République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université A/Mira de Béjaïa
Faculté des Sciences Exactes
Département d'Informatique

École doctorale Réseaux et Systèmes Distribués

Mémoire de Magister

En Informatique

Option : *Réseaux et Systèmes Distribués*

Thème

Conception de bases de données volumineuses sur le Cloud

Présenté par :
Brighen Assia.

Soutenu devant le jury composé de :

Président	Mr TARI Abdelkamel	MCA	U. Béjaia.
Rapporteur	Mr BELLATRECHE Ladjel	Professeur	U. de Poitiers, France.
Examinateur	Mr MELIT Ali	Professeur	U. Jijel.
Examinateur	Mr IDOUGHI Djilali	MCA	U. Béjaia.
Invité	Mr SLIMANI Hachem	MCB	U. Béjaia.

Béjaïa, septembre 2012.

⁕ Remerciements ⁕

Je tiens à remercier en premier lieu le Dieu le tout puissant et miséricordieux, qui m'a donné la force et la patience d'accomplir ce Modeste travail.

En second lieu, je tiens à remercier Mr L. Bellatreche, le Directeur de mémoire, pour m'avoir accordé l'honneur de travailler avec lui et pour ses orientations et ses précieux conseils et son aide durant toute la période du travail.

Je tiens à remercier sincèrement Monsieur H. Slimani , qui, en tant que Co-directeur de mémoire, s'est toujours montré à l'écoute et très disponible tout au long de la réalisation de ce mémoire, ainsi pour l'inspiration, l'aide et le temps qu'il a bien voulu me consacrer et sans qui ce travail n'aurait pas pu être mené au bon port.

Mes remerciements s'adressent aussi à Mr TARI Abdelkamel, Mr. MELLIT Ali, et Mr. IDOUGHI Djilali, qui me font le grand honneur d'accepté d'examiner mon travail.

Merci pour Mr A. TARI, le responsable de l'école doctorale ReSyD, pour les efforts qu'il a fourni pour que notre formation puisse se dérouler.

Enfin, j'adresse mes plus sincères remerciements à tous mes amies, qui m'ont toujours soutenu et encouragé au cours de la réalisation de ce mémoire.

⁂ *Dédicaces* ⁂

Je dédie ce modeste travail à :

⁂ *Mes très chers parents ;*

⁂ *Mes frères ;*

⁂ *Toutes ma famille ;*

⁂ *Mes amies et collègues.*

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

ACID Atomicity, Consistency, Isolation and Durability

Amazon EC2 Amazon Elastic Cloud Computing

Amazon S3 Amazon Simple Storage Service

AWS Amazon Web Services

BD Base de Données

BI Business Intelligence

CAP Consistent, Availability, Partition tolerance

CAPEX Capital Expenditure

DDL Data Description Language

DFS Distributed File System

DML Data Manipulation language

ETL procédures d'Extraction, de Transfert et de Consolidation

FH Fragmentation Horizontale

FV Fragmentation Verticale

GFS Google File System

HDFS Hadoop Distributed File System

IaaS Infrastructure as a Service

JVM Java Virtuelle Machine

OLAP On-Line Analytical Processing

OPEX Operating Expenses

NoSQL Not Only SQL

PaaS Plate-forme as a Service

PME Petites et Moyennes Entreprises

RPC Remote Procedure Call

SaaS Software as a Service

SCOPE Structured Computations Optimized for Parallel Execution

SGBD Système de Gestion de Base De Données

SSBM Star Schema Benchmark

VMs Vues Matérialisées

VMR Vues MapReduce

UDF User Define Function

Introduction générale

Le Cloud Computing, "l'informatique dans les nuages" en français, est un nouveau concept qui consiste à externaliser l'infrastructure informatique vers des sociétés spécialisées, qui proposent les ressources informatiques (matérielles et logicielles) à la demande. Ces ressources sont accessibles via une connexion internet de n'import quel lieu, par n'importe qui et à tout moment. Avec ce nouveau paradigme, la consommation des ressources informatiques est devenue similaire à celle de l'électricité. Les applications, la puissance de calcul et les différentes ressources informatiques sont proposées comme un service par un prestataire et facturé après l'utilisation selon un modèle de tarification.

Aujourd'hui, le marché Cloud a reçu beaucoup d'attention et il est de plus en plus en croissance. Ce marché est partagé entre plusieurs fournisseurs, tels qu'Amazon et Microsoft. Chaque fournisseur propose différents fonctionnalités selon un modèle de tarif approprié. Les entreprises, particulièrement les petites et moyennes entreprises (PME), veulent bénéficier de ce nouveau modèle qui leur permet de réduire les coûts et les délais liés aux déploiement d'une infrastructure informatique et les promis par une grande puissance de calcul, un stockage illimité et une haute disponibilité.

Sur le Cloud on trouve de nombreuses types d'applications dont la plupart nécessite la gestion de grande quantité de données. Parmi ces applications, on trouve la BI (Business Intelligence). La BI nécessite l'analyse d'une grande masse de données qui sont généralement stockées au sein d'une base de données volumineuse qu'on appelle entrepôt de données. Les entrepôts de données sont généralement modélisés par un schéma en étoile [40], contenant une table de faits centrale de très grande taille liée à un certain nombre de tables de dimension de plus petite taille.

Les entrepôts de données forment la base des processus d'aide à la décision, qui nécessitent des requêtes décisionnelles complexes et très coûteuses en temps de traitement (des heures et peut être des jours), car ces requêtes requièrent des jointures en étoile entre la table de faits et les tables de dimension. Plusieurs techniques d'optimisation sont utilisées pour améliorer la performance de ces requêtes, telles que les index, les vues matérialisées et la fragmentation horizontale.

Pour chaque objet d'optimisation (Index, Vue, ...) plusieurs configurations sont possibles. Durant la conception physique, l'administrateur doit choisir la configuration optimale ou proche de l'optimale. Le choix de la configuration optimale des structures est basé sur l'evaluation de la qualité des configurations générées, qui peut s'effectuer à l'aide de modèles de

coût. À cet effet, plusieurs modèles de coût analytiques ont été élaborés. Ces modèles sont caractérisés par leur simplicité et leur facilité à mettre en oeuvre. Cependant, dans un environnement Cloud Computing, ces modèles de coût ne sont pas directement applicables, du fait qu'ils ne prennent pas en considération les caractéristiques typiques du Cloud, tels que le modèle de paiement.

En outre que le modèle de paiement, un environnement Cloud Computing se diffère d'un environnement traditionnel par ses nouvelles technologies qui sont développées pour faire face à la volumétrie de données et la charge de travail. Les outils traditionnels de gestion de données sont insuffisantes pour le traitement et la gestion de grandes quantités de données dont la taille variée de quelques centaines de téraoctets à des dizaines de pétaoctets. Le stockage de données est basé sur des nouveaux systèmes de fichiers distribués et le traitement de données est réalisé par des nouveaux paradigmes dont le plus populaire est MapReduce de Google [25] et son implémentation open source Hadoop [37]. Ainsi, l'interrogation de données est réalisée par des nouveaux langages d'interrogation, qui sont conçus spécialement pour cacher les détails de l'environnement d'exécution et pour simplifier la complexité des paradigmes d'exécution.

Problématique et objectifs du travail

Notre travail entre dans le cadre de la conception physique des entrepôts de données, qui consiste à construire une configuration physique de l'entrepôt de données sur le support de stockage. La conception physique s'appuie particulièrement sur la sélection des structures d'optimisation (Index, vues matérialisées, fragmentation, ...), pour optimiser les requêtes de jointure en étoile. Dans le contexte des entrepôts de données, les VMs permet de réduire de manière efficace le temps de réponse des requêtes décisionnelles, mais engendre des coûts supplémentaires : le coût de stockage et de maintenance.

Le stockage sur le Cloud est supposé illimité mais facturé en fonction de la quantité de données stockées. Matérialiser les résultats des requêtes ou augmenter les ressources informatiques (nombre de nœuds et la puissance du calcul) pour avoir un coût faible de traitement de la charge de requêtes, est un choix décisif. Ce choix nécessite la présence d'un modèle de coût pour évaluer le meilleur choix. Ainsi, l'utilisation d'un algorithme de sélection des VMs implique l'utilisation d'un modèle de coût pour quantifier les différentes configurations possibles.

L'objectif de notre travail est d'élaborer un modèle de coût analytique pour la sélection des vues matérialisées sur le Cloud dans un environnement MapReduce, en prenant en compte les caractéristiques du Cloud, notamment le modèle de paiement et l'environnement d'exécution. Ce modèle de coût doit être capable de comparer entre la matérialisation des requêtes et l'augmentation des ressources informatiques ainsi que de diriger les algorithmes de sélection des vues matérialisées.

Organisation du mémoire

Nous avons organisé notre mémoire en cinq chapitres :

- ✓ Le premier chapitre est consacré au concept Cloud Computing. À travers ce chapitre, nous essayons de bien clarifier le terme Cloud Computing et de présenter les différents concepts liés à ce nouveau modèle, ses principales caractéristiques, ses acteurs et leurs tarifications ainsi que les avantages et les inconvénients du recours à ce nouveau modèle.
- ✓ Le deuxième chapitre est dédié à l'étude de la notion de gestion de données sur le Cloud. Nous présentons les principaux systèmes de fichiers distribués de ce dernier, les nouveaux paradigmes ainsi que ses langages d'interrogation aux données.
- ✓ Le troisième chapitre est consacré à la description générale des notions de l'entrepôt de données et la définition des problématiques liées à la conception physique des entrepôts de données sur le Cloud. Le chapitre présente aussi les différents algorithmes d'accès aux données sur le cloud utilisant le modèle de programmation le plus populaire MapReduce.
- ✓ Dan le quatrième chapitre, nous proposons un nouveau modèle de coût analytique pour la conception physique des entrepôts de données sur le Cloud dans un environnement MapReduce, en choisissant les VMs comme structure d'optimisation.
- ✓ Pour l'évaluation du modèle de coût théorique élaboré, nous exécutons dans le cinquième chapitre un ensemble de requêtes sur un entrepôt de données réel issu d'un benchmark dans un système MapReduce. Ensuite, nous comparons notre modèle de coût aux résultats obtenus par des tests pratiques.

Enfin, nous terminons par une conclusion dans laquelle nous dressons un bilan et une synthèse du travail effectué, et nous discutons un ensemble de perspectives qui peuvent être considérées comme des directions de recherche future.

Chapitre 1

Cloud Computing : Une nouvelle façon de consommation des ressources informatiques

1.1 Introduction

Le Cloud Computing est un nouveau paradigme qui fait référence à la consommation des ressources informatiques à la demande, à travers un réseau mondial. Ce nouveau concept est très similaire à celui de l'électricité. Il a défini une nouvelle façon de consommation des ressources et des applications informatiques. Les applications, les ressources et la puissance de traitement sont proposées comme un service par un fournisseur et sont accessibles en ligne via un simple navigateur Web. Les logiciels et les données sont aussi hébergés dans des centres de données d'un prestataire ou dans les infrastructures internes d'une entreprise.

Les Clouds Computing peuvent être employés dans des modes différents : *privé* exécutés par ou pour une seule entreprise, *publique* fournissant un seul ou plusieurs niveau de service et stockant les données sur des ressources partagées, *hybride* qui est un mélange entre les deux modes précédents, Cloud de *communauté* pour les entreprises ayant les mêmes exigences de service et Cloud *à usage spécial*.

Les services proposés sur le Cloud Computing sont classés en trois types de services différents, IaaS (Infrastructure as a service), PaaS (platform as a service) et SaaS (Software as a service). Ces services sont disposés aux utilisateurs à la demande selon une tarification spécifie par le fournisseur du Cloud. La puissance de traitement est payée en fonction du taux d'utilisation des ressources matérielles (CPU par exemple). Le stockage et le transfert de données sont mesurés en fonction de la quantité de données stockées ou transférées. Pour la plupart des utilisateurs le Cloud Computing n'est pas très différent du Web, du fait qu'un certain nombre d'applications du Cloud ont été déjà proposées en ligne depuis longtemps. Cependant, il a apporté des caractéristiques spécifiques, telles que l'évolutivité, la flexibilité et la mise en pool des ressources, comme principaux différenciateurs du Cloud Computing.

Il existe d'autres concepts qui sont liés au Cloud Computing. L'objet de ce chapitre est de présenter la notion du Cloud Computing et les différents concepts liés à ce nouveau paradigme.

1.2 Définitions

Le terme "Cloud Computing" se traduit en français par "Informatique dans les nuages". Ce terme est une métaphore désignant des ressources informatiques accessibles sur Internet ou Intranet. De nombreuses définitions du Cloud Computing existent, dans cette section nous ferons un tour d'horizon en représentant les principales d'entre elles.

1.2.1 Clarification du terme Cloud

Depuis longtemps, avant la naissance du Cloud Computing, les concepteurs des réseaux utilisent le nuage (cloud en anglais) pour schématiser l'Internet dans les schémas des réseaux intra et inter-entreprise (voir la figure 1.1), du fait que l'Internet est un réseau mondial complex, constitué de millions de connexions et géré par des milliers de sociétés publiques et privées. Ainsi, pour la plupart des utilisateurs, l'Internet est un monde abstrait, considéré comme s'il est situé dans un espace sans réalité physique. Par conséquent, dans le monde informatique, le mot nuage fait référence à Internet.

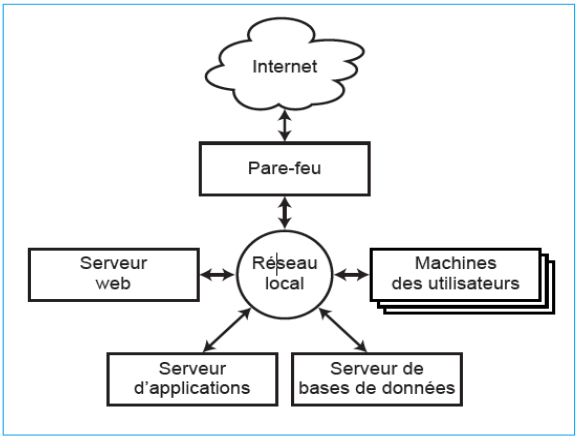


FIG. 1.1 – Nuage est l'image utilisée pour schématiser l'Internet

1.2.2 Cloud Computing (Informatique dans les nuages)

Andy Isherwood, VP en charge des services logiciels pour l'Europe, HP, a dit : "Beaucoup de gens sautent dans le train du Cloud, mais je n'ai pas entendu deux personnes dire la même chose sur ce sujet ... Il existe de multiples définitions de ce qu'est le Cloud." [8]). De nombreuses définitions du Cloud Computing ont été proposées, mais chacune de ces définitions se concentre uniquement sur certains aspects technologiques. Les chercheurs proposent des définitions selon leurs objectifs de recherche et les industriels personnalisent des définitions par leurs intérêts commerciaux. Dans ce qui suit, nous présentons quelques unes qui ont retenu notre attention :

Définition 1.2.1. Définition de l'université de Berkeley [8] : Le Cloud Computing désigne à la fois les applications qui sont livrées comme des services sur Internet, ainsi que le matériel et les logiciels dans les datacenters qui fournissent ces services. Les services eux-mêmes ont été longtemps appelés Software as a Service (SaaS), donc nous utilisons ce terme. Le matériel datacenter et le logiciel sont ce que nous appellerons un nuage.

Définition 1.2.2. Définition de Buyya et al. [17] : Un nuage est un type de système parallèle et distribué composé d'un ensemble d'ordinateurs interconnectés et virtualisés et qui sont provisionnés dynamiquement et présentés comme une ou plusieurs ressources informatiques unifiées, basées sur des accords de service établis par la négociation entre le prestataire et les consommateurs.

Définition 1.2.3. Définition des experts de l'Union Européenne [47] : Un «nuage» est un environnement d'exécution élastique de ressources impliquant de multiples intervenants et de fournir un service tarifé à granularités multiples pour un niveau de qualité (de service).

Définition 1.2.4. Définition de Vaquero et al. [73] : Les nuages sont un grand pool de facilement d'utilisation et des ressources virtualisées accessibles (telles que matérielles, plates-formes de développement et/ou services). Ces ressources peuvent être dynamiquement reconfigurées pour s'adapter à une charge variable (échelle), ce qui permet également une utilisation optimale de ces ressources. Ce pool de ressources est généralement exploité par un modèle paiement à l'utilisation, dans lequel les garanties sont offertes par le fournisseur d'infrastructure par le biais de SLA (Service-Level Agreements) personnalisées.

1.3 Concepts liés au Cloud

1.3.1 Nœud

Un nœud du Cloud peut être une machine ou une machine virtuelle. Il est caractérisé par une adresse propre. Il est constitué par un système de virtualisation (VMM = Virtual Machine Monitor), adapté à l'infrastructure matérielle et qui gère le cycle de vie de plusieurs images logicielles. Une application logicielle bas niveau « node » gère également les interactions entre le gestionnaire du Cloud et l'hyperviseur (Scheduler) pour le lancement et l'arrêt d'images.

1.3.2 Cluster

Le cluster (gape) est un ensemble d'ordinateurs interconnectés entre eux et travaillent en collaboration pour réaliser une fonction unique. Il existe deux approches différents afin de servir deux objectifs différents. Il y a des clusters de bascullement qui sont conçus pour fournir une haute disponibilité ou de solution de récupération rapide et il y a des clusters de performances pour offrir plus de performances.

Le Cloud Computing permet de construire un nuage de clusters, il permet la connexion entre un ensemble de machines sur un réseau bien défini. Par conséquent, les utilisateurs

peuvent déployer des machines virtuelles dans ce nuage, ce qui leur permet d'utiliser un certain nombre de ressources (Par exemple de l'espace disque, de la mémoire vive, ou encore du CPU).

1.3.3 Datacenter

Un centre de données est un lieu où sont installés les serveurs physiques informatiques. Leur superficie varie de quelques centaines de mètres carrées jusqu'à 65000 m^2 pour Google et Microsoft. Un fournisseur de Cloud doit disposer de plusieurs centres de données reliés entre eux. L'alimentation électrique, la connectivité, la sécurité et la surveillance sont des conditions nécessaires pour le fonctionnement de ces centres. Dans un centre de données, les nœuds sont organisés physiquement dans des racks et les nœuds du même rack sont interconnectés par un reseau Ethernet gigabit. Il peut y avoir de nombreux racks, dans ce cas les racks sont interconnectés à un autre niveau réseau par des Switch.

1.3.4 Virtualisation

La virtualisation n'est pas une idée nouvelle. En fait, elle remonte aux débuts de l'informatique. Elle consiste à faire executer plusieurs systèmes d'exploitation sur une seule machine comme s'ils fonctionnaient sur des machines distincts.La virtualisation fonctionne selon le principe suivant :

1. Un système d'exploitation principal (appelé système d'exploitation hôte) est installé dans l'ordinateur et sert de système d'accueil à d'autres systèmes d'exploitation.
2. Dans le système d'exploitation hôte, un logiciel de virtualisation (appelé hyperviseur) est installé. Celui-ci crée des environnements clos, isolés, avec des ressources bien précises : ces environnements clos sont appelés des machines virtuelles (VM).
3. D'autres systèmes d'exploitation (appelés systèmes d'exploitation invités) peuvent alors être installées dans des machines virtuelles. Leur instance est totalement isolée des autres systèmes hôtes et systèmes invités

Pour le Cloud, la plupart des architectures utilisées sont des datacenters et des serveurs avec plusieurs niveaux de virtualisation.

1.3.5 Pay-as-you-go

C'est la capacité de payer pour l'utilisation des ressources informatiques à court terme à la demande (par exemple, les processeurs à l'heure et le stockage de jour en jour) et les libérer au besoin.

1.4 Architecture du Cloud Computing

L'architecture d'un système Cloud Computing peut être divisé en trois couches [63] : (a)Ressources (b)Plate-forme et (c)Application. Ces trois couches sont illustrées par la figure 1.2. La couche des ressources est la couche d'infrastructure qui est composée de ressources physiques et virtuelisées de calcul, stockage et réseau. La couche de plate-forme est la partie la plus complexe qui pourrait être divisée en sous-couches nombreuses. Par exemple un framework informatique qui gère la répartition de transaction et/ou l'ordonnancement des tâches. La sous-couche de stockage offre un stockage illimité et la capacité de mise en cache. Le serveur d'application et d'autres composants fournissent une capacité à la demande et une gestion flexible.

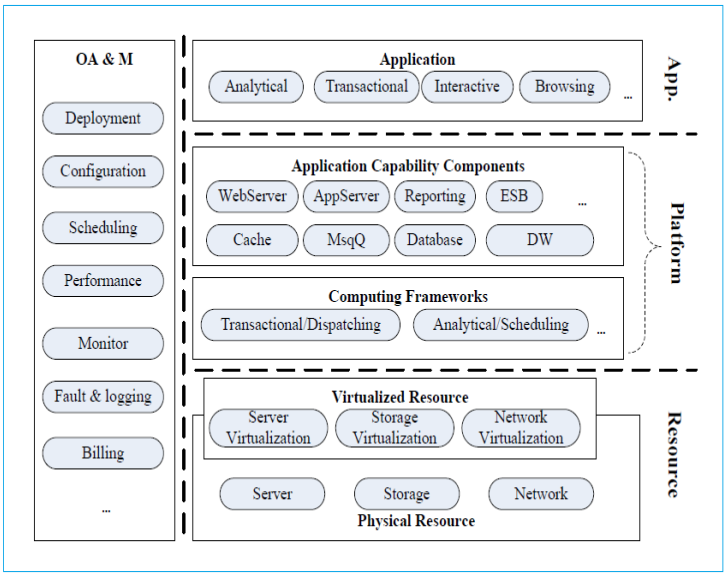


FIG. 1.2 – L'architecture de référence [63]

1.5 Caractéristiques

Les caractéristiques du Cloud Computing sont partitionnées en trois aspects : aspects non fonctionnels, aspects économiques et aspects technologiques [47] :

1.5.1 Aspects non fonctionnels

Les aspects non fonctionnels représentent des qualités ou des propriétés d'un système, plutôt que des exigences technologiques spécifiques.

- ◇ **Élasticité :** Elle définit la capacité d'une infrastructure donnée, d'ajuster les ressources à la hausse (expansion) ou à la basse (réduction), en fonction de la demande, de manière dynamique avec facilité et un minimum de frictions.

- ◇ **Fiabilité :** La fiabilité désigne la capacité d'assurer un fonctionnement constant sans interruption du système, c'est à dire sans perte de données et sans réinitialisation de code lors de l'exécution. La fiabilité est généralement réalisée grâce à l'utilisation des ressources redondantes.
- ◇ **QoS :** Les paramètres de base de QoS sont le temps de réponse, le débit, le nombre d'opérations par second, ETC. La fiabilité est un aspect particulier de QoS qui forme une exigence de qualité spécifique.
- ◇ **Agilité et adaptabilité :** Il inclut le temps de réaction aux changements dans la quantité de demandes et la taille des ressources, mais aussi l'adaptation aux changements dans des conditions environnementales qui nécessitent par exemple différents types de ressources. Implicitement, l'agilité et l'adaptabilité nécessitent que les ressources doivent être autonomes et permettent de fournir des capacités automatiquement.
- ◇ **Disponibilité de service :** La disponibilité d'une application désigne le ratio de temps pendant lequel elle est en état de fonctionnement correct sur une période de temps donnée. La disponibilité est mesurée par un pourcentage, 99% désigne le fait que le service est indisponible moins de 3,65 jours par an et 99,9 désigne une inactivité de 8,75 heures par an. La disponibilité est atteinte par la réplication de données/services et de les distribuer à travers différentes ressources pour réaliser l'équilibrage de la charge.

1.5.2 Aspects économiques

- ◇ **Réduction des coûts :** Le Cloud Computing s'inscrit dans un nouveau mode de consommation des ressources informatiques. Au lieu d'acheter ou de posséder des ressources informatiques, les entreprises tournent vers l'allocation de ces ressources en fonction de ses besoins. On peut dire que le Cloud computing permet de réduire de manière très important le coût total de possession et de maintenance
- ◇ **Payment par utilisation :** Les services et les ressources du Cloud sont exploités par le biais d'un nouveau modèle économique "pay per use" (paiement à l'utilisation). Ce modèle de paiement signifie que l'utilisateur paie uniquement le service en fonction de son taux d'utilisation.
- ◇ **Amélioration du temps sur le marché :** L'amélioration du temps sur le marché est essentielle en particulier pour les petites et moyennes entreprises (PME) qui veulent vendre leurs services rapidement et facilement avec peu de retard causé par l'acquisition et la mise en place de l'infrastructure informatique, en particulier dans un champ compatible et compétitif avec les grandes industries. Les grandes entreprises ont besoin d'être en mesure de publier de nouvelles fonctionnalités avec peu de surcharge pour rester compétitifs. Les nuages peuvent supporter celà en offrant des infrastructures, éventuellement dédiés à des cas d'utilisation spécifiques qui prennent plus de capacités essentielles pour soutenir l'approvisionnement facile et donc de réduire les délais de commercialisation.

- ◇ **Un retour sur l'investissement :** L'emploi d'un système de Cloud doit s'assurer que le coût et l'effort investis dedans est compensé par ses avantages à être commercialement persistant. Cela peut entraîner un retour sur l'investissement direct (par exemple, plus de clients) et indirect (par exemple bénéfice de la publicité).
- ◇ **Passer des coûts matérielles (CAPEX) aux coûts opérationnelles (OPEX) :** Les dépenses en capital (Capital expenditure : CAPEX) sont nécessaires pour construire une infrastructure locale, mais avec l'externalisation des ressources informatiques vers les systèmes de Cloud en fonction de la demande, une entreprise sera effectivement rétribuée les dépenses opérationnelles (OPEX) pour l'approvisionnement de ses capacités, car elle va acquérir et utiliser les ressources en fonction des besoins opérationnels.
- ◇ **Going Green :** Une grande partie de l'impact environnemental de l'informatique est liée à la consommation énergétique des parcs informatiques. L'allocation des ressources en fonction des besoins permet de réduire cette consommation ce qui permet de diminuer l'empreinte écologique de la société. En d'autre part, Les acteurs du Cloud commencent à utiliser des energies renouvelables dans leurs centres de données [62].

1.5.3 Aspects technologiques

- ◇ **Virtualisation :** La virtualisation est une caractéristique technologique essentielle du Cloud, qui cache la complexité technologique aux utilisateurs et permet une flexibilité accrue. La virtualisation peut rendre plus facile pour un utilisateur de développer de nouvelles applications et de réduire les frais généraux pour contrôler le système. Par l'exposition d'un environnement virtualisé, les services peuvent être accessibles indépendamment de l'emplacement physique des utilisateurs et des ressources.
- ◇ **Multi-allocation :** Dans un environnement Cloud, l'emplacement de code et/ou de données est inconnu et la même ressource peut être attribuée à plusieurs utilisateurs en même temps. Cela affecte les ressources de l'infrastructure ainsi que les données, les applications et les services qui sont hébergés sur des ressources partagées, mais doivent être mis à disposition dans plusieurs instances isolées. Multi-allocation peut impliquer beaucoup d'éventuels problèmes, tels que la protection de données.
- ◇ **Sécurité, confidentialité et conformité :** est évidemment essentiel dans tous les systèmes traitant des données et des codes potentiellement sensibles .
- ◇ **Gestion des données :** La gestion des données est un aspect essentiel en particulier pour les Clouds de stockage, où les données sont distribuées de manière flexible sur des multiples ressources. Implicitement, la cohérence des données doit être maintenue sur une large répartition des sources de données répliquées. Dans le même temps, le système doit toujours être conscient de l'emplacement des données prenant les latences et en particulier la charge de travail en considération.
- ◇ **Améliorations des API et/ou programmation** sont indispensables pour exploiter les caractéristiques des Clouds : des modèles de programmation commune exigent que

le développeur prend en considération de l'évolutivité et les capacités autonomes lui-même, tandis qu'un environnement de cloud fournit les fonctionnalités d'une manière qui permet à l'utilisateur de laisser une telle gestion au système.

- ◇ **Mesure** de tout type de consommation des ressources et de services est indispensable afin d'offrir des prix de chargement et de facturation élastiques. Il est donc une condition préalable à l'élasticité de nuages.

1.6 Mode de déploiement

Selon les approches des entreprises, les Clouds peuvent être hébergés et employés dans des modes différents. On peut distinguer les modes de déploiement suivants :

1.6.1 Cloud publique

Les entreprises peuvent externaliser ses ressources informatiques chez un fournisseur de Cloud afin de réduire les coûts et les efforts de construction de leur propre infrastructure, ou respectivement proposent leurs propres services aux autres utilisateurs en dehors de l'entreprise. On peut citer comme exemple Amazon Web Services et Microsoft Azure.

1.6.2 Cloud privé

Le terme "Cloud Privé" est utilisé pour désigner les datacenters internes d'une entreprise ou autre organisation qui ne sont pas mis à la disposition public. L'utilisation d'un cloud privé ou dédié répond à l'une des principales objections concernant le Cloud computing : le risque de vols de données et d'interruptions de service.

1.6.3 Cloud hybride

Bien que les Clouds publics permettent aux entreprises d'externaliser une partie de leur infrastructure à des fournisseurs de cloud, elles perdraient en même temps le contrôle des ressources et la gestion de code et de données. Dans certains cas, ce n'est pas souhaité par l'entreprise concernée. Les Clouds hybrides constitués d'un mélange de l'emploi des infrastructures de Cloud privé et public afin d'atteindre un maximum de réduction des coûts grâce à l'externalisation tout en conservant le degré souhaité de contrôle sur les données sensibles.

1.6.4 Cloud de communauté

L'infrastructure est partagé entre plusieurs organisations supportant une communauté précise et ayant des préoccupation communes.

1.6.5 Cloud à usage spécial

La spécialisation implique de fournir des fonctionnalités plus spécialisées pour des cas d'utilisations spécifiques. Par exemple, App Engine fournit des capacités spécifiques destinées à la gestion des documents distribués.

1.7 Modèles de Cloud

Le Cloud Computing se décompose généralement en trois types de services (voir la figure 1.3) : IaaS, PaaS et SaaS. Ces modèles de services sont définis comme suit :

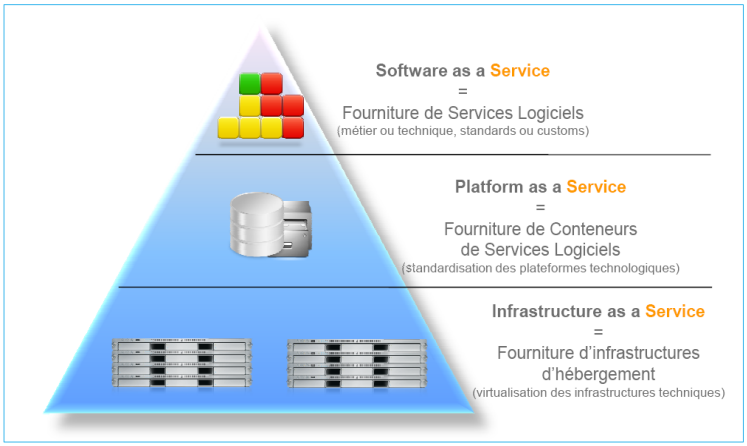


FIG. 1.3 – Trois types d'offre de services sur le Cloud

1.7.1 IaaS

Ce terme signifie "*Infrastructure as a Service*". Au lieu d'acheter des serveurs, des logiciels, des centres de données ou de l'équipement de réseau, les clients louent ces ressources comme un service. Dans ce modèle, l'entreprise utilisatrice maintient la gestion des applications, de l'architecture et du fonctionnement de son système d'information ainsi que les bases de données et les logiciels. Les principales composantes de ce modèle sont :

- ◇ Utilitaire de calcul de facturation ;
- ◇ Une plateforme de virtualisation de l'environnement ;
- ◇ Du matériel informatique ;
- ◇ Un réseau d'ordinateurs ;
- ◇ Une connexion Internet.

1.7.2 PaaS

Ce terme signifie "*Plate-forme as a Service*". Il désigne une plate-forme d'exécution hébergée par un opérateur et accédé depuis Internet. Par exemple une plate-forme PaaS peut être un environnement de développement et de test. Les principales offres de PaaS

sont : Amazon Web Services (AWS), Force.com, Google App Engine et Microsoft Azure Services Plateform. En outre que l'environnement d'exécution, la plate-forme PaaS propose les services suivantes [62] :

- ◇ Un service de persistence de données ;
- ◇ Un service d'hébergement d'application ;
- ◇ Un service d'intégration technique qui permet d'intégrer l'application hébergée avec d'autres applications ;
- ◇ Un service de surveillance ou monotoring qui permet d'assurer le suivi de la disponibilité des applications hébergées sur la plate-forme PaaS ;

1.7.3 SaaS

Ce terme signifie "*Software as a Service*". Il désigne un logiciel fourni sous forme de service et non sous forme de programme informatique. Dans ce modèle, le fournisseur vend des logiciels opérationnels prêt à l'emploi, sans passer par une étape d'installation et sans aucune tache de maintenance. Dans ce modèle, l'entreprise utilisatrice abandonne la totalité de la gestion des ressources informatiques.

1.8 Applications du Cloud

1.8.1 Principales applications du Cloud

De nombreuses applications du Cloud ont été déjà proposées en ligne depuis longtemps. Les principales applications que l'on trouve sur les Clouds sont les suivantes :

- ◇ La messagerie ;
- ◇ Les outils collaboratifs et de web-conferencing ;
- ◇ Les environnements de développement et de test ;
- ◇ Automatisation de la force de vente et la Business Intelligence ;
- ◇ La gestion de la relation client (CRM) ;
- ◇ Utilitaires bureautiques ;
- ◇ Archivage et sauvegarde de données ;
- ◇ Les applications d'ingénierie mathématique (modélisations 3D, simulations, CAO, ...) ;
- ◇ Les applications financières (analyse des marchés d'actions, analyses sur le long terme...) ;
- ◇ Comptabilité (gestion de trésorerie, de facturation, ...) ;
- ◇ Ressources humaines (gestion de recrutement, de la paie, ...).

1.8.2 Business Intelligence (BI) : Exemple d'une application de Cloud

La Business Intelligence (Intelligence d'affaire ou informatique décisionnelle en français) désigne les moyens, les outils et les méthodes qui permettent l'exploitation des données de l'entreprise pour aider les utilisateurs à prendre de bonnes décisions d'affaires [53]. La BI permet de regrouper les informations stratégiques et nécessaires à la construction des processus d'aide à la décision, qui permettent aux dirigeants de l'entreprise d'obtenir rapidement un portrait clair de la situation de l'entreprise, ceci crée un contexte idéal pour prendre les bonnes décisions au bon moment.

1.8.2.1 Entrepôt de données, ETL et OLAP

La BI est basée sur l'exploitation d'un système d'information décisionnel alimenté par l'extraction de données à partir des données de production de l'entreprise. Ces données, qui proviennent de plusieurs sources de données (bases de production de l'entreprise, fichiers textes, documents web, etc.), sont généralement stockées au sein d'une base de données particulière : l'entrepôt de données (Data warehouse). L'entrepôt est alimenté par des procédures d'extraction, de transfert et de consolidation (ETL).

L'entrepôt de données doit être accessible par de nombreux utilisateurs avec des outils analytiques complexes pour satisfaire leurs besoins en informations. OLAP (On-Line Analytical Processing), introduit en 1993 par E.F Codd [22], est l'ensemble des technologies qui permet aux utilisateurs finaux (des analystes et des décideurs) de traiter leurs données de façon analytique, interactive, rapide et permettant de voir les données de l'entreprise sous plusieurs dimensions. Dans l'architecture de BI (voir la figure 1.4), OLAP est une couche entre l'entrepôt de données et l'utilisateur final des outils de BI.

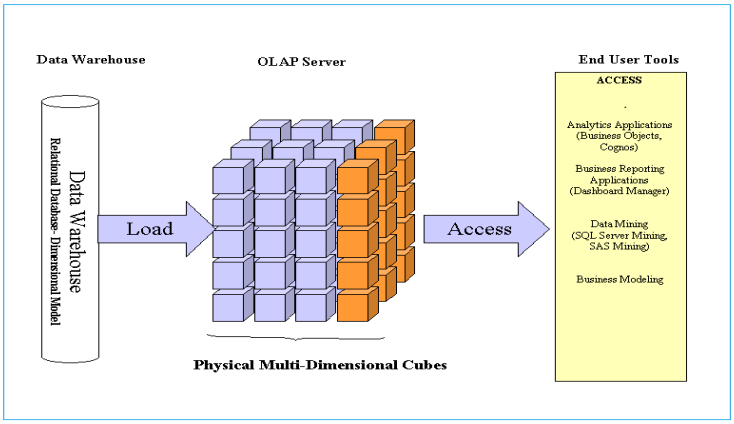


FIG. 1.4 – Architecture de BI

1.8.2.2 BI dans le Cloud Computing

Dans l'économie actuelle, chaque organisation tend à devenir une organisation intelligente et à obtenir un avantage de concurrence sur le marché par l'utilisation de nouveaux et des innovantes solutions de BI. Selon l'IDC [1] : "une étude préliminaire de la taille du marché a révélé que le marché des outils logiciels d'informatique décisionnelle a augmenté de 2,6 % en 2009, pour atteindre 8,1 milliards de dollars ... ce marché devrait connaître un taux de croissance annuel composé de 6,9% jusqu'en 2014, pour atteindre 11,3 milliards de dollars" [33].

L'intégration d'une solution de BI implique un grand investissement humain, matériel, logiciel et financier. Des gros investissements dans les solutions BI traditionnelles sont souvent peu pratiques et peu attrayants, tandis que les services BI à la demande ou Cloud BI sont de plus en plus populaires [51]. "Le modèle d'informatique décisionnelle basé sur un logiciel-service traite les points sensibles et promet de réduire les coûts et la durée nécessaire au déploiement, mais uniquement pour les entreprises dont les exigences sont adaptées à l'informatique en nuage", Wayne Eckerson, directeur de la recherche de TDWI (The Data Warehousing Institute) [33].

La BI dans le Cloud est un marché en croissance rapide et l'intégration des solutions Cloud BI a un intérêt spécial pour les organisations, du fait que le Cloud offre des avantages attirants en matière d'efficacité, de disponibilité, d'évolutivité flexible et de déploiement rapide. Concernant les risques du Cloud, la sécurité, la conformité et la qualité de service représentent les principaux risques.

1.9 Acteurs du Cloud

Le concept de Cloud Computing est comparable à celui de la distribution électrique, car les ressources informatiques sont proposées à la consommation par des compagnies spécialisées qu'on appelle "Acteurs".

1.9.1 Catégories des acteurs

Les acteurs du Cloud Computing sont nombreux, Plouin [62] a classifié ces acteurs en deux classes : des acteurs historiques et des acteurs issus du Web.

1.9.1.1 Acteurs historiques

Cette classe des acteurs englobe en premier lieu les éditeurs traditionnels de logiciels. Les plus connus sont : Microsoft, IBM, Adobe et Oracle. Ces acteurs tendent à devenir opérateurs SaaS et de trouver une place dans le monde des opérateurs SaaS.

[1] Un cabinet d'études de marché de premier ordre sur les marchés des technologies de l'information, des télécommunications et des technologies grand public.

Deuxièmement, on trouve HP, SUN et DELL qui sont des constructeurs les plus connus et qui ont entamé le domaine du Cloud. Généralement, ces acteurs proposent des PaaS afin d'héberger des SaaS.

Troisièmement, on trouve les opérateurs télécom. Les offres de services des opérateurs télécom sont basées sur des SaaS, par exemple le service d'agenda et de carnet d'adresses synchronisé avec le téléphone mobile. Enfin, on trouve les sociétés d'infogérance.

