Universidade Federal de Viçosa
Sistemas Distribuídos e Paralelos

TRABALHO PRÁTICO DE SISTEMAS DISTRIBUÍDOS

Daniel Kneipp de Sá Veira[1]

Resumo. Algoritmos de otimização são baseados em heurísticas, e por conta disso geram resultados distintos quando executados várias vezes. Este trabalho tem propõe um sistema distribuído que aciona a execução de várias softwares baseados em heurísticas e utiliza um protocolo de eleição de líderes para a obtenção da melhor solução.

Para se verificar a eficácia da execução do sistema utilizou-se para testes 4 processos em uma mesma máquina conectados por interfaces virtuais. O problema tratado foi de minimização da função conhecida como Rastrigin [Pro15] modificada por [Hed15]. As métricas utilizadas para a avaliação do sistema foram o próprio resultado que o mesmo obteve na tentativa de achar a melhor solução para a função objetivo proposta em comparação com a melhor solução possível e o tempo que o sistema demora para achar uma solução específica ou melhor.

Keywords: Sistemas Distribuídos, Protocolo de Eleição de Líderes

Sumário

25 de abril de 2015.

[1] Matrícula 1257

1. Introdução

Problemas de otimização são conhecidos por serem muito complexos e exaustivos de serem computados, sendo inviável a obtenção da solução ótima por meio de algoritmos de força bruta, por isso se faz necessário o uso de métodos que se obtenha uma solução satisfatória para este tipos de problema, e é ai que entram as heurísticas.

Meta-heurísticas servem como modelos para o desenvolvimento de heurísticas para se obter uma boa solução nos mais diversos problemas [Net15].

Um ponto importante sobre heurísticas é que como elas tem uma natureza probabilística, elas retornam soluções distintas com diferentes execuções, sendo que obviamente apenas uma é a melhor, mas como estas execuções são totalmente independentes entre si, elas podem ser feita sem máquinas distintas acionadas por um sistema distribuído.

2. Distributed Optimization System

Como já foi dito anteriormente, o *Distributed Optimization System* (DOS) é um sistema distribuído que se propõe a acionar a execução de várias heurísticas de otimização (que não necessariamente precisam ser as mesmas) e por fim apresentar o melhor resultado.

O sistema foi desenvolvido utilizando a linguagem de programação JavaScript com a API de *Sockets* oferecida pela plataforma Node.js [Joy15].

2.1. Modelo de Arquitetura

Para o desenvolvimento do sistema adotou-se um modelo onde cada máquina é um servidor obrigatoriamente e pode ser instanciado um cliente para cada servidor que se deseja conectar (configurando uma rede *peer-to-peer*) e a topologia da rede a de uma rede completa.

Essa configuração de 1 servidor e 1 até N clientes foi encapsulado em uma estruturada chamada de nó. Deve haver ao menos um cliente conectado à algum servidor, caso contrário não se poderá propagar a solução (já que é o cliente quem retorna a solução para o servidor conectado à ele).

2.2. Modelo de Interação

É determinado que o cliente é quem recebe a requisição de inicialização da heurística e propagar a solução determinada na estrutura de nó pelo protocolo de eleição de líderes. Já que se tem um cliente pra um servidor conectado, mas pode-se ter vários clientes (pelo fato de se poder ter vários servidores conectados) mais de uma requisição de execução poderia chegar (mesmo enquanto a heurística já estivesse sendo executada), por conta disso é a estrutura de nó que efetivamente inicializa a execução.

Já o servidor fica responsável por receber os resultados enviados pelos clientes conectados à ele, repassar esses resultados para a estrutura de nó para que o mesmo possa decidir qual solução propagar para os outros servidores utilizando os clientes. O servidor também faz a requisição de inicialização de execução aos clientes conectados à ele.

A estrutura de nó é quem coordena o servidor e os clientes em uma mesma máquina, propagando requisições de execução recebidas pelos clientes para os clientes conectados ao servidor. Também é ela quem de fato inicializa a execução da heurística e impede a execução da mesma mais de uma vez definindo estados para a máquina. Neste caso os estados que uma máquina pode assumir são:

- `WAITING`: Especifica que a máquina está ociosa, esperando alguma requisição de execução ou o recebimento de um *id* de outra máquina para que ela se prontifique a repassar o menor (neste caso) *id* (entre o dela e o recebido).
- `RESULT_RECEIVED`: Determina que a máquina já recebeu o *id* de outra, o que implica em que ela deve compara este *id* recebido com o seu e repassar o menor. Caso a máquina ainda esteja processando a heurística e não obteve seu *id*, e repasse do *id* correto será feito imediatamente após o processamento acabar.

