4
Aplicações da técnica de DHT em sistemas de computação distribuída

O objeto deste trabalho é investigar aplicações da técnica de DHT em sistemas de computação distribuída. Para isto, o ALua foi escolhido como plataforma de estudo, pois como visto no Capítulo 3, o ALua é uma infra-estrutura que, por ser bastante flexível, permite o desenvolvimento de sistemas de computação distribuída baseados em diversas arquiteturas, inclusive na arquitetura P2P. Porém, como o ALua não oferecia nenhuma das características que foram vistas no Capítulo 2 — localização eficiente de recursos, utilização uniforme dos nós e replicação de dados —, era de responsabilidade do desenvolvedor implementá-las. Assim o ALua foi modificado para oferecer funcionalidades de uma arquitetura P2P.

Escolheu-se a arquitetura P2P estruturada, por ela ser mais robusta que a não-estruturada. Na Seção 2.4 encontra-se a avaliação de quatro protocolos candidatos a serem utilizados na implementação.

Como na arquitetura P2P estruturada não existem conexões diretas entre todos os nós da rede, é necessário que o roteamento de mensagens seja eficiente e que seja escalável com o aumento da rede. Mais adiante, na Subseção 4.1.1, será mostrado como foi feita a integração deste roteamento com o mecanismo de trocas de mensagens do ALua, e quais são as operações disponíveis para a camada de aplicação.

Na Subseção 4.1.2 está descrita a implementação de grupos e como eles são utilizados para prover tolerância a falhas e fazer o balanceamento de carga entre os seus membros. Também está descrita a API para a utilização dos grupos, bem como alguns eventos que podem ser úteis para monitorar entrada e saída de membros.

4.1
Implementação

Criou-se um novo módulo, *dht*, responsável por criar e operar a rede P2P. Este módulo possui dois submódulos: *route* e *storage*. No primeiro estão as funções básicas de roteamento de mensagens, entrada na rede e atualização das informações de roteamento. O segundo implementa as operações de publicação e busca de recursos. Também criou-se o módulo *group* que é o responsável por todas as operações de grupo (Subseção 4.1.2). Na Figura 4.1 pode-se ver o

PUC-Rio - Certificação Digital Nº 0912828/CA

diagrama com os módulos do ALua, e os novos módulos estão destacados com cor mais clara.

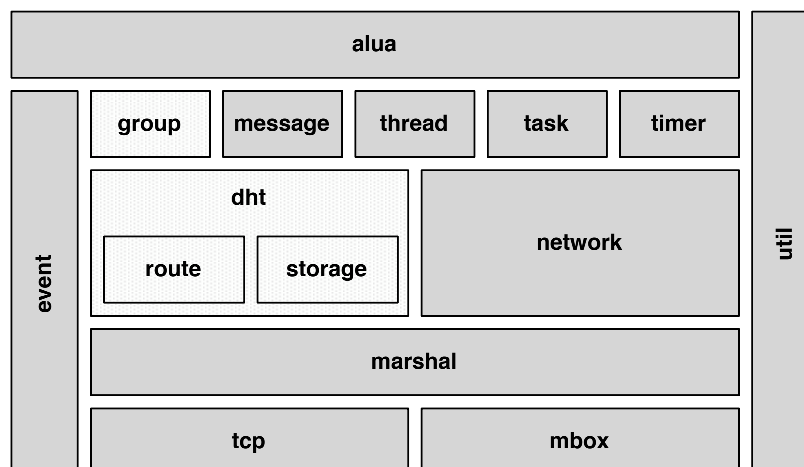

Figura 4.1: Diagrama dos módulos do ALua

A função de *hash* utilizada foi a SHA-1 que gera saídas de 160 bits.

PUC-Rio - Certificação Digital Nº 0912828/CA

4.1.1 Roteamento de mensagens

Como visto na Subseção 3.3.2, toda troca de mensagens entre processos é realizada através da rede por estes formada. Na nova versão, esta é uma rede P2P baseada na técnica de DHT onde os processos desempenham o papel de nós da rede. O roteamento segue o algoritmo básico mostrado no Código 4.1.

```
1 function network.route(dst, msg)
2   if dst == node_id then
3     task.schedule(msg)
4   elseif proc(dst) == proc(node_id) then
5     mbox.send(proc(dst), msg)
6   else
7     dht.route_msg(dst, msg)
8   end
9 end
```

Código 4.1: Algortimo básico de roteamento

Primeiramente verifica-se, na linha 2, se o destino da mensagem é o processo local e, caso seja, a mensagem é colocada na sua fila para ser processada. Se o destino não é o processo local mas é um *trabalhador Lua* associado ao processo local (linha 4), então a mensagem é encaminhada a este *trabalhador Lua*. Se o destino é ou está em outro processo diferente do processo local, a mensagem é encaminha pela rede P2P a um processo mais próximo do destino (linha 7).

