PADI@HOME

1. INTRODUÇÃO

O sistema PADI@HOME tem como objectivo paralelizar a execução de makefiles de forma distribuída, numa rede *peer to peer* onde poderá existir um grande número de nós. A efectiva execução dos comandos dos makefiles é realizada pelos voluntários, nós que se registam num dos coordenadores. Estes são responsáveis por receber o makefile da aplicação cliente, analisá-lo, determinar que comandos podem ser executados em paralelo e distribuí-los pelos voluntários registados em todo o sistema, colaborando com os outros coordenadores. Para suportar o controlo de versões existem servidores de nomes que atribuem um identificador universal a cada nova versão de um ficheiro, o qual será utilizado pelos componentes interessados em ler ou escrever o ficheiro. Tanto os ficheiros de entrada como de saída são guardados no repositório distribuído. O PADI@HOME será implementado progressivamente em três fases: Arquitectura Centralizada, Arquitectura Replicada e, finalmente, Arquitectura *Peer to Peer*. Neste documento apresenta-se a solução final e mais adiante as simplificações planeadas para as duas fases iniciais.

2. DESCRIÇÃO DA SOLUÇÃO FINAL - P2P

2.1 Componentes do Sistema

2.1.1 Repositório

O repositório é uma estrutura onde são guardados os ficheiros necessários para a execução dos makefiles. Trata-se de um repositório distribuído, composto por Clientes e Voluntários e que está organizado para utilizar o protocolo Chord[1]. Este protocolo tem como objectivo disponibilizar um único serviço, o lookUp: dado um ID, devolver o endereço do nó responsável por esse ID. No PADI@Home, propõe-se a utilização do Chord[1] para localizar os ficheiros de entrada no repositório e para guardar os ficheiros de saída no nó adequado. Desta forma, os nós estarão organizados em anel, em que cada nó não precisa de conhecer a totalidade dos nós da rede. A cada nó atribui-se um ID, que corresponde ao *hash* SHA-1 de m bits do seu endereço IP. m tem de ser grande o suficiente para que se possa desprezar a probabilidade de dois nós terem o mesmo id. Cada um dos nós mantém uma *finger table* que contém o endereço de, no máximo, m nós. Seja sucessor(n) o sucessor do nó n, a tabela de dispersão de n contém o endereço dos nós sucessor($n + 2^{k-1}$), com $1 \leq k \leq m$. A primeira entrada da *finger table* corresponde, portanto, ao sucessor de n. Com isto, embora cada nó não conheça a totalidade do anel, consegue-se uma performance de localização eficiente e escalável, como é mostrado no artigo que apresenta o protocolo Chord[1].

Quando um nó pretende fazer lookUp a um ficheiro, vai ao Servidor de Nomes obter o ID do ficheiro (gerado a partir do nome do ficheiro, da mesma forma que o ID dos servidores). O ficheiro com ID k é atribuído ao primeiro nó cujo identificador é igual ou maior a k (no espaço de nomes dos nós). Como a tabela de dispersão não tem a totalidade dos nós, a localização será efectuada recursivamente. O nó irá enviar uma mensagem de lookUp para o endereço do nó mais próximo do destino que conhece. Por sua vez, este nó executará o lookUp e assim sucessivamente, até que seja encontrado o endereço do nó responsável pelo ficheiro.

2.1.2 Clientes

Os Clientes são os utilizadores do serviço. Assim, são eles que submetem os makefiles e os ficheiros necessários à sua execução. Depois disto, aguardam por uma notificação assíncrona de que o trabalho esteja concluído para poderem recolher os artefactos produzidos. Cada cliente tem o endereço de um Coordenador e de um Servidor de Nomes.

2.1.3 Servidores de Nomes

Os servidores de nomes têm como objectivo fazer o controlo de versões e manter o mapeamento entre cada versão de um ficheiro e o seu identificador no sistema (ID). O ID é gerado a partir da *hash* SHA-1 de m bits (com m igual ao usado para gerar os IDs dos nós do repositório) da concatenação do nome do ficheiro com a versão em causa.

Um servidor de nomes disponibiliza dois serviços, um de leitura e outro de escrita. Ambos recebem o nome de um ficheiro e devolvem um ID. No caso do serviço de leitura, o servidor de nomes devolverá o ID da última versão do ficheiro. Já o serviço de escrita, devolverá um ID para a próxima versão do ficheiro. O objectivo do controlo de versões é permitir que os componentes interessados em ficheiros dos quais já possuam uma cópia local actualizada não necessitem de os transferir novamente.