Este é um modelo assíncrono onde não se sabe quanto tempo vai demorar pra se ter as soluções de todos os clientes conectados ao servidor para poder propagar a escolhida aos servidores.

A figura 1 mostra uma possível implementação deste sistema utilizando a topologia de anel onde as soluções são propagadas de forma unidirecional (sentido anti-horário na perspectiva da figura).

2.3. Modelo de Falha

O modelo de falhas é altamente dependente da topologia da rede, que é determinada pelo algoritmo de eleição de líderes utilizado, então mesmo que o sistema distribuído suporte a conexão de vários servidores, não quer dizer que ele necessariamente vá ter. A figura 1 mostra apenas um servidor requisitando a solução para outra máquina (por meio do módulo cliente).

Como no caso está se utilizando um algoritmo que requer uma rede completa, é possível desenvolver vários artifícios que permitam o sistema se manter em funcionamento em condições adversas.

DOS suporta a queda de uma máquina no meio da execução da heurística, no meio da execução do algoritmo de eleição de líderes, o sistema continua em funcionamento mesmo com apenas uma máquina na rede.

Segue os casos de falha suportados:

- *Queda de alguma máquina no meio da execução do algoritmo de eleição de líderes*: Sendo m_2 a máquina que caiu e m_1 a máquina que deveria enviar seu *id* para m_2, m_1 envia o melhor *id* obtido até aquele momento para máquina m_r que inicializou o algoritmo. Caso $m_2 = m_r$, m_1 apresenta o

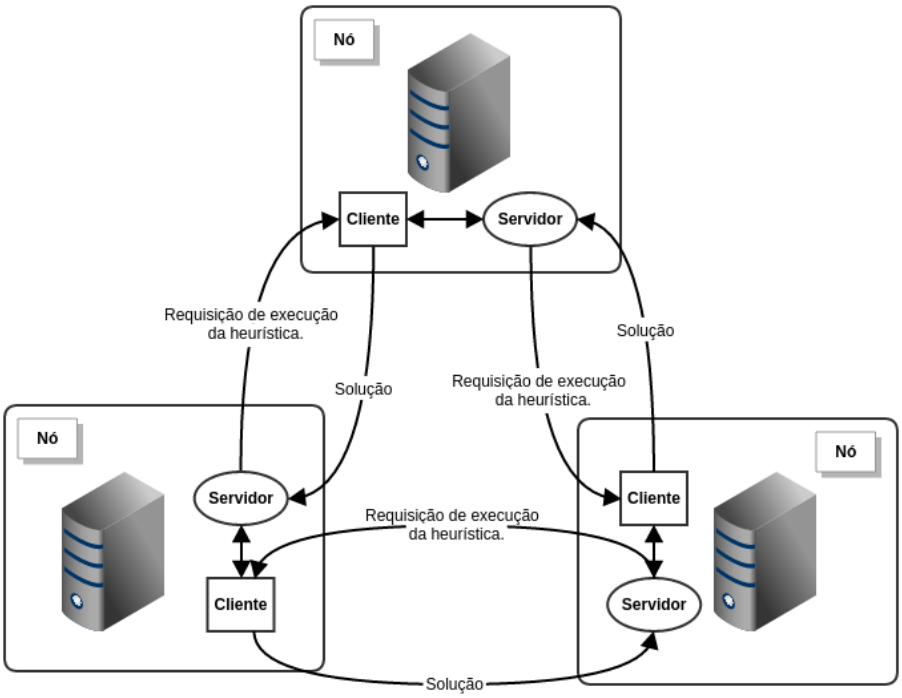

FIGURA 1. Possível implementação do sistema utilizando uma topologia de rede em anel.

melhor *id* como resultado e finaliza o algoritmo. Caso uma máquina m_1 caia antes de conseguir enviar seu *id* para m_2 mas já tenha recebido o *id* de outra máquina, não se apresenta nenhum resultado e o algoritmo de eleição para, mas pode-se obter o resultado por meio de uma nova requisição em outra máquina (os resultados de cada máquina não são perdidos).

- *Queda de alguma máquina no meio da execução da heurística*: A execução nas outras máquinas permanece sem alterações e o sistema continua intacto, podendo requisitar o melhor resultado utilizando as máquinas ainda em funcionamento.

