MIF32 - Compte-Rendu du projet CAN

Adrien FAURE(p1006169) & RaphaAZENAVE-LEVEQUE (p1410942)

2014-2015

Abstract

Nous prntons une implntation en C/MPI d'une DHT utilisant le protocole CAN. Nos choix d'implntation permettent une recherche efficace dans la DHT depuis n'importe quel noeud. Nous offrons aussi des reprntations graphiques en SVG et textuelles de l't de la DHT ainsi qu'une couverture de nos fonctions le splus critiques gr s tests unitaires. Ces reprntations sont gr automatiquements lors de l'execution de la DHT, ou demande si l'on utilise l'interface en ligne de commande.

1 Fonctionalitisponibles

Les cinq pes dites dans le sujet on respect. A l'exception de la suppression des noeuds, jugrop complexe lors des sces de TP. Nous proposons cependant une solution fin de ce rapport.

1.1 Etape 1 : Crion du rau et insertion de noeud

Lors de l'execution de notre programme, le nombre de processus est statique, dni lors de l'appel commande *mpirun*. Le rang du processus root est dni par une macro. Celui-ous servira de coordinateur, et ne fera pas partie de la DHT. Le coordinateur demandera successivement aux noeuds de s'insr.

1.2 Etape 2 : Insertion de donn

Via le coordinateur (ou depuis n'importe quel noeud, car les modes sont griques), il est possible d'insr des donn e position *(x, y)* dans l'espace carten. M si dans notre projet, nous n'insns que des int, l'implntation sous-jacente permet de dnir le type des donn sr via un syst de TAG et en spfiant la taille. Les donn seront achemin, via notre algorithme de routage, au sein de l'overlay, sous forme d'un tableau d'octet.

1.3 Etape 3 : Recherche et lecture de donn

Pour effectuer la rption d'une donn une requ contenant le rank du demandeur et les coordon seront envoy dans l'overlay et acheminusqu'au noeud responsable de la zone correspondante. Le noeud responsable renverra, si elle existe, la donninsi que les mdonn fournies (type et taille) au demandeur. Il sera alors de la responsabilit l'utilisateur de les rncoder lors de la rption grace aux TAG et en utilisant la taille dans le cas d'un tableau par exemple.

1.4 Etape 4 : Suppression de noeud

Suite au retrait cette pe, nous proposons la solution suivante, que nous n'avons pas implnt Lors de la suppression d'un noeud, le noeud en question emmettra une reque s voisins pour drminer le plus apte rer une autre zone. Le noeud nouvellement en charge ne fuisionnera pas les zones, appliquera la logique de notre DHT sur chacunes de ces zones. Il serait alors possible d'exter de temps en temps un algorithme de romogisation des zones de la DHT. Il est amusant de constater qu'un m processus pourrait e en charge de zones non-juxtapos.

1.5 Etape 5 : Suppression de donn

La suppression d'une donnst similaire e recheche, si ce n'est qu'une fois achemin celle-era supprim

2 Choix de conception

2.1 Architecture

Notre projet propose l'architecture suivante. Une partie correspondant interface de la DHT. La suppression et l'ajout d'une donn Une partie correspondant logique de la DHT, elle m srn deux sous-parties. L'une est d la coordination (ajout, suppression de noeuds) et utilisar le processus coordinateur. Cependant son role est superficiel, car envoit l'ordre aux noeuds d'initier la procre d'insertion dans l'overlay. Le noeud ainsi sommmandera un point d'entrt une fois celui-btenu, il emettra une requete au point d'entrtenur pour rejoindre le rau. Cette procre permettrait de s'emanciper totalement du noeud coordinateur pour s'insr dans l'overlay. L'autre sous-partie est l'ensemble des traitements associ une requ se par les noeuds de l'overlay. La derni partie de notre architecture est compose toute la logique de manipulation des espaces cartens abtrait de l'utilisation d'MPI. Cette partie nt source d'erreur, elle est en majeure partie couverte par des tests unitaires.