O PADI@HOME suporta vários de servidores de nomes, que mantêm a informação coerente e replicada. A replicação da informação dos IDs garante-se com o *broadcast* das alterações efectuadas num servidor para todos os outros. Para isso, todos os servidores de nomes da rede têm uma lista ordenada com os endereços dos outros servidores. Existe o conceito de Servidor de Nomes Central, onde os candidatos a servidor de nomes se têm de registar e de onde recebem a informação correspondente ao estado actual da base de dados de IDs.

Cada cliente/voluntário tem o endereço de um servidor de nomes e, como tal, quando pretende obter o ID para escrita de uma nova versão de um ficheiro, vai pedi-lo ao seu servidor de nomes. Antes de dar a resposta, o servidor tem de ter a certeza de que tem o ID da última versão, a fim de gerar o ID da versão seguinte. Para isso, vai pedir a todos os outros servidores de nomes que façam *lock* ao ficheiro em causa (isto é, que bloqueiem qualquer tipo de alterações ao mesmo). O pedido de *lock* inclui o ID da última versão de que o servidor tem conhecimento. Se algum dos servidores que vai analisar o pedido de *lock* tiver uma versão mais recente é porque esta informação ainda não chegou ao servidor que o despoletou. Dado que a propagação de alterações é feita através de *broadcast*, sabe-se que a actualização vai chegar ao servidor que pediu o *lock* dentro de algum tempo. Por isso, o servidor de nomes que analisou o pedido, vai responder ao *lock* com uma mensagem que informará o servidor remetente da necessidade de aguardar pela mensagem de actualização. Quando o servidor de nomes que está a tentar criar a nova versão obtiver a confirmação do *lock* de todos os outros, vai gerar o novo ID, enviá-lo ao cliente/voluntário e fazer *broadcast* de mensagens de *unlock*, as quais incluem a actualização efectuada. Quando existem *locks* concorrentes, será descartado o que tiver sido pedido pelo servidor que está no grupo há menos tempo (último da lista de endereços), o qual só voltará a tentar depois de receber a actualização do servidor vencedor.

Num pedido de leitura, um servidor de nomes devolve o ID da última versão que tiver na sua base de dados local.

2.1.4 *Coordenadores*

Os coordenadores são responsáveis pela distribuição dos diversos comandos existentes num *makefile*, pelos diferentes voluntários. Esta tarefa pode ser realizada de duas formas. Primeiro, caso o coordenador tenha voluntários registados, envia um dos comandos para cada um deles. Segundo, caso ainda restem regras para distribuir, o coordenador envia-as para o seu sucessor e este repetirá o primeiro processo, assumindo o papel de coordenador auxiliar. Cada coordenador auxiliar, depois de receber de todos os seus voluntários a confirmação de que os comandos foram executados, envia uma mensagem ao predecessor. Este processo ocorre recursivamente até chegar ao coordenador que despoletou a execução, o qual informará o cliente de que já pode descarregar os ficheiros de saída. Existe a noção de ordem nos comandos a compilar, ou seja, o coordenador tem de detectar quais os comandos que dependem de outros, numerando-os por ordem de execução. Sendo assim, a distribuição é feita em várias passagens. Primeiro o coordenador distribui as regras com ordem zero, após estas terminarem, distribui as de ordem um e assim sucessivamente, até que todo o makefile tenha sido executado. Por sucessor entende-se que é o próximo nó no anel, isto porque todos os coordenadores estão organizados numa estrutura circular. Cada coordenador possui o endereço do antecessor, do sucessor e do sucessor do seu sucessor, para efeitos de recuperação de falhas, como será explicado na secção 2.3.1. Pelos mesmos motivos, é enviada uma cópia do makefile recebido para o sucessor e quando terminar a execução o sucessor é notificado para que possa remover a réplica do makefile.