2.4. Modelo de Segurança

Este sistema não possui um modelo de segurança definido.

2.5. Algoritmo de Eleição de Líderes

Como já foi dito anteriormente, este algoritmo exige que se tenha uma rede com topologia completa, ou seja, todas as máquinas conectadas à todas as outras.

Os passos do algoritmo são:

(1) O algoritmo inicia a partir de um gatilho disparado em uma máquina (e.g.: requisição do usuário), esta máquina m_r primeiramente vai gerar um anel virtual dentro da rede completa, definindo pra qual máquina m_{k+1} a máquina m_k deve enviar o *id* (m_r está incluída no anel).
(2) Após isto, m_r envia uma *flag* para a máquina seguinte m_1 no anel para enviar seu *id* para m_2 sem esperar qualquer *id*.
(3) Em seguida começa o ação em cadeia em que m_k envia o melhor *id* entre os ids de m_k e m_{k-1} para m_{k+1}.
(4) Quando $m_k = m_r$, m_r apresenta o melhor resultado entre m_r e m_{k-1} em sua tela.

A utilização de um anel virtual foi inspirada por [Vil+05], em que também se utiliza um subconjunto das conexões existentes na rede para concretizar as eleições de líderes, mas diferente do algoritmo proposto em [Vil+05], este mantém a estrutura definida no começo da execução do algoritmo até o final (caso não ocorra nenhuma falha) com uma comunicação unidirecional.

3. Testes e Resultados

Para a execução dos testes do DOS utilizou-se 4 processos operando em uma mesma máquina (esta com um processador com 4 núcleos físicos), em que estes são conectados por meio de interfaces de rede virtuais.

O algoritmo de otimização utilizado foi um baseado na meta-heurística *Simulated Annealing* (SA), em que este tenta minimizar a função conhecida como Rastrigin [Pro15] modificada por [Hed15]. O código em Python foi obtido a partir de [Hed15].

A equação que deve ser **minimizada** tem seu mínimo igual à 0 e é definida como:

$$\begin{aligned} f(\bar{x}) &= 0.2 + x_1^2 + x_2^2 - 0.1cos(6.0\pi x_1) - 0.1cos(6.0\pi x_2), \\ \bar{x} &= \{x_1, x_2\}, \quad x_i = \{k \in \mathbb{R} : -5 \leq k \leq 5\} \end{aligned} \tag{1}$$

O SA foi configurado com $n = 1000$ iterações totais e $L = 1000$ (que configura o número de interações com um mesmo valor de temperatura), uma temperatura inicial $t_0 = \frac{-1}{ln(0.7)}$ que é atualizada a cada interação de acordo com $t_i := rt_{i-1}$, em que $r = \left(\frac{t_{n-1}}{t_1}\right)^{\frac{1}{n-1}}$ e $t_{n-1} = \frac{-1}{ln(0.001)}$. A probabilidade de se aceitar uma solução pior em uma dada iteração i é calculada com base em $p = e^{\frac{-\Delta f}{\Delta_\mu f t_i}}$, em que Δf é o módulo da diferença entre a solução atual e a melhor solução encontrada, $\Delta_\mu f$ é a média de todos os Δf calculados até a iteração i.

Executou-se 11 vezes a otimização tanto no SA no modo autônomo quanto utilizando o DOS, obtendo valores de tempo de execução e resultado obtido (menor valor da função encontrado). No caso dos valores de tempo de execução para o DOS, levou-se em consideração também o tempo de obtenção do melhor resultado na rede e não somente a execução da heurística em si.

Como mostra a figura 2, os tempos de execução do DOS são mais instáveis do que SA em modo autônomo, não tendo nenhuma vez demorado menos. Isso deve-se ao fato de que DOS com 4 processos é equivalente a 4 SAs em modo autônomo, exigindo mais da máquina que se está usando, e também porque existe o passo da obtenção do melhor resultado entre os processos. DOS obteve tempos de execução com uma média de $t(\text{DOS})_\mu = 17.4981684871s$ com desvio padrão $t(\text{DOS})_\sigma = 2.5090479674s$ enquanto o SA em modo autônomo obteve uma média de $t(\text{SA})_\mu = 11.7332685427s$ com um desvio padrão de $t(\text{SA})_\sigma = 0.115128069478s$.

FIGURA 2. Comparativo entre os tempos de execução do DOS e do *Simulated Annealing* em modo autônomo.

