Algoritmo Distribuído com abordagem em *cache* cooperativo

Pedro Paulo Simões Freitas, Ricardo Augusto Rabelo
PPGCC - Programa de Pós-Graduação em Ciência da Computação
UFOP - Universidade Federal de Ouro Preto
Ouro Preto, Minas Gerais, Brasil
email: pedropufop@gmail.com, rrabelo@gmail.com

***Resumo*—A utilização da técnica de *cache* é essencial para melhorar a recuperação de dados e desempenho da rede para usuários de dispositivos móveis. Neste artigo serão apresentados algoritmos distribuídos com abordagem em cooperação de cache. Também será feita uma análise de complexidade dos mesmos. Em se tratando de algoritmos distribuídos podemos perceber que uma melhor análise será em função da troca de mensagens. O DAJ será utilizado como ambiente de simulação, para que os testes sejam realizados. Os teste serão feito em um ambiente onde há pouca mobilidade, sem desconexões.**

***Keywords*-algoritmos distribuídos, *cache* cooperativo, complexidade.**

I. Introdução

Os algoritmos distribuídos são a base dos sistemas distribuídos. De acordo com [1], algoritmos distribuídos são definidos como uma coleção de processos interligados por uma rede de canais de comunicação.

Esses processos geralmente executam o mesmo algoritmo. Dependendo do sistema considerado, uma rede deste tipo pode consistir de conexões ponto-a-ponto, onde cada elo lida com o tráfego de comunicação entre dois processos, ou mesmo representado por grafos onde os vértices representam os processos e as arestas os canais de comunicação.

Nestes tipos de sistemas, os processos não compartilham qualquer tipo de memória. Assim a troca de informações entre os processos deve ser realizada necessariamente por meio de mensagens. As trocas de mensagens são feitas pelos canais de comunicação existentes entre os nós. De acordo com [2], o sentido da comunicação, pode ser:

- *Simplex*: as transmissões podem ser feitas apenas em um só sentido, de um dispositivo emissor pra um ou mais dispositivos receptores;
- *Half-Duplex*: as transmissões podem ser feitas nos dois sentidos, mas alternadamente, ou seja, uma vez enviando, uma vez recebendo, não podendo ser nos dois sentidos ao mesmo tempo;
- *Full-Duplex*: para este caso, as transmissões podem ser feitas nos dois sentidos ao mesmo tempo.

Com isso vem a ideia do *cache* cooperativo. De acordo com [3], esta estratégia coloca cópias de documentos mais utilizados mais próximos dos clientes, diminuindo o tráfego na rede e a taxa de requisições recebidas pelos servidores de origem, assim ocorre uma redução do tempo de acesso. A utilização desses *caches* é uma boa opção para diminuir o número de requisições feitas a servidores remotos.

Com o grande aumento de dispositivos móveis que possuem várias interfaces de redes, existe a possibilidade de um grande aumento do número de pessoas que podem estar conectadas a rede, por exemplo a internet. O cenário que analisaremos, considera que várias pessoas com algum dispositivo móvel. Estes dispositivos tem acesso a alguma rede, incluindo uma rede de comunicação *ad hoc*, com acesso local entre as mesmas.

Pensando no cenário citado acima onde a conexão é através de uma rede sem fio, o *cache* é um conjunto de réplicas armazenadas localmente nos dispositivos móveis. Essa replicação é uma técnica chave para fornecer uma maior disponibilidade e desempenho nos ambientes móveis [4].

Dessa maneira, o *cache* é compartilhado entre os dispositivos móveis, introduzindo um novo cenário com algumas questões: Como encontrar a informação que o usuário necessita? É mais barato pesquisar com vizinhos ou ir direto ao servidor?

A outra situação envolve a criação de uma tabela de roteamento, para saber as localizações uns dos outro, e qual a validade. Com isso teremos outra pergunta: é mais barato manter uma tabela ou o usuário ser informado diretamente de quem tem a informação necessária? Ou ainda a mesma questão citada anteriormente, acessar direto o servidor que está buscando.