Um exemplo de uma rede formada pelo ALua com a técnica de DHT pode ser vista na Figura 4.2. Pode-se observar que não existem mais conexões entre todos os processos da rede, mas sim as conexões suficientes para permitir todos os processos se comuniquem entre si e para manter o custo médio da comunicação em um nível no qual a rede opere sem perda de desempenho.

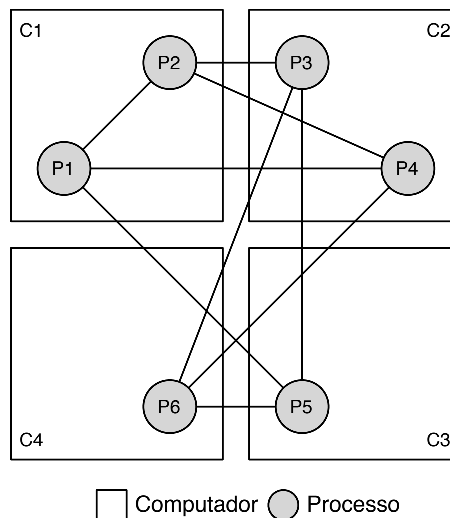

Figura 4.2: Exemplo de rede formada pela técnica de DHT

PUC-Rio - Certificação Digital Nº 0912828/CA

O submódulo de roteamento de mensagem (*route*) foi implementado de forma independente do submódulo *storage*, permitindo sua substituição por outro submódulo que implemente um protocolo de roteamento distinto. Para isto, foi definida uma interface com as operações que este submódulo deve oferecer. A interface está definida no Código 4.2. Esta estratégia facilitou a troca do protocolo e permitiu uma rápida avaliação experimental dos dois protocolos escolhidos.

```
1 function route.join(remote_node, callback)
2 function route.route_msg(dst_node, message)
3 function route.route_mult_msg(dst_nodes, message)
4 function route.get_replica_nodes()
5 function route.register_handler(handler)
```

Código 4.2: Interface do módulo route

A primeira função, `route.join`, recebe como argumentos o identificador de um processo já pertencente a rede e opcionalmente uma função de *callback* para ser chamada com o resultado da operação (retorno assíncrono). A função `route.route_msg` recebe como primeiro argumento o identificador do processo destino e como segundo argumento a mensagem a ser enviada e retorna um verdadeiro ou false no caso de sucesso ou de erro respctivamente. Já a terceira

função, `route.route_mult_msg`, recebe uma lista com os identificadores dos processos de destino e a mensagem a ser enviada e também retorna verdadeiro ou falso. Por fim, a função `route.get_replica_nodes` não recebe argumentos e retorna a lista dos processos nos quais pode-se replicar os dados locais.

O ALua oferece uma função para o envio de mensagens, `alua.send`, que pode receber um destinatário ou uma lista de destinatários. No primeiro caso ela envia mensagens do tipo *unicast* e no segundo, mensagens do tipo *multicast*. Assim, o módulo `route` tem duas funções de encaminhamento de mensagens: `route.route_msg` e `route.route_mult_msg` que permitem o envio de mensagens do tipo *unicast* e *multicast*, respectivamente. Uma forma de implementar a função `route.route_mult_msg` seria enviar uma mensagem individual para cada destinatário da lista. Porém isto implicaria em uma grande quantidade de mensagens circulando pela rede P2P. Por isto esse tipo de mensagem foi implementado para que a cada *hop* da rota os destinatários sejam agrupados em conjuntos. Para cada conjunto é enviada uma mensagem do tipo *multicast* (ou do tipo *unicast* se o conjunto tiver apenas um destinatário). Esses conjuntos são definidos com base nas tabelas de roteamento do processo. No exemplo da Figura 4.3, o processo **A** recebe uma mensagem *multicast* endereçada a **B**, **E**, **G**, **D**, **I** e **J**. **A** então agrupa os destinatários em três grupos, com base em sua tabelas de roteamento (na figura são representadas pelas setas), e envia três mensagens (duas do tipo *multicast* e uma do tipo *unicast*).

