Réseaux Datacenters/HPC

TP - Simulation de topologie et analyse des performances

CEA/ENSIIE – 2017-2018

11 avril 2018

Résumé

Le premier objectif de ce TP est de prendre en main les outils capables de simuler des réseaux de type datacenters/HPC. Le second objectif est d'être capable d'interpréter les métriques extraites des différents outils dans le but d'analyser les performances en fonction des topologies ainsi que d'étudier les mécanismes de routage mises en place.

Table des matières

1 Notation et rendu

Comme indiqué en cours, ce module sera noté en grande partie sur le rendu du TP mais aussi sur votre participation. Les éléments de notation qui seront particulièrement observés :

1. La qualité rédactionnelle : la forme et le fond
2. La qualité des schémas
3. La pertinence des réponses
4. Tous les éléments de réflexion personnelle démontrant l'acquisition des connaissances

Les rapports peuvent être faits à maximum 2. Tout nom manquant sur le rendu aura automatiquement 0. Si vous le faites à 2, il n'y a qu'un seul rendu attendu.

1.1 Le rendu

Le rendu final sera une archive (zip) contenant les éléments suivants :

1. Le rapport
2. Les scripts écrits
3. Les graphiques (on limitera à maximum 2MB par graphique, si vous en avez des plus lourds, ne les mettez pas dans l'archive)

La transmission du rendu, vous devez envoyer par mail l'archive au format NOM1_NOM2.zip à l'adresse damien.gros@cea.fr.

Vous avez jusqu'au lundi 14 Mai 19h00 pour envoyer vos rendus.

Tout retard dans le rendu entrainera la note de 0 aux personnes du groupe. En cas de problème, n'attendez pas la dernière minute pour nous en faire part.

1.2 Barème des questions

Le barème est indicatif et pourra être modifié. Les réponses attendues doivent être concises et précises.

— Une ligne de commande : 1 point
— Un schéma fait : 1.5 points
— Les analyses, les réflexions : entre 2 points et 4.

2 Apprentissage

Rappels des différents outils utilisés :
— CreateIBNet.py [Hoefler, 2013]
— res.py
— ibsim
— openSM
— un ensemble de scripts :
  — ibnetdiscover
  — dump_lfts
  — nodeset [CEA, 2017]
— utilParser.py

Les outils ibsim, openSM, ibnetdiscover et dump_lfts sont fournis par Open Fabrics [Open Fabrics, 2017].

Dans cette première partie, nous allons nous intéresser au fonctionnement des outils. L'objectif est donc d'apprendre à les utiliser pour pourvoir, par la suite, analyser l'impact sur les performances des différences topologies ainsi que des algorithmes de routage.

Pour ce faire, nous allons partir sur un exemple simple : un **Fat Tree**, de niveau 3 avec 8 nœuds composé de 4 ilots de 2 nœuds chacun.

Ex. 1 — Prise en main des outils

1. Qu'est-ce qu'une topologie de type Fat Tree ? Quels sont ces principales caractéristiques ?
2. Combien faut-il de switchs pour interconnecter tous les nœuds de calcul ?
3. Sur un schéma, représenter le résultat attendu

2.1 Connexion à la machine virtuelle

Pour réaliser la partie pratique, nous allons utiliser une machine virtuelle préparée spécifiquement pour ce TP. Elle est déjà en place sur le cluster.

La machine virtuelle que l'on va utiliser est dans le dossier : `TP1_M1`.

Le contenu du fichier `.pcocc/templates.yaml`.

```
ensiie:
  resource-set: ens-cluster
  image : /home/grosd/VM/M1_TP1
```

Pour vous connecter, soit vous définissez un utilisateur grâce à l'instruction `user-data` en générant une bi-clé pour SSH, soit vous vous connectez en **root** avec le mot de passe : root.

Pour lancer la machine, utilisez la commande suivante :

```
pcocc alloc -c 1 ensiie:1
```

2.2 Génération de la topologie

Une fois cette première étape réalisée, nous allons générer la topologie grâce à l'outil **createIBNet.py**.