2.1.5 *Voluntários*

Os voluntários são os componentes que efectivamente executam os comandos do *makefile*. Cada voluntário tem o endereço de um servidor de nomes e regista-se num coordenador, do qual vai receber os pedidos de execução de comandos. Após receber um pedido, um voluntário vai, para cada ficheiro de entrada, pedir o ID da sua última versão ao servidor de nomes. Cada voluntário também mantém uma directoria local onde guarda os ficheiros que vai utilizando. Assim, se o voluntário já tiver uma cópia da última versão do ficheiro pretendido, não será necessário descarregá-lo. Caso contrário, fará lookUp no repositório para obter o endereço do nó responsável por armazenar o ficheiro e então descarregá-lo. Se algum ficheiro estiver em falta, o voluntário notificará o coordenador, que abortará a operação, informando o cliente do sucedido. Depois de ter todos os ficheiros de entrada na sua posse, o voluntário procede à execução do comando. Quando tiver terminado, colocará cada ficheiro de saída no respectivo nó do repositório e envia uma mensagem para o seu coordenador a sinalizar a conclusão da operação.

2.2 Funcionamento Normal

Para mostrar o funcionamento do PADI@Home ao longo do processo de execução distribuída de um makefile utilizaremos um exemplo suportado pela Figura 1, onde se representam as trocas de mensagens e as redes de servidores de nomes, de coordenadores e do repositório distribuído. Quando um cliente Cl1 pretende executar um makefile, dá início ao processo pedindo ao seu servidor de nomes (SN1) o ID do(s) ficheiro(s) de entrada (mensagem 1). Após receber do servidor de nomes o respectivo ID (2), o cliente faz lookUp no repositório distribuído (do qual faz parte) a fim de localizar o nó responsável pelo armazenamento do ficheiro. Quando tiver o seu endereço, o cliente envia-lhe o ficheiro. No caso do exemplo, trata-se do voluntário V4 (3).

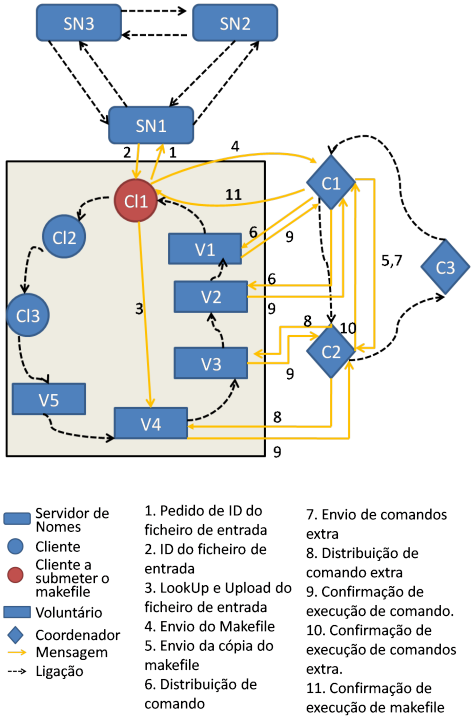

Figura 1 *- Representação esquemática da troca de mensagens para a execução de um makefile.*

Depois da transferência do ficheiro estar concluída, o cliente envia o makefile para o seu coordenador C1 (4) e este envia uma cópia do mesmo makefile para o sucessor C2 para fins de replicação e tratamento de falhas (5). O coordenador C1 analisa o conteúdo do makefile a fim de determinar quais os comandos que podem ser executados paralelamente e as respectivas prioridades. Após esta análise, o coordenador distribui um comando para cada um dos seus voluntários (6). Como estes não são suficientes, ou seja, o número de comandos é maior do que o número de voluntários, o coordenador envia os comandos restantes ao seu sucessor C2 no anel de coordenadores (7), o qual também distribui os comandos pelos seus voluntários (8), e assim sucessivamente até que todos os comandos sejam atribuídos. No exemplo, o makefile submetido, possui apenas quatro comandos de prioridade zero, os quais serão executados pelos voluntários V1 e V2 do coordenador C1 e pelos V3 e V4 do coordenador C2. Um voluntário, depois de executar o comando que lhe estava atribuído, envia ao seu coordenador uma mensagem a indicar o fim da execução (9). O coordenador após receber esta mensagem, se não for o coordenador responsável por aquele makefile, envia uma mensagem de confirmação do fim da execução dos comandos extra que tinha recebido (10). Esta operação ocorre recursivamente se existirem vários coordenadores auxiliares. No caso do coordenador responsável, este espera pela confirmação do seu sucessor e envia a notificação ao cliente, confirmando a execução do makefile (11). Após conclusão do processo anterior o cliente já pode, para cada ficheiro de saída, fazer lookUp no repositório distribuído e descarregá-lo.

