Application de K-Means à la définition du nombre de VM optimal dans un Cloud

Khaled Tannir; Hubert. Kadima ; Maria Malek

Laboratoire LARIS/EISTI Ave du Parc 95490 Cergy-Pontoise – France
contact@khaledtannir.net, hubert.kadima@eisti.fr, maria.malek@eisti.fr

Résumé

Ce papier présente les premiers éléments de définition d'un algorithme permettant de déterminer le nombre optimal de machines virtuelles (*VM – Virtual Machines*) lors de l'exécution des applications de fouille de données dans un environnement Cloud. L'efficacité de traitement des problèmes de fouille de données requiert d'obtenir au préalable un *partitionnement intelligent de données* par *clustering* de manière à effectuer le plus indépendamment que possible les traitements des fragments de données à cohérence sémantique forte.

Nous pensons que l'exécution sur les données distribuées dans le Cloud d'une variante parallèle de l'algorithme de *clustering h-means* adaptée en phase de présélection du processus PMML [18] pour (*Predictive Model Markup Language*) permettrait d'assurer un partitionnement optimal des données et de déterminer un nombre de VM optimal avant l'exécution de l'application.

Mots-clés : *Cloud computing, h-means, classification, parallélisme dans des grilles, partitionnement de données, fouille de données.*

1. Introduction

Ce papier présente les premiers éléments de définition d'un algorithme permettant de déterminer le nombre optimal de machines virtuelles pour une exécution efficace des applications de fouille de données dans un environnement Cloud. Un environnement *Data Intensive Computing*, tel que celui-ci, devrait permettre d'exécuter efficacement des applications qui permettent la production, manipulation ou analyse des gros volumes de données – de quelques centaines de *megabytes* aux *petabytes* et au-delà [1]. A cause des coûts de communication élevés dans cet environnement, les traitements parallèles doivent être les plus autonomes que possibles et effectués avec le moins de synchronisations possibles. L'efficacité de traitement des problèmes de fouille de données requiert d'obtenir au préalable un *partitionnement intelligent de données* de manière à effectuer le plus indépendamment possible les traitements des fragments de données en tenant compte des caractéristiques spécifiques aux Clouds (*élasticité, monitoring* fin d'utilisation des ressources, *speed-up* …).

L'utilité du *partitionnement intelligent* obtenu à partir du *clustering* pour le problème des règles d'association a été déjà étudiée dans [2] par exemple. Nous pensons que l'exécution sur les données distribuées dans le Cloud d'une variante parallèle de l'algorithme de

clustering h-means en phase de présélection du processus PMML [18] (*Predictive Model Markup Language*) permettrait d'assurer un partitionnement optimal des données et de déterminer un nombre de VM optimal. Nous présentons dans ce papier notre approche qui consiste de partitionner les données en fonction de leurs similarités. Nous procédons par deux étapes : l'exécution de l'algorithme de partitionnement puis distribuer les données partitionnées sur le nombre de machines virtuelles trouvé.

Dans notre environnement d'exécution, les applications de fouille de données sont représentées sous forme d'un *workflow* conformément au processus PMML. Toute activité du *workflow* impliquant la manipulation de données génère des requêtes sur les données distribuées dans le Cloud. On trouve dans [3] une présentation de plusieurs techniques spécifiques utilisées par les systèmes de *workflows* pour exécuter les applications *workflows data intensive* en utilisant des ressources globalement distribuées. La plupart des systèmes utilisent une combinaison des techniques pour fournir les performances élevées et la haute disponibilité à faibles coûts. Une seule technique n'est pas suffisante pour minimiser les effets sur la performance et les coûts et augmenter l'efficacité des opérations de transferts de données.

Le partitionnement de données, le placement de données ou la réplication peuvent être réalisés avant ou durant l'exécution de *workflows* dans un environnement *Cloud Data Intensive* pour améliorer les performances d'exécution de l'application. Kosar et al. [4] ont défini un *scheduler* pour les activités de placement de données dans le Grid. Ils proposent d'en faire un axe majeur des travaux de recherche à effectuer dans ce domaine. Les activités de placement de données peuvent être intégrées ou découplées des activités de *scheduling des tasks.* La réplication de données dans des ressources distribuées est le mécanisme commun pour augmenter la disponibilité de données. La décision de placement et de réplication de données sont basées sur les objectifs à optimiser, la localité de référence de données et la minimisation de la distance de répartition entre les données et les traitements associés.

