Auditer l'énergie. Avant de déployer ses modèles : Vers des optimiseurs verts de requêtes analytiques

Simon Pierre Dembele

THÈSE

pour l'obtention du Grade de

Docteur de l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique

(Diplôme National - Arrêté du 25 mai 2016)

Ecole Doctorale : Sciences et Ingéniérie des Systèmes, Mathématiques, Informatique
Secteur de Recherche : Informatique et Applications

Présentée par :

Simon Pierre DEMBELE

****

Auditer l'Énergie – Avant de Déployer ses Modèles : Vers des optimiseurs verts de requêtes analytiques

****

Directeur de thèse : **Ladjel BELLATRECHE**

****

Soutenue le 8 Juillet 2021
devant la Commission d'Examen

****

JURY

Président :
Isabelle COMYN-WATTIAU Professeur, ESSEC, Paris

Rapporteurs :
Philippe FOURNIER-VIGER Professeur, Harbin Institute of Technology (HIT), Chine
Nabil LAYAIDA Directeur de Recherches (HDR), INRIA, Grenoble

Membres du jury :
Laurent LEFEVRE Chargé de Recherches (HDR), INRIA, Lyon
Carlos ORDONEZ Professeur, Université de Houston TX, USA
Ladjel BELLATRECHE Professeur, ISAE - ENSMA, Poitiers

Intelligence is the ability of a system to adjust appropriately to a changing world.

— Christopher Evans

Remerciement

Si l'objectif de la thèse est d'approfondir ses capacités intellectuelles, sa réalisation est un travail fastidieux de longue haleine couronnée par des moments de doute, de frustration mais aussi de joie. Ce travail de thèse n'aurait été possible sans le soutient, l'accompagnement et l'encouragement de nombreuses personnes. Je prie toutes ces personnes de trouver en ces mots toute ma gratitude.

Mes plus vifs remerciements vont à Monsieur **Ladjel Bellatreche**, professeur à l'ISAE-ENSMA et directeur de l'école doctorale Sciences et Ingénierie des Systèmes, Mathématiques, Informatique, de m'avoir accueilli au sein de son équipe et d'être mon directeur de thèse. Les échanges souvent mortifiants ont été toujours riches et fructueux pour me façonner. Qu'il soit aussi remercié pour le temps conséquent qu'il m'a accordé, ses orientations pédagogiques et scientifiques, sa franchise, sa sympathie et pour les nombreux encouragements qu'il m'a prodigué.

Mes remerciements s'adressent également aux professeurs **Philippe FOURNIER-VIGER**, **Nabil LAYAIDA**, **Laurent LEFEVRE**, **Isabelle COMYN-WATTIAU**, **Carlos ORDONEZ** pour l'honneur qu'ils me font en acceptant d'évaluer ce travail de thèse. Je remercie l'ensemble des permanents du laboratoire LIAS à travers le Directeur adjoint **Emmanuel GROLLEAU**, pour leur accueil et pour leur soutien. Je remercie d'une manière particulière **Mickael BARON** et **Bénédicte BOINOT** pour leur accompagnement et les nombreux conseils tout au long de ce travail.

Je remercie tous les enseignants de l'université de Poitiers à travers le professeur **Eric Andres**, chef du département informatique pour avoir partagé avec moi leur expérience du milieu académique. Je formule également ma gratitude aux collègues de la faculté des sciences des techniques et des technologies de Bamako (USTTB-FST) à travers tout le soutien technique et administratif que m'a apporté le chef du département mathématique-informatique en la personne du professeur **Yaya Koné**. Je tiens à remercier également le professeur **Salmi Cheick** de l'université Hamed Bouguera de Boumerdès pour sa confiance et son soutien pour que je réussisse cette thèse.

Je remercie mes camarades doctorants et anciens doctorants rencontrés au laboratoire pour leurs encouragements et pour les moments de joie et de peines que nous avons partagés tout au long de ces années de thèse. Je souhaite à tous, tout le bonheur du monde.

Ma gratitude et mes pensées émues vont également à mes parents qui m'ont guidé toute ma vie. Leur conseil et leur rigueur ont fait échos en mois tout au long de mon parcours dans un esprit de travail et de réussite. C'est alors avec plaisir et émotion qu'à mon tour je leur dédie le fruit de mes efforts. J'espère être à la hauteur de leur fierté inconditionnelle. Sont joints à mes remerciements mes frères **Francis**, **Philippe**, **Aloïs**, **Drissa**, **Jean Pierre**, **Jean**; mes sœurs **Bernadette**, **Augustine**, **Émelie**, **Awa**; mes amis **Gervais Baya**, **Clément Keita**, **Marie augustine Sidibé**, **Nicole Mouton**, **Didier malezi**, **Amaguime Luc**, **Emmanuel Diarra** et tous mes cousins et cousines. Enfin je ne pourrai jamais assez remercier ma femme, **Geneviève** pour l'encouragement et l'amour qu'elle m'a porté pendant les moments difficiles. J'ai une pensée particulière à mes enfants et tous mes neveux qui ont suit se passer de moi durant mes absences.'

Table des matières

Table des matières **5**

I Introduction Générale 11

1 Introduction Générale **13**
1.1 Contexte et Problématique . . . 15
1.1.1 La consommation d'énergie dans les SSDs . . . 15
1.1.2 Impact environnemental et sanitaire . . . 16
1.1.3 Apport de l'efficacité énergétique des SSDs . . . 16
1.1.4 Vers des initiatives écologiques pour les SSDs . . . 17
1.2 Notre vision de la caractérisation de l'énergie dans les BDs . . . 18
1.3 Objectifs . . . 20
1.4 Contributions de la Thèse . . . 21
1.4.1 Taxonomie des approches d'efficacité énergétique . . . 21
1.4.2 Étude comparative de la consommation énergétique . . . 21
1.4.3 Définition des modèles de coût énergétique . . . 21
1.4.4 Traitement éco-énergétique des requêtes . . . 21
1.4.5 Analyse des compteurs de performance matérielle . . . 22
1.5 Organisation du manuscrit . . . 22
1.6 Publications . . . 23

II État de l'art 25

2 Les Systèmes de traitement des requêtes (STRs) **27**
2.1 Introduction . . . 29
2.2 Les STRs dans les SSDs . . . 29
2.2.1 Architecture générique des STRs . . . 29
2.2.1.1 Formulation de la requête . . . 30
2.2.1.2 Compilation . . . 30
2.2.1.3 Optimisation . . . 30
2.2.1.4 Moteur d'exécution . . . 31
2.2.2 Optimisation des requêtes . . . 31
2.2.2.1 STRs dans les systèmes de stockage centralisés . . . 31
2.2.2.2 STRs dans les systèmes de stockage distribués . . . 40
2.2.2.3 STRs dans les systèmes multi-bases . . . 41
2.2.2.4 STRs dans les systèmes multi-stores . . . 41
2.3 Modèle de coût . . . 43
2.4 Les paramètres sensibles à la dimension énergétique . . . 44
2.4.1 Types de la Requête . . . 44
2.4.2 Types des SSDs . . . 44
2.4.3 Caractéristiques matérielles . . . 45
2.4.4 Configurations architecturales . . . 45
2.4.5 Influences environnementales . . . 46
2.5 Conclusion . . . 46

3 Gestion de l'énergie dans les Systèmes de Traitements des Requêtes 47
3.1 Introduction . . . 49
3.2 Préliminaire . . . 49
3.2.1 Concept de l'énergie . . . 49
3.2.2 Efficacité énergétique . . . 50
3.2.3 Motivations . . . 50
3.2.3.1 Problème lié à la consommation excessive de l'énergie . . . 50
3.2.3.2 Bénéfice de l'efficacité énergétique . . . 51
3.2.4 Méthode d'évaluation de l'EE . . . 51
3.2.4.1 Métriques énergétiques . . . 51
3.2.4.2 Modèles énergétiques . . . 52
3.3 Taxonomie des Techniques éco-énergétiques . . . 55
3.3.1 Solutions matérielles . . . 55
3.3.1.1 Technique de l'ajustement dynamique de la tension (DVFS) . . . 56
3.3.1.2 Désactivation Dynamique des Composants . . . 57
3.3.1.3 Co-traitements . . . 58
3.3.2 Solutions logicielles . . . 58
3.3.2.1 Technique de virtualisation . . . 59
3.3.2.2 Technique de l'infonuagique (Cloud Computing) . . . 59
3.3.3 Gestion de la température . . . 60
3.3.4 Standards et Conduite . . . 60
3.4 Approches d'EE dans les STRs . . . 61
3.4.1 Approches Orientées Matérielles . . . 61
3.4.1.1 Les unités de traitement . . . 61
3.4.1.2 Technique de Co-traitement . . . 63
3.4.1.3 Les unités de stockage . . . 63
3.4.2 Approches Orientées Logicielles . . . 64
3.4.2.1 Traitement éco-énergetique . . . 64
3.4.2.2 Conception physique éco-énergetique . . . 67
3.5 Conclusion . . . 68

III Contributions 69

4 Méthodologie de conception de nos McE et Formulation 71
4.1 Introduction . . . 73
4.2 Modèles de coût énergétique (McE) dans les STRs . . . 73
4.2.1 Niveau de modélisation . . . 73
4.2.2 Mode d'exécution des requêtes . . . 73
4.3 Méthodologie de conception des McE . . . 74
4.3.1 Audit des SGBDs . . . 75
4.3.2 Identification des paramètres . . . 75
4.3.3 Formalisation du modèle . . . 76
4.3.4 Validation du modèle . . . 76
4.3.5 Exploitation . . . 76
4.4 Audit des systèmes SSDs étudiés . . . 76
4.4.1 PostgreSQL . . . 77
4.4.1.1 Architecture . . . 77
4.4.1.2 Exécution de la requête . . . 77
4.4.2 MonetDB . . . 78
4.4.2.1 Architecture . . . 79
4.4.2.2 Exécution de la requête . . . 79
4.4.3 Hyrise . . . 79
4.4.3.1 Architecture . . . 80
4.4.3.2 Exécution de la requête . . . 80
4.5 Analyse et études comparatives des SSDs étudiés . . . 81
4.5.1 Environnement de l'étude . . . 81
4.5.2 Évaluation des résultats . . . 82

4.5.2.1 Temps et énergie consommée lors du chargement des données . . . 82
4.5.2.2 Temps d'exécution et énergie consommée lors du traitement des requêtes 82
4.5.3 Identification des paramètres du modèle énergétique . . . 84
4.5.3.1 Stratégie d'exécution d'une requête . . . 84
4.5.3.2 Effet du mode d'exécution . . . 86
4.5.3.3 Effet de la taille du BD . . . 87
4.6 Conception des McE . . . 88
4.6.1 Hypothèses . . . 88
4.6.2 Formulation . . . 89
4.6.2.1 Modélisation des pipelines (PostgreSQL) . . . 89
4.6.2.2 Modélisation des opérateurs (MonetDB & Hyrise) . . . 91
4.6.3 Machine learning . . . 92
4.6.3.1 Les techniques régressives . . . 92
4.6.3.2 Les techniques cognitives . . . 94
4.6.4 Valeurs des paramètres . . . 97
4.6.4.1 Configuration matérielle . . . 97
4.6.4.2 Données d'apprentissage . . . 97
4.6.4.3 Application avec le logiciel R . . . 98
4.7 Validation des modèles . . . 106
4.7.1 Méthode d'évaluation . . . 106
4.7.2 Jeux de données . . . 106
4.7.3 Erreurs d'estimation . . . 107
4.7.3.1 Évaluation sur PostgreSQL . . . 107
4.7.3.2 Évaluation sur MonetDB . . . 109
4.7.3.3 Évaluation sur Hyrise . . . 110
4.7.4 Discussion . . . 112
4.8 Conclusion . . . 116

5 Frameworks pour le déploiement de nos modèles de coût énergétique 117
5.1 Introduction . . . 119
5.2 Optimisation énergétique dans le module de traitement des requêtes . . . 119
5.2.1 Influence des différentes étapes de traitement . . . 119
5.2.1.1 Analyse syntaxique . . . 119
5.2.1.2 Réécriture . . . 119
5.2.1.3 Plan d'exécution . . . 120
5.2.1.4 Exécution . . . 120
5.2.2 Intégration avec PostgreSQL . . . 121
5.2.2.1 Modèle de coût . . . 121
5.2.2.2 Plan d'évaluation . . . 123
5.2.2.3 Architecture fonctionnelle . . . 123
5.2.2.4 Résultats expérimentaux . . . 126
5.2.3 Intégration avec MonetDB . . . 128
5.2.3.1 Modèle de coût . . . 129
5.2.3.2 Architecture fonctionnelle . . . 130
5.2.3.3 Résultats expérimentaux . . . 131
5.3 Estimation dynamique de l'énergie lors du traitement des requêtes (Simulation) . . . 132
5.3.1 Architecture fonctionnelle . . . 133
5.3.2 Résultat expérimentaux . . . 133
5.4 Conclusion . . . 135

