Minimisation du temps de complétion pour les coflux

Youcef Magnouche[1], Sebastien Martin[1], Jeremie Leguay[1],
Francesco De-Pellegrini[2], Rachid El-Azouzi[2] et Cedric Richier[2]

[1] Huawei Technologies, 18 Quai du Point du Jour, Boulogne-Billancourt, 92100, France
[2] University of Avignon, 74 Rue Louis Pasteur, Avignon, 84029, France
[1]firstname.name@huawei.com,[2]firstname.name@univ-avignon.fr

Mots-clés : *Branch-and-Benders-Cut, Complexité, Progr. en nombres entiers, Réseau.*

1 Introduction

De nos jours, la plupart des fournisseurs de cloud proposent des services de traitement de données. Les clients peuvent lancer leurs tâches de calcul sur des logiciels Big-Data tels que MapReduce ou Spark. Ces logiciels reposent sur le modèle de calcul *dataflow* pour le traitement de données à grande échelle. Celui-ci est basé sur un paradigme de calcul distribué où chaque étape de calcul intermédiaire est répartie sur un ensemble de nœuds et sa sortie est transférée à d'autres nœuds pour l'étape suivante. Entre deux étapes de calcul, ces flux de données produisent un ensemble de flux, appelé *coflux* [2], qui sont liés entre eux par la même tâche applicative. Les coflux représentent une abstraction de modèle de trafic standard dans les datacenters. Dans MapReduce par exemple, un coflux est un ensemble de flux simultanés envoyés par des nœuds dits *mappers*, vers un ensemble de nœuds dits *reducers*. Ces flux sont lancés dès que les nœuds mappers ont terminé leurs tâches de calcul. Dans cet article nous considérons le problème de d'ordonnancement de coflux pour minimiser le temps de complétion, qui est, en général, fortement NP-difficile.

Une idée intéressante apparue dans de nombreux travaux de recherche suggère d'égaliser les temps de transfert de flux par coflux [3] pour laisser tous les flux d'un coflux finir en même temps. Varys [4] est une référence de base pour les heuristiques. Même si les algorithmes récents comme Sincronia [1] ont de meilleures performances, Varys a introduit plusieurs concepts clés à la fois. Tout d'abord, il combine la priorité des flux les plus courts avec l'égalisation des temps de complétion au sein du coflux. De plus, il fonctionne sur la notion de *bottleneck* d'un coflux, c'est-à-dire le lien qui connaît le temps maximal de transfert de données. Même si plusieurs heuristiques et algorithmes d'approximation ont été proposés dans la littérature, très peu d'études se sont intéressées aux solutions exactes du problème d'ordonnancement de coflux.

Dans cet article, nous proposons une formulation linéaire compacte en nombres entiers pour le problème et développons un algorithme Branch-and-Benders-Cut pour le résoudre efficacement. Ces modèles exacts utilisent une discrétisation du temps et considèrent que l'allocation pour un flux à chaque pas de temps est constante. Sur un ensemble diversifié d'instances, nous montrons que notre algorithme améliore considérablement le temps d'exécution par rapport à la solution du modèle compact. Nous montrons également l'amélioration du temps de complétion par rapport à Varys et Sincronia.

2 Modèle mathématique

Considérons un graphe orienté $G = (V, \mathcal{L})$ où V est un ensemble de nœuds et $\mathcal{L}$ est un ensemble d'arcs ou chaque arc l a une capacité $b_l \in \mathbb{R}^+$, pour tous les $l \in \mathcal{L}$. Nous considérons $K \in \mathbb{N}$ coflux qui s'exécutent en parallèle $\mathcal{C} = \{C_1, C_2, ..., C_K\}$. Chaque coflux C_k est composé de $n^k \in \mathbb{N}$ flux avec $F_k = \{f^{k1}, f^{k2}, .., f^{kn^k}\}$. Chaque flux constituant f^{kj} est défini par