Podemos considerar também que para acessar um servidor, os usuários podem utilizar uma rede 3G, a qual seria a interface de rede para ter uma conexão. A utilização desta interface para acessar o servidor é mais cara, em relação a acessar um vizinho ao lado que pode ter a informação. Este custo está relacionado ao gasto de bateria.

Neste contexto, o principal objetivo deste trabalho é descrever, realizar análise de complexidade e apresentar testes em algoritmos que utilizam a abordagem de *caching* cooperativo.

O restante do artigo está organizado da seguinte forma. Na Seção II, são apresentados alguns trabalhos relacionados. Os métodos serão mostrados na Seção III. Na Seção IV, é apresentado a análise de complexidade dos métodos. E por fim na Seção V, são descrito os resultados obtidos nos testes.

II. Trabalhos Relacionados

Abordagens utilizando o *caching* em sistemas cooperativo, estão sendo utilizadas cada vez mais, pois, com a utilização dessa técnica, a troca de mensagens em uma rede pode diminuir bastante.

De acordo com [5], a maioria dos métodos atuais para coerência de cache em ambientes móveis não foram projetados para trabalhar com sistemas cooperativos. Em [6] é apresentado uma técnica de manutenção da coerência de cache e provisão de percepção em um ambiente cooperativo computadorizado. Este ambiente suporta mobilidade dos usuários através de uma interface de comunicação sem fio.

III. Métodos

Os métodos descritos serão o Propagação de Informação com *Feedback* (PIF), *Cooperative Caching* (COCA) e o *Distributed Group-Based Cooperative Caching* (DGCOCA). Para fazer esta análise teremos que utilizar uma ambiente de simulação para algoritmos distribuídos.

Nas Seções III-A, III-B e III-C, serão descritos os algoritmos.

A. PIF

De acordo com [1],o algoritmo PIF consiste em difundir uma informação presente em um nó s, para todos os outros nós vizinhos. O fim da execução acontece quando, o nó s estiver recebido a confirmação do recebimento da informação por todos os outro nós. Com isso a ideia de propagação da informação com *feedback*. O algoritmo do PIF segue no Programa 1.

```
Variáveis:
  parent(i) = nil; //vizinho ni por onde inf
      chegou inicialmente
  count(i) = 0; //número de cópias recebidas de
      inf
  reached(i) = false //indica se n(i) já recebeu
      inf ou não

Input:
  msg(i) = nil;
Action if(i) pertence N(0)
  reached(i) = true;
  Send inf to all n(j) pertende Neig(i);

Input:
  msg(i) = inf | origin(i)(msg(i))
Action:
  count(i) = count(i) + 1;
  if not reached(i)
  then begin
    reachec(i) = true;
    parent(i) = n(j);
    Send inf to all n(k) pentence Neig(i) | n(k)
        parent(i);
  end;
  if count(i) = |Neif(i)|
  then if parent(i) diferente nil
    then Send inf to parent(i);
```

Programa 1. Algoritmo do PIF [1]

B. COCA

As informações sobre o COCA foram extraídos de [7]. Para este trabalho faremos algumas definições. Será assumido que cada cliente (dispositivo móvel), m_i, contém um identificador único, ou seja, a rede seria composta por $m_1, m_2, ..., m_n$, onde n é o número total de clientes na rede. Em COCA, cada cliente e seus colegas vizinhos, ou seja, os clientes que residem na faixa de transmissão do outro cliente, trabalham juntos como um grupo dinâmico de partilhar os seus itens de dados, que estão em *cache*, de forma cooperativa, através dos canais P2P.

Na Figura 1, é ilustrado essa arquitetura. Se um cliente, m_2, encontra o item dado necessário em seu *cache* local, constitui um sucesso na busca no cache local, caso contrário, é uma falha. Quando m_2 encontra um falha na busca em cache local, ele sintoniza o canal de transmissão, e envia um pedido a seus pares vizinhos, m_1 e m_3. Se um deles contiver o item dados necessários, acontece um sucesso de busca no cache global dos vizinhos, caso contrário, acontece uma falha, e m_2 precisa esperar até que o item de dados apareça no canal de *broadcast*.