IV Conclusion et perspectives 137

6 Conclusion générale et Perspectives 139
6.1 Conclusion . . . 139
6.1.1 Audit des STRs . . . 140
6.1.2 L'efficacité énergétique dans les systèmes informatiques . . . 140
6.1.3 Modèles de coût énergétique . . . 140
6.1.4 L'utilisation de l'apprentissage automatique . . . 141

6.1.5 Compromis de traitement entre la performance et l'énergie . . . 141
6.1.6 Développement des outils pour le traitement vert des requêtes . . . 141
6.2 Perspectives . . . 142
6.2.1 Extension du modèle de coût . . . 142
6.2.2 Traitement écologique des requêtes . . . 142
6.2.3 Optimisation énergétique dans les systèmes de stockage en mémoire . . . 142
6.2.4 Prise en compte de d'autres modèles et des requêtes . . . 143
6.2.5 Vers un simulateur pour le choix d'un SSD adapté à un client . . . 143
6.2.6 Retour d'expérience: Learned Modèles de coût . . . 143
6.2.7 Autres approches d'optimisation énergétique dans les BDs . . . 143

Bibliographie **145**

V Annexes **159**

A Modèle de coût dans un système de stockage orienté ligne **161**
A.1 Introduction . . . 161
A.2 Estimation de la cardinalité des opérateurs . . . 161
A.2.1 cardinalité des résultats intermédiaires . . . 161
A.2.1.1 Sélection . . . 162
A.2.1.2 Projection . . . 162
A.2.1.3 Produit cartésien . . . 162
A.2.1.4 Jointure . . . 162
A.2.1.5 Union . . . 163
A.2.1.6 Intersection . . . 163
A.2.1.7 Différence . . . 163
A.2.1.8 Agrégation . . . 163
A.2.2 Estimation du coût des opérateurs physiques . . . 163
A.2.2.1 Coût de la sélection . . . 165
A.2.2.2 Coût de projection . . . 165
A.2.2.3 Coût de jointure . . . 165
A.2.2.4 Coût de tri . . . 166
A.2.2.5 Coût des opérations ensemblistes . . . 167
A.2.2.6 Coût d'agrégation . . . 167

B Benchmarks et les requêtes d'apprentissage **169**
B.1 Introduction . . . 169
B.2 Requêtes d'apprentissage . . . 169
B.3 Requêtes TPC-DS utilisées . . . 175

Liste des figures **183**

Liste des tableaux **187**

Acronymes

BAT:	Binary Association Table
BD:	Base de données
CO_2:	Dioxyde de carbone
CPU:	Processeur
DdP:	Degré de parallélisme
DVFS:	Dynamics voltage fréquence scaling (Ajustement Dynamique de la Fréquence)
E/S:	Entrées/Sorties disque
EE:	Efficacité énergétique
FAR:	forêts aléatoires de régression
GDK:	Goblin Database Kernel
MAL:	MonetDB Assembly Language (Langage d'Assemblage de MonetDB)
MC:	Modèle de coût
McE:	Modèle de coût énergétique
NF:	Non Fonctionnel
NLR:	Régression Non Linéaire
OLAP:	Online Analytical Processing
OLTP:	OnLine Transaction Processing
RNA:	Réseaux de Neurones Artificiels
SF:	Scale factor (Facteur d'échelle)
SGBD:	Système de gestion de base de données
SPJ:	Sélection-Projection-Jointure
SSD:	Système de stockage des données
STR:	Système de traitement des requêtes

Première partie

Introduction Générale

CHAPITRE 1

Introduction Générale

La science consiste à oublier ce qu'on croit savoir, et la sagesse à ne pas s'en soucier.

— Novalis

Sommaire

1.1 Contexte et Problématique 15
1.1.1 La consommation d'énergie dans les SSDs 15
1.1.2 Impact environnemental et sanitaire 16
1.1.3 Apport de l'efficacité énergétique des SSDs 16
1.1.4 Vers des initiatives écologiques pour les SSDs 17
1.2 Notre vision de la caractérisation de l'énergie dans les BDs 18
1.3 Objectifs 20
1.4 Contributions de la Thèse 21
1.4.1 Taxonomie des approches d'efficacité énergétique 21
1.4.2 Étude comparative de la consommation énergétique 21
1.4.3 Définition des modèles de coût énergétique 21
1.4.4 Traitement éco-énergétique des requêtes 21
1.4.5 Analyse des compteurs de performance matérielle 22
1.5 Organisation du manuscrit 22
1.6 Publications 23

Résumé

1.1 Contexte et Problématique

Les systèmes de stockage des données (SSD) comme les data centers et les systèmes de gestion de base de données (SGBDs), jouent un rôle important dans notre vie quotidienne. Ils ont pour rôle de stocker et traiter des données massives et d'héberger des applications et des services repartis et à grande échelle. Ils sont omniprésents dans le fonctionnement des « Très Petites Entreprises » (*TPE*) et des « Grandes Entreprises » (*GE*) et constituent le socle de l'économie moderne. Aujourd'hui, les compagnies comme **Google**, **Amazon**, **Facebook**, **Apple** et **Intel**, copropriétaires des *data centers hyperscales* font partie des compagnies les plus rentables au monde. Cela est dû au fait qu'elles tirent profit de la demande croissante des besoins de stockage, de gestion et de traitement efficace des données. Cependant, avec l'explosion des données numériques et l'émergence des services tels que: les réseaux sociaux, les vidéos en continu, les jeux en ligne, les services de courrier, e-commerce, l'apprentissage automatique. ces SSDs consomment une quantité substantielle d'énergie électrique. Cela conduit à des coûts opérationnels exorbitants et à des émissions importantes de *Gaz à Effet de Serre* (GES). L'électricité consommée est principalement liée aux installations numériques dans les systèmes tels que les serveurs et les dispositifs de stockage et de communication. La grande quantité de chaleur émise par ces SSDs nécessite des systèmes de refroidissement pour permettre leur fonctionnement à une température adéquate.

Aujourd'hui, les SSDs sont devenus des gouffres énergétiques à cause de l'explosion massive des données. Ces dernières sont au cœur du nouvel ordre mondial et en même temps le nouveau polluant environnemental, comme mentionné dans le magazine The Economist [1] *«The world's most valuable resource is no longer oil, but data »*. Comme tout pétrole, les données polluent comme l'a mentionné Martin Tisné dans un blog [2] publié le 24 juillet 2019 *« Data isn't the new oil, it's the new CO_2 »*. Cette pollution est essentiellement causée par le stockage, le traitement et le partage des données.

1.1.1 La consommation d'énergie dans les SSDs

Pour soutenir le développement de la digitalisation numérique, les SSDs en tant qu'infrastructures de stockage et de traitement des données risqueraient de consommer encore plus d'énergie dans les jours à venir. Plusieurs statistiques intéressantes ont été publiées par des organismes publiques et privés sur la consommation des data centers que nous souhaitons partager afin de sensibiliser les producteurs et les consommateurs des SSDs sur les questions énergétiques. Un data center classique effectue une consommation énergétique équivalente à celle de 25 000 foyers. Sa consommation est 100 à 200 fois celle d'un bureau standard de taille similaire [139]. Selon l'Agence de Protection de l'Environnement (EPA) des États-Unis, les data centers auraient consommé environ 61 milliards de KiloWattHeures (kWh) d'énergie en 2006, soit $1,5\%$ de la consommation totale d'électricité des États unis[18]. En 2014, les auteurs dans [167] ont estimé que les data centers auraient consommé environ 70 milliards de kWh – ce qui représente $1,8\%$ de l'énergie totale consommée aux États-Unis. Selon la Commission Européenne (*CE*), la quantité d'énergie consommée par les data centers en Europe augmenterait de 21% en 2025 par rapport à la consommation faite en 2018 où les data centers auraient consommé $2,7\%$ de toute l'électricité produite [3]. En 2016, les data centers au niveau mondial auraient consommés $416,2$ milliards de kWh d'électricité, soit bien plus que la consommation totale du Royaume-Uni [4]. Aujourd'hui, on estime que les data centers consomment 200 TéraWattHeures (TWh) par an, ce qui est supérieur aux besoins énergétiques d'un ensemble de pays en Afrique. La consommation des data centers ne cesse de croître et tendent à devenir l'un des plus gros consommateurs d'énergie au monde avec un ratio qui s'élèvera de 3% en 2017 à $4,5\%$ en 2025 selon les études dans [104]. En termes d'émission de CO_2, les data centers y sont aussi responsables, leur empreinte carbone est similaire à celle du secteur aéronautique avec 2% des émissions totales de gaz à effet de serre [128].

L'éco-système d'un SSD est composé des équipements matériels (informatiques, climatisation, bâtiments hébergeant le SSD) et des logiciels permettant de garantir son fonctionnement et ses services. Selon les statistiques publiées dans [169], les équipements informatiques à eux seuls consommeraient environ 50% de l'énergie totale d'un SSD. La figure 1.1 donne des détails sur la répartition de cette énergie sur l'ensemble

1. https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data
2. https://luminategroup.com/posts/blog/data-isnt-the-new-oil-its-the-new-co2
3. https://www.datacenterknowledge.com/regulation/europe-edges-closer-green-data-center-laws
4. https://www.independent.co.uk/climate-change/news/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html

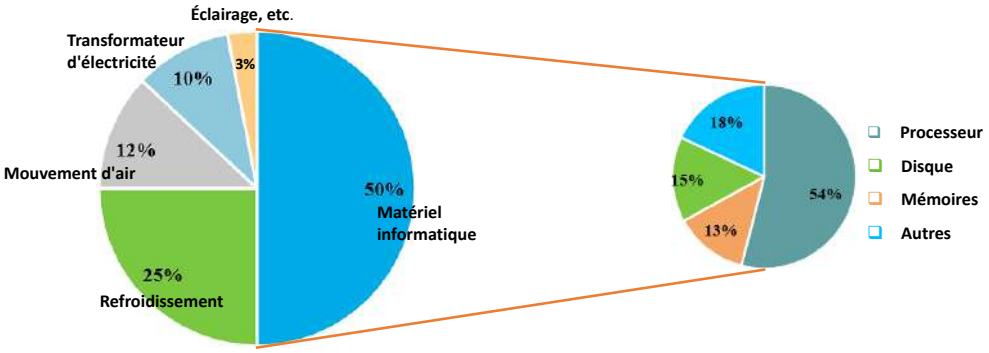


FIGURE 1.1 – répartition de la consommation énergétique dans un data center [157].

des composants informatiques. Ces statistiques montrent clairement que le processeur (CPU) consomme une grande partie de l'énergie, suivi par les dispositifs de stockage [157].

1.1.2 Impact environnemental et sanitaire

L'énergie est indispensable à la qualité de vie de l'humanité et contribue énormément au développement socio-économique. Néanmoins sa production et sa consommation émettent des gaz à effet de serre (comme le CO_2) responsables du dérèglement climatique. L'augmentation anormale de ce gaz dans l'atmosphère (due aux activités humaines comme la surconsommation énergétique), conduit au changement climatique et à l'avènement de nombreuses maladies. À défaut d'agir aujourd'hui pour atténuer ces émissions, on ferait peser des risques coûteux sur les générations futures concernant l'économie et des dégâts environnementaux (cyclones, canicules, typhons, insuffisance alimentaire, etc.).

Devant cette situation, tout citoyen, chercheur, entreprise, institution, gouvernement, État doivent *être préparé pour éviter ces risques*. Il est important de rappeler que certaines crises mondiales prennent de nombreuses personnes par surprise. Cela ne sera pas le cas des dégâts environnementaux liés au changement climatique. Depuis plusieurs décennies, des voies citoyennes, associatives, politiques alertent l'opinion publique et les dirigeants sur l'accélération de la dégradation environnementale. La crise sanitaire du *Covid-19* en est un bon exemple de la mauvaise préparation de l'humanité face à des situations dramatiques. Le seul moyen d'éviter le pire insoluble dans le cas d'une crise environnementale serait de repenser nos modes de vie, en respectant plus la nature dans son fonctionnement normal. Une autre piste intéressante concerne la prioritisation des exigences non fonctionnelles des citoyens et des chercheurs. Souvent la performance des accès aux services informatiques est la pré-condition pour les acquérir. Cette exigence peut être substituée ou couplée avec la consommation énergétique de ces services. La pandémie de la *Covid-19* nous a plus au moins poussé à intégrer ces priorités dans nos vies quotidiennes et surtout a légèrement contribué à la réduction de cette consommation grâce aux restrictions de l'activité économique, du trafic aérien, terrestre et maritime, ainsi que la fermeture d'industries (voir figure 1.2). En même temps, le travail chez soi imposé par cette pandémie a généré une utilisation massive des ordinateurs et en conséquence une augmentation de la consommation énergétique.