2.3 Entradas, Saídas e Falhas de Componentes

Cada rede de componentes tem de lidar com entradas, saídas e falhas dos nós que os compõem. No funcionamento do PADI@HOME considera-se que um componente falhou quando existiu um atraso excessivo na resposta a uma men-

sagem ou caso tenham existido excepções que resultem de invocações remotas. Contudo, é possível que um servidor que tenha sido dado como estando em falha volte a efectuar pedidos à rede. Quando um servidor da mesma rede receber um destes pedidos, deve sinalizar o servidor remetente com uma mensagem de terminação, prevenindo que o sistema se possa tornar incoerente. Note-se que o modelo de falhas apenas prevê a falha de um servidor de cada vez, até que a rede estabilize.

2.3.1 Coordenadores

Entrada: Um coordenador ao entrar, terá de conhecer o endereço de um outro coordenador que já esteja na rede. A figura 2 e a figura 3 mostram a organização dos coordenadores antes e depois da entrada de um novo, respectivamente. A figura 3 ilustra ainda as mensagens trocadas entre eles.

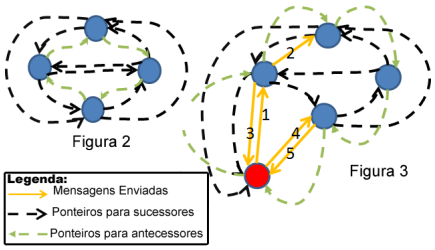

Figura 2 *- Cada coordenador conhece dois sucessores e um antecessor.*

Figura 3 *- Esquema da troca de mensagens em que o novo coordenador esteve envolvido para entrar no anel:* ***1.*** *Pedido de Entrada* ***2.*** *Mudar sucessor para o recém-chegado* ***3.*** *Pedido de entrada aceite e indicação de quais os dois sucessores* ***4.*** *Indicação de qual o novo antecessor e pedido de makefiles replicados do antigo antecessor.* ***5.*** *Envio de réplicas de makefiles e confirmação do fim do processo.*

Saída Ordeira: Se o coordenador que sair estiver a executar algum makefile, espera que o processo termine antes de iniciar a saída. O processo de saída implica a seguinte troca de mensagens:

Com o sucessor: Indica que vai sair e envia o endereço do seu antecessor. Além disso, ainda envia os makefiles replicadas do seu antecessor, caso existam.

Com o antecessor: Envia indicação de que vai sair, juntamente com os endereços dos seus dois sucessores, para que o seu antecessor possa actualizar os seus.

Saída por falha: No anel de coordenadores as falhas são detectadas através de uma mensagem de ping que é periodicamente enviada de um nó para o seu sucessor. Se o sucessor não responder (timeout ou excepção), assume-se que o nó falhou. Como só há comunicação entre nós adjacentes na rede, caso um sucessor receba uma mensagem de transferência de makefile ou atribuição de comandos de um nó que não é o seu predecessor, responder-lhe-á com uma mensagem de terminação.

Caso o coordenador que falhou esteja a executar algum makefile, é preciso que alguém recomece o seu trabalho. Para tal, existe uma cópia do makefile no seu sucessor e a indicação de qual o cliente que aguarda pelo resultado da execução. Sendo assim, quando for detectada a falha e após estabilização do sistema, o sucessor do coordenador em falha, recomeça a execução do makefile. No processo de estabilização do sistema, o antecessor e o sucessor do coordenador em falha trocam as seguintes mensagens:

1. O antecessor detecta a falha e envia uma mensagem para o sucessor, indicando que houve uma falha no nó entre eles. Esta mensagem inclui a indicação do novo antecessor do sucessor, bem como cópias dos seus makefiles em execução. 2. O sucessor do nó que falhou, por sua vez, indica ao antecessor deste qual o seu sucessor para que possa actualizar a lista de sucessores.

2.3.2 Servidores de Nomes

Entrada Ordeira: Para que um novo servidor de nomes se junte à rede, apenas precisa de conhecer o endereço de um dos já existentes, ao qual irá pedir o endereço do servidor central. Naturalmente que o primeiro servidor da rede é o que assume o papel de servidor de nomes central. Conhecendo-o, o servidor de nomes candidato envia-lhe uma mensagem de pedido de registo. O servidor central regista o novo servidor e anuncia a sua entrada a todos os outros. Depois de a alteração ter sido espalhada, ainda há mais um passo para que o servidor de nomes candidato esteja efectivamente pronto a receber pedidos: o servidor central deve enviar-lhe uma mensagem contendo o estado actual da base de dados de IDs de ficheiros. É possível que enquanto esta transferência se está a efectuar o servidor candidato receba pedidos de *lock* e actualização de versões de ficheiros. Deve responder afirmativamente aos pedidos de *lock* e guardar os pedidos de actualização, os quais executará, por ordem de chegada, depois de já ter a cópia local dos dados.

