq[n_maq]/t_medProd_dis_2[n_maq]/2)
    t_medProd_dis_3_por_maquina.append(t_medProd_seq[n_maq]/t_medProd_dis_3[n_maq]/3)
    t_medProd_dis_4_por_maquina.append(t_medProd_seq[n_maq]/t_medProd_dis_4[n_maq]/4)
    t_medProd_dis_5_por_maquina.append(t_medProd_seq[n_maq]/t_medProd_dis_5[n_maq]/5)
    t_medProd_dis_6_por_maquina.append(t_medProd_seq[n_maq]/t_medProd_dis_6[n_maq]/6)
    t_medProd_dis_7_por_maquina.append(t_medProd_seq[n_maq]/t_medProd_dis_7[n_maq]/7)
```

```
data = [t_medProd_dis_1_por_maquina, t_medProd_dis_2_por_maquina,t_medProd_dis_3_por_maquina,t_medProd_dis_4_por_maqui
df = pd.DataFrame(data, columns=["Categoria c/ 33 prod", "Categoria c/ 90 prod", "Categoria c/ 178 prod", "Categoria c
df.index = num_maq
df.index.name = 'Número de máquinas'
print('\n')
print("                                GANHO EM TEMPO MEDIO POR PRODUTO POR MAQUINA")
print('\n')
print(df)
```

```
                                GANHO EM TEMPO MEDIO POR PRODUTO POR MAQUINA


                    Categoria c/ 33 prod  Categoria c/ 90 prod  \
Número de máquinas
1                               1.385797              0.949230
2                               2.016782              1.386065
3                               2.033250              1.489892
4                               2.192307              1.412890
5                               2.945345              1.369167
6                               2.195883              1.390933
7                               1.928187              1.566293

                    Categoria c/ 178 prod  Categoria c/ 216 prod
Número de máquinas
1                                1.729915               1.482677
2                                2.445081               2.083782
3                                2.443163               2.603849
4                                2.802241               2.701189
5                                2.809540               2.832074
6                                2.720472               2.482981
7                                3.067265               2.753130
```

```
t_total_dis_1_por_maquina =[]
t_total_dis_2_por_maquina =[]
t_total_dis_3_por_maquina =[]
t_total_dis_4_por_maquina =[]
t_total_dis_5_por_maquina =[]
t_total_dis_6_por_maquina =[]
t_total_dis_7_por_maquina =[]
for n_maq in range (0, 4):
    t_total_dis_1_por_maquina.append(t_total_seq[n_maq]/t_total_dis_1[n_maq]/1)
    t_total_dis_2_por_maquina.append(t_total_seq[n_maq]/t_total_dis_2[n_maq]/2)
    t_total_dis_3_por_maquina.append(t_total_seq[n_maq]/t_total_dis_3[n_maq]/3)
    t_total_dis_4_por_maquina.append(t_total_seq[n_maq]/t_total_dis_4[n_maq]/4)
    t_total_dis_5_por_maquina.append(t_total_seq[n_maq]/t_total_dis_5[n_maq]/5)
    t_total_dis_6_por_maquina.append(t_total_seq[n_maq]/t_total_dis_6[n_maq]/6)
    t_total_dis_7_por_maquina.append(t_total_seq[n_maq]/t_total_dis_7[n_maq]/7)
```

```
data = [t_total_dis_1_por_maquina, t_total_dis_2_por_maquina,t_total_dis_3_por_maquina,t_total_dis_4_por_maquina,t_tot
df = pd.DataFrame(data, columns=["Categoria c/ 33 prod", "Categoria c/ 90 prod", "Categoria c/ 178 prod", "Categoria c
df.index = num_maq
df.index.name = 'Número de máquinas'
print('\n')
print("                                GANHO EM TEMPO TOTAL POR MAQUINA")
print('\n')
print(df)
```

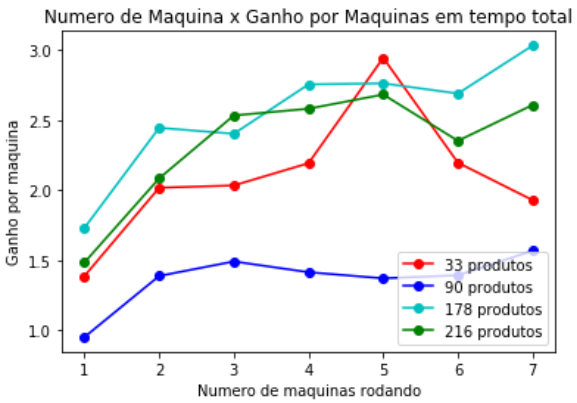
```
                                GANHO EM TEMPO TOTAL POR MAQUINA


                    Categoria c/ 33 prod  Categoria c/ 90 prod  \
Número de máquinas
1                               1.385793              0.949230
2                               2.016775              1.386065
3                               2.033240              1.489898
4                               2.192301              1.412893
```