Devant cette situation, en tant que enseignant/chercheurs en informatique, nous sommes dans l'obligation de fournir des solutions moins énergivores en termes de composants matériels et logiciels qu'il faudrait développer afin d'assurer une transition énergétique.

1.1.3 Apport de l'efficacité énergétique des SSDs

L'efficacité énergétique d'un SSD est l'énergie totale consommée par rapport à la quantité d'énergie nécessaire à son fonctionnement. Hormis les apports pécuniers, elle est l'un des outils principaux pour réduire les émissions de CO_2 et de répondre facilement à la demande croissante d'énergie. Il peut également y avoir de nombreux autres impacts socio-économiques et sanitaires, tels que des maisons vertes, la longévité des biens et des personnes, la réduction des factures d'électricité et des coûts de maintenance. La réduction de la consommation d'énergie garantit la longévité des composants électroniques. En fait, la dissipation d'énergie a un effet néfaste sur la fiabilité des circuits électroniques [119]. Le dysfonctionnement ainsi que la détérioration des composants sont causés par de nombreux facteurs, notamment par une forte

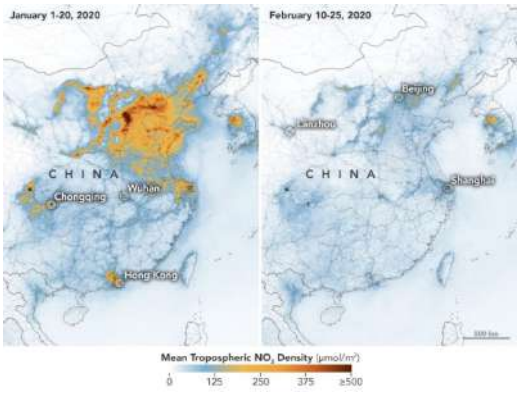


FIGURE 1.2 – Dioxyde d'azote(NO_2) mesurée par la NASA - [SOURCE=[137]]

élévation de la température, une surtension aux bornes des dipôles électriques, etc. Le taux de défaillance des équipements augmente lorsque sa température dépasse 15° Celsius [28].

1.1.4 Vers des initiatives écologiques pour les SSDs

Toutefois, pour pallier au problème écologique tout en continuant à fournir la demande de service en matière de gestion des données, les différents acteurs du secteur des SSDs, organisations, associations, scientifiques, industriels soutenus par les gouvernements ont proposé des initiatives afin d'améliorer l'efficacité énergétique des infrastructures. Ces initiatives visent à: (i) concevoir des équipements moins gourmands en énergie, (ii) éco-concevoir les logiciels, (iii) établir une gestion efficiente des ressources disponibles et (iv) contrôler et monitorer l'éco-système d'un SSD [83]. L'informatique verte est apparue comme une nouvelle tendance pour encourager les transitions. En effet, les ordinateurs, y compris leurs composants (matériels et logiciels), et le comportement de leurs utilisateurs contribuent de manière intensive au phénomène du réchauffement climatique. Le stockage et le traitement des données sont des sous-systèmes importants de tout SSD. Cela s'est accentué en raison de l'avènement de nouveaux paradigmes tels que le cloud computing et la virtualisation [31]. De nos jours, les bases de données (BDs) sont l'épine dorsale du fonctionnement de toutes les entreprises, des agences gouvernementales et des fournisseurs de service de stockage de données. Dans un SSD, il a été remarqué que les systèmes de traitement des données (STRs) sont l'un des composants les plus énergivore durant leurs activités complexes de traitement et de stockage des données [139]. Cette importance a été largement soulignée dans le rapport de Claremont [4] publié en 2008 et cosigné par environ une trentaine de chercheurs, experts et architectes du monde des bases de données. Ce rapport a mis en avant la nécessité de concevoir des SGBDs sensibles à l'énergie qui limitent les coûts énergétiques sans sacrifier le passage à l'échelle. Ce point est également repris dans le rapport de Beckman [2] publié en 2014 qui place l'efficacité énergétique comme un défi à relever dans le domaine des données massives. L'intégration de l'énergie dans les STRs concerne tous ses composants et ses acteurs, notamment les fournisseurs de matériel et de logiciels, les concepteurs, les administrateurs des bases de données et les utilisateurs finaux, etc. Il est important de rappeler que le processus traditionnel de stockage et de traitement des données est guidé par la minimisation du temps de réponse lors du traitement des données[46]. Les rapports ci-dessus cités ont recommandé de revisiter cette vision traditionnelle afin d'intégrer l'efficacité énergétique.

La proposition des approches moins gourmandes en énergie dans les BDs a été l'objet de plusieurs travaux. Les années 2010 ont été une décennie marquée par le début de l'intégration de l'énergie dans la conception des optimiseurs de requêtes où des approches matérielles et logicielles ont été analysées, implémentées et évaluées [96]. Les solutions matérielles préconisent l'exploitation des matériels modernes qui consomment moins d'énergie et les solutions logicielles recommandent de revisiter les stratégies d'exécution des requêtes, l'ordonnancement des tâches, les techniques de conception physique et de gestion des données afin de respecter le temps de réponse et permettre une meilleure gestion de l'énergie. Dans ce sens deux papiers d'orientation ont été publié par des chercheurs venant du milieu académique (Stavros Harizopoulos et al. [62] de MIT [5]) et du milieu industriel (Goetz Graefe [54] de HP [6]). Ces papiers mettent

5. Massachusetts Institute of Technology
6. Hewlett-Packard

également l'accent sur l'intégration de la contrainte énergétique dans la conception et l'exploitation des STRs.

En examinant les travaux existants dans le monde des BDs, nous identifions qu'ils sont dominés par les solutions orientées *Hardware*. Notre analyse de l'évolution des articles scientifiques publiés sur les approches d'intégration de l'énergie dans les BDs de 1994 à 2018 confirme cette tendance, où plus de 100 papiers se sont concentrés sur les approches matérielles, alors que seulement 33 se sont confrontés aux approches logicielles. Cela s'explique par le fait que plusieurs études considèrent que les systèmes d'exploitation et les micrologiciels (drivers) doivent gérer l'efficacité énergétique, par conséquent économisent l'énergie des optimiseurs des requêtes. Les solutions logicielles explorées ne proposent pas une démarche de conception détaillée et facile à reproduire des processeurs des requêtes permettant l'intégration de la dimension énergétique et ignorent la variabilité des différentes classes des systèmes de stockage des données (SSDs) lors de la définition de leur modèle de coût. En dehors des apports écologiques et pécuniers, ce constat constitue l'une des motivations principales de cette thèse.

1.2 Notre vision de la caractérisation de l'énergie dans les BDs

La caractérisation de l'efficacité énergétique dans les BDs est une tâche difficile. Elle nécessite l'identification des composants ainsi que leurs paramètres qui impactent la consommation énergétique. Dans cette thèse, nous nous focalisons sur l'un des composants les plus importants d'une BD à savoir le **processeur de traitement de requêtes**. Ce dernier reçoit en entrée une requête au format textuel écrite en SQL. Une fois reçue, elle est analysée ensuite optimisée. Cette optimisation est assurée par le choix des algorithmes efficaces implémentant les opérations de base et des méthodes d'accès adéquates. L'optimiseur de requêtes est l'un des modules importants d'un STR. Il a pour rôle de fournir: (i) un espace de recherche des plans d'exécution de la requête, (ii) des estimations de coût de toutes les opérations composant la requête, et (iii) une stratégie d'exploration de cet espace afin de sélectionner le meilleur plan selon le besoin non fonctionnel de l'étude [39]. La figure 1.3 illustre les principaux composants d'un optimiseur de requêtes traditionnel.

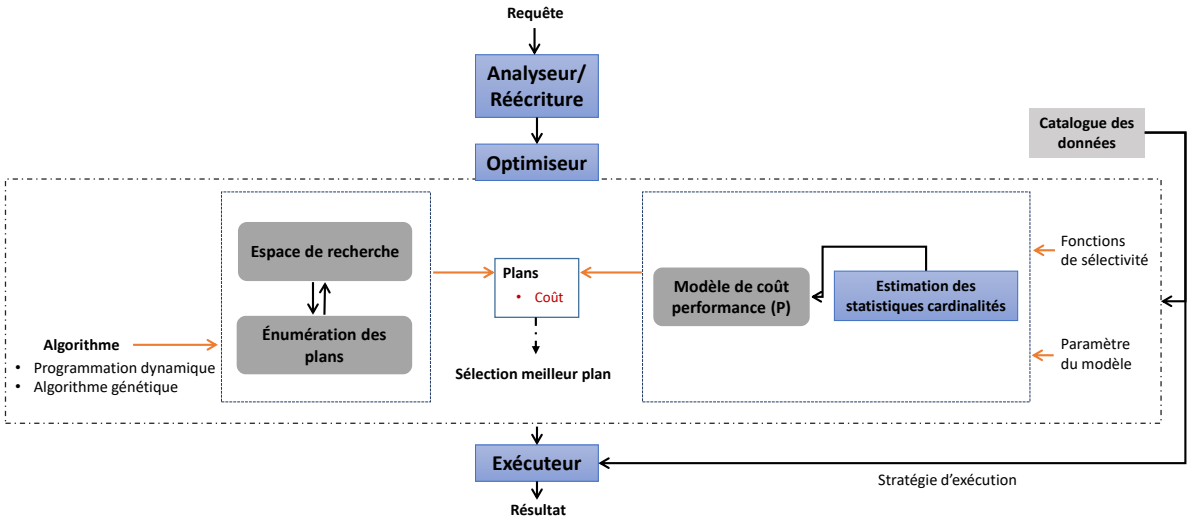


FIGURE 1.3 – Architecture d'un optimiseur de requête.

Il est important de rappeler que les STRs actuels sont conçus pour optimiser les accès aux données. Pour caractériser l'énergie d'un STR, deux visions principales sont possibles: (i) **une vision dirigée par les tests réels**, (ii) et celle **dirigée par des modèles de coût analytiques**. La première vision consiste à tester une charge de requêtes définie sur une BD déployée sur une infrastructure. Cette dernière inclut le SSD et sa plateforme. Avec des outils de mesure d'énergie adaptés, chaque requête de cette charge aura son coût énergétique. Cette vision nécessite à la fois l'achat du SSD, la plateforme et la mise en place de la batterie de tests par des testeurs. De plus, elle n'offre pas la possibilité d'intégrer l'énergie dans la phase d'optimisation. (ii) La deuxième vision dirigée par des modèles de coût mathématique permet de simuler le fonctionnement du module en question afin de proposer des métriques qui correspondent aux besoins de l'énergie. La majorité des STRs des SSDs actuels sont presque tous basés sur cette vision pour

satisfaire leurs besoins non fonctionnels tels que la performance des requêtes. La caractérisation de cette performance est établie par des outils normalisés déployés sur les SSDs. Ces outils fournissent les coûts des ressources utilisées par la requête donnée. Le tableau 1.1 donne un aperçu de l'outil «Explain» de PostgreSQL où le coût *cost* décrit le coût en performance des opérations de la requête:

```
SELECT  n_name FROM region, nation
WHERE n_regionkey = r_regionkey AND r_name = 'AFRICA'
```

TABLEAU 1.1 – Aperçu du plan d'exécution généré par par l'outil EXPLAIN (PostgreSQL)

	Plan de la requête
1	Hash Join (cost=12.14..24.48 rows=1 width=104)(actual time=0.023..0.027 rows=5 loops=1)
2	Hash Cond: (nation.n_regionkey = region.r_regionkey)
3	− > Seq Scan on nation (cost=0.00..11.70 rows=170 width=108) (actual time=0.009..0.010 rows=25 loops=1)
4	− > Hash (cost=12.12..12.12 rows=1 width=4)(actual time=0.007..0.007 rows=1 loops=1)
5	Buckets: 1024 Batches: 1 Memory Usage: 9kB
6	− > Seq Scan on region (cost=0.00..12.12 rows=1 width=4) (actual time=0.005..0.006 rows=1 loops=1)
7	− > Filter: (r_name = 'AFRICA'::bpchar)

Dans cette thèse, nous considérons cette deuxième vision afin d'intégrer l'efficacité énergétique dans les STRs. Cette vision ne peut pas être appliquée directement, car elle est associée à une question primordiale: *comment exploiter l'expertise de développement des modèles de coût souvent dédiés à la caractérisation de la performance des requêtes pour l'appliquer à la caractérisation de l'énergie dans les STRs?*

Deux pistes de réflexions sont possibles pour répondre à cette question en prenant en compte trois aspects fondamentaux (nommés *ETAC*) qui sont : l'effort fourni, le temps consacré et surtout les arguments pour convaincre les acteurs des SSDs à repenser leur conception.