1.9.1.2 Acteurs issus du web

Google, Yahoo et Amazon sont les grands acteurs du Cloud issus du Web. Ces acteurs voient dans le modèle SaaS une continuité de leur modèle grand publique. Par exemple, Google a été démarré par un moteur de recherche gratuit et il est devenu aujourd'huit un acteur totalement incontournable du cloud et dispose du plus grand parc de datacenters connus [62]. Les offres de services de Google couvrent des services de recherche, de cartographie, services collaboratifs Google Apps, etc. et le service PaaS google App Engine.

1.9.2 Principaux acteurs & leurs offres de services

1.9.2.1 Amazon

Amazon a été la première société à proposer une plate-forme de Cloud Computing avec Amazon Web Services (AWS). AWS été la premiére offre PaaS disponible en 2002 et il s'agit de déporter des traitements dans les nuages.

Le site de commerce électronique Amazon.com a dimensionné ces ressources informatiques (centres de données) pour supporter la charge pendant la période des fêtes de Nöel, mais pendant le reste de l'année ces ressources sont restées inutilisées. Pour les rentabiliser, le commerçant a décidé de louer une partie de ses capacités à des entreprises intéressées en fonction de la durée d'utilisation.

L'offre AWS est accessible sur aws.amazon.com. Elle se décompose en plusieurs couches de services interagissent les uns avec les autres [74], telles que :

- ◇ **Amazon EC2 (Amazon Elastique Cloud Computing)** Ce service met en disposition une capacité de traitement ajustable.
- ◇ **Amazon S3 (Amazon Simple Storage Service)** est un système distribué de stockage de données.
- ◇ **Amazon elastic MapReduce** permet de créer des traitements sous forme de job flows.
- ◇ **SQS (Simple Queue Service)** est une file de messages distribuée permet de stocker les messages en transit entre les ordinateurs et les composants applicatifs.

Amazon laisse aux clients le choix d'utilisation de modèle de programmation, le langage ou le système d'exploitation (Windows Server 2003, Linux). Ainsi que le choix d'utiliser un seul service ou utiliser un sous ensemble de services.

Les services d'Amazon sont accédés par des compagnies différentes, citons comme exemple New York Times avec son projet « TimesMachine » [60]. Ce journal a fait appel aux services d'Amazon pour mettre en ligne les archives du journal de 1922 à 1951, car la conversion des documents dans un format utilisable nécessitant une puissance de calcul de loin supérieure à celle que le journal avait imaginée.

1.9.2.2 Google

Google fournit une plate-forme Cloud intitulé Google App Engine. Google App Engine est une plate-forme qui fournit une grande puissance de traitement et une grande capacité de stockage, destinée pour le développement et l'hébergement des applications Web. Les services App Engine sont les suivantes :

- URL Fetch (service de rappatriement d'URL) ;
- Messagerie ;
- Memcache ;
- Manipulation d'Images ;
- Tâches Planifiées ;
- Stockage distribué de données (comprend un moteur de recherche et la gestion des transactions).

1.9.2.3 Microsoft

La plate-forme Cloud de Microsoft est intitulée Microsoft Azure. Elle s'agit d'une offre d'hébergement des applications et des données et de services de stockage, synchronisation des données, bus de messages, contacts, etc. La plate-forme Windows Azure est composée des éléments suivants :

- **Windows Azure** est le socle de la plate-forme cloud computing de Microsoft qui permet de développer, héberger et administrer des services.
- **SQL Azure** est un service de stockage relationnel dans le Cloud.
- **Windows Azure AppFabric** simplifie l'intégration entre les serveurs déployés en entreprise et le Cloud.
- **Windows Azure Market Place** est une place de marché en ligne pour les professionnels de l'informatique permettant de partager, acheter et vendre des composants et des données, pour construire des applications ou des services tirant profit de la plate-forme Windows Azure.

1.9.2.4 IBM

IBM a construit Bluehouse un ensemble de centres de données destinés à proposer un certain nombre de ses applications en mode SaaS. C'est un logiciel qui permet aux équipes de collaborer en ligne et faire organiser leurs travails. On trouve dans Bluehouse les fonctionnalités suivantes :

- **Meeting :** un outil de Web conférence ;
- **Store & share :** outil de partage d'information ;
- **Activities :** outil de partage des taches ;
- **Forms & LiveCharts :** outil de création des formulaires et de graphiques.

IBM SmartCloud Managed Backup est un service de stockage et de reprise pour la protection des données fournit par IBM. Ce service offre une infrastructure logicielle et matérielle de sauvegarde des données.

1.9.2.5 Salesforce

Salesforce fut le premier hébergeur de Cloud en 1999. La principale application proposée par salesforce est la CRM. Il offre aussi des fonctionnalités de marketing, analyse, gestion des ressources humains (GRH), gestion de vente, etc.

Salesforce propose aussi une plate-forme pour le développement et le déploiement des applications de gestion. Ainsi, force.com permet de créer des application de gestion à la demande.

1.9.2.6 Sun

Sun Microsystem a annoncé en 2009 un service Cloud public ouvert aux développeurs, aux étudiants et à tous ceux qui ne veulent pas dépenser beaucoup d'argent pour se doter d'une infrastructure serveur. Sun Open Cloud est composé de deux services : le Sun Cloud Storage Service et le Sun cloud Compute Service. Les services offerts sont assez similaires à ceux d'Amazon Web Service.

1.10 Modèle de paiement

Le Cloud Computing s'appuie sur un nouveau modèle économique de paiement à l'utilisation, qu'on appelle "pay-as-you-go".

1.10.1 Composants d'une tarification cloud computing

Les composants de base d'une tarification Cloud sont les suivants :

- ◇ **Le temps de traitement par heure :** C'est le temps d'utilisation de CPU par heure.
- ◇ **Le stockage :** Le stockage est facturé généralement par Géga octet (GO) et par mois et avec des prix dégressifs en fonction du volume.
- ◇ **Les transactions :** Généralement facturées par le nombre de requêtes réalisées, avec séparation des transactions entrantes et sortantes.
- ◇ **La bande passante :** Les transferts de données sont facturés en fonction de la quantité de données transférées vers et depuis la plate-forme, parfois avec des tarifs dégressifs en fonction du volume de données transférées et des tarifs négligés pour la bande entrante.

1.10.2 Types de facturation

Deux types de facturation sont disponibles, facturation par ressources utilisées et facturation par nombre d'utilisateurs [60].

1.10.2.1 Facturation par ressources utilisées

La plupart des fournisseurs proposent des facturations basées sur les ressources utilisées sur une machine virtuelle. De plus, la plupart demande une tarification supplémentaire sur le débit consommé (entrante et sortante).

1.10.2.2 Facturation par nombre d'utilisateurs

Certains fournisseurs de Cloud fournissent une facturation par nombre d'utilisateurs. L'inconvénient de cette méthode est qu'elle ne prend pas en compte les ressources utilisées. En effet, un client mettant en place plusieurs dizaines, voir plusieurs centaines d'applications, payera le même prix au mois qu'un client ne disposant que des services de base et ne demandant quasiment pas de ressources.

1.10.3 Tarifications cloud computing

Nous présentons dans ce qui suit des tarifs fournis par certains fournisseurs de Cloud. Ces prix sont valides à la date du 15 septembre 2012 :

1.10.3.1 Modèle tarifaire de Amazon

Le tableau 1.1 présente les différents prix d'utilisation de AWS (S3[2] et EC2 [3]). Ces tarifs sont variés selon la région et dégressifs en fonction de la quantité de données stockées/transférées ou en fonction du type d'instances utilisées [4]. Il existe également des tarifs gratuits pour les nouveaux clients. Par exemple, les nouveaux clients reçoivent 5 Go de stockage Amazon S3, 20 000 requêtes Get, 2000 requêtes Put, 750 heures d'utilisation d'instance EC2 (sous Linux et Windows) et 15 Go de transfert de données sortantes chaque mois pendant un an.

1.10.3.2 Modèle tarifaire de Google AppEngine

Le tableau 1.2 présente les tarifs appliqués aux ressources informatiques disposées par Google App Engine [5]. Chaque 24 heures, les applications App Engine peuvent consommer gratuitement une certaine quantité de ressources informatiques. Si un client veut dépasser les quotas gratuits, il peut activer la fonction de la facturation de leurs applications pour acheter des ressources suplémentaires selon ses besoins.

[2] `http://aws.amazon.com/fr/s3/pricing/`
[3] `http://aws.amazon.com/fr/ec2/pricing/`
[4] Les tarifs présentés corespondent aux instances par défaut situées aux USA (Virginia)
[5] `http://cloud.google.com/pricing/`

Consommation	**Amazon S3**	**Amazon EC2**
Stockage redondant (GO/mois)	0.125 $	0.10 $
Stockage redondant limité (GO/mois)	0.093 $	-
Bande passante sortante (GO)	0.12 $	0.120 $
Bande passante entrante (GO)	0.00 $	0.00 $
Coût d'exécution/heure (Windows)	-	0.115 $
Coût d'exécution/heure (Linux/UNIX)	-	0.080 $

TAB. 1.1 – Tableau des tarifs Amazon AWS (S3 et EC2)

Consommations	**Tarif (unité)**	**Quotas gratuits par jour**
Stockage	0.24 $ (GO/mois)	1GO
Bande passante sortante	0.12 $ (GO)	1 GO
Bande passante entrante	0.00 $ (GO)	1 GO
Temps processeur	0.08 $/H	28 H
Recipients Emailed	0.01 $/100 destinataire	100 destinataires

TAB. 1.2 – Tableau des tarifs Google App Engine

Google App Engine permet aux clients de déterminer les ressources à acheter chaque jour et de contrôler le montant de la facture quotidienne. Pour le faire, le client doit saisir le budget quotidien maximum (Max Daily Budget) en dollar. Il s'agit du montant maximum que dispose le client à payer sur une période de 24H et seulement la consommation réelle qui sera facturée.

1.10.3.3 Modèle tarifaire de IBM

Le tableau 1.3 présente les tarifs appliqués par IBM sur l'utilisation d'une machine virtuelle 32 bit avec des prix par heure non réservés [6].

Consommations	**Tarif (unité)**
Stockage (GB/H)	0.00015 $
Traitement par heure	
Redhat Linux	de 0.125 $ à 0.360 $
Nouvelle SUSE Linux	de 0.095 $ à 0.330 $
Windows Server (2003 et 2008)	de 0.100 $ à 0.370 $
Bande passante (In et Out)	0.15 $ (10 TO/ mois)
Transaction	0.11 $ (pour un million de requêtes par bloc)

TAB. 1.3 – Tableau des tarifs IBM Smart Cloud

IBM fournit des tarifs par heure réservés ou non réservés, en outre que des prix mensuels réservés. Les coûts de transfert sont dégressifs en fonction de la quantité de données à transférer. Cependant, les tarifs de traitement sont variés selon un ensemble de critères tels que la configuration de la machine virtuelle (32 bit ou 64 bit).

[6] `http://www-935.ibm.com/services/us/en/cloud-enterprise/tab-pricing-licensing.html`

1.10.3.4 Modèle tarifaire de Microsoft Azure

Le tableau 1.4 présente les tarifs appliqués par Microsoft Azure [7]. Microsoft Azure dispose aussi d'une offre gratuite pour le test de la plate-forme pour une durée de trois mois et une offre pour une période de 30 jours inclus Windows Server et SQL Server.

Le modèle de paiement à l'usage de Microsoft Azure se diffère en fonction de la zone des applications et le type d'instance. Les prix de stockage sont mesurés en GO sur une période mensuelle et aussi en terme de transaction de stockage et la bande passante est quantifiée en fonction de la quantité de données transférées. Ainsi, ces prix sont dégressifs en fonction de la quantité de données stockées ou transférées. Les prix de traitement sont évalués en heure CPU et ils augmentent par petits incréments en fonction de la taille des instances d'exécution.

Consommation	**Tarifs** (€)
Calculs	0.17
Base de données (GO/mois)	7.085
Stockage (GO/mois)	0.0887
Stockage (par 100000 transactions de stockage)	0.0071
Transfert de données (sortante /GO)	0.0852
Taille de cache (128MB)	45.00

TAB. 1.4 – Tableau des tarifs Microsoft Azure

1.10.3.5 Modèle tarifaire de Salesforce

Le tableau 1.5 présente les différents prix d'utilisation des offres de Salesforce [8]. Ce fournisseur fournit une facturation par nombre d'utilisateurs avec des essais gratuits pendant une période précise. Par exemple 07 jours d'essai pour la gestion des contacts et 14 jours d'essai pour la gestion des ventes et marketing.

Les offres	**Tarifs** ($/usr/mois)
Gestion des contacts (5 usr)	5
Ventes& marketing (5 usr)	15
CRM complet ed. Professionnel	65
CRM personnalisé ed. entreprise	125
CRM illimité	250
Plate-forme d'application (1 app)	15

TAB. 1.5 – Tableau des tarifs de Salesforce

[7] `http://www.windowsazure.com/fr-fr/pricing/details/`
[8] `http://www.salesforce.com/crm/editions-pricing.jsp`

1.11 Avantages & inconvénients

Le Cloud Computing peut représenter des avantages pour certains et des inconvénients et des limites pour d'autres. Nous présentons dans ce qui suit des listes non exhaustives des avantages et des inconvénients & risques de Cloud Computing.

1.11.1 Avantages

- ◇ **Simplicité d'utilisation et de gestion :** Le Cloud Computing permet de faciliter l'élaboration, la gestion des coûts et le suivi financier des opérations informatiques de l'entreprise. Le fournisseur du Cloud est chargé de la gestion technique, financière et opérationnelle de l'infrastructure et des licences logicielles.
- ◇ **Réduction des coûts liés aux infrastructures** tels que le coût de maintenance, le coût d'exploitation et le coût de déploiement et d'installation (pas de serveur ni de logiciel à installer, pas de réseau à étendre, pas de formation exploitant à acquérir), du fait que l'acquisition de matériels et logiciels est remplacée par la location de services de traitement.
- ◇ **Recentrage sur le métier** : L'entreprise n'étant plus responsable de son infrastructure informatique, ceci implique un gain de budget et de temps qui servaient à la maintenance des serveurs et la mise à jour du parc. L'entreprise peut donc mettre ce gain d'argent et de temps dans son corps de métier et de concentrer sur les problématiques de métier et non plus sur l'infrastructure.
- ◇ **Evolutivité illimitée à la demande et allocation dynamique de capacité** (permettant en particulier de s'adapter aux pics de charge)
- ◇ **Réduction des délais de déploiement** (économies de temps dans les phases de paramétrage (pré-packagé)). Le Cloud Computing permet d'accélérer et de faciliter l'installation d'un environnement informatique. En effet, la société proposant une offre Cloud dispose déjà des serveurs dont a besoin l'entreprise cliente. L'entreprise ne se soucie donc plus de l'installation d'une infrastructure informatique.
- ◇ **Gagne de l'espace et de l'économie d'énergie et la réduction de la sous-utilisation des ressources informatiques :** En effet, d'après plusieurs études un serveur d'entreprise en mode classique est utilisé en moyenne pour 10% du temps mais reste branché en permanence, consommant une grande quantité d'électricité inutile et occupant de l'espace. En mode Cloud, le même service applicatif est rendu sur des ressources partagées et les temps de « sommeil » de l'application permettent l'utilisation des ressources par d'autres applications.
- ◇ **Mobilité** : Grâce aux nouveaux progrès du matériel informatique (Ordinateurs, Smartphones, Laptop, PDA, ...), les utilisateurs peuvent avoir accès aux données, applications, serveurs, informations ou plate-formes indépendament du terminal, de n'importe où et à tout moment.

- ◇ **Souplesse du plan de reprise d'activité :** Les acteurs du Cloud disposent généralement de plusieurs datacenters distants ce qui leur permet de mettre en place des plans de reprise d'activités de grandes qualité.
- ◇ **Adaptabilité** du datacenter pour l'accès, l'organisation et la volumétrie des données.

1.11.2 Inconvénients & risques de Cloud computing

Le passage au Cloud Computing pour les entreprises représente un ensemble de risques et des inconvénients. La liste suivante présente les majeurs inconvénients et risques du Cloud Computing. Ces inconvénients sont des obstacles de succès du Cloud Computing :

- ◇ **Disponibilité et performance de services :** Le Cloud est situé quelque part dans le monde, dans un ou des centres de données où les défaillances, les ruptures et les incidents ne sont pas écartés. L'arrêt électrique, coupure malveillant des fibres optiques, arrêt de refroidissement dans un centre de données informatique, crash des disques, sont des exemple des incidents ou des défaillances qui peuvent impliquer l'inaccessibilité aux applications et l'indisponibilité de services. Cela peut conduire à des impacts négatifs pour les entreprise utilisatrices.
- ◇ **Sécurité des données :** La sécurité est la préoccupation principale dans le cadre d'un service Cloud. Dans le cas du Cloud Computing, l'entreprise devra connecter ses postes à Internet, d'une manière direct ou indirect, et les exposer à un risque d'attaque et de violation de confidentialité. Ainsi, les entreprises utilisatrices peuvent considérer divers typologies de risques, tels que : le risque de vol de secret industriel, risque de vol de données confidentielles sur ses clients (le cas d'une banque par exemple) ou des données sensible, risque de vol de données de fonctionnement interne, etc.
- ◇ **Variation de la conformité réglementaire :** Les réglementations sont variées d'un pays à un autre et d'un continent à un autre. L'hébergement des applications Cloud dans d'autres pays ou d'autres continents peut avoir des conséquences négatives sur l'entreprise utilisatrice, car la loi qui s'applique est celle du pays où se trouve les serveurs. Par exemple, un datacenter hébergé en Chine subit la réglementation chinoise où l'autorité centrale peut demander à consulter les données stockées sur les serveurs.
- ◇ **Dépendance au réseau :** Le recours au Cloud computing suppose que la connexion de l'entreprise au reseau Internet est de haute qualité. La rupture de la connexion Internet implique la perte d'accès aux applications et aux données.
- ◇ **Augmentation du trafic réseau :** Par exemple pour envoyer un message à un collègue via Google Apps, on le fait transiter deux fois par Internet, une fois pour le transmettre à la plate-forme Google et une fois pour le collecter de la plateforme Google.
- ◇ **Perte de contrôle et de ressources :** Dans le cas de Cloud Computing, l'accès aux données et aux applications est réalisé en utilisant un simple poste de travail équipé par simple navigateur Web. Donc, le poste de travail tend à devenir une coquille vide de données. Cela peut donner à l'utilisateur une impression de dépossession de données.

D'autre part, l'entreprise pert la totalité de la gestion des ressources informatiques et le contrôle de l'implémentation de leurs données, ainsi que le cycle de vie des applications.

1.12 Conclusion

La naissance du Cloud Computing a donné un nouveau mode de consommation de l'informatique et a changé la manière d'investissement des entreprises dans les infrastructures informatiques. Un coût faible, stockage évolutif illimité et une grande puissance de calcul sont les promesses du Cloud Computing. Tandis que la sécurité et la disponibilité représentent les enjeux et les freins majeurs de ce nouveau paradigme.

Dans ce chapitre, nous avons essayé de faire un survol sur le Cloud Computing et ses concepts. Dans un premier lieu, nous avons présenté une clarification du terme "Cloud" et quelques définitions des experts de ce paradigme. Puis, nous avons abordé quelques concepts liés à ce nouveau paradigme et ses différentes caractéristiques. Ensuite, les types de déploiement et les modèles de services sont présentés. Enfin, pour montrer mieux ce paradigme, nous avons exposé ses acteurs, ses applications, les tarifications et ses points forts et points faibles. Sur le Cloud on trouve de nombreux types d'applications et nous avons présenté comme exemple la BI. Cette application nécessite l'analyse de grande quantité de données qui sont généralement stockées dans un entrepôt de données.

Dans le chapitre suivant, nous concentrerons sur la présentation de la gestion de données volumineuses sur le Cloud, qui est un aspect essentiel en particulier pour les Clouds de stockage, ainsi que les nouveaux systèmes de fichiers distribués, les nouveaux paradigmes d'exécution et les languages d'accès aux données de ce nouvel environnement.

Chapitre 2

Gestion de données volumineuses sur le Cloud

2.1 Introduction

Depuis quelques années, la plate-forme Cloud Computing a reçu beaucoup d'attentions comme une nouvelle tendance de la gestion des données. Implicitement, les ressources dans les Clouds peuvent accueillir et/ou traiter d'énormes quantités de données. Les grands acteurs du Cloud disposent de plusieurs datacenters dispersés dans le monde avec hébergement des dizaines de milliers de nœuds qui permettent de fournir un stockage illimité et un calcul massivement parallèle et distribué.

La gestion de données sur cette nouvelle plate-forme est différente de celle d'un environnement traditionnel. Les outils traditionnels de gestion des données sont insuffisants pour le traitement et la gestion de grandes quantités de données dont la taille variée de quelques centaines de téraoctets à des dizaines de pétaoctets. Par conséquent, l'amélioration de la gestion et de la structuration de données est nécessaire pour faire face aux besoins de stockage et de traitement évolutifs.

Actuellement, la plupart des infrastructures Cloud sont basées sur les systèmes de fichiers distribués (DFS) dont ses caractéristiques principales sont la réplication et la distribution de données. Les principaux systèmes de fichiers et de stockage distribués de l'environnement Cloud sont GFS (Google file system) [32], HDFS (Hadoop distributed file system) [76], S3 (Simple Storage Service) [68] et Dynamo [27]. Ces systèmes présentent le niveau de stockage de données. Le niveau d'exécution se comporte de nouveaux modèles et paradigmes d'exécution dont le plus populaire est le modèle de programmation MapReduce [25] de Google et son implémentation open source Hadoop [37].

L'apparition des nouveaux paradigmes d'exécution et de traitement de grosses quantités de données sur le Cloud, nécessite le développement des nouveaux langages d'interrogations de données. Ces langages sont conçus pour cacher les détailles de l'environnement d'exécution et pour simplifier la complexité des paradigmes d'exécution. Parmi les langages du Cloud les plus connus, on peut citer Hive [72], Pig [28] et Dryadlinq [79].

L'objet du présent chapitre est de présenter la notion de gestion de données sur le Cloud et d'exposer les principaux systèmes de fichier distribués de ce dernier et les nouveaux modèles

de programmation et les moteurs d'exécution. Nous présentons aussi les nouveaux langages de programmation et d'interrogation du Cloud Computing.

2.2 Organisation et traitement de données sur le Cloud

La caractéristique typique de la plate-forme Cloud est qu'elle permet de traiter les taches et la charge de travail en parallèle sur un grand nombre de machines. Ces machines sont classées en deux types [80] : nœuds maitres et nœuds esclaves (voir la figure 2.1). Les nœuds maitres sont chargés de stocker les métadonnées sur l'ensemble du système en outre que les données régulières. Les nœuds esclaves stockent les enregistrements de données et leurs réplications dans le but de garantir la disponibilité de données et un haut degré de sécurité. Les données sur le Cloud sont stockées sous forme de fichiers gérés par des systèmes de gestion de fichier distribué (DFS), par exemple : GFS (Google file system) et HDFS (Hadoop distributed file system). Ces systèmes utilisent généralement le modèle de stockage clé-valeur pour organiser les données. Par conséquent, les systèmes du Cloud, tels que HDFS et Hadoop, ne supportent que la recherche par mot clé.

Les requêtes des clients sont souvent envoyées aux nœuds maitres [80] qui décident quels sont les nœuds esclaves pertinents pour le traitement des requêtes considérées. Ensuite les clients seront communiqués directement avec ces nœuds. Le traitement des requêtes peut être divisé en deux phases : la phase de localisation des nœuds pertinents pour les requêtes et la phase de traitement des requêtes où les données correspondantes sont extraites du DFS en conformité avec les mots clés contenus.

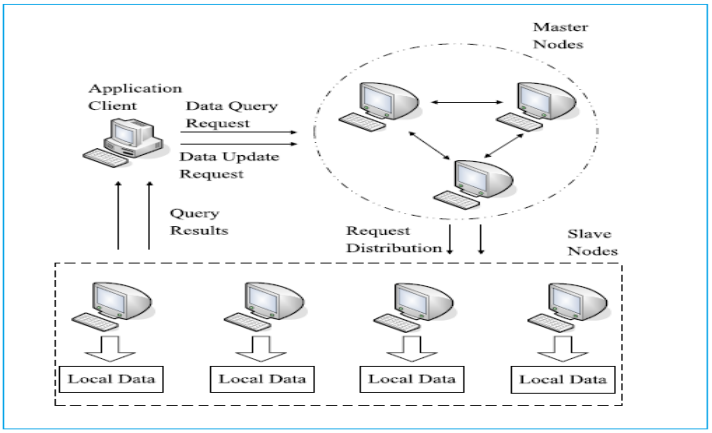


FIG. 2.1 – Classes des machines du Cloud et traitement des requêtes [80]

2.3 Gestion des données transactionnelles

Les applications de gestion des transactions reposent sur la propriété ACID (Atomicity, Consistency, Isolation, Durability)[1] fournit par les bases de données. Cette propriété est difficile à réaliser sur le stockage des données répliquées et distribuées. Les applications de gestion des transactions, tels que les services bancaires, le commerce électronique et la réservation des compagnies aériennes, ne sont pas adaptées pour le déploiement sur le Cloud pour les raisons suivantes [1] :

- ◇ Les systèmes de gestion des données transactionnelles n'ont pas l'habitude d'utiliser une architecture shared-nothing ;
- ◇ Il est difficile de garantir la propriété ACID sur des données répliquées sur de grandes distances géographiques ;
- ◇ Il y a des risques énormes dans le stockage de données transactionnelles sur un hôte non sûr (des trous de sécurité ou les violations de confidentialité sont inacceptable).

2.4 Gestion des données analytiques

Les applications de gestion de données analytiques sont utilisées pour résoudre les problèmes de planification des entreprises et pour l'aide à la décision. Les données impliquées par la gestion de données analytiques sont souvent des données historiques, de grande taille, rarement modifiées et nécessitent seulement l'opération de lecture. Les systèmes de gestion de données analytiques sont bien adaptés pour fonctionner dans un environnement Cloud [1]. Ces applications seront parmi les premières applications de gestion de données à être déployées dans le Cloud pour les raisons suivantes [1] :

- ◇ L'architecture Shared-nothing est bien adaptée pour la gestion des données analytiques (les charges de travail d'analyse des données, qui se composent de nombreuses agrégations multidimensionnelles et de la jointure en étoile, sont assez faciles à paralléliser sur les nœuds dans un réseau shared-nothing) ;
- ◇ Les propriétés ACID généralement ne sont pas nécessaires ;
- ◇ Les données sensibles peuvent être souvent exclues de l'analyse.

2.5 Systèmes de fichiers distribués

Les données sur le Cloud sont de volume très important. Ces données sont gérées par des systèmes de fichier distribués dont ses caractéristiques principales sont la réplication, la distribution de données et la taille des blocks peut être 1000 fois plus grande que celle dans les systèmes de fichier classiques. Les données sont automatiquement répliquées sans intervention des utilisateurs. La disponibilité et la durabilité sont atteintes grâce à la réplication. Dans cette section, nous présentons les principaux DFS du Cloud.

[1]Des propriétés sur lesquelles étaient basées les SGBD traditionnels.

2.5.1 Google File Système (GFS)

GSF [32] est un système de fichier distribué développé et implémenté par Google depuis 2001. GFS est destiné pour le stockage des données très volumineuses et la gestion d'énormes fichiers répartis et redondants.

Les fichiers sous GFS sont de taille très importantes voir plus d'un Gegaoctet. Chaque fichier est divisé en un ensemble de blocs de 64 MB qu'on appelle *Chunk*. Chaque *Chunk* a un identifiant unique appelé *Chunk Handle*, lui attribué au moment de leur création. Les *Chunks* sont répliqués trois fois dans des endroits différents, dans le but d'éviter la perte de données en cas de défaillance des machines. Les opérations supportées par GFS sont les opérations classiques d'un système de fichier.

Un cluster GFS est constitué d'un ensemble de machines standards exécutant le système d'exploitation Linux. Chaque cluster est composé d'un seul *Master* et plusieurs *serverchunks* (voir la figure 2.2). Le *Master* est chargé de gérer toutes les métadonnées de tous les fichiers

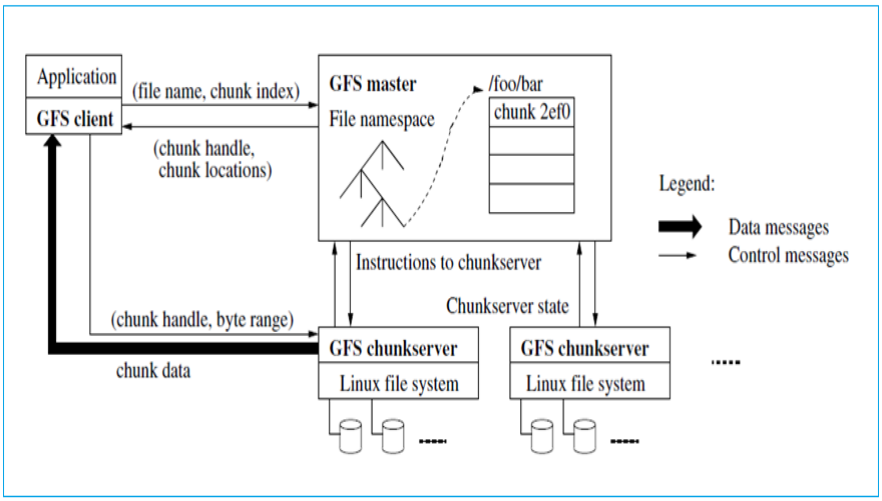


FIG. 2.2 – Architecture de GFS [32]

du système : l'espace des noms (arborescence des répertoires), les informations du contrôle d'accès, la correspondance entre le fichiers et ses *chunks* et les emplacements des répliques de chaque *Chunk*.

Le *Master* communique régulièrement avec les *Chunkservers* par des messages "*Heart-Beat*" pour lui donner les instructions (créer un nouveau *Chunk* ou supprimer un *chunk*) et pour collecter leurs états (quels sont les répliques qui sont stockées dans un Chunkserver, les *Chunkservers* en panne, les disques en panne).

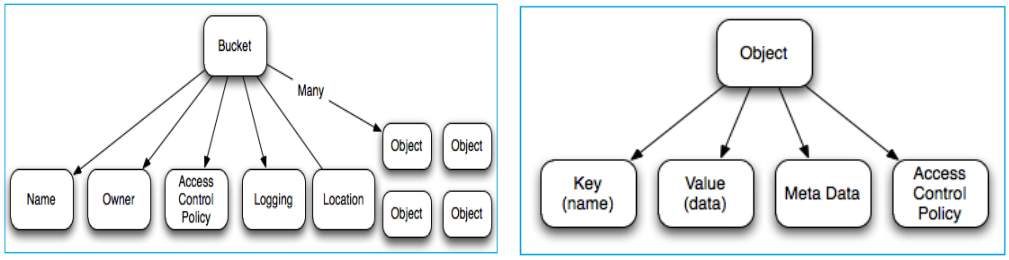
Un cluster est simultanément accessible par des machines clients. Un client GFS interagit avec le *Master* et les *Chunkservers* pour lire ou écrire des données. Il récupère les métadonnées de *Master* et les données des *chunkservers*. GFS fournit une haute tolérance aux pannes utilisant deux stratégies simples : récupération rapide et la réplication. Le *Master* et les *chunkservers* sont conçus pour redémarrer et restorer leurs états dans quelques secondes. La

réplication des *Chunks* sur des machines différentes permet la récupération de données en cas des problèmes. Si le *Master* tombe en panne, il sera remplacé par une autre machine sans perte de données, grace au fichier log et les points de contrôle qui sont dupliqués sur d'autres machines. De plus, Chaque *Chunk* a une donnée de contrôle (*checksum*) associée, qui permet de détecter la corruption des données.

2.5.2 Amazon S3

Amazon simple storage service [68] est un utilitaire de stockage, permet de stocker des données sur le Cloud à faible coût. S3 peut gérer des données de grosse quantité et un grand nombre d'accès concurrents. La notion de sous dossiers n'est pas utilisée sous S3. Les données sont stockées de manières redondantes sur un espace de nommage à deux niveaux dont le niveau supérieur est les Buckets et le niveau inférieur sont les objets.

Les Buckets servent à grouper un nombre illimité des objets de données [67]. Chaque Bucket a un nom, globalement unique, un propriétaire, une politique d'accès (désigne les permissions d'accès à un Bucket), information de Log, un emplacement et de nombreux objets (voir la figure 2.3(a)).



(a) Buckets en S3 (b) Objets de S3

FIG. 2.3 – Espace de nommage de S3 [58]

Un objet est un conteneur de données, il est stocké sous forme d'une séquence de bits [58]. Chaque objet doit appartenir à un Bucket. Ainsi, chaque objet a une clé, un propriétaire, la valeur, des métadonnées et une polytique de controle d'accès (désigne les permissions d'accès à un objet) (voir la figure 2.3(b)). La clé d'un objet est son nom qu'il ne peut pas être changé après sa création et il doit être unique dans un Bucket. La valeur est les données que contient un objet dont la taille est limitée par 5GB. Ainsi, chaque objet peut avoir deux types de métadonnées : (a) des métadonnées du système (tels que la date de la dernière modification, la taille de l'objet en octet) (b) des métadonnées utilisateurs qui sont des données ajoutées par l'utilisateur.

2.5.3 Dynamo

Dynamo [27] est un système de stockage distribué, contruit pour la plate-forme Amazon. Il est utilisé par certains services de base d'Amazon, qui nécessitent une haute fiabilité, disponibilité et durabilité de données. Ces propriétés sont atteintes par la duplication et le partitionnement de données.

Dynamo utilise le modèle de stockage clé-valeur. Les données sont stockées sous forme d'objets binaire identifiés par des clés uniques. Ces objets sont de petite taille, qui est généralement moins de 1 MB. L'accès aux objets est réalisé par deux opérations de base : la lecture (get) et l'écriture (put) des données. Chaque opération est appliquée sur un seul objet.

Les données sous Dynamo sont partitionnées et répliquées sur plusieurs nuœds. Le partitionnement de données est basé sur le hachage uniforme pour distribuer la charge sur les nœuds de stockage. Chaque élément de données est attribué à un nœuds en appliquant une fonction de hachage sur la clé.

2.5.4 HDFS (Hadoop Distributed File System)

HDFS [37, 76] est un système de fichier distribué conçu pour le stockage de gros volumes de données et pour être fonctionné sur plusieurs machines de faible coût. Il offre aux applications un accès rapide aux données. HDFS est adapté pour les applications qui traitent une grande quantité de données. Il est largement inspiré de GFS.

Un cluster HDFS est composé d'un seul nœuds maitre (Namenode) et d'un ensemble de nœuds esclaves (Datanode) (voir la figure 2.4). Namenode et Datanode sont des logiciels conçus pour fonctionner sur des machines exécutant le système d'exploitation GNU/Linux. Le nœud maitre est chargé de conserver et gérer les métadonnées, pour tous les fichiers et les répertoires de l'arborescence du système de fichier. Il est chargé aussi de contrôler l'accès des clients aux fichiers et d'exécuter les opérations d'ouverture, fermeture, changement de nom de fichier, répertoire, ETC. Les métadonnées sont stockées d'une manière permanente sur le disque local sous forme de deux fichiers : l'image de l'espace de fichier et un fichier log.

Chaque nœud esclave est responsable de la gestion de stockage (création, suppression et réplication des blocs) sur le nœud qui l'execute et de servir les requêtes de lecture et d'écriture des clients du système de fichier, sous le contrôle de Namenode.

HDFS utilise le concept de bloc. Un bloc est défini comme étant la quantité de données minimum de lecture et d'écriture sur le disque. La taille d'un bloc HDFS est de 64MB par défaut. Un fichier sous HDFS est haché en un nombre de blocs qui sont stockés comme des unités indépendantes. Tous les blocs ont la même taille sauf le dernier et chaque bloc de fichier est stocké sur un Datanode différent. De plus, chaque bloc est dupliqué sur un petit nombre de machines, qui sont physiquement séparées, suivant un facteur de réplication. Le facteur de réplication est généralement égal à trois (trois copies de chaque bloc).

La requête client de création d'un fichier n'est pas directement transmise au nœud maitre.

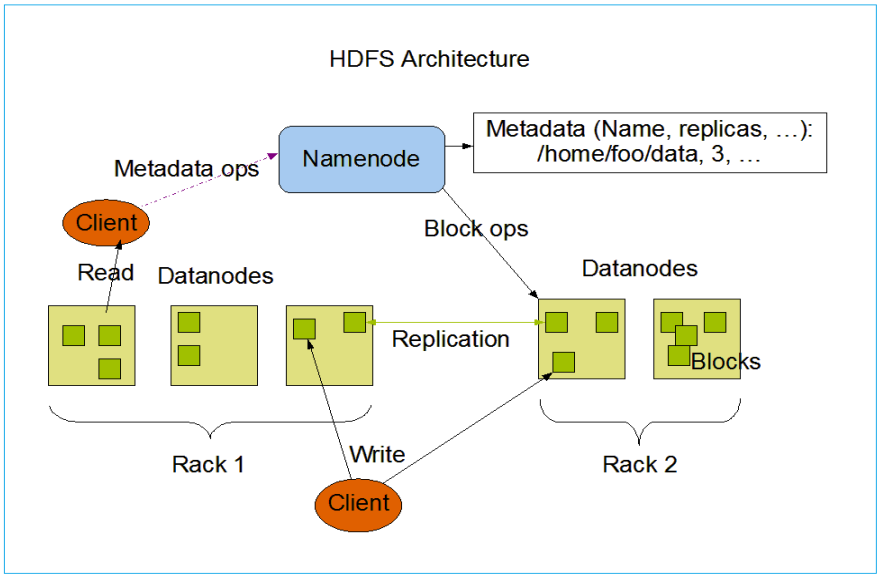


FIG. 2.4 – Architecture de HDFS [37]

Les données sont conservées dans un cache local. Lorsque le volume de données atteint la taille d'un bloc, le client HDFS demande au Namenode la liste des Datanode qui vont stocker les copies du bloc. Le client envoie le bloc de données vers le premier Datanode. Le premier Datanode enregistre ces données et les envoie vers le deuxième Datanode qui va les enregistrer et les envoyer vers le troisième Datanode dans la liste.

L'objectif principal de HDFS est de fournir un stockage fiable de données en présence des défaillances des machines. La fiabilité de HDFS s'appuie sur la réplication de données qui sont, généralement, dupliquées sur trois emplacements. la réplication de données permet de fournir la disponibilité de données et de faciliter la récupération de données en cas de défaillance d'une ou plusieurs machines. Il existe trois types de défaillances :(a) défaillance de Namenode (b) défaillance de Datanode (c) perte de la connectivité réseau entre les Namenode et les Datanode.

Le Namenode détecte le bon fonctionnement des Datanodes par le message "Heartbeat". Le message "Heartbeat" est envoyé périodiquement par les Datanodes au Namenode. Si le message "Heartbeat" n'est pas reçu d'un Datanode au bout d'un interval du temps, le Namenode considère que le Datanode est en panne. Par conséquent, les copies des blocs sont considérées comme perdues. Cela peut conduire à initialiser la réplication de certains blocs.

2.5.5 Synthèse

Le tableau 2.1 ci-dessous résume une comparaison entre les différents DFS du Cloud.

Système	**Modèle de données**	**Taille du bloc**	**Fournisseurs**	**Licence**	**Usage**
GFS	clé-valeur	64 MO	Google	Privée	Interne
Amazon S3	objet	limitéé par 5 GO	Amazon	Privée	Interne
Dynamo	clé-valeur	1 MO	Amazon	Privée	Interne
HDFS	clé-valeur	64 MO	Apach Hadoop	Open-source	Largement utilisé(Yahoo, Amazon, ...