La réponse apportée par h-*means* pour répondre à ces derniers problèmes se traduit en termes de traitement des requêtes par similarité sur de gros volumes de données, notamment à travers, par exemple, les calculs des *k* plus proches voisins (*k*NN). Au processus de *clustering*, s'ajoutent les problèmes de parallélisme de traitements et de la distribution de données. On doit se donner les moyens d'étudier les performances de cette approche de partitionnement appliquée sur une grosse base de données pour obtenir et exploiter des classes et déterminer une taille et un nombre optimal de ces classes pour que les requêtes issues du *workflow* puissent être traitées en temps optimal et avec une haute précision. La mise en œuvre de h-*means parallèle* introduit des méthodes de traitement de requêtes parallèles sur une grille de machines pour réaliser l'allocation optimale des données grâce à la recherche efficace des *k*NN en parallèle. On vise l'obtention d'un nombre réduit de VM distribués à travers les Clouds permettant néanmoins des temps de recherche sous-linéaires et optimaux vis-à-vis des classes déterminées précédemment.

Dans un premier temps, une application structurée en *workflow* et utilisant les techniques de fouille de données est exécutée pour des besoins de tests dans un environnement Cloud hybride comportant l'infrastructure Cloud *OpenNebula* [5] interconnectée à *Amazon EC2* [6].

Les tests s'effectuent sur les instances *EC2 Amazon* en connexion avec un moteur d'orchestration de *workflow* implanté côté *OpenNebula*.

La suite de ce papier est structurée comme suit. Dans la section 2, nous présentons notre algorithme CloudMeans variante de h-*means parallèle* que nous utilisons. Dans la section 3, nous décrivons la configuration de l'environnement d'exécution de l'application de tests basée sur l'utilisation du moteur de *workflow* KNIME [17] et de l'infrastructure Cloud OpenNebula interconectée à Amazon via les services AWS (*Amazon Web Services*). A titre de conclusion, dans la section 4, nous présentons le plan des premières actions entreprises et les principaux axes de nos futurs travaux.

2. Présentation de points essentiels de l'algorithme

CloudMeans est une variante de *h-means* [9] lui même variante de l'algorithme *k-means* [8]. K-means est un des algorithmes de segmentation les plus connus. Il est souvent utilisé quand il y a un besoin d'extraire des regroupements (appelés encore catégories, groupes ou "clusters") à partir d'un ensemble de données d'une façon non supervisée. Chaque catégorie est représentée par son barycentre.

Entrées L'algorithme prend comme entrées un ensemble de n exemples, un entier k qui est le nombre de catégories à trouver, ainsi qu'une mesure de distance d() (comme la distance euclidienne) qui permet de calculer la distante entre deux exemples. Plus cette distance entre deux exemples est petite, plus les deux exemples sont similaires.

Déroulement L'algorithme assigne à chaque itération un exemple donné au barycentre le plus proche. Des mesures d'inerties inter-catégories et intra catégories sont calculées. Ces mesures d'inerties sont basées sur les densités des groupes. L'algorithme converge vers une situation de stabilisation de barycentres quand l'inertie inter-catégories est minimisée, autrement dit quand les catégories trouvées sont les plus denses possibles (ou les mieux séparés).

Sortie La sortie de l'algorithme consiste en une partition des n données en k groupes (ou catégories) chacun est représenté par son barycentre. Une variante très connue de k-means est l'algorithme h-means [9] dont le but est d'optimiser le temps de convergence vers les barycentres. La différence se fait au moment du calcul du barycentre qui se fait chaque fois qu'une catégorie est traitée complètement et non pas après le traitement de chaque exemple.