1 Cette solution consiste à substituer la propriété non fonctionnelle qui est la performance des accès des requêtes par la consommation de l'énergie et suivre toutes les étapes de développement des modèles de coût énergétique. Rappelons que la consommation énergétique dépend de plusieurs aspects: le chargement des données du disque vers la mémoire (en cas des systèmes orientés disque), le temps CPU et le coût de communication (réseau). Développer des modèles de coût énergétique *à partir de zéro* pour chaque aspect est une tâche très lourde, car elle nécessite un travail de fourmis pour définir ou adapter les modèles de coût existants. Cela passe par une étude exhaustive de tous les modèles de coût existants. Le monde des STRs est très complexe du fait qu'ils intègrent des informations diverses et variées liées à la BD, la charge des requêtes, le SSD, la plateforme de déploiement, les algorithmes implémentant les opérations de base, les statistiques, etc. Cette solution a été rapidement écartée selon les aspects ETAC.

2 La deuxième solution consiste à réutiliser les modèles de coût existants qui ont prouvé leur efficacité et leur usage. Ces modèles concernent les briques fondamentales de l'énergie à savoir le coût CPU et les entrées-sorties. Nous avons privilégié cette réponse car elle permet rapidement d'assurer cette caractérisation et surtout avoir des résultats pour *convaincre* les consommateurs et les producteurs des SSDs de l'intérêt d'intégrer l'énergie dans les STRs. Afin d'assurer cette reproduction de l'expérience autour des modèles de coût nous nous sommes focalisés sur des SSDs existants libres comme *PostgreSQL*, *MonetDB* et *Hyrise* conçus initialement pour satisfaire la performance des requêtes. Pour chaque système, nous avons ajouté un modèle de coût énergétique construit à partir de ses modèles existants. Ce modèle permet de guider l'optimiseur de requêtes pour sélectionner les plans de requêtes ayant un coût énergétique raisonnable. Le fait que nous avons gardé le modèle de coût initial de chaque système, nous permet de proposer des techniques de sélection de plans de requêtes assurant un compromis entre les deux propriétés non fonctionnelles (la performance de requêtes et la consommation énergétique). La figure 1.4, illustre le fonctionnement de l'optimiseur après l'intégration de la dimension énergétique.

Cette réponse permet de déléguer certaines tâches (comme l'énumération de l'espace de recherche, les stratégies d'exploration des plans) aux optimiseurs de requêtes existants afin de *bénéficier de leur expérience* et *de simplifier le processus d'intégration de l'efficacité énergétique dans les SSDs*. Plus précisément, nous n'avons pas altéré le principe de génération de l'espace de recherche, ni l'énumération des plans de l'optimiseur. Pour la sélection du meilleur plan, nous nous referons à un compromis entre les deux grandeurs comme illustrée sur la figure 1.4.

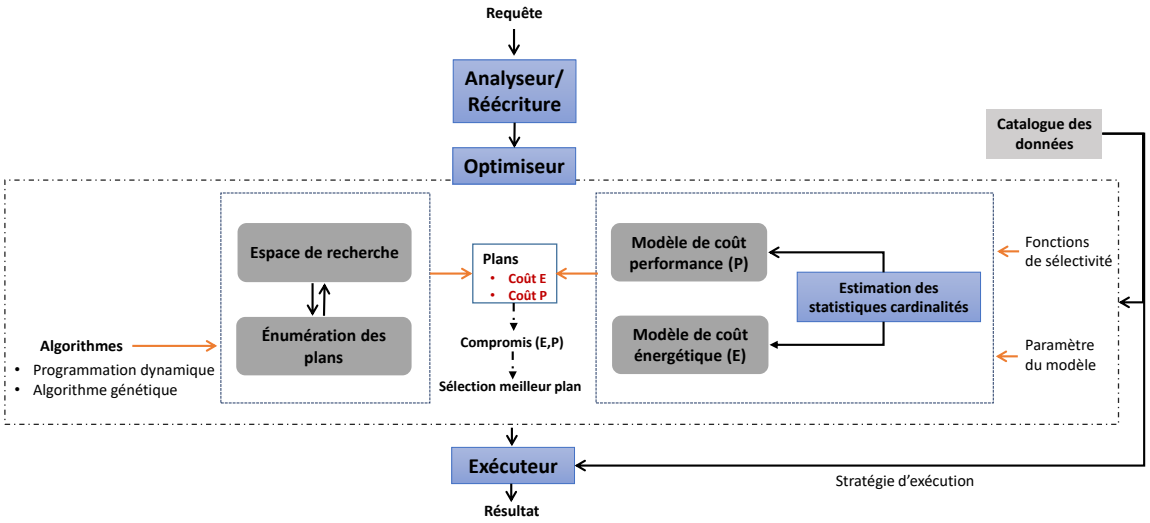


FIGURE 1.4 – Architecture d'un optimiseur de requête (avec la dimension énergétique).

1.3 Objectifs

Notre approche orientée logicielle est fondamentalement basée sur l'utilisation des modèles de coût énergétique. L'un des principaux objectifs de notre travail est donc de proposer des modèles capables d'estimer la consommation énergétique d'un SSD lors de l'exécution d'une requête. Ces modèles doivent incorporer les attributs sensibles à la consommation énergétique, tels que la charge de requêtes, les architectures matérielles et logicielles, les statistiques de la base de données, les algorithmes implémentant des opérateurs (sélection et jointure) de requête. Les contributions majeures des modèles obtenus sont: **(1)** l'estimation de l'efficacité énergétique d'un STR d'un SSD déployé sur une architecture particulière, **(2)** la sélection d'un plan d'exécution de requête moins énergivore, **(3)** le choix d'un SSD vert pour une application donnée intégrant une BD et une charge de requêtes, **(4)** la sélection des structures d'optimisation physiques comme les index et les vues matérialisées, et **(5)** le développement des advisors (wizards) pour assister les administrateurs de BDs dans leurs tâches quotidiennes afin d'intégrer l'énergie consommée dans les modules qu'ils gèrent (qui concernent l'administration et le tuning des BDs).

Afin de réaliser les objectifs ci-dessus, notre travail de thèse a consisté en cinq étapes consécutives basées sur une analyse quantitative des études existantes et notre volonté de rendre générique ces étapes afin de faciliter leur reproduction pour tout SSD. Ces étapes sont résumées comme suit:

1. **Audit du processeur des requêtes:** Lors de cette étape, nous cherchons à comprendre le comportement énergétique des STRs traditionnels et modernes afin de découvrir les points d'intégration possible de la dimension énergétique. Nous avons également identifié un certain nombre de facteurs potentiels qui peuvent influencer la consommation d'énergie de ces systèmes.

2. **Taxonomie:** Nous avons exploré un grand nombre de travaux dédiés à la consommation énergétique dans les SSD. Cette exploration nous a permis de définir une feuille de route permettant d'orienter les enseignants-chercheurs et les étudiants dans leurs missions afin d'intégrer l'énergie dans l'enseignement et la recherche.

3. **Approche de modélisation:** Nous avons proposé des modèles de coût basés sur des études empiriques pour estimer l'énergie consommée par les STRs pendant l'exécution des requêtes. Cette modélisation considère trois SSD libres qui sont PostgreSQL, MonetDB et Hyrise.

4. **Validation:** Nous proposons ensuite une étape de validation de notre approche de modélisation en utilisant des données réelles de référence et trois SSD.

5. **Déploiement:** En dernière étape, nous avons déployé nos modèles de coût dans le SSD pour des fin d'estimation de la consommation énergétique des requêtes ou pour des études d'optimisation énergétique.

1.4 Contributions de la Thèse

Nos contributions portent sur les points suivants: (i) la proposition d'une taxonomie des approches d'efficacité énergétique dans les bases de données, (ii) une étude comparative de la consommation énergétique de différentes classes de systèmes de stockage des données, (iii) la définition des modèles de coût énergétique, (iv) le traitement éco-énergétique des requêtes, et (v) l'exploitation des compteurs de performance matériel pour modéliser le coût énergétique dans les BDs en mémoire.

1.4.1 Taxonomie des approches d'efficacité énergétique

L'objectif de notre travail étant d'améliorer l'efficacité énergétique dans les systèmes de traitement des requêtes, il nous a semblé important de porter une analyse approfondie sur les travaux existants pour comprendre les techniques utilisées dédiées à l'amélioration de l'efficacité énergétique. Sur la base de ce travail, nous avons proposé une feuille de route décrivant les importants travaux selon trois catégories: (a) les approches basées sur la gestion de l'énergie des composants matériels, (b) les approches basées sur les techniques logicielles et (c) les approches basées sur les techniques de gestion environnementale et réglementaire.

1.4.2 Étude comparative de la consommation énergétique

Avant toute étude d'efficacité énergétique, les utilisateurs doivent être conscients de la consommation énergétique de leur SSD par rapport à d'autres qui peuvent effectuer les mêmes fonctions. Dans cet objectif, nous avons mené une étude comparative qui évalue le temps et la puissance énergétique de différents SSDs en utilisant les mêmes charges de travail dans les mêmes conditions d'environnement d'exécution. Nous avons porté notre attention sur le système le plus énergivore afin d'expliquer la raison de cette forte consommation par rapport aux autres. Les observations et les résultats tirés de cette étude soulignent la variabilité entre les SSD pour une même charge de travail donnée. Les résultats nous montrent également qu'il est difficile d'utiliser un seul modèle énergétique pour différentes classes de STR et aussi que l'administrateur de base de données doit prendre des conseils en matière d'efficacité énergétique avant le déploiement de ses données.

1.4.3 Définition des modèles de coût énergétique

Nous avons proposé une méthodologie complète pour construire un modèle de coût énergétique lors de l'exécution des requêtes au niveau des SSDs. Nous avons ensuite utilisé notre méthodologie pour proposer plusieurs modèles énergétiques pour trois SSD (PostgreSQL, MonetDB, et Hyrise). Ces systèmes libres offrent des politiques de fonctionnement différentes: *PostgreSQL* est un SSD orienté ligne, *MonetDB* est un SSD orienté colonne tandis que *Hyrise* est un SSD totalement en mémoire principale, qui utilise la technique de stockage hybride [7]. La définition des ces modèles de coût a nécessité un travail important pour estimer les valeurs de certains paramètres. Cette estimation a été menée à l'aide des algorithmes d'apprentissage automatique (Machine learning). Contrairement à la majorité des articles lus dans ce domaine, nous avons constaté que l'algorithme de la régression linéaire simple n'était pas efficace pour construire des modèles de coût précis sur aucun des systèmes que nous avons étudié dans cette thèse. Dans notre étude, nous avons utilisé trois algorithmes d'apprentissage automatique : *la régression non-linéaire*, *les réseaux de neurones artificiels* et *l'algorithme des forêts aléatoires*. Pour évaluer la précision de nos modèles, nous avons réalisé des expériences approfondies sur un banc d'essai physique basé sur les systèmes PostgreSQL, MontetDB et Hyrise en utilisant des charges de travail générées par différents bancs. Cette évaluation nous a permis de valider notre proposition et de confronter la précision des différents modèles d'apprentissage automatique.

1.4.4 Traitement éco-énergétique des requêtes

Lors de la phase d'audit, nous avons exploré les différentes étapes du processus de traitement des requêtes afin de dégager les points d'intégration possibles de la dimension énergétique pour chaque système. Les étapes d'optimisation sont celles qui impactent énormément sur la consommation énergétique du système lors de l'exécution de la requête. Pour le système PostgreSQL, nous avons intégré notre modèle

7. Le stockage hydride cumule les avantages des stockages de données orienté ligne et colonne

dans le noyau de l'optimiseur en proposant un modèle d'évaluation des plans ayant pour objectif de sélectionner un plan qui optimise la consommation énergétique. Pour MonetDB, nous avons identifié trois points d'intégration possible de l'énergie situé au niveau des trois moteurs d'optimiseur du système et pour Hyrise, nous avons mené une étude d'évaluation du modèle.

1.4.5 Analyse des compteurs de performance matérielle

Pour les SSDs totalement en mémoire, dans le but d'avoir un modèle de coût facilement exportable soit d'un SSD en mémoire à un autre ou d'une architecture à une autre, nous avons proposé un modèle de coût énergétique construit sur la base de l'analyse des compteurs de performance du système lors de l'exécution des requêtes. Vu que les SSDs totalement en mémoire sont dispensés des coûts de chargement des données à partir du disque, cette approche révèle plus de détails sur le comportement énergétique du système lors de l'exécution de la requête. Ensuite, expérimentalement, nous avons montré que notre processus de sélection des évènements des compteurs et d'implémentation avec le système *Hyrise* sont valides en exécutant des requêtes de plusieurs benchmarks avant d'exporter notre modèle sur deux SSDs en mémoire (HSQLDB [8], Hypersonic 2 (H2) [9]) pour des études d'estimation énergétique des requêtes.

