Modèles de coût pour la sélection de vues matérialisées dans le nuage, application aux services Amazon EC2 et S3

Romain Perriot, Jérémy Pfeiffer, Laurent d'Orazio, Bruno Bachelet, Sandro Bimonte, Jérôme Darmont

Modèles de Coût pour la Sélection de Vues Matérialisées dans le Nuage, Application aux Services Amazon EC2 et S3

Romain Perriot*, Jérémy Pfeifer*, Laurent d'Orazio*,
Bruno Bachelet*, Sandro Bimonte**, Jérôme Darmont***

*Clermont Université, CNRS, Université Blaise Pascal, LIMOS UMR 6158
nom.prenom@univ-bpclermont.fr
**IRSTEA, UR TSCF, Clermont-Ferrand
sandro.bimonte@irstea.fr
***ERIC Lyon 2, Université de Lyon
jerome.darmont@univ-lyon2.fr

Résumé. La performance des entrepôts de données est classiquement assurée grâce à des structures comme les index ou les vues matérialisées. Dans ce contexte, des modèles de coût permettent de sélectionner un ensemble pertinent de ce type de structures. Toutefois, cette sélection devient plus complexe dans les nuages informatiques, car en plus des temps de réponse, il faut simultanément optimiser le coût monétaire. Nous proposons dans cet article de nouveaux modèles de coût intégrant le paiement à la demande en vigueur dans les nuages. Sur la base de ces modèles, nous définissons un problème d'optimisation consistant à sélectionner, parmi des vues candidates, celles à matérialiser pour minimiser le coût d'interrogation et de maintenance, ainsi que le temps de réponse pour une charge de requêtes donnée. Dans un premier temps, nous optimisons les deux critères séparément : le temps est optimisé sous contrainte de coût et vice versa. Notre proposition est ensuite validée de manière expérimentale.

1 Introduction

L'informatique dans le nuage (*cloud computing*), sous l'impulsion d'entreprises comme Google, Microsoft et Amazon, suscite une attention particulière. Ce paradigme permet un accès à la demande à des ressources configurables pouvant être rapidement mises à disposition avec une maintenance minimale. Selon un modèle de facturation à l'utilisation, les clients ne payent que pour les ressources (stockage et calcul) qu'ils utilisent. Les performances dans les nuages reposent généralement sur l'utilisation d'un grand nombre d'instances avec un parallélisme transparent pour l'utilisateur.

Les entrepôts de données et les systèmes OLAP (On-Line Analytical Processing) représentent des technologies d'aide à la décision qui permettent l'analyse en ligne de gros volumes de données. Ces technologies reposent sur des techniques d'optimisation comme des index, des caches ou encore des modèles logiques dénormalisés qui permettent l'analyse multidimensionnelle (agrégations sur plusieurs axes d'analyse) des données tout en garantissant des bonnes

performances. Si classiquement, les entrepôts de données sont gérés au sein de l'organisation qui les exploitent, avec l'arrivée des nuages, ces organisations ont tendance à déployer leurs outils d'analyse au sein des nuages pour profiter de leur puissance de calcul et de stockage, et éliminer les coûts de maintenance.

Dans cet article, nous nous intéressons aux problématiques liées aux vues matérialisées et à leur impact sur le modèle de facturation à l'utilisation des nuages. Les vues matérialisées permettent de stocker physiquement des résultats de requêtes pertinentes et fréquentes afin de réduire les temps de réponse. Un des principaux défis consiste à sélectionner les vues à matérialiser. Traditionnellement, les critères utilisés pour la sélection comprennent principalement le stockage et les coûts de maintenance Aouiche et Darmont (2009); Baril et Bellahsene (2003). Dans le nuage, le stockage est virtuellement infini, permettant de stocker toutes les vues. Cependant, chaque vue matérialisée implique un coût de stockage supplémentaire. Le problème d'optimisation des performances revient alors à trouver un compromis entre temps de réponse et coûts, et dépend des besoins d'un utilisateur particulier. À une extrémité du spectre, les utilisateurs sous une contrainte budgétaire forte peuvent accepter des temps de réponse importants, alors qu'à l'autre extrémité, les utilisateurs peuvent ne pas tenir compte des coûts s'ils ont besoin d'une réponse très rapide.

Nous abordons le problème d'optimisation multicritère de sélection d'un ensemble de vues à matérialiser afin d'optimiser à la fois le coût budgétaire du stockage et de l'interrogation d'un entrepôt de données dans le nuage, ainsi que le temps de réponse global. Pour atteindre cet objectif, notre principale contribution est la conception de modèles de coût pour le stockage, la maintenance et l'interrogation des vues matérialisées dans le nuage. Cet article étend notre précédente proposition Nguyen et al. (2012) selon trois directions. D'abord, il propose des modèles de coûts plus flexibles qui peuvent être appliqués à différents fournisseurs, même si dans cet article nous nous focalisons sur certains services Amazon. Ensuite, il introduit une nouvelle formulation qui permet de résoudre ce problème d'optimisation en utilisant des solveurs du type CPLEX [1]. Enfin, notre solution est validée avec le banc d'essais Star Schema Benchmark O'Neil et al. (2009).

Cet article est organisé de la manière suivante. La Section 2 présente le contexte de notre travail. Les Sections 3 et 4 définissent les modèles de coût pour la gestion de données dans les nuages et les vues matérialisées, respectivement. La Section 5 décrit le problème d'optimisation basé sur ces modèles de coût. La Section 6 présente les expériences que nous avons menées pour valider notre solution. La Section 7 positionne notre travail par rapport à l'état de l'art. Enfin, la Section 8 conclut cet article et énonce des pistes de recherche future.

2 Motivations

Pour définir les motivations du travail, cette section décrit les tarifications de services Amazon et définit la problématique de la sélection des vues matérialisées dans le nuage.