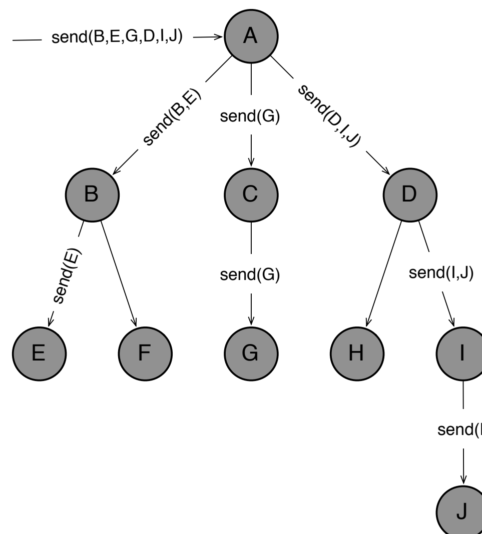

Figura 4.3: Exemplo *multicast*

PUC-Rio - Certificação Digital Nº 0912828/CA

A operação `route.register_handler` permite o registro de tratadores que são chamados quando um dos seguintes eventos ocorrem:

`DHT_NODE_JOIN` evento para notificar a entrada de novos processos na rede. A chamada do tratador deve ter como parâmetro o identificador do novo nó. Uso restrito à camada DHT;

`NEW_NODE_JOIN` análogo ao evento `DHT_NODE_JOIN`, porém para a camada de aplicação;

`DHT_NODE_LEAVE` evento de notificação de saída de processos da rede (ou detecção de que um processo está inativo). Assim como evento `DHT_NODE_JOIN`, o seu tratador recebe o identificador no processo saindo ou processo inativo;

`DHT_REPLICA_NODES_UPD` evento que notifica a alteração da lista de processos responsáveis por guardar as réplicas do processo local. O tratador recebe a nova lista de processos;

`DHT_ROUTING_INF_UPD` este evento é disparado sempre que um novo processo é adicionado à vizinhança do processo local. Seu tratador recebe como parâmetro o processo adicionado.

Todos estes eventos devem ser implementados e seus tratadores invocados nas situações descritas acima na implementação de um submódulo de roteamento.

PUC-Rio - Certificação Digital Nº 0912828/CA

4.1.2
Grupos de processos

A implementação de grupos de processos utiliza o armazenamento distribuído da rede P2P. Um grupo recebe um identificador no momento de sua criação, gerado por uma função de *hash* a partir do seu nome, e este identificador é associado a um processo através do roteamento da rede P2P, assim como no SCRIBE[29]. Este processo será responsável por guardar a lista de processos pertencentes ao grupo e outras informações definidas pela aplicação. Essa associação pode ser alterada dinamicamente se um novo processo, com um identificador mais próximo ao do grupo, entrar na rede, ou se o processo atualmente responsável pelo grupo sair (nesse caso uma das réplicas assumirá seu papel).

O grupo de processos pode compartilhar informações definidas pela aplicação através de uma tabela *hash* na forma de chave/valores. Essa tabela *hash* só é acessível através de chamadas à API de grupos, e é mantida fisicamente no processo responsável pelo grupo.

É possível enviar três tipo de mensagens para grupos: *unicast*, *broadcast* e *anycast*. No tipo *unicast*, a mensagem é processada pelo processo responsável pelo grupo. As mensagens do tipo *broadcast* são processadas por todos os membros do grupo. Para isto, a mensagem é inicialmente encaminhada ao processo responsável pelo grupo que a envia a cada membro do grupo utilizando o tipo de mensagem *multicast* visto na seção anterior. Já mensagens do tipo *anycast* são processadas, de forma balanceada, pelos membros do grupo, e como ocorre com o tipo *broadcast*, são inicialmente encaminhada ao responsável pelo grupo e depois encaminha a um dos membros do grupo segundo uma política de escalonamento.

Broadcast

Em [14] são mostradas outras formas possíveis de implementação de mensagens do tipo *broadcast* em redes P2P. Elas dividem-se entre as técnicas de inundação e as baseadas em árvores. Nas técnicas baseadas em inundação, a mensagem do tipo *broadcast* parte do processo responsável pelo grupo e é encaminhada a todos os seus vizinhos. Os processos subseqüentes verificam se são membros do grupo, e em caso afirmativo processam a mensagem. Em seguida, encaminham a mensagem a alguns de seus vizinhos, excluindo os vizinhos que receberam a mensagem. Esta seleção é dependente do protocolo de roteamento utilizado e é possível implementá-la para os protocolos Chord e Pastry.