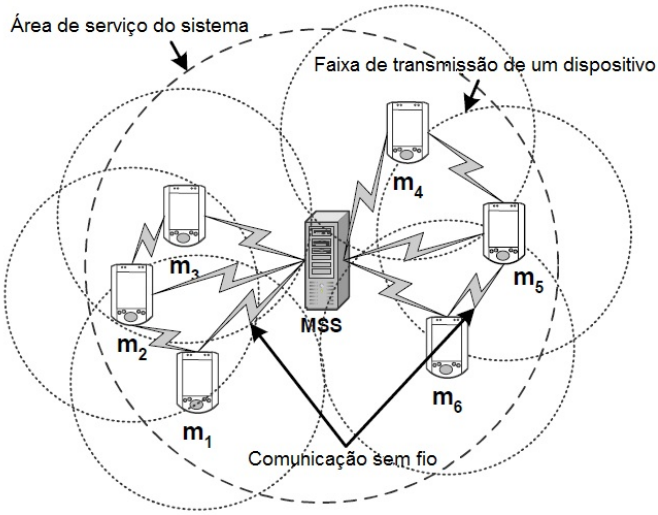

Figura 1. Arquitetura COCA [8]

COCA é apropriado para um ambiente no qual um grupo de clientes possuem interesses em comum. Um exemplo seria em um *shopping center*. Os clientes que estão na mesma loja, provavelmente farão acesso a informações relacionadas à loja.

Com base na arquitetura do sistema COCA, o modelo de rede COCA e protocolo de comunicação são descritos a seguir.

1) Modelo de rede: Em COCA, está disponível um protocolo de descoberta de vizinho. Este protocolo funciona da seguinte maneira, a conectividade com o vizinha está mantida através de um sinal periódico. Como o período do

sinal é curto o suficiente, o protocolo pode detectar com precisão um novo link para um cliente conectado, e desligar um link para um cliente desconectado. No modelo P2P, há dois paradigmas de comunicação P2P: ponto-a-ponto e de difusão. Para a primeira, há apenas um cliente de destino para a mensagem a ser enviada a partir do um cliente origem, enquanto no segundo, todos os clientes no alcance da transmissão do cliente origem recebem a mensagem.

2) Protocolo de Comunicação: A comunicação entre clientes funciona da seguinte maneira. A primeira etapa a ser realizada após o pedido do cliente é, procurar em seu cache local. Se não encontrar o item desejado, ele sintoniza o canal de transmissão, e transmite uma mensagem de pedido a seus pares vizinhos. O vizinho que conter o item de dados em seu cache, envia uma mensagem de resposta ao cliente requerente. Assim o cliente seleciona o primeiro vizinho dos que enviaram a mensagem de resposta como um ponto-alvo. Em seguida envia uma mensagem para recuperar o item de dados, para o vizinho alvo. E por fim, o vizinho marcado como ponto-alvo se transforma no item dados requerido. Caso os dados necessários são exibidos no canal de transmissão antes de qualquer resposta de seus vizinho, o cliente coleta do canal e não envia qualquer mensagem de recuperar. Em caso de não encontrar o item no cache dos vizinhos, em um determinado período de tempo, o cliente precisa esperar o item de dados aparecer no canal de broadcast.

C. DGCOCA

De acordo com [8], o DGCOCA é uma extensão do COCA. O DGCOCA, utiliza um algoritmo estável de descoberta de vizinho. Este algoritmo tem o propósito de descobrir um padrão de mobilidade entre um cliente e seus vizinhos.

Existem quatro tipos de relacionamento entre dois cliente: desconhecido, estranho, amigo, e membro. Para classificar estes relacionamentos o algoritmo contém uma técnica de memorização. Quando um cliente recebe vários sinais de um outro cliente, por um determinado período de tempo, a relação entre eles torna-se mais perto. Caso contrário, se um cliente não obtiver nenhum sinal a partir de outros clientes por um mesmo período de tempo, sua relação diminuirá. No inicio da rede, a relação entre quaisquer clientes é desconhecida.