Pour ce faire, déplacez-vous dans le dossier nommé ENSIIE (transmis par l'intervenant). Ce dossier doit contenir les éléments suivants ;

— createIBNet.py

— res.py
— utilParser.py

Lancer le script python de la manière suivante :

```
root@host:# python createIBNet.py --help
```

Listing 1 – Affichage de l'aide de l'outil createIBNet.py

Vous devriez obtenir la sortie suivante :

```
Program to create an input file for the InfiniBand Network Simulator (ibsim)
Usage:
  createIBNet [-o <outputfile>] [-t <topology>]
              [-n <#endpoints>] [-np <#ports_per_endpoint>]
              [-s <#switches>] [-sp <#ports_per_switsch>]
              [-d[1|2|3] <size in i. dimension>]
              [-t[k|n] <k,n for k-ary-n-Tree> ]
              [-k[b|n] <b,n for Kautz graph K(b,n)> ]
              [-x[h|m|w] <h,m,w for XGFT> ]
              [-f[a|p|h] <a,p,h for Dragonfly> ]

    -o    name of the output file
    -i    name of existing topology file to load and modify
    -rid  name of existing rootguid file for existing topology
    -t    the topology (default = 2D-Mesh) of the InfiniBand Network
                [ FatTree | k-ary-n-Tree | 2D-Mesh | 3D-Mesh |
                  2D-Torus | 3D-Torus | Kautz | XGFT | Random |
                  Dragonfly | Cascade | Tofu | Taurus | MMS | load]
    -n    number of endpoints (Hca) in the network (default = 8)
    -np   number of ports for each endpoint (default = 1) [ 1 | 2 ]
    -s    number of switches (default = 4)
    -sp   number of ports for each switch (default = 4)
    -d1   size of the 1. dimension for Mesh / Torus / Tofu
    -d2   size of the 2. dimension for Mesh / Torus / Tofu
    -d3   size of the 3. dimension for Mesh/Torus (only when 3D) and Tofu
    -ml   enable multi-link configuration between switches for Mesh/Torus/Kautz/
   MMS
    -tk   k (k-ary-n-Tree) is half the number of ports for each switch
    -tn   n (k-ary-n-Tree) is the number of levels in the tree
    -kb   b for Kautz K(b,n) is the base of the Kautz string
    -kn   n for Kautz K(b,n) is the length of the Kautz string
    -xh   h for XGFT(h,m,w) is the height of the tree
    -xm   m for XGFT(h,m,w): a comma separated list without whitespaces
    -xw   w for XGFT(h,m,w): a comma separated list like -xm (e.g. 2,3,2)
    -cl   connection links for Random: number of links between switches
    -rs   seed for Random
    -fa   a for Dragonfly(a,p,h) is the number of routers in each group
    -fp   p for Dragonfly(a,p,h) is the number of HCA connected to each router
    -fh   h for Dragonfly(a,p,h) is the number of links within each router to
   connect to other groups
    -fg   g for Dragonfly(a,p,h) is a optional number of groups; w/o we use a*h
   +1 groups
    -cg   g for Cascade(96,8,10) is a number of groups
    -cb   global links between 2 groups for Cascade(96,8,10); min: 4, max: a*h /
    (g-1)
```

Listing 2 – Paramètres de l'outil createIBNet.py

On peut noter que ce script est capable de générer un nombre important de topologies réseaux et qu'il possède un certain nombre de valeurs par défaut, qu'il faudra faire attention à bien modifier pour obtenir les résultats voulus par la suite.

Comme dit précédemment, nous allons générer un **Fat Tree** avec 8 nœuds.

```
root@host:# python createIBNet.py -t FatTree -n 8 -xh 3 -xm 2,2,2 -s 7 -sp 3
```

Listing 3 – Génération d'un FatTree

Explication des commandes utilisées :

```
-t FatTree -n 8 -xh 3 -xm 2,2,2 -s 7 -sp 3
```