Saída Ordeira: Quando um servidor de nomes pretende abandonar a rede, envia uma mensagem com essa indicação ao servidor central e, pode então terminar a sua execução. No caso de o servidor de nomes central querer abandonar a rede, vai enviar uma mensagem para o primeiro servidor da sua lista, a indicar-lhe que será o novo servidor de nomes central.

Se um servidor de nomes quer sair enquanto está a pedir um *lock* ou quando já o tem, deve enviar mensagens de *unlock* a todos os servidores de nomes, antes de enviar a mensagem de terminação para o servidor central.

Saída por falha: Nos servidores de nomes, a detecção de falhas dá-se aquando da tentativa de *broadcast* de uma mensagem (novo ID ou *lock*) para todos os outros servidores de registados. Caso seja detectada uma falha (timeout ou excepção) num dos servidores aos quais se está a enviar a mensagem, deve-se retirar o seu endereço da lista e enviar uma mensagem para todos os outros servidores a instruir que tomem a mesma medida, para que o servidor em falha seja eliminado de todas as listas.

Para prevenir que uma mensagem proveniente de um servidor de nomes erradamente dado como "morto" coloque em causa a coerência do sistema, faz-se com que os servidores respondam a todas as mensagens conhecidas com uma terminação que instrui o servidor problemático a terminar a sua execução.

Um servidor de nomes que detecte uma falha num outro que tem um *lock* em curso deve libertá-lo e, além de retirar o endereço do servidor em falha da sua lista e de propagar esta alteração pelos outros servidores, adicionar à mensagem a indicação de que o *lock* pode ser libertado por todos.

2.3.3 Voluntários

Entrada: Quando um voluntário entra no sistema, tem de contactar um dos coordenadores, enviando uma mensagem de pedido de registo.

Saída Ordeira: O voluntário tem de avisar o coordenador que conhece de que vai abandonar o sistema. Desta forma, o coordenador sabe que já não pode enviar mais trabalho para o voluntário. Um voluntário só poderá abandonar o sistema após ter terminado todas as actividades. Contudo, assim que fica sinalizado para terminars já não aceitará mais trabalho.

Saída por falha: Na ligação entre voluntários e coordenadores, a detecção de uma falha num voluntário é feita pelo coordenador onde este se registou. Cada coordenador mantém uma lista com os endereços de todos os seus voluntários e existe uma falha caso haja timeout ou excepção num pedido a um voluntário. Se o coordenador receber mais alguma mensagem de um voluntário que não esteja na sua lista deve responder-lhe com uma mensagem de terminação.

Uma outra situação é a falha de um voluntário durante a execução de um comando. Para detectar o problema, o coordenador envia periodicamente mensagens de ping para cada um dos seus voluntários. No caso de ser detectada uma falha num coordenador que estivesse a executar um comando, este deve ser redistribuído para outro voluntário.

2.3.4 Repositório

Visto que o repositório é constituído por um anel de voluntários e clientes, as entradas, saídas e falhas são tratadas da mesma forma, exceptuando os casos que se explicam nas secções 2.3.3 e 2.3.5. Chama-se nó a um elemento deste anel, seja ele voluntário ou cliente. Como referido, o repositório assenta sobre o protocolo P2P Chord[1]. Para explicar como este lida com os novos nós e com as saídas (faltosas ou ordeiras) é preciso clarificar os conceitos de funções de estabilização. Sempre que o anel é modificado, é necessário estabilizá-lo. Considera-se estabilizado quando todas as listas de sucessores, os ponteiros para os antecessores e as *finger tables* de todos os nós estão correctas. Para que os nós conheçam sempre os endereços correctos e lidem com as entradas e saída, é necessário que regularmente corram a função de estabilização e actualização das *finger tables*, denominada *fixFingers*. Quando o nó V corre a função de estabilização, pergunta ao seu sucessor (S) qual é o seu predecessor (P) e, caso P não seja V, então o sucessor de V passa a ser P. Isto acontece quando P se juntou recentemente ao anel.