A relação entre dois clientes pode ser mostrada da seguinte maneira. Dois clientes, m_i e m_j, com uma relação desconhecida, após m_i receber um sinal de m_j, m_j torna-se um estranho para m_i. Quando m_i recebe consecutivos sinais de m_j por um certo período de tempo, m_j torna-se amigo de m_i. Para uma próxima evolução da relação, m_i deve receber mais sinais de m_j por mais um certo período de tempo. Assim m_i trata m_j como um membro da rede. No entanto, se m_i não receber qualquer farol de m_j, durante um período, a sua relação será rebaixada para amigo. Em seguida, se m_j não mandar mais nenhum sinal para m_i, por um novo período, m_j trona-se estranho para m_i. Finalmente, continuando o mesmo processo de m_j não mandar mais sinais a m_i, m_j será considerado desconhecido para m_i. A Figura 2, mostra o comportamento das relações.

Nos Programas 2 e 3, é mostrado o algoritmo da descoberta estável de vizinho, protocolo utilizado pelo DGCOCA. O Programa 2 é executado periodicamente para atualizar as alterações de relação entre nós. O Programa 3 é executado quando um cliente recebe um sinal de outro nó.

```
DescoberEstaveldeVizinho(Relation Ri,
    CacheSignature Si)
  // rel(i, j) mantém relacionamento atual entre i
       e j
  mi and mj
  // Ri contém as relacoes de mi,
  for all rel(i,j) pertence Ri do
    // now() retorna tempo atual
    if now()-last_beacon_tsj > BeaconPeriod then
      if rel(i, j) == estranho then
        rel(i, j) = desconhecido;
        Ri = Ri - {rel(i, j)};
        next mj ;
      end if
      if not disappearedj then
        last_disappeared_tsj = now();
        disappearedj = true;
      end if
    end if
    if disappearedj then
      if now()-last_disappeared_tsj >= pi then
        if rel(i,j) == membro then
          rel(i,j) = amigo;
        else if rel(i,j) == amigo then
          rel(i,j) = estranho;
          if Sj pertence Si then
            Si = Si - {Sj};
          end if
        end if
        last_disappeared_tsj = now();
      end if
    else
    if now()-first_consecutive_beacon_tsj >= pi
        then
      if rel(i, j) == estranho then
        rel(i, j) = friend;
        first_consecutive_beacon_tsj = now();
      else if rel(i,j) == amigo then
        if sim(mi,mj) >= d then
          rel(i,j) = member;
          if Sj nao pertence Si then
            Si = Si uniao {Sj};
          end if
        end if
      end if
    end if
  end for
```

Programa 2. Atualiza ralação [8]

```
OnReceiveBeacon(Relation Ri, Cliente mj)
  // mi recebe uma messagem  hello beacon de mj
  last_beacon_tsj = now();
  if rel(i,j) nao pertence Ri then
    rel(i,j) = estranho;
    Ri = Ri uniao {rel(i,j)};
    first_consecutive_beacon_tsj = now();
  else if disappearedj then
    disappearedj ? false;
```

```
10      first_consecutive_beacon_tsj = now();
    end if
```

Programa 3. Recebe sinal [8]

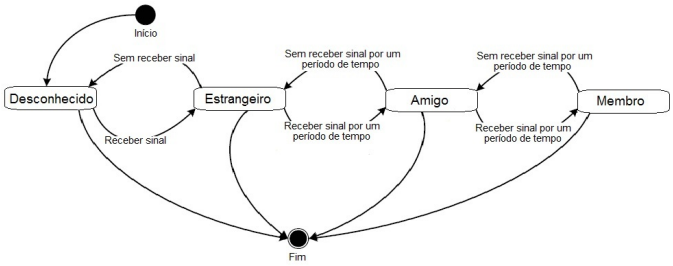

Figura 2. Relacionamentos [8]

IV. Análise de Complexidade

Será feito uma caracterização da conectividade e complexidade de troca de mensagens, já que falando de algoritmos distribuídos, o número de mensagens influência muito na latência da rede.