Já a figura 3 mostra a superioridade da qualidade dos resultados obtidos a partir do DOS se comparado com o SA em modo autônomo, alcançando valores de custo bem menores. DOS obteve custos com média $r(\text{DOS})_\mu = 0.00538080908429$ e desvio padrão $r(\text{DOS})_\sigma = 0.00378422466426$ enquanto o SA em modo autônomo obteve soluções com média de custo $r(\text{SA})_\mu = 0.0427484902005$ e desvio padrão $r(\text{SA})_\sigma = 0.042122092619$.

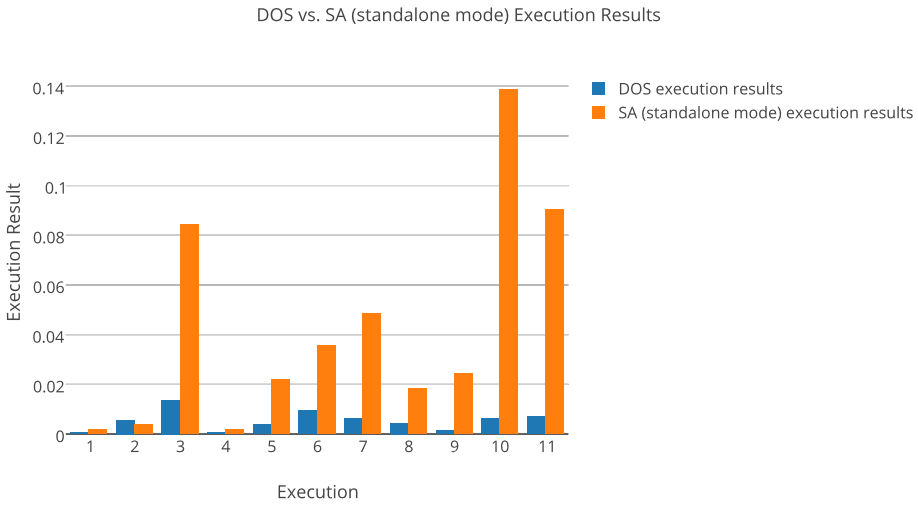

FIGURA 3. Comparativo entre os resultados obtidos na optimização pelo DOS e pelo *Simulated Annealing* em modo autônomo.

4. CONLUSÃO

Como é possível notar pelos resultados, a utilização do DOS proporciona melhores resultados como esperado já que, por conta de uma heurística possuir uma natureza probabilística, é altamente provável ter um resultado diferente a cada execução, e com isso, é mais provável de se ter resultados melhores com várias execuções.

No caso de tempo de execução, DOS tende a ser mais lento e exigir bem mais processamento (no caso de mais de um processo operando em uma mesma máquina), o que faz todo sentido sendo que são executados várias heurísticas em paralelo e não somente uma. Vale salientar que os testes do DOS foram conduzidos em uma mesma máquina utilizando 4 processos, o que significa que os tempos de execução tendem a diminuir no caso de se utilizar várias máquinas.

Com isso, pode-se concluir que a utilização do DOS vale a pena caso o usuário do sistema possua um alto poder de processamento para comportar a carga exigida.

Como oportunidade para trabalhos futuros pode-se destacar a modificação do algoritmo de eleição de líderes para não ser mais exigido que todos os processos se conheçam, aumentando assim a escalabilidade do sistema, mas note que isso pode afetar diretamente a capacidade do sistema de tolerar falhas.

Referências

[Hed15] John D. Hedengren. *Simulated Annealing Tutorial*. 20 de jun. de 2015. URL: http://apmonitor.com/me575/index.php/Main/SimulatedAnnealing.

[Joy15] Joyent, Inc. *node.js*. 15 de abr. de 2015. URL: https://nodejs.org/.

[Net15] Metaheuristics Network. *Metaheuristics Network web site*. 11 de abr. de 2015. URL: http://www.metaheuristics.net/index.php?main=1.

[Pro15] Go Test Problems. *TEST FUNCTIONS: Rastrigin Function*. 20 de jun. de 2015. URL: http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2607.htm.

[Vil+05] J. Villadangos et al. "Efficient leader election in complete networks". Em: *Parallel, Distributed and Network-Based Processing, 2005. PDP 2005. 13th Euromicro Conference on*. Fev. de 2005, pp. 136–143. DOI: 10.1109/EMPDP.2005.21.