1.5 Organisation du manuscrit

Le reste de la thèse est constitué de 6 chapitres (figure 1.5)

— **Le chapitre 2** présente un audit de fonctionnent des STRs plus particulièrement en se focalisant plus sur le modèle relationnel. Les différentes étapes de traitement d'une requête sont présentées et discutées allant de l'architecture de déploiement homogène (centralisée, distribuée) aux architectures complexes hétérogènes (multi-bases, multi-stores). Ce chapitre décrit également les principaux facteurs qui peuvent influencer la consommation énergétique du système lors de l'exécution des requêtes partant des caractéristiques de la charge des requêtes aux architectures de déploiement.

— **Le chapitre 3** présente une taxonomie sur les différentes techniques de réduction de la consommation énergétique au niveau des SSDs en se focalisant plus sur la gestion de l'énergie dans les STRs. Les différents concepts requis ainsi que les notions de base sur l'énergie et les modèles de coût énergétique y sont présentés. Ce chapitre aborde aussi brièvement les impacts environnementaux et sanitaires causés par une émission excessive de CO_2 et les apports de l'efficacité énergétique.

— **Le chapitre 4** aborde le problème de la modélisation des coûts énergétiques des opérateurs constituant le plan d'exécution d'une requête. Nous avons étudié le comportement de l'énergie de différents SSDs puis nous avons présenté notre approche de conception des différents modèles de coût énergétique. Nous proposons quatre modèles de coût: deux modèles pour les requêtes exécutées en mode parallèle sur le système PostgreSQL, un modèle énergétique pour les systèmes orientés colonnes (MonetDB) et un modèle pour les systèmes en mémoire (Hyrise). Ce chapitre comporte plusieurs résultats expérimentaux commençant par une étude comparative de la consommation énergétique des SSDs étudiés et allant jusqu'à la validation de nos modèles de coût proposés sur différents bancs d'essai.

8. `http://hsqldb.org/`
9. `https://www.h2database.com/html/main.html`

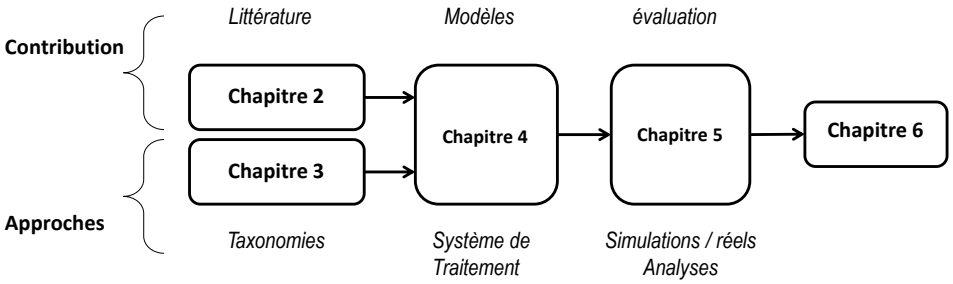


FIGURE 1.5 – La répartition des chapitres de la thèse.

— **Le chapitre 5** présente notre processus de déploiement de nos modèles validés dans des études d'optimisation ou d'estimation énergétique. Les étapes du processus de traitement des requêtes sont analysées afin de trouver les points d'intégration possible de nos modèles. Sur la base de cette analyse, différents Frameworks qui intègrent nos modèles de coût énergétique sont proposés.

— **Le chapitre 6** résume notre thèse et propose les perspectives qui s'offrent à l'issue de notre travail.

— Ce manuscrit comprend **deux annexes**. L'Annexe A contient le processus d'estimation des coûts dans la phase d'optimisation des requêtes dans un SGBD orienté ligne. Les formules d'estimation des cardinalités des résultats intermédiaires et celles des coûts d'entrées/sorties des opérateurs y sont détaillées. L'Annexe B présente les bancs d'essai et requêtes utilisés pour l'apprentissage et la validation.

1.6 Publications

1. **Revues internationales**
   (a) **Simon Pierre Dembele**, Ladjel Bellatreche, Carlos Ordonez, Amine Roukh, Think big, start small: a good initiative to design green query optimizers, Cluster Computing, Springer, 23(3), 2020, pp. 2323-2345, DOI: 10.1007/s10586-019-03005-0. [IF : 3.458]
   (b) **Simon Pierre Dembele**, Ladjel Bellatreche, Carlos Ordonez, Big Steps Towards Query Eco- Processing - Thinking Smart. Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées, Arima - episciences, Vol. 34, 2021.
2. **Conférence internationale:**
   (a) **Simon Pierre Dembele**, Ladjel Bellatreche, Carlos Ordonez, Towards Green Query Processing - Auditing Power Before Deploying (Energy-Efficient ML & Big Data Workshop), IEEE International Conference on Big Data (IEEE Big Data), pp. 2492-2501, Atlanta, GA, USA, December 10-13, 2020.

3. **Conférences nationales:**
   (a) **Simon Pierre Dembele**, Ladjel Bellatreche, Carlos Ordonez, Vers le traitement et optimisation écologique des requêtes, Colloque Africain sur la Recherche en Informatique et en Mathématiques Appliquées (CARI), Thiès, Sénégal, Oct. 2020.
   (b) **Simon Pierre Dembele**, Amine Roukh, Ladjel Bellatreche, Vers des Optimiseurs Verts de Requêtes en Mode Parallèle, Entrepôt de données et Analyse en ligne: BI and Big Data (EDA 2018), edited by Revue de New Technologies de l'Information (RNTI), 2018, pp. 179–194.

Deuxième partie

État de l'art

CHAPITRE 2

Les Systèmes de traitement des requêtes (STRs)

L'Éternel marchera lui-même devant toi, il sera lui-même avec toi. Il ne te délaissera pas, il ne t'abandonnera pas. N'aie pas peur et ne te laisse pas effrayer.

— Deutéronome 31:8

Sommaire

2.1 Introduction 29
2.2 Les STRs dans les SSDs 29
2.2.1 Architecture générique des STRs 29
2.2.1.1 Formulation de la requête 30
2.2.1.2 Compilation 30
2.2.1.3 Optimisation 30
2.2.1.4 Moteur d'exécution 31
2.2.2 Optimisation des requêtes 31
2.2.2.1 STRs dans les systèmes de stockage centralisés 31
2.2.2.2 STRs dans les systèmes de stockage distribués 40
2.2.2.3 STRs dans les systèmes multi-bases 41
2.2.2.4 STRs dans les systèmes multi-stores 41
2.3 Modèle de coût 43
2.4 Les paramètres sensibles à la dimension énergétique 44
2.4.1 Types de la Requête 44
2.4.2 Types des SSDs 44
2.4.3 Caractéristiques matérielles 45
2.4.4 Configurations architecturales 45
2.4.5 Influences environnementales 46
2.5 Conclusion 46

Ce chapitre donne un aperçu sur le fonctionnement des STRs dans le monde des BDs relationnelles. Les différentes étapes de traitement d'une requête sont présentées et discutées en prenant en compte les aspects suivants: les architectures de déploiement homogène (centralisée, distribuée), et les architectures complexes hétérogènes (multi-bases, multi-stores). Ensuite, nous présentons les principaux facteurs impactant la consommation énergétique des SSDs lors de l'exécution des requêtes.

2.1 Introduction

Les BDs, depuis leur apparition à nos jours constituent une discipline fondamentale de la science de l'informatique. Elles constituent un socle indispensable au fonctionnement de nombreuses entreprises où la performance des systèmes d'information et la prise de décision sont des enjeux stratégiques. Elles sont omniprésentes dans de nombreuses applications et enseignées dans pratiquement toutes les filières universitaires. Depuis plus de quatre décennies, elles n'ont guère cessé de s'étendre pour s'adapter aux besoins des utilisateurs. Cette expansion provient d'un ensemble de connaissances et de technologies développées au cours des années en proposant de nouveaux modèles de données (modèle hiérarchique, relationnel, orientée-objet, multidimensionnel, etc.), des modèles de stockage (stockage en ligne, en colonne et mixte), des supports de stockages (HDD, flash, DRAM, etc.), des modèles d'accès (séquentiel, aléatoire, parallèle, etc.), des modèles de traitement des données et des plateformes de déploiement (centralisées, distribuées, parallèle, Cloud, etc.). Un SSD est un logiciel spécialisé englobant plusieurs technologies facilitant la création et la gestion des BDs. En plus de fournir un modèle de stockage sûr et persistant, les SSDs permettent aux utilisateurs ou aux programmes d'accéder aux données stockées à travers des langages de requête comme le SQL (Strutured Query Language). Ainsi ils dissimulent complètement toute la complexité du traitement des requêtes au sein des SSDs en interprétant et exécutant les opérations requises pour retrouver les données demandées par l'utilisateur dans la requête exprimée. Au sein des SSDs, le traitement des requêtes est une tâche complexe, car d'une part il a pour tâche d'accéder et de manipuler les données massives d'une manière efficace et d'autre part il interagit avec d'autres phases de traitement des requêtes qui sont liées à la sélection et maintenance des structures d'optimisation comme les méthodes d'indexation.

Un STR est un ensemble de composants dans le SSD qui transforme une requête émise en une séquence d'opérations puis les exécute. Le développement de tout STR est réalisé pour satisfaire des besoins non fonctionnels comme la performance des temps d'accès, la sécurité, ou la consommation énergétique.

2.2 Les STRs dans les SSDs

Avec la recrudescence de la taille des informations, il fut nécessaire d'améliorer l'efficacité des techniques et des algorithmes impliqués dans le traitement des requêtes. Dans le processus de traitement des requêtes, la phase d'optimisation est vue comme la plus complexe, qu'il s'agisse d'un SSD relationnel, ou d'un système NoSQL. L'étape d'optimisation permet de trouver un plan (succession d'opérateurs) considéré comme optimal pour exécuter la requête. Malgré que de nombreux efforts aient été faits dans le domaine de l'optimisation, certains défis sont toujours discutés dans la littérature à nos jours. Dans cette section, nous présentons l'architecture générique du système de traitement des requêtes et des techniques d'exécution des requêtes qui sont communes à tous les SGBDs. Puis, nous examinerons en détails le processus de traitement des requêtes pour certaines architectures de déploiement des SSDs.

2.2.1 Architecture générique des STRs

Les STRs font référence à un ensemble d'activités impliquées dans l'extraction des données stockées sur les mémoires secondaires. Le STR est un élément fondamental dans le fonctionnement de tout SSD. Il représente la vitrine par laquelle les programmes ou les utilisateurs accèdent aux données pour des fins de restitutions ou de mise à jour. Pour s'acquitter efficacement de sa fonction, le STR doit être capable de fournir les résultats des requêtes dans un délai requis (défini) ou raisonnable. Ainsi le STR doit produire des plans efficaces qui minimisent l'utilisation des ressources du système tout en s'assurant que les résultats générés sont corrects. Pour traiter une requête, le STR doit exécuter différentes étapes sur les modules qui le composent tels que le compilateur, l'optimiseur, l'exécuteur et le gestionnaire de stockage [46]. La figure 2.1 illustre l'architecture générique des STRs.

Avant que le processus de traitement de la requête ne commence, le système doit lire une requête exprimée dans un langage haut niveau compréhensible par l'utilisateur (le cas du SQL) et la traduire sous une forme exploitable par le système. Cette étape de traduction est similaire au travail effectué en amont par un compilateur à travers son analyseur. Puis l'étape d'optimisation, lors de laquelle le module sélectionne un plan d'exécution qui répond aux exigences des utilisateurs. L'optimiseur génère un ensemble de plans d'exécution et en sélectionne un, ayant le moindre coût d'exécution. Le coût d'un plan d'une requête est estimé à partir des informations disponibles dans le dictionnaire des données [171].

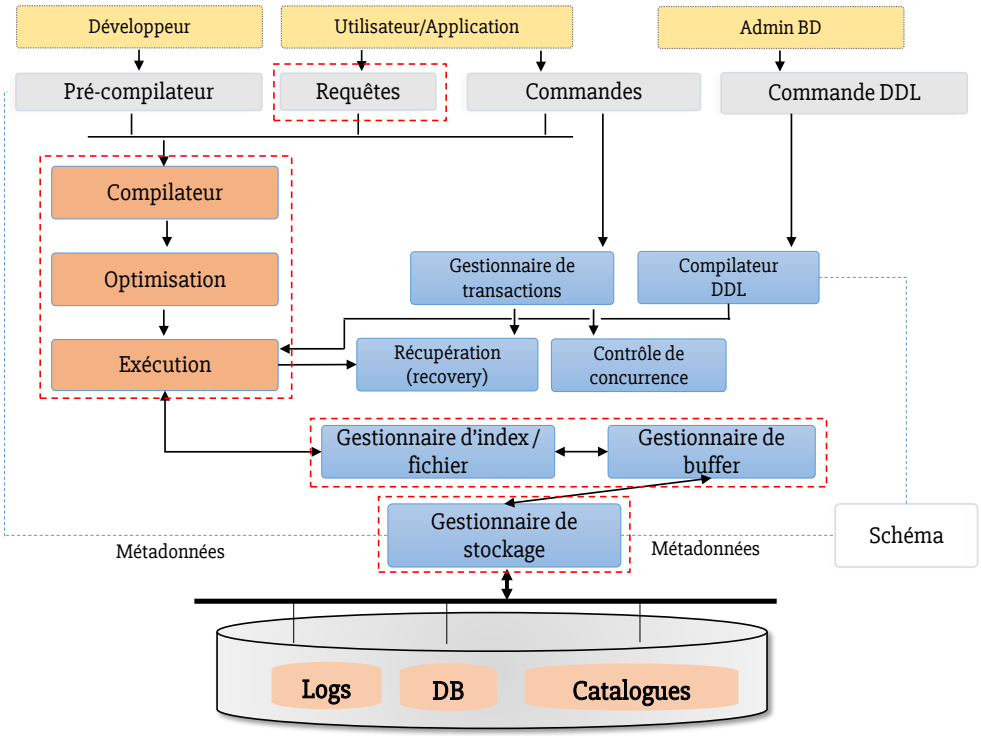


FIGURE 2.1 – Architecture générique des STRs.

