Universidade Federal do Espírito Santo

Processamento Paralelo e Distribuído

**Arquitetura Mestre Escravo com Java RMI**

Ataque de Dicionário em Mensagem Criptografada

Allan Araujo Silva

César Henrique Bernabé

Vitória, 27 de Junho de 2016

1. **Introdução**

Este trabalho tem por objetivo pôr em prática as técnicas de programação paralela e distribuída estudadas em sala de aula, destacando os recursos oferecidos pela linguagem Java, mais especificamente do pacote RMI, cuja principal finalidade é permitir que objetos presentes em máquinas diferentes sejam invocados local ou remotamente.

Para isso, foi implementada a arquitetura Mestre-Escravo, realizando a busca por uma palavra-chave em um arquivo de palavras criptografado (ataque de dicionário). O procedimento de ataque de dicionário e detalhes de implementação são descritos abaixo.

* 1. **Ataque de Dicionário**

Em criptografia e segurança de computadores, define-se ataque de dicionário uma técnica que consiste em testar inúmeras palavras-chaves que possam decodificar uma senha ou uma chave criptografada, para anular um criptograma (texto que obedece um tipo de lógica para decodificação) ou algum outro mecanismo de autenticação de mensagens, esse processo é conhecido com ataque de força bruta.

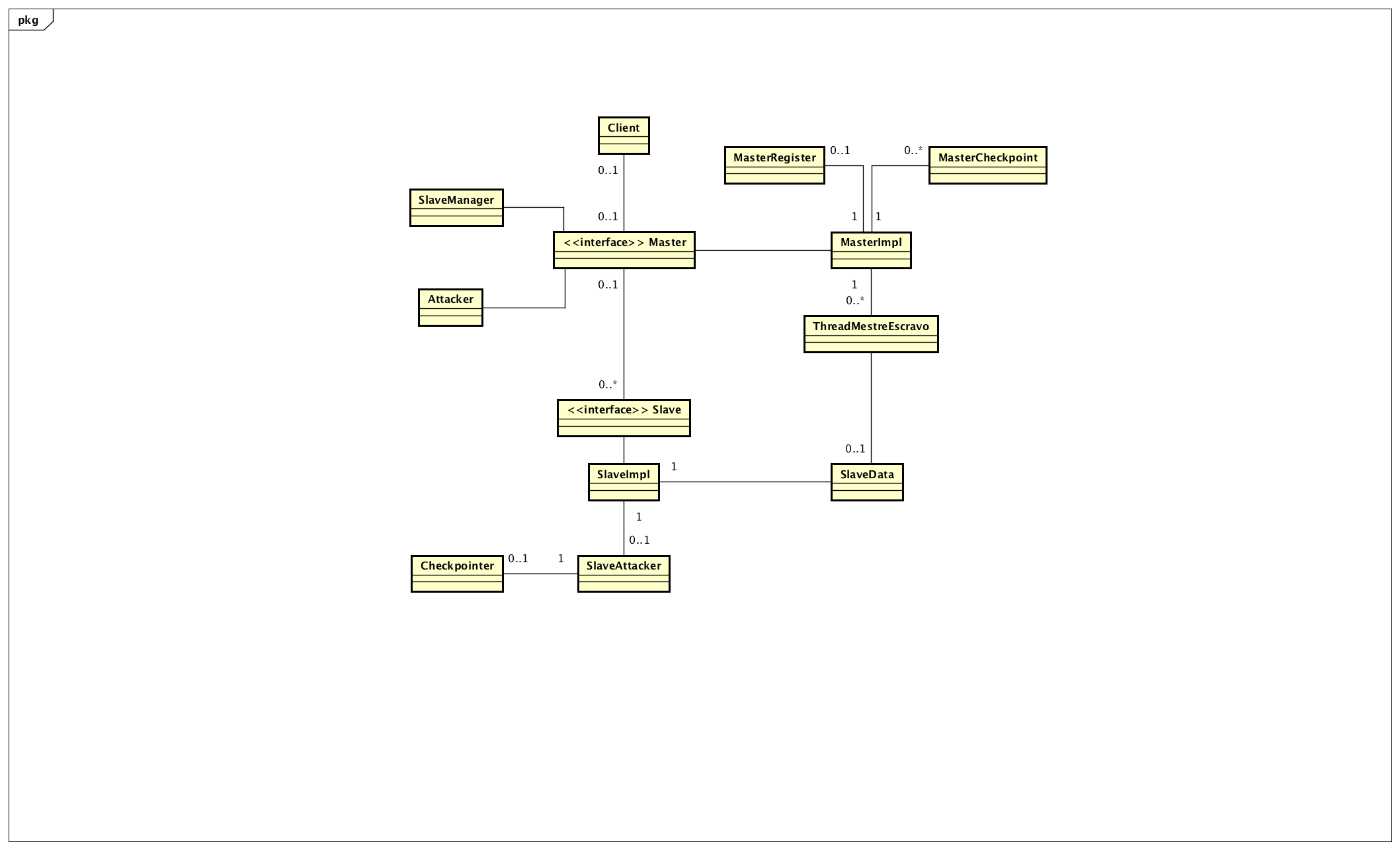
* 1. **Arquitetura Utilizada**

Uma arquitetura Mestre-Escravo é formada basicamente por três programas principais: Cliente, Mestre e Escravo. O cliente faz a requisição do serviço ao mestre, que por sua vez é responsável por receber a requisição e dividir o trabalho não-paralelizável entre o número de escravos disponíveis. Por final, o escravo é o programa que realiza o trabalho paralelizável. Como o objetivo é aplicar conceitos de distribuição e paralelismo, sabemos que todos estes programas devem estar rodando em diferentes computadores. Para que isto seja possível, é necessário um serviço de nomes para que os objetos possam se comunicar remotamente. O serviço de nomes utilizado foi o Java RMI Registry, que atua como mediador entre todos os componentes da arquitetura. Ele funciona como um mural público, onde os objetos remotos podem registrar suas referências remotas e procurar por referências de outros objetos. Referência remota é o meio como um objeto pode se comunicar com outro, mesmo eles não estando presentes no mesmo computador.