2.1 Tarifications dans les nuages

Les fournisseurs de services dans le nuage (ou CSP pour *Cloud Service Providers*) mettent à disposition un ensemble de ressources telles que du matériel (processeurs, stockage, réseaux),

1. IBM ILOG CPLEX Optimizer : http ://www.ibm.com/ software/integration/optimization/cplex-optimizer/

des plateformes de développement et des services. Il existe de nombreux CSP sur le marché, tels qu'Amazon, Google ou Microsoft. Chaque CSP propose différents services et des tarifications variées. Ce travail se limite à un modèle simplifié, mais suffisamment représentatif, comprenant les éléments communément facturés, à savoir le processeur, le stockage et la consommation de bande passante [2]. Ce modèle est en particulier conforme aux services EC2 et S3 offerts par Amazon, qui seront utilisés comme paramètres du modèle. Les tarifs proposés par EC2 et S3 sont en particulier utilisés dans les différents exemples. Il est également important de noter que les valeurs mentionnées dans les exemples sont celles obtenues avec le système d'analyse de données Pig Olston et al. (2008). Cependant, la solution que nous proposons est indépendante du système et peut donc être utilisée avec d'autres outils.

Amazon Elastic Compute Cloud (EC2) [3] met à disposition des ressources de calcul. Différentes configurations d'instances peuvent être louées (*micro*, *extra small*, *small*, *large*, *extra large*, etc.) à différents prix, comme le montre la Table 1(a). Par exemple, le prix pour une instance *small* (correspondant à une machine avec 1,7 Go RAM, 1 unité de calcul EC2, 160 Go de stockage local) avec Linux est de 0,06 $ par heure.

La consommation de bande passante est facturée en fonction du volume de données (Table 1(b)). Chez Amazon, les transferts de données en entrée sont gratuits, alors que le coût de la récupération de données varie en fonction de la quantité de données.

Amazon S3 [4] propose un stockage global ne dépendant pas du nombre d'instances (contrairement par exemple à Amazon EBS) dont le tarif varie en fonction du volume de données (Table 1(c)) permettant ainsi des économies d'échelle (le prix à payer diminue lorsque le volume de données augmente).

2.2 Problématique

Type	**Prix /heure**
Micro	0,02 $
Small	0,06 $
Medium	0,12 $
Large	0,24 $
Extra large	0,48 $

(a) Instances EC2

Volume de données	**Prix /mois**
Données entrantes	
Toutes les données	Gratuit
Données sortantes	
Premier Go	Gratuit
Jusqu'à 10 To	0,12 $ /Go
40 To suivants	0,09 $ /Go
100 To suivants	0,07 $ /Go

(b) Bande passante

Volume	**Prix /mois**
Premier To	0,095 $ /Go
49 To suivants	0,080 $ /Go
450 To suivants	0,070 $ /Go

(c) Stockage S3

FIG. 1 – *Tarification Amazon.*

Nous supposons disposer d'un ensemble de vues candidates à la matérialisation qui ont été préselectionnées à l'aide de méthodes existantes de sélection de vues (par exemple Baril et Bellahsene (2003)). L'objectif de ce travail est de choisir les meilleures vues au regard du modèle de facturation à l'utilisation proposé par les CSP, en considérant ainsi des contraintes budgétaires avant la matérialisation. Les perspectives de recherche que nous avons considèrent

2. Des travaux futurs devraient permettre de proposer un canevas générique visant à considérer les différents CSP.
3. Amazon EC2 : http ://aws.amazon.com/ec2/
4. Amazon S3 : http ://aws.amazon.com/s3/

en particulier l'extension des algorithmes de sélection existants prenant en compte les aspects économiques, afin de fournir un processus uniforme.

3 Modèles de tarification dans un nuage

Cette section présente des modèles de coût généraux pour la gestion de données dans un nuage, c'est-à-dire sans considérer l'utilisation de vues matérialisées. Avec le nuage, des utilisateurs louent des ressources à un CSP pour exécuter des applications. Les coûts sont généralement liés à la consommation de bande passante pour le transfert des données en entrée et la récupération des résultats, ainsi que le temps de traitement des applications.

Soit C_c la somme des coûts de calcul, C_s la somme des coûts de stockage et C_t la somme des coûts de transfert de données. Ainsi le coût total C de la gestion de données dans le nuage est :

$$C = C_c + C_s + C_t. \tag{1}$$

Définissons les paramètres généraux et les fonctions que nous utilisons pour exprimer nos modèles de coût (Table 1). Soit $Q = \{Q_i\}_{i=1..n_Q}$ la charge, c'est-à-dire l'ensemble des requêtes à exécuter, et $A = \{A_i\}_{i=1..n_Q}$ les réponses à ces requêtes. La fréquence des requêtes est considérée par le fait qu'une requête peut apparaître plusieurs fois dans la charge Q. L'ensemble de données complet est noté D. La fonction $s(X)$ retourne la taille en Go de X, par exemple $s(A_i)$ est la taille de la réponse A_i. La fonction $t(X)$ retourne le temps de stockage de X, par exemple $t(D)$ est le temps de stockage de la base D dans le nuage.

Paramètre	**Description**
$Q = \{Q_i\}_{i=1..n_Q}$	Charge, requêtes à exécuter
$A = \{A_i\}_{i=1..n_Q}$	Réponses aux requêtes
$D = \{D_k\}_{k=1..n_D}$	Ensemble de données
$IC = \{IC_j\}_{j=1..n_{IC}}$	Configuration des instances
$s(X)$	Taille en Go de X
$t(X)$	Temps de stockage de X

TAB. 1 – *Paramètres généraux*

3.1 Fonctions linéaires par morceaux

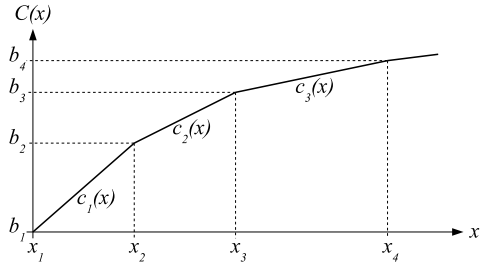


FIG. 2 – *Fonction linéaire par morceaux*

Certains coûts (les coûts de transfert et de stockage, notamment) sont des fonctions linéaires par morceaux. Dans la suite de l'article, nous définissons une fonction de coût $C(x)$ linéaire par morceaux comme une fonction qui peut être décomposée en segments (Figure 2).