Dans l'étape finale, le moteur d'exécution évalue les opérations du plan d'exécution retenu dans l'étape d'optimisation en extrayant les données des sources de stockage (mémoires secondaires).

2.2.1.1 Formulation de la requête

Les utilisateurs interagissent avec les SSDs en soumettant des demandes d'accès aux données, ces demandes sont dites des requêtes. Les requêtes expriment les besoins de restitution des données ou de mise à jour. Les requêtes sont écrites dans un langage de haut niveau, compréhensible et facile pour l'humain, le plus utilisé est le langage *SQL* [172]. Le succès de beaucoup de SGBDs repose sur le fait que les utilisateurs peuvent aisément formuler leurs requêtes à travers des langages d'interrogation (exemple : SQL) intégrés d'une manière intuitive sans avoir à se soucier de la manière dont les données sont stockées ou extraites des mémoires secondaires. Ainsi l'une des premières tâches d'un STR est de fournir un module ayant pour fonction de s'assurer de la bonne formulation de la requête de l'utilisateur et ensuite la traduire en une forme compréhensible par le système.

2.2.1.2 Compilation

Cette étape consiste à transformer la requête exprimée par l'utilisateur à une forme interne facilitant son traitement par le STR. La forme interne est générée après un ensemble de processus ayant pour but de vérifier que la requête est syntaxiquement et sémantiquement correcte. Par exemple dans les SSDs relationnels utilisant le langage d'expression SQL, le système vérifie que les noms des tables, les noms des attributs, les types des valeurs, etc. qui apparaissent dans la requête correspondent aux informations dans le catalogue des données.

2.2.1.3 Optimisation

Lors de cette étape, l'optimiseur applique sur la structure représentant la forme interne de la requête un ensemble de règles (commutativité, associativité, etc.) dans le but de trouver une représentation plus efficiente. Il s'agit de trouver un plan d'évaluation qui optimise une des mesures de performance ou plusieurs. Ces mesures de performance peuvent être le temps d'attente de l'utilisateur, l'utilisation du processeur (CPU), le nombre d'accès au disque (E/S), la consommation énergétique du système, ou le coût

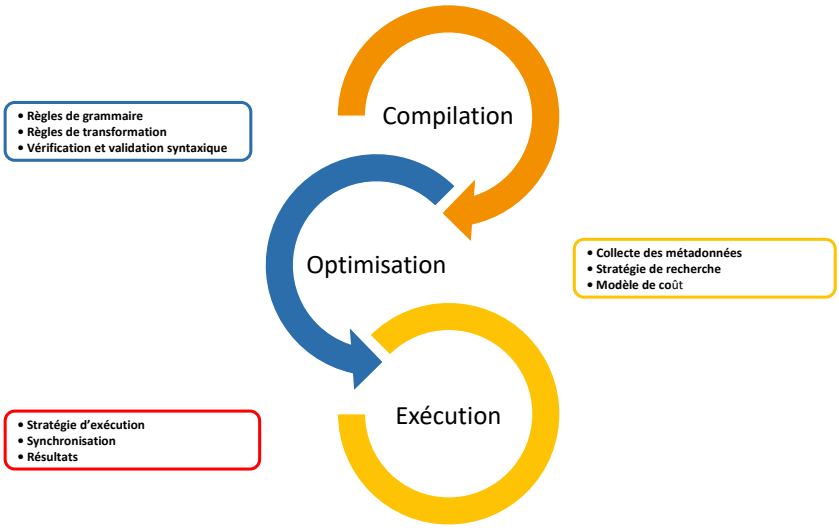


FIGURE 2.2 – Principales étapes d'un STR.

monétaire du traitement sur le cloud, etc. Pour une optimisation basée sur un modèle de coût dans les systèmes relationnels, le STR utilise les informations statistiques stockées dans le catalogue des données pour générer différentes expressions algébriques équivalentes puis sélectionne la meilleure expression en comparant leur coût estimé par un modèle.

2.2.1.4 Moteur d'exécution

L'étape finale est l'évaluation du meilleur plan retenu lors de la phase d'optimisation. Il est constitué d'un ensemble d'opérations dont l'évaluation permettra de charger les données stockées sur les mémoires secondaires. L'exécution des opérations de la requête peut se faire de plusieurs manières. D'une manière simple dite séquentielle, en exécutant les opérations une à une ou en exécutant certaines opérations en parallèle. Le parallélisme peut se faire de manière totalement indépendante (parallélisme inter-opération indépendant), ou en mode pipeline (parallélisme inter-opération dépendant) ou en mode intra-opération (parallélisme intra-opération). La manière d'évaluer la requête n'affecte pas les données en sortie mais plus plutôt les performances de traitement en temps d'exécution et celles de la consommation énergétique [34].

Une approche générale décrivant les principales étapes d'un STR dans les BDs est illustrée sur la figure 2.2. Ce schéma donne seulement une vue générique, chaque SSD peut utiliser d'autres composants en plus ou fusionner certaines étapes en une. Nous détaillerons dans la section 2.2.2, les principales étapes suivies pour exécuter une requête sur certaines architectures tout en focalisant le modèle relationnel.

2.2.2 Optimisation des requêtes

Dans cette section, nous aborderons les approches de traitement et les techniques d'optimisation implémentées au sein de certaines architectures, en donnant des exemples pour chaque étape si nécessaire.

2.2.2.1 STRs dans les systèmes de stockage centralisés

L'approche conventionnelle du STR dans un SGBD consiste à analyser les instructions SQL puis à produire une représentation interne de la requête dite logique de type calcul relationnel. À partir du plan logique l'optimiseur génère un ensemble de plan physique puis choisit le plan le moins coûteux. Lorsqu'une requête est exprimée dans un langage de requête de haut niveau comme SQL, l'analyseur identifie les mots clés de la requête (les mots clés SQL, les noms d'attributs, les noms de relations, etc.) qui apparaissent dans le corps de la requête, puis vérifie qu'elle est correctement formulée selon les règles syntaxiques (règles grammaticales) du langage. Puis la requête est validée en vérifiant que tous les noms des attributs et des relations sont valides et sémantiquement significatifs dans le schéma de la BD. En utilisant des règles de transformation, une représentation interne de la requête est ensuite créée, habituellement sous la forme d'une structure de données arborescentes appelée arbre algébrique. Le

SGBD doit alors concevoir une stratégie d'exécution pour extraire les résultats de la requête à partir des fichiers de la BD. Une requête peut posséder de nombreuses stratégies d'exécution, et le processus pour choisir une qui serait appropriée pour traiter la requête est connue sous le nom d'*optimiseur de requête* [39].

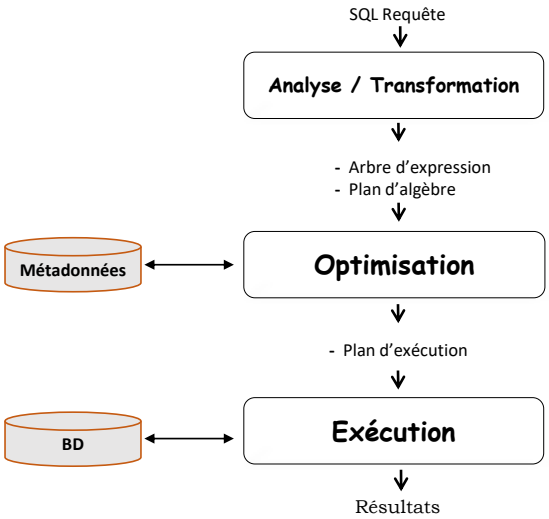


FIGURE 2.3 – STR d'une BD centralisée.

La figure 2.3 illustre les principaux modules du STR pour traiter une requête SQL sur une BD relationnelle. Pour mieux comprendre la démarche du STR pour extraire les tuples enregistrés dans la BD, considérons le schéma relationnel suivant (Figure 2.4) d'une BD décrivant la gestion d'une bibliothèque. La bibliothèque prête des livres aux sociétés et aux particuliers.

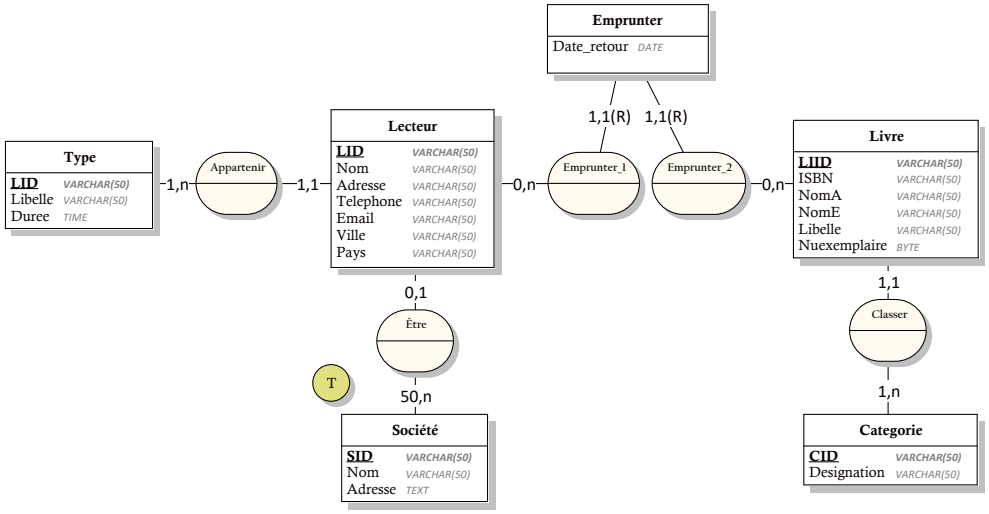


FIGURE 2.4 – schéma de modélisation logique déduit du schéma conceptuel E/A .

Supposons qu'un utilisateur souhaite connaître « Le Nom et l'Email par ordre alphabétique de tous les lecteurs de la ville de *Poitiers* travaillant avec la société *ABAN*». Ce besoin est formulé en SQL par la requête (Q_i) :

```
1 SELECT L.Nom, L.Email
2 FROM Lecteur L, Societe S
3 WHERE  L.ID   = S.LID  AND
4        S.Nom   = 'ABAN' AND
```

```
5       L.Ville = 'Poitiers'
6 ORDER BY L.Nom, L.Email DESC;
```

(i) **Analyse et pré-traitement:**
La fonction primaire de l'analyseur consiste à extraire de la requête SQL exprimée par l'utilisateur les atomes, puis de construire un arbre d'analyse dont les nœuds correspondent à des atomes ou à des catégories syntaxiques. Cette forme de représentation permet ainsi à l'analyseur de vérifier puis de valider que la requête exprimée est syntaxiquement et sémantiquement correcte. Par exemple, l'analyseur doit : (1) vérifier les noms des relations après le mot clé $FROM$, (2) vérifier l'existence des attributs de la clause $SELECT$, $WHERE$ ou encore (3) vérifier les types des valeurs des attributs, etc.