A função *fixFingers* é a forma como os novos nós iniciam a sua *finger table* e como os antigos incorporam os novos nós na sua tabela. Sempre que um nó corre esta função, também chama a *checkPredecessor* que serve para um nó detectar se o seu antecessor ainda está a funcionar. Caso tenha falhado, limpa o ponteiro e, dessa forma, quando o seu antecessor correr a função de estabilização, vai estar apto para actualizar o seu ponteiro para o antecessor.

Entrada: Quando um novo nó pretende entrar no anel, precisa de conhecer a localização de outro. Após decidido qual o seu ID no anel (através do hash do IP) é localizado o local onde vai entrar, encontrando o seu sucessor. Este local será entre o nó NP, com identificador IDp, e o nó NS, com identificador IDs, tal que IDs $<$ ID $<$ IDp, onde IDs e IDp são os identificadores numéricos mais próximos de ID, no anel. Após inserido o novo nó, é necessário que os restantes façam mudanças à sua *finger table*, de modo a estabilizar a rede. É para isso os nós correm regularmente a função de estabilização.

Saída Ordeira: Um nó V que vai sair deve notificar e e enviar os ficheiros de que é responsável para o sucessor. Além disso, V deve notificar o seu antecessor(P) e enviar-lhe o seu segundo sucessor, para que este possa actualizar a sua lista. Da mesma forma, S irá substituir o seu antecessor pelo antecessor de V, ou seja, P.

Saída por falha: Uma vez que só se considera uma falha até que o sistema estabilize, cada nó V irá manter uma lista de sucessores com duas entradas, uma para o seu sucessor e outra para o sucessor do seu sucessor. Sempre que V detectar a falha do seu sucessor(S1), precisa de estabilizar a sua lista de sucessores. Esta estabilização é feita pedindo ao seu segundo sucessor(S2) o endereço do seu sucessor(S3). Assim V actualiza a sua lista, de sucessores, com S2 e S3. S2, ao receber o pedido, actualiza o seu antecessor para o nó remetente (V). Este processo é executado correndo a função de estabilização e de actualização das *finger tables*.

2.3.5 Clientes

Saída Ordeira e por falha: A saída de um cliente, seja por falha ou ordeiramente, não implicará trabalho adicional. Caso haja uma execução de makefile, a seu pedido, o coordenador leva-a até ao fim, apesar de depois não conseguir notificar o cliente.

3. SIMPLIFICAÇÕES PARA AS FASES INTERMÉDIAS

3.1 Arquitectura Centralizada

Neste sistema usamos uma arquitectura baseada na P2P mas com algumas simplificações. O servidor de nomes possui todas as propriedades descritas anteriormente, mas como existe apenas um, não há replicação, nem o conceito de servidor de nomes central. Outra simplificação é o facto de apenas existir um repositório centralizado. Assim, não é necessário fazer lookUp pelo servidor responsável pelo armazenamento de um ficheiro.

O coordenador, ao receber o makefile executa os mesmos procedimentos. A diferença está no facto de haver apenas um coordenador e, como tal, no caso de haverem mais comandos do que voluntários, o coordenador continua a distribuição voltando ao seu primeiro voluntário, seguindo a mesma ordem, até não haver mais comandos para distribuir.

3.2 Arquitectura Replicada

A arquitectura replicada é muito semelhante à P2P, existindo múltiplos servidores de nomes e múltiplos coordenadores. A diferença é o facto de nesta arquitectura, como na centralizada, existir apenas um repositório centralizado, o que faz com que não seja necessário fazer lookUp pela localização dos ficheiros, acedendo-se directamente ao repositório. Naturalmente, a concorrência na escrita de novas versões será tratada apenas com *locks* ao nível das threads do servidor de nomes.

4. CONCLUSÃO

A solução aqui proposta para a resolução do problema de distribuição e execução paralela de makefiles no PADI@HOME, apresenta-se como uma alternativa simples e robusta para o modelo de falhas exigido, também tendo em conta que parte importante do sistema assenta sobre o protocolo Chord[1], de eficácia comprovada e que se encontra bem documentado.

5. REFERÊNCIAS

[1] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proceedings of the 2001 SIGCOMM conference*, 31(4):149–160, 2001.