Os escravos existentes registram-se com o mestre, através de um método disponibilizado pelo mesmo, com isso, o mestre delega para cada um dos escravos envolvidos no processo uma parcela do dicionário de palavra e a mensagem criptografada. Então, os processos escravos realizam os ataques à procura de uma possível palavra que possa decodificar a mensagem

**2. Código Fonte**

O código fonte é apresentado em linguagem Java, composto por classes concretas e interfaces padronizadas que foram disponibilizadas, como apresentado a seguir:



**2.1 Implementação MasterImp**

Os principais métodos da classe são listados e explicados a seguir.

Um a*ttack* consiste em criar threads com escravos à procura de uma palavra chave em um arquivo criptografado.

//partes de código

Quando um escravo falha, ele/sua referência é enviado para a lista de escravos falhos. Nesse caso, é feito o *rearrangeattack,* alocando um novo escravo com esse trabalho em uma nova thread e a remoção daquele escravo da lista de escravos falhos.

//parte de código

Além desses métodos, o mestre possui métodos de adição e remoção de escravos de seu mapa de escravos, bem como métodos de *foundguess* e *checkpoint* que são herdados da interface Attacker.

**2.2 Implementação SlaveImpl**

Assim como o mestre, os escravos realizam um ataque ao dicionário, declarado como *startSubAttack*, onde cada escravo recebe uma parcela do arquivo para fazer a procura de chaves candidatas.

Escravos se registram pelo mestre pelo método de *registerSlave*, onde ele busca pela referência do mestre e faz o registro. Caso ele não o ache, faz uma procura pela referência do mestre no registro de nomes.

**2.5 Modo de Execução**

Considerando diretório como raiz, foi usado a seguinte sequência de comandos para execução, via terminal:

acesso às maquinas

*ssh<nome\_maquina>*

registro de nomes

*rmiregistry &*

*executar mestre*

*java -cp . –Djava.rmi.server.hostname=<IPMestre> br.inf.ufes.pp2016\_01.MasterImpl*

*executar escravo(s)*

*java -cp . -Djava.rmi.server.hostname=<IPEscravo> br.inf.ufes.pp2016\_01 <nome\_escravo> <IPMestre>*