```
5                              2.945344             1.369170
6                              2.195872             1.390928
7                              1.928185             1.566294
```

| Número de máquinas | Categoria c/ 178 prod | Categoria c/ 216 prod |
|---|---|---|
| 1 | 1.729918 | 1.482685 |
| 2 | 2.445083 | 2.083781 |
| 3 | 2.402664 | 2.533472 |
| 4 | 2.755789 | 2.581660 |
| 5 | 2.762974 | 2.683028 |
| 6 | 2.690246 | 2.352305 |
| 7 | 3.033189 | 2.608228 |

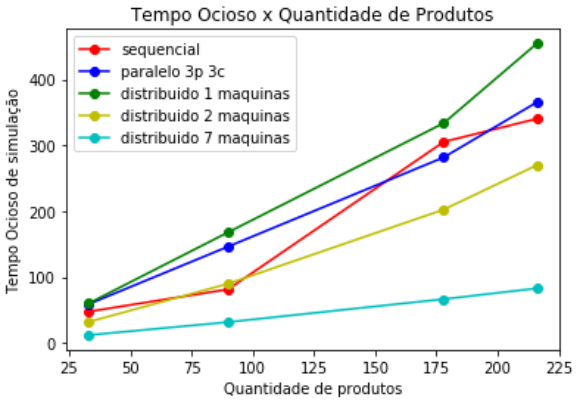
```
tempo_total_por_maquina_url1 =[]
tempo_total_por_maquina_url2 =[]
tempo_total_por_maquina_url3 =[]
tempo_total_por_maquina_url4 =[]
for d in data:
    tempo_total_por_maquina_url1.append(d[0])
    tempo_total_por_maquina_url2.append(d[1])
    tempo_total_por_maquina_url3.append(d[2])
    tempo_total_por_maquina_url4.append(d[3])
```

```
url1, = plt.plot(num_maq, tempo_total_por_maquina_url1, 'ro-', label='33 produtos')
url2, = plt.plot(num_maq, tempo_total_por_maquina_url2, 'bo-', label='90 produtos')
url3, = plt.plot(num_maq, tempo_total_por_maquina_url3, 'co-', label= '178 produtos')
url4, = plt.plot(num_maq, tempo_total_por_maquina_url4, 'go-', label= '216 produtos')
plt.xlabel('Numero de maquinas rodando')
plt.ylabel('Ganho por maquina')
plt.title("Numero de Maquina x Ganho por Maquinas em tempo total")
plt.legend(handles=[url1, url2, url3, url4])
plt.show()
```



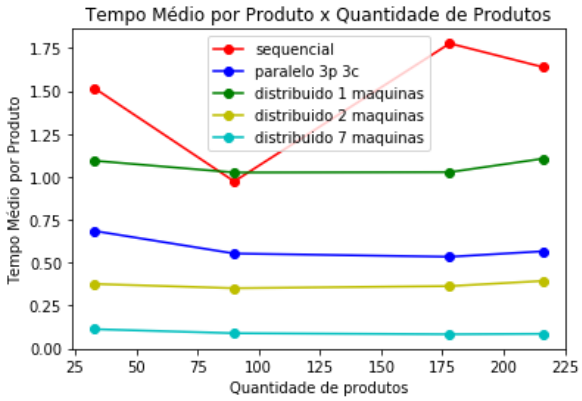

Analisando as informações a cima é possível perceber que quanto mais máquinas, maior o ganho por maquinas, tanto em tempo total quanto em tempo médio por produto. A quantidade de produto por categoria não parece interferir nos ganhos por máquina.

```
seq, = plt.plot(t_num_prod, t_ocioso_seq, 'ro-', label='sequencial')
par33, = plt.plot(t_num_prod, t_ocioso_par_3_3, 'bo-', label='paralelo 3p 3c')
dis1, = plt.plot(t_num_prod, t_ocioso_dis_1, 'go-', label= 'distribuido 1 maquinas')
dis2, = plt.plot(t_num_prod, t_ocioso_dis_2, 'yo-', label= 'distribuido 2 maquinas')
dis7, = plt.plot(t_num_prod, t_ocioso_dis_7, 'co-', label= 'distribuido 7 maquinas')
plt.xlabel('Quantidade de produtos')
plt.ylabel('Tempo Ocioso de simulação')
plt.title("Tempo Ocioso x Quantidade de Produtos")
plt.legend(handles=[seq, par33, dis1, dis2, dis7])
plt.show()
```



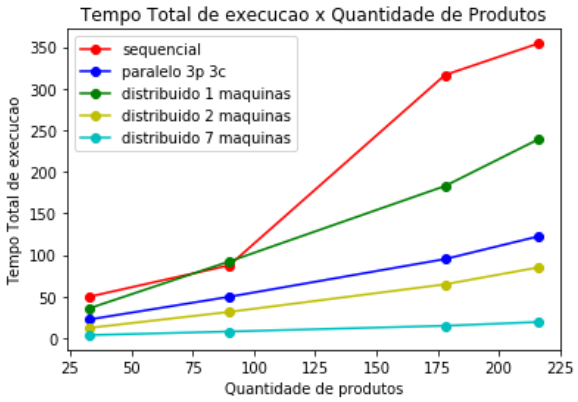

A partir do gráfico a cima é possível perceber que o tempo ocioso é pouco afetado pelos métodos de concorrência, ele se comporta do mesmo modo no três tipos de crawler

