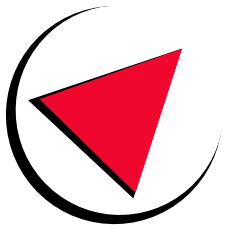

L I F C

Université de Franche-Comté

Mémoire Master 2 Recherche

DST - Distributed Spanning Tree : Une topologie pour la recherche de services de calcul sur la grille

par Christophe Enderlin

le 20 octobre 2012

encadré par

Laurent Philippe ⋆ Professeur à l'Université de Franche-Comté

soutenu le 30 septembre 2011 *devant le jury*

Jean-Marc Nicod ⋆ Professeur à l'École Nationale Supérieure de Mécanique et des Microtechniques de Besançon

David Laiymani ⋆ Maître de Conférence à l'Université de Franche-Comté

Remerciements

Me voici arrivé au terme d'une aventure (le mot n'est pas trop fort !) commencée à Besançon, en octobre 2005. Ces six années ont été riches en expériences, bonnes et moins bonnes, en émotions et en enseignements de toutes sortes, tant dans mes vies familiales et professionnelles que d'étudiant. Pouvoir reprendre des études à mon grand âge (!) et renouer avec le plaisir d'apprendre est une fabuleuse opportunité et je voudrais ici exprimer ma grande reconnaissance envers toutes les personnes qui ont rendu cette reprise d'études possible.

Je veux donc remercier Isabelle Jacques, Eveline Renard et sa remplaçante Stéphanie Jubin, pour leur accueil, leur rigueur et leur professionnalisme. Je voudrais particulièrement remercier Sylvie Damy pour sa compétence, sa gentillesse et son dévouement.

Je veux aussi remercier l'ensemble des enseignants qui ont accepté de jouer le jeu difficile de l'enseignement à distance et qui ont dû supporter nos centaines de questions ! (Pardon à Mmes Greffier et Damy que nous avons traumatisées avec nos questions ! ☺) Étudier seul chez soi n'est pas toujours facile et peut être décourageant par moment ; j'ai donc apprécié à sa juste valeur les efforts déployés pour nous par ces enseignants.

Je profite de cet espace pour remercier mes fidèles compagnons de route, Anne Alvarez et Jean-Luc Joly, pour les heures passées à travailler ensemble, les mails encourageants et les bons restaus ! Un grand merci aussi aux autres étudiants qui ont participé à la très bonne ambiance des forums.

Des remerciements spéciaux pour ma famille qui a dû me supporter pendant ce périple, tout particulièrement ma femme qui n'a jamais cessé de m'encourager et pour sa relecture de ce rapport ! ☺

Un grand merci également à l'équipe de Simgrid, en particulier Arnaud Legrand et Martin Quinson, pour leur sympathie et l'aide qu'ils m'ont apporté en répondant à mes questions.

Pour finir, je souhaite remercier Laurent Philippe, mon encadrant de stage, pour sa gentillesse, sa disponibilité et sa compétence et pour m'avoir, avec Bénédicte Herrmann, proposé un sujet si intéressant. Je considère comme un privilège d'avoir pu travailler avec lui pendant ce stage.

Table des matières

1 Retrait de sommets d'un DST (suite des travaux) III
1.1 Transfert III

Chapitre 1

Retrait de sommets d'un DST (suite des travaux)

1.1 Transfert

Algorithme 1 : Transfert de nœuds du groupe courant vers un groupe appelant

```
1:  procedure TRANSFERT(st, right, cut_pos)
2:      if (right = 1) then
3:          start ← cut_pos
4:          end ← size(me.brothers[stage]) − 1
5:          answer.stay_id ← me.brothers[stage][cut_pos − 1].id
6:      else
7:          start ← 0
8:          end ← cut_pos
9:          answer.stay_id ← me.brothers[st][cut_pos + 1].id
10:     end if

11:     for i ← start to end do
12:         answer.rep_array[i − start] = me.brothers[st][i]
13:     end for

14:     BROADCAST(me, st, cut_node(st, right, cut_pos))

15:     return answer
16: end procedure
```