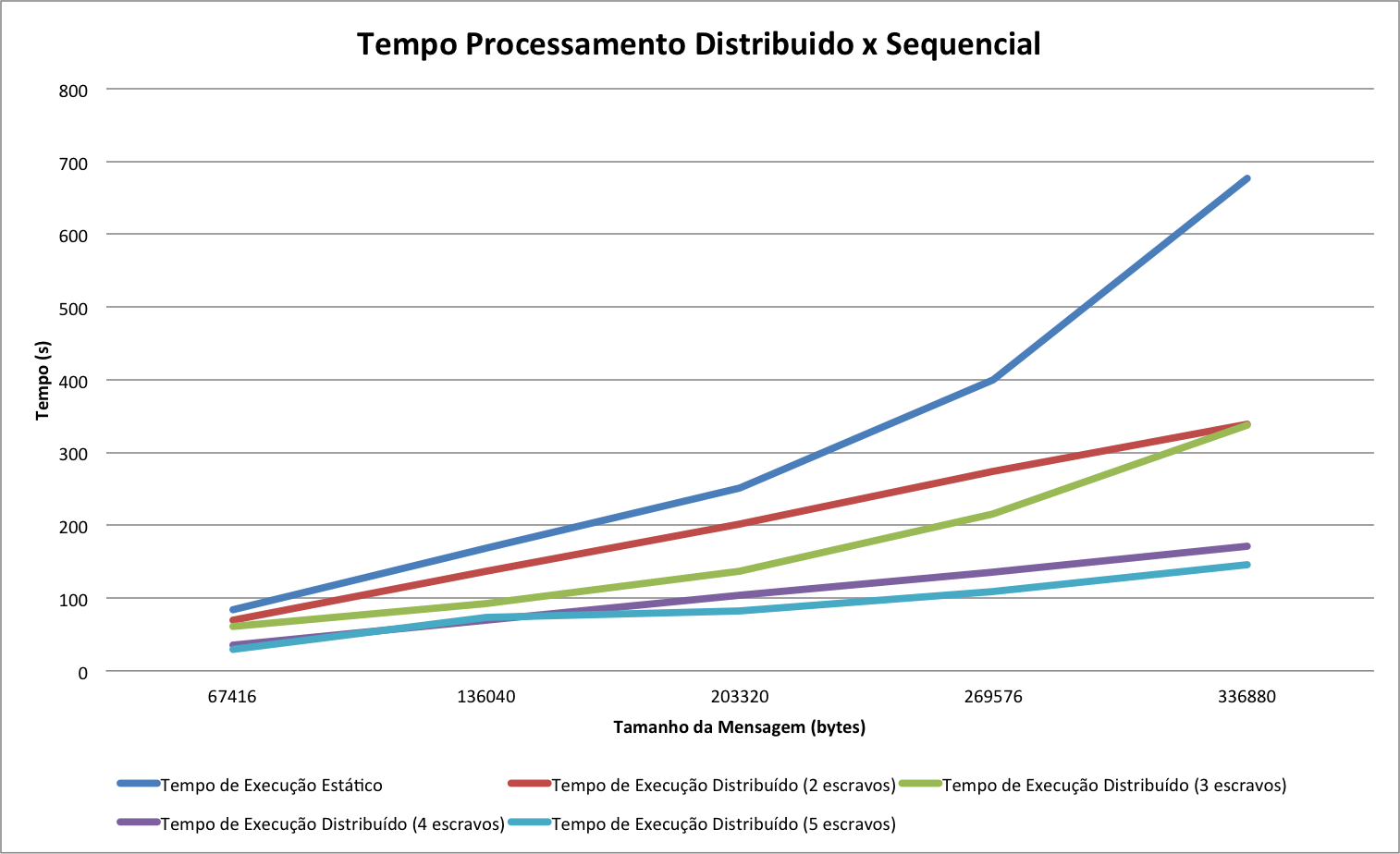
*executar cliente*

*java -cp . -Djava.rmi.hostname=<IPCliente> br.inf.ufes.pp2016\_01.Client <chave\_candidata> <mensagem\_criptografada> <IPMestre>*

**3 Testes**

Foram realizados testes para comparar o processamento de modo sequencial com modo distribuído, além de avaliar desempenho, eficiência e robustez do programa, bem como à tratamentos de falhas.