Já nas técnicas baseadas em árvores, uma árvore de distribuição é construída e mantida no processo responsável pelo grupo através das mensagens de *join* e *leave* do grupo. Durante o roteamento de mensagens de *join* ao processo responsável pelo grupo, as rotas criadas são inseridas como ramos da árvore. Depois, para enviar mensagens do tipo *broadcast*, basta seguir estes ramos e os processos pertencentes ao grupo processam as mensagens.

Uma característica comum a estas duas técnicas é que os membros dos grupos são armazenados de forma descentralizada, porém como a implementação do balanceamento de cargas está centralizada no responsável pelo grupo, pois é ele quem tem lista de membros, nenhuma destas técnicas foi utilizada.

Balanceamento de carga

Um grupo de processos pode ser visto como um provedor de recurso computacional, da mesma forma que um processo. Se ao enviar uma mensagem a um processo, esta é processada no destino, uma analogia pode ser feita para

PUC-Rio - Certificação Digital Nº 0912828/CA

grupos: ao enviar uma mensagem a um grupo ela será processada pelo grupo. Como visto anteriormente, existem três tipos de mensagens: *unicast*, *broadcast* e *anycast*. No primeiro tipo, ela é processada pelo processo responsável pelo grupo, no segundo, por todos os membros do grupo e, no terceiro, por um dos membros do grupo. Este mecanismo de escolha do membro no qual a mensagem será processada permite que a carga recebida pelo grupo seja distribuída entre seus membros.

A idéia inicial era utilizar a técnica de DHT como mecanismo de distribuição das mensagens: os membros do grupo formariam uma rede P2P e a mensagem seria encaminhada a um membro via roteamento nesta rede formada apenas pelos membros do grupo. Para isto, cada mensagem receberia um identificador e a rede a encaminharia ao membro com o identificador mais próximo ao da mensagem. Portanto seriam utilizadas duas camadas de DHT, uma formada pelos processos da rede e outra formada pelos processos membros dos grupos. Porém esta idéia foi abandonada, pois seria complexo utilizar duas camadas de DHT. Além da complexidade, o fato de definir uma política de escalonamento poderia restringir o uso do ALua, pois existem sistemas para os quais essa política não é a mais indicada. Por isto optou-se por implementar uma política mais simples utilizando uma fila circular. Os membros são colocados no final da fila a medida que entram no grupo. Quando uma mensagem do tipo *anycast* é recebida pelo grupo, ela é encaminhada ao primeiro da fila que em seguida é colocado no fim da fila. Como esta política de escalonamento também não é adequada a todos os tipos de problemas, é oferecida uma API para programar a política de escalonamento das mensagens entre os processos membros do grupo. Uma desvantagem desta implementação em relação àquela que usa duas camadas DHT é que enquanto naquela o encaminhamento da mensagem está dividido entre os membros do grupo, nesta o encaminhamento das mensagens é feito pelo processo no qual o grupo está armazenado. Entretanto ganha-se em flexibilidade na programação da política de escalonamento.

PUC-Rio - Certificação Digital Nº 0912828/CA

API de Grupos

A API de grupos disponibiliza funções para a criação e exclusão de grupos, para inserir e excluir processos membros em grupos, para obter a lista de membros do grupo, para inserir e excluir pares de chave/valor da tabela *hash* do grupo, para enviar mensagens dos tipos *broadcast* e *anycast* (para enviar mensagens do tipo *unicast*, utiliza-se a mesma função, do ALua, que envia mensagens a processos) e para a definição da política de escalonamento

de mensagens do tipo *anycast*.