Algorithme 2 : Scinde un nœud lors d'un transfert

```
1: procedure CUT_NODE(stage, right, cut_pos)
2:     pos_me ← index(me.brothers[stage], me.id)

3:     if (right = 0) then
4:         start ← 0
5:         end ← cut_pos
6:         new_node ← me.brothers[stage][cut_pos + 1]
7:     else
8:         start ← cut_pos
9:         end ← size(me.brothers[stage]) − 1
10:        new_node ← me.brothers[stage][0]
11:    end if

12:    if (pos_me ≥ start and pos_me ≤ end) then
13:        SHIFT_BRO(stage + 1, new_node, right)
14:    end if

15:    if (right = 0) then
16:        if (pos_me ≤ cut_pos) then
17:            start ← cut_pos + 1
18:            end ← size(me.brothers[stage] − 1)
19:        else
20:            start ← 0
21:            end ← cut_pos
22:        end if
23:    else
24:        if (pos_me < cut_pos) then
25:            start ← cut_pos
26:            end ← size(me.brothers[stage] − 1)
27:        else
28:            start ← 0
29:            end ← cut_pos − 1
30:        end if
31:    end if

32:    DEL_MEMBER(stage, start, end)
33: end procedure
```

Algorithme 3 : Supprime une partie du groupe courant à un étage donné

```
1: procedure DEL_MEMBER(stage, start, end)
2:     nb_del ← end − start + 1
3:     if (nb_del = 0) then
4:         return
5:     end if
                                  ▷ mémorise les nœuds à effacer avant de commencer
6:     for i ← 0 to nb_del − 1 do
7:         id_del[i] ← me.brothers[stage][start + i].id
8:     end for
                                                          ▷ boucle d'effacement
9:     for i ← 0 to nb_del − 1 do
10:        if (id_del[i] <> me.id) then                   ▷ ne pas effacer 'moi'
11:            pos2del ← index(me.brothers[stage], id_del[i])

12:            if (pos2del < size(me.brothers[stage] − 1)) then
13:                for j ← pos2del to size(me.brothers[stage] − 1) do
14:                    me.brothers[stage][j] ← me.brothers[stage][j + 1]
15:                end for
16:            else
17:                j ← pos2del
18:            end if

19:            me.brothers[stage][j].id ← −1
                                  ▷ 'moi' n'est plus prédécesseur de id_del[i]
20:            SEND_MSG_ASYNC(id_del[i], del_pred(stage, me.id))
21:        end if
22:    end for
23: end procedure
```

Algorithme 4 : Décale les membres du groupe pour en accueillir un nouveau à la position de '*moi*'. Le membre en trop est détruit.

```
1:  procedure SHIFT_BRO(stage, new_node, right)
                ▷ s'assure de n'exécuter cette fonction qu'une fois (elle est diffusée par cut_node)
2:      pos_new_node ← index(me.brothers[stage], new_node.id)
3:      if (pos_new_node > −1) then
4:          return
5:      end if

6:      pos_me ← index(me.brothers[stage], me.id)

7:      if (right = 1) then
8:          if (pos_me < b) then
9:              lost_id ← me.brothers[stage][pos_me + 1].id
10:             me.brothers[stage][pos_me + 1] ← me.brothers[stage][pos_me]
11:         else
12:             Affiche un message d'erreur et stoppe la procédure
13:         end if
14:     else
15:         if (pos_me > 0) then
16:             lost_id ← me.brothers[stage][pos_me − 1].id
17:             me.brothers[stage][pos_me − 1] ← me.brothers[stage][pos_me]
18:         else
19:             Affiche un message d'erreur et stoppe la procédure
20:         end if
21:     end if

22:     me.brothers[stage][pos_me] ← new_node

                                          ▷ 'moi' doit être un prédécesseur de new_node
23:     SEND_MSG_ASYNC(new_node.id, add_pred(stage, me.id))
                                          ▷ 'moi' ne doit plus être un prédécesseur de lost_id
24:     SEND_MSG_ASYNC(lost_id, del_pred(stage, me.id))
25: end procedure
```