**3.1 Tempo Sequencial x Tempo Paralelo Distribuido**

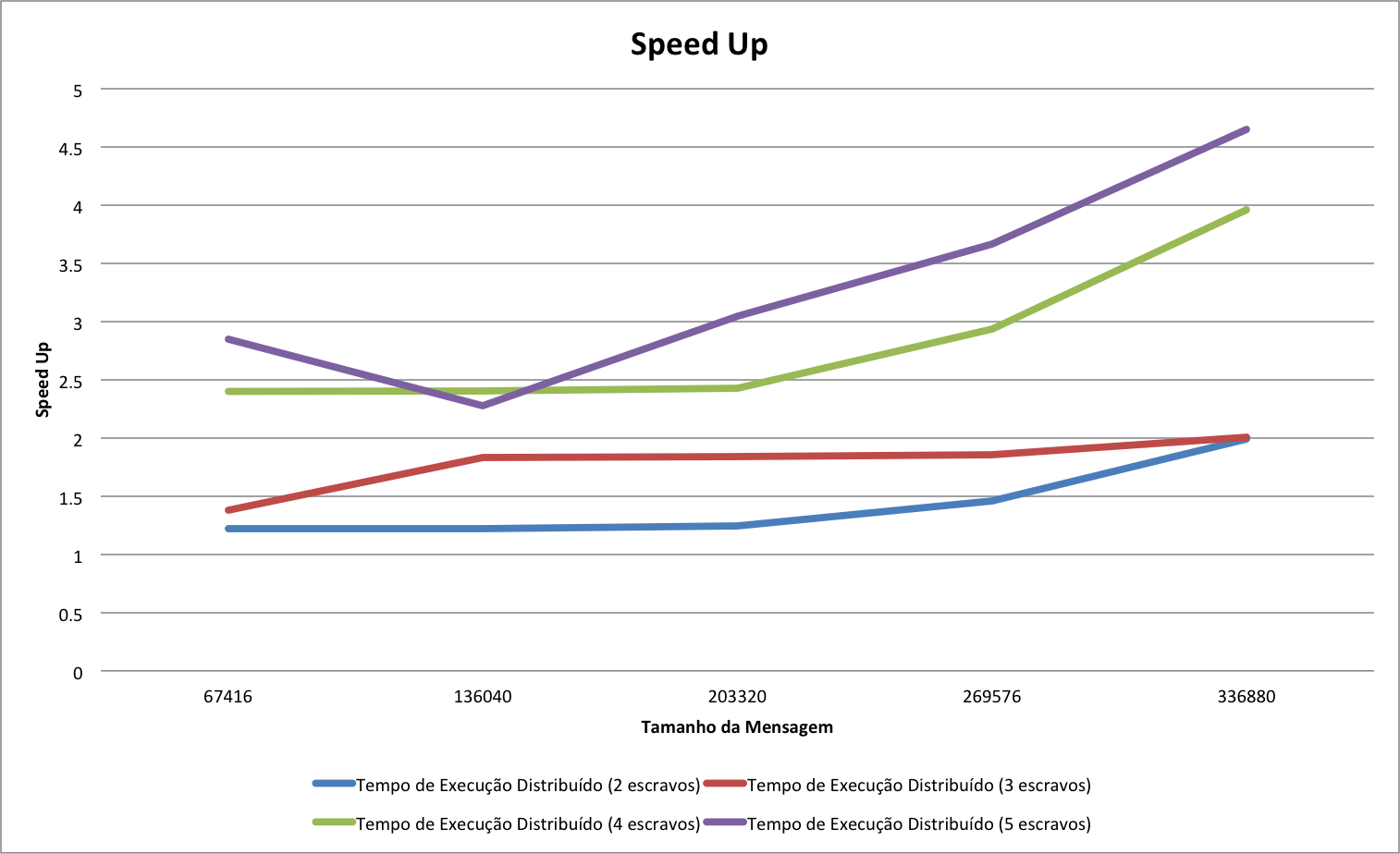
****

//gráfico – temo sequencial vai ficar maior que o de distribuido

Analisando o gráfico da figura#, é fácil perceber que o tempo de execução de forma sequencial é muito maior que os outros tempos.

Como o próprio cliente é quem irá fazer a busca em todo o dicionário, o que não é uma tarefa barata, o trabalho é muito maior se comparado ao processamento de forma distribuída, onde cada escravo fica com apenas uma parcela do arquivo para fazer a busca, além de serem executados de forma paralela.