No Código 4.3 estão listadas todas as funções e eventos disponíveis para interagir com grupos de processos.

```
1  function group_create(groupname, callback)
2  function group_delete(groupname, callback)
3
4  function group_join(groupname, callback)
5  function group_leave(groupname, callback)
6
7  function group_add_member(groupname, new_member, callback)
8  function group_remove_member(groupname, member, callback)
9  function group_get_members(groupname, callback)
10
11 function group_set_key(groupname, key, value, check_flag, callback)
12 function group_get_key(groupname, key, callback)
13
14 function group_multicast(groupname, code)
15 function group_message(groupname, code, callback)
16
17 function set_scheduling_policy(policy)
18
19 GROUP_MEMBER_LEAVE
20 GROUP_NEW_MEMBER
```

Código 4.3: API de Grupos

PUC-Rio - Certificação Digital Nº 0912828/CA

As funções `group_create` e `group_delete` são usadas para criar e excluir grupos e recebem como argumento o nome do grupo e uma função de *callback* opcional. Para um processo entrar ou sair de um grupo, ele deve chamar as funções `group_join` e `group_leave` com o nome do grupo e, opcionalmente, uma função de *callback* como argumentos. Pode-se inserir e retirar membros de um grupo usando as funções `group_add_member` e `group_remove_member` que recebem como argumentos o nome do grupo, o identificador do processo sendo inserido/retirado e opcionalmente uma função de *callback*.

As funções `group_set_key` e `group_get_key` são usadas para acessar a tabela *hash* de um grupo e podem ser invocadas por qualquer nós da rede. Como esta tabela é uma memória compartilhada e as funções não possuem mecanismos de contenção, é necessário atenção ao utilizá-las. As funções quando utilizadas isoladamente não apresentam problemas, pois como o ALua é baseado em eventos, elas serão tratadas isoladamente no responsável pelo grupo. Porém quando utilizadas em conjunto, por exemplo ao usar-se a `group_get_key` para pegar o valor de uma chave e `group_set_key` para incrementar este valor, não há garantia de que entre as duas chamadas o valor não será alterado. A função `group_set_key` recebe o nome do grupo, uma chave, um valor, uma indicação de verificação da chave — se a chave já estiver na tabela ela não é alterada, se não é inserido o par chave/valor recebido — e opcionalmente uma função de *callback*. E a função `group_get_key` recebe o nome do grupo, uma chave e uma função de *callback* para receber o retorno.

A função `group_get_members` retorna a lista de membros de um grupo e tem como argumentos o nome do grupo e uma função de *callback*.

Para o envio de mensagem existem as funções `group_multicast` e `group_message`. Através da função `group_multicast` envia-se mensagens do tipo *broadcast*, e que recebe como argumentos o nome do grupo e o código para executar no destino. A função `group_message` permite o envio de mensagens do tipo *anycast* e recebe o nome do grupo, o código e uma função de *callback* opcional. Para o envio de mensagens do tipo *unicast* usa-se a função `alua.send`, que não é da API de grupo, tendo como argumento o nome do grupo, o código e uma função de *callback* opcional.

Existe ainda a possibilidade de registrar tratadores para os eventos `GROUP_MEMBER_LEAVE` e `GROUP_NEW_MEMBER`, que são, respectivamente, o evento disparado quando um processo membro deixa ou é retirado de um grupo, e quando um processo entra ou é inserido em um grupo. Os tratadores são invocados no processo responsável pelo grupo e recebem como argumentos o nome do grupo e o identificador do processo. O registro de tratadores é realizado pela função do ALua `handler_registration` que tem como argumento o evento e o tratador.

A função `set_scheduling_policy` é responsável por definir a política de escalonamento de um grupo para as mensagens do tipo *anycast*. Ela recebe como argumento uma função `policy` que será responsável por escolher para qual membro do grupo a mensagem do tipo *anycast* deve ser encaminhada. Esta função `policy` será chamada pelo ALua passando como argumento a lista de membros do grupo.

PUC-Rio - Certificação Digital Nº 0912828/CA

4.2 Camada DHT

Para o desenvolvimento de aplicações utilizando o ALua é interessante que a aplicação tenha acesso a uma API para manipular a camada DHT. A API oferece a possibilidade do registro de tratadores para eventos gerados pela camada DHT tais como:

- entrada de processo na rede;
- saída de processo da rede;
- entrada de processo no grupo;
- saída de processo do grupo.

Ao permitir o registro de tratadores é possível, por exemplo, realizar a configuração inicial de um processo lhe enviando uma mensagem assim que ele

entre na rede. Ou que um novo processo de um grupo receba o código Lua que o torna capaz de realizar uma tarefa específica deste grupo.

Outros eventos que são usados internamente pelo ALua (replicação de dados, mudança do processo que guarda a informação de um grupo) não são acessíveis pelas aplicações porque são eventos com pouca relevância para a camada de aplicação.

PUC-Rio - Certificação Digital Nº 0912828/CA