Listing 4 – Explication des paramètres

- le paramètre "-t" permet de spécifier la topologie que l'on veut générer : ftree
- le paramètre "-n" permet de spécifier le nombre de nœuds finaux : 8
- le paramètre "-xh" permet de spécifier la hauteur de notre arbre : 3
- le paramètre "-xm" permet de définir le nombre de fils que doit avoir (au maximum) chaque switch de notre topologie : 2 pour chacun des swtichs. Comme notre arbre a une hauteur de 3, il est nécessaire de spécifier 3 valeurs même si elles sont identiques.
- le paramètre "-s" permet de spécifier le nombre de switchs de notre topologie : 7
- le paramètre "-sp" permet de spécifier le nombre de ports par switchs : 3

`Nota Bene` : il est parfois nécessaire de spécifier un nombre de ports petits sur les switchs pour obtenir la topologie voulue.

Une fois le script exécuté, on peut vérifier que les paramètres donnés ont bien été pris en compte en analysant la sortie de l'outil (et qu'ils correspondent bien à ce que l'on voulait obtenir)

```
python createIBNet.py -t FatTree -n 8 -xh 3 -xm 2,2,2 -s 7 -sp 3
Selected configuration:
  Output:  /home/Damien/net.txt
  Topology:  FatTree
  Number of endpoints:  8
  Number of ports per endpoints:  1
  Number of switches:  7
  Number of ports per switch:  3
  Dimensions for Mesh/Torus: -1, -1, -1
  k-ary-n Tree: -1 -ary- -1
  Kautz K(b,n): K(-1; -1)
  XGFT(h,m,w): XGFT(3; 2,2,2; -1)
  Dragonfly(a,p,h) : Dragonfly(-1, -1, -1) with -1 groups
  Cascade(a,p,h,g) : Cascade(96, 8, 10) with -1 groups
  Tofu(d1,d2,d3) : Tofu(-1, -1, -1) with -1 groups

This program also writes an dot-file to plot the graph with Graphviz.
Try:
  dot / neato net.dot -Tpng -o graph.png && evince graph.png

Finish!
```

Listing 5 – Exemple de sortie de l'outil createIBNet.py

2 fichiers ont été générés (avec les noms par défaut si vous n'avez pas modifié cette option) :

1. net.txt
2. net.dot

On notera que le script génère un fichier au format .dot, interprétable par **graphviz**, permettant de générer de manière graphique le résultat. Faites-le et vérifier que le résultat obtenu correspond bien au schéma que vous aviez réalisé à la question 2.

```
dot  net.dot -Tpng -o graph.png && evince graph.png
```

Listing 6 – Génération du graph de la topologie

NB : si **evince** n'arrive pas à afficher le fichier généré, vous pouvez le faire grâce à la commande **eog**.

2.3 Création des groupes

Comme nous construisons notre topologie dans un simulateur, nous allons aussi devoir créer les ilots (de calcul, de stockage, de services, etc.).

Les informations des ilots sont renseignées dans le fichier **nodes**. Dans une première approche, nous allons vouloir nous placer dans le cas le plus favorable. A partir de la topologie générée et du graphique, renseignez le fichier **nodes**. Pour vous aider, vous pouvez trouver un exemple 7.

On définit :
— cas favorable : les groupes sont sur les mêmes leafs
— cas défavorable : les groupes sont répartis de manière aléatoire sur les leafs

ATTENTION : il sera peut-être nécessaire d'adapter à votre topologie.

```
@stockage Hca[0,1]
@calcul Hca[2,3]
@services Hca[4,5]
@autres Hca[6,7]
```

Listing 7 – Exemple de groupe de noeuds

2.4 Chargement de la topologie

Nous allons maintenant charger la topologie générée dans le simulateur IB. Le fichier contenant la topologie s'appelle `net.txt`.

```
ibsim -s net.txt
```

Listing 8 – Exemple d'utilisation d'ibsim

Le paramètre "-s" indique à ibsim qu'il doit activer la topologie.

Une fois la topologie chargée, nous allons pouvoir utiliser l'outil openSM (open Subnet Manager pour les réseaux InfiniBand).