O grafo resultante, chamado $G = (V, E)$, onde V é o conjunto de nós (vértices) e $E = (r(n), c(n))$, é o conjunto de arestas não direcionadas para cada nó vizinho.

O grafo pode ser colocado como um modelo para outros cenários reais de redes *ad hoc*, onde nós são obrigados a manter um pequeno número de conexões simultâneas, por causa dos recursos limitados, tanto enérgico quanto computacional, ou quando estabelecem ligações para os vizinhos que é muito caro no tempo ou energia.

Em [9], é mostrado que para qualquer r fixo, onde $r > 0$ existe uma (grande) constante c tal que $G = (r, c)$ tem chance de expandir a rede. Em [10], se for provado está chance de expansão, $G = (r, c)$ é conectado para qualquer constante fixa $r > 0$ e $c = 2$ sempre que n se torna suficientemente grande.

O grau de cada vértice será proporcional ao cenário de mobilidade e conexão do ambiente de testes. Para este artigo, consideramos que a comunicação se da através de uma interface de rede sem fio como *wifi* ou *bluetooth*, cujo alcance máximo varia de 100m (*wifi*) e 20m (*bluetooth*) e o cenário de utilização desta aplicação surge em um ambiente fechado, como uma sala ou um andar, considera-se que os valores de r e c são viáveis para que exista uma conectividade entre os elementos. A saída de nós da rede se dará pelo desligamento do equipamento ou pela saída do ambiente. A mobilidade considerada será baixa, de maneira que os valores de r e c, serão pequenos. Será desconsiderado o custo do estabelecimento de conexão.

A complexidade do envio de mensagens será proporcional a troca de mensagens em cada algoritmo. Cada vértice do grafo contribuirá com $\Theta(grau - do - vertice)$. Considerando que o cenário será sem mobilidade, teremos um grafo completo K_n, no qual cada vértice contribuirá com $\Theta(n-1)$ mensagens. A cada ciclo de envio, tem-se $O(n^2)$ mensagens na rede.

Para averiguar o custo de cada solução será necessário medir quantos ciclos de mensagens são necessários para a convergência, ou a parada final do algoritmo. Além disso, a cada ciclo, o algoritmo pode diminuir o envio de mensagens.

Os algoritmos COCA e DGCOCA, utilizam diferentes protocolos de descoberta de vizinho, este protocolo que influência na troca de mensagens. O protocolo do COCA sempre considera todos vizinhos do cliente como possíveis, cooperadores de cache, através de um sinal periódico. Já no DGCOCA, existem relacionamentos, assim o nó poderá ser vizinho e não ter nenhum vinculo, não realizando troca de mensagens. O fato de não ter troca de mensagens entres nós, faz com que não tenha nenhum tipo de relacionamento. O relacionamento se torna mais forte de acordo com o aumento das trocas de mensagens. O relacionamento membro é o mais forte, pois o nó se torna membro da rede do cliente, isso significa que a probabilidade de o item de dados necessário estar em seu *cache* é maior.

Com isso COCA troca mais mensagens que o DGCOCA, pois com a ideia de relacionamento, tende a diminuir o número de mensagens em busca do item de dados. Já no COCA o número de mensagens sempre será o mesmo.

Para o caso COCA o número de mensagens vai depender das quantidade de vizinhos. E no DGCOCA, o número de mensagens depende dos vizinhos e seus relacionamentos entre eles, que será igual a $O(n^2)$.

Já no PIF a sua complexidade em relação a troca de mensagem será igual ao do grau do vértice, que é o número de arestas que chegam e saem do vértice, multiplicado pelo número de vértices, $\Theta(grau - do - vertive \times numero - de - vertice)$. Podemos pegar o grau de qualquer vértice, já que todos vértices são ligados a todos, assim o grau é igual.