2.2 Communications

Les processus non-coordinateurs seront en attente, en premier lieu de l'ordre de s'insr du coordinateur. Ils attendent ensuite de recevoir et de traiter une requ. Les noeuds seront en mesure, lors d'une rption, de traiter les requs gr au syst de tag proposr MPI.

2.3 Journalisation et debug

La complexit code augmentant, nous avons mis en place plusieurs systs afin de comprendre le comportement de notre logiciel. En premier lieu, nous avons mis en place la journalisation des manipulations de l'overlay via le coordinateur. Le coordinateur dispose de routine afin de demander des informations aux noeuds de l'overlay. Les informations rpes sont stock dans un rrtoire de logs sous deux formes. Une forme textuelle, et une forme graphique au format SVG. Nous utilisions beaucoup l'"aloire", la reproductibilitait donc impossible, donc nous avons mis en place un syst de seed paramable. Celle-st automatiquement journalisans le fichier *logs/global.txt* et peut e spfiors de l'execution du programme. De plus nous pouvons spfier que nous souhaitons lancer le programme et manipuler la DHT grace prompt.

3 Tests unitaires

Au cours du projet, il est devenu nssaire pour nous d'ire des tests unitaires afin d'accrer le dggage et d'assurer le bon fonctionnement de notre code. Nous avons donc isols zones de codes comportant nos algorithmes des zones utilisants MPI, cela nous a permit d'obtenir une bonne couverture du code par les tests, et d'e relativement sr du comportement de notre code.

4 Algorithme de routage

5 Relocation des donn

6 Les structures de donn mises en oeuvre

Chaque noeud poss une liste capable de stocker les donn dont le noeud est responsable. Ces donn sont encapsul dans une structure afin de pouvoir y conserver la position des donn.

Les noeuds possnt une liste de frontis reprntant le voisinage proche du noeud.

7 Ajout dynamique de processus lors de l'execution

Nous avons souhaitouter la possibilitajouter des processus au cours de l'execution. Nous avons effectus essais en utilisant *MPI_Spawn*, puis *MPI_Intercomm_merge* pour obtenir un communicateur intra contenant tous les processus ainsi que le nouveaux processus. Malheureusement, il n'est pas possible de re-merger ce communicateur [1] lors de spawn successifs, car *MPI_Intercomm_merge* est une routine collective et donc chaque processus prdnt spawnit participer au prochain merge.

A Annexe 1 - README

```
1 # pour builder et tester:
2
3     make
4
5
6 # Pour builder et executer les tests
7
8     make runtest
9
10
11 # Pour builder sans executer les tests
12
```

[1] Nous avons du abandonner cette iduite lecture de ce topic de la mailling-list d'openMPI - https://www.open-mpi.org/community/lists/users/2007/10/4312.php

```
13      make test
14
15
16  # pour builder l'appli sans executer les tests  :'(
17
18      make mpi_can
19
20
21  # execution avec prompt :
22
23      mpirun -np 7 ./mpi_can debug
24
25
26  # execution sans prompt :
27
28      mpirun -np 7 ./mpi_can
29
30
31  # Execution with yield
32      mpirun --mca mpi_yield_when_idle 1 -np 10 mpi_can
33
34  # execution avec prompt et en fixant une seed  s p cifique  au   g  nteur   pseudo- a l a
35
36      mpirun -np 7 ./mpi_can seed 42 debug
37
38
39  # execution sans prompt et en fixant une seed  s p cifique  au   g  nteur   pseudo- a l a
40
41      mpirun -np 7 ./mpi_can seed 42
42
43
44  # Prompt:
45
46      -- 9 nodes availables --
47      > status              : show log about the state of the DHT
48      > insert 2            : insert the node 2 in the overlay
49      > insert all          : insert all nodes in the overlay and run etape 3
50      > log                 : add a textual/SVG log on logs/ directory
51      > put <x> <y> <data> : put <data> in position (x, y)
52      > get <x> <y>         : get a data from position (x, y)
53      > shuffle <nb>        : randomly insert <nb> random data
54      > rm <x> <y>          : remove a data in position (x, y)
55      > etape3 <x>          : insert <x> element and try to retrieve the 5 last and f
```