On peut laisser openSM détecter/choisir un routage, mais on peut aussi le lui imposer.

```
export LD_PRELOAD=/usr/lib64/umad2sim/libumad2sim.so
opensm -R ftree -f stdout
```

Listing 9 – Exemple d'utilisation d'openSM

Dans ce cas, le paramètre "-R" permet d'imposer le routage.

Il faut noter que l'on peut donner à openSM un routage non adapté à la topologie, ou qu'il ne connait pas, dans ces deux cas, il choisira par défaut un algorithme de routage pour la topologie donnée.

Vérifier, grâce aux informations fournies par openSM que le routage que vous avez choisi est bien le **ftree**.

Une partie des informations est affichée sur le sortie standard, mais des informations peuvent manquer. Pour avoir l'ensemble des informations, il faut se référer au fichier `/var/log/opensm.log` (par défaut).

Ex. 2 — OpenSM

1. Quelles sont les informations fournies par l'outil **openSM** lors de son lancement sur la topologie ?

2.5 Dump des informations

Maintenant, nous allons récupérer les informations sur la topologie que nous venons de créer. Le but étant de pouvoir exploiter ces résultats par la suite.

Note importante : pour pouvoir utiliser correctement les outils de simulation et assurer l'interaction entre eux, il est nécessaire d'exporter une libraire (en LD_PRELOAD) pour qu'elle soit chargée par les outils. Pour cela, il suffit de taper la ligne suivante :

```
export LD_PRELOAD=/usr/lib64/umad2sim/libumad2sim.so
```

Listing 10 – Export LD_PRELOAD

Si des résultats de **ibnetdiscover** (en terme de nombres de switch/noeuds) ne sont pas conformes à ce que vous avez chargé dans ibsim, il est fort probable que vous n'ayez pas tapé l'export avant d'exécuter le programme (ou que vous avez changé de *shell*).

Le premier outil que l'on va utiliser est **ibnetdiscover**. Comme son nom l'indique, il permet d'afficher des informations relatives au réseau IB sur lequel on se trouve.
Lancer simplement *ibnetdiscover*.

Ex. 3 — IBnetDiscover Part 1

1. Quelles sont les informations fournies par la commande **ibnetdiscover** ?

```
1 ibnetdiscover > topo
```

Listing 11 – Extraction de la topologie avec ibnetdiscover

On va sauvegarder les informations dans un fichier nommé `topo`.

Ensuite, on va uniquement lister les nœuds connectés (on peut générer des topologies avec des nœuds qui sont non-connectés)

```
1 ibnetdiscover -l > hosts
```

Listing 12 – Génération des noeuds du réseau avec ibnetdiscover

On va sauvegarder les informations dans un fichier nommé `hosts`.

Ex. 4 — IBnetDiscover Part 2

1. Quelles sont les informations fournies par la commande **ibnetdiscover -l** ? Quelles sont les différences avec la commande **ibnetdiscover** ?

La prochaine étape consiste à récupérer les tables de routage de chaque switch de notre réseau. Pour cela, on va utiliser la commande **dump_lfts**.

```
dump_lfts.sh > ltfs
```

Listing 13 – Récupération des tables de routages des switchs grâce à la commande dump_lfts

Ex. 5 — Table de routage

1. En vous aidant d'un exemple concret, décrivez la table de routge d'un switch.

A la fin de ces différentes manipulations, vous devriez avoir dans le dossier courant les éléments suivants :

— `topo`
— `hosts`
— `lfts`
— `nodes`

Créer un dossier, nommé par exemple `exo1` et déplacer ces 4 fichiers à l'intérieur.

Copiez aussi les fichiers **utilParser.py** et **res.py** dans ce même dossier.
Déplacez-vous dans le dossier `exo1`. Editez le fichier **res.py**. La variable DIR contient le dossier, relatif au dossier de travail courant, qui va accueillir une partie des résultats que l'on va générer.

Exécuter simplement ce script python :

```
python res.py
```

Listing 14 – Génération des métriques

A l'intérieur du dossier `exo1`, un dossier nommé `test` a été créé et contient les résultats des différentes analyses réalisées par le script.