Plusieurs algorithmes de parallélisation de k-means ont été proposés [10,11,12,13]. Dans [10], une parallélisation de l'algorithme h-means est présentée. Nous présentons ici une parallélisation possible (sur plusieurs processeurs) de l'algorithme h-means. Cette parallélisation constitue le point de départ pour notre algorithme.

Version parallèle de l'algorithme h-means

Répartir en bloc de manière égale les exemples sur *p* processeurs

Pour chaque processeur k_p dans {0,1,..,*p*-1} (**en parallèle**)
Attribuer arbitrairement chaque exemple dans un processeur k_p à une catégorie

/* Calcul de affectations des exemples aux catégories en parallèle */

Tant que les barycentres ne sont pas stables
Pour tous les exemples Exemples(i) dans processeur k_p
Calculer la distance entre Exemple(i) et tous les barycentres
Trouver k' tel que k'ème barycentre soit le plus proche de Exemple(i)
Attribuer Exemple(i) au barycentre k'

/* Recalcule des barycentres des catégories en parallèle */

Calculer la somme partielle des exemples appartenant à la même catégorie.
Envoyer la somme partielle aux autres processeurs.

Recevoir les sommes partielles.

Pour chaque catégorie **faire**

Effectuer la somme des sommes partielles.
Diviser la somme totale par le nombre d'exemples de la catégorie courante.

Notre adaptation des ces algorithmes étend l'algorithme de parallélisation *h-means* en l'englobant dans d'autres taches plus complexes. Le déroulement de l'algorithme, illustré par la figure 1, peut être résumé ainsi : on commence par indiquer un nombre choisi aléatoirement (p) que nous introduisons comme paramètre en entrée à l'algorithme. Ce nombre servira comme base de répartition des blocs de données. Nous étudions actuellement l'impact des variations de ce nombre sur l'efficacité de notre algorithme et travaillons actuellement pour déterminer ce nombre de façon optimale. Une fois le nombre de catégories nous a été retourné par cette partie de l'algorithme nous l'utiliserons comme paramètre pour la création d'un nombre égale de machines virtuelles sur le cloud local. Les données actuellement partitionnées par catégorie seront redistribuées sur ces machines. Dans le cas où les ressources du cloud local sont insuffisantes pour absorber cette demande, nous faisons appel au cloud public afin qu'il nous fournisse les ressources manquantes.

Actuellement nous travaillons sur cette partie afin de définir de façon optimale la stratégie à adopter afin de maximiser l'efficacité des traitements tout en tenant compte du volume de données à traiter, du nombre de machines virtuelles à créer. Finalement, le traitement des données est lancé et le résultat est récupéré et mis à disposition de la prochaine étape du workflow global du traitement.

Les données internes nécessaires au fonctionnement de l'algorithme comme par exemple les tables de hachages communes sont stockées en local. Nous étudions également la possibilité d'utiliser une base NoSQL telle que Cassandra [21] pour améliorer la disponibilité et les performances de ces données. Une base de données NoSQL (*Not only SQL*) désigne une catégorie de base de données apparue en 2009 qui se différencie du modèle relationnel que l'on trouve dans des bases de données connues comme MySQL ou PostgreSQL. Ceci permet d'offrir une solution alternative aux

problèmes récurrents des bases de données relationnelles de perte de performance lorsque l'on doit traiter un très gros volume de données.