Chaque segment e représente $C(x)$ dans un intervalle des valeurs d'entrées $[x_e; x_{e+1}[$ et est caractérisé par une pente a_e et un coût initial $b_e = C(x_e)$. Considérant un segment e tel que $x \in [x_e; x_{e+1}[$, $C(x) = c_e(x) = a_e \times (x - x_e) + b_e$. La fonction linéaire par morceaux $C(x)$ est donc exprimée comme suit :

$$\begin{aligned} C(x) &= c_e(s(x)) \\ &= a_e \times (x - x_e) + b_e \end{aligned} \tag{2}$$

avec e tel que $x_e \leq x < x_{e+1}$.

3.2 Coût de transfert

Le coût de transfert de données C_t dépend de la taille des données envoyées, c'est-à-dire les requêtes Q_i et l'ensemble de données D (incluant l'ensemble initial et les données additionnelles éventuellement insérées), et de la taille des données reçues, c'est-à-dire les réponses A_i aux requêtes. Le coût peut être décomposé en un coût de transfert montant C_t^- et un coût de transfert descendant C_t^+ :

$$C_t(D, Q, A) = C_t^-(D, Q) + C_t^+(A). \tag{3}$$

Notons qu'Amazon EC2 ne facture pas le transfert de données montant, donc les requêtes et l'ensemble de données peuvent être ignorés pour l'instant. En conséquence, le coût de transfert de données peut être réduit à $C_t^+(A)$ et dépend donc uniquement de A :

$$C_t(A) = C_t^+(A). \tag{4}$$

La facturation d'Amazon EC2 est variable. Elle est nulle pour le premier Go. Ensuite, elle devient 0,12 $ par Go jusqu'à 10 To, 0,09 $ par Go jusqu'à 40 To et ainsi de suite (Section 2.1). La fonction de coût C_t d'une telle politique de facturation est une fonction linéaire par morceaux (Formule 2).

Exemple 1 *Avec 10 Go de consommation de bande passante, $x_2 \leq s(A) < x_3$, donc le coût de transfert de données est calculé à l'aide du segment* 2 *de la fonction : $C_t(A) = c_2(s(A)) = 0,12 \times (10 - 1) + 0 = 1,08$ \$.*

3.3 Coût de calcul

Le coût de calcul C_c dépend de la charge Q, de la configuration IC des instances de calcul, c'est-à-dire le type (*micro*, *small*, *medium*, etc.) et le nombre de nœuds utilisés : $C_c(Q, IC)$.

Amazon associe un prix à chaque type d'instance. Chaque instance peut proposer des performances variables (par rapport à son nombre de cœurs, à sa mémoire vive, etc.) et donc des coûts différents. Le fournisseur calcule ensuite le prix à payer par instance comme le produit du temps d'utilisation par le prix pour chaque instance. Finalement, il additionne les résultats pour toutes les instances allouées.

Supposons maintenant que les requêtes sont exécutées sur une configuration d'instances IC composée de n_{IC} instances de calcul IC_j : $IC = \{IC_j\}_{j=1..n_{IC}}$. Le coût de location de

l'instance IC_j est noté $c_c(IC_j)$. Le temps de traitement d'une requête Q_i sur une instance IC_j est noté $t(Q_i, IC_j)$. Ainsi, le coût de traitement de l'ensemble des requêtes $Q = \{Q_i\}_{i=1..n_Q}$ peut être exprimé par la fonction suivante :

$$C_c(Q, IC) = \sum_{i=1}^{n_Q} \sum_{j=1}^{n_{IC}} t(Q_i, IC_j) \times c_c(IC_j). \quad (5)$$

Exemple 2 *Considérons que la charge $Q = \{Q_1\}$ est traitée en 50 heures sur deux instances de type "small" d'Amazon EC2. Ainsi, le coût de traitement est : $C_c(Q, IC) = t(Q_1, IC_1) \times c_c(IC_1) + t(Q_1, IC_2) \times c_c(IC_2) = 50 \times 0,06 + 50 \times 0,06 = 6$ \$.*

3.4 Coût de stockage

Le coût de stockage C_s dépend de la taille et du temps de stockage de l'ensemble de données complet D, et de la configuration des instances IC : $C_s(D, IC)$.

Le temps de stockage $t(D)$ de l'ensemble de données D peut être divisé en périodes telles que $t(D) = \sum_{k=1}^{n_D} t(D_k)$, où D_k représente l'ensemble de données complet pour la période k. Dans chaque période k, la taille $s(D_k)$ des données stockées est fixe. Le coût total de stockage est donc la somme des prix à payer pour chaque période :

$$C_s(D, IC) = \sum_{k=1}^{n_D} C_s(D_k, IC). \quad (6)$$

La tarification d'Amazon S3 est variable. Elle est de 0.14 \$ par Go pour le premier To. Ensuite, elle devient 0.125 \$ par Go jusqu'à 450 To et ainsi de suite (Section 2.1). Contrairement à Amazon EBS, le tarif d'Amazon S3 est indépendant du nombre d'instances EC2. Cette facturation c_s est une fonction linéaire par morceaux (Formule 2) et le coût de stockage peut être exprimé comme suit :

$$C_s(D, IC) = \sum_{k=1}^{n_D} c_s(s(D_k)) \times t(D_k) \quad (7)$$

Exemple 3 *Avec Amazon S3 (Table 1(c) pour la tarification) et deux instances EC2 de type "small", on considère que 0.5 To (512 Go) de données ont été stockées pendant 12 mois. Au début du 8ème mois, on insère 2 To (2048 Go) de nouvelles données dans le nuage. Ainsi, nous avons deux périodes, et à cause de leur volume de données, le segment 1 de la fonction de coût est considéré pour la période 1, et le segment 2 pour la période 2. Le coût de stockage complet est : $C_s(D, IC) = c_1(s(D_1)) \times t(D_1) + c_2(s(D_2)) \times t(D_2) = (a_1 \times (s(D_1) - x_1) + b_1) \times t(D_1) + (a_2 \times (s(D_2) - x_2) + b_2) \times t(D_2) = (0,095 \times (512 - 0) + 0) \times 7 + (0,08 \times ((512 + 2048) - 1024) + 1024 \times 0,095) \times 5 = 1441,28$ \$.*