Dans le but d'avoir une vision plus claire des résultats générés, rapatriez les 2 fichiers ayant pour préfixe **test_** sur votre station et ouvrez les avec un tableur (LibreOffice, OpenOffice, MsOffice).
NB : pour une lecture facilitée, lors de l'import du fichier, choisissez comme séparateur de colonne "|".

Les commentaires des fichiers se feront par l'intervenant.

En modifiant le fichier `nodes`, comme décrit à la section 2.3, créer des groupes de nœuds non efficients et relancer l'outil 14.

Ex. 6 — Premières modifications et interprétations

1. Comment établir un regroupement non efficient ?
2. Sur un schéma, montrez la répartition des ilots au sein de la fabrique.
3. Quelles sont les différences que vous pouvez observez au niveau des métriques ?
4. Comment expliquez-vous ces différences ?

3 Génération de topologies

Comme on l'a vu dans le listing 2, l'outil *createIBNet* offre la possibilité de générer un certain nombre de topologies, très différentes les unes des autres.

OpenSM est le gestionnaire de sous-réseau. Il est capable de déterminer, en découvrant la topologie, quel algorithme de routage est le plus adapté.

A partir des exemples vu en cours, donnez les commandes pour générer les topologies suivantes :

Ex. 7 — Génération de topologies et visualisation

1. Tor 4-ary 1-cube : quel algorithme de routage est choisi par OpenSM (par défaut) ? Détaillez la ligne qui vous a permis de générer cette topologie.
2. Tor 4-ary 2-cube : quel algorithme de routage est choisi par OpenSM (par défaut) ? Détaillez la ligne qui vous a permis de générer cette topologie.
3. Un dragonfly quelconque : quel algorithme de routage est choisi par OpenSM (par défaut) ? Détaillez la ligne qui vous a permis de générer cette topologie.
4. Pour chaque topologie générée, visualisez la avec l'outil graphviz et intégrez le résultat dans le rapport final.

4 Changement de routage et de topologie

Dans le cours, nous avons vu un certain nombre de topologies différentes. Certaines sont très théoriques, d'autres sont plus pratiques et se définissent comme des dérivations de modèles déjà existants.

Dans cette partie, nous nous proposons d'étudier un dérivé du FatTree : le **eXtended General Fat Tree** ou XGFT [Öhring et al., 1995].

L'un des principales problèmes générés par les Fat Tree est la mauvaise diversité des chemins. Le second exemple présenté dans le cours est une première solution à ce problème mais il en existe d'autre.

4.1 Génération d'un XGFT

Commencez par générer une topologie simple en vous aidant de l'aide de l'outil **createIBNet** comme décrit au listing 2.

Ex. 8 — Topologie XGFT

1. Décrivez cette topologie.
2. Caractérisez cette nouvelle topologie (en terme de diversité des chemins, répartition de la charge sur les liens, etc.)

Tout comme vous l'avez fait pour Fat Tree, chargez cette topologie dans le simulateur **ibsim** et lancez **openSM** sans aucune option de routage.

Ex. 9 — Description de la topologie

1. Par défaut, quel est l'algorithme de routage choisi ?
2. Est-ce que les informations de topologie données par **openSM** sont correctes vis-à-vis de ce que vous avez généré et que vous observez ? Recommencez en imposant le Fat Tree comme algorithme de routage.

Nous allons étudier les conséquences de cette nouvelle topologie sur le routage et la congestion de la fabric. Pour ce faire, nous allons reprendre la méthodologie de la première partie.

Générez les ilots les plus *favorables* (dans le fichier `nodes`). Refaites l'ensemble des opérations vu dans la section II pour générer les métriques.

Ex. 10 — Comparaison dans une bonne répartition avec Fat Tree

1. En comparant avec le Fat Tree fait en exemple, pouvez-vous caractériser cette topologie au travers de ce routage ?

Ex. 11 — Comparaison avec une répartition aléatoire avec Fat Tree

1. Faites la même chose en modifiant le fichier `nodes` dans le but de faire une répartition aléatoire et reprenez la question **??**.