```
seq, = plt.plot(t_num_prod, t_medProd_seq, 'ro-', label='sequencial')
par33, = plt.plot(t_num_prod, t_medProd_par_3_3, 'bo-', label='paralelo 3p 3c')
dis1, = plt.plot(t_num_prod, t_medProd_dis_1, 'go-', label= 'distribuido 1 maquinas')
dis2, = plt.plot(t_num_prod, t_medProd_dis_2, 'yo-', label= 'distribuido 2 maquinas')
dis7, = plt.plot(t_num_prod, t_medProd_dis_7, 'co-', label= 'distribuido 7 maquinas')
plt.xlabel('Quantidade de produtos')
plt.xlabel('Quantidade de produtos')
plt.ylabel('Tempo Médio por Produto')
plt.title("Tempo Médio por Produto x Quantidade de Produtos")
plt.legend(handles=[seq, par33, dis1, dis2, dis7])
plt.show()
```




Pela análises a cima é possível perceber que normalmente o tempo médio por produto permanece estável independente do número total de produtos sendo processados.

```
seq, = plt.plot(t_num_prod, t_total_seq, 'ro-', label='sequencial')
par33, = plt.plot(t_num_prod, t_total_par_3_3, 'bo-', label='paralelo 3p 3c')
dis1, = plt.plot(t_num_prod, t_total_dis_1, 'go-', label= 'distribuido 1 maquinas')
dis2, = plt.plot(t_num_prod, t_total_dis_2, 'yo-', label= 'distribuido 2 maquinas')
dis7, = plt.plot(t_num_prod, t_total_dis_7, 'co-', label= 'distribuido 7 maquinas')
plt.xlabel('Quantidade de produtos')
plt.ylabel('Tempo Total de execucao')
plt.title("Tempo Total de execucao x Quantidade de Produtos")
plt.legend(handles=[seq, par33, dis1, dis2, dis7])
plt.show()
```




```
seq_par33_total_media=0
seq_dis_1_total_media=0
seq_dis_7_total_media=0
dis_1_par33_total_media=0
par33_dis_7_total_media=0
count =0

for i in range (0, len(t_total_seq)):
    for i in range(0, len(t_total_par_3_3)):
        seq_par33_total_media = seq_par33_total_media + (t_total_seq[i]/t_total_par_3_3[i])
        seq_dis_1_total_media = seq_dis_1_total_media + (t_total_seq[i]/t_total_dis_1[i])
        seq_dis_7_total_media = seq_dis_7_total_media + (t_total_seq[i]/t_total_dis_7[i])
        dis_1_par33_total_media = dis_1_par33_total_media + (t_total_dis_1[i]/t_total_par_3_3[i])
        par33_dis_7_total_media = par33_dis_7_total_media + (t_total_par_3_3[i]/t_total_dis_7[i])
        count+=1

print("Média de ganho tempo total de paralelo (3 prod 3 cons) para seq= " + str(seq_par33_total_media/count))
print("Média de ganho tempo total de distribuido (1 maquina) para seq= " + str(seq_dis_1_total_media/count))
print("Média de ganho tempo total de distribuido (7 maquinas) para seq= " + str(seq_dis_7_total_media/count))
print("Média de ganho tempo total de paralelo (3 prod 3 cons) para distribuido (1 maquina)= " + str(dis_1_par33_total_
print("Média de ganho tempo total de distribuido (7 maquinas) para paralelo (3 prod 3 cons)= " + str(par33_dis_7_total
```

```
Média de ganho tempo total de paralelo (3 prod 3 cons) para seq= 2.549296557890009
Média de ganho tempo total de distribuido (1 maquina) para seq= 1.3869063551633605
Média de ganho tempo total de distribuido (7 maquinas) para seq= 15.987819448884398
Média de ganho tempo total de paralelo (3 prod 3 cons) para distribuido (1 maquina)= 1.832015868915663
Média de ganho tempo total de distribuido (7 maquinas) para paralelo (3 prod 3 cons)= 6.253147838528595
```

Com as análises acima, percebe-se que tanto o crawler paralelo quanto o distribuido possuem ganhos significativos em relação ao sequencial. O paralelo é cerca de 2.54 vezes mais rápido que o sequencial enquanto o distribuido com 1 máquina (4 processos) é 1.38 vezes mais rápido que o sequencial. Tendo isso em vista, o paralelo com 3 threads produtoras e 3 threads consumidoras é mais rápido que o distribuido com 4 processos cerca de 1,83 vezes. No entanto, aumentanto apenas 1 máquina (total de 8 processos) o crawler distribuido se torna mais rápido que o crawler paralelo, como pode ser visto no gráfico a cima. Ao comparar o crawler distribuido mais eficiente disponivel (7 maquinas, 28 processos) com o sequencial, tem-se um ganho de quase 16 vezes, e ao compará-lo com o paralaleo 3p 3c esse distribuído é mais rápido que ele 6.25 vezes.