— ***Analyse Syntaxique:*** Les atomes correspondent aux éléments lexicaux tels que les mots clés (par exemple, $UPDATE$), les noms des attributs ou des relations, les constantes, les parenthèses, les opérateurs tels que + ou <, etc., et les catégories syntaxiques sont les noms des sous-parties de requête. Comme tous les autres langages de programmation, SQL possède des règles syntaxiques et grammaticales qui contrôlent la façon dont les mots peuvent être intégrés dans la formulation de la requête. Nous donnons dans ce qui suit le processus d'analyse de la requête Q_i exprimées ci-dessus. La syntaxe générique d'une requête destinée extraire les données de la BD est:
Forme(Q)::= Select <cols> From <tabs> Where <cond> [Order by <clos>,...]
Où `<cols>`, `<tabs>` et `<cond>` décrivent les catégories syntaxiques qui suivent les mots clés `Select`, `From` et `Where` respectivement. Ces catégories syntaxiques sont définies ainsi:
`<cols>`: liste le nom des attributs dans la requête.

```
<cols> ::= attribut, <cols>
<cols> ::= attribut
```

`<tabs>`: liste le nom des relations dans la requête séparées par des virgules.

```
<tabs> ::= Relation, <tabs>
<tabs> ::= Relation
```

`<cond>`: définit les conditions de sélection des données sur les attributs, les formes données ci-dessous ne sont pas pas exhaustifs.

```
<cond>::= <Cond> [AND <Cond>]
<cond>::= attribut IN (Forme(Q))
<cond>::= attribut = valeur
<cond>::= attribut LIKE \%Motif\%
```

Nous illustrons l'analyse syntaxique de notre requête Q_i selon la règle grammaticale proposée ci-dessous sur la figure 2.5. Le graphe représentant la requête permet ainsi à l'analyseur de vérifier qu'aucune règle grammaticale n'a été violée.

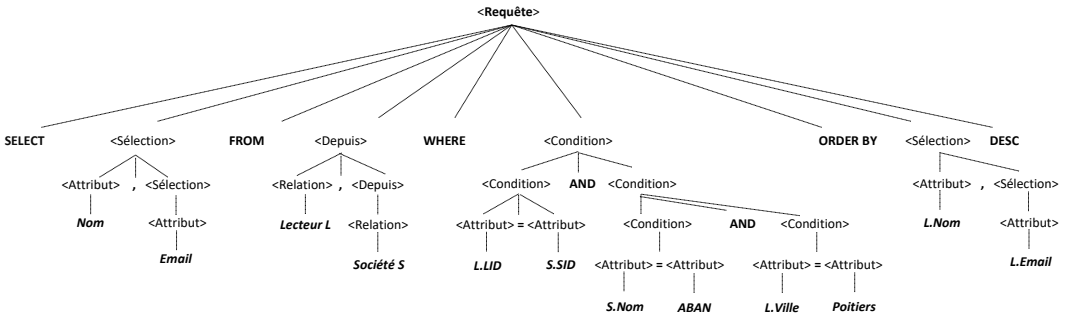
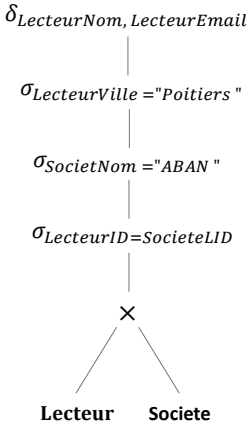


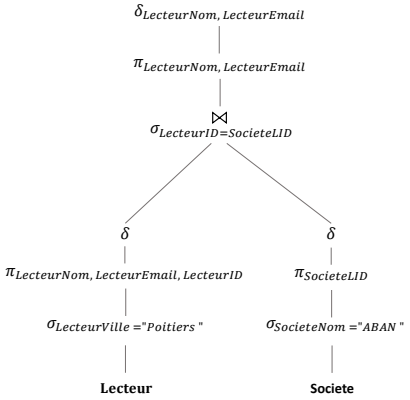
FIGURE 2.5 – Graphe de l'analyse syntaxique de la requête Q_i.

— ***Analyse sémantique (Pré-traitement):*** Une requête peut syntaxiquement être correcte mais sémantiquement poser des problèmes. L'analyseur sémantique ou le prétraitement a la responsabilité de vérifier que la requête est sémantiquement correcte. Il s'agit de s'assurer par exemple que tous les noms après la clause $< From >$ réfèrent à un nom d'une relation dans le schéma de la BD en cours, que les noms des attributs existent dans les relations invoquées et qu'aucune contrainte d'intégrité n'est outrepassée. L'arbre d'analyse validé est ensuite transformé en une expression de l'algèbre relationnelle étendue. L'expression produite est une structure algébrique dont les nœuds correspondent aux opérateurs de l'algèbre relationnelle et les feuilles représentent les relations dans

la BD. L'issu de cette phase conduit à la construction d'un arbre standardisé sur lequel, des règles d'élimination de la redondance ou des règles de réécriture des expressions algébriques [77] peuvent s'appliquer afin d'optimiser l'arbre algébrique. La figure 2.6a illustre des exemples d'expressions algébriques de la requête Q_i.



(a) Plan initial de l'expression algébrique.



(b) Plan logique après quelques règles de transformation.

FIGURE 2.6 – Graphe des expressions algébriques de la requête Q_i

(ii) **Réécriture de la requête:**
L'algèbre relationnelle proposée par E. Codd dans les années 70 [47] est un ensemble d'opérations formelles qui s'applique sur des relations et produisent de nouvelles relations en résultats. Dans cette sous-section, nous énumérons un ensemble de règles algébriques qui transforme un arbre algébrique en un arbre équivalent plus efficace. Ces règles résultent des propriétés d'associativité et de commutativité des opérateurs algébriques. L'issu de l'application de ces heuristiques conduit à une structure de données appelée plan de requête logique [46].

— ***Règles de transformation algébriques:*** Les opérateurs algébriques de base sont scindés en deux types: les opérateurs binaires, c'est-à-dire qu'à partir de deux relations ils en retournent une nouvelle relation tels l'union (U), la différence ($-$) et le produit cartésien (X) et les opérateurs unaires dits spécifiques tels que la projection(π), restriction(σ), etc. En exploitant certaines propriétés mathématiques telles que la commutativité et l'associativité, il est possible de déduire une combinaison d'évaluation des opérateurs qui conduit à un résultat plus optimal. Ci-dessous, nous énonçons quelques règles de combinaison des opérateurs utiles dans le processus de la réécriture d'une requête SQL [46][39].

(a) **Décomposition de l'opérateur** σ: La sélection conjonctive peut être décomposée en une cascade (c'est-à-dire une séquence) d'opérations σ individuelles:
$\sigma_{c_1\ \wedge\ c_2\ \wedge\ \ldots\ and\ c_n}(\mathcal{R}) \equiv \sigma_{c_1}(\sigma_{c_2}(\ldots(\sigma_{c_n}(\mathcal{R})\ldots)))$.

(b) **Commutativité de** σ: L'opérateur σ est commutative.
$\sigma_{c_1}(\sigma_{c_2}(\mathcal{R})) \equiv \sigma_{c_2}(\sigma_{c_1}(\mathcal{R}))$.

(c) **Décomposition de l'opérateur** π: Dans une séquence de π, tous les π peuvent être ignorés sauf le dernier:
$\pi_{Liste_1}(\pi_{Liste_2}(\ldots(\pi_{Liste_n}(\mathcal{R}))\ldots)) \equiv \pi_{Liste_1}(\mathcal{R})$.

(d) **Commuter** σ **avec** π: Si la contrainte de sélection c ne nécessite que les attributs $A_1; \ldots; A_n$ dans la liste de projection, les deux opérations peuvent être commutées:
$\pi_{A_1,A_2,\ldots,A_n}(\sigma_c(\mathcal{R})) \equiv \sigma_c(\pi_{A_1,A_2,\ldots,A_n}(\mathcal{R}))$.

(e) **Commutativité de** $\bowtie$ **(et de** $\times$**)**: L'opérateur de jointure est commutatif ainsi que le produit cartésien.
$\mathcal{R} \bowtie_c \mathcal{S} \equiv \mathcal{S} \bowtie_c \mathcal{R}$
$\mathcal{R} \times \mathcal{S} \equiv \mathcal{S} \times \mathcal{R}$.

(f) **Commuter** σ **avec** $\bowtie$ **(ou** $\times$**)**: On peut commuter les deux opérateurs si les attributs de la contrainte (c) de sélection n'impliquent que les attributs d'une des relations jointes.
$\sigma_c(\mathcal{R} \bowtie \mathcal{S}) \equiv (\sigma_c(\mathcal{R})) \bowtie S$.

Si la contrainte (c) peut être décomposée en $c_1 \wedge c_2$, où c_1 ne porte que les attributs de R et c_2 ne concerne que celui de S. Ainsi les opérateurs peuvent commuter comme suit:
$\sigma_c(\mathcal{R} \bowtie \mathcal{S}) \equiv (\sigma_{c_1}(\mathcal{R})) \bowtie (\sigma_{c_2}(S))$.

(g) **Commuter l'opérateur π avec $\bowtie$ (ou $\times$)**: Supposons que les attributs à projeter sont $L = \{A_1, \ldots, A_n, B_1, \ldots, B_n\}$, où $A_1, \ldots, A_n$ proviennent de la relation $\mathcal{R}$ et $B_1, \ldots, B_n$ de S. Si la jointure n'implique que les attributs de L, alors les deux opérateurs peuvent être commutés.
$\pi_L(\mathcal{R} \bowtie_c \mathcal{S}) \equiv (\pi_{A_1,\ldots,A_n}(\mathcal{R})) \bowtie_c (\pi_{B_1,\ldots,B_m}(\mathcal{S}))$.

Si dans la condition c, on trouve des attributs autres que ceux figurant dans L, un dernier opérateur de projection est alors ajouté contenant ces attributs. Par exemple, si les attributs $A_{n+1}, \ldots, A_{n+k}$ de $\mathcal{R}$ et $B_{m+1}, \ldots, B_{m+p}$ de $\mathcal{S}$ sont invoqués dans la condition c alors les opérateurs peuvent commuter comme suit:
$\pi_L(\mathcal{R} \bowtie_c \mathcal{S}) \equiv \pi_L((\pi_{A_1,\ldots,A_n}(\mathcal{R})) \bowtie_c (\pi_{B_1,\ldots,B_m}(\mathcal{S})))$.

Pour $\times$, il n'y a pas de condition c, il faut juste remplacer $\bowtie_c$ par $\times$ dans la règle de transformation.

(h) **Commutativité des opérateurs ensemblistes**: Les opérations ensemblistes $\cup$ et $\cap$ sont commutatives, mais la différence$(-)$ ne l'est pas.

(i) **Associativité des opérateurs** $\bowtie, \times, \cup, \cap$: Ces opérateurs sont dits associatifs; C'est-à-dire si θ représente le même opérateur (un des quatre opérateurs), on a:
$(\mathcal{R}\ \theta\ \mathcal{S})\ \theta\ \mathcal{T} \equiv \mathcal{R}\ \theta\ (\mathcal{S}\ \theta\ \mathcal{T})$.

(j) **Commuter σ avec des opérations ensemblistes** $-, \cup, \cap$: L'opération σ commute avec les opérations ensemblistes. Si θ représente l'une de ces trois opérations dans une expression, on a:
$\sigma_c(\mathcal{R}\ \theta\ \mathcal{S}) \equiv (\sigma_c(\mathcal{R}))\ \theta\ (\sigma_c(\mathcal{S}))$

(k) **L'opération π commute avec $\cup$**:
$\pi_{\mathcal{L}}(\mathcal{R} \cup \mathcal{S}) \equiv (\pi_{\mathcal{L}}(\mathcal{R})) \cup (\pi_{\mathcal{L}}(\mathcal{S}))$

(l) **Conversion d'une séquence (σ,$\times$) en $\bowtie$** :Si la condition c d'une σ qui suit un $\times$ correspond à une condition de jointure, convertir la séquence (σ,$\times$) en une $\bowtie$ peut s'écrire comme suit:
$\sigma_c(\mathcal{R} \times \mathcal{S}) \equiv (\mathcal{R}\ \bowtie_c \mathcal{S})$

(m) **Éclater l'opérateur ensembliste $-$ avec σ**:
$\sigma_c(\mathcal{R} - \mathcal{S}) = (\sigma_c(\mathcal{R})) - (\sigma_c(\mathcal{S}))$

(n) **Éclater l'opérateur ensembliste $\cap$ avec σ seulement sur un argument**:
$\sigma_c(\mathcal{R} \cap \mathcal{S}) = (\sigma_c(\mathcal{R})) \cap (\mathcal{S})$

Cette énumération n'est qu'exhaustive, d'autres règles d'équivalence impliquant des opérateurs telles que l'agrégation(δ),l'élimination des doublons(δ) etc. existent. Nous illustrons à présent un exemple d'utilisation des règles d'équivalence en considérant notre BD de la bibliothèque sur la requête Q_i. L'expression algébrique initiale de la requête Q_i pourrait être (Figure 2.6a):

$$Q_i = \delta_{LecteurNom,\, LecteurEmail}(\sigma_{LecteurVille="Poitiers"}(\sigma_{SocieteNom="ABAN"}(\sigma_{LecteurID=SocieteLID}(Lecteur \times Societe))))$$

L'expression ci-dessus peut être transformée en une autre équivalente mais produisant moins de données intermédiaires. Cette transformation est effectuée en appliquant les règles citées ci-dessus, sur la requête ou sur une partie de la requête (Figure 2.6b).

$$Q_i = \delta_{LecteurNom,\, LecteurEmail}(\pi_{LecteurNom,\, LecteurEmail}((\pi_{LecteurNom,\, LecteurEmail,\, LecteurID}(\sigma_{LecteurVille="Poitiers"}(Lecteur))) \bowtie_{LecteurID=SocieteLID} (\pi_{SocieteID}(\sigma_{SocieteNom="ABAN"}(Societe)))))$$

— ***Quelques règles d'amélioration du plan algébrique:*** Les règles définies ci-dessus sont utilisées par le système pour transformer un arbre algébrique initial en un plan logique plus optimal à s'exécuter. Les règles de transformation utilisées par les heuristiques incluent les démarches suivantes:

(a) Décomposer toute opération de $SELECT$ avec des conditions conjonctives en une séquence d'opérations de $SELECT$. Cette stratégie donne une grande souplesse de faire descendre l'opérateur de sélection ($SELECT$) dans l'arborescence dans la mesure du possible.