TAB. 2.1 – Tableau récapitulatif des DFS du Cloud

2.6 Modèles et moteurs d'exécutions

2.6.1 MapReduce

Le paradigme MapReduce a rencontré un grand succès pour les applications qui s'appuient sur des grandes quantités de données. Il a été initialement proposé par Google pour faciliter le développement des applications de recherche web sur un grand nombre de machines. Dans cette section, nous présentons ce framework comme il a été défini dans le papier original [25] de Google.

2.6.1.1 Qu'est ce que c'est ?

MapReduce [25] est un modèle de programmation parallèle destiné pour le calcul distribué de grosses quantités de données. MapReduce est créé par Google, en 2003, dans le but de simplifier et répartir le traitement parallèle de données sur un grand nombre de machines avec une abstraction qui cache les détails de la couche matérielle aux programmeurs. Ce modèle de programmation est inspiré des primitives Map et Reduce de LISP et des langages de programmation fonctionnelles.

Dans MapReduce, les calculs des utilisateurs sont spécifiés par deux fonctions, la fonction de distribution Map et la fonction de réduction Reduce (voir la figure 2.5). Ces deux fonctions sont indépendantes et l'exécution de chaque fonction est automatiquement parallélisé sur les machines des clusters. Ces fonctions sont écrites par les utilisateurs en implèmentant la bibliothèque MapReduce avec certaines configurations telles que les noms des fichiers de lecture et d'écriture et la manière de partitionnement de données d'entrée en un ensemble de clé-valeur [25].

- ◇ **Map :** La fonction Map prend comme argument un couple clé-valeur et produit une liste de couples clé-valeur intermédiaires :

  Map (key1, value1) → list (key2, value2).

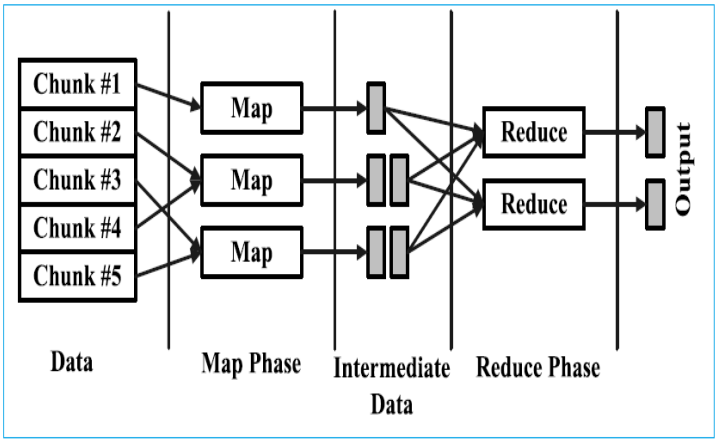


FIG. 2.5 – Modèle MapReduce [38]

Plusieurs instances de la fonction Map peut fonctionner à la fois, et tous les outputs générés sont transmis par un contrôleur maitre à l'une des fonctions Reduce, de sorte que toutes les paires qui ont la même clé seront envoyées à la même fonction Reduce.

◇ **Reduce :** La fonction Reduce prend comme argument une clé et la liste des valeurs intermédiaires générées par les différentes fonctions Map pour cette clé et combine ces valeurs ensemble selon un algorithme particulier pour fournir un résultat unique pour cette clé. Généralement, une instance Reduce se termine par produire une seule sortie. La fonction Reduce est définie comme suit :

Reduce (key2, list (value2)) → list (value2).

2.6.1.2 Exemple

Prenons un exemple simple et très populaire d'utilisation de MapReduce, cité dans [25]. Cet exemple traite le problème du comptage de nombre d'occurrences de chaque mot dans un ensemble de documents (voir la figure 2.6). Le déroulement de cet exemple est illustré par la figure 2.7 :

```
map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
        EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
// key: a word
// values: a list of counts
int result = 0;
for each v in values:
        result += ParseInt(v);
Emit(AsString(result));
```

FIG. 2.6 – Exemple d'un programme MapReduce

✓ Chaque fonction Map reçoit comme entrée un document dont la clé est le nom de ce document et la valeur c'est ses éléments.

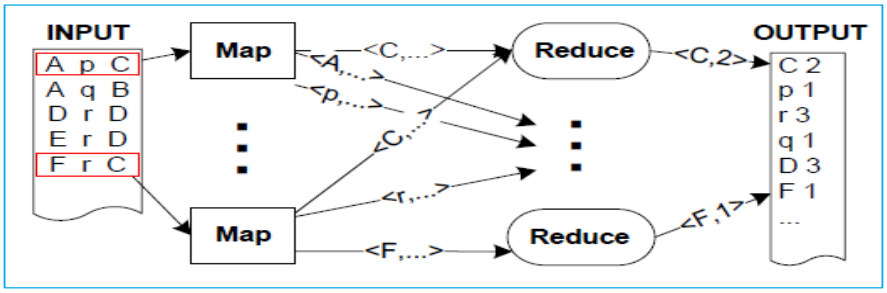


FIG. 2.7 – Exécution de l'exemple présenté dans la figure 2.6

- ✓ La fonction Map parcourt le document qu'elle a reçu comme paramètre et renvoie chaque mot trouvé et le nombre d'occurrences associé (seulement "1" dans cet exemple).
- ✓ Les résultats intermédiaires sont la liste des couples clé-valeur où les clés sont les différents mots trouvés dans les documents d'entrée et les valeurs sont la liste des valeurs envoyées pour chaque mot par Map. Les résultats intermédiaires des différentes clés sont groupés par la bibliothèque MapReduce avant qu'ils soient envoyés à la fonction Reduce.
- ✓ La fonction Reduce prend une clé (un mot) et la liste des valeurs associées à cette clé (le nombre d'occurrence "1") et faire la somme de ces valeurs. Le résultat obtenu est renvoyé comme sortie.

2.6.1.3 Flux d'exécution d'une opération de MapReduce

Quand le programme d'utilisateur fait appelle à la fonction MapReduce, les séquences suivantes se produisent [25] (voir la figure 2.8) :

1. D'abord, la bibliothèque MapReduce sépare les fichiers de données d'entrée en M morceaux de taille fixe de 16 à 64 MB par morceau. Ensuite, il commence des nombreuses copies du programme d'utilisateur sur les machines de cluster.
2. L'une des copies est spéciale, le *master*, qui affecte le travail aux autres copies qui sont des workers. Il y a M taches (processus) Map et R taches Reduce à attribuer. Le *master* attribue à chaque worker une tache Map ou une tache Reduce. Sachons que toutes les tâches Map doivent terminer avant que toutes les tâches Reduce ne peuvent commencer.
3. Le worker qui a la tache Map lit le contenu du morceau d'entrée correspondant. Il extrait les couples clés-valeur et transmet chaque couple à une instance de la fonction d'utilisateur Map. Les couples clé-valeur intermédiaires générées par les fonctions Map sont conservés en mémoire.
4. Les couples conservés en mémoire sont régulièrement écrits sur le disque local et sont partitionnés en R régions par une fonction de partitionnement. Les emplacements sur les disques locaux de ces couples sont transmis au *master*. Ce dernier transmit ces emplacements aux worker Reduce.

5. Quand un worker est notifié par le *master* de ces emplacements, il utilise les appels de procédure distante (RPC) pour lire les données stockées sur les diques locaux des worker Map. Quand un worker Reduce a lu toutes les données intermédiaires, il les trie par les clés intermédiaires de sorte que toutes les occurrences de la même clé soient regroupées. Si les données intermédiaires sont trop volumineuses pour les conserver en mémoire, un tri externe est utilisé.
6. Ensuite, le worker Reduce parcourt les données intermédiaires triées et pour chaque clé intermédiaire unique rencontrée, il passe la clé et la liste correspondante à la fonction utilisateur Reduce. La sotie de la fonction Reduce est ajoutée au fichier final de la sortie de cette partition Reduce.
7. Lorsque toutes les taches Map et Reduce ont été accomplies, le worker *master* réveille le programme utilisateur. À ce point l'appelant de MapReduce dans le programme utilisateur retourne vers le code utilisateur.

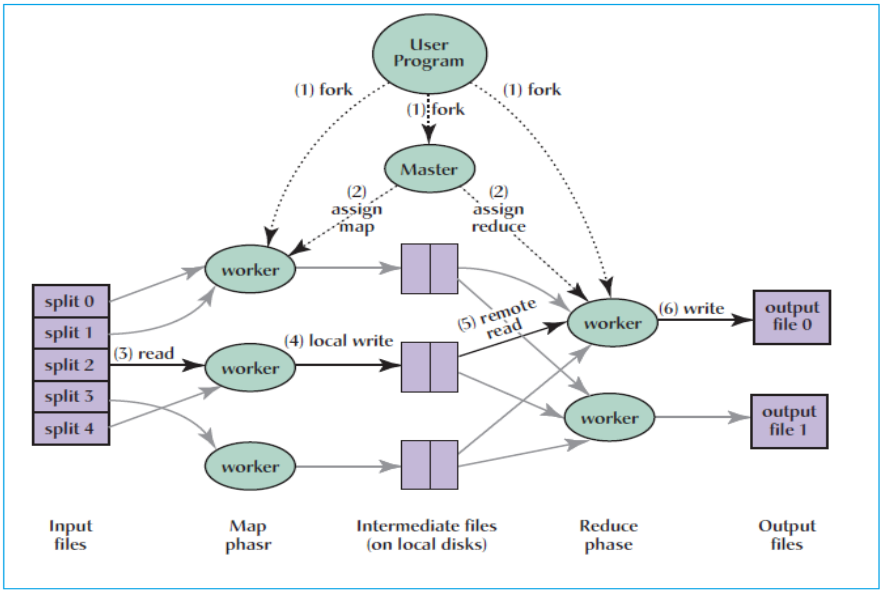


FIG. 2.8 – Flux d'exécution d'une opération de MapReduce [25]

2.6.1.4 Structures de données Master

Le *master* conserve plusieurs structures de données [25]. Pour chaque tâche Map et Reduce, il stocke l'état (inactif, en cours ou terminé), et l'identité de la machine worker (pour des tâches non-repos). Le *master* est le canal par lequel les emplacements des fichiers intermédiaires des tâches Map se propagent vers les tâches Reduce. Par conséquent, pour chaque tâche Map achevée, le *master* conserve les emplacements et les tailles des R fichiers intermédiaires générés par les tâches Map. Des modifications de ces informations sont reçues à chaque terminaison d'une tache Map. Ces informations sont envoyées progressivement aux workers Reduce qui sont en états "en cours".

2.6.1.5 Tolérance aux pannes

MapReduce est conçu pour des environnements où les erreurs et les défaillances ne sont pas exceptionnelles. MapReduce gère les pannes par la ré-exécution des workers en panne sur certaines machines dans le cluster. Le *master* interrompe périodiquement chaque worker et s'il ne reçoit aucune réponse d'un worker il le marque comme étant en panne. Ensuite, la tache du worker marquée en panne sera attribuée à un autre worker. Ainsi, une tache Map en panne et en état "terminée" sera réinitialisée à l'état "inactive" et ré-exécutée par un autre worker, du fait que ses résultats sont stockés sur le disque local de la machine en panne et ils sont inaccessibles. Par contre, une tache Reduce achevée, ne sera pas réinitialisée, du fait que ses outputs sont stockées dans le système de fichier global. Si la machine qui exécute le *master* tombe en panne, Mapreduce doit être ré-exécuté entièrement.

2.6.1.6 Implémentation

Chez Google MapReduce est implémenté sous GFS. La première version de la bibliothèque MapReduce été créée en 2003 utilisant le language de programmation C++ avec des interfaces Java et Pyton. Ce framework a été utilisé pour régénérer entièrement les index Internet de Google. Il a remplacé les ancients programmes utilisées pour la mise à jour et l'analyse de ces index. Avec MapReduce, Google a augmenté ses traitements au 100 To de données par jour en 2004 [25] pour le traitement de 20 Po par jour en 2008 [26].

Depuis sa proposition par Google, MapReduce a reçu beaucoup d'attention de la communauté industrielle et académique. Plusieurs implémentations de MapReduce ont été réalisées, chaque implémentation est destinée pour un environnement différent. Hadoop [37] est une implémentation open-source de MapReduce similaire à celle de Google, Map-Reduce-Merge [77] qui est une extension de MapReduce, Phoenix [65] est destiné pour les machines multi-processeurs, Mars [39] est développé pour les calculs parallèles sur les processeurs graphiques.

2.6.1.7 Avantages et limites

MapReduce est largement utilisé dans les plate-formes du Cloud. Il est un framework puissant, fiable et simple. Il permet de traiter de très gros volumes de données utilisant des primitives simples Map et Reduce. Il fournit aux programmeurs une abstraction totale des détails hardware et des mécanismes de parallélisation. Sa fiabilité est tiré du mécanisme de réplication de données et de la politique de la gestion des erreurs et des échecs. Cependant, les opérations de tris limitent les performances du Framework et le flux de données en deux étapes le rend très rigide.

2.6.2 MapReduce-Merge

MapReduce-Merge [77] est un autre modèle de programmation, qui s'étend MapReduce avec une troisième phase "Merge", qui permet de fusionner les résultats de deux différents

programmes MapReduce. MapReduce est attendu pour supporter l'algèbre relationnelle et le traitement des ensembles de données hétérogènes simultanément. Ce framework est développé par le group de recherche de Yahoo !. Les trois primitives Map, Reduce et Merge sont définies comme suit (voir la figure 2.9) :

$$Map : (k1, v1) \rightarrow [(k2, v2)]$$
$$Reduce : (k2, [v2]) \rightarrow (k2, [v3])$$
$$Merge : ((k2, [v3]), (k3, [v4])) \rightarrow (k4, [v5])$$

Dans la phase Map, la fonction Map transforme le couple clé-valeur $(k1, v1)$ à une liste de couple clé-valeur intermédiaires. Ensuite, la fonction Reduce accumule la liste des valeurs [v2] associé à k2 et produire une liste de valeurs $v3$ associé à $k2$. La troisième phase combine deux sorties de deux programmes MapReduce différents $(k2, [v3])$ et $(k3, [v4])$ dans une liste de sortie clé-valeur $[(K4, v5)]$.

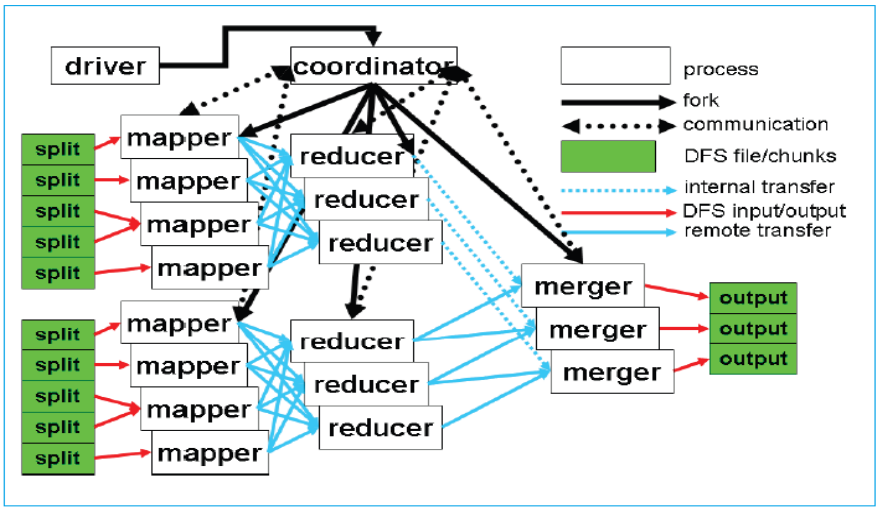


FIG. 2.9 – Flux de contrôle et de données du framework Map-Reduce-Merge [77]

Map-Reduce-Merge est utilisé pour implémenter les différents opérateurs relationnels tels que : la sélection, la projection, l'agrégation, le produit cartésien et les différents algorithmes de jointures : tri fusion, boucles imbriquées et hachage. Le traitement de ces opérations est réalisés en parallèle utilisant les trois primitives Map, Reduce et Merge. Les clés et les valeurs sont choisies de l'ensemble des attributs des relations impliquées.

2.6.3 Hadoop

Hadoop [37] est un projet open source géré par la Fondation Apach des logiciels libres (Apache Software Foundation). Ce projet, qui est constitué d'une collection de sous projets (voir la figure 2.10), est destiné pour le traitement par lot de grandes quantités de données sur des larges clusters d'ordinateurs. parmi les sous projets Hadoop, qui ont reçu un très grand succès et qui sont devenus très populaires, on trouve Hadoop MapReduce et HDFS

qui sont inspirés de MapReduce et GFS de Google. Hive, Hbase et pig ont trouvé leur succès dans la gestion des données structurées tels que les entrepôts de données.

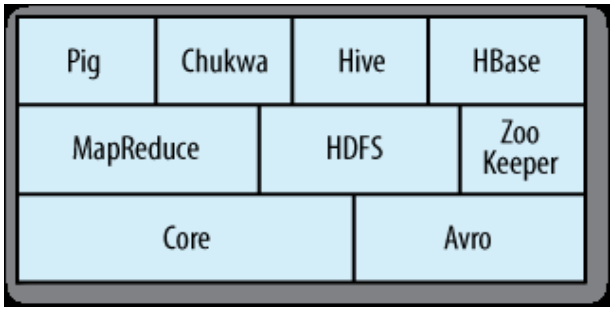


FIG. 2.10 – Les sous projets de Hadoop

2.6.3.1 Hadoop MapReduce

Bien que le framework MapReduce a trouvé un grand succès pour l'analyse et le traitement de grandes quantités de données sur des larges clusters, son implémentation chez Google malheureusement n'est pas libre. Alors que, Hadoop MapReduce [37, 76] est une implémentation Java open source de MapReduce sous HDFS (voir la figure 2.11). C'est un framework logiciel conçu pour faciliter l'écriture des applications qui traitent de grandes quantités de données en parallèle sur des clusters de grande taille (plusieurs milliers de nœuds) de manière fiable et tolérante aux pannes.

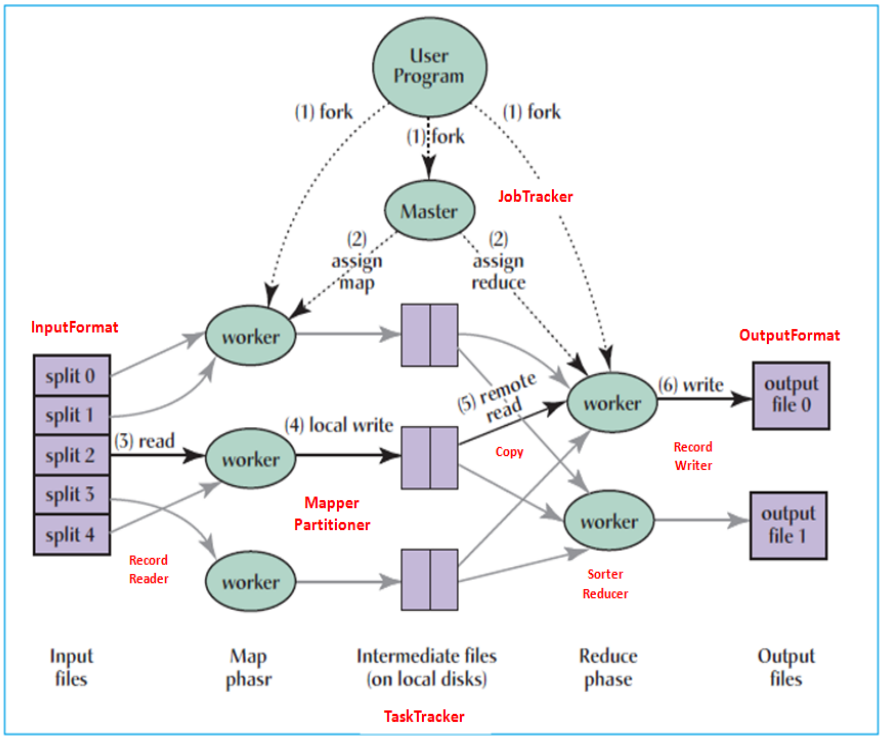


FIG. 2.11 – Implémentation Hadoop de MapReduce

Hadoop MapReduce est réalisé sur une architecture simple constituée d'un seul nœud maitre et plusieurs noeuds esclaves (voir la figure 2.12) :

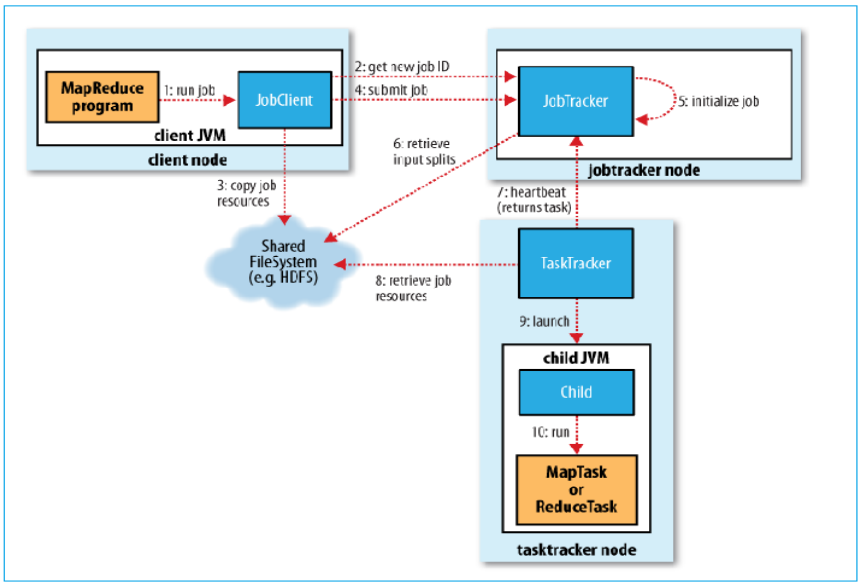


FIG. 2.12 – Cluster Hadoop [76]

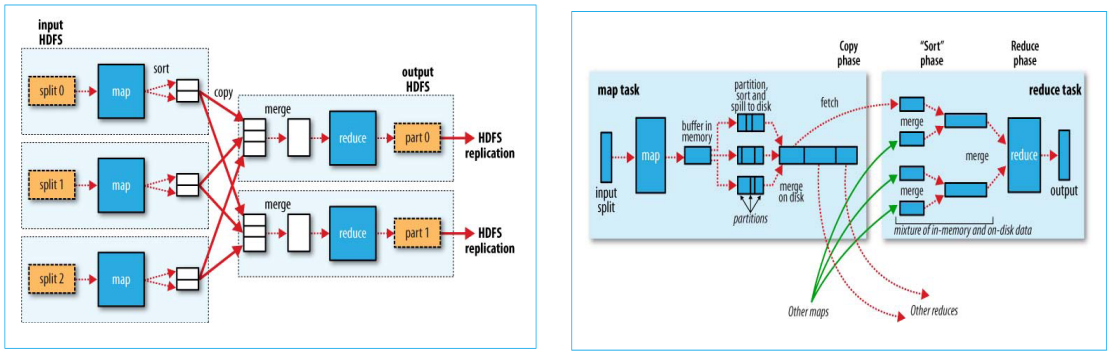
- ◇ **Hadoop job :** L'unité de travail de Hadoop (Hadoop job) est "MapReduce job", qui englobe les données d'entrée, le programme MapReduce et les informations de configuration [76]. Hadoop divise le job en deux types de taches : les taches Map et les taches Reduce.
- ◇ **Jobtracker et Tasktracker :** Les taches Map et les taches Reduce sont contrôlées par deux types de nœuds, un Jobtracker et un certain nombre de Tasktracker. Le rôle du nœud Jobtracker est de planifier les taches à exécuter dans le système sur les nœuds Tasktrackers. Les nœuds Tasktrackers exécutent les taches et envoient le rapport de progression au Jobtracker qui va garder la trace d'avancement globale de chaque job. En cas de défaillance d'une tache, le Jobtracker peut le re-planifier sur un autre Tasktracker.

2.6.3.2 Fonctions Map et Reduce

Hadoop divise les entrées de MapReduce en des morceaux de taille fixe (qui est généralement la taille d'un bloc HDFS, 64 MO) appelé *input split* ou *split*. Hadoop crée une seule tache Map pour chaque *split*, qui exécute la fonction Map définie par l'utilisateur. Les taches Map sont des taches indépendantes, qui transforment les enregistrements d'entrées en des enregistrements intermédiaires. Le nombre de Map est généralement généré à partir du nombre de *split*. Cependant le nombre des taches Reduce est spécifié indépendamment de la taille totale de données d'entrées. Les taches Reduce permettent de fusionner les sorties des taches Map par clé pour former le résultat final.

2.6.3.3 Les entrées/Sorties

Généralement, en raison d'efficacité, les inputs d'une tache Map se trouvent dans le même nœud d'exécution de cette tache. Les sorties de Map, qui seront les entrées des taches Reduce, sont triées et stockées sur le disque local. Lorsqu'il y a plusieurs taches Reduce, la tache Map crée pour chacune une partition de données, en utilisant généralement un partitionnement par hachage. Les enregistrements ayant la même clé se trouvent dans une seule partition. Les sorties de Map doivent être transférées à travers le réseaux vers le nœud où la tache Reduce s'exécute. Notons que l'entrée d'une seule tache Reduce est les sorties de toutes les taches Map (voir la figure 2.13(a)). Par la suite, ces données sont fusionnées et transférées à la fonction utilisateur Reduce. En raison de fiabilité, la sortie de chaque tache Reduce est stockée sous HDFS.



(a) Flux de données MapReduce avec de multiples tâches de Reduce

(b) Hadoop MapReduce Shuffle & Sort

FIG. 2.13 – Flux de données Hadoop MapReduce [76]

2.6.3.4 Shuffle & Sort

Une tache Reduce a trois phases principales : *shuffle, trier et Reduce* [37] (voir la figure 2.13(b)). *Shuffle* se réfère à la partie du processus où les sorties Map sont récupérées par les tâches Reduce [76]. Si les sorties des taches Map sont assez petites, elles sont copiées dans la mémoire des tasktrackers des taches Reduce. Dans le cas contraire elles sont copiées sur le disque. Lorsque toutes les sorties Map sont copiées, la tâche Reduce passe à la phase de tri, qui consiste à faire regrouper les sorties Map par clé.

2.6.4 Elastic MapReduce

Elastic MapReduce [66] est un service de Amazon qui est une implémentation de Hadoop MapReduce sur des instances Amazon EC2. Amazon Elastic MapReduce fonctionne en conjonction avec Amazon EC2 pour créer un cluster Hadoop, et avec Amazon S3 pour stocker les scripts, les données d'entrée, les fichiers journaux et les résultats de sortie. Le processus

Elastic MapReduce est présenté dans la figure 2.14. Les applications des utilisateurs sont

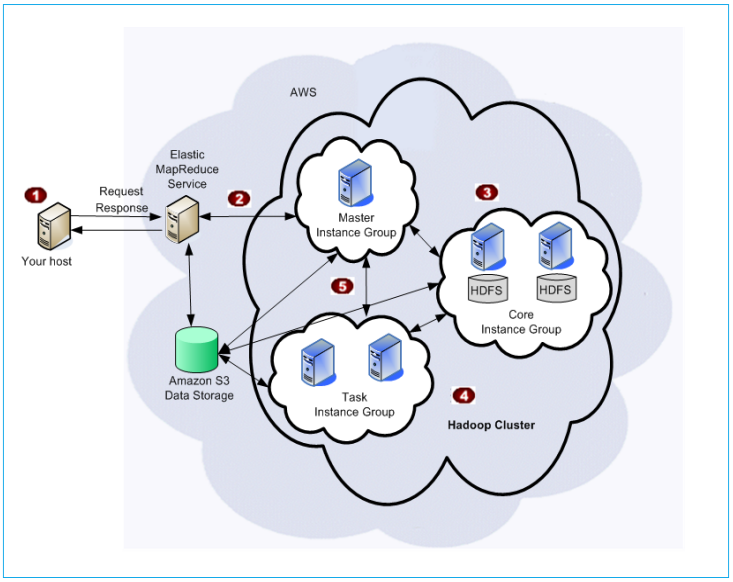


FIG. 2.14 – Architecture d'Elastic MapReduce [66]

développées dans des langages similaires à SQL, tels que Hive et Pig.

2.6.5 Dryad de Microsoft

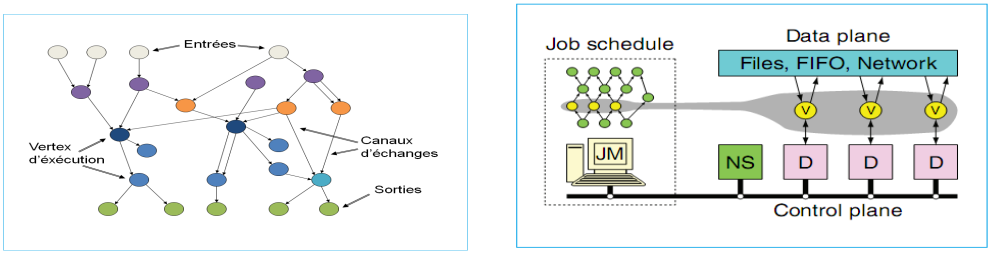
Dryad [41] est un moteur de calcul distribué destiné pour le traitement de données volumineuses sur des clusters. Il est introduit par Microsoft dans le but d'améliorer et d'optimiser les calculs intensifs. Dryad permet à un programmeur d'utiliser les ressources d'un cluster pour exécuter des programmes sur des données volumineuses en parallèle.

2.6.5.1 Structure d'un Job Dryad

Un job Dryad est un graph cyclique dirigé où les sommets (Vertices) sont les programmes et les arrêts représentent le flux de données entre eux (voir la figure 2.15(a)). L'exécution du job revient à exécuter les sommets en planifiant leurs exécutions sur les ressources disponibles et de transporter les données entre les sommets.

2.6.5.2 Architecture du système Dryad

L'organisation de Dryad est illustrée par la figure 2.15(b). Un système Dryad est constitué d'un gestionnaire de jobs (JM), un serveur de noms, les vertices et un demon (D). Un Job est coordonné par le gestionnaire de Jobs qui contient le code de l'application spécifique pour la construction du graph de communication du Job avec le code de la bibliothèque pour planifier le Job à travers les ressources disponibles. Les ordinateurs disponibles dans un cluster sont numérotés et localisés par le serveur de noms (NS). L'exécution des sommets est réalisé utilisant le demon D comme un proxy.



(a) job Dryad (b) Architecture de Dryad

FIG. 2.15 – Vue globale de Dryad [41]

2.6.5.3 Types des canaux

L'échange de données entre les sommets est implémentée soit par des fichiers temporaires où le producteur écrit sur le disque et le consommateur lit à partir de ce fichier, soit par l'établissement des connections TCP direct entre les sommets, ou par mémoire partagée FIFO.

2.6.6 Synthèse

Le tableau 2.2, donné à la fin du chapitre résume une comparaison entre les différents paradigmes d'exécution du Cloud.

2.7 Langages d'interrogation

2.7.1 Hive

Hive [72, 36] est une infrastructure d'entrepôt de données construite sur Hadoop. Cette solution open-source NoSQL est exploitée avec un langage similaire à SQL appelé HiveQL, qui permet aux utilisateurs d'analyser, structurer et intérroger des données volumineuses stockées dans des fichiers Hadoop. Hive est utilisé sur Amazon Elastic MapReduce et Amazon S3 pour rendre plus facile l'écriture de scripts d'analyse de données, sans connaissance approfondie du paradigme de développement MapReduce. Ainsi, Hive et Hadoop sont largement utilisés dans Facebook pour des différents types de traitement des données [36].

2.7.1.1 Modèle de données

Hive supporte les concepts des bases de données traditionnelles, telles que les tables, les colonnes, les lignes et les partitions. Les données sous Hive sont organisées sous forme de tables, où chaque table a un répertoire HDFS correspondant. Dans le répertoire les données sont partitionnées et distribuées dans des sous répertoires où chaque table peut avoir un ou plusieurs partitions. Les partitions sont aussi divisées en un ensemble de Buckets basés sur

le hachage d'une colonne dans la table. Chaque Buccket est stocké comme un fichier dans le répertoire de la partition.

Les tables de données sous Hive sont similaires aux bases de données traditionnelles. Chaque table est constituée d'un nombre de lignes et chaque ligne se compose d'un nombre de colonnes où chaque colonne a un type spécifié. Les types de données supportés par Hive sont les types simples tels que les entiers, les réels, les booléens, les strings et les types complexe tels que les tableaux, les listes et les structures.

2.7.1.2 HiveQL

Le language de requête HiveQL comprend un sous ensemble du language SQL. Il supporte les primitives de manipulation de données (DML) telles que la sélection, la projection, la jointure, l'agrégation, le groupement, l'union, le produit cartésien et les sous requêtes dans la clause from. Comme il supporte les primitives de définition de données (DDL) pour créer des tables et des partitions. Ainsi, il permet aux utilisateurs de charger des données provenant des sources externes et d'insérer les résultats des requêtes dans des tables Hive via les primitives de manipulation de données : la lecture et l'insertion. Cependant, HiveQL ne supporte pas la modification et la suppression des lignes dans des tables existantes.

2.7.1.3 Architecture de Hive

Les principales composantes de Hive sont les suivantes (voir la figure 2.16) :

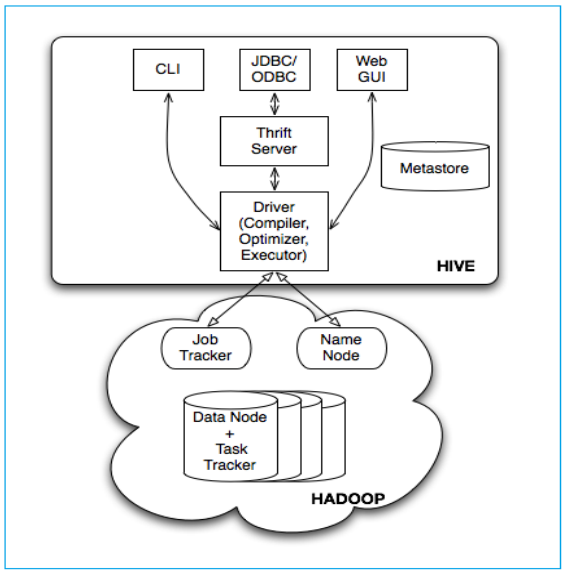


FIG. 2.16 – Architecture de Hive [72]

◇ **Les interfaces externes** : Hive fournit des interfaces utilisateurs invite de commande, des interfaces utilisateur web et des interfaces pour la programmation des applications (API) tels que ODBC et JDBC.

- ◇ **Le serveur Thrift Hive** : Ce composant fournit une interface thrift et un serveur JDBC/ODBC, comme il fournit un moyen d'interagir avec d'autres applications ou d'autres scripts écrites en d'autres languages.
- ◇ **Le Metastore** est le composant qui stocke le catalogue de système et des métadonnées sur les tables, les colonnes et leurs types, les partitions et les emplacements des tables. Ces métadonnées sont spécifiées lors de la création de la table et ré-utilisées à chaque fois que la table est référenciée dans HiveQL.
- ◇ **Le Driver** : Le driver gère le cycle de vie des primitives HiveQL durant la compilation, l'optimisation et l'exécution.
- ◇ **Compilateur** : Le compilateur compile les requêtes HiveQL en un graph cyclique dirigé des taches MapReduce. Ensuite, ces taches sont envoyées au moteur d'exécution Hadoop.

2.7.1.4 Compilation & Optimisation

Le compilateur transforme les requêtes HiveQL en un plan d'exécution logique, qui correspond à un DAG des Jobs MapReduce. L'optimiseur, effectue plusieurs passes sur le plan logique et il le réécrit de plusieurs manières, en utilisant un petit nombre de règles. L'optimiseur est un optimiseur à base de règles qui s'appui sur un petit nombre de règles pour optimiser la quantité de données transférées entre les différentes taches MapReduce.

2.7.2 SCOPE

SCOPE (Structured Computations Optimized for Parallel Execution) est un nouveau langage déclaratif destiné pour le traitement intensif d'analyse de données et il est utilisé quotidiennement pour une variété d'analyses de données et des applications de data mining au sein de Microsoft [19]. Ce langage est largement inspiré de SQL, ce qui permet aux utilisateurs familiers de SQL de le comprendre facilement.

2.7.2.1 Plate-forme d'exécution

Microsoft a développé une plate-forme du calcul distribué pour le stockage et l'analyse de grosses quantités de données sur des larges clusters, appelée Cosmos (voir la figure 2.17). Les principaux composants de Cosmos sont les suivants :

- ◇ **Le système de stockage Cosmos** : C'est un sous-système de stockage distribué conçu pour stocker efficacement de manière fiable de très gros fichiers séquentiels.
- ◇ **L'environnement d'exécution** : Est un environnement de déploiement, d'exécution et de débogage des applications distribuées. Une application est modélisée sous forme d'un graph de flux de données DAG.
- ◇ **SCOPE** : langage de script de haut niveau pour l'écriture des programmes d'analyse des données. Le compilateur et l'optimiseur SCOPE traduisent les scripts à des plans d'exécution parallèle efficaces.

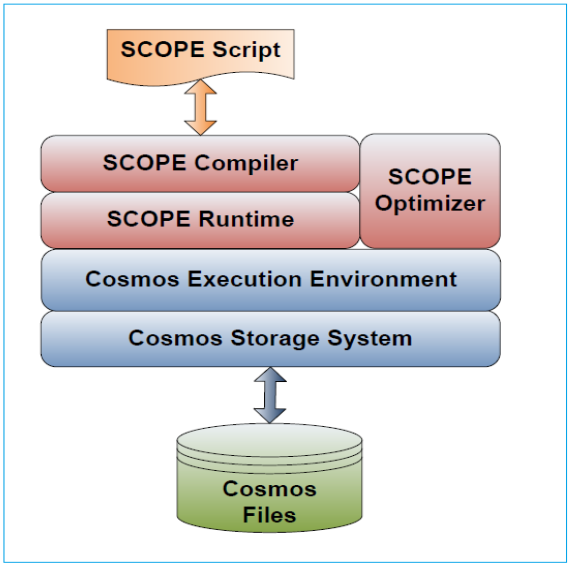


FIG. 2.17 – Environnement d'exécution de SCOPE [19]

2.7.2.2 Langage de script SCOPE

SCOPE est un langage de script qui rassemble à SQL mais avec des expressions C# qui peuvent utiliser la bibliothèque C# [19]. Similaire au langage SQL, les données sous SCOPE sont modélisées comme un ensemble de lignes, où chaque ligne est composée d'un ensemble de colonnes, chacun a un type spécifique. Les types de données supportés par SCOPE sont les entiers, les réels, les dates, les strings et les booléens.

SCOPE comprend la commande *Select* de SQL et les différentes fonctions d'aggregation telles que : COUNT, AVG, MIN, MAX, SUM, COUNTIF, FIRST et LAST. Dans SCOPE, les requêtes imbriquées ne sont pas supportées.

D'autres fonctions sont introduites dans SCOPE : PROCESS, REDUCE et COMBINE. Ces fonctions complètent la commande Select et offrent les mêmes fonctionnalités des fonctions Map et Reduce du modèle de programmation MapReduce et la fonction Merge de Mapreduce-Merge.

2.7.2.3 Compilation & Optimisation

Le compilateur et l'optimiseur SCOPE sont chargés de générer un plan d'exécution efficace. Le compilateur examine le script, vérifie la syntaxe et résout les noms. Le résultat de la compilation est un arbre parse interne qui est ensuite transformé en un plan d'exécution physique. Le plan d'exécution physique est une spécification d'un Job Cosmos qui décrit le DAG de flux de données. Ensuite, le DAG est construit et ordonné par le Job Manager (JM).