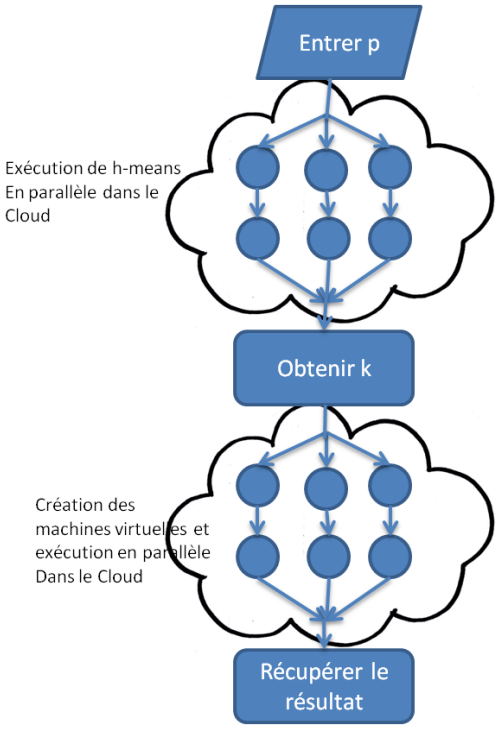

Figure 1 -Les points essentiels de l'algorithme

3. Un environnement distribué piloté par les workflows applicatifs

Dans notre environnement d'exécution, les applications de fouille de données sont représentées sous forme d'un *workflow* conformément au processus PMML [18] (*Predictive Model Markup Language*). Toute activité du *workflow* impliquant la manipulation de données génère des requêtes sur les données distribuées dans le Cloud.

PMML est un langage basé sur XML définissant une manière standard de décrire les modèles statistiques et de traitement de données au sein des applications décisionnelles. Ce langage est poussé par un consortium d'éditeurs, le *Data Mining Group* (DMG). Il a pour but de permettre le partage de modèles analytiques entre outils de « Business Intelligence » supportant ce standard. L'idée est bien de dépasser l'obstacle des technologies propriétaires en fournissant la possibilité de définir un modèle de traitement de données dans une application, qui puisse être consulté et réexploité dans un autre outil. Le format de description proposé prend la forme d'un XML Schema.

PMML intègre l'ensemble des informations nécessaires à l'exploitation d'un modèle d'analyse : le type de données en entrée et en sortie, le mode d'exécution du modèle, et la manière de générer et d'interpréter les résultats en s'appuyant sur ce standard. Egalement, il a pour vocation de couvrir l'ensemble des grandes méthodes de datamining.

Le moteur de workflow utilisé dans notre environnement est KNIME [16]. Il s'agit d'un environnement logiciel d'exécution de workflow (appelé aussi pipeline) libre de droit. Cet environnement est basé sur un principe modulaire, c'est à dire que les différentes étapes du pipeline sont représentées par des nœuds. Chaque nœud a une fonction précise et possède aucune ou plusieurs entrées et/ou aucune ou plusieurs sorties. De plus chaque nœud peut prendre en entrée la ou les sorties d'un ou plusieurs nœuds. KNIME est fourni avec un certain nombre de nœuds spécialisés pour la visualisation de données, l'analyse statistique et la fouille de données et offre la possibilité d'en créer des nouveaux pour interfacer par exemple des applications existantes. KNIME inclut des fonctionnalités de lecture / écriture au format PMML. Il est donc capable d'importer un fichier respectant le standard PMML et de le traduire en son propre format de pipeline comme d'exporter un pipeline KNIME au format PMML.

Notre environnement de test est basé sur un Cloud hybride. Ce type de Cloud est connu aussi sous l'appellation « Cloudbursting ». Il s'agit d'une extension des ressources du Cloud privé local avec celles d'un Cloud public. Ceci dans le but de combiner les ressources locales avec celles fournies par le Cloud public. Dans notre cas le fournisseur public est Amazon. L'intérêt d'une architecture de cloudbursting c'est qu'elle permet une haute évolutivité des environnements et améliore l'aspect « élasticité » des ressources.

Le choix du Cloud OpenNebula [6, 16] est motivé par le fait qu'il est l'un des plus riches en fonctionnalités open source des gestionnaires d'infrastructures virtuelles (*Virtual Infrastructure*). Il est basé sur le système d'exploitation Linux et a été initialement conçu pour gérer des infrastructures virtuelles locales tout en incluant des interfaces d'accès distants permettant la création de clouds publics.

OpenNebula met à disposition quatre API (*Application Programmable Interface*) au total :

- XML-RPC [19] pour les interactions locales.
- Libvirt [15] pour les interactions locales.
- un sous-ensemble de l'API Amazon EC2 (Query).
- l'API du cloud OpenNebula lui-même pour un accès public [14, 16].