(b) La propriété de commutativité de l'opérateur $SELECT$ avec d'autres opérateurs, permet de l'empiler aussi loin dans l'arborescence que le permettent les attributs requis dans la condition de sélection. Si ces attributs proviennent d'une seule table alors l'opérateur est déplacé jusqu'au niveau de la feuille de l'arbre. Sinon si les attributs concernent les deux tables, faire descendre l'opérateur si possible juste au-dessus après la combinaison des deux tables.

(c) Pour les opérateurs binaires, réécrire les nœuds finaux (feuilles) de sortes que les nœuds avec les opérateurs $SELECT$ les plus restrictifs soient évalués en premier dans l'arborescence, c'est-à-dire ceux qui fournissent une relation avec moins de tuples.

(d) Pour l'opérateur de projection, on décompose et on pousse les attributs de projection vers le bas autant que possible, en créant si nécessaire de nouveaux opérateurs de projection. Seuls les attributs requis dans les sorties et dans d'autres opérateurs postérieurs sont à sauvegarder après l'évaluation de chaque opérateur.

Les règles d'amélioration de la requête ont pour objectif de chercher à appliquer les opérateurs qui réduisent la taille des données intermédiaires, pour cela elles évaluent le plus tôt possible les opérateurs de sélection ($SELECT$) pour réduire le nombre de tuples et les opérateurs de projection ($PROJECT$) pour réduire le nombre d'attributs en les faisant descendre le plus loin possible dans l'arbre de la requête [173].

(iii) **Sélection d'un plan d'exécution**
Une fois que la requête ait été analysée et transformée en un plan logique, elle doit ensuite être traduite en plan physique. Un plan physique représenté sous la forme d'un arbre de requête inclut des informations sur les méthodes d'accès aux données à extraire de chaque relation ainsi que les algorithmes à utiliser pour calculer les opérateurs relationnels figurant dans l'arbre. Pour un plan logique donné, différents plans d'exécution peuvent être générés donc il est important de pouvoir comparer les plans en termes de coût (estimé), et de choisir le meilleur. Pour ce faire, l'optimiseur doit pouvoir estimer le coût de chaque opération et les combiner pour obtenir le coût du plan. Le coût de l'exécution des requêtes est mesuré sur la base de l'utilisation d'un certain nombre de ressources, comprenant les entrées/sorties (accès) disques, le temps CPU pour exécuter les opérateurs, etc. [53]. L'estimation des coûts nécessite la connaissance des paramètres de la BD, tels que le nombre de pages et les index disponibles, etc. et les paramètres du système tels que le coût du CPU, etc. Ces statistiques sont tenues à jour dans les catalogues du SSD [148]. Trouver la combinaison d'opérations physiques la plus optimale pour produire un résultat en utilisant une optimisation basée sur les modèles coûts évoque deux principaux problèmes connexes: la génération des plans physiques et l'estimation des coûts des plans [34].

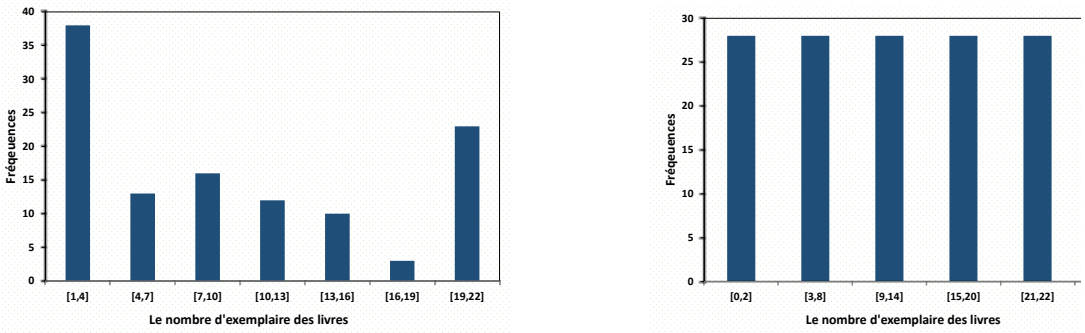
— ***Stratégie de recherche (Génération des plans):*** Explorer tous les plans possibles peut être très coûteux pour des requêtes complexes. La plupart des optimiseurs fonctionnent avec des heuristiques qui réduisent le coût de l'optimisation, au risque potentiel de ne pas trouver un plan optimal. À partir d'un plan logique, la stratégie de recherche explore l'espace de plan possible et en choisit celui qui est « optimal », c'est-à-dire qui minimise une fonction de coût. De nombreuses stratégies de recherche ont été proposées, et peuvent être soit déterministes ou randomisées [99].

- **Les stratégies de recherche déterministes:** Les stratégies de recherche déterministes sont des algorithmes qui choisissent toujours le même plan pour une requête donnée dans un espace de recherche. Les stratégies déterministes consistent à construire des plans, en partant des relations de base, en additionnant une relation supplémentaire à chaque étape jusqu'à l'obtention du plan final. La variante la plus populaire dans les optimiseurs de requêtes est la programmation dynamique [92, 163] fondée sur une approche *Breadth-First* (exploration en largeur). Elles partent d'un nœud racine, puis explorent premièrement les nœuds voisins avant de se déplacer vers les voisins de niveau suivant. Les stratégies Greedy (gourmandes) [174] sont fondées sur une approche heuristique qui produit rapidement des solutions de haute qualité [86]. Cette approche est dite descendante car elle se base sur l'exploration en profondeur (*deepth-first*) c'est-à-dire en sélectionnant un nœud comme nœud racine, explore autant que possible le long de chaque branche avant de revenir en arrière.
- **Les stratégies de recherche randomisées:** Les stratégies aléatoires se concentrent sur la recherche de la solution optimale autour d'un point particulier. Ils ne garantissent pas l'obtention de la meilleure solution, mais évitent une phase de recherche onéreuse pour l'optimiseur. Ils essaient d'améliorer un plan physique heuristiquement choisi (point de départ) jusqu'à l'obtention d'une solution optimale respectant une condition d'arrêt. La stratégie d'escalade(*Hill-Climbing*)

[118] en est un exemple, elle est fondée sur les solutions aléatoires ; c'est un algorithme itératif qui commence avec un plan physique choisi et tente d'apporter des changements au plan, par exemple en remplaçant une méthode pour exécuter un opérateur par une autre ou en réordonnant des jointures en utilisant les règles de transformation associatives et/ou commutatives, pour trouver un plan « proche » qui a un coût inférieur. Le changement produisant une meilleure solution est considéré comme la nouvelle solution. Ce processus est répété jusqu'à ce qu'il n'y ait aucune autre amélioration possible. Outre ces stratégies, les stratégies basées sur l'algorithme génétique sont aussi utilisées. L'algorithme génétique [23] est une approche heuristique utilisée dans le système *PostgreSQL* pour la sélection de l'arbre de jointure.

— ***Modèle de coût et estimation des paramètres:*** La collecte des statistiques et l'estimation du coût sont étroitement interdépendantes. Pour chaque algorithme disponible, l'optimiseur est capable à l'aide d'un modèle de coût mathématique (dont la métrique est généralement en unité de temps) d'estimer le coût d'évaluation d'une opération sur la base des statistiques fournies en entrée (taille des données). Le modèle de coût considère un certain nombre de paramètres tels que le nombre de pages (E/S) de disque à récupérer pour une relation donnée, ou le nombre de tuples requis (CPU), d'autres paramètres peuvent être fixés par l'administrateur pour quantifier la performance du système matériel qui héberge le SGBD.

L'estimation des paramètres repose sur les statistiques collectées par le système en mode hors ligne, et elles sont mises à jour périodiquement. L'exactitude et la précision de ces statistiques ont une incidence importante sur la qualité de l'estimation des coûts et par conséquent sur la performance de l'optimiseur. L'optimiseur de requêtes calcule (cherche) la cardinalité de chaque table à partir du dictionnaire des données et pour les attributs indexés, il peut déterminer les valeurs minimales et maximales. Les contraintes d'intégrité telles que la contrainte de clé primaire ou secondaire peuvent aider l'optimiseur à déduire les statistiques sur les distributions de certaines valeurs d'attributs. La plupart des SSDs sauvegardent la distribution des valeurs de chaque attribut sous la forme d'histogramme. Un histogramme sur un attribut X est construit en partitionnant la distribution des données de X en b ($>= 1$) sous-ensembles mutuellement disjoints appelés *Bucket* en approximant les fréquences et les valeurs dans chaque *bucket*. Chaque sous-ensemble est défini par ses bornes et sa fréquence [44]. Les histogrammes les plus utilisés sont l'histogramme: **à largeur égale** [140], **à égal fréquence ou à profondeur égale** [140]. Les figures 2.7a et 2.7b donnent des exemples d'histogrammes de distribution des valeurs de l'attribut: *Nombre d'exemplaire - (Nuexemplaire)* de la table Livre.



(a) Largeur égale.

(b) Profondeur égale.

FIGURE 2.7 – Exemples d'histogrammes de distribution.

Sur la base de ces statistiques collectées, l'optimiseur peut estimer le coût des opérations. Le modèle de coût est une fonction importante car il guide pour le choix du meilleur plan généré. L'étape d'estimation des coûts tente de déterminer le coût des opérations constituant un plan physique en combinant le coût de certaines ressources telles que $C_{cpu}, C_{E/S}, C_{Ram}$ *et* C_{com}. L'exactitude des estimations dépend des paramètres de structuration des données (le tri des données dans les fichiers, le type d'index utilisé, etc.), l'utilisation des mémoires tampons. Cependant la capacité de faire une estimation précise et adéquate du coût des opérations et des statistiques restent encore très difficile

dans les optimiseurs de requêtes. La Figure 2.8 donne un aperçu du processus d'optimisation des requêtes.

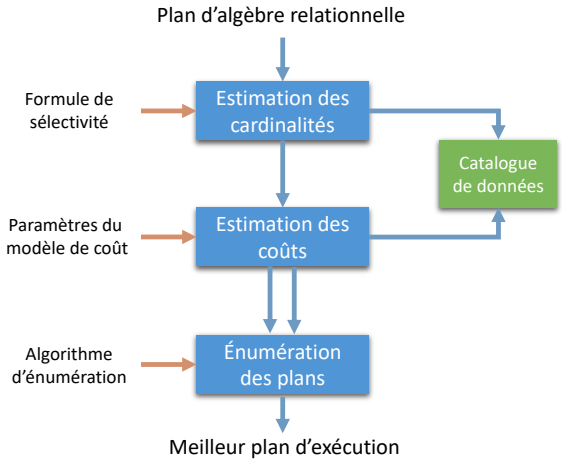


FIGURE 2.8 – Moteur d'optimisation des requêtes.

(iv) **Exécution du plan**
L'étape finale du processus consiste à l'exécution du plan physique choisi. Lors de l'exécution, outre le mode d'exécution séquentiel ou matérialisé certaines opérations de la requête peuvent être traitées en mode parallélisé.

— ***Mode matérialisé:***
Lors de l'exécution en mode matérialisé, chaque résultat est physiquement matérialisé sur un dispositif de stockage tampon avant d'être récupéré par l'opération suivante. Ce mode d'exécution peut s'avérer très coûteux si la relation intermédiaire produite ne tient pas en mémoire tampon et qu'elle doit être swappée sur les mémoires secondaires.

— ***Mode parallélisé***
Avec le mode parallèle, plusieurs processus peuvent travailler simultanément pour exécuter une partie de la SQL. En répartissant le travail nécessaire entre plusieurs unités de traitement (processeur ou cœurs), le moteur d'exécution peut alors traiter la requête plus rapidement que sur une seule unité de traitement. Le mode d'exécution parallélisé peut se décomposer en plusieurs variantes (figure 2.9). Le parallélisme *inter-requête* vise à améliorer le débit des transactions courtes généralement de type OLTP (*On Line Transaction Processing*) [15] en répartissant les ressources entre les utilisateurs et le parallélisme *intra-requête* consiste à exécuter les opérations d'une même requête en parallèle de manière *indépendante*, en *pipeline* ou en