Le compilateur invoque l'optimiseur pour trouver le meilleur plan d'exécution des requêtes complexes. L'optimiseur SCOPE est un optimiseur à base de règles, il s'appuie sur des règles de transformation pour générer tous les plans possibles correspondant aux expressions de la

requête. Ensuite, l'optimiseur choisit le plan dont le coût estimé est le minimum.

2.7.3 Sawzall

Sawzall [61] est un nouveau langage de programmation procédural interprété, utilisé chez Google sous MapReduce pour l'analyse parallèle de grosses quantités de données distribuées sur plusieurs machines. Ce langage est largement influencé par les langages de programmation C et Pascal.

2.7.3.1 Types de données

Les différents types de données supportés par Sawzall sont les types simples et les collections. Les types simples sont les entiers (int, désigne une valeur de 64 bits signé), virgule flottante (float), des entiers spéciaux $'time'$ et $'fingerprint'$, le type *byte*, qui est similaire à un tableau C de type *char* et le type *string*.

Les collections englobent les tableaux, les maps et les tuples. Les tableaux sont indexés par des valeurs entières, les maps sont comme des tableaux associatifs ou des dictionnaires Python et peuvent être indexés par tout type avec des indices non ordonnées, et les tuples représentent le groupement arbitraire de données, comme une structure C ou un enregistrement Pascal. Ansi, Sawzall fournit la conversion des valeurs d'un type à un autre.

2.7.3.2 Modèle de calculs

La figure 2.18 illustre le modèle de calcul de Sawzall. La source de données est une collection des enregistrements stockés dans des fichiers GFS, qui sont traités en deux phases : la phase de traitement et la phase d'agrégation. La phase de traitement est l'exécution d'un programme utilisateur Sawzall, qui prend un seul enregistrement à la fois. Le résultat de traitement est envoyé à un ou plusieurs agrégateurs externes par la seule primitive Sawzall de sortie $'emit'$. Un agrégateur permet d'accumuler les résultats de la première phase.

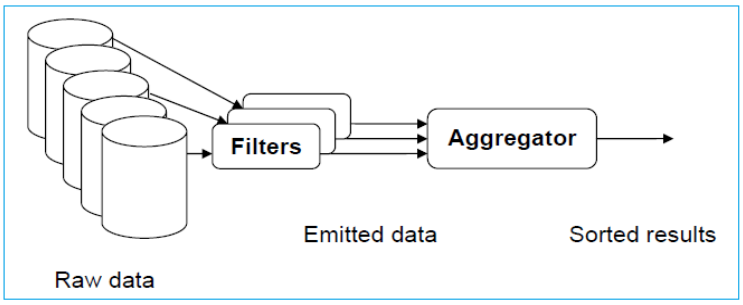


FIG. 2.18 – Modèle de calculs de Sawzall [61]

2.7.3.3 Agrégateur

L'agrégation des données est effectué par des fonctions d'agrégation qui sont écrites dans un autre langage (comme C + +) et mise en disposition par la bibliothèque. Pour un programmeur Sawzall, il suffit d'indiquer la fonction utilisée, ensuite la parallélisation et la planification des tâches agrégateur sont réalisés par le système.

Les agrégateurs s'appellent tables dans Sawzall. Il sont déclarés et typés par le mot clé *table*. Les principaux agrégateurs Sawzall sont les suivants :

- ◇ **sum** : calcul la somme de toutes les valeurs envoyées.
- ◇ **collection** : renvoie une simple liste de toutes les valeurs envoyées, dans un ordre arbitraire.
- ◇ **sample** : comme *collection*, mais choisit un échantillon des valeurs émises. La taille de l'échantillon désirée est fournie comme un paramètre.
- ◇ **maximum** : les valeurs sont étiquetées par un poids et la valeur dont le poids est le plus élevé est choisie.
- ◇ **top** : estime les valeurs qui sont les plus populaires.
- ◇ **unique** : calcul la taille de l'ensemble des valeurs d'entrée. Le résultat est toujours un nombre, quelque soit le type des valeurs envoyées.

2.7.4 Pig Latin

Pig latin est un langage d'interrogation de haut niveau développé par Olston et al. [55]. Ce langage est conçu pour être un intermédiaire entre le langage déclaratif de haut niveau SQL et le modèle de programmation MapReduce. Il est utilisé par des programmeurs au sein de Yahoo ! pour simplifier la création des taches d'analyse de grandes quantités de données.

2.7.4.1 Programmes Pig

Un programme Pig est similaire à une spécification d'un plan d'exécution. Un programme est décrit comme étant une séquence des étapes où chaque étape représente une seule transformation de données. Ces étapes de transformation de données sont définies utilisant des primitives analogues aux primitives SQL telles que l'extraction, le groupement et l'agrégation de données.

Pig est extensible, il est conçu pour supporter des fonctions définies par les utilisateurs (UDFs). Ces fonctions peuvent être écrites en Java ou autre langage (C/C++, Perl et Python) et permettent de définir un traitement de données spécialisé.

Pig Latin fournit un nombre d'opérateurs qui sont similaires aux primitives du langage SQL. On peut citer comme éxemple : LOAD, STORE, FILTER, JOIN, COGROUP, DISTINCT, ETC.

2.7.4.2 Modèle de données

Le modèle de données Pig contient quatre types de données :

1. **Atom** : contient une valeur atomique simple telles que une chaîne de caractère ou un nombre par exemple $'Bob'$ ou 24.
2. **Tuple** : est une séquence d'éléments, chacun peut être de n'importe quel type de données, par exemple $('alice','laker')$
3. `Bag` : est une collection de tuples avec la possibilité de la duplication, par exemple : $\{('alice','laker'),('alice',('ipod','apple'))\}$
4. **Map** : est une collection des éléments de données où chaque élément a une clé associée. Tous les éléments de données dans Map ne sont pas forcément de même type. Exemple : $\begin{bmatrix} `fan\ of' \rightarrow ('alice','lakers') \\ 'age' \rightarrow 20 \end{bmatrix}$

2.7.4.3 Compilation & Optimisation

Les programmes Pig Latin sont compilés en des séquences de Jobs MapReduce et qui sont exécutés utilisant l'environnement Hadoop MapReduce. Chaque programme Pig passe par une série d'étapes de transformation avant d'être exécuté [31]. Ces étapes de compilation et d'exécution sont illustrées par la Figure 2.19 : La première étape est l'étape d'analyse.

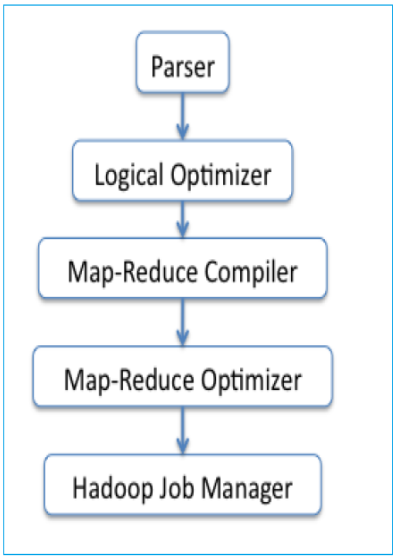


FIG. 2.19 – Étapes de compilation et d'exécution de Pig [31]

L'analyseur (parser) vérifie que le programme est syntaxiquement correct et que toutes les variables référencées sont définies. Le résultat de l'analyseur est un plan logique canonique avec une correspondance un à un entre les états Pig latin et les opérateurs logiques qui sont rangés dans un graphe acyclique dirigé (DAG). Le plan logique généré par l'analyseur

est passé à un optimiseur logique pour réaliser des optimisations logiques. Le plan logique optimisé est alors compilé en une série de Jobs MapReduce qui sont ensuite passés à travers une autre phase d'optimisation. Le DAG des Jobs MapReduce optimisé est alors logiquement trié et les Jobs sont envoyés à Hadoop pour l'exécution.

2.7.5 DryadLINQ

Dryadlinq [79] est un environnement de programmation simple, puissant et élégant, utilisé au sein de Microsoft pour la création des taches de traitement parallèle des données volumineuses distribuées sur de large clusters. Un programme DryadLINQ est un programme séquentiel composé des expressions LINQ [50]. Le système DryadLINQ compile les programmes LINQ en un plan d'exécution distribué s'exécute dans la plate-forme d'exécution Dryad [41]. La pile des logiciels DryadLINQ est illustrée par la figure 2.20.

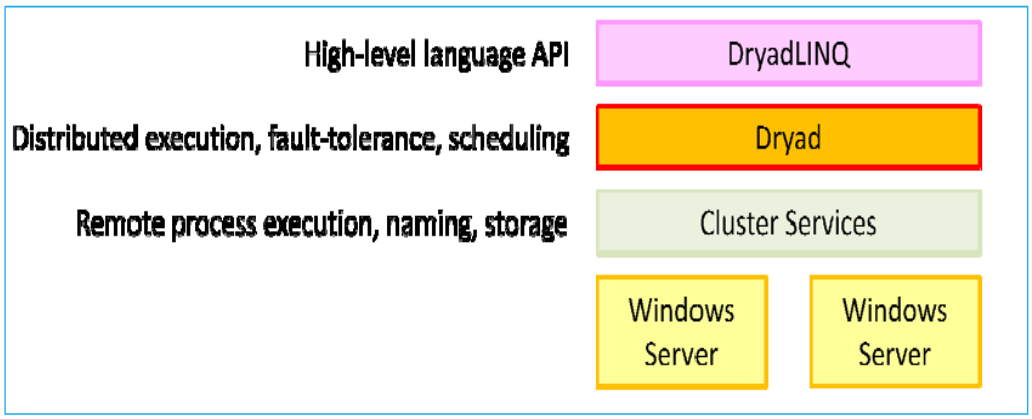


FIG. 2.20 – Pile des logiciels DryadLINQ

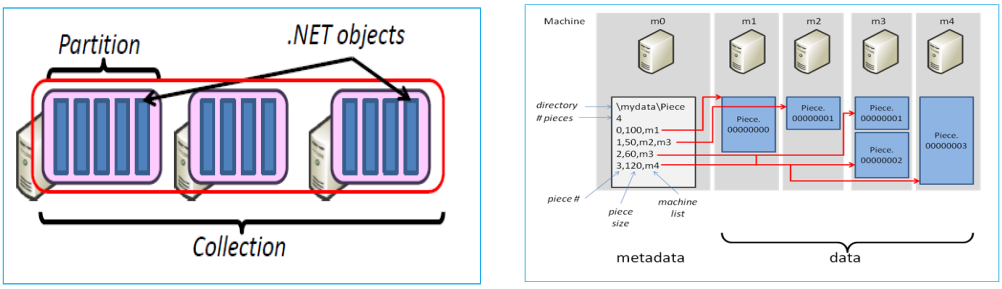
2.7.5.1 LINQ

LINQ (Language INtegrated Query) est un nouveau langage créé par Microsoft. Il est intégré dans des langages .NET (C#, VB, ...) . Il ajoute aux langages C# et Visual Basic et autres langages .NET des capacités d'interrogation des données avec une syntaxe proche de celle de SQL. Il fournit un ensemble des opérateurs permettant d'effectuer des requêtes sur n'importe quelle source de données. Il supporte aussi les opérateurs relationnels tels que la sélection, la jointure, le groupement et l'agrégation. Bien que les extensions LINQ ont été apportées aux langages Visual Basic et C#, le compilateur DryadLINQ ne supporte que C#.

2.7.5.2 Modèle de données

DryadLINQ permet de traiter des collections de données LINQ de type .NET, qui sont partitionnées et distribuées sur des ordinateurs de cluster (voir la figure 2.21(a)). La stratégie de partitionnement de données utilisée est : le partitionnement par hachage, par intervalle

et le partitionnement par round-robin. Un fichier partitionné sur le disque est composé de deux parties (voir la figure 2.21(b)) : (a) Les partitions de données qui sont toutes placées dans le même répertoire sur toutes les machines et (b) Les métadonnées. Les métadonnées représentent une description de tous les morceaux d'un fichier partitionné. Cette description comporte le nom de chaque partition, le nom de répertoire de stockage, le nombre de partitions, la taille de la partition en octet et la liste des machines de stockage (une partition peut être répliquée sur plusieurs machines).



(a) Modèle de données DryadLINQ [79] (b) Structure des fichiers partitionnés [49]

FIG. 2.21 – Modèle de données & Structure des fichiers de DryadLINQ

2.7.5.3 Optimisation des requêtes

Les expressions LINQ sont converties par DryadLINQ en un plan d'exécution graphique (EPG). L'EPG est un graphe acyclique dirigé, où chaque nœud est un opérateur et les arêtes représentent ses E/S. Ensuite DryadLINQ applique des optimisations sur l'EPG. L'optimiseur DryadLINQ est similaire à des optimiseurs des bases de données classiques [42]. Il est constitué de deux composantes, une composante statique et une composante dynamique.

Les optimisations statiques s'appuient sur une heuristique glotante [79]. Elles sont des règles de réécriture conditionnelle graphique sur l'EPG. La plupart des optimisations statiques sont concentrées sur la réduction des E/S disque et réseau. Les optimisations dynamiques sont appliquées lors de l'exécution des Jobs Dryad, et consistent à réécrire le graphe des Jobs selon les statistiques sur les données de l'exécution.

2.7.6 Synthèse

Le tableau 2.3 ci-dessous résume une comparaison entre les différents langages d'interrogation du Cloud.

Paradigme	Modèle de données	Système de fichier	Licence	Fournisseur
MapReduce	clé-valeur	GFS	Usage interne chez Google	Google
MapReduce-Merge	clé-valeur	-	Modèle de programmation	Yahoo !
Hadoop	clé-valeur	HDFS	Open-source	Apach Hadoop
Elastic MapReduce	clé-valeur	HDFS, S3	Usage interne chez Amazon	Amazon
Dryad	-	-	Usage interne chez Microsoft	Microsoft

TAB. 2.2 – Tableau récapitulatif des paradigmes d'exécution du Cloud

Langage	Modèle de données	Type	Compilation	Fournisseurs	License	Usage
Hive	tuple (ligne+ colonne)	SQL-like	MapReduce	A. Hadoop	Open-source	Facebook, Amazon, ...
SCOPE	tuple	SQL-like	DAG	Microsoft	Privé	Interne chez Microsoft
Sawzall	données simple, collection	procedural	MapReduce	Google	Privé	Interne chez Google
Pig	atom, tuple, bag, map	SQL-like	MapReduce	A. Hadoop	Open-source	Yahoo !
Dryadlinq	objet.net	SQL-like	Dryad	Microsoft	Privé	Interne chez Microsoft

TAB. 2.3 – Tableau récapitulatif des langages d'interrogation du Cloud

2.8 Conclusion

Dans ce chapitre, nous avons exposé la notion de gestion de données sur le Cloud. Dans ce contexte, le chapitre présente les principaux nouveaux systèmes de fichier distribués. Ces systèmes représentent le niveau de stockage de données, tandis que le traitement et l'interrogation de données reposent sur des nouveaux paradigmes et des nouveaux langages d'interrogation. Ces langages sont analogues au langage de requêtes SQL et ils comportent divers prémitives de celui-ci. Les nouveaux paradigmes, dont le plus populaire est celui de Google MapReduce et son implémentation open source Hadoop, sont destinés pour le traitement intensif de données volumineuses distribuées sur de larges clusters.

Nous avons vu aussi que les applications d'analyse de données sont parmi les premières applications de gestion de données à être déployées dans le Cloud. La plupart des fournisseurs du Cloud proposent un service de stockage dans lequel le modèle relationnel et le langage de requête SQL ont disparu et sont remplacés par le NoSQL. Dans le suivant chapitre, nous présentons la notion NoSQL et la gestion des entrepôts de données qui sont plus utilisés dans le domaine d'analyse de données.

Chapitre 3

Données analytiques sur le Cloud

3.1 Introduction

Avec la réputation progressive du Cloud Computing, de multiples applications se déplacent vers le Cloud. Les applications de gestion de données analytiques sont des candidates éventuelles pour le déploiement sur le Cloud [1]. Ces applications sont destinées aux processus d'aide à la décision, qui requièrent l'analyse d'un grand masse de données. Ces données sont souvent stockées au sein d'un entrepôt de données.

Aujourd'hui, les entrepôts de données forment la base de différents types d'analyse de données : OLAP, datamining, statistiques,... [24]. Ils sont plus utilisés dans le domaine d'analyse de données, du fait qu'ils permettent une analyse efficace de grosses quantités de données. Les processus d'analyse de données nécessitent des requêtes décisionnelles complexes et consomment un temps d'exécution non négligeable, ce qui nécessite l'utilisation des structures d'optimisations pour améliorer la performance de ces requêtes.

Depuis des années, les SGBDs présentent la plateforme standard et robust pour l'entreposage de données. Dans le Cloud Computing, les DFSs, MapReduce et autres systèmes de gestion de données NoSQL (Not Only SQL) présentent une nouvelle variante pour telle application. Ces nouvelles technologies ne supportent pas la propriété ACID des bases de données traditionnelles et s'appuient sur des nouveaux modèles de données, dont le plus populaire est le modèle de données clés-valeur.

MapReduce a reçu un grand intérêt pour le traitement massivement parallèle d'énormes quantités de données, à cause de sa tolérance aux pannes et sa capacité de fonctionner dans un environnement hétérogène. Ce paradigme est peut être vu comme un complément d'un SGBD pour les applications analytiques [70]. Ainsi, les opérations d'algèbre relationnelle sont facilement implémentées par MapReduce.

Le présent chapitre est consacré à la présentation de la gestion de données analytiques sur le Cloud et la définition des différentes problématiques liées à l'optimisation des requêtes décisionnelles. Ainsi que la description des différents algorithmes d'accès aux données dans un environnement MapReduce.

3.2 Bases de données NoSQL

L'acronym NoSQL signifie "Not Only SQL", littéralement "pas seulement SQL". C'est un movement qui est né avec les ténors du Cloud, dont l'objectif est de proposer une alternative au language SQL et au modèle relationnel des bases de données traditionnelles. Cette alternative permet de répondre aux besoins de traitement des volumes de données énormes, du fait que les SGBDs classiques présentent des limites lorsqu'il s'agit de gérer de grandes quantités de données et d'importantes charges. Les bases de données NoSQL se caractérisent par des schémas dynamiques ou l'absence de schéma, le partitionnement horizontal sur plusieurs noeuds et la réplication des données. Les bases de données NoSQL répondent aussi au théorème du CAP d'Eric Brewer [16] qui est plus adapté aux systèmes distribués et souvent ne tentent pas de fournir la garantie ACID.

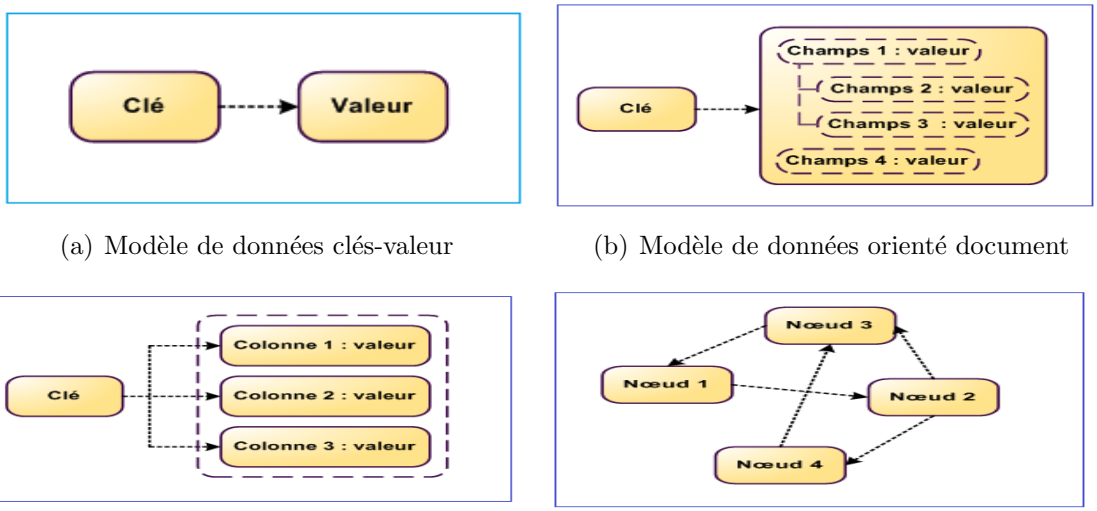
3.3 Types des bases de données NoSQL

1. **Les bases clé-Valeur :** Il n' y a pas de modèle de données et tout type de données peut être associé à une clé (voir la figure 3.1(a)). Il s'agit d'une grosse table de hashage persistée. L'accès à un élément stocké est par son identifiant uniquement. Ce genre de stockage convient pour le stockage des informations rarement modifiées mais beaucoup accédées. Le premier exemple d'utilisation de cette technologie est Amazon.
2. **Les bases orientées documents :** Il s'agit d'une extension du concept de clé-valeur qui représente la valeur sous la forme d'un document (voir la figure 3.1(b)). Ce sont des bases de données utilisées pour la gestion de données type documents. Le principe est de regrouper au même endroit toutes les données.
3. **Les bases orientées colonnes :** Ce sont des bases de données qui stockent les données par colonne et non par ligne (voir la figure 3.1(c)). Une base de données orientée colonne stocke les valeurs d'une colonne ensemble, puis les valeurs de la colonne suivante, etc.
4. **Les bases orientées graphes :** On utilise ces bases pour gérer un graphe (voir la figure 3.1(d)) : comme les liaisons entre les membres d'un réseau social.

3.4 Théorème de CAP

Le théorème de CAP est formulé par Eric Brewer (UC Berkeley) [16]. D'après ce théorème, un système de calcul distribué ne peut pas garantir à un instant T les trois propriétés suivantes simultanément :

- ◇ **Cohérence (consistency) :** Tous les noeuds du système voient exactement les mêmes données au même temps.
- ◇ **Disponibilité (Availability) :** La petre de noeuds n'empêche pas la continuité de fonctionnement correcte.



(a) Modèle de données clés-valeur

(b) Modèle de données orienté document

(c) Modèle de données orienté colonne

(d) Modèle de données orienté graph

FIG. 3.1 – Modèles de données NoSQL [29]

◇ **Résistance au partitionnement (Partition Tolerance) :** Aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement (en cas de morcellement en sous réseaux, chacun doit pouvoir fonctionner de manière autonome).

3.5 Systèmes de gestion de données structurées/semi-structurées sur le Cloud

De nombreux systèmes de gestion de données à base de Cloud ont été proposés. Shi et al. [69] ont classifié ces systèmes en deux catégories : (1) Systèmes à base des systèmes de fichiers (2) Systèmes à base des SGBDs. Ces systèmes sont largement utilisés pour les applications de gestion de données analytiques ou pour la gestion de données web.

3.5.1 Système de gestion de données à base des systèmes de fichiers distribués

Les systèmes de gestion de données à base des systèmes de fichiers distribués, tels que Hbase [18], Bigtable de Google [21] et Hive [72], exploitent les systèmes de fichiers distribués pour le stockage de données. Ces systèmes ne sont pas relationnels, ils sont basés sur des modèles de données de type NoSQL et ils ne supportent pas le langage d'interrogation SQL. Généralement, ces systèmes sont combinés avec MapReduce, ce qui leurs permet de bénéficier de ses avantages tels que la tolérance aux pannes, l'évolutivité et l'adaptation à des environnements hétérogènes, etc.

3.5.2 Systèmes de gestion de données à base des SGBDs

Les systèmes de gestion de données à base des SGBDs, tels que HadoopDB [2], PNUT de Yahoo ! [23] et SQL Azure de Microsoft, utilisent les SGBDs traditionnels pour le stockage de données. Ce type de système supporte le langage SQL et s'appuie sur le modèle de données relationnel, avec une certaine variété pour supporter des applications distribuées. Ainsi, l'optimisation des requêtes et l'indexation sont facilement utilisées dans tels systèmes.

3.6 Entrepôts de données

Un entrepôt de données est une base de données regroupant l'ensemble des données fonctionnelles de l'entreprise destinée aux processus d'aide à la décision. Ce concept est apparu dans les années 1990.

Définition 3.6.1. Définition de Bill Inmon [40] : "Un entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles, historisées et utilisées pour supporter un processus d'aide à la décision"

Les données dans un ED sont très volumineuses structurées par thème, issues de différentes sources hétérogènes, une fois créées ne sont jamais mis à jours (les deux seules opérations appliqués sont l'alimentation périodique de l'entrepôt et la lecture des données) et représentent l'activité d'une entreprise pendant une longue période.

L'entrepôt de données est basé sur une modélisation multidimensionnelle qui est une variante de modèle E/A dans les BD. Le modèle multidimensionnel représente les données dans un cube qui permet de voir les données suivant plusieurs dimensions qui déterminent une mesure d'intérêt "le fait". Les entrepôts de données sont généralement modélisés par un schéma en étoile [40] contenant une table de faits centrale de très grande taille et un certain nombre de tables de dimension de plus petite taille représentent les descriptions des faits (voir la figure 3.2).

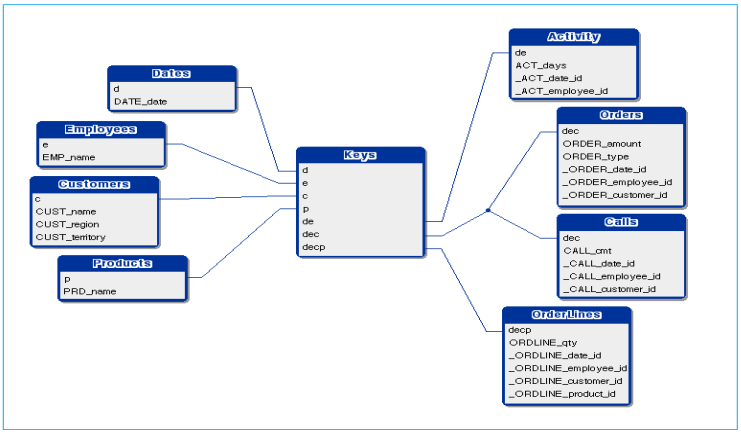


FIG. 3.2 – Exemple d'une modélisation en étoile d'un entrepôt de données

3.7 Gestion et analyse des entrepôts de données

Depuis longtemps les SGBDs présentent la technologie standard pour l'entreposage de données, notamment avec l'apparition des SGBDs parallèles qui présentent un plate-forme robust et hautement performant [59], basés sur le modèle relationnel et fournit SQL comme langage d'interrogation standard. Les DFs et MapReduce présentent une nouvelle technologie pour telles applications [57]. Ils ont montré leur performance pour la gestion d'énorme quantité de données où les SGBDs présentent des limites. Cette nouvelle plate-forme massivement parallèle de traitement de données est jugé le meilleur candidat pour les applications complexes d'analyse de données (datamining et de clustering, ...) [70], du fait que ces applications nécessitent un flux de données complexe.

MapReduce n'exige pas un schéma préalable de données et fonctionne bien sur les données non structurées et rarement modifiées, alors qu'un SGBD convient pour les requêtes transactionnelles et pour les ensembles de données qui sont structurés, normalisés et continuellement modifiés. Plusieurs études comparatives ont été réalisées entre MapReduce et les SGBDs [48, 57, 59, 70]. Le tableau 3.1 récapitule les points communs et les différences entre MapReduce et les SGBDs.

MapReduce peut être vu comme un complément d'un SGBDs pour les applications analytiques, du fait que divers problèmes d'analyse complexe nécessitent les capacités fournies par les deux technologies [70]. Aujourd'huit, les constructeurs et les éditeurs des SGBDs, tels que Teradata, Microsoft et Sybase s'ouvrent à l'architecture MapReduce et le voient comme le complément idéal à leur offre[1]. Au sein de Microsoft, Hadoop sera intégré à SQL Server et à son offre Windows Azure [2].

3.8 Optimisation des requêtes OLAP

Les requêtes OLAP définies sur un schéma en étoile nécessitent une ou plusieurs jointures entre la table de fait et les tables de dimension. Le coût de traitement de ces jointures (connues par requêtes de jointure en étoile), en terme de temps de calcul, est très important, notamment lorsque ces opérations sont appliquées sur une grosse quantité de données. Donc, il est indispensable d'améliorer la performance de ces requêtes en exploitant les différentes techniques d'optimisation.

[1] `http://pro.01net.com/editorial/546972/sybase-souvre-a-larchitecture-mapreduce/`
[2] `http://social.technet.microsoft.com`

	SGBDR	**MapReduce**
Taille de données	Gigaoctet	Petaoctet
Accès	Intéractive et par lot	Par lot
Mise à jour	Lire et écrire plusieurs fois	Écrire une fois et lire plusieurs fois
Modélisation	Exigent un schéma de données relationnel	N'exige aucun schéma de données préalable
Type de données	Fonctionnent sur des données bien structurées et normalisées et ne manipulent pas les types de données complexes (tableaux, ...)	Fonctionne sur n'importe quel type de données
Performance	Fournit la capacité de création des index et l'analyse partiale de données	Nécessite un scan complet de données d'entrée
Langage d'interrogation	SQL avec la capacité de créer des UDFs (Java, C++) ou autres programmes	Implémenté en Java/C++ et fournit la possibilité de créer des programmes par des multitudes langage (Java, C++, Python, Hive, Pig, ...)
Tolérance aux pannes	Tolérance aux pannes pour les transactions et le traitement des requêtes	Tolérance aux pannes seulement pour le traitement des jobs
	Une panne implique la ré-exécution de la requête	Continué à travailler sans interruption et ré-exécuter seulement les taches interrompues
Coût	Les meilleurs SGBDs sont commercialisés et nécessitent un matériel spécial trop cher	Open source (Hadoop), fonctionne sur des machines de commodité

TAB. 3.1 – Différences entre MapReduce et SGBD

3.8.1 Classification des techniques d'optimisation

Plusieurs structures d'optimisation ont été proposées pour améliorer la performance des requêtes OLAP. Ces structures d'optimisation peuvent être groupées en deux classes [14] :

3.8.1.1 Techniques redondantes

Elles regroupent la fragmentation verticale, les index et les vues matérialisées. Ces techniques sont redondantes du fait qu'elles nécessitent un espace de stockage ou une duplication de données.

- ◇ **Index :** Les index sont des structures physiques permettant un accès direct aux données. Les index exigent une contrainte d'espace de stockage, mais ils sont efficaces à la réduction du temps d'exécution des requêtes. Parmi les index couramment utilisés, on trouve : Index B-arbre, Index de projection, Index de jointure, Index binaire (bitmap), Index de jointure en étoile, etc.
- ◇ **Vues matérialisées :** Sont des résultats stockés des requêtes très complexes ou très fréquentes. Les vues matérialisées permettent de donner un temps de réponse rapide pour des requêtes très complexes, mais souffre d'un problème majeur "problème de maintenance".
- ◇ **Fragmentation verticale (FV) :** La FV découpe une table en sous tables par projection sur des attributs différents permettant de sélectionner les colonnes composantes chaque fragment. Les sous tables générées par la FV doivent contenir un attribut commun (la clé de la relation initiale). La relation initiale recomposée par jointure des fragments.

3.8.1.2 Techniques non redondantes

Elles regroupent la fragmentation horizontale et le traitement parallèle. Ces techniques sont non redondantes du fait qu'elles ne dupliquent pas les données et ne nécessitent aucun espace de stockage, mais elles nécessitent un partitionnement de données.

- ◇ **Fragmentation horizontale (FH) :** la FH découpe une table en sous tables par utilisation de prédicats permettant de sélectionner les lignes appartenant à chaque fragment. La reconstruction de la relation initiale se fait par union des fragments. Une FH valide doit respecter trois propriétés : la complétude, la reconstruction et la disjonction. La complétude élimine la perte de la table initiale à partir de ses fragments et la disjonction garantie la non duplication des données.

3.8.2 Problème de sélection des techniques d'optimisation dans un environnement traditionnel

Durant la conception physique des entrepôts de données, l'administrateur doit choisir un ensemble de structures physiques d'optimisation (Index, VMs, ...) afin de minimiser le

temps de réponse des requêtes décisionnelles. Pour chaque objet d'optimisation, plusieurs configurations sont possibles. La configuration choisie doit être optimale ou proche de l'optimale. Donc, de manière générale, nous pouvons définir le problème de sélection des structures d'optimisation comme suit :

Définition 3.8.1. Le problème de sélection des structures physiques d'optimisation consiste à construire une configuration optimale de structures physiques qui permet d'optimiser le coût d'exécution d'une charge de requêtes sous certaines contraintes (espace de stockage disponible, coût de maintenance, ...).

Dans la littérature, les VMs ont montré leur efficacité et ils ont largement utilisé dans le context de base et entrepôt de données. Formellement, le problème de sélection des VMs est défini comme suit :

Définition 3.8.2. [4] Soit $V = \{v_1, v_2, ..., v_n\}$ un ensemble des vues candidates, $Q = \{q_1, q_2, ..., q_m\}$ l'ensemble des requêtes de la charge et S la taille de l'espace disque disponible pour stocker les vues à sélectionner, alors il faut trouver une configuration de vues matérialisées $Config$ telle que :

- ◇ le coût d'exécution C des requêtes de la charge soit minimal, c'est-à-dire :

$$C_{Config(Q)} = Min(C_{V(Q)});$$

- ◇ L'espace de stockage des vues de $Config$ ne dépasse pas S, c'est-à-dire :

$$\sum Taille(v_i) \leq S.$$

3.8.3 Définition de problème de sélection des techniques d'optimisation dans un environnement Cloud Computing

Le stockage illimité et l'utilisation des ressources (stockage, traitement, ...) à la demande selon un modèle de paiement sont parmi les caractéristiques typiques d'un environnement Cloud Computing. Durant la sélection des structures d'optimisation dans tel environnement, ces deux caractéristiques écartent la contrainte d'espace de stockage et la remplacent par la contrainte de budget. Bien que l'élargissement de l'espace de stockage alloué aux structures d'optimisation peut réduire le temps de réponse des requêtes décisionnelles et par conséquent le coût de paiement associé au traitement des requêtes, il engendre une influence négative sur le coût de paiement de stockage des structures d'optimisation et par conséquent sur le coût global de paiement. À cet effet, nous pouvons définir le problème de sélection des structures d'optimisation comme suit :

Définition 3.8.3. Le problème de sélection des structures physiques d'optimisation dans un environnement Cloud Computing consiste à construire une configuration optimale de sorte que le temps de traitement des requêtes soit minimisé et/ou le coût de paiement global (lié au traitement et au stockage des structures d'optimisation) soit minimisé sous une contrainte de budget.

Formellement, nous pouvons définir, comme exemple d'illustration le problème de sélection des VMs dans le Cloud Computing comme suit :

Définition 3.8.4. Soit $V = \{v_1, v_2, ..., v_n\}$ un ensemble de vues candidates, $Q = \{q_1, q_2, ..., q_m\}$ l'ensemble des requêtes de la charge et B le budget destiné au traitement des requêtes sur le Cloud. Alors, le problème de sélection des VMs dans le Cloud consiste à trouver une configuration de vues matérialisées $Config$ telle que :

- ◇ Le coût d'exécution C en terme de temps de calcul ou/et le coût de traitement en terme de coût de paiement C' associé au calcul des requêtes de la charge soit minimal, c'est-à-dire :

$$C_{Config(Q)} = Min(C_{V(Q)}),$$

et par conséquent :

$$C'_{Config(Q)} = Min(C'_{V(Q)}).$$

- ◇ La somme du coût de paiement de traitement des requêtes en présence des VMs, de l'espace de stockage C_s et de maintenance C_m de l'ensemble des vues de $Config$ soit inférieure à B. c'est-à-dire :

$$\sum C_p(v_i) + \sum C_s(V_i) + \sum C_m(v_i) \leq B,$$

en d'autre terme, l'ensemble des vues de $Config$ minimise le temps de réponse sans violer la contrainte B.

3.8.4 Démarche générale de la résolution de problème de la sélection des structures d'optimisation

La résolution de problème de la sélection des structures d'optimisation suit généralement la démarche décrite dans la figure 3.3. Cette démarche se compose de trois étapes. La première étape consiste à construire des objets candidats à partir de la charge de requêtes. Dans la deuxième étape, un élagage de l'espace de recherche est réalisé pour diminuer le nombre des objets candidats par des algorithmes simples (gloutons) ou par des heuristiques. Ces algorithmes s'appuient généralement sur des modèles de coût pour quantifier et évaluer les bénéfices des objets candidats afin de ne garder que les objets pertinents. La troisième étape consiste à implémenter la configuration optimale ou proche de l'optimale obtenue à l'étape précédente.

3.8.5 Modèle de coût

Les modèles de coût jouent un rôle primordial dans l'évaluation correcte des performances de la conception physique du schéma de l'entrepôt de données. Ils sont nécessaires pour mesurer les coûts et les bénéfices des structures d'optimisation afin de choisir la configuration optimale des structures physiques d'optimisation. Le modèle de coût est défini comme suit :

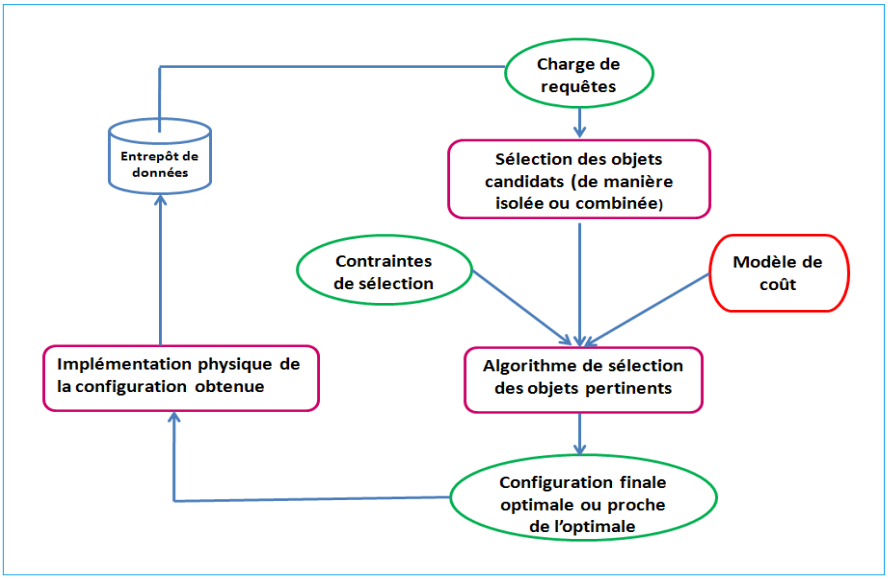


FIG. 3.3 – Démarche générale de sélection des structures d'optimisation

Définition 3.8.5. Un modèle de coût est un ensemble de formules permettant d'estimer le coût d'un plan d'exécution [30]. Le coût est estimé en exploitant des statistiques sur le système et des statistiques sur les données de la base, telles que : la taille d'une relation, la cardinalité d'un attribut. Le modèle de coût est constitué de plusieurs composants, qui sont les métriques qui vont aider le concepteur (l'optimiseur) à choisir le meilleur plan d'exécution.

La quantification de la qualité de la configuration générée est effectuée de deux manières : (a) Par l'utilisation d'un modèle de coût mathématique, ou (b) Par un appel à l'optimiseur de requêtes du SGBD.

3.8.5.1 Modèle de coût mathématique

Il se compose d'un ensemble de formules mathématiques qui sont élaborées dans le but d'évaluer le coût d'exécution des requêtes en présence ou non des structures d'optimisation. Ces formules s'appuient sur des hypothèses simplificatrices pour simplifier les estimations mathématiques des coût et un ensemble de paramètres et des statistiques sur les données, la charge de requêtes et des informations sur les nœuds de calcul. Le coût estimé dépend aussi de la méthode d'accès aux données.