L'architecture de OpenNebula est une architecture modulaire. Elle englobe plusieurs composants enfichables spécialisés. Le module principal orchestre les serveurs physiques avec leurs contrôleurs haut niveau (*hypervisors*), les nœuds de stockage et le réseau. Les opérations d'administration sont effectuées à travers des pilotes enfichables (*pluggable drivers*), qui interagissent avec les API des hyperviseurs, du stockage, du réseau et des clouds publics.

OpenNebula contient un module ordonnanceur (*Scheduler*) qui offre des fonctionnalités d'allocations de ressources dynamiques. Il est chargé d'assigner les requêtes de machines virtuelles (VM) en attente aux hôtes physiques. Le Scheduler donne la possibilité aux administrateurs de choisir différents types de gestion de ressources tels que le déploiement des VM sur un nombre réduit de hôtes physiques ou de maintenir une configuration d'équilibrage de charge. Egalement, il prend en charge plusieurs approches de virtualisations telles que Xen Hypervisor, KVM, et VMware vSphere. Ceci tout en permettant de déplacer des instances en cours d'exécution entre les différents nœuds connectés. Cependant, OpenNebula supporte uniquement les fonctionnalités de base de l'API de EC2 SOAP et de EC2 Query du cloud Amazon.

Notre plateforme vise à supporter l'exécution de notre algorithme CloudMeans lequel, comme décrit dans la section précédente, englobe une variante de l'algorithme k-means. La parallélisation implique de rendre asynchrone la suite des traitements en les décomposant de manière efficace comme une suite de processus connectés les uns aux autres avec des files d'attente de messages selon le pattern *pipe and filter*. Les unités de traitement successives attendent l'arrivée d'une tâche, la traitent et passent le résultat à l'étape suivante.

Une requête de parallélisation implique une phase de mappage des données à traiter, de lancer le traitement approprié sur ces données puis de procéder à la collecte du résultat du traitement effectué. Pour cela nous avons découpé le traitement en trois phases principales : la phase d'initialisation, la phase de mappage et la phase de collecte.

Pour implémenter le mécanisme d'échange des données et la collecte du résultat final entre les différentes images du cloud public Amazon EC2, nous avons utilisé Amazon SQS [20] (*Simple Queue Service*) qui permet la création de *pipelines* (files de traitements) hautement adaptables avec des unités de traitement interconnectées de façon souple. Ces files d'attentes permettent la flexibilité, l'asynchronisme et la tolérance aux pannes. Chaque étape du pipeline récupère des unités de travail d'une instance du service de file d'attente, traite l'unité de travail de façon appropriée et écrit les résultats dans une autre file d'attente pour les traitements ultérieurs. Il faut noter que les files d'attente fonctionnent bien lorsque les ressources nécessaires (temps, capacité processeur, vitesse d'entrées-sorties) de chaque étape pour une unité de travail particulière varient beaucoup.

Une file SQS contient des indicateurs sur l'emplacement des données à traiter. Dans la version actuelle de notre environnement de test nous avons définit manuellement le nombre des AMI (Amazon Machine Image) qui seront instanciées dans l'environnement EC2. Ce nombre sera défini par l'algorithme CloudMeans une fois finalisé en tenant compte des paramètres et contraintes citées dans la section précédente.

4. Conclusion et suite de travaux

Dans ce papier nous avons présenté les premiers éléments de notre algorithme ayant pour objectif de définir le nombre optimal de machines virtuelles à créer dans un environnement de cloud en mettant à profit une variante parallélisée de l'algorithme *k-means.* Nous avons également présenté notre environnement cloud de test lequel est un cloud hybride construit avec OpenNebula et Amazon EC2. Cet environnement constitue la base de nos futurs travaux sur la parallélisation et l'amélioration des performances des algorithmes de clustering dans le Cloud pour les applications de fouille de données.

Nous avons également présenté les principales technologies et les composants constituant notre environnement de tests. Notre environnement exploite efficacement l'ensemble des avantages attendus d'un environnement cloud à savoir l'élasticité, la scalabilité, la flexibilité et la simplicité d'utilisation.

Les différents tests ont démontré que l'algorithme *h-means* constituait un bon point de départ de notre algorithme qui vise à optimiser le nombre de machines virtuelles dans le but d'utiliser les ressources plus efficacement. La prochaine étape consiste à mettre à profit CloudMean dans le but de segmenter les données dans la phase du pré-traitement du "data mining".