Algorithme 5 : Procédure exécutée par un nœud qui quitte le DST

```
1: procedure LEAVE( )
                                          ▷ travaille sur des copies des tables de routage
2:     cpy_brothers ← me.brothers
3:     cpy_preds ← me.preds

4:     for stage ← 0 to height(me.brothers) − 1 do
                                          ▷ INFORME MES PRÉDÉCESSEURS DE MON DÉPART
5:         for pred ← 0 to size(cpy_preds[stage]) − 1 do

6:             if (cpy_preds[stage][pred].id ≠ me.id) then

7:                 if (stage = 0) then
8:                     SEND_MSG_ASYNC(cpy_preds[stage][pred].id, del_bro(stage, me.id))
9:                 else

10:                    new_rep_id ← un de mes frères de l'étage 0 choisi aléatoirement
                                          ▷ en informe le prédécesseur
11:                    if (new_rep_id ≠ me.id) then

12:                        SEND_MSG_ASYNC(cpy_preds[stage][pred].id,
                               repl_bro(stage, new_rep_id))
13:                    end if
14:                end if
15:            end if
16:        end for                                          ▷ prédécesseur suivant

                                          ▷ INFORME MES FRÈRES DE MON DÉPART
17:        for brother ← 0 to size(cpy_brothers[stage]) − 1 do

18:            if (cpy_brothers[stage][brother].id ≠ me.id) then

19:                SEND_MSG_ASYNC(cpy_brothers[stage][brother].id,
                       del_pred(stage, me.id))
20:            end if
21:        end for                                          ▷ frère suivant
22:    end for                                              ▷ étage suivant

                ▷ CHARGE UN DE MES FRÈRES DE TRAITER LES FUSIONS OU TRANSFERTS
23:    if (size(me.brothers[0]) ≤ a) then

24:        idx ← index d'un de mes frères de l'étage 0
25:        SEND_MSG_SYNC(me.brothers[0][idx].id, merge_req())
26:    end if
27: end procedure
```

Algorithme 6 : Traite les fusions ou transferts rendus nécessaires par un départ

```
1: procedure MERGE_REQUEST( )

2:     stage ← 0
3:     size_last_stage ← 0
                                                  ▷ parcourt l'ensemble des étages
4:     while (size(me.brothers[stage]) < a) and stage < height(me.brothers)−1) do

5:         pos_contact ← merge_or_transfer(me, stage)

6:         if (pos_contact > −1) then                  ▷ UNE FUSION EST POSSIBLE

7:             transfer ← 0
8:             pos_me ← index(stage + 1, me.id)

                                  ▷ demande au contact de réaliser une première fusion . . .
9:             if (pos_me > pos_contact) then
10:                right ← 11
11:            else
12:                right ← 10
13:            end if

14:            SEND_MSG_SYNC(me.brothers[stage + 1][pos_contact].id,
                   merge(stage, pos_me, pos_contact, right, me.brothers[stage]))

                                  ▷ . . .puis lui demande de diffuser une tâche de fusion
15:            if (pos_me > pos_contact) then
16:                right ← 1
17:            else
18:                right ← 0
19:            end if

20:            SEND_MSG_SYNC(me.brothers[stage + 1][pos_contact].id,
                   broadcast_merge(stage, pos_me, pos_contact, right, me.brothers[stage]))

21:                ▷ après la fusion, l'étage supérieur contient deux représentants du même groupe

22:            CLEAN_UPPER_STAGE(stage, pos_me, pos_contact)      ▷ exécution locale
                                                                   ▷ diffusion
23:            BROADCAST(me, stage+1, clean_upper_stage(stage, pos_me, pos_contact))

24:        else
```

```
25:                                          ▷ FUSION IMPOSSIBLE - IL FAUT FAIRE UN TRANSFERT
26:         pos_me_up ← index(stage + 1, me.id)
27:         if (pos_me_up = 0) then
28:             pos_contact ← 1
29:             right ← 0
30:             cut_pos ← b − a − 1
31:         else
32:             pos_contact ← pos_me_up − 1
33:             right ← 1
34:             cut_pos ← a
35:         end if
36:         contact_id ← me.brothers[stage + 1][pos_contact].id
37:         answer ← SEND_MSG_SYNC(contact_id,
                          transfer(stage, right, cut_pos, me.id))
38:                                                  ▷ AJOUTE LES NŒUDS REÇUS
39:         current_bro ← me.brothers[stage]          ▷ sauvegarde l'étage courant
40:                                  ▷ chaque membre de la branche courante ajoute les nœuds reçus
41:         BR_ADD_BRO_ARRAY(stage, answer.rep_array, mod((right + 1), 2)
42:                                  ▷ chaque membre de la branche voisine ajoute les nœuds courants
43:         SEND_MSG_SYNC(answer.rep_array[0].id,
                br_add_bro_array(stage, current_bro, right))
44:                                                  ▷ nettoyage de l'étage supérieur
45:                                                  ▷ exécution locale
46:         UPDATE_UPPER_STAGE(stage, pos_contact, answer.stay_id)
47:                                                  ▷ diffusion
48:         BROADCAST(me, stage + 1,
                update_upper_stage(stage, pos_contact, answer.stay_id))
49:     end if
50:     stage + +                                    ▷ étage suivant
51: end while
```

52: ▷ TRAITEMENT DE LA RACINE

53: **end procedure**