3.8.5.2 Optimiseur de requêtes du SGBD

L'optimiseur de requêtes est l'un des composants essentiels d'un SGBD qui transforme une requête SQL en un plan d'exécution. Les optimiseurs sont classés en deux catégories [34] : (a) Rule based optimiser qui détermine le plan d'exécution à travers des règles strictes sans tenir compte des statistiques sur les données et des coûts d'accès aux données, et (b) cost based optimizer qui estime le coût d'exécution de chaque plan d'exécution possible à

partir des statistiques sur les données, et ensuite il choisit le plan dont le coût estimé est le plus faible.

3.8.5.3 Avantages et inconvénients

L'utilisation d'un optimiseur de requête durant la sélection des structures d'optimisation de l'entrepôt de données permet d'avoir un calcul de coût plus fiable et précis, mais rend la sélection dépendante d'un SGBD donné, et les appels fréquents de l'optimiseur peuvent dégrader les performances [14, 24]. D'un autre côté, l'utilisation d'un modèle de coût mathématique a l'avantage de la rapidité et d'être simple à mettre en oeuvre et indépendant du SGBD utilisé [24]. L'inconvénient majeur d'un modèle de coût analytique réside dans les hypothèses simplificatrices qui peuvent diverger le coût obtenu de la réalité (un coût surestimé ou sous-estimé).

3.8.5.4 Modèles de coût analytiques de l'environnement traditionnel VS Cloud Computing

Dans un environnement traditionnel, de multiples modèles de coût ont été proposés et utilisés durant la sélection des techniques d'optimisation dans le context des entrepôts de données [4, 5, 6, 7, 9, 10, 11, 12, 15, 20, 34, 35, 78]. Ces modèles de coût ont pour but de diriger les algorithmes de sélection des structures d'optimisation et d'évaluer et quantifier les bénéfices apportés par ces structures. Cependant, les modèles de coût analytiques proposés, même ceux qui traitent les entrepôts de données parallèles [11, 12], ne sont pas adaptés pour être utilisés dans un environnement Cloud Computing, du fait qu'ils ne prennent pas en compte les caractéristiques de cet environnement, notamment le modèle de paiement et les méthodes d'accès aux données utilisées.

Dans le Cloud, à notre connaissance un seul modèle de coût a été proposé [54] très récemment. Nguyen et al. [54] ont prouvé que l'utilisation des VMs sur le Cloud permet de diminuer le coût de traitement des requêtes de manière efficace et ils ont proposé un modèle de coût théorique pour la gestion de données sur le Cloud, par l'utilisation des VMS pour optimiser la performance des requêtes. Toutefois, malgré que ce modèle de coût traite l'entreposage de données sur le Cloud et prend en considération le modèle de paiement, il n'aborde pas la manière d'évaluation des requêtes ainsi que les caractéristiques physiques des nœuds de calcul, qui ont un impact significatif sur le coût de traitement des requêtes. Ainsi, aucune validation sur un système réel et sur des données concret n'a été proposée.

3.9 Algorithmes d'accès aux données dans un environnement MapReduce

La manière d'évaluation des requêtes dans le contexte des bases et entrepôts de données classiques est différente de celle utilisée dans un système MapReduce. Les méthodes classiques

de réalisation des jointures [52] ne sont plus utilisées, telles que la jointure par hachage, boucle imbriqué et trie fusion. Les systèmes à base de MapReduce, tels que Hive [72], Hadoop [37] et Pig [55] s'appuient sur des nouveaux algorithmes d'évaluation des requêtes. Dans cette section, nous présentons une description détaillée de ces algorithmes.

3.9.1 Opérations d'algèbre relationnelle utilisant MapReduce

Les opérations d'algèbre relationnelle, telles que la sélection, la projection et le groupement, sont facilement implémentées par MapReduce. Dans [64], Rajaraman et Ullman ont présenté les différents algorithmes de calcul des opérations d'algèbre relationnelle par MapReduce. Dans cette section, nous allons faire un tour d'horizon de ces algorithmes.

3.9.1.1 Selection

L'implémentation de la sélection, notée $\sigma_C(R)$, par MapReduce est très simple. Une opération de sélection est peut être réalisée en Map-only ou en Reduce-only.

- ◇ **Fonction Map :** Pour chaque tuple t, la fonction Map teste si la condition C est satisfaite. Si c'est le cas, elle produit le couple (t, t), t désigne à la fois la clé et la valeur.
- ◇ F**onction Reduce :** Renvoie chaque couple clé-valeur à la sortie.

3.9.1.2 Projection

La projection, notée $\Pi_S(R)$, est réalisée de la même manière que la sélection, et du fait que la projection peut engendrer des duplications des tuples, le rôle de la fonction Reduce est l'élimination des doublons.

- ◇ **Fonction Map :** Pour chaque tuple dans R, la fonction Map construit le tuple t' par l'élimination des attributs qui ne sont pas dans S. Ensuite, pour chaque tuple, renvoyé le couple clé-valeur (t', t').
  La fonction *Combiner* associée à chaque Map peut éliminer les duplications produites localement et la fonction Reduce élimine tout simplement les duplications produites par des taches Map différentes.
- ◇ **Fonction Reduce :** Transforme la liste des valeurs associées à une clé en une seule valeur, du fait que chaque clé peut y avoir plusieurs valeurs associées $(t', [t', t', ..., t'])$.

3.9.1.3 Union

L'union de deux relations R et S, avec R et S ont le même schéma, est réalisé comme suit :

- ◇ **Fonction Map :** Transforme chaque tuple t en un couple clé valeur (t, t).
- ◇ **Fonction Reduce :** Pour chaque clé t est associée une ou deux valeurs. Dans les deux cas, la fonction Reduce renvoie (t, t).

3.9.1.4 Intersection

Pour calculer l'intersection entre deux ensembles de données, la même fonction Map est utilisée. Cependant, la fonction Reduce doit produire un tuple seulement si les deux relations ont le tuple.

- ◇ **Fonction Map :** Transforme chaque tuple t en un couple clé valeur (t, t).
- ◇ **Fonction Reduce :** Si une clé a une liste de valeur $([t, t])$, alors la fonction Reduce produit (t, t). Dans d'autres cas, produit $(t, NULL)$. Une seule valeur associée à une clé, implique que cette valeur appartient à une seule relation. La valeur $NULL$ est utilisée pour indiquer si l'ensemble d'intersection est vide.

3.9.1.5 Agrégation & Groupement

Étant donnée une relation $R(a, b, c)$, l'opération du groupement et l'agrégation de la relation R, notée $\gamma_a, \theta_{(b)}(R)$, sous MapReduce est réalisée comme suit :

- ◇ **Fonction Map :** Pour chaque tuple (a, b, c) la fonction Map produit le couple clé-valeur (a, b). S'il y a plusieurs attributs de groupement, la clé est la liste des valeurs de tous ces attributs.
- ◇ **Fonction Reduce :** Pour chaque clé a, la fonction Reduce applique l'opérateur d'agrégation sur la liste des valeurs associées $[b_1, b_2, ..., b_n]$ et renvoie le couple (a, x), où x est le résultat de l'agrégation (SUM, par exemple).

3.9.1.6 Calcul de la jointure naturelle par MapReduce

3.9.1.6.1 Jointure de deux ensembles de données (tow-way join) : L'équi-jointure (tow-way join) entre deux ensembles de données $R(a, b)$ et $S(b, c)$ est la combinaison des tuples de R et S, tel que $R.b = S.b$, avec b est la clé de la jointure. Sous MapReduce, la jointure entre R et S est calculée par un seul job MapReduce. L'algorithme de calcul de la jointure de deux ensembles de données ($R(a, b) \bowtie S(b, c)$) par MapReduce est présenté par la figure 3.4.

- ◇ **Fonction Map :** Pour chaque tuple (a, b) de R, la fonction Map produit un couple clé-valeur $(b, (R, a))$ et pour chaque tuple (b, c) de S, elle produit un couple clé-valeur $(b, (S, c))$. Autrement dit, une collection de processus Map transforme chaque tuple des deux relations en un couple clé-valeur dont la clé est la clé de la jointure et la valeur associée est le reste du tuple. Ensuite, étiqueté chaque tuple par le nom de la relation pour que la fonction Reduce puisse distinguer la source de données de chaque tuple.
- ◇ **Fonction Reduce :** Le rôle de la fonction Reduce est de combiner les tuples de R et S ayant la même valeur de b. Ainsi, les tuples qui ont la même valeur de la clé doivent être envoyé au même processus Reduce par le biais d'une fonction de hachage. Les processus Reduce écrivent les résultats de la jointure (les tuples (a, b, c)) dans un seul fichier de sortie.

Map(k, v)
Input k is a key.
v is a record of relation R or S.

1: **if** (v is a record of relation R) **then**
2: Turn input tuple (a, b) into key-value pair $(b, (a, R))$.
3: **endif**
4: **if** (v is a record of relation S) **then**
5: Turn input tuple (b, c) into key-value pair $(b, (c, S))$.
6: **endif**
7: Emit $(b, (a, R))$ or $(b, (c, S))$.

Reduce(k, v)
Input k is a key (join key).
v is a set of records that have the same join key.

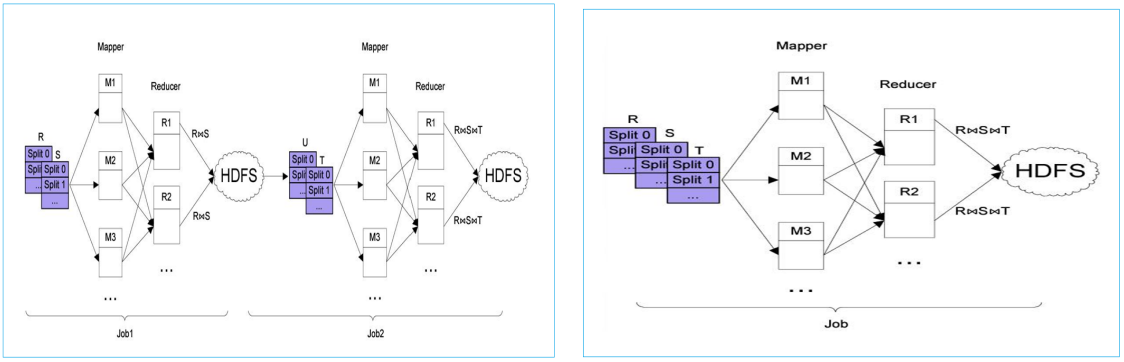
1: Match all $(b, (a, R))$ with all $(b, (c, S))$ and make an output (a, b, c).
2: Emit (a, b, c).

FIG. 3.4 – Algorithme de jointure de deux ensembles de données par MapReduce [38].

3.9.1.6.2 Jointure de plus de deux ensembles de données (multi-way join) : La jointure multi-way implique la jointure de plus de deux ensembles de données. Cette jointure est calculée par une séquence de jointure tow-way [3]. Supposons la jointure de trois relations R, S et T ($R(a,b) \bowtie S(b,c) \bowtie T(c,d)$) avec des agrégations sur les résultats de la jointure. La plupart des systèmes basés sur MapReduce transforment cette requête en quatre jobs MapReduce [43]. Le premier job calcule la jointure de R et S et écrit le résultat U dans le système de fichier. Le deuxième job calcule la jointure de T et U et produit V qui sera aussi écrit dans le système de fichier (voir la figure 3.5(a)). Le troisième job calcule des agrégations sur V et si plus d'un Reducer sont utilisés dans l'étape trois, un quatrième job est utilisé pour fusionner les résultats et écrire les résultats sous le système de fichier. Durant l'étape une et deux de la jointure, l'algorithme de la figure 3.4 est utilisé. Une autre manière de la jointure permet de joindre ces trois relations par un seul job MapReduce (voir la figure 3.5(b)). Les processus Map envoient chaque tuple de R et T à plusieurs processus Reduce différents, tandis que chaque tuple de S est envoyé à un seul processus Reduce [3].

3.9.2 Stratégies de jointure

On fait référence par stratégie de jointure à la manière de transformation d'un opérateur de jointure en un plan d'exécution physique. Blanas et al. [13] ont présenté et discuté quatre stratégies de jointures par MapReduce :



(a) Jointure de trois relations par MapReduce (b) Jointure de trois relations en un seul job MapReduce

FIG. 3.5 – Exemple de jointure de plus de deux relations sous MapReduce

3.9.2.1 Jointure par répartition

La jointure par répartition est la stratégie de jointure la plus utilisée dans le framework MapReduce. Elle est similaire à la jointure tri-fusion partitionnée dans les SGBDs parallèles. Cette jointure est implémentée par un seul job MapReduce. Chaque tache Map s'exécute sur un *Split* de l'un des deux relations de la jointure (R ou S). Dans cette phase, chaque tuple est étiqueté par le nom de sa relation d'origine et transformé en un couple clé-valeur. Ensuite, les enregistrements de chaque clé de la jointure sont groupés et envoyés au Reducer. Dans la phase Reduce, la fonction Reduce sépare les enregistrements d'entrée en deux ensembles selon la source de données et effectue le produit cartésien (cross-product) entre les enregistrements des deux ensembles. La figure 3.6 illustre un exemple d'une jointure par repartition entre deux relations L et R.

3.9.2.2 Jointure par diffusion

La plus petite table est diffusée dans son intégralité à chaque nœud, pour éviter le tri des deux tables et surtout le transfert de la plus grande table sous réseau. Cette petite table est peut être simplement répliqué sur tous les nœuds. Ensuite la jointure est effectuée localement. Cette jointure est effectuée en Map uniquement.

3.9.2.3 Semi-jointure

Ce type de jointure est exploitée pour minimiser la quantité de données transférées sous réseau. Les enregistrements de la plus large relation qui ne sont pas référenciés par la jointure ne seront pas transférés à travers le réseau. Cette jointure se déroule en trois phases, chacune correspond à un job MapReduce :

- ◇ **Phase 1 :** Correspond à un job MapReduce complet. La fonction Map extrait les clés de jointure uniques dans la première relation (supposée la plus large). Ensuite, la fonction

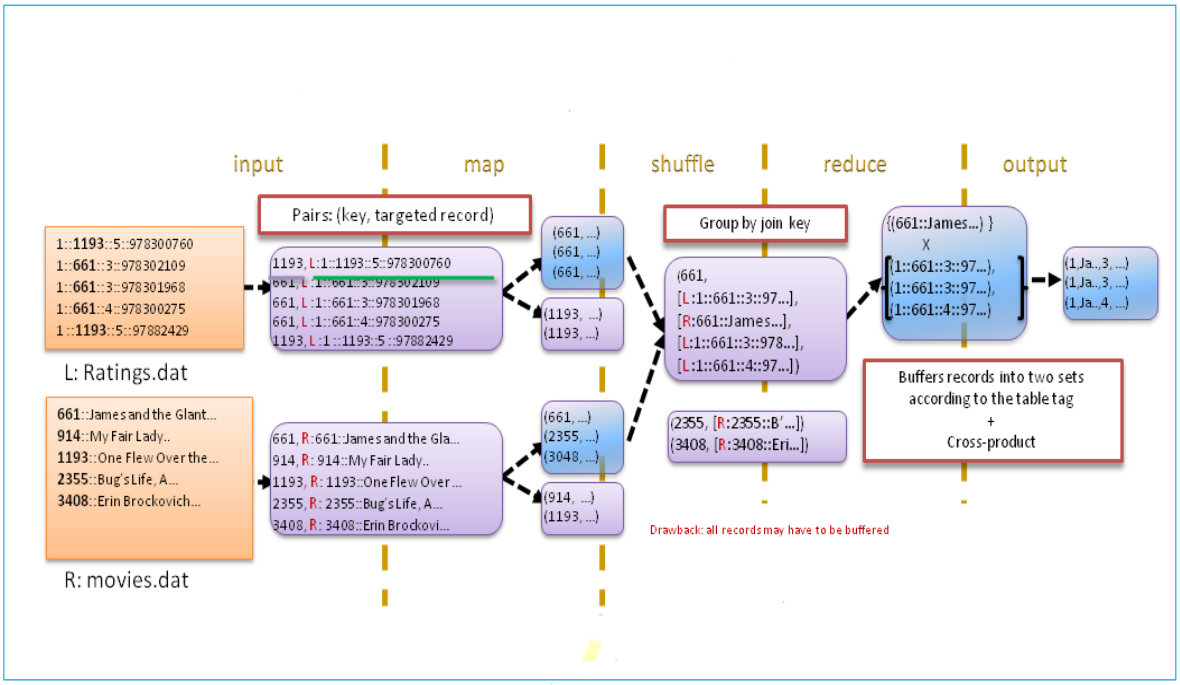


FIG. 3.6 – Exemple de jointure par répartition de deux relations

Reduce fusionne toutes les clés uniques dans un seul fichier.

◇ **Phase 2 :** S'exécute en un job Map-only et similaire à la jointure par diffusion. La fonction Map extrait chaque enregistrement dans la deuxième relation (supposée la plus petite) dont sa clé se trouve dans le fichier des clés uniques construit dans la phase 1. La sortie de cette phase est un fichier pour chaque *split* qui contient des enregistrements filtrés de la deuxième relation.

◇ **Phase 3 :** Utilisé la jointure par diffusion pour joindre les fichiers construits dans la phase2 et la première relation.

3.9.2.4 Semi-jointure par-split

Cette jointure se déroule en trois phases :

◇ **Phase 1 :** S'exécute en un job Map-only. Cette phase génère les clés uniques de jointure dans la première relation et les met dans des fichiers DFS (un fichier par *split*).

◇ **Phase 2 :** S'exécute en un job MapReduce complet. Cette phase utilise les fichiers générés dans la phase1 pour filtrer la deuxième relation.

◇ **Phase 3 :** S'exécute en un job Map-only. Cette phase effectue la jointure entre les *splits* de la première relation et les enregistrement filtrés de la deuxième relation.

3.9.3 Implémentation Hadoop de la jointure tow-way

3.9.3.1 Jointure côté Reduce (Reduce-side join)

Comme son nom l'indique, la jointure est réalisée durant la phase Reduce du framework MapReduce. La jointure Reduce-side [76] est implémentée en un seul job MapReduce complet :

- ◇ **Phase Map :** La fonction Map lit un seul tuple à un moment des deux ensembles de données à partir de HDFS. La valeur de l'attribut de la jointure est extraite comme la clé de la fonction Map et le reste du tuple est extrait comme la valeur associée à cette clé. La relation d'origine est identifiée et chaque clé et chaque valeur sont marquées par leur relation d'origine (voir la figure 3.7.a), ensuite, les tuples lus sont partitionnés et envoyés aux Reducers. Le partitionnement est basé sur la clé de la jointure, de sorte que les tuples des deux ensembles de données ayant la même clé sont forcément envoyés au même Reducer.
- ◇ **Phase Reduce :** Les sorties de Map ayant la même clé sont récupérées par le même Reducer. Chaque Reducer reçoit un ensemble d'enregistrements de plusieurs Mapper (voir la figure 3.7.b). Ces enregistrements seront triés et groupés avant de les utilisés par la fonction Reduce. Le tri est réalisé sur la clé de la jointure et un tri secondaire réalisé sur la balise. Les enregistrements ayant la même clé sont regroupés ensemble. Ensuite, la fonction Reduce calcule le produit cartésien de chaque sous ensemble.

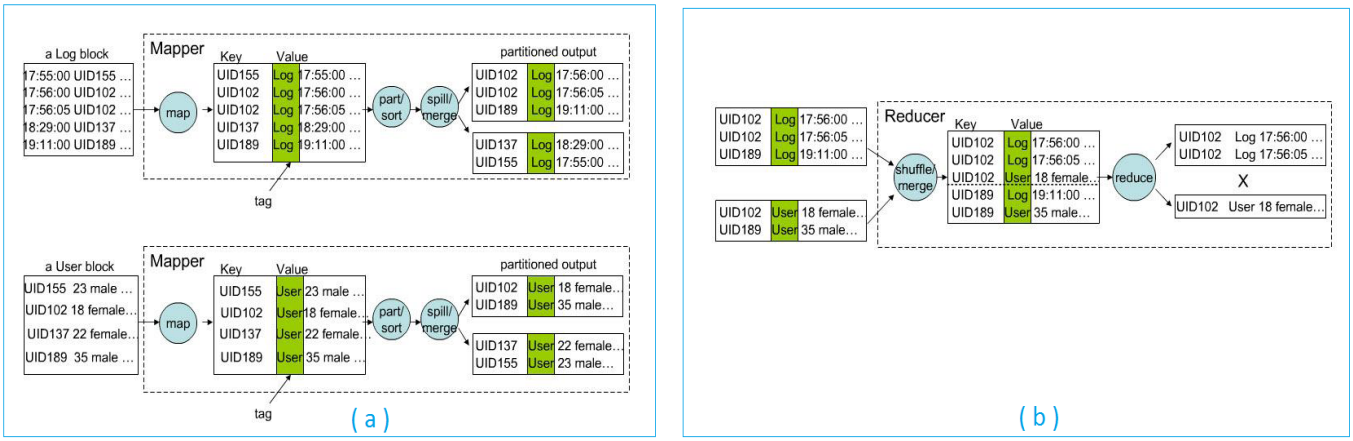


FIG. 3.7 – Jointure de la relation Log et User

3.9.3.2 Jointure côté Map (Map-side join)

Hadoop offre une autre manière de jointure de deux ensembles de données : *Map-side join*. Cette jointure est une amélioration de la jointure précédente [46] par l'élimination de la phase Reduce et le transfert de données sur le réseau entre les taches Map et les taches Reduce. Ce type de jointure peut être utilisé pour joindre les résultats de plusieurs jobs

précédents qui ont le même nombre de Reducer, les mêmes clés et les fichiers sorties qui ne sont pas fractionnables (plus petit qu'un bloc HDFS par exemple) [76], du fait que la jointure Map-side exige que les données d'entrée soient partitionnées et triées de la même manière [76]. Tous les ensembles de données doivent être partitionnés par le même partitioner et triés par le même comparator, ainsi que le nombre de partitions doit être le même pour chaque ensemble de données [75].

Dans [45], Lin et Dyer ont considéré un exemple avec deux ensembles de données S et T, ces deux ensembles sont divisés en dix fichiers, partitionnés de la même façon par la clé de la jointure et dans chaque fichier les tuples sont triés par clés de jointure. Dans ce cas, il ne reste que de fusionner le premier fichier de S avec le premier fichier de T, le second fichier de S avec le second fichier de T, etc. Cela, peut être réalisée en parallèle par l'initialisation de dix taches Map. Chaque tache Map doit récupérer deux partitions, une pour chaque relation, cela presque toujours implique une lecture non locale [45].

3.9.4 Implémentation de la jointure multi-way

Dans un environnement MapReduce, la jointure de plusieurs tables peut être optimisée en un seul job MapReduce. Par exemple, Hive transforme une requête de jointure de plusieurs tables en un seul job MapReduce, si pour chaque table le même attribut est utilisé dans les clauses de la jointure. Cependant, si les attributs de la jointure ne sont pas les mêmes, la requête sera transformée en plusieurs jobs MapReduce.

3.9.5 Jointure en étoile

La requête de jointure en étoile est une requête analytique populaire dans les entrepôts de données. La jointure en étoile comporte souvent des jointures multiples entre la table des faits et les tables de dimension. Généralement, la table de faits est très grande, tandis que les tables de dimension sont plus petites. Sous MapReduce, selon le coût de communication, la jointure de la table de fait et les tables de dimension par une jointure multiway est presque certain d'être plus efficace que de joindre les relations en paire (cascade des jointures binaires) [64].

Sous les systèmes basés sur Hadoop, tels que Hive, l'exécution d'une opération de jointure en étoile de n relations nécessite $(n-1)$ ou $2*(n-1)$ phases MapReduces dont chaque phase correspond à un job MapReduce, du fait que ces systèmes peuvent traiter l'opération de jointure seulement pour deux relations [38]. La jointure entre la table des faits et n tables de dimension est peut être effectuée selon le plan d'exécution présenté par la figure 3.8. Le premier job calcule la jointure entre F et D_1, le deuxième job calcule la jointure de D_2 et le résultat de la première phase, etc. Une manière d'optimisation de la taille des résultats intermédiaires et par conséquent le coût de transfert de données sous réseau est d'ordonner l'exécution des phases dans un ordre croissant des sélectivités des tables de dimension.

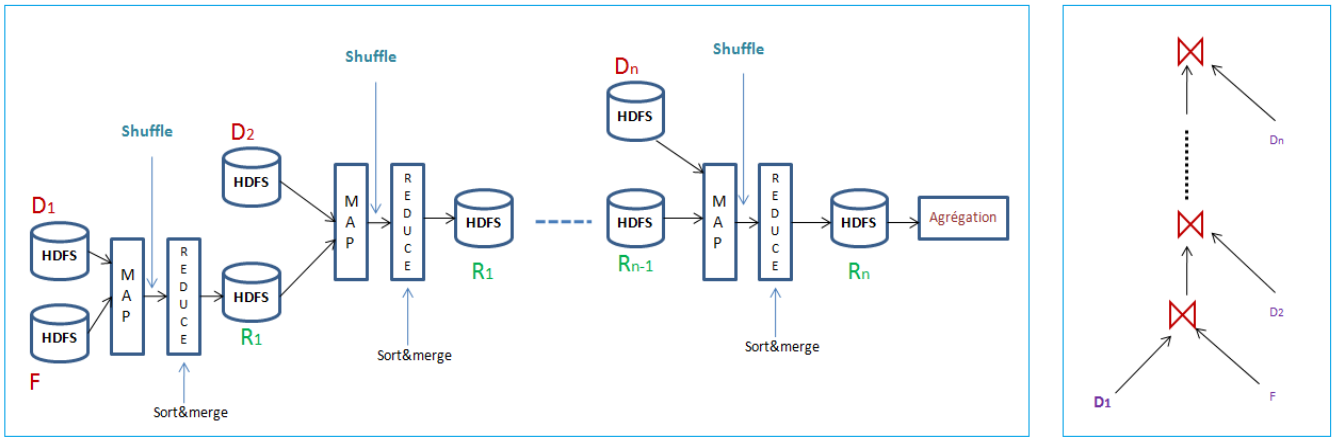


FIG. 3.8 – Plan d'exécution de la jointure en étoile de la table des fais et n tables de dimension

3.9.6 Correspondences des requêtes SQL sous MapReduce

Une requête SQL correspond à un programme MapReduce qui se compose d'un ou plusieurs jobs MapReduce. Considérons les trois relations suivantes : *Documents*, *Rankings* et $UserVisits$. Le schéma de chaque relation est présenté par la figure 3.9. Dans ce qui suit, nous présentons l'exécution de trois requêtes SQL, q1, q2 et q3, sous Hadoop (plus de détail voir [59]).

Exemple 3.9.1. Soit la requête SQL suivante

Requête $q1$:

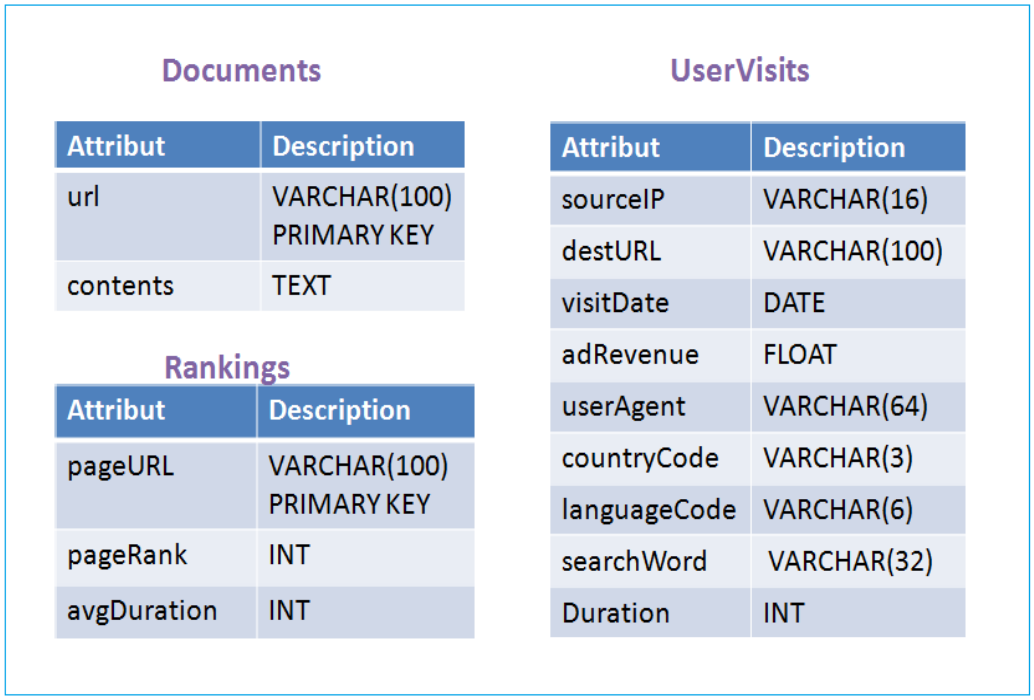
```
SELECT pageURL, pageRank
FROM Rankings WHERE pageRank > X ;
```

Le programme MapReduce d'exécution de $q1$ utilise seulement une fonction Map qui transforme les valeurs d'entrées en des couples clés-valeur, dont la clé et la valeur sont $pageURL$ et $pageRank$ et renvoie comme sortie les couples dont la valeur de $pageRank$ supérieur à X . Chaque $pageURL$ est unique dans la table *Rankings*. Cela implique que cette tache ne nécessite pas une phase d'élimination des doubles.

Exemple 3.9.2. Soit la requête q2 suivante :

Requête $q2$:

```
SELECT sourceIP, SUM(adRevenue)
FROM UserVisits GROUP BY sourceIP ;
```



Documents

Attribut	Description
url	VARCHAR(100) PRIMARY KEY
contents	TEXT

Rankings

Attribut	Description
pageURL	VARCHAR(100) PRIMARY KEY
pageRank	INT
avgDuration	INT

UserVisits

Attribut	Description
sourceIP	VARCHAR(16)
destURL	VARCHAR(100)
visitDate	DATE
adRevenue	FLOAT
userAgent	VARCHAR(64)
countryCode	VARCHAR(3)
languageCode	VARCHAR(6)
searchWord	VARCHAR(32)
Duration	INT

FIG. 3.9 – Schéma des relations *Documents*, *Rankings* et $UserVisits$

L'exécution de $q2$ nécessite un programme MapReduce qui se compose de deux fonctions Map et Reduce. La fonction Map transforme les valeurs d'entrées en des couples clés-valeur et faire sortir les champs $sourceIP$ et $adRevenue$ comme clé-valeur. La fonction Reduce tout simplement fait la somme de toutes les valeurs $adRevenue$ pour chaque $sourceIP$ et renvoie le revenu total.

Exemple 3.9.3. Soit la requête q3 suivante :

Requête $q3$

```
SELECT INTO Temp sourceIP, AVG(pageRank) as avgPageRank, SUM(adRevenue) as
totalRevenue
FROM Rankings AS R, UserVisits AS UV
WHERE R.pageURL = UV.destURL AND UV.visitDate BETWEEN Date('2000-01-15')
AND Date('2000-01-22')
GROUP BY UV.sourceIP ;
```

Le programme MapReduce d'exécution de $q3$ se compose de trois phases. La phase (1) utilise deux fonctions Map et Reduce pour filtrer les enregistrements de $UserVisits$ et garder seulement les enregistrements satisfaisants les conditions déterminées par la requêtes et ensuite effectuer la jointure entre les enregistrements de *Rankings* et les enregistrements filtrés précédemment. La phase (2) utilise une seule fonction Reduce pour calculer le revenu total

et *pageRank* moyenne pour chaque *sourceIP* . Dans la dernière phase une seule fonction Reduce est définie pour renvoyer les enregistrements ayant le plus grand revenu total.

3.10 Pre-calcul des requêtes dans un environnement MapReduce

Dans un environnement MapReduce, le pré-calcul des requêtes est similaire aux vues matérialisées dans les SGBDs ; MapReduce est peut être considéré comme une définition des vues et les résultats calculés comme des vues matérialisées [44]. Il existe plusieurs similarités et différences entre les vues matérialisées et les vues MapReduce (VMR), qu'on peut résumer dans le tableau 3.2.

	VM	**VMR**
Similarités	Calculées d'une ou plusieurs relations	Les données sont extraites d'un ou plusieurs ensembles de données
	Stockées dans la BD	Stockées dans DFS
	Définies par l'utilisateur dans des requêtes SQL	La transformation de données est définie par les fonctions utilisateur Map et Reduce
	Rafraîchissement des vues pour obtenir les résultats des modifications des sources de données	Ré-exécution des programmes MapReduce pour obtenir des mises à jours
Différences	Existe des techniques de rafraîchissement automatique	/
	Définies sur un schéma relationnel	Générique

TAB. 3.2 – Vues matérialisées et le pré-calcul des requêtes MapReduce

Similaire aux VMs, VMR permet d'optimiser le temps d'accès aux données avec des coûts supplémentaires : coût de stockage et coût de maintenance.

3.11 Conclusion

L'immigration des entrepôts de données vers un environnement Cloud Computing peut bénéficier des avantages de celui-ci, comme il peut engendrer des nouveaux challenges. La conception physique des entrepôts de données peut être considérée parmi les principaux challenges, du fait que dans cet environnement le stockage et l'accès aux données sont effectués d'une manière différente et les algorithmes de construction d'une configuration optimale des structures d'optimisation utilisés dans un environnement traditionnel ne sont pas directement adaptés pour tel environnement.

L'adaptation ou la définition des nouveaux algorithmes du choix de la configuration optimale nécessite l'élaboration d'un nouveau modèle de coût analytique qui prend en

considération les différentes caractéristiques du Cloud, pour quantifier les différentes configurations possibles afin de choisir la meilleure configuration.

L'élaboration d'un modèle de coût analytique pour la conception physique des entrepôts de données dans le Cloud n'est pas une taches facile. Les coûts à estimer dépendent de plusieurs facteurs : les caractéristiques physiques des nœuds, le framework d'exécution des requêtes, les statistiques sur les données stockées ainsi que le modèle de paiement.

Dans le chapitre suivant, nous désirons développer un modèle de coût analytique capable d'estimer le coût d'exécution de la charge de requêtes et de quantifier le coût d'utilisation des VMs dans le Cloud en se basant sur le framework d'accès aux données MapReduce.

Chapitre 4

Modèle de coût pour la conception physique des entrepôts de données sur le Cloud

4.1 Introduction

Nous avons vu dans les chapitres précédents que les données analytiques sont bien adaptées pour le déploiement dans un environnement Cloud Computing et que les entrepôts de données peuvent bénéficier des avantages promis par ce nouveau modèle de consommation des ressources informatiques. L'émigration des entrepôts de données vers le Cloud nécessite de concevoir ou d'adapter des algorithmes de sélection des techniques d'optimisation afin de minimiser le coût de traitement des requêtes décisionnelles dans ce nouvel environnement. Dans le Cloud, on peut distinguer deux types de coût différents engendrés par le traitement des requêtes décisionnelles : le coût en termes de temps d'exécution (temps de réponse) et un coût en termes de coût de paiement.

Généralement, la sélection des techniques d'optimisation basée sur un modèle de coût analytique, qui englobe des fonctions mathématiques permettant d'estimer le coût engendré par le traitement des requêtes décisionnelles en présence (ou non) des structures d'optimisation. Cette évaluation des différents coûts permet de diriger les algorithmes de sélection des structures d'optimisation pour trouver la meilleure configuration physique de l'entrepôt de données.

Dans le présent chapitre, nous désirons élaborer un modèle de coût analytique capable d'évaluer le coût de traitement des requêtes décisionnelles sur le Cloud, en choisissant le paradigme d'exécution le plus populaire sur cet environnement, MapReduce, comme environnement d'exécution.

Dans le context des entrepôts de données, les VMs permettent de réduire de manière efficace le temps de réponse des requêtes décisionnelles, par le pré-calcul des requêtes les plus fréquemment utilisées. Ainsi, nous avons vu dans le chapitre précédent (section 3.10) qu'il existe une forte similarité entre les VMs et les VMR. À cet effet, nous avons choisi dans notre étude les VMs comme structure d'optimisation.

4.2 Description générale et utilité du modèle de coût

Étant donné un ensemble de vues candidates, le concepteur doit choisir un sous-ensemble de vue permettant une optimisation optimale du coût d'accès aux données sous un certain nombre de contraintes, coût de stockage et/ou coût de maintenance. Dans notre étude, nous supposons que ce choix s'appuie sur un modèle de coût analytique (voir la figure 4.1) et nous désirons généraliser le coût d'exécution de la charge de requêtes ainsi que le coût d'utilisation des vues matérialisées pour accéder aux données dans un environnement MapReduce. Ce coût se compose des coûts suivants : coût de stockage, coût de transfert des résultats des requêtes, le coût d'accès aux données en présence ou non des VMs, coût de maintenance et le coût de paiement.

Le modèle de coût à élaborer permet la prise en compte du parallélisme, l'environnement d'exécution, le modèle de paiement, les caractéristiques physiques des nœuds de calcul et des statistiques et des informations sur l'entrepôt de données et la charge de requêtes.

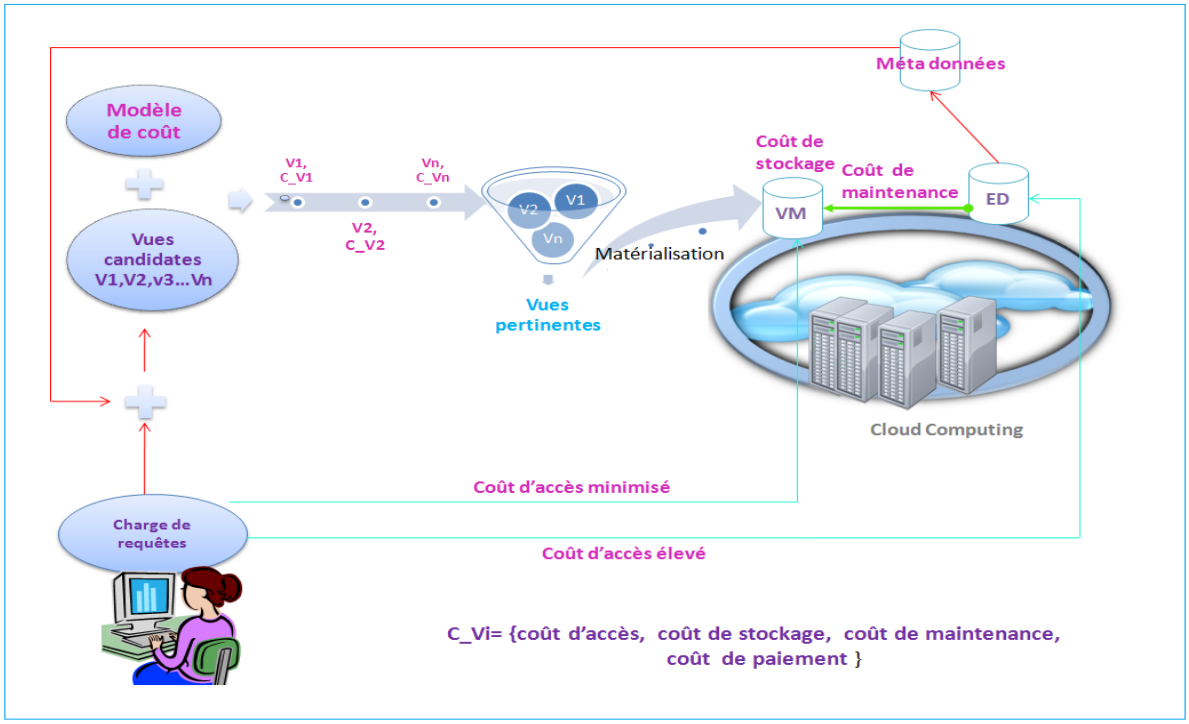


FIG. 4.1 – Utilité du modèle de coût à proposer

Ce modèle de coût peut jouer un rôle primordial durant la conception physique des entrepôts de données sur le Cloud, et plus particulièrement dans un environnement MapReduce. Il peut être utilisé pour comparer entre la matérialisation des requêtes ou l'augmentation des ressources de traitement des requêtes ou pour évaluer les différents plans d'exécution d'une requête afin de choisir le meilleur plan, ainsi que pour diriger les algorithmes de sélection des VMs dans cet environnement.