4 Coût de la matérialisation de vues dans un nuage

Cette section présente des modèles de coût pour la matérialisation de vues dans le nuage, en s'appuyant sur les modèles de coût développés dans la Section 3. Nous supposons ici que

les requêtes sont exécutées sur un nombre constant n_{IC} d'instances de calcul identiques IC_0 ($IC_j = IC_0,\ \forall j = 1..n_{IC}$). Dans de futurs travaux, nous devrons considérer le processus d'évaluation sur de multiples instances variables.

Soit $V_{cand} = \{V_k\}_{k=1..n_V}$ l'ensemble des vues candidates à la matérialisation fourni par une technique existante de sélection de vues (Section 2.2). Dans cette section, nous supposons que les vues à matérialiser ont été sélectionnées côté client, produisant un ensemble final de vues $V \subset V_{cand}$ qui sont matérialisées dans le nuage. Le problème de choisir le meilleur ensemble de vues V à partir de V_{cand} en s'appuyant sur les modèles de coût présentés ici sera traité dans la Section 6.

Il est important de noter que la matérialisation étant réalisée dans le nuage, les coûts de transfert dus à la matérialisation sont nuls. En conséquence, le coût total de transfert de données C_t n'est pas impacté et reste exprimé par la formule 3.

4.1 Coût de calcul

Utiliser les vues matérialisées implique de modifier le modèle du coût de calcul, dès lors que le traitement des requêtes exploite les vues matérialisées et que les vues doivent être matérialisées et maintenues. Le coût de calcul C_c dépend maintenant de V : $C_c(Q, V, IC)$.

En appliquant le modèle de tarification d'Amazon, et en considérant qu'on utilise dans le nuage un nombre constant d'instances de calcul identiques [5], la Formule 5 devient alors :

$$C_c(Q, V, IC) = T(Q, V) \times c_c(IC_0) \times n_{IC}. \tag{8}$$

$T(Q, V)$ est le temps total de calcul, ce qui correspond à la somme du temps $T_{proc}(Q, V)$ de traitement des requêtes de la charge Q en utilisant l'ensemble V de vues matérialisées, du temps $T_{mat}(V)$ de matérialisation de ces vues, et du temps $T_{maint}(V)$ pour leur maintenance. En conséquence, ce temps peut être exprimé comme :

$$T(Q, V) = T_{proc}(Q, V) + T_{mat}(V) + T_{maint}(V). \tag{9}$$

Pour matérialiser une vue, la requête associée doit être exécutée, ce qui doit être payé dans le nuage. Notons $t_{mat}(V_k)$ le temps pour matérialiser la vue V_k. Le temps total de matéralisation est :

$$T_{mat}(V) = \sum_{V_k \in V} t_{mat}(V_k). \tag{10}$$

Le coût de la maintenance des vues matérialisées est directement proportionnel au temps requis pour mettre à jour les vues matérialisées quand elles sont impactées par des modifications des données source. A noter que nous considérons que l'interrogation et la maintenance ne se produisent pas au même moment. Par exemple, les requêtes sont posées pendant la journée et la maintenance est réalisée pendant la nuit. Soit $t_{maint}(V_k)$ le temps de maintenance de la vue V_k, le temps total pour la maintenance de V est :

$$T_{maint}(V) = \sum_{V_k \in V} t_{maint}(V_k). \tag{11}$$

Quand les vues matérialisées sont utilisées, le temps de traitement des requêtes est défini par deux paramètres : la charge des requêtes Q et l'ensemble des vues matérialisées V. Les

5. Considérer un nombre variable d'instances est en dehors du champ d'étude de cet article et fera l'objet de futurs travaux.

requêtes peuvent utiliser le contenu des vues matérialisées au lieu de recalculer leur résultat. A noter que nous considérons une charge Q fixe, une charge variable est laissée pour de futurs travaux. Comme les vues sont généralement matérialisées en fonction d'une charge donnée et que la nôtre est fixe, alors V est également fixe. Soit $t(Q_i, V)$ le temps de traitement de la requête Q_i en exploitant l'ensemble de vues matérialisées V, le temps total de traitement de Q à l'aide de V est alors :

$$T_{proc}(Q, V) = \sum_{i=1}^{n_Q} t(Q_i, V). \tag{12}$$

4.2 Coût de stockage

Utiliser les vues matérialisées n'a pas d'impact sur le modèle du coût de stockage tel qu'il est présenté par les Formules 6 et 7. Exploiter les vues matérialisées pour améliorer les performances des requêtes implique de stocker ces vues dans le nuage et de payer le coût correspondant. En conséquence, certaines données sont dupliquées. Dans ce cas, la taille de $D + V$, c'est-à-dire $s(D) + \sum_{V_k \in V} s(V_k)$, est utilisée à la place de la taille $s(D)$ de D seule pour la facturation. Ainsi, le modèle du coût de stockage dépendant de D et V peut être exprimé comme suit :

$$C_s(D, V, IC) = C_s(D + V, IC). \tag{13}$$

A noter que nous supposons que les données originales et les vues matérialisées sont stockée pendant toute la période de stockage considérée.

Exemple 4 *Avec Amazon S3, l'ensemble de données de 0,5 To est stocké pendant une année, et la taille des données dupliquées par la matérialisation des vues est 50 Go. En outre, aucune donnée n'est insérée pendant la période considérée. Ainsi, nous avons une seule période, et le coût de stockage est* $C_s(D, V, IC) = (512 + 50) \times 0.095 \times 12 = 641$ \$.