4.2 Sur une plus grande topologie

Nous avons travailler sur une topologie avec uniquement 12 nœuds, qui ne reflète pas la réalité. C'est pourquoi nous allons augmenter le nombre de switchs et de nœuds au sein de notre topologie pour analyser les modifications de placement sur les performances globales.

Générez une topologie XGFT avec 128 nœuds, des switchs de 16 ports et de hauteur 3. XGFT(3 ;8,4 ;4,8)

Ex. 12 — XGFT avec 128 nœuds

1. Donnez votre ligne de commande.
2. Combien faut-il de switchs ?

Avec l'aide d'un script (shell ou python), renseignez le fichier `nodes` pour le cas le plus favorable.

Procédez à l'analyse du routage tel que vu dans le section II du TP.

Ex. 13 — Comparaison avec Fat Tree

1. En comparant avec le **Fat Tree** fait en exemple, pouvez-vous caractériser cette topologie au travers de ce routage ?

Ex. 14 — Comparaison avec Fat Tree

1. Faites la même chose en modifiant le fichier `nodes` dans le but de faire une répartition aléatoire. Et reprenez la question de l'exercice **??**

5 Étude d'une topologie réelle

Nous venons de voir 2 topologies : le Fat Tree et le XGFT tel que définit dans la théorie. Ces 2 topologies sont difficilement applicables dans la pratique comme on l'a vu tout au long de ce TP.

Pour finir ce TP, nous allons voir une topologie un peu plus proche de la réalité. Elle reprend l'ensemble des éléments que nous avons vu au cours de ce TP.

Dans le dossier transmis par l'intervenant, vous trouverez un fichier qui se nomme `real.txt`. Ce fichier est une topologie qui a été générée à la main (c'est-à-dire sans utiliser l'outil **createIBNet.py**).

Dans le but de pouvoir faire des comparaisons avec ce qui a été faits précédemment dans le TP, nous avons volontairement réduit le nombre de nœuds à 128.

Ex. 15 — Etude de la topologie

1. Quelles sont les caractéristiques de cette nouvelle topologie ? Pour vous aidez, commencez par donner le nombre de switchs en L1, L2, L3.

En reprenant les fichiers `nodes` que vous avez générés pour le XGFT, étudiez cette topologie réelle.

Ex. 16 — Répartition aléatoire

1. Étudiez cette topologie avec une répartition `aléatoire` des groupes

Ex. 17 — Répartition optimale

1. Étudiez cette topologie avec une répartition `optimale` des groupes

6 Analyses des performances

Dans ce TP, nous avons étudier en détails 2 topologies assez proches : le **Fat tree** et le **XGFT**. Nous avons aussi une topologie réelle, basée sur le XGFT. En modifiant les ilots (ou les groupes de calcul), nous avons pu voir l'impact de ces placements au sein de la fabrique.

Maintenant, en vous basant sur ces deux études, répondez aux questions suivantes. Il est attendu une réponse construite, avec des schémas au besoin. Les questions ne sont la que pour orienter votre analyse.

Ex. 18 — Synthèse

1. Quelles sont les conséquences que l'on peut déduire de ces analyses.
2. Quels sont les enseignements possibles vis-à-vis du placement des jobs les uns par rapport aux autres ? Et par rapport à un élément **partagé** ?
3. Peut-on trouver une solution efficace pour résoudre ce problème de partage de ressources entre plusieurs jobs ?

Références

[CEA, 2017] CEA (2017). Python framework for efficient cluster administration. http://cea-hpc.github.io/clustershell/. Accessed : 2017-12-20.

[Hoefler, 2013] Hoefler, T. (2013). createIBNet : script de génération de topologies. http://htor.inf.ethz.ch. Accessed : 2017-12-20.

[Öhring et al., 1995] Öhring, S. R., Ibel, M., Das, S. K., and Kumar, M. J. (1995). On generalized fat trees. In *Proceedings of the 9th International Symposium on Parallel Processing*, IPPS '95, pages 37–, Washington, DC, USA. IEEE Computer Society.

[Open Fabrics, 2017] Open Fabrics (2017). Ensemble d'outils pour gérer et administrer les réseaux ib. https://www.openfabrics.org. Accessed : 2017-12-20.