4.3 Paramètres d'estimation du coût d'exécution des requêtes

Notre étude consiste à faire estimer le coût d'exécution des requêtes OLAP et le coût d'utilisation des VMs dans la plate-forme Cloud Computing. Les différents coûts sont estimés par des formules mathématiques qui prennent en considération des caractéristiques physiques des nœuds et un certain nombre de paramètres et de statistiques sur l'entrepôt de données et la charge de requêtes, qui sont indiqués dans le tableau 4.1 et des paramètres de paiement qui sont indiqués dans le tableau 4.2.

Symbole	**Description**
M	Nombre des taches Map
R	Nombre des taches Reduce
D_i	Table de dimension i
F	Table de faits
$\|\|S\|\|$	Nombre de tuples d'une relation S
$f(S,T)$	Facteur de sélectivité de jointure entre la table S et T
N	Nombre de nœuds utilisés
D	Temps nécessaire pour transférer un octet
n	Nombre de tables de dimension
m_i	Nombre des taches Map qui s'exécutent sur le nœud i
r_i	Nombre des taches Reduce qui s'exécutent sur le nœud i
Fr_q	Fréquence d'utilisation de la requête q
$taille(a)$	Taille en octet de l'attribut a
Sel_S	Sélectivité de prédicat défini sur la relation S
$\|a\|$	Cardinalaté de l'attribut a
V	l'ensemble des VMs
K	Nombre des VMs
T_{Init}	Temps moyen d'initialisation d'un job MapReduce
T_{fin}	Temps moyen de close d'un job MapReduce
$split$	Taille en octet d'un split
$size(S)$	Taille en octet d'une relation S
$Debit$	Débit de transfert de données (MB/s)
Rd	Ratio disque
Rp	Ratio de réplication de données
B	Taille du buffer
$Fupdate$	Fréquence de mise à jour

TAB. 4.1 – Paramètres d'estimation du coût d'execution des requêtes

4.4 Hypothèses

La proposition d'un modèle de coût mathématique s'appuie généralement sur des hypothèses simplificatrices, pour simplifier l'élaboration des fonctions mathématiques des

Symbole	**Description**
P	Période de stockage
$NbrInst$	Nombre d'instances utilisées
$Prix_{Inst}$	Prix d'utilisation d'une instance par unité de temps (généralement \$/H)
$Prix_{Debit}$	Prix de transfert de données (\$/GO)
$Prix_{stockage}$	Prix de stockage de données par unité de temps (\$/GO/U)

TAB. 4.2 – Paramètres d'estimation du coût de paiement

Statistiques sur les données			
Table	**Nombre de tuple**	**Taille (MO)**	**Type de table**
lineorder	600 000 000	60000	Faits
customer	3 000 000	360	Dimension
supplier	200 000	200	Dimension
part	200000	2000	Dimension
date	3555	0.2	Dimension
Caractéristiques physiques			
Debit(MB/s)	**Rd(MB/s)**	**D**	**Pc(GHz/s)**
100	120	10^{-8}	2
Paramètres de paiement			
Fournisseur	**Stockage**	**Débit**	**Exécution**
Amazon EC2	0.24\$/(GO/mois)	0.12\$/GO	0.080\$/H
Google	0.15 \$/(GO/mois)	0.12\$/GO	0.10\$/H

TAB. 4.3 – Exemple des paramètres d'estimation des coûts

différents coûts d'exécution des requêtes. Dans notre étude, nous considérons les hypothèses suivantes :

- ◇ Les données sont uniformément distribuées et les valeurs des attributs sont indépendantes. Ces deux hypothèses sont largement utilisées dans l'estimation de temps d'exécution des requêtes dans les SGBDs traditionnels et permettent de simplifier l'élaboration des coût d'exécution. Sous l'hypothèse d'indépendance des valeurs des attributs, tous les attributs sont traités comme s'il étaient autonomes des autres ;
- ◇ Le coût CPU est négligeable par apport au coût d'E/S, du fait que le coût d'E/S est le coût dominant dan les systèmes informatiques ;
- ◇ La stratégie de jointure est la jointure par répartition (Repartition join) ;
- ◇ La taille de $tag - table$ est négligeable ;
- ◇ Tous les nœuds de calcul ont les mêmes caractéristiques physiques et la taille du buffer sur chaque nœud est suffisamment large pour conserver les données ;
- ◇ Nous considérons que la charge de requêtes englobe seulement des requêtes de jointure en étoile avec des agrégations. La structure générale des requêtes de jointure en étoile considérées est représentée par la figure 4.2.

Exemple 4.4.1. La requête suivante est un exemple d'une requête de jointure en étoile sur le schéma en étoile défini par la figure 4.3. Cette requête est extraite de l'ensemble de

```
Select  a1, a2, ..., aj, ag1(b1), , ag2(b2), ..., , agx(bx)
From    F, D1, D2, ...., Dn
Where   F.id1=D1.id and F.id2=D2.id ... and  F.idn= Dn.id
          and  p1 and  p2 and ... py
Groupby a1, a2, ....
Order by a1, ....
```

FIG. 4.2 – Structure générale d'une requête de jointure en étoile

requêtes défini sur le schéma en étoile SSBM (Star Schema Benchmark) [56] :

Exemple d'une requête de jointure en étoile

Select *c_nation*, s_*nation*, *d_year*, sum(*lo_revenue*) as reve-nue
From *customer*, *lineorder*, *supplier*, *date*
Where *lo_custkey* = *c_custkey* and *lo_suppkey* = *s_suppkey* and *lo_orderdate* = *d_datekey* and *c_region* = 'ASIA' and *s_region* = 'ASIA' and *d_year* $\geq$ 1992 and *d_year* $\leq$ 1997
Group by *c_nation*, *s_nation*, *d_year*
Order by d_year asc, revenue desc

La requête Q calcule le chiffre d'affaires total des opérations de *lineorder* dans une région donnée et durant une certaine période. La requête comporte une jointure en étoile entre les relations *customer*, *lineorder*, *supplier* et *date* suivie par un opérateur d'agrégation (SUM) sur les résultats de la requête.

4.5 Coût de traitement de la charge de requêtes

Dans cette section, nous proposons des formules analytiques permettant d'estimer le coût de traitement de la charge de requêtes en terme de temps d'exécution.

4.5.1 Coût d'exécution d'une requête de jointure en étoile

Dans un système MapReduce, une requête de jointure en étoile, entre F et n tables de dimension, s'exécute en un nombre de phases, dont chaque phase correspond à un job MapReduce. Le nombre de jobs MapReduce dépend du nombre de jointures et la présence ou non des agrégations et de tri de données. On peut distinguer trois cas de figure : (1) la requête contient seulement n opérations de jointure successive entre F et n tables de dimension (2)

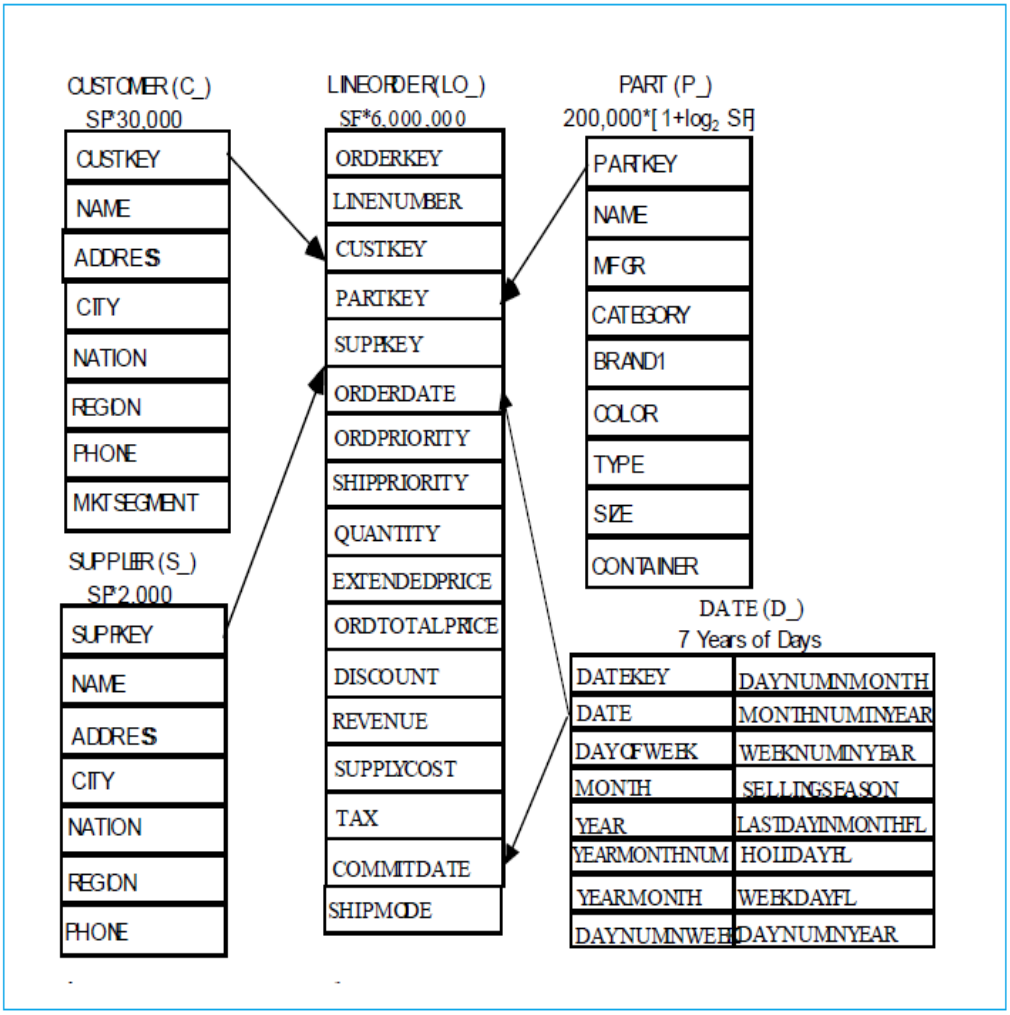


FIG. 4.3 – Exemple d'un schéma en étoile [56]

les opérations de jointure sont suivies par des opérations de groupement et des agrégations sur les résultats de la jointure (3) un tri est appliqué sur les résultats obtenus. Ce nombre peut être estimé par la formule suivante :

$$Nbr_{job}(q) = n + x, \tag{4.1}$$

avec $x = \begin{cases} 0 : \text{si la requête comporte seulement des opérations de jointure ;} \\ 1 : \text{si la requête comporte des opérations de groupement et d'agrégation ;} \\ 2 : \text{si les résultats obtenus sont triés.} \end{cases}$

par conséquence, le temps total d'exécution d'une requête de jointure en étoile est calculé par la somme des temps d'exécution des jobs constituant cette requête, qu'on peut définir par la formule suivante :

$$Texc(q) = \sum_{i=1}^{Nbr_{job}(q)} Tjob_i. \tag{4.2}$$

Exemple 4.5.1. La requête Q de l'exemple 4.4.1, s'exécute en cinq jobs MapReduce. Le premier job calcule la jointure entre la table de faits *Lineorder* et la table de dimension

customer. Le deuxième job calcule la jointure entre la deuxième table de dimension *supplier* et le résultat de la première jointure. Le troisième job calcule la jointure entre la troisième table de dimension *date* et le résultat de la deuxième jointure. Le quatrième job s'occupe de l'agrégation et du groupement des résultats et le dernier job trie les résultats obtenus.

4.5.2 Estimation du temps d'exécution d'un job MapReduce

Généralement, un job MapReduce s'exécute en deux phase : phase Map et phase Reduce. Les sorties des taches Map sont envoyées à travers le réseau vers les nœuds d'exécution des taches Reduce. À cet effet, on peut calculer le temps d'exécution d'un job MapReduce par la formule suivante :

$$Tjob_i = Tinit + Tmap_i + Tshuffle_i + Treduce_i + Tfin,\ \forall i \in [1, Nbr_{job}(q)]. \tag{4.3}$$

4.5.2.1 Coût d'initialisation et de close d'un job MapReduce

Le temps d'initialisation et le temps de close d'un job MapReduce sont liés à des taches d'initialisation et de fermeture d'un job au niveau des nœuds de traitement. Les actions d'initialisation d'un job comporte par exemple la réception de données et la création des listes des taches. La fermeture d'un job consiste à établir une connexion avec le Jobtracker dans le but d'envoyer les résultats intermédiaires, les notifications de fin du travail, etc.

Par expérimentation, Pavlo et al. [59] ont trouvé qu'un programme MapReduce prend un certain temps avant que tous les nœuds fonctionnent à pleine capacité. Sur un cluster de 100 nœuds, il faut 10 secondes à partir du moment où un travail est envoyé au JobTracker avant que la première tâche Map commence à s'exécuter, et 25 secondes jusqu'à ce que tous les nœuds du cluster commencent l'exécution du travail.

4.5.2.2 Coût coté Map

Le coût d'exécution d'une phase Map depend du nombre des taches Map, le nombre de nœuds d'exécution et la quantité de données en entrées et en sortie. Les taches Map dans des nuœds différents s'exécutent en parallèle l'une indépendamment des autres. L'équation suivante généralise le temps d'exécution d'une phase Map :

$$Tmap_i = \max_{j=1...N} m_j * [\frac{(size(inMap_i) + size(outMap_i))}{M_i * RD}],\ \forall i \in [1...Nbr_{job}(q)], \tag{4.4}$$

Souvent, le nombre des taches Map, qu'on peut définir par la formule suivante, dépend de la quantité de données en entrée (noté $inMap$) :

$$M_i = \frac{inMap_i}{split},\ \forall i \in [1, Nbr_{job}(q)], \tag{4.5}$$

Exemple 4.5.2. Soit les statistiques sur des données présentées dans le tableau 4.3. Ces données sont liées au schéma en étoile de la figure 4.3 avec $SF = 100$. Avec une taille de 128

MO d'un fichier *split*, le nombre de Map nécessaire pour traiter la jointure entre *lineordre* et *custumer* est égale à $\frac{60000}{128} + \frac{360}{128} = 471$ taches Map.

La quantité de données en entrée d'une phase Map (notée $inMap$) est calculé par les formules suivantes :

$$||inMap_1|| = ||F|| + ||D_1||, \tag{4.6}$$
$$||inMap_i|| = ||RI_{i-1}|| + ||D_i||, \ \forall i \in [2, n] \tag{4.7}$$
$$||inMap_{n+1}|| = ||RI_n||, \tag{4.8}$$
$$||inMap_{n+2}|| = ||RI_{n+1}||, \tag{4.9}$$

avec RI présente les résultats intermédiaires : RI_1 présente le résultat du premier job, RI_2 le résultat du deuxième job et ainsi de suite.

Exemple 4.5.3. Prenons la même requête Q de l'exemple 4.4.1 :

- ◇ La quantité de données en entrées de la première phase de traitement de la requête Q, $||inMap_1||= ||lineorder|| + ||customer||= 603000000$ tuples.
- ◇ Durant le deuxième job $||inMap_2||=||supplier|| + ||RI_1||$, tel que RI_1 représente le résultat de la jointure entre *lineorder* et *customer*.
- ◇ Dans la troisième phase de jointure $||inMap_3||=||date|| + ||RI_2||$, tel que RI_2 représente le résultat de la jointure entre *supplier* et RI_1.
- ◇ La quatrième phase de traitement de Q requière une quantité de données en entrées $||inMap_4||=||RI_3||$, tel que RI_3 représente le résultat de la jointure en étoile.
- ◇ Le dernier job MapReduce nécessite en entrées $||inMap_5||=||RI_4||$, tel que RI_4 représente le résultat de groupement et d'agrégation des résultats de la jointure en étoile.

La quantité de données en sortie d'une phase Map est calculée par les formules suivantes :

$$||outMap_1|| = ||F|| * \prod sel_F + ||D_1|| * \prod sel_{D_1}, \tag{4.10}$$
$$||outMap_i|| = ||RI_{i-1}|| + ||D_i|| * \prod sel_{D_i}, \ \forall i \in [2, n], \tag{4.11}$$
$$||outMap_{n+1}|| = ||RI_n||, \tag{4.12}$$
$$||outMap_{n+2}|| = ||RI_{n+1}||, \tag{4.13}$$

Exemple 4.5.4. Au cours d'exécution de la requête Q :

- ◇ $||outMap_1|| = ||lineorder|| * \frac{1}{5} + ||customer|| = 123000000$ tuples ;
- ◇ $||outMap_2|| = ||RI_1|| + ||supplier|| * \frac{1}{5}$;
- ◇ $||outMap_3|| = ||RI_2|| + ||date|| * \frac{1}{6}$;
- ◇ $||outMap_4|| = ||RI_3||$;
- ◇ $||outMap_5|| = ||RI_4||$.

Le nombre de tuples des résultats intermédiaires (RI) peut être calculer par les formules suivantes :

$$||RI_1|| = ||F|| * \prod sel_F * ||D_1|| \prod sel_{D1} * f(F, D_1), \tag{4.14}$$

$$||RI_i|| = ||RI_i - 1|| * ||D_i|| * \prod sel_{D_i} * f(F, D_i),\ \forall i \in [2, n], \tag{4.15}$$

$$||RI_{n+1}|| = Agreg(RI_n), \tag{4.16}$$

$$||RI_{n+2}|| = ||RI_{n+1}||, \tag{4.17}$$

avec :
$Agreg(RI_n)$ est une fonction qui renvoie le nombre maximum de tuples des résultats de jointure après l'opération de groupement et d'agrégation. En se basant sur les facteurs de sélectivité des prédicats et la cardinalité des attributs de la clause Group by et sous les hypothèses de la distribution uniforme de données et l'indépendance des attributs, nous définissons la fonction $Agreg(RI_n)$ comme suit :

$$Agreg(RI_n) = \prod |a_i| * sel_i, \tag{4.18}$$

avec a_i présente l'ensemble des attributs de groupement,

et si absence de la clause Group By on obtient : $Agreg(RI_n) = 1$. (4.19)

Exemple 4.5.5. Le nombre de tuples des résultats intermédiaires du clalcul de la jointure en étoile de la requête Q est calculé comme suit :

- ◇ Le premier job calcule la jointure entre *lineorder* et *customer* et produit comme résultat $RI_1 = 600\ 000\ 000 * \frac{1}{5} = 120\ 000\ 000\ tuples$
- ◇ Le deuxième job calcule la jointure entre *supplier* et RI_1 et produit comme résultat $RI_2 = 120\ 000\ 000 * \frac{1}{5} = 24\ 000\ 000\ tuples$.
- ◇ Le troisième job calcule la jointure entre *date* et RI_2 et produit comme résultat $RI_3 = 24\ 000\ 000 * \frac{6}{7} = 205\ 714\ tuples$.

Remarque 4.5.1. La taille en octet d'une relation S peut être donnée par l'équation suivante :

$$size(S) = ||S|| * \sum taille(a_j), \tag{4.20}$$

où, $\{a_j\}_j$ représente l'ensemble des attributs constituant la relation S.

4.5.2.3 Coût coté Reduce

Pour évaluer le coût d'exécution durant une phase Reduce nous supposons que la taille du buffer est suffisamment large pour contenir les données envoyées par les taches Map (aucune lecture disque n'est impliquée), et que les taches Reduce reçoivent la même quantité de données (équilibrage de la charge du travail entre les taches Reduce). Les taches Reduce dans des nœuds différents s'exécutent en parallèle, l'une indépendamment des autres. On peut

diviser le coût d'une phase Reduce en deux coûts : coût de tri (noté C_1), qu'on peut estimer en se basant sur la formule définie par Mishra et al. [52], et coût d'écriture des copies des résultats sous DFS avec un taux de réplication Rp (noté C_2). Par conséquent, le coût coté Reduce peut être estimé par les formules suivantes ($\forall i \in [1, Nbr_{job}(q)]$) :

$$C1 = \frac{size(inReduce_i)}{R_i} * \log_B(\frac{size(inReduce_i)}{R_i}), \tag{4.21}$$

$$C2 = D * Rp * \frac{size(outReduce_i)}{R_i * RD}, \tag{4.22}$$

$$Treduce_i = \max_{j=1...N} r_j * [C1 + C2], \tag{4.23}$$

tel que,$\forall i \in [1, Nbr_{job}(q)]$:

$$||inReduce_i|| = ||outMap_i||, \tag{4.24}$$

$$||outReduce_i|| = ||RI_i||. \tag{4.25}$$

4.5.2.4 Coût Shuffle

Le coût de transfert de données entre les taches Map et les taches Reduce est limité par la quantité de données transférées et la bande passante réseau. La quantité de données à transférer variée selon le stage de traitement de la requête. On peut estimer le temps de transfert de données entre les taches Map et les taches Reduce par l'équation suivante :

$$Tshuffle_i = size(outMap_i) * D, \ \forall i \in [1, Nbr_{job}(q)]. \tag{4.26}$$

Exemple 4.5.6. Avec un débit théorique de $100MB/s$, $Tshuffle_i = 23 * 10^{-9} * 10^{-8} = 3.8mn$.

4.5.3 Coût de transfert des résultats

Le temps de transfert des résultats d'une requête dépend de la taille de ces résultats et le débit de transfert, qu'on peut calculer par la fonction suivante :

$$Ttransfert(q) = \frac{size(RI_{n+x})}{Debit}. \tag{4.27}$$

4.5.4 Coût total d'exécution de la charge de requêtes

Le coût de traitement de la charge de requêtes en terme du temps d'exécution ($TotalCost(Q)$) est égale à la somme : de la somme du temps d'exécution de toutes les requêtes et du temps de transfert multiplié par la fréquence d'utilisation de chaque requête. Ce coût est donné par la formule suivante :

$$TotalCost(Q) = \sum_{q \in Q} [(Txec(q) + Ttransfert(q)) * Fr_q]. \tag{4.28}$$

4.6 Coût du paiement de traitement de la charge de requêtes

Dans cette section, en se basant sur les paramètres de paiement et les coûts définis dans la section précédente nous avons élaboré des fonctions mathématiques d'estimation de coût de paiement des requêtes.

4.6.1 Coût de paiement d'une requête

Le calcul du coût de paiement d'une requête q, $Cost_{finance}(q)$, est basé sur le temps d'exécution de la requête et les paramètres de paiement indiqués dans le tableau 4.2, ainsi que le coût de transfert des résultats de la requête. Ce coût peut être donné par la formule suivante :

$$Cost_{finance}(q) = Texec(q) * NbrInst * PrixInst + cost_{transfert}(q), \tag{4.29}$$

tel que :

$$cost_{transfert}(q) = size(RI_{n+x}) * Prix_{Debit}. \tag{4.30}$$

4.6.2 Coût de paiement total de la charge de requêtes

Le coût de traitement de la charge de requête en termes de coût de paiement ($TotalCost_{finance}(Q)$) est égale à la somme du coût de paiement de toutes les requêtes multiplié par la fréquence d'utilisation de chaque requête. Ce coût est donné par la formule suivante :

$$TotalCost_{finance}(Q) = \sum_{q \varepsilon Q} Cost_{finance}(q) * Fr_q. \tag{4.31}$$

4.7 Coût d'exécution de la charge de requêtes en présence des VMs

Le coût d'exécution de la charge de requêtes en présence des VMs se compose des coûts suivants :

- ◇ Coût de stockage de l'ensemble des VMs
- ◇ Coût d'accès aux VMs
- ◇ Coût de maintenance de l'ensemble des VMs

Ce coût peut être généraliser par la formule suivante :

$$TotalCost(Q,V) = TotalCost_{stockage}(V) + TotalCost_{acces}(Q,V) + TotalCost_{maint}(V). \tag{4.32}$$

Dans ce qui suit, nous définissons les fonctions du calcul des sous coûts indiqués ci-dessus.

4.7.1 Coût de Stockage des VMs

Le coût de stockage d'une vue v_i est proportionnel à sa taille et à la période de stockage chez le fournisseur de service ainsi que le prix de stockage par unité du temps (U : qui peut être par jour, par heure ou par mois). Ce coût peut être donné par la formule suivante :

$$Cost_{stockage}(v_i) = size(v_i) * Prix_{stockage} * P, \ \forall i \in [1, K]. \tag{4.33}$$

Exemple 4.7.1. Considérons la matérialisation dans une vue v_1 le résultat de la requête $Q2$ (définie ci-dessous) dont la taille $\backsimeq 5GO$. Si la taille de v_1 est fixe pendant une période donnée P = deux mois par exemple, le coût de paiement de stockage est de $5 * 0.1 * 2 = 1\$$ chez amazon EC2, et $5 * 0.24 * 2 = 2.4\$$ chez Google.

Requête Q2

Select *lo_orderkey*, *lo_linenumber*, *lo_custkey*, *lo_partkey*, *lo_suppkey*, *lo_orderdate*, *lo_revenue*, *c_nation*, s_*nation*, *d_year*, sum(*lo_revenue*) as reve-nue
From *customer*, *lineorder*, *supplier*, *date*
Where *lo_custkey*= *c_custkey* and *lo_suppkey* = *s_suppkey* and *lo_orderdate* = *d_datekey* and *c_region* ='ASIA' or *c_region* = 'AMERICA' and *s_region* ='ASIA' or *s_region* = 'AMERICA' and *d_year* $\geq$ 1992 and *d_year* $\leq$ 1997

Les mises à jours des relations de base peuvent impliquer des changements sur la taille des VMs. De ce fait, la taille des VMs n'est pas statique. Supposons que la taille de v_i augmente par un taux moyen de ϕ_i chaque unité du temps U. Dans ce cas, on peut estimer le coût de paiement de stockage durant une période P par la formule suivante :

$$Cost_{stockage}(v_i) = [(2 * size(v_i) + \phi_i * (P-1)) * \frac{P}{2}] * Prix_{stockage}, \ \forall i \in [1, K]. \tag{4.34}$$

Exemple 4.7.2. Supposons la vue v_1 de l'exemple précédent. Si la taille de v_1 augmente par un taux moyen de 1.2 GO par mois, le coût de paiement de stockage d'une période de trois mois est de : $[(2 * 5 + 1.2 * (3-1))3/2] * 0.10\$ = 1,86\$$ chez amazon EC2, et $[(2 * 5 + 1.2 * (3-1))3/2] * 0.24\$ = 4.464\$$ chez Google.

Le coût de stockage de l'ensemble des VMs est donné par la formule suivante :

$$TotalCost_{stockage}(V) = \sum_{i=1}^{K} Cost_{stockage}(v_i). \tag{4.35}$$

4.7.2 Coût d'extraction de données

4.7.2.1 Coût d'exécution d'une requête en présence des VMs

Le coût d'évaluation d'une requête q en présence de l'ensemble V dépend de la taille des vues impliquées et la taille des résultats de q. Plusieurs cas de figures sont distingués et dans notre cas nous ne considérons que les deux scénarios suivants :

4.7.2.1.1 Scénario 1 : Nous considérons que l'exécution de la requête q implique l'extraction de données seulement d'une vue v_i par une opération de sélection simple avec des agrégations, qui correspond à un seul job MapReduce. Ce job s'exécute en deux phases, phase Map et phase Reduce. La phase Map filtre les données de v_i et la phase Reduce calcule les agrégations et renvoie le résultat de la requête vers HDFS. On peut généraliser le coût d'évaluation de q à partir de v_i par la formule suivante :

$$Texec(q,v) = T_{init} + Tmap_v + Treduce_v + Tshuffle_v + Tfin + Ttransfert(q), \quad (4.36)$$

avec :

$$Tmap_v = \max_{i=1}^{N} m_i(\frac{size(inMap_v) + size(outMap_v)}{M_v * Rd}) \ , \quad (4.37)$$

tel que le nombre des taches Map, notée M_v, nécessaire peut être estimé comme suit :

$$M_v = \frac{size(v)}{split} \ , \quad (4.38)$$

$$Tshuffle_v = size(outMap_v) * D. \quad (4.39)$$

On peut aussi évaluer les inputs et outputs des taches Map en utilisant les formules suivantes :

$$size(inMap_v) = size(v), \quad (4.40)$$

$$outMap_v = ||v|| * \prod sel_v. \quad (4.41)$$

Le coût côté reduce, $Treduc_v$, peut être divisé en deux coûts : coût de tri et coût de création des copies des résultats sous DFS. Ce coût peut être estimer par les formules définies dans la section 4.5.2.3, de sorte que :

$$size(inReduce_v) = size(outMap_v), \quad (4.42)$$

$$||outReduce_v|| = Agr(inReduce_v). \quad (4.43)$$

Exemple 4.7.3. La requête Q, de l'exemple 4.4.1, peut être évaluée à partir de la vue v_1 par une simple sélection suivie par l' opérateur d'agrégation SUM et l'opérateur de tri $Order\ by$.

4.7.2.1.2 Scénario 2 : L'exécution d'une requête q implique la jointure entre une vue v_i et p tables de dimension, suivie par des opérateurs de groupement et agrégation. Cela nécessite $p + x$ jobs MapReduce, tels que les p premiers jobs calculent la jointure entre v_i et p tables de dimension et les derniers jobs calculent le groupement, l'agrégation et le tri de données.

Exemple 4.7.4. Soit la requête suivante :

Requête Q3

Select *c_nation*, s_*nation*, *d_year*, sum(*lo_revenue*) as reve-nue
From *customer*, *lineorder*, *supplier*, *date*, *Part*
Where $lo\_custkey = c\_custkey$ and $lo\_suppkey = s\_suppkey$ and $lo\_partkey = p\_partkey$ and $lo\_custkey = c\_custkey$ and c_region = 'ASIA' or and s_region = 'ASIA' and p_mfgr= 'MFGR1#' and $d\_year \geq 1992$ and $d\_year \leq 1997$
Group by *c_nation*, *s_nation*, *d_year*.

La requête $Q3$ est évaluée par la réalisation d'une jointure entre la vue v_1 et la table de dimension $Part$.

Dans ce scénario, le coût d'exécution de la requête q peut être donné par la formule suivante :

$$Texec(q, v) = \sum_{j=1}^{p+x} Tjob_j, \ v \in V. \tag{4.44}$$

Le coût d'exécution d'un job MapReduce peut être donné par la formule 4.3, en remplaçant la table de faits F par la vue v_i.

4.7.2.2 Coût total d'exécution de la charge de requêtes en présence des VMs

Le coût total d'exécution de la charge de requêtes en présence des VMs égale à la somme de temps d'exécution des requêtes en présence de l'ensemble V plus le temps de transfert des résultats des requêtes multiplié par la fréquence d'utilisation des requêtes, qui peut être estimé par la formule suivante :

$$TotalCost_{access}(Q, V) = \sum_{q \in Q, v \in V} [(Txec(q, v) + Ttransfert(q)) * Fr_q], \tag{4.45}$$

4.7.3 Coût de paiement de traitement des requêtes en présence des VMs

Le coût de paiement lié au traitement d'une requête en présence des VMs depend du temps de traitement de la requête et le type et le nombre d'instances. Ce coût peut être généralisé par la formule suivante :

$$Cost_{finance}(q, v) = [Texec(q, v) * NbrInst * PrixInst] + cost_{transfert}(q), \ v \in V. \tag{4.46}$$

Par conséquent, le coût total de paiement de la charge de requêtes en présence des VMs peut être calculé par la formule suivante :

$$TotalCost_{finance}(Q, V) = \sum_{q \varepsilon Q} Cost_{finance}(q, v) * Fr_q, \ v \in V. \tag{4.47}$$

4.7.4 Coût de maintenance

4.7.4.1 Estimation du coût de maintenance

Étant donnée qu'actuellement un environnement MapReduce ne possède pas de techniques de rafraîchissement des vues matérialisées, nous considérons que la maintenance d'une vue consiste à faire recalculer la vue de ses sources de données. Par conséquent, le coût de maintenance englobe le coût de ré-exécution des requêtes de construction de v_i sur les relations de base.

Soit Q' l'ensemble des requêtes de reconstruction de l'ensemble V. La reconstruction d'une vue v_i consiste à ré-exécuter la requête q ($q \in Q'$) correspondante. Par conséquent, le temps d'exécution de la requête q ($q \in Q'$) peut être estimé par les formules définies précédemment dans la section 4.5.1.

Le coût total de maintenance de l'ensemble V se calcule à partir de la somme des coûts de la ré-execution de l'ensemble des requêtes Q', qu'on peut généraliser par la formule suivante :

$$TotalCost(Q) = \sum_{q \varepsilon Q} [Txec(q) * Fupdate_E]. \tag{4.48}$$

avec E désigne l'ensemble des relations de base d'une vue v_i, $E \subset \{F, D_1, D_2, ..., D_n\}$ et $Texec(q)$ peut être estimé par la fonction (4.2) définie précédemment.

4.7.4.2 Coût de paiement lié à la maintenance des VMs

La ré-exécution des programmes MapReduce de reconstruction de l'ensemble V engendre un coût de paiement qui peut être évaluer par la formule suivante :

$$TotalCost_{finance}(Q) = \sum_{q \in Q'} Cost_{finance}(q) * Fupdate_E, \tag{4.49}$$

tel que :

$$Cost_{finance}(q) = Texec(q) * NbrInst * PrixInst, \ q \in Q'. \tag{4.50}$$

Exemple 4.7.5. Soit la vue v_1. La maintenance de v_1 consiste à recalculer $Q2$ de ses relations de base (*customer*, *lineorder*, *supplier* et *date*). La requête $Q2$ comporte une jointure en étoile suivie par l'agrégation du résultat de la jointure. La quantification de coût de traitement de $Q2$ peut être réalisée par les formules définie auparavant dans la section (4.5.1), en écartant le coût de transfert des résultats.

4.8 Conclusion

Dans le présent chapitre, nous avons présenté un nouveau modèle de coût analytique pour la conception physique des entrepôts de données sur le Cloud. Ce modèle se comporte d'un ensemble de formules mathématiques qui prennent en entrée des paramètres, des statistiques

sur les données et la charge de requêtes et des caractéristiques du Cloud, et renvoient comme résultats le coût engendré par le traitement des requêtes décisionnelles sur le Cloud.

Le modèle élaboré est caractérisé par sa simplicité et facilité de mettre en œuvre et permet la prise en compte des caractéristiques du Cloud, particulièrement le modèle de paiement et l'environnement d'exécution. Ainsi, il est capable de diriger les algorithmes de sélection des VMs sur le Cloud dans un environnement MapReduce et de comparer entre la matérialisation des requêtes et l'augmentation de nombre du nœuds de calcul. Comme il peut être utilisé pour choisir le meilleur plan d'exécution d'une requête dans tel environnement. Cependant, le modèle de coût s'appuie sur des hypothèses simplificatrices, qui peuvent diverger le coût estimé de la réalité.

Nous allons dans le suivant chapitre présenter une étude expérimentale dans un système MapReduce à travers laquelle nous déployons un entrepôt de données issue d'un benchmark de données et nous exécutons des requêtes réelles sur des données concrètes, dans le but d'évaluer notre proposition.

Chapitre 5

Déploiement d'un entrepôt de données dans un système MapReduce et analyse du coût d'E/S

5.1 Introduction

Dans le chapitre précédent, nous avons introduit un nouveau modèle de coût analytique dédié à la prédiction des coûts engendrés par le traitement des requêtes décisionnelles, en présence ou non des VMs, sur le Cloud dans un environnement MapReduce. Nous allons, dans ce chapitre, présenter une étude expérimentale, dans un système MapReduce, sur des données réelles issues d'un benchmark de données et un ensemble de requêtes définies sur ce benchmark de données.

Pour évaluer notre modèle de coût proposé sur un entrepôt de données réelles, nous avons considéré celui issu du SSBM [56]. Les données de SSBM sont générées et exportées vers Hive, qui est un système MapReduce open source conçu pour les entrepôts de données.

Dans ce qui suit, nous décrivons les différentes expérimentations réalisées et les résultats obtenus suivi par une comparaison des résultats pratiques aux estimations théoriques obtenues par l'application du modèle de coût analytique.

5.2 Déploiement d'un entrepôt de données dans un système à base de MapReduce

Dans cette section, nous présentons les étapes suivies et les outils utilisés pour réaliser une solution BI basée sur le concept de Cloud Computing.

5.2.1 Infrastructure de déploiement d'un entrepôt de données sur le Cloud Computing

Dans le but d'utiliser une solution BI basée sur le concept de Cloud Computing, nous avons assimilé l'infrastructure décrite par la figure 5.1, dont l'architecture matérielle se compose des machines quelconques. Ces machines sont équipées d'un système d'exploitation GNU/Linux

ou des machines virtuelles, équipées de même système d'exploitation, qui sont créées sur un système d'exploitation principal par l'intermédiaire d'un système de virtualisation.

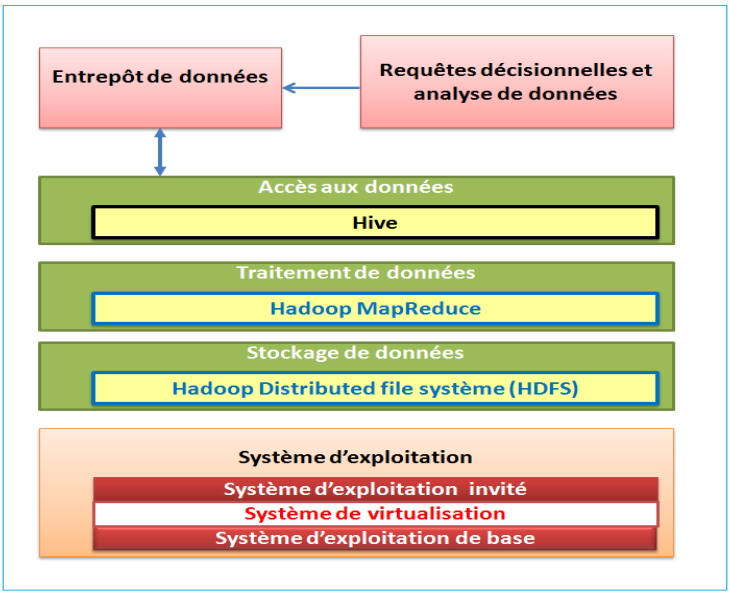


FIG. 5.1 – Infrastructure de la solution BI Cloud Computing

Dans notre travail, nous avons utilisé une machine virtuelle Ubuntu équipée par le système d'exploitation Ubuntu-11.10[1] avec un noyau Linux version 2.6.27-11-server et doté d'un processeur de 2.10 GHz et d'une mémoire de 1 GO. Une fois la configuration matérielle est réalisée, une pile de logiciels doit être installée et configurée : HDFS, Hdoop et Hive (voir l'annexe A). La version Hadoop utilisée est la version hadoop-0.20.2 fonctionne sur Java6.

Pour le déploiement de l'entrepôt de données, nous avons exploité Hive qui est une infrastructure des entrepôts de données open source et largement utilisée dans le Cloud. Hive s'appuie sur le système de fichier distribué HDFS pour le stockage de données et Hadoop MapReduce pour le traitemment de données (voir la figure 5.2). La version Hive utilisée est la version hive-0.8.0.