Nous appliquons ce modèle sur l'algorithme *Apriori* qui a comme but la découverte des règles d'association à partir d'un ensemble de données appelé transactions [7]. Les règles d'association, étant une méthode d'apprentissage non supervisé, elles permettront de découvrir à partir d'un ensemble de transactions, un ensemble de règles qui exprime une possibilité d'association entre différents items. Une transaction est une succession d'items exprimée selon un ordre donné ; de même, l'ensemble de transactions contient des transactions de longueurs différentes.

Nous travaillons actuellement pour proposer un nouvel algorithme dont le but est de chercher plus efficacement les sous-ensembles fréquents ; l'idée étant d'appliquer auparavant notre version actuelle de l'algorithme dans le but d'améliorer le "point de départ" de la recherche des sous ensembles fréquents (au lieu de partir des candidats de longueurs 1). Nous sommes en train de réaliser des tests sur ce nouvel algorithme ainsi que l'influence des différents paramètres d'entrée sur les résultats ; notre but, dans ce cas, étant de le compléter pour assurer la couverture des sous-ensembles fréquents.

5. Références bibliographiques

[1] R. Moore, T.A Prince and M. Ellisman "Data-intensive computing and digital libraries" Commun. ACM, vol. 41, nà 11, pp 56-62, 1998.

[2] V. Fiolet, B. Toursel, Distributed data mining, Scalable Computing: Practice and Experiences 6 (1) (2005) 99–109.

[3] Suraj Pandey «Scheduling and Management of Data Intensive Application Workflows in Grid and Cloud Computing Environments» – PhD University of Melbourne – December 2010.

[4] T.Kosar and M. Livny "Stork : Making Data Placement a First Class Citizen in the Grid" in ICDCS'04 Proceedings of the 24th International Conference on Distributed Computing Systems. Washington, DC, USA IEEE 2004 pp. 342-349

[5] OpenNebula. http://opennebula.org

[6] Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2.

[7] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In ACM SIGMOD Conference on Management of Data, 1993.

[8] J. Hartigan. Clustering Algorithm. John Wiles and Sons, NewYork,1975.

[9] R. Howard. Classifying a population into homogeneous groups. J.R. Lawerence (Ed.), Operational Research in the Social Sciences, 1966.

[10] K.Stoffel and A.Belkoniene. Parallel k/h-means clustering for large data sets. In Lecture Notes in Computer Science 1685, Proceedings of the 5th International Euro-Par Conference on Parallel Processing. Springer, 2004.

[11] S. Rahimi, M. Zargham, A. Thakre, and D. Chhillar. A parallel fuzzy c-mean algorithm for image segmentation. In IEEE Annual Meeting of the Fuzzy Information, Processing NAFIPS 4, volume 1, pages 234–237,2004.

[12] T.Jinlan, Z.Lin, Z.Suqin, and L.Lu. Improvement and parallelism of k-meansclustering algorithm. Tsinghua Science and Technology, 3(10) :277–281, 2005.

[13] Y.Zhang, Z.Xiong, J.Mao, and L.Ou. The study of paralle lk-means algorithm. In 6th World Congress on Intelligent Control and Automation 2, pages 5868–5871, 2006.

[14] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Computing, 13(5):1422, September/October, 2009.

[15] Libvirt: The Virtualization API, Terminology and Goals, http://libvirt.org/goals. html, 22/4/2010.

[16] Distributed Systems Architecture Group, OpenNebula: The open source toolkit for cloud computing, http://www.opennebula.org, 22/4/2010.

[17] KNIME http://www.knime.com/

[18] PMML http://www.dmg.org/v4-0-1/GeneralStructure.html

[19] XML-RPC Specification http://xmlrpc.scripting.com/spec.html#update3

[20] Amazon Simple Queue Service (Amazon SQS) http://aws.amazon.com/fr/sqs/

[21] Avinash Lakshman and Prashant Malik. Cassandra - A Decentralized Structured Storage System.