V. Experimentos

Como já foi mencionado na Seção III o ambiente de simulação utilizado é o DAJ (*Distributed Algorithms in Java*). De acordo com [11], o DAJ é um *framework* para escrever programas em Java que implementam algoritmos distribuídos. Estes programas mostram as estruturas de dados de cada nó, para o nosso caso serão os dispositivos móveis, e permitir que o uma construção interativa do cenário. Em uma situação de aprendizagem, a execução interativa é preferível, pelo fato de pode assistir a simulação.

Nas Figuras 3, 4 e 5, são apresentados gráficos de quantidade de nós versus número de mensagens. Foram geradas instâncias de 2 a 30 nós. O cenário analisado é o de uma rede móvel, onde os nós são os dispositivos móveis. Uma característica fundamental dessa rede é a baixa mobilidade, evitando a desconexão de nós.

O primeiro gráfico a ser apresentado será o da Figura 3, onde é mostrado o desempenho em relação à troca de

mensagens do PIF. O número de mensagens aumenta com o número de nós.

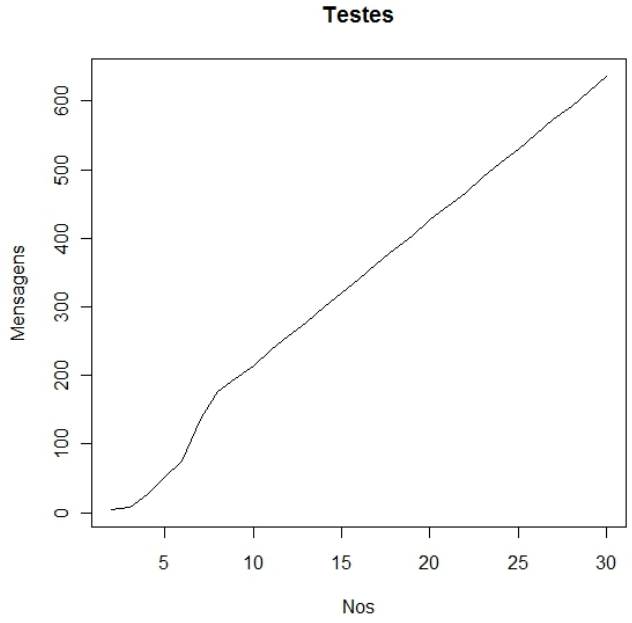

Figura 3. PIF

Já no segundo gráfico, o da Figura 4, é mostrado a evolução da troca de mensagens pelo número de nós do DGCOCA. Neste algoritmo também podemos observar o aumento do número de mensagens, quando é modificado o número de nós.

A partir do gráfico da Figura 5, que é uma junção dos dois gráficos em um, podemos perceber que, o custo de manter um relacionamento é alto em relação a troca de mensagens. Assim um algoritmo mais simples como o PIF, no cenário apresentado, pode obter um resultado bem mais satisfatório. Pois o número de troca de mensagens é bem menor, comparado ao DGCOCA.

A seguir mostraremos na Tabela I, resultados pontuais do número de mensagens trocadas entre os algoritmos. Podemos perceber pela terceira coluna da tabela, que a diferença do número de mensagens, aumenta muito com o aumento de nós.

Tabela I
PIF X PGCOCA

nº nós	nº mensagens PIF	nº mensagens DGCOCA	Diferença em nº de mensagens
10	214	235	21
20	426	1135	709
30	636	2701	2065