5.2.2 Démarche de déploiement d'un ED dans un environnement MapReduce

Nous avons utilisé l'entrepôt de données issu de benchmark de test SSBM [56] qui est une variante de TPC-H benchmark[2]. Le schéma en étoile de ce benchmark de données, décrit dans la figure 4.3, est composé d'une seule table de faits centrale (Lineorder) référenciée par quatre tables de dimensions (Customer, Supplier, Part et Date). Ces tables sont générées par un générateur de données DBGEN livré avec SSBM[3] dont le fichier exécutable DBGEN.EXE

[1] `http://www.ubuntu.com`
[2] `http://www.tpc.org`
[3] `https://nodeload.github.com/electrum/ssb-dbgen/zipball/master`

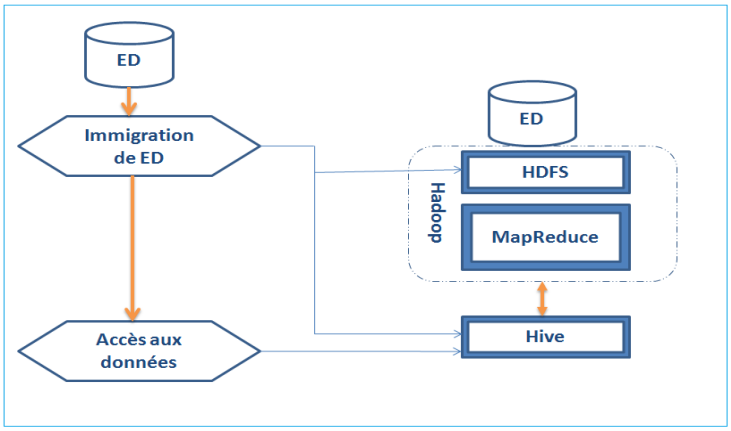


FIG. 5.2 – Vue globale de déploiement d'un ED sur le Cloud

doit être construit par le biais de Visual Studio 2008 [4].

Selon la capacité du système utilisé, nous avons généré les données avec $SF = 1$, qui nous donne une taille totale de ≃570 MO de données. Les principales statistiques des données sont illustrées par le tableau 5.1.

La démarche de déploiement d'un entrepôt de données sur le Cloud dans un environnement MapRdeuce est caractérisée par quatre principales phases (voir la figure 5.3) : (1) Préparation de données (2) Importation de données (3) Réécriture des requêtes (4)Interrogation de données.

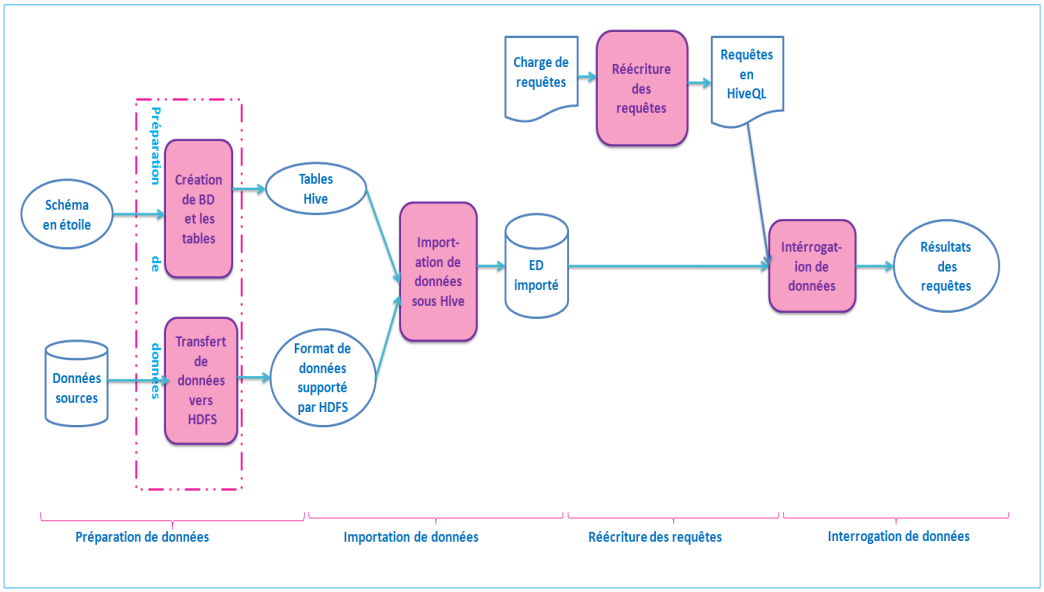


FIG. 5.3 – Démarche de déploiement d'un ED dans un système MapReduce

[4] `http://downloads.malavida.net/files/fr/4000/visual-studio-2008-windows-malavida.exe`

5.2.2.1 Préparation de données :

Cette étape est composée de deux sous-étapes : (1) Transfert de données vers le système de fichier distribué HDFS (2) Création des tables. Le transfert de données vers HDFS consiste à faire importer les tables de données vers HDFS dans un format supporté par ce dernier. À cet effet, les commandes suivantes ont été utilisées :

Préparation de données

```
$HADOOP_HOME/bin/hadoop fs -mkdir /tpchssb/
$HADOOP_HOME/bin/hadoop fs -mkdir /tpchssb/customer
$HADOOP_HOME/bin/hadoop fs -mkdir /tpchssb/lineorder
$HADOOP_HOME/bin/hadoop fs -mkdir /tpchssb/part
$HADOOP_HOME/bin/hadoop fs -mkdir /tpchssb/date
 $HADOOP_HOME/bin/hadoop fs -mkdir /tpchssb/supplier

$HADOOP_HOME/bin/hadoop fs -copyFromLocal customer.tbl /tpchssb/customer/
$HADOOP_HOME/bin/hadoop fs -copyFromLocal lineorder.tbl /tpchssb/lineorder/
$HADOOP_HOME/bin/hadoop fs -copyFromLocal date.tbl /tpchssb/date/
$HADOOP_HOME/bin/hadoop fs -copyFromLocal part.tbl /tpchssb/part/
$HADOOP_HOME/bin/hadoop fs -copyFromLocal supplier.tbl /tpchssb/supplier/
```

Nom de la table	**Taille (MO)**	**Nombre de tuples**
Lineorder	550.33	6001215
Customer	2.73	30000
Supplier	0.160	2000
Part	16.54	200000
Date	0.221	2556

Tab. 5.1 – Données de SSBM avec SF=1

La création des tables consiste à obtenir le schéma logique de l'entrepôt de données et la récupération des noms des tables, leurs attributs et leurs types de données. Hive fournit des primitives de LDD qui permettent de créer des tables et de les stocker dans un format approprié. Les primitives suivantes permettent de créer les tables de l'entrepôt de données générées précédemment :

Création de la table Customer

```
CREATE TABLE Customer(c_custkey INT, c_name STRING, c_address STRING, c_city STRING, c_nation STRING, c_region STRING, c_phone STRING, c_mktsegment STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';
```

Création de la table Supplier

```
CREATE TABLE Supplier(s_suppkey INT, s_name STRING, s_address STRING, s_city
STRING, s_nation STRING, s_region STRING, s_phone STRING) ROW FORMAT DE-
LIMITED FIELDS TERMINATED BY '|';
```

Création de la table Part

```
CREATE TABLE part(p_partkey INT, p_name STRING, p_mfgr STRING, p_category
STRING, p_brand STRING, p_color STRING, p_type STRING, p_size INT, p_container
STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';
```

Création de la table Date

```
CREATE TABLE date(d_datekey INT, d_date STRING, d_dayofweek STRING,
d_month STRING, d_year INT, d_yearmonthnum INt, d_yearmonth STRING,
d_daynuminweek INT, d_daynuminmonth INT, d_daynumyear INT, d_monthnuminyear
INT, d_weeknuminyear INT, d_sellingseason STRING, d_lastdayinweek INT,
d_lastdayinmonthfl INT, d_holidayfl INT, d_weekdayfl INT) ROW FORMAT DE-
LIMITED FIELDS TERMINATED BY '|';
```

Création de la table de faits Lineorder

```
CREATE TABLE lineorder(lo_orderkey INT, lo_linenumber INT, lo_custkey INT,
lo_partkey INT, lo_suppkey INT, lo_orderdate INT, lo_orderpriority STRING,
lo_shippriority STRING, lo_quantity INT, lo_extendedprice INT, lo_ordtotalprice INT,
lo_discount INT, lo_revenue INT, lo_supplycost INT, lo_tax INT,lo_commitdate INT,
lo_shipmod STRING) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|';
```

5.2.2.2 Importation de données :

Cette étape permet de transférer les données de l'entrepôt vers l'infrastructure Hive, en exploitant les primitives DML de Hive. Pour ce faire, nous avons utilisé les commandes suivantes :

Importation des tables de données sous hive

```
LOAD DATA INPATH '/tpchssb/customer/customer.tbl' OVERWRITE INTO TABLE
customer;
LOAD DATA INPATH '/tpchssb/supplier/supplier.tbl' OVERWRITE INTO TABLE
supplier;
LOAD DATA INPATH '/tpchssb/part/part.tbl' OVERWRITE INTO TABLE part;
LOAD DATA INPATH '/tpchssb/date/date.tbl' OVERWRITE INTO TABLE date;
```

5.2.2.3 Réécriture des requêtes :

Cette étape permet de transformer les requêtes SQL en des requêtes Hive-QL, en gardant la même sémantique et en respectant la syntaxe de Hive-QL. La charge de requêtes définie sur l'entrepôt de données issu du benchmark SSBM se compose de treize requêtes (voir l'annexe B). Ces requêtes sont des requêtes de jointure en étoile et respectent la syntaxe que nous avons considérée dans la section 4.4. Dans les tests que nous avons réalisés, nous n'avons considéré que quatres requêtes (Q1.1, Q2.2, Q3.1, Q4.3), avec des adaptations aux hypothèses définies dans le chapitre précédent (voir la section 4.4).

Les requêtes de la charge sont écrites en SQL. Avant l'exécution de ces requêtes sous Hive, une ré-écriture de ces requêtes en HiveQL est réalisée. La ré-écriture de ces requêtes en HiveQL exige des changements syntaxiques mais la sémantique reste la même. La clause SELECT de SQL qui implique une selection de multiples tables devient une clause JOIN en HiveQL et la clause WHERE devient la clause ON dans la clause JOIN.

Q1.1 en HiveQL

```
CREATE TABLE Q1(revenu BIGINT) INSERT OVERWRITE TABLE Q1
SUM(l.lo_extendedprice * l.lo_discount) AS revenue
FROM Lineorder l JOIN Date d
ON
l.lo_orderdate = d.d_datekey and d.d_year=1993 and l.lo_discount >= 1 and
l.lo_discount<= 3 and l.lo_quantity < 25 ;
```

Q2.2 en HiveQL

```
CREATE TABLE Q2(revenu BIGINT, year INT, brand STRING)
INSERT OVERWRITE TABLE Q2
SELECT SUM(l.lo_revenue * l.lo_discount) AS revenue, d_year as year, p_brand as brand
FROM Lineorder l JOIN Date d
ON
l.lo_orderdate = d.d_datekey
JOIN part p
ON
l.lo_partkey=p.p_partkey and p.p_brand >= 'MFGR#2221' and
p.p_brand<='MFGR#2228'
JOIN supplier s
ON
l.lo_suppkey=s.s_suppkey and s.s_region='ASIA'
GROUP BY d_year, p_brand
ORDER BY year, brand ;
```

Q3.1 en HiveQL

```
CREATE TABLE Q3(region1 STRING, region2 STRING, year INT, revenue BIGINT)
INSERT OVERWRITE TABLE Q3
SELECT c_region as region1, s_region AS region2, d_year AS year, SUM(lo_revenue) AS
revenue
FROM Lineorder l JOIN Date d
ON
l.lo_orderdate = d.d_datekey and d.d_year>=1992 and d.d_year<=1997
JOIN supplier s
ON
l.lo_suppkey=s.s_suppkey and s.s_region<>'ASIA'
JOIN Customer c
ON
l.lo_custkey=c.c_custkey and c.c_region<>'ASIA'
GROUP BY c_region, s_region, d_year
ORDER BY year ASC, revenue DESC;
```

Q4.3 en Hive

```
CREATE TABLE Q4(year int, profit BIGINT, city STRING, color STRING, category
string)
INSERT OVERWRITE TABLE Q4 SELECT d.d_year as year, SUM(l.lo_revenue-
l.lo_supplycost) AS profit, s.s_city AS city, p.color as color, p.category as category
from lineorder l JOIN Customer c
ON l.lo_custkey = c.c_custkey and c.c_region='AMERICA'
JOIN Supplier s
ON l.lo_suppkey = s.s_suppkey and s.s_nation='UNITED STATES'
JOIN Part p
ON l.lo_partkey=p.p_partkey and p.p_category<>'MFGR#14'
JOIN Date d
ON l.lo_orderdate = d.d_datekey and d.d_year>=1997
GROUP BY d.d_year, p.category, p.clor
ORDER BY year, category, color
```

5.2.2.4 Interrogation de données :

Cette étape permet l'exécution des requêtes HiveQL sur l'entrepôt de données et la récupération des résultats, et c'est cela ce que nous allons détailler dans la section suivante.

5.3 Expérimentation

Dès que l'assimilation de l'infrastructure et le déploiement de l'entrepôt de données sous Hive sont prêts, nous avons exécuté les requêtes choisies sur l'entrepôt de données importé et nous avons collecté les résultats obtenus. Notons que le système a été utilisé avec les configurations par défaut.

5.3.1 Exécution des requêtes et les résultats obtenus

Au cours de l'expérimentation, chaque requête a été exécutée trois fois et les résultats obtenus sont présentés dans le tableau 5.2. Les résultats sont collectés en terme de nombre de jobs MapReduce constituant chaque requête, la quantité de données d'entrée, la taille des résultats et le temps total d'exécution des requêtes. Le temps total d'exécution présente la moyenne des trois exécutions et les résultats obtenus sont présentés dans la figure 5.4. Notons que nous n'avons pas ajouté le temps d'importation de données et de création des tables aux temps total d'exécution. Les résultats des requêtes sont stockés sous HDFS dans des tables Hive.

Requêtes	**Nombre de Jobs**	**Temps total d'exécution**	**Qte de données lues (MO)/nbr tuples**	**Taille des résultats/nbr tuples**
Q1.1	02	165,527 s	550,551/ 6003771	0,01 KO/1
Q2.2	05	561,653 s	550,551/ 6003771	1,42 KO/56
Q3.1	05	619,597 s	550,551/ 6003771	3,25 KO/96
Q4.3	06	493,826 s	553,062/ 6031215	105,97 KO/3880

TAB. 5.2 – Tableau des résultats d'exécution des requêtes de SSBM sous Hive

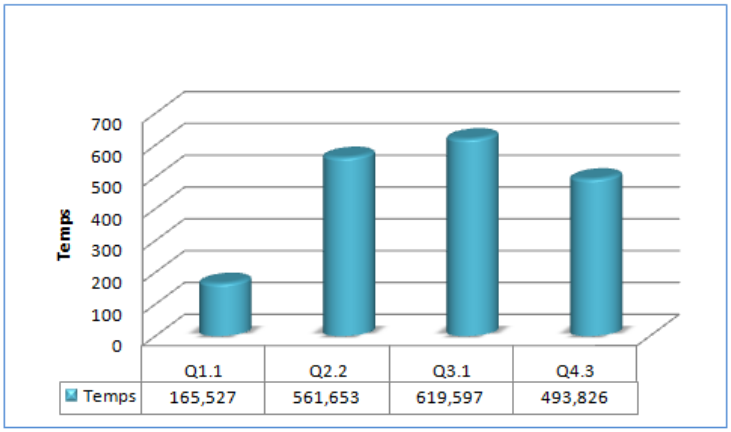


FIG. 5.4 – Résultats d'exécution des requêtes du SSBM sous Hive

Les résultats obtenus montrent que chaque requête s'exécute en un nombre de jobs MapReduce. Ces jobs s'exécutent de manière séquentielle, l'un à la suite des autres. Chaque job se comporte de deux taches, une tache Map et une tache Reduce. Dans les tableaux 5.2 et 5.3, nous avons collecté pour chaque requête le nombre des taches Map et Reduce constituant chaque job, le temps moyen d'exécution et la quantité d'entrée et de sortie de chaque tache ainsi que le temps moyen d'exécution de chaque job.

	Temps total	**M**	**R**	**Tmap**	**Treduce**	**Tinit**	**Tfin**	**Qte shuffle**	**M.input (MO/tuple)**	**M.output (MO/tuple)**	**R.input (MO/tuple)**	**R.output (MO/tuple)**
Requête Q1.1												
Job1	135s	10	1	109,666	11,333	5	2,333	16,765	550,551/ 6003771	16,192/ 955878	16,765/ 955878	2,748 / 144301
Job2	26,333s	2	1	9	4	5	2,333	1,771	2,748/ 144301	1,496/ 144301	1,771/ 144301	13octet/ 1
Requête Q2.2												
Job1	281,666s	10	1	167,333	94,666	7,666	5	181,314	550,551/ 6003771	169,862/ 6003771	181,314/ 6003771	211,969/ 6001215
Job2	193,666s	5	1	130	45	6,666	03	120,804	228,505/ 6201215	161,588/ 6002799	120,804/ 6002799	1,843 / 47572
Job3	26,333s	2	1	8	3,333	5,666	3	1,624	2,004/ 49572	1,537/ 48021	1,624/ 48021	0,362/ 10513
Job4	26,666s	2	1	9	3,666	5	3		0,362/ 10513	10279/ 10513	0,299/ 10513	0,0021/ 56
Job5	23s	2	1	6	4	5	3,333	0,299	0,002/ 56	0,0025/ 56	0,0023/ 56	0,00138/ 56
Requête Q3.1												
Job1	287s	10	1	174,666	91,666	8,333	3	153,24	550,551/ 6003771	153,23/ 6003407	153,23/ 6003407	177,76/ 5466188
Job2	198s	4	1	120,333	59,666	6	3	110,06	177,92/ 5468188	142,44/ 5467739	110,06/ 5466637	154,68/ 4238992
Job3	65,666s	4	1	36	14	6	3	0,11	157,41/ 4268992	122,74/ 4262941	122,74/ 4262941	132,93/ 3386151
Job4	34,666s	2	1	13	5,666	7	3	4,14	132,93/ 3386151	115,93/ 3386151	115,93/ 3386151	0.004 /96
Job5	26s	2	1	8	3,666	6,333	3	0,003	0,004/ 96	0,004/ 96	0,004/ 96	0,004/ 96

TAB. 5.3 – Tableau d'exécution des jobs MapReduce constituant les requêtes Q1.1, Q2.2, Q3.1

	Temps total	**M**	**R**	**Tmap**	**Treduce**	**Tinit**	**Tfin**	**Qte shuffle**	**M.input (MO/tuple)**	**M.output (MO/tuple)**	**R.input (MO/tuple)**	**R.output (MO/tuple)**
Requête Q4.3												
Job1	300,333s	10	1	211,333	71,333	8	3	257,134	553,062/ 6031215	245,676/ 6007207	257,134/ 6007207	51,589/ 1192697
Job2	70,666s	2	1	35	14,666	8,333	3	0,0016	51,750/ 1194697	43,753/ 1192773	43,753/ 1192773	2,247/ 45355
Job3	29s	2	1	11	3,666	6	3	0,166	18,783/ 245355	5,93/ 237326	5,93/ 237326	2,048/ 43533
Job4	27,33s	2	1	9	3,333	6	3	0,118	2,04/ 46089	1,58/ 44262	1,58/ 44262	0,45/ 10567
Job5	26,33s	2	1	9	3	5,333	3	0,016	0,45/ 10567	0,34/ 10567	0,34/ 10567	0,15/ 3880
Job6	25,666s	2	1	7	3,666	7	3	0,0092	0,15/ 3880	0,2/ 3880	0,2/ 3880	0,1/ 3880

TAB. 5.4 – Tableau d'exécution des jobs MapReduce constituant la requête Q4.3

5.3.2 Création des VMS

Matérialiser une vue consiste à exécuter la requête correspondante et stocker son résultat. Cette vue matérialisée peut être utilisée pour évaluer des requêtes utilisateur. L'exécution des requêtes utilisateur en présence des VMs nécessite la réécriture de ces requêtes en termes des vues existantes. Dans notre expérimentation, les résultats des requêtes exécutées précédemment sont stockés dans des tables Hive sous HDFS. On peut les considérés comme des VMs. À cet effet, le coût de matérialisation égale au coût d'exécution des requêtes correspondantes. Ainsi, la taille des résultats de la requête correspondante présente la taille de la vue.

Dans le context des entrepôt de données, la dimension temps est une dimension primordiale. Un entrepôt de données doit toujours comporter une dimension temps, qui peut être fréquemment référenciée par les requêtes de jointure en étoile. Dans les requêtes de SSBM, la dimension date est toujours présente. De ce fait, supposons la matérialisation de la jointure entre la table de faits *Lineorder* et la table de dimension *date*, dans une table LD, qui peut être effectuée par la requête $Q6$ suivante :

Q6 en HiveQL

```
CREATE TABLE LD(lo_orderkey int, lo_linenumber int, lo_custkey int, lo_partkey int,
lo_suppkey int, lo_orderdate int, lo_quantity int, lo_ordtotalprice int, lo_discount int,
lo_revenue int, lo_supplycost int, d_datekey int, d_date string, d_year INT) ;
INSERT OVERWRITE TABLE LD
SELECT lo_orderkey, lo_linenumber, lo_custkey, lo_partkey, lo_suppkey, lo_orderdate,
lo_quantity, lo_ordtotalprice, lo_discount, lo_revenue, lo_supplycost, d_datekey, d_date,
d_year
FROM Lineorder l JOIN Date d
ON l.lo_orderdate = d.d_datekey
```

Le coût de matérialisation de LD est égale au coût d'exécution de Q6 et la taille des résultats de la requête Q6 présente la taille de LD. Le coût d'exécution de Q6 et la taille des résultats sont décrits dans le tableau 5.5.

5.3.2.1 Exécution des requêtes en présence des VMs et les résultats obtenus

Considérons la requête $Q5$ définie ci-dessous. Cette requête peut être évaluée à partir des résultats de la requête $Q3.1$, qui sont stockés sous HDFS dans la table $Q3$. Ainsi, les requêtes $Q1.1$, $Q2.2$, $Q3.1$ et $Q4.3$ peuvent être évaluées à partir de la vue LD, décrite auparavant.

Q5 en HiveQL

```
SELECT c_region as region1, s_region AS region2, d_year AS year, SUM(lo_revenue) AS
revenue
FROM Lineorder l JOIN Date d
ON l.lo_orderdate = d.d_datekey and d.d_year=1994 JOIN supplier s
ON l.lo_suppkey=s.s_suppkey and s.s_region <>='ASIA'
JOIN Customer c
ON l.lo_custkey=c.c_custkey and c.c_region <>'ASIA'
GROUP BY c_nation, s_nation, d_year
ORDER BY year ASC, revenue DESC;
```

L'exécution de la requête Q5 en présence de la table Q3 nécessite une réécriture de la requête en terme de Q3. Cette requête devient comme suit :

Q5 en HiveQL

```
SELECT *
FROM Q3 WHERE year = 1994;
```

De la même manière, l'exécution des requêtes $Q1.1$, $Q2.2$, $Q3.1$ et $Q4.3$ en présence de LD nécessite la réécriture de ces requêtes en terme de LD. Cette réécriture peut se faire par l'élimination de la jointure entre *Lineorder* et *Date* et le remplacement de la table de faits *Lineorder* par LD. Par exemple, la requête Q4.3 devient comme suit :

Réécriture de la requête Q4.3 en terme de LD

```
INSERT OVERWRITE TABLE Q4 SELECT l.d_year as year, SUM(l.lo_revenue-
l.lo_supplycost) AS profit, s.s_city AS city, p.p_brand AS brand
FROM LD l JOIN Customer c
ON l.lo_custkey = c.c_custkey and c.c_region='AMERICA' and l.d_year>= 1997
JOIN Supplier s
ON l.lo_suppkey = s.s_suppkey and s.s_nation='UNITED STATES'
JOIN Part p
ON l.lo_partkey=p.p_partkey and p.p_category='MFGR#14'
GROUP BY l.d_year, s.s_city, p.p_brand
ORDER BY year, city, brand
```

Après l'exécution des requêtes Q5 et Q4.3 en présence respectivement de Q3 et LD sous Hive, nous avons obtenu les résultats indiqués dans le tableau 5.6. La requête Q5 nécessite un seul job MapReduce map-only et par contre la requête Q4.3 requière cinq jobs MapReduce. Le tableau 5.7 présente le coût d'exécution des jobs MapReduce constituant ces requêtes et le temps d'exécution des requêtes $Q1.1$, $Q2.2$, $Q3.1$ et $Q4.3$ est présenté par l'histogramme de la figure 5.5.

Nbr Jobs	**T. d'exécution**	**Qte de données lues/ tuples**	**Taille des résultats/ tuples**	**M/R**
01	323s	550,55 MO /6003771	582,11 MO/ 6001215	10/1
		job1		
T. total	**TMap**	**TReduce**	**Tinit**	**Tfin**
320	148s	95s	4s	3s

Qte shuffle	**M.input (MO/ tuples)**	**M.output (MO/ tuples)**	**R.input (MO/ tuples)**	**R.output (MO/ tuples)**
386,524	550,55/ 6003771	503/ 6003771	503/ 6003771	582,11/ 6001215

TAB. 5.5 – Tableau des résultats de la matérialisation de la requête Q6 sous Hive

Requête	**Nombre de Jobs**	**Temps total d'exécution**	**Qte de données lues**	**Taille des résultats**
Q5	01	20,056 s	3,25 KO /150	0,54 KO/ 16
Q4.3	05	206,915 s	584,84 MO /6031215	0.1 MO/ 3880

TAB. 5.6 – Coût d'exécution des requête Q5 et Q4.3 en présence de Q3 et resp. de LD

	M/R	**Ttotal**	**TM**	**TR**	**Tinit**	**Tfin**	**Qte shuffle**	**M.input**	**M.output**	**R.input**	**R.output**
Requête Q5											
Job1	2/0	18s	12s	-	3s	3s	-	3,25 KO/ 96	0,54 KO/ 16	-	-
Requête Q4.3 en présence de LD											
Job1	10/1	104	60s	12s	6s	3	30,6	584,84/ 60031215	53,7/ 1451038	53,7/ 1451038	0,27/ 289578
Job2	2/1	28	9	5s	4s	3	0,97	11,572/ 291578	9,51/ 289654	9,51/ 289654	0,38/ 10986
Job3	2/1	20	6	2s	6s	3	0,596	16,92/ 210986	4,84 / 202957	4,84 / 202957	0,456/ 10567
Job4	2/1	20	6s	2s	6s	3	0.35	0,456/ 10567	0,35/ 10567	0,35/ 10567	0,15/ 3880
Job5	2/1	20	6s	2s	5s	3	0,2	0,15/ 3880	0,2/ 3880	0,2/ 3880	0,1/ 3880

TAB. 5.7 – Coût d'exécution des jobs constituant les requêtes Q5 et Q4.3 sous Hive

Notons que, M.input, R.input, M.output et R.output sont donnés par : MO/Tuples.

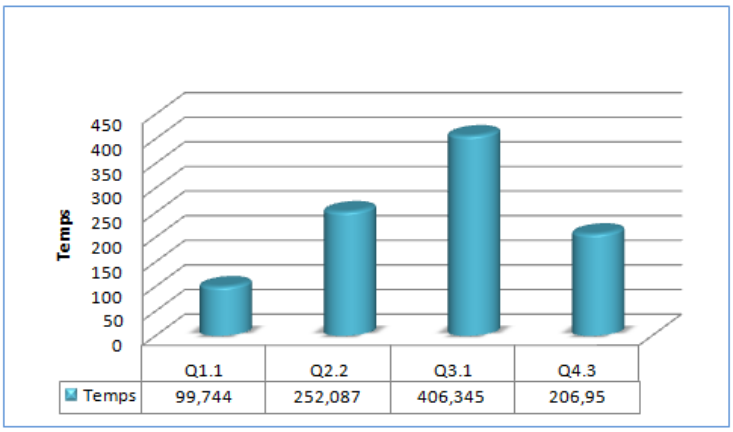


FIG. 5.5 – Résultats d'exécution des requêtes sous Hive en présence des VMs

5.3.2.2 Discussion

Les résultats d'exécution des requêtes en présence des VMs (voir les tableaux 5.6 et 5.7 et le diagramme de la figure 5.5), montrent que l'utilisation des VMs peut améliorer considérablement la performance des requêtes, notamment l'opération de jointure, du fait qu'elle est l'opération la plus coûteuse et qu'elle est présente dans les requêtes de jointure en étoile.

Sans matérialisation de la requête $Q3.1$, la requête $Q5$ nécessite 5 jobs Mapreduce et peut consommer un temps considérable et implique plusieurs jointures entre la table de faits *lineorder* et les tables de dimension *date*, *supplier* et *customer*. Cependant, la matérialisation de la requête $Q3.1$, permet d'améliorer de manière efficace le temps d'exécution de $Q5$ et de réduire le nombre de jobs MapReduce et la quantité de données impliquée.

Pareillement, la comparaison des résultats d'exécution de la requête Q4.3, indiqués dans les tableaux 5.2 et 5.4, aux résultats d'exécution de la même requête en présence de LD, qui sont indiqués dans les tableaux 5.6 et 5.7, indique la même chose. La matérialisation de LD permet d'améliorer de manière efficace le temps de réponse de $Q4.3$ ainsi que de diminuer le nombre de jobs MapReduce et de réduire la quantité de données impliquée.

5.4 Analyse de coût d'E/S

5.4.1 Estimations théoriques

Avant l'application de notre modèle théorique, nous définissons :

$$split = 64MO \text{ et } R = 1.$$

En utilisant le modèle de coût analytique introduit dans le chapitre 4, en tenant compte des valeurs des statistiques sur les données de SSBM (voir annexe B) et en se basant sur

les formules de calcul des facteurs de sélectivité présentées dans l'annexe C, nous présentons dans les tableaux 5.8 et 5.9 les résultats d'application de notre modèle analytique pour les requêtes Q1.1, Q2.2, Q3.1 et Q4.3.

Pour chaque requête, nous avons calculé le nombre de jobs MapReduce et pour chaque job nous avons estimé le nombre des taches Map et le nombre de tuples et la taille de données en entrée et en sortie des taches Map et Reduce constituant chaque job.

Les estimations théoriques sont obtenues par l'application des formules mathématiques élaborées dans la section 4.5.

Requêtes	**Nombre de Jobs**	**Qte de données lues(MO)/nbr tuples**	**Taille des résultats/nbr tuples**
Q1.1	02	550,551/ 6003771	10 Octet/ 1
Q2.2	05	550,551/ 6003771	0.002 MO/56
Q3.1	05	550,551/ 6003771	0.004 MO/ 96
Q4.3	06	553,062/ 6031215	0.17 MO/ 4512

Tab. 5.8 – Tableau des résultats des estimations théoriques

	M	**inMap**	**outMap**	**inReduce**	**outReduce**
Requête Q1.1					
job1	10	550,551/ 6003771	16.32/ 950704	16.32/ 950704	2.6/ 135761
job2	1	2.6/ 135761	1.42/ 135761	1.42/ 135761	10 Octet/ 1
Requête Q2.2					
job1	10	550,551/ 6003771	166.04/ 6003771	166.04/ 6003771	211.75/ 6001215
job2	5	228.29 / 6203771	160.28/ 6002615	160.28/ 6002615	1.83/ 48009
job3	2	1.99/ 50009	1.52/ 48409	1.52/ 48409	0.35/ 9601
job4	1	0.35 / 9601	0.26/ 9601	0.26/ 9601	0.002/ 56
job5	1	.002/ 56	0.002/ 56	0.02/ 56	0.002/ 56
Requête Q3.1					
job1	10	550,551/ 6003771	154.6/6003406	154.6/ 6003406	166.8/5143899
job2	4	166.96/ 5145899	132.5/ 5145499	132.5/ 5145499	149.13/4115119
job3	2	151.86/ 4145119	118.42/ 4139119	118.42/ 4139119	128.72/ 3292095
job4	1	128.72/ 3292095	113.03/ 3292095	113.03/3292095	0.004/ 96
job5	1	0.004/ 96	0.004/ 96	0.004/ 96	0.004/ 96
Requête Q4.3					
job1	10	553.06/ 6031215	240.6/6007215	240.6/ 6007215	51.5/1200243
job2	2	51.66/ 1202243	43.5/1200323	43.5/ 1200323	2.33/48009
job3	2	18.87/ 248009	5.95/ 240009	5.95/ 240009	2.15/ 46088
job4	2	2.371/ 48644	1.65/ 46818	1.65/ 46818	0.55/ 13168
job5	1	0.55/ 13168	0.41/ 13168	0.41/ 13168	0.17/ 4512
job6	1	0.17/ 4512	0.17/ 4512	0.17/ 4512	0.17/ 4512

Tab. 5.9 – Tableau détaillé des résultats des estimations théoriques

Notons que le coût d'E/S est donné par : MO/Tuples.

5.4.2 Comparaison des résultats théoriques aux résultats pratiques

En comparant les résultats d'application du modèle de coût analytique présentés dans les tableaux 5.8 et 5.9 à ceux fournis par l'expérimentation qui sont présentés dans les tableaux 5.2, 5.3 et 5.4, nous remarquons que les formules proposées ont donné une bonne approximation du coût d'E/S. Les valeurs du coût estimées sont légèrement sous-prédite ou sur-prédite par apport aux valeurs réelles. Cela peut être attribué à plusieurs raisons.

Nous avons supposé comme hypothèse dans le chapitre précédent que les données sont uniformément distribuées. Cependant, dans la pratique ce n'est pas toujours le cas. Ainsi, pendant la création des fichiers intermédiaires le framework ajoute des données de contrôle qui augmentent la quantité de données lues ou écrites.

Finalement, les résultats d'expérimentation indiquent que notre modèle de coût théorique est capable d'évaluer le plan d'exécution des requêtes décisionnelles en terme de coût d'E/S dans un environnement MapReduce avec une bonne approximation.

5.5 Mise en œuvre

Dans cette section, nous allons automatiser les formules de prédiction des coûts introduites dans le chapitre 4 dans le but d'élaborer un système de prédiction des coûts d'exécution des requêtes décisionnelles dans un système MapReduce. Le système de prédiction est composé de trois composantes principales (voir la figure 5.6) : (a) analyseur de requêtes (b) calculateur d'E/S (c) calculateur financier.

1. **Analyseur :** Ce module reçoit en entrée des statistiques sur les données et la description de(s) requête(s) de jointure en étoile ou des requêtes de création des VMs candidates et renvoi comme sortie les détailles du plan d'exécution de la requête considérée ; les tables, les attributs de projection et les prédicats de sélection constituant la requête. Ansi, ce module permet de calculer le nombre de jobs MapReduce nécessaire pour l'exécution de la requête considérée.
2. **Calculateur d'E/S :** Chaque job MapReduce implique un ensemble d'opérations d'E/S. Le calculateur d'E/S permet de calculer le coût d'exécution de chaque job en terme d'E/S, ainsi que la taille des résultats intermédiaires.
3. **Calculateur financier :** Cette étape prend en entrée l'ensemble des coûts calculés lors de l'étape précédente et le modèle de paiement de la plate-forme Cloud choisie et fournit en sortie le coût de paiement.

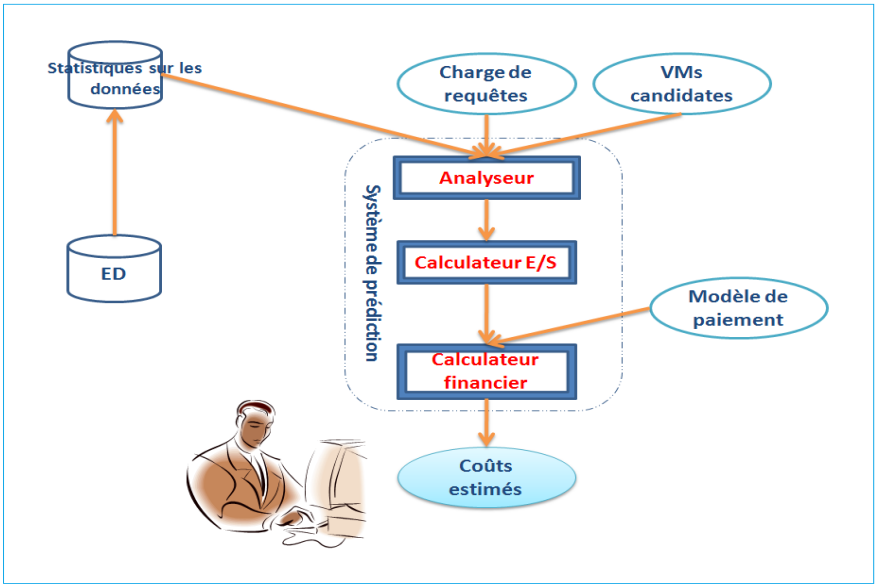


FIG. 5.6 – Architecture d'implémentation

Pour l'implémentation du système de prédiction, nous avons utilisé le langage Java, qui est un langage de programmation orienté objet permet de créer des logiciels indépendants de l'architecture matérielle et compatibles avec de nombreux systèmes d'exploitation (Windows, Mac, Linux, ...). Pour l'environnement de développement intégré (IDE), nous avons exploité Netbeans [5]. Ce dernier est un IDE open source pour le développement avec Java, PHP, C++ et autres langages de programmation et fonctionne sur la plupart des systèmes d'exploitation qui comportent une machine virtuelle Java (JVM)[6].

L'implémentation de ce système nécessite une base de données qui va contenir des statistiques sur l'entrepôt de données et la charge de requêtes et le modèle de paiement des différents fournisseurs. La base de données est stockée sous MySQL [7] qui est un SGBD relationnel diffusé sous une license libre. MySQL fonctionne indifféremment sur un très grand nombre de plate-formes et de systèmes d'exploitation.

Après l'exécution de l'application, on obtient les interfaces données dans la figure 5.7 qui représente quelques captures d'écran relatives à l'implémentation du système de prédiction. Dans cette figure :

- ◇ (a) représente l'interface globale.
- ◇ (b) représente l'interface où l'utilisateur doit entrer la requête pour estimer son coût.
- ◇ (c) représente l'interface où l'utilisateur choisit un sous ensemble des requêtes prédéfinies (stockées dans la BD).
- ◇ (d) représente le résultat d'estimation de la requête entrée dans (b) (qui représente aussi le coût de matérialisation d'une requête).
- ◇ (e) représente les résultats d'estimation lié aux requêtes choisies dans (c).