un 4-tuple $(s^{kj}, d^{kj}, \mathcal{P}^{kj}, v^{kj})$ où $s^{kj}, d^{kj} \in V$ sont, respectivement, des nœuds source et de destination. L'ensemble $\mathcal{P}^{kj}$ représente les chemins entre s^{kj} et d^{kj}, et $v^{kj} \in \mathbb{R}^+$ le volume du flux, c'est-à-dire : la quantité totale de données à transférer par le flux f^{kj}. Un coflux n'est considéré comme terminé que lorsque tous ses flux sont terminés et la dernière fois où le coflux est actif est appelé *le temps de complétion du coflux pour le coflux* C_k. Nous utilisons w_k pour indiquer le poids, c'est-à-dire l'importance de chaque coflux $C_k \in \mathcal{C}$. Soit T un horizon temporel que nous partitionnons en $T_s \in \mathbb{N}$ créneaux temporels de durée Δ unités de temps, notées par u. Soit $\mathcal{T} = \{1, \ldots, T_s\}$. Nous supposons que b_l représente la capacité de liaison associée à $l \in \mathcal{L}$, en Mb par créneau temporel. Pour ce modèle, trois types de variables sont nécessaires : $x_p^{kj}(t) \in [0,1]$, qui représente la fraction du volume total du flux f^{kj}, associé au coflux $C_k \in \mathcal{C}$ envoyé pendant le créneau te $t \in \mathcal{T}$ sur le chemin $p \in \mathcal{P}^{kj}$; $y^k(t) \in \{0,1\}$, qui est égal à 1 si créneau temporel $t \in \mathcal{T}$ est le dernier créneau temporel utilisé par coflux $C_k \in \mathcal{C}$, 0 sinon ; $\gamma^k(t) \in [0,1]$, qui représente le pourcentage inutilisé du créneau temporel final $t \in \mathcal{T}$ (le dernier où le coflux $C_k \in \mathcal{C}$ est actif).
Par conséquent le problème de la somme pondérée des fins d'exécution des coflux est équivalent au programme linéaire en nombres entiers suivant :

$$\min \sum_{C_k \in \mathcal{C}} w_k \sum_{t \in \mathcal{T}} \Delta(t y^k(t) - \gamma^k(t)) \tag{1}$$

$$\sum_{t \in \mathcal{T}} y^k(t) = 1 \qquad \forall C_k \in \mathcal{C}, \tag{2}$$

$$\sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}^{kj}} x_p^{kj}(t) = 1 \qquad \forall C_k \in \mathcal{C}, \forall f^{kj} \in F_k, \tag{3}$$

$$\gamma^k(t) \leq y^k(t) \qquad \forall C_k \in \mathcal{C}, \forall t \in \mathcal{T} \tag{4}$$

$$\sum_{C_k \in \mathcal{C}} \sum_{f^{kj} \in F_k} v^{kj} \sum_{p \in \mathcal{P}^{kj} : l \in p} x^{kj}(t) \leq b_l \qquad \forall l \in \mathcal{L}, \forall t \in \mathcal{T}, \tag{5}$$

$$\sum_{t'=t}^{T_s} \Big(\sum_{p \in \mathcal{P}^{kj}} x_p^{kj}(t') - y^k(t') \Big) \leq 0 \qquad \forall C_k \in \mathcal{C}, \forall f^{kj} \in F_k, \forall t \in \mathcal{T}, \tag{6}$$

$$\sum_{f^{kj} \in F_k} v^{kj} \sum_{p \in \mathcal{P}^{kj} : l \in p} x^{kj}(t) \leq (1 - \gamma^k(t)) b_l \qquad \forall C_k \in \mathcal{C}, \forall l \in \mathcal{L}, \forall t \in \mathcal{T}, \tag{7}$$

Les contraintes (2) sélectionnent exactement un créneau temporel final pour chaque coflux. Les contraintes (3) garantissent que tous les flux sont servis. Les contraintes (4) relient les variables y et γ. Les contraintes (5) représentent les contraintes de capacité du port. Les contraintes (6) garantissent que, pour chaque coflux, tous les flux sont envoyés avant la fin du créneau final. Enfin, les contraintes (7) diminuent la capacité du port pendant le dernier créneau temporel. Cela permet de calculer la partie inutilisée du créneau final de chaque coflux.

L'algorithme Branch-and-Benders-Cut permet de découpler les deux types de variables, en gardant les variables entières dans le problème maître, et les variables continues dans le sous-problème. Le problème maître sélectionne le créneau temporel final pour chaque coflux et le sous-problème vérifie si tous les flux peuvent être envoyés sur le réseau.

Références

[1] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. Sincronia : Near-optimal network design for coflows. In *Proc. ACM SIGCOMM*, 2018.

[2] Mosharaf Chowdhury and Ion Stoica. Coflow : A networking abstraction for cluster applications. In *Proc. ACM HotNets workshop*, 2012.

[3] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM Computer Communication Review*, 41(4) :98–109, 2011.

[4] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *Proc. ACM SIGCOMM*, 2014.