**3.2 Análise de Speed Up e Eficiência**

****

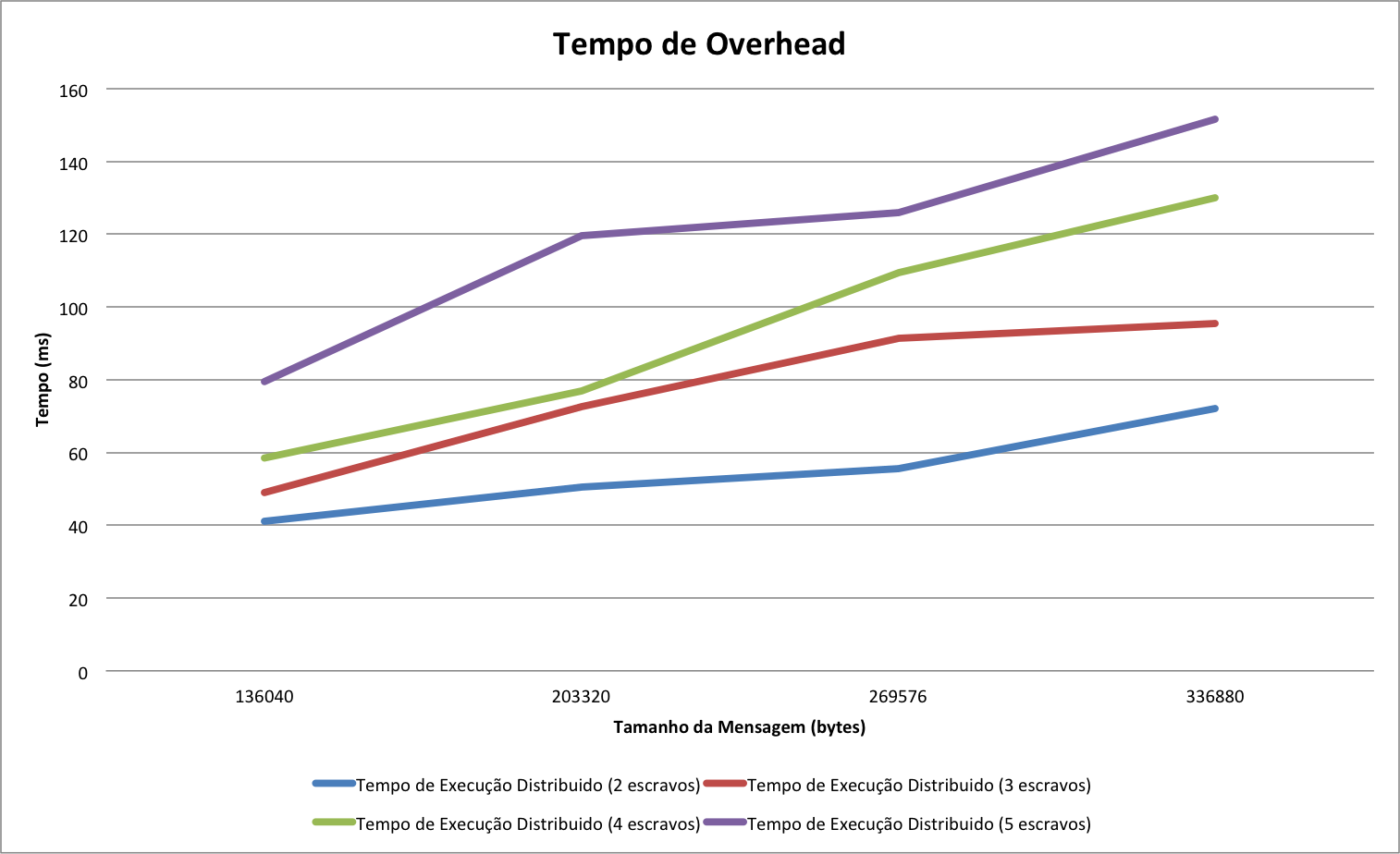
**//**gráfico de speedup

**//**para cada caso(sequencial, 2esc, 3esc...) speedup deve tender para um valor constante

Como mostrado no gráfico da figura#, o speedup, para cada caso, está bem próximo ao número de escravos usados. Existe interferência de overhead no tempo de execução, porém não influencia na análise. **//**mostrar no gráfico os valores e escrever algo aqui.

A conclusão tirada é que, par um problema de ataque de dicionário, a abordagem distribuída é melhor em relação a sequencial.

**3.3 Testes de Overhead**

****

//já a eficiência tende para um mesmo valor para todos os casos de escravos

**4. Conclusão**

A partir dos testes executados e análises feitas, conclui-se que, para um problema como ataque de dicionário, a abordagem paralela distribuída é muito mais eficiente em relação a sequencial. O tempo gasto na busca por chaves candidatas no dicionário e a comparação de cada palavra eleva o tempo de execução. Podemos ressaltar também o fato das execuções dos escravos não são dependentes, o que fortalece o uso de processamento distribuído.

**5. Referências**