5 Optimiser la matérialisation de vues dans le nuage

Dans cette section, nous étudions comment sélectionner les vues à matérialiser dans le but d'améliorer la performance des requêtes avec un minimum de surcoût de stockage. Nous définissons des problèmes d'optimisation pour sélectionner le meilleur ensemble de vues matérialisées en exploitant les modèles de coût introduits à la section 4. Ces problèmes sont exprimés ici comme des programmes linéaires avec des variables continues et discrètes qui peuvent être résolus efficacement en utilisant un solveur de programmation entière mixte (MIP) tel que CPLEX.

5.1 Objectifs d'optimisation

A partir des idées de Kllapi et al. (2011), nous proposons trois problèmes d'optimisation, notés MV_1 à MV_3, avec différents objectifs pour satisfaire les besoins et les contraintes des utilisateurs.

Problème MV_1 : trouver un ensemble de vues V qui minimise le temps de réponse T_{proc} avec une limite de budget C_{max} sur le coût total C.

$$(MV_1) \begin{cases} \text{minimiser} \quad T_{proc} \\ \\ \text{avec} \quad C \leq C_{max} \\ \qquad \textit{et les contraintes qui définissent :} \\ \qquad \textit{- le modèle de coût} \\ \qquad \textit{- la sélection des vues} \end{cases} \tag{14}$$

Problème MV_2 : trouver un ensemble de vues V qui minimise le coût total C avec une limite T_{max} sur le temps de réponse T_{proc}.

$$(MV_2) \begin{cases} \text{minimiser} \quad C \\ \\ \text{avec} \quad T_{proc} \leq T_{max} \\ \qquad \textit{et les contraintes qui définissent :} \\ \qquad \textit{- le modèle de coût} \\ \qquad \textit{- la sélection des vues} \end{cases} \tag{15}$$

Résoudre le problème d'optimisation multicritère (c'est-à-dire minimiser à la fois C et T_{proc}) n'est pas directement traité dans cet article, car cela nécessite des techniques d'optimisation spécifiques (Coello et al. (2007)) qui diffèrent des techniques d'optimisation monocritère. Il n'est pas possible de fournir une solution qui optimise les deux objectifs à la fois. Cependant, des solutions non dominées peuvent être trouvées (des solutions de la frontière de Pareto), c'est-à-dire des solutions qui ne peuvent pas être améliorées sur l'un des objectifs sans empirer l'autre objectif, Ehrgott (2005). Le modèle présenté ici est totalement valide pour une optimisation multicritère, en retirant les limites sur le budget et le temps de réponse.

Néanmoins, dans le but de percevoir la nature des solutions qui visent à optimiser les deux objectifs, nous proposons d'optimiser le problème MV_3 suivant, où un coefficient α indique l'importance relative du critère de temps de réponse par rapport au critère de coût. Un tel coefficient doit être fixé par le gestionnaire de l'entrepôt de données.

Problème MV_3 : trouver un ensemble de vues V qui est un compromis entre un temps de réponse T_{proc} minimum et un coût total C minimum.

$$(MV_3) \begin{cases} \text{minimiser} \quad \alpha \times T_{proc} + (1 - \alpha) \times C \\ \\ \text{avec} \quad \textit{les contraintes qui définissent :} \\ \qquad \textit{- le modèle de coût} \\ \qquad \textit{- la sélection des vues} \end{cases} \tag{16}$$

5.2 Programme linéaire

Dans le but de sélectionner un ensemble de vues V à matérialiser, nous utilisons un algorithme existant tel que celui de Baril et Bellahsene (2003) qui permet d'obtenir un ensemble de vues candidates pour la matérialisation $V_{cand} = \{V_k\}_{k=1..n_V}$. Dans une première étape, nous posons des hypothèses sur V_{cand}. Notamment, pour déterminer le gain apporté par l'utilisation d'une vue ou plusieurs vues pour une requête donnée, nous devons soit connaître suffisamment de détails sur le fonctionnement du nuage pour exprimer analytiquement ce gain, soit réaliser des expériences pour le mesurer ou l'évaluer.

Par simplification, nous supposons ici que chaque requête Q_i peut utiliser une seule vue parmi l'ensemble des vues candidates, autrement dit, pour chaque requête Q_i, il y a un ensemble $V^i \subset V$ de vues candidates, et pas plus d'une vue dans cet ensemble ne doit être sélectionnée pour la requête Q_i. Dans de futurs travaux, nous pourrons considérer les vues candidates d'une requête Q_i comme étant un ensemble $V^i = \left\{V^{ij}\right\}_{j=1..n_i}$ qui contient n_i ensembles de vues candidats $V^{ij} \subset V$. Il s'agira alors de choisir un ensemble de vues V^{ij} pour une requête Q_i au lieu d'une seule vue.

Le problème d'optimisation doit sélectionner au mieux une vue V_k pour chaque requête Q_i. Dans ce but, des variables de décision x_{ik} sont introduites : $x_{ik} = 1$ si la requête Q_i utilise la vue V^k, et $x_{ik} = 0$ sinon. Soit g_{ik} le gain sur le temps de réponse d'une requête Q_i en utilisant la vue V_k. Les gains sont considérés constants dans ce problème, autrement dit, ils ont été mesurés ou estimés en amont (à partir d'expériences, de statistiques ou de modèles). Le temps de réponse de la requête Q_i en utilisant les vues est exprimé comme suit :

$$t(Q_i, V) = t_i - \sum_{k=1}^{n_V} g_{ik}\, x_{ik}, \tag{17}$$

où t_i est le temps de réponse de la requête Q_i sans utiliser aucune vue, et en supposant qu'il n'y a pas plus d'une vue sélectionnée pour la requête Q_i. Ce dernier point est garanti par les contraintes suivantes :

$$\sum_{k=1}^{n_V} x_{ik} \leq 1,\ \forall i = 1..n_Q. \tag{18}$$

Des variables de décision x_k sont également introduites pour déterminer si une vue V_k est matérialisée : $x_k = 1$ si la vue V_k est matérialisée, et $x_k = 0$ sinon. Une vue V_k est matérialisée si elle est utilisée par au moins une requête (quand il existe au moins une requête Q_i telle que $x_{ik} = 1$), ce qui s'exprime par les contraintes suivantes :

$$x_{ik} \leq x_k,\ \forall i = 1..n_Q,\ \forall k = 1..n_V. \tag{19}$$

En outre, il n'est pas nécessaire de matérialiser V_k si elle n'est pas utilisée du tout, ce qui s'exprime par les contraintes suivantes :

$$x_k \leq \sum_{i=1}^{n_Q} x_{ik},\ \forall k = 1..n_V. \tag{20}$$

Comme pour les gains sur les temps de réponse, les temps de matérialisation et de maintenance ($t_{mat}(V_k)$ et $t_{maint}(V_k)$) sont estimés en amont. Ces temps doivent être considérés pour une vue V_k uniquement si celle-ci est matérialisée :

$$\begin{aligned} T_{mat}(V) &= \sum_{k=1}^{n_V} t_{mat}(V_k)\, x_k, \\ T_{maint}(V) &= \sum_{k=1}^{n_V} t_{maint}(V_k)\, x_k. \end{aligned} \tag{21}$$

Tant que nous supposons qu'il n'y a pas de données insérées dans l'entrepôt durant son exploitation, nous pouvons considérer une seule période de longueur $t(D)$ pour le stockage.

La taille S de ces données et son coût de stockage sont exprimés comme suit, avec c_s la fonction du coût de stockage (nous supposons qu'elle est linéaire par morceaux ; Section 3.4) :

$$\begin{aligned} S &= s(D) + \sum_{k=1}^{n_V} s(V_k)\, x_k, \\ C_s(D, V, IC) &= c_s(S)\, t(D), \end{aligned} \tag{22}$$

Notons que le coût de transfert C_t n'est pas impacté par la sélection des vues. Il s'agit d'une valeur constante dans le problème d'optimisation et reste exprimée par la Formule 3. En résumé, le problème d'optimisation complet, par exemple MV_1 (MV_2 et MV_3 étant très similaires), est finalement exprimé comme suit [6] :

$$MV_1 \begin{cases} \text{minimiser} \quad T_{proc} = \displaystyle\sum_{i=1}^{n_Q} \left(t_i - \sum_{k=1}^{n_V} g_{ik}\, x_{ik} \right) & \\ \text{avec} \quad C = C_c + C_t + C_s \leq C_{max} & C_s = c_s(S)\, t(D) \\ \qquad C_c = (T_{proc} + T_{mat} + T_{maint})\ c_c(IC_0)\, n_{IC} & S = s(D) + \displaystyle\sum_{k=1}^{n_V} s(V_k)\, x_k \\ \qquad \displaystyle\sum_{k=1}^{n_V} x_{ik} \leq 1,\ \forall i = 1..n_Q & T_{mat} = \displaystyle\sum_{k=1}^{n_V} t_{mat}(V_k)\, x_k \\ \qquad x_{ik} \leq x_k,\ \forall i = 1..n_Q,\ \forall k = 1..n_V & T_{maint} = \displaystyle\sum_{k=1}^{n_V} t_{maint}(V_k)\, x_k \\ \qquad x_k \leq \displaystyle\sum_{i=1}^{n_Q} x_{ik},\ \forall k = 1..n_V & \\ \qquad x_{ik} \in \{0,1\},\ \forall i = 1..n_Q,\ \forall k = 1..n_V & \\ \qquad x_k \in \{0,1\},\ \forall k = 1..n_V & \end{cases} \tag{23}$$

A noter que nous avons établi que c_s est linéaire par morceaux. Elle peut donc être reformulée avec des contraintes linéaires sur des variables continues et discrètes Chen et al. (2010).

6 Validation expérimentale

Des expérimentations ont été menées afin de valider l'utilisation des modèles de coût pour atteindre les différents objectifs. Cette section décrit l'environnement expérimental utilisé et les résultats obtenus. Les expériences ont été réalisées à la fois dans le nuage et côté client. L'ensemble des données et des vues matérialisées est stocké dans le nuage et les requêtes sont traitées dans le nuage. Les algorithmes de sélection des vues sont exécutés côté client. L'idée est en effet de sélectionner les vues du côté du client et de les matérialiser dans le nuage.

6.1 Environnement expérimental

Les expériences ont été menées sur une grappe composée de 20 machines virtuelles avec 8 Go de disque dur, 2 Go de RAM et 1 vCPU. L'architecture physique correspond à douze

6. Pour des raisons de clareté, la notation est simplifiée : les paramètres Q, D, V et IC ont été masqués.

quadri-pros 800MHz avec 96 Go de RAM. Toutes les machines disposent de Hadoop (version 0.20.2)[7] et Pig (version 0.9.1) Olston et al. (2008). La configuration du client n'a pas d'incidence sur les résultats présentés. C'est pourquoi il n'est pas pertinent de la détailler.

Il existe deux façons principales pour générer des charges de travail afin de tester ce type de configuration. La première est l'utilisation de traces réelles. Cette approche permet généralement de fournir une bonne approximation pour des cas réels d'utilisation. Cependant, une trace ne représente qu'un cas particulier et ne permet souvent pas de représenter la réalité dans sa globalité. De plus, si l'objectif principal est de comprendre pourquoi une solution est adaptée à un contexte donné, l'utilisation de trace sera insuffisante pour mettre en lumière les mécanismes en action. La seconde approche correspond à l'utilisation d'une charge de travail synthétique. L'inconvénient principal d'une telle solution provient de sa nature artificielle. Cependant, cela permet de jouer sur les configurations. Ainsi, si des traces sont disponibles, il est possible de les utiliser pour choisir le modèle et son calibrage. Le choix du modèle est très important pour offrir une bonne représentation du contexte ciblé. Notre but est de mettre en avant l'intérêt de notre approche pour les entrepôts de données. C'est pourquoi nous avons choisi d'utiliser le banc d'essais Star Schema Benchmark O'Neil et al. (2009) (version 2.1.8.18). Les requêtes sont écrites en Pig Latin et sont exécutées par le compilateur Pig en terme de tâches MapReduce Dean et Ghemawat (2008) via une grappe Hadoop.

6.2 Résultats expérimentaux

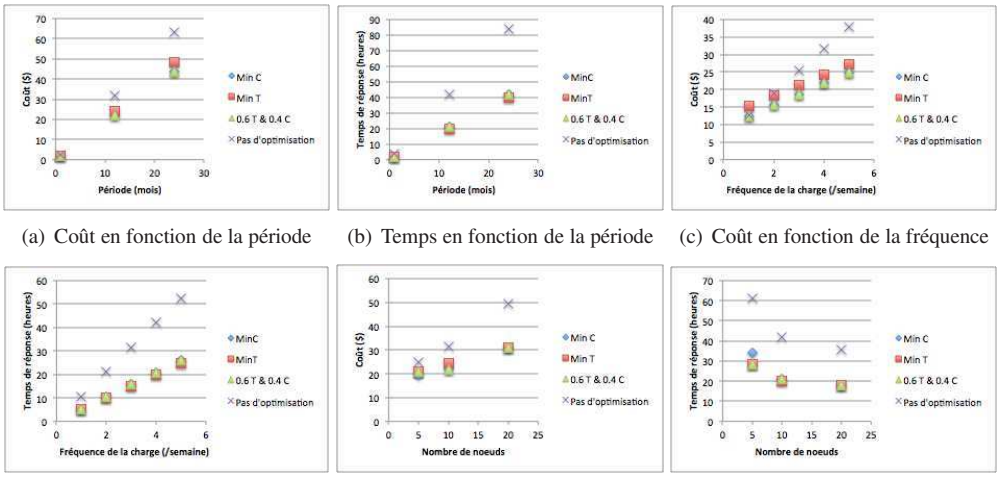
Nous avons testé les scénarios MV_1, MV_2 et MV_3 décrits dans la Section 5.1 à l'aide d'un jeu de données de 5.5 Go et les tarifications d'Amazon S3 et EC2. Les mesures sont réalisées selon des paramètres variables. Le premier paramètre est la période d'expérimentation, donnée en mois et allant de 1 à 24. Le deuxième paramètre est la fréquence du banc d'essais, autrement dit le nombre de fois où le banc est exécuté pendant la période étudiée, donné en nombre d'exécutions par semaine et allant de 1 à 5. Le troisième paramètre est le nombre de nœuds utilisés pour évaluer les requêtes, allant de 5 à 20. Une expérience type fixe deux des paramètres en faisant varier le dernier. Les valeurs fixes utilisées pour la période d'expérimentation, la fréquence de la charge de travail et le nombre de nœuds sont respectivement 12, 4 et 10.

La figure 3 présente les résultats obtenus. Ces résultats démontrent clairement que notre approche permet de sélectionner des vues qui améliorent considérablement à la fois le temps de réponse et le prix à payer. En effet, le temps de réponse est diminué d'environ un facteur 2 (par exemple, avec une fréquence égale à 4 fois par semaine, pendant 12 mois et sur une grappe de 10 nœuds, le temps de réponse est de 20 heures avec des vues matérialisées, alors qu'il correspond à 42 heures quand les vues matérialisées sont ignorées) et le coût est jusqu'à deux fois moins important (par exemple, pour une fréquence égale à 4 fois par semaine, pendant 12 mois et sur une grappe de 10 nœuds, le coût avec vues est de 22 $, alors que celui sans vue est de 32 $).

Notre solution permet donc d'atteindre l'objectif fixé. Quand le but est de réduire le coût, l'ensemble des vues à matérialiser permet de payer le minimum. Quand l'objectif est la performance, cet ensemble conduit au temps de réponse minimum. Il convient cependant de noter que dans le contexte proposé, les objectifs ne sont pas réellement contradictoires. Ainsi, la différence entre le coût et le temps n'est relativement pas majeure (environ 10 % pour le coût

7. Hadoop : http ://hadoop.apache.org/

et 5 % pour le temps). Cependant, ces objectifs peuvent conduire à la matérialisation de vues différentes, dont l'impact ne serait pas si négligeable à très grande échelle, où un gain de 5 à 10 % se mesure en milliers de dollars et en heures de calcul.



(a) Coût en fonction de la période (b) Temps en fonction de la période (c) Coût en fonction de la fréquence

(d) Temps en fonction de la fréquence (e) Coût en fonction des nœuds (f) Temps en fonction des nœuds

FIG. 3 – *Résultats expérimentaux.*

7 État de l'art et positionnement

Nous présentons dans cette section les travaux relatifs aux différents domaines abordés dans cet article : l'optimisation de l'accès aux données grâce aux vues matérialisées et les modèles de coûts conçus pour les systèmes distribués à grande échelle.

Dans le nuage, la question de la performance est majoritairement gérée via l'élasticité en terme de puissance de calcul. Pour autant, les techniques d'optimisation des performances bien connues dans le domaine des bases de données, comme la matérialisation de vues, peuvent également être mises à profit pour abaisser le coût global. De nombreuses approches ont été proposées pour sélectionner Yang et al. (1997); Mami et Bellahsene (2012) et maintenir Ceri et Widom (1991); Zhou et al. (2007); Luo et al. (2003); Agrawal et al. (2009) les vues matérialisées, que ce soit dans les bases de données transactionnelles, spécifiquement dans les entrepôts de données ou même sur le Web. Nos modèles de coût visent à étendre ces stratégies en y incluant le modèle économique de paiement à la demande des nuages informatiques, afin de trouver le meilleur compromis coût de stockage (vues matérialisées comprises) / coût de calcul.

Les modèles de coûts existants dans ce domaine traitent l'ordonnancement de flux de données en fonction du coût monétaire et du temps d'exécution de leur traitement Kllapi et al. (2011), l'amortissement du coût des structures de données pour assurer la viabilité économique du fournisseur de service Kantere et al. (2011) ainsi que d'autres techniques d'optimisation des performances Upadhyaya et al. (2012). D'autres modèles de coût ont été développés pour des applications spécifiques, notamment en astronomie. Dans ce domaine, des expériences de si-

mulation ont montré qu'un bon compromis entre stockage de structures d'optimisation et puissance de calcul permet de réduire le coût global de traitement des données dans le nuage (en l'occurrence, la solution gratuite d'Amazon) sans amoindrir les performances Deelman et al. (2008). Notre travail vient en complément de ces modèles et s'en différencie de deux manières principales. Premièrement, nous considérons un modèle de facturation suffisamment fin, qui est à même de représenter ceux de tous les fournisseurs de service. Deuxièmement, notre proposition s'appuie sur un modèle détaillé du processus d'optimisation menant à la sélection des vues matérialisées. Elle est également indépendante de toute application particulière.

8 Conclusion

Cet article propose une approche pour améliorer la gestion des données dans les nuages à l'aide des vues matérialisées. Nos contributions principales sont de nouveaux modèles de coûts qui complètent les modèles de coûts des vues matérialisées existants avec la facturation à l'utilisation des nuages. Ensuite, nous exploitons ces modèles dans un processus d'optimisation qui fournit un compromis entre l'amélioration des performances liée à la matérialisation et les contraintes budgétaires. Nos premières expérimentations démontrent l'intérêt de l'approche.

Ce travail ouvre de nombreuses perspectives : (i) enrichir les modèles pour palier les limites énoncées (instances multiples variables, etc.) ; (ii) tester les modèles avec d'autres tarifications ; (iii) intégrer les modèles dans les algorithmes de sélection de vues existants et comparer les différentes solutions ; (iv) prendre en compte d'autres techniques d'optimisation (index, caches, etc.) ; (v) valider nos propositions à plus grande échelle et pour d'autres requêtes (se prêtant facilement ou non au parallélisme) ; et (vi) proposer des algorithmes plus rapides que les solveurs MIP pour la résolution des problèmes d'optimisation (e.g. métaheuristiques).

Remerciements

Ce travail est financé par les projets SYSEO (ANR-10-TECSAN-005-01) et GOD (STIC-Asie-2012). Nous tenons à remercier Thi Van Anh Ngyen et les membres d'ERIC, de l'Irstea et du LIMOS.

Références

Agrawal, P., A. Silberstein, B. F. Cooper, U. Srivastava, et R. Ramakrishnan (2009). Asynchronous view maintenance for vlsd databases. In *SIGMOD*, Providence, USA, pp. 179–192.

Aouiche, K. et J. Darmont (2009). Data mining-based materialized view and index selection in data warehouses. *JIIS 33*(1), 65–93.

Baril, X. et Z. Bellahsene (2003). Selection of materialized views : A cost-based approach. In *CAISE*, pp. 665–680.

Ceri, S. et J. Widom (1991). Deriving production rules for incremental view maintenance. In *VLDB*, pp. 577–589.

Chen, D.-S., R. G. Batson, et Y. Dang (2010). *Applied Integer Programming : Modeling and Solution*. John Wiley and Sons.

Coello, C. A. C., G. B. Lamont, et D. A. V. Veldhuisen (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer.

Dean, J. et S. Ghemawat (2008). Mapreduce : simplified data processing on large clusters. *Communications of the ACM 51*(1), 107–113.

Deelman, E., G. Singh, M. Livny, G. B. Berriman, et J. Good (2008). The cost of doing science on the cloud : the montage example. In *SC*, pp. 50.

Ehrgott, M. (2005). *Multicriteria Optimization*. Springer.

Kantere, V., D. Dash, G. Gratsias, et A. Ailamaki (2011). Predicting cost amortization for query services. In *SIGMOD*, pp. 325–336.

Kllapi, H., E. Sitaridi, M. M. Tsangaris, et Y. E. Ioannidis (2011). Schedule optimization for data processing flows on the cloud. In *SIGMOD*, pp. 289–300.

Luo, G., J. F. Naughton, C. J. Ellmann, et M. Watzke (2003). A comparison of three methods for join view maintenance in parallel rdbms. In *ICDE*, pp. 177–188.

Mami, I. et Z. Bellahsene (2012). A survey of view selection methods. *SIGMOD Record 41*(1), 20–29.

Nguyen, T.-V.-A., S. Bimonte, L. d'Orazio, et J. Darmont (2012). Cost models for view materialization in the cloud. In *EDBT Workshops*, pp. 47–54.

Olston, C., B. Reed, U. Srivastava, R. Kumar, et A. Tomkins (2008). Pig latin : a not-so-foreign language for data processing. In *SIGMOD*, pp. 1099–1110.

O'Neil, P. E., E. J. O'Neil, X. Chen, et S. Revilak (2009). The star schema benchmark and augmented fact table indexing. In *TPCTC*, pp. 237–252.

Upadhyaya, P., M. Balazinska, et D. Suciu (2012). How to price shared optimizations in the cloud. *PVLDB 5*(6), 562–573.

Yang, J., K. Karlapalem, et Q. Li (1997). Algorithms for materialized view design in data warehousing environment. In *VLDB*, pp. 136–145.

Zhou, J., P.-Å. Larson, et H. G. Elmongui (2007). Lazy maintenance of materialized views. In *VLDB*, pp. 231–242.

Summary

Data warehouses performance is usually achieved through physical data structures such as indexes or materialized views. In such a context, several cost models enable to select a relevant set of these structures. Nevertheless, this selection becomes more complex in the cloud. Indeed, the criterion to optimize is at least two-dimensional, with the monetary cost balancing the overall query response time. This paper introduces new cost models that fit into the pay-as-you-go paradigm of cloud computing. Based on these cost models, an optimization problem is defined to find, among candidate views, those to be materialized to minimize both the overall cost of using and maintaining the database in a cloud and the total response time to a given set of queries. First, we optimize the two objectives separately: one criterion is optimized under a bounding constraint on the other one. Our proposal is validated through experiments.