[5] `http://netbeans.org`
[6] `http://java.com/`
[7] `http://www.mysql.com/`

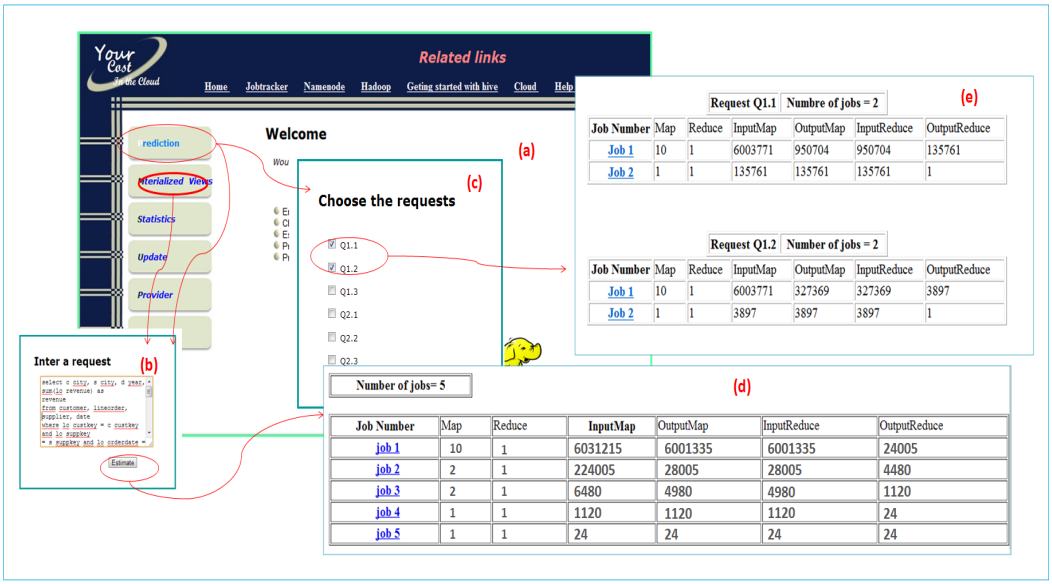


FIG. 5.7 – Captures d'écran de l'implémentation

5.6 Conclusion

Dans ce chapitre, nous avons mené plusieurs expérimentations pour évaluer le modèle de coût analytique proposé, en utilisant des données réelles issues d'un benchmark de données, et nous avons collecté les coûts réels d'exécution des requêtes de jointure en étoile, en présence ou non des VMs, dans un système MapReduce qui se compose de Hive, Hadoop et HDFS.

Les résultats montrent la complexité d'exécution des requêtes de jointure en étoile et l'utilité d'utilisation des VMs. L'utilisation des VMs permet de réduire le nombre de jobs MapReduce, la quantité de données à interroger par les requêtes et par conséquent le temps total d'exécution des requêtes.

La comparaison des résultats d'application du modèle de coût aux résultats pratiques montrent une convergence entre les estimations théoriques et les résultats pratiques. Par conséquent, le modèle de coût élaboré est capable d'estimer le coût d'E/S des requêtes décisionnelles dans un environnement MapReduce avec une bonne approximation.

Conclusion générale

Dans le cadre de ce mémoire, nous nous sommes intéressés à la conception physique des entrepôts de données sur le Cloud. Un domaine très actif, il vise à déterminer comment une requête doit être exécutée efficacement sur l'entrepôt de données. L'objectif de notre travail est de proposer un modèle de coût analytique pour la conception physique des entrepôts de données sur le Cloud. Ce modèle sert à diriger les algorithmes de sélection des VMs et de comparer entre la matérialisation des requêtes et l'augmentation des instances de calculs, pour quantifier le meilleur choix.

Notre proposition s'appuie sur le modèle d'exécution le plus populaire sur le Cloud, MapReduce. Ce paradigme a reçu beaucoup d'attention ces dernières années pour le traitement parallèle de grosses quantités de données. À travers le modèle de coût proposé, nous avons arrivé à estimer le coût d'exécution des requêtes décisionnelles sur le Cloud, dans un environnement MapReduce, en termes de temps d'exécution et en termes de coût de paiement et en présence ou non des VMs.

Le modèle élaboré constitué d'un ensemble de formules mathématiques qui s'appuient sur des hypothèses simplificatrices et des paramètres d'estimation et se caractérisent par leur simplicité, facilité de mettre en œuvre et par rapidité du calcul. Ces formules permettent d'estimer le nombre de jobs MapReduce nécessaires pour l'exécution des requêtes, la taille des résultats et des résultats intermédiaires, le temps d'exécution de chaque job et de chaque phase (phase Map et phase Reduce), ainsi que le temps total d'exécution de la charge de requêtes. L'ensemble des formules développées sont aussi dédiées à l'estimation des différents coûts de paiement liés à l'exécution des requêtes et au stockage et maintenance des VMs sur le Cloud.

Les résultats de notre solution ont été comparés à ceux obtenus par le traitement réel des requêtes décisionnelles sur des données concrètes, issues d'un benchmark de données, dans un système MapReduce. Nous avons exploité comme système MapReduce Hive, qui est une infrastructure open source, largement utilisée dans les plates-formes Cloud Computing et destinée pour les entrepôts de données. Les résultats obtenus ont montré que notre modèle de coût est capable de prédire le coût d'E/S des requêtes décisionnelles dans un environnement

MapReduce avec une bonne approximation.

Dans le but d'une continuité de nos travaux de recherche, nous envisageons de compléter le modèle de coût élaboré par l'ajout des index ou autres structures d'optimisation et la validation de la solution dans une plate-forme Cloud réelle, telle qu'Amazon, Microsoft, etc.

Ce travail nous a permis de dégager plusieurs perspectives de recherche, qui concernent plus spécifiquement les aspects liés à la conception physique des entrepôts de données, dans un environnement MapReduce :

- ◇ Conception d'une démarche pour la sélection automatique des VMs, les index ou la fragmentation horizontale des entrepôts de données, en se basant sur le modèle de coût élaboré ;
- ◇ Amélioration de la jointure en étoile dans un système à base de MapReduce, en réduisant le trafic réseau et le coût d'E/S ;
- ◇ Développement d'une solution générale pour la maintenance incrémentale des VMs en réduisant le coût d'E/S ;
- ◇ Etude multicritère des plates-formes Cloud Computing (choix du fournisseur, etc.) ;
- ◇ Formalisation du problème de conception des entrepôts de données sur le Cloud en impliquant la théorie des jeux.

Bibliographie

[1] D.J. Abadi. Data management in the cloud : Limitations and opportunities. *In IEEE Data Engineering Bulletin, Vol. 32, No. 1, pp. 3-12*, (2009).

[2] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Rasin, and A. Silberschatz. Hadoopdb : An architectural hybrid of mapreduce and dbms technologies for analytical workloads. *In PVLDB'09, The Proceedings of the Very Large Data Bases Endowment, Vol. 2, No. 1, pp. 922–933, Lyon, France*, (2009).

[3] F.N. Afrati and J.D. Ullman. Optimizing joins in a map-reduce environment. *In EDBT'10, Proceedings of the 13th International Conference on Extending Database Technology, pp. 99-110, Lausanne, Switzerland*, (2010).

[4] K. Aouiche. *Techniques de fouille de données pour l'optimisation automatique des performances des entrepôts de données*. Thése de Doctorat en Informatique, Université Lumière Lyon 2, École Doctorale de Sciences Cognitives, France, (2005).

[5] K. Aouiche, O. Boussaid, and F. Bentayeb. Automatic selection of bitmap join indexes in data warehouses. *In DAWAK'05, Proceedings of the 7th International Conference on Data Warehousing and Knowledge Discovery, pp. 64-73, Copenhagen, Denmark*, (2005).

[6] K. Aouiche and J. Darmont. Data mining-based materialized view and index selection in data warehouses. *Journal of Intelligent Information Systems, Vol. 33, No. 1, pp. 65-93*, (2009).

[7] K. Aouiche, P.E. Jouve, and J. Darmont. Clustering-based materialized view selection. *In ADBIS'06, Proceedings of the 10th East-European Conference on Advances in Databases and Information Systems, pp. 81-95, Thessaloniki, Hellas, Greece*, (2006).

[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. *Above the Clouds : A Berkeley View of Cloud Computing*. Technical Report n° UCB/EECS-2009-28, Berkeley, 2009.

[9] X. Baril and Z. Bellahsène. Selection of materialized views : A cost-based approach. *In CAiSE'03, Proceedings of the 15th International Conference on Advanced information Systems Engineering, pp. 665-680, Klagenfurt, Austria*, (2003).

[10] A. Bauer and W. Lehner. On solving the view selection problem in distributed data warehouse architectures. *In SSDBM'03, Proceedings of the 15th International Conference on Scientific and Statistical Database Management, pp. 43-54, Cambridge, Massachusetts, USA*, (2003).

[11] L. Bellatreche and S. Benkrid. A joint design approach of partitioning and allocation in parallel data warehouses. *In DaWaK, Proceedings of the 11th International Conference on Data Warehousing and Knowledge Discovery, Vol. 5691, pp. 99-110, Slinz, Austria*, (2009).

[12] L. Bellatreche and S. Benkrid. F&A : A methodologie for effectively and efficiently desining parallel data warehouses on database clusters. *In DaWaK'10, Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery, Vol. 6263, pp. 89-104, Bilbao, Spain*, (2010).

[13] S. Blanas, J.M. Patel, V. Ercegovac, J. Rao, E.J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. *In SIGMOD'10, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 975–986, Indiana, USA*, (2010).

[14] K. Boukhalfa. *De la Conception Physique des Entrepôts de Données au Tuning et Outil d'Administration, Thèse de Doctorat en Informatique.* Université de Poitiers, France, (2009).

[15] K. Boukhalfa, L. Bellatreche, and P. Richard. *Fragmentation Primaire et Dérivée : Étude de Complexité, Algorithmes de Sélection et Validation sous ORACLE10g.* Rapport de stage, ENSMA, (2008).

[16] E. Brewer. A certain freedom : thoughts on the CAP theorem. *In PODC'10, Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, pp. 335, Zurich, Switzerland*, (2010).

[17] R. Buyya, C.S. Yeo, and S. Venugopal. Market-oriented cloud computing : Vision, hype, and reality for delivering IT services as computing utilities. *In HPCC'08, Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications, pp. 5-13, Dalian, China*, (2008).

[18] D. Carstoiu, E. Lepadatu, and M. Gaspar. Hbase - non sql database, performances evaluation. *In IJACT, International Journal of Advancements in Computing Technology, Vol. 2, No. 5, pp. 42–52*, (2010).

[19] R. Chaiken, B. Jenkins, P.A. Larson, B. Ramsey, D. Shakib, S. Weaver, and J. Zhou. SCOPE : easy and efficient parallel processing of massive data sets. *In PVLDB'08, The Proceedings of the Very Large Data Bases Endowment, Vol. 1, No. 2, pp. 1265–1276, Auckland, New Zealand*, (2008).

[20] G. K. Y. Chan, Q. Li, and L. Feng. Optimized design of materialized views in a real-life data warehousing environment. *In IJIT, International Journal of Information Technology, Vol. 7, No. 1, pp. 30-54*, (2001).

[21] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fikes, and R.E. Gruber. Bigtable : A distributed storage system for structured data. *In OSDI'06, The 7th Symposium on Operating Systems Design and Implementation, pp. 205-218, Seattle, WA, USA*, (2006).

[22] E.F. Codd, S.B. Codd, and C.T. Salley. Providing OLAP to User-Analysts : An IT mandate. *Technical Report, E.F. Codd Associates*, (1993).

[23] B. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts : Yahoo !'s hosted data serving platform. *In PVLDB'08, Proceedings of the the Very Large Database Endowment, Vol. 1, No. 2, pp. 1277–1288, Auckland, New Zealand*, (2008).

[24] J. Darmont. *Optimisation et évaluation de performance pour l'aide à la conception et à l'administration des entrepôts de données complexes*. Habilitation à Diriger des Recherches, Spécialité Informatique, Laboratoire ERIC, Université Lumière Lyon 2, France, (2006).

[25] J. Dean and S. Ghemawat. Mapreduce : Simplified data processing on large clusters. *In OSDI'04, The 6th Symposium on Operating System Design and Implementation, Vol. 6, pp. 137–150, California, USA*, (2004).

[26] J. Dean and S. Ghemawat. Mapreduce : Simplified data processing on large clusters. *In Communications Of The ACM, Vol. 51, No. 1, pp. 107-113, ACM New York, USA*, (2008).

[27] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo : Amazon's highly available key-value store. *In SOSP'07, The Proceedings of 21st ACM Symposium on Operating Systems Principles, pp. 205–220, Washington, USA*, (2007).

[28] L. D'Orazio and S. Bimonte. Intégration des tableaux multidimensionnels en Pig pour l'entreposage de données sur les nuages. *6 emes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2010), Djerba, Tunisie, RNTI, Cépadués, Toulouse, Vol. B-6, pp. 21-34*, (Juin 2010).

[29] M. Figuière. NoSQL Europe : Tour d'horizon des bases de données NoSQL. `http://blog.xebia.fr/2010/04/21/nosql-europe-tour-dhorizon-des-bases-de-donnees-nosql`, (2010)(Consulté le 15/04/2012).

[30] G. Gardarin. *Base de données*. 5ème edition, Eyrolles, France, (Avril 2003).

[31] A. Gates, O. Natkovich, S. Chopra, P. Kamath, S. Narayanam, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava. Building a highlevel dataflow system on top of mapreduce : The pig experience. *In PVLDB'09, The Proceedings of the Very Large Data Bases Endowment, Vol. 2, No. 2 pp. 1414–1425, Lyon, France*, (2009).

[32] S. Ghemawat, H. Gobioff, and S. Leung. Google file system. *In SOSP'03, Proceedings of the 19th ACM Symposium on Operating Systems Principles, Vol. 37, No. 5, pp. 29–43, Bolton Landing, NY USA*, (2003).

[33] M. Goel. *Informatique décisionnelle « cloud ready » avec Oracle Business Intelligence 11g*. Oracle Corporation, (Octobre 2010).

[34] M. Golfarelli, E.Rizzi, and S.Saltarelli. Index selection for data warehousing. *In DMDW'2002, Proceedings 4th International Workshop on Design and Management of Data Warehouses, pp. 33-42, Toronto, Canada*, (2002).

[35] M. Golfarelli and S. Rizzi. A methodological framework for data warehouse design. *In DOLAP'98, 1st ACM international workshop on Data warehousing and OLAP, pp. 3-9, Maryland, USA*, (1998).

[36] A. Hadoop. Welcome to Hive!. `http://hadoop.apache.org/hive/`, (Consulté le 07/09/2012).

[37] A. Hadoop. Hadoop. `http://hadoop.apache.org`, (Consulté le 10/09/2012).

[38] H. Han, H. Jung, H. Eom, and H.Y. Yeom. Scatter-gather-merge : An efficient star-join query processing algorithm for data-parallel frameworks. *In Cluster Computing, the Journal of Networks, Software Tools and Applications, Vol. 14, No. 2, pp. 183-197*, (2011).

[39] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. Mars : a mapreduce framework on graphics processors. *In PACT '08, Proceedings of the 17th international conference on Parallel Architectures and Compilation Techniques, pp. 260–269, Ontario, Canada*, (2008).

[40] W. H. Inmon. *Building the Data Warehouse*. Fourth Edition, Wiley Publishing, Indianapolis, Indiana, USA, (2005).

[41] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad : distributed data-parallel programs from sequential building blocks. *In EuroSys, The Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer, pp. 59–72, Lisbon, Portugal*, (2007).

[42] M. Isard and Y. Yu. Distributed data-parallel computing using a high-level programming language. *In SIGMOD'09, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 987-994, Rhode Island, USA*, (2009).

[43] D. Jiang, A. K. H. Tung, and G. Chen. Map-join-reduce : Towards scalable and efficient data analysis on large clusters. *In TKDE, IEEE Transactions on Knowledge and Data Engineering, Vol. 23, No. 9, pp. 1299-1311*, (2011).

[44] T. Jörg, R. Parvizi, H. Yong, and S. Dessloch. Incremental recomputations in mapreduce. *In CloudDB'11, Proceedings of the Third International Workshop on Cloud Data Management, pp. 7-14, Glasgow, Scotland, UK*, (October 2011).

[45] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool publishers, (2010).

[46] G. Luo and L. Dong. Adaptive join plan generation in hadoop. *For CPS296.1 Course Project, Université Duke, USA, www.cs.duke.edu/~gang/documents/Hadoop%20Join%20Report.pdf*, (Consulté le 10/09/2012).

[47] S. Lutz. *The Future of cloud Computing Opportunities*. Expert Group Report, European Cloud Computing Beyound, (2010).

[48] M.L Mchome. Comparison study between mapreduce (MR) and parallel data management systems (DBMS) in large scale data anlysis. *Honors Projects, Paper 21*, (2011), `http://digitalcommons.macalester.edu/mathcs_honors/21`.

[49] Microsoft. Dryadlinq tutorial. `http://research.microsoft.com/en-us/projects/DryadLINQ/DryadLINQTutorial.docx`, (Consulté le 10/12/2011).

[50] Microsoft. The LINQ project. `http://msdn.microsoft.com/netframework/future/linq/`, (Consulté le 12/12/2011).

[51] M. Mircea, B. Ghilic-Micu, and M. Stoica. Combining business intelligence with cloud computing to delivery agility in actual economy. *In ECECSR, Journal of Economic Computation and Economic Cybernetics Studies and Research, Vol. 45, No. 1, pp. 39-54, Romania*, (2011).

[52] P. Mishra and M. H. Eich. Join processing in relational databases. *ACM Computmg Surveys, Vol. 24, No. 1, pp. 63–113, ACM New York, USA*, (March 1992).

[53] S. Negash. Business Intelligence. *In CAIS, the Communications of the Association for Information Systems, Vol. 13, pp. 177-195*, (2004).

[54] T. Nguyen, L. d'Orazio, S. Bimonte, and J. Darmon. Cost models for view materialization in the cloud. *In EDBT-ICDT'12, Proceedings of the 2012 Joint EDBT/ICDT Workshops, pp. 47-54, Berlin, Allemagne*, (March 2012).

[55] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin : a not-so-foreign language for data processing. *In SIGMOD'08, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1099–1110, Vancouver, BC, Canada*, (2008).

[56] P. O'Neil, B. O'Neil, and X. Chen. The star schema benchmark. (2007), `http://www.cs.umb.edu/~poneil/StarSchemaB.pdf`.

[57] C. Ordonez, I.Y Song, and C. Garcia-Alvarado. Relational versus non-relational database systems for data warehousing. *In DOLAP'10, Proceedings of the ACM 13th international workshop on Data warehousing and OLAP, pp. 67-68, Ontario, Canada*, (2010).

[58] S. Patten. *The S3 COOKBOOK : Get cooking with Amazon's Simple Storage Service*. Spatten Design, (2009).

[59] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. *In SIGMOD'09, Proceedings of the 2009 International Conference on Management of Data, pp. 165-178, Rhode Island, USA*, (July 2009).

[60] G. Petri. *Abécédaire : Éclairage sur le cloud computing*. CA EMEA Product Marketing Advisor, États-Unis, (2010).

[61] R. Pike, S. Dorward, R. Griesemer, and S. Quinlan. Interpreting the data : Parallel analysis with Sawzall. *In Scientific Programming - Dynamic Grids and Worldwide Computing , Vol. 13, No. 4, pp. 277-298, Amsterdam, The Netherlands*, (2005).

[62] G. Plouin. *Cloud Coputing et SaaS : Une rupture décisive pour l'informatique d'entreprise*. Dunod, Paris, (2009).

[63] L. Qian, Z. Luo, Y. Du, and L. Guo. Cloud computing : An overview. *In CloudCom'09, The Proceedings of the 1st International Conference on Cloud Computing , pp. 626–631, Beijing, China*, (2009).

[64] A. Rajaraman and J.D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, (2011).

[65] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis. Evaluating mapreduce for multi-core and multiprocessor systems. *In HPCA 2007, High Performance Computer Architecture, pp.13-24*, (2007).

[66] Amazon Web Service. *Amazon Elastic MapReduce : Developer Guide*. Amazon Web Services LLC or its affiliates, (2011).

[67] Amazon Web Services. *Amazon S3 : Developer Guide*. Amazon Web Services LLC or its affiliates, (2007).

[68] Amazon Web Services. Amazon Simple Storage Service (amazon S3). `http://aws.amazon.com/s3`, (Consulté le 10/05/2012).

[69] Y. Shi, X. Meng, J. Zhao, X. Hu, B. Liu, and H. Wang. Benchmarking cloud-based data management systems. *In CloudDB'10, Proceedings of the second international workshop on Cloud data management, pp. 47-54, Ontario, Canada*, (October 2010).

[70] M. Stonebraker, D. Abadi, D.J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin. Mapreduce and parallel DBMSs : friends or foes ?. *In Communications Of The ACM, Vol. 53, No. 1, pp. 64-71*, (2010).

[71] M. Tamer and P.Valduriez. *Principals of distributed database systems.* 3eme edition, Springer Science+Business Media, New York, USA, (2011).

[72] A. Thusoo, J. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive - a warehousing solution over a mapreduce framework. *In VLDB'09, The Proceedings of the Very Large Data Bases Endowment, Vol. 2 No. 2, pp. 1626–1629, Lyon, France*, (2009).

[73] L.M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds : Towards a cloud definition. *In ACM SIGCOMM Computer Communication Review, Vol. 39, No. 1, pp. 50-55, Barcelona, Spain*, (2009).

[74] J. Varia. *Architecture dans le nuage : les bonnes pratiques.* Amazon Web Services, (2010).

[75] J. Venner. *Pro HADOOP.* first edition, Apress, Berkeley, California, (2009).

[76] T. White. *Hadoop : The Definitive Guide.* First Edition, O'Reilly Media, June (2009).

[77] H. Yang, A. Dasdan, R-L. Hasiao, and D.S Parker. Map-reduce-merge : Simplified relational data processing on large clusters. *In SIGMOD'07, Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1029-1040, Beijing, China*, (2007).

[78] J. Yang, K. Karlapalem, and Q. Li. Algorithm for materialized view design in data warehousing environment. *In VLDB'97, Proceedings of the 23rd International Conference on Very Large Data Bases, pp. 136-145, Athens, Greece*, (1997).

[79] Y. Yu, M. Isard, D. Fetterly, M. Budiu, U . Erlingsson, P. Gunda, and J. Currey. Dryadlinq : A system for general-purpose distributed dataparallel computing using a high-level language. *In OSDI'08, The Proceedings of the 8th USENIX conference on Operating systems design and implementation, pp. 1–14, California, USA*, (2008).

[80] X. Zhang, J. Ai, Z. Wang, J. Lu, and X. Meng. An efficient multi-dimensional index for cloud data management. *In CloudDB'09, Proceedings of the first International Workshop on Cloud Data Management, pp. 17-24, Hong Kong, China*, (2009).

Annexe A

Installation de Hadoop et Hive

A.1 Système d'exploitation

L'utilisation de Hadoop et Hive nécessite une plateforme GNU/Linux. Pour ce faire, nous avons créé une machine virtuelle utilisant "Oracle VM VirtualBox", qui nous a permis d'installer le système d'exploitation Ubuntu-11.10 [1] sous windows.

A.2 Installation Java6 :

Hadoop nécessite une installation préalable de Java. Pour ce faire nous avons suivi les étapes suivantes, utilisant le terminal de commande sous la machine virtuelle Ubuntu :
$ sudo -i
entrer le mot de passe root
$ gedit /etc/apt/sources.list
ensuite, ajouter la line suivante au fichier "sources.list"
deb http ://ppa.launchpad.net/ferramroberto/java/ubuntu oneiric main
enregistrer et sortir, ensuite taper les commandes suivantes :
$ apt-get update
$ apt-get install sun-java6-jdk
$ apt-get install sun-java6-plugin
ensuite définir la variable d'environnement JAVA_HOME :
$ gedit /etc/environment
ajouter les lines suivantes à la fin du fichier :
JAVA_HOME=/usr/lib/jvm/sun-java6
CLASSPATH=. :$JAVA_HOME/lib/dt.jar :$JAVA_HOME/lib/tools.jar

[1] `http://www.ubuntu.com`

A.3 Installation Openssh-server et génération de clé

Hadoop requiert SSH pour la gestion de ses nœuds (local ou à distance) :

$ apt-get install openssh-server

$ ssh-keygen -t rsa (ou : $ ssh-keygen -t rsa -P "")

$ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys

A.4 Installation Hadoop

- ◇ Télécharger une version stable de Hadoop depuis `http://apache.cs.utah.edu/hadoop/core/` ou autre site méroire. Nous avons télécharger et installer hadoop-0.20.2[2]. Pour l'installation nous avons suivi les étapes suivantes :
- ◇ Installer Hadoop par les commandes suivantes :
  $ tar -xvf hadoop-0.20.2.tar.gz
  $ mv /home/hadoop-0.20.2 usr/local/hadoop
  définir la variable d'environnement HADOOP_HOME :
  $ gedit /etc/environment
  HADOOP_HOME=/usr/local/hadoop
  un simple test des commandes Hadoop :
  $ $HADOOP_HOME/bin/hadoop fs

A.5 Configuration Hadoop

Un ensemble de configurations de Hadoop sont nécessaires. Les fichiers de configuration se trouvent dans le répertoire $HADOOP_HOME/conf.

1. Editer le fichier hadoop-env.sh ($ gedit hadoop-env.sh) et changer la line :
   # The java implementation to use. Required.
   #export JAVA_HOME=/usr/lib/j2sdk1.5-sun
   en :
   # The java implementation to use. Required.
   # export JAVA_HOME=/usr/lib/jvm/java-6-sun
   # export HADOOP_HOME=/usr/local/hadoop
2. gedit hdfs-site.xml et ajouter les lines suivantes :
   <property>
   <name>dfs.replication</name>

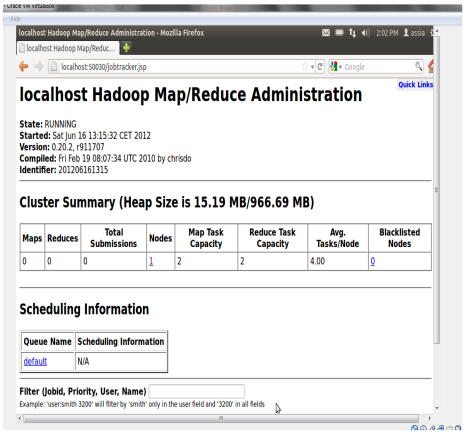
[2] `http://apache.cs.utah.edu/hadoop/core/hadoop-0.20.2/hadoop-0.20.2.tar.gz`

<value>1</value>
</property>

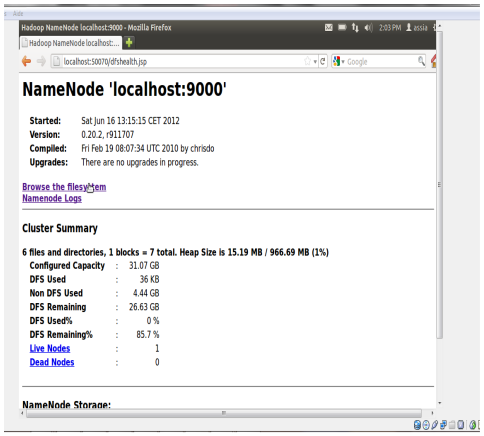
3. $ gedit mapred-site.xml et ajouter les lines suivantes :
   <property>
   <name>mapred.job.tracker</name>
   <value>localhost :9001</value>
   <description>
   </description>
   </property>
4. $ gedit core-site.xml et ajouter les lines suivantes :
   <property>
   <name>hadoop.tmp.dir</name>
   <value>/tmp/hadoop-$user.name</value>
   </property>
   <property>
   <name>fs.default.name</name>
   <value>localhost :9000</value>
   <description>
   </description>
   </property>
5. Lancer la commande suivante pour formater le système de fichier HDFS :
   $HADOOP_HOME/bin/hadoop namenode -format
6. Lancer la commande suivante pour démmarer le nœud Hadoop :
   $HADOOP_HOME/bin/start-all.sh
   Réaliser un simple test par la commande jps
   Pour arrêter le nœud Hadoop utiliser la commande suivante :
   $HADOOP_HOME/bin/stop-all.sh

A.6 Interface web de Hadoop

Hadoop est livré avec des interfaces web qui sont par défaut disponibles à ces adresses :
http ://localhost :50030/ : interface web de Jobtracker (voir la figure (A.1(a)))
http ://localhost :50070/ : interface web de datanode (voir la figure A.1(b))



(a) Interface web de Jobtracker



(b) Interface web de Datanode

A.7 Installation Hive

Dans un premier temps, il faut Télécharger une version stable de Hive depuis `http://apache.claz.org/hive/` ou autre site miroir. Nous avons télécharger et installer hive-0.8.0 [3]. Pour l'installation nous avons suivi les étapes suivantes :

$ tar -xvf hive-0.8.0.tar.gz

$ mv hive-0.8.0 /usr/lacal/hive

Définir la variable d'environnement HIVE_HOME :

$ gedit /etc/environment

HIVE_HOME=/usr/local/hive

Hive stocke ses données dans des répertoires spécifiques dans HDFS. Cela nécessite la création des répertoires suivants :

$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive

$ $HADOOP_HOME/bin/hadoop fs -mkdir /user/hive/warehouse

Pour l'accés à l'environnement hive, nous avons utilisé la commande suivante :

$ $HIVE_HOME/bin/hive (voir la figure A.1)

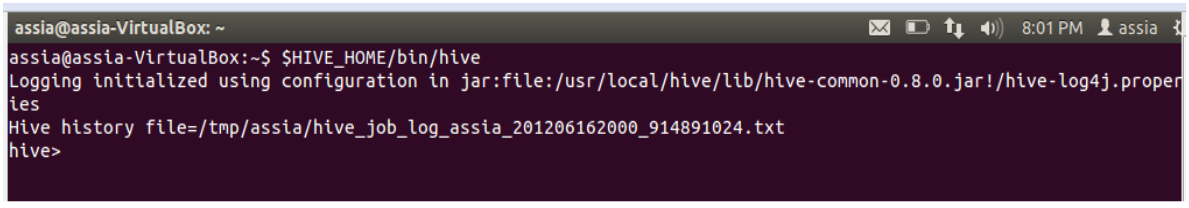


FIG. A.1 – Accès à l'environnement Hive

[3] `http://apache.claz.org/hive/hive-0.8.0/hive-0.8.0.tar.gz`

Annexe B

Charge de requêtes et description des tables de SSBM

B.1 Charge de requêtes

Q1.1 -
select sum(lo_extendedprice*lo_discount) as revenue **from** lineorder, date
where lo_orderdate = d_datekey and d_year = 1993 and lo_discount between 1 and 3 and lo_quantity < 25 ;

Q1.2-
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey and d_yearmonth = 199401 and lo_discount between4 and 6 and lo_quantity between 26 and 35 ;

Q1.3-
select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey and d_weeknuminyear = 6 and d_year = 1994 and lo_discount between 5 and 7 and lo_quantity between 26 and 35 ;

Q2.1-
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier **where** lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_category = 'MFGR#12' and s_region = 'AMERICA'
group by d_year, p_brand1
order by d_year, p_brand1 ;

Q2.2-
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand1between 'MFGR#2221 and 'MFGR#2228' and s_region = 'ASIA'
group by d_year, p_brand1
order by d_year, p_brand1 ;

Q2.3-
select sum(lo_revenue), d_year, p_brand1
from lineorder, date, part, supplier
where lo_orderdate = d_datekey and lo_partkey = p_partkey and lo_suppkey = s_suppkey and p_brand1= 'MFGR#2239' and s_region = 'EUROPE'
group by d_year, p_brand1
order by d_year, p_brand1 ;

Q3.1-
select c_nation, s_nation, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_region = 'ASIA' and s_region = 'ASIA' and d_year >= 1992 and d_year <= 1997
group by c_nation, s_nation, d_year
order by d_year asc, revenue desc ;

Q3.2-
select c_city, s_city, d_year, sum(lo_revenue) as

revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and c_nation = 'UNITED STATES' and s_nation = 'UNITED STATES' and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc ;

Q3.3 -
select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and (c_city='UNITED KI1' or c_city='UNITED KI5') and (s_city='UNITED KI1' or s_city='UNITED KI5') and d_year >= 1992 and d_year <= 1997
group by c_city, s_city, d_year
order by d_year asc, revenue desc ;

Q3.4-
select c_city, s_city, d_year, sum(lo_revenue) as revenue
from customer, lineorder, supplier, date
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_orderdate = d_datekey and (c_city='UNITED KI1' or c_city='UNITED KI5') and (s_city='UNITED KI1' or s_city='UNITED KI5') and d_yearmonth = 'Dec1997'
group by c_city, s_city, d_year
order by d_year asc, revenue desc ;

Q4.1-
select d_year, c_nation, sum(lo_revenue - lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, c_nation
order by d_year, c_nation ;

Q4.2-
select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and c_region = 'AMERICA' and s_region = 'AMERICA' and (d_year = 1997 or d_year = 1998) and (p_mfgr = 'MFGR#1' or p_mfgr = 'MFGR#2')
group by d_year, s_nation, p_category **order by** d_year, s_nation, p_category ;

Q4.3-
select d_year, s_city, p_brand1, sum(lo_revenue - lo_supplycost) as profit
from date, customer, supplier, part, lineorder
where lo_custkey = c_custkey and lo_suppkey = s_suppkey and lo_partkey = p_partkey and lo_orderdate = d_datekey and s_nation = 'UNITED STATES' and (d_year = 1997 or d_year = 1998) and p_category = 'MFGR#14'
group by d_year, s_city, p_brand1 ;
order by d_year, s_city, p_brand1 ;

B.2 Description des tables de SSBM

Attribut	Type	Taille	Card
Lineorder			
lo_orderkey	entier	4	6000000
lo_linenumber	entier	4	7
lo_custkey	entier	4	20000
lo_partkey	entier	4	200000
lo_suppkey	entier	4	2000
lo_orderdate	entier	4	2406
lo_orderpriority	text	15	5
lo_shippriority	text	1	-
lo_quantity	entier	4	50
lo_extendedprice	entier	4	-
lo_ordtotalprice	entier	4	-
lo_discount	entier	4	11
lo_revenue	entier	4	-
lo_supplycost	entier	4	
lo_tax	entier	4	9
lo_commitdate	entier	4	-
lo_shipmode	text	10	-
Supplier			
s_suppkey	entier	4	2000
s_name	text	25	-
s_address	text	25	-
s_city	text	10	250
s_nation	text	15	25
s_region	text	12	5
s_phone	text	15	-

Attribut	Type	Taille	Card
Date			
datekey	entier	4	2556
d_date	text	10	2556
d_dayofweek	text	8	7
d_month	text	9	12
d_year	entier	4	7
d_yearmonthnum	entier	4	84
d_yearmonth	text	7	84
d_daynuminweek	entier	4	7
d_daynuminmonth	entier	4	31
d_daynumyear	entier	4	366
d_monthnuminyear	entier	4	12
d_weeknuminyear	entier	4	53
d_sellingseason	text	12	2
d_lastdayinweekfl	booleen	1	2
d_lastdayinmonthfl	booleen	1	2
d_holidayfl	booleen	1	2
d_weekdayfl	booleen	1	2
Customer			
c_custkey	entier	4	30000
c_name	text	25	-
c_address	text	25	-
c_city	text	10	250
c_nation	text	15	25
c_region	text	12	5
c_phone	text	15	-
mktsegment	text	10	-

Attribut	Type	Taille	Card
Part			
p_partkey	entier	4	200000
p_name	text	22	-
p_mfgr	text	6	5
p_category	text	7	25
p_brand	text	9	1000
p_color	text	11	94
p_type	text	25	150
p_size	entier	4	50
p_container	text	10	40

Annexe C

Calcul des facteurs de sélectivité

La sélectivité est l'un des éléments essentiels pour l'estimation de coût des requêtes. La sélectivité désigne un coefficient associé à une opération qui représente une proportion de tuples satisfaisant une condition définie par l'opération. Généralement, L'estimation de la sélectivité suppose, une distribution uniforme des valeurs des attributs et que tous les attributs sont indépendants [71].

1. **Facteur de sélectivité des opérations de sélection**
   Soit R une relation, A et B deux attributs de R. Tuple(σ(R)) = s * Tuple(R) [71] tel que s dénote le facteur de sélectivité de sélection qui peut être calculé comme suit[71] :
   - ◇ s (A = valeur) = 1/NDIST(A)
   - ◇ s (A > valeur) = (max(A) - valeur) / (max(A) - min(A))
   - ◇ s (A < valeur) = (valeur - min(A)) / (max(A) - min(A))
   - ◇ s (A IN liste valeurs) = (1/NDIST(A)) * Card(liste valeurs)
   - ◇ s (A et B) = s(A) * s(B)
   - ◇ s (P ou Q) = s(P) + s(Q) - s(P) * s(Q)
   - ◇ s (not P) = 1 - s(P)
2. **Facteur de sélectivité des opérations de projection**
   Tuple(Π(R)) = (1-d) * Tuple(R), tel que d = probabilité de doubles
3. **Facteur de sélectivité des opérations de jointure**
   Le facteur de sélectivité de jointure est défini comme le ratio du nombre de tuples participant à la jointure apporté au nombre de tuples dans le produit cartisien des relations de la jointure [52]. Le facteur de sélectivité de jointure f=$\frac{|R \infty T|}{|R|.|T|}$ qui est une valeur réelle entre 0 et 1 [71], telle que :
   - ◇ f = 0 si aucun tuple ne joint
   - ◇ f = 1 / max(NDIST(A),NDIST(B)) s'il existe une distribution uniforme équiprobable des attributs A et B sur un même domaine
   - ◇ f = 1 si produit cartésien

   Le nombre de tuples de jointure entre deux relations R et T est donné par la formule suivante : Tuple(R ∞ T) = f * Tuple(R) * Tuple(T) [71].

RÉSUMÉ

La naissance du Cloud Computing a donné un nouveau mode de consommation de l'informatique et a changé la manière d'investissement des entreprises dans les infrastructures informatiques. Avec la réputation progressive du Cloud, de multiples applications se déplacent vers le Cloud. Les applications de gestion de données analytiques sont des candidates éventuelles pour le déploiement sur le Cloud. Ces applications sont destinées aux processus d'aide à la décision, qui requièrent l'analyse d'une grande masse de données, qui sont souvent stockées au sein d'un entrepôt de données, et nécessitent des requêtes complexes et très coûteuses en temps de traitement.

Plusieurs techniques d'optimisation sont utilisées pour améliorer la performance de ces requêtes, telles que les index et les vues matérialisées. Généralement, le choix de la configuration optimale des structures d'optimisation s'appuie sur l'utilisation d'un modèle de coût, pour quantifier la qualité de chaque configuration générée.

Notre travail consiste à élaborer un modèle de coût analytique pour la sélection des vues matérialisées sur le Cloud dans un environnement MapReduce. Ce modèle de coût est capable de comparer entre la matérialisation des requêtes et l'augmentation des ressources informatiques ainsi que de diriger les algorithmes de sélection des vues matérialisées. Les résultats de notre solution ont été comparés à ceux obtenus avec le traitement réel des requêtes décisionnelles sur des données concrètes, issues d'un benchmark de données, dans un système MapReduce. Nous avons exploité comme système MapReduce Hive, qui est une infrastructure des entrepôts de données open source, largement utilisée dans le Cloud. Les résultats obtenus ont montré que les estimations théoriques de coût D'E/S sont proches de la réalité.

Mots clés : Cloud Computing, MapReduce, Hadoop, Entrepôt de données, Jointure en étoile, Modèle de coût.

ABSTRACT

The cloud Computing has given a new approach of IT consumption and has changed the manner of enterprise investment in IT infrastructure. With the progressive reputation of cloud computing, multiple applications are moving to the Cloud. The analytical data management applications are potential candidates for deployment in the cloud. These applications are intended to decision support process, which require the analysis of a large mass of data, using complex queries. These data are often stored in a data warehouse and require complex queries and very time consuming to treatment. Several optimization techniques are used to improve the performance of these decision queries, such as indexes, materialized views. Usually, the choice of the optimal configuration of optimization structures is based on a cost model, in order to quantify the quality of each generated configuration.

The purpose of our work is to develop an analytical cost model for the selection of materialized views in the Cloud, using MapReduce environment. This cost model is able to compare the materialization of queries with increased resources and conduct the algorithms of selection materialized views. The results of our solution are compared to those obtained with the real decision-support queries on concrete data, which are generated from a data benchmark, in a MapReduce system. We have operated Hive as MapReduce system, which is an open source infrastructure of data warehouse widely used in the cloud computing platform. The obtained results have shown that the estimates of the I/O cost are close to the reality.

Key words : Cloud Computing, MapReduce, Hadoop, Data warehouse, Star join, Cost model.