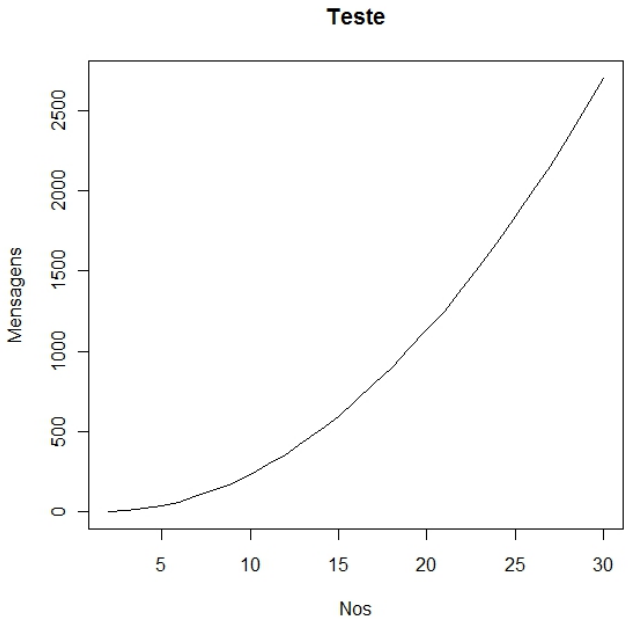

Figura 4. DGCOCA

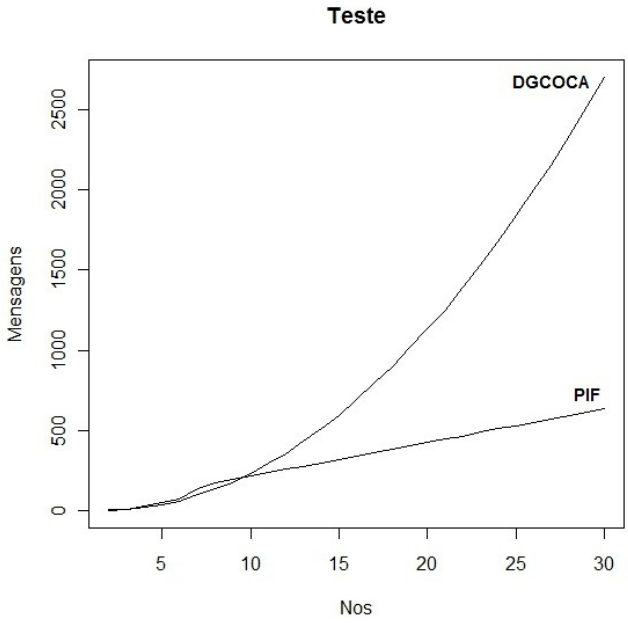

Figura 5. PIF X DGCOCA

REFERÊNCIAS

[1] V. C. Barbosa, *An Introduction to Distributed Algorithms*. The MIT Press, 1996, iSBN 0-262-02412-8.

[2] A. S. Tanenbaum, *Redes de Computadores*, 4th ed. Campus,

2003, iSBN: 8535211853.

[3] R. F. Brandão and R. de Oliveira Anido, "Cooperação entre caches para web," 1999.

[4] Y. Saito and M. Shapiro, "Optimistic replication," *ACM Comput. Surv.*, vol. 37, pp. 42–81, March 2005.

[5] Z. Wang, S. K. Das, H. Che, and M. Kumar, "A scalable asynchronous cache consistency scheme (saccs) for mobile environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, pp. 983–995, November 2004.

[6] C. D. M. Berkenbrock and C. M. Hirata, "Coerência de cache e percepção em ambientes de cooperação móvel," *XXVII Congresso da Sociedade Brasileira de Computação*, pp. 1–14, 2007.

[7] C.-Y. Chow, H. V. Leong, and A. Chan, "Peer-to-peer cooperative caching in mobile environments," in *Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04) - Volume 7*, ser. ICDCSW '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 528–533.

[8] C.-Y. Chow, H. V. Leong, and A. T. S. Chan, "Distributed group-based cooperative caching in a mobile broadcast environment," in *Proceedings of the 6th international conference on Mobile data management*, ser. MDM '05. New York, NY, USA: ACM, 2005, pp. 97–106.

[9] A. Panconesi and J. Radhakrishnan, "Expansion properties of (secure) wireless networks," in *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, ser. SPAA '04. New York, NY, USA: ACM, 2004, pp. 281–285.

[10] D. Dubhashi, C. Johansson, O. Häggström, A. Panconesi, and M. Sozio, "Irrigating ad hoc networks in constant time," in *Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures*, ser. SPAA '05. New York, NY, USA: ACM, 2005, pp. 106–115.

[11] M. Ben-Ari, "Interactive execution of distributed algorithms," *J. Educ. Resour. Comput.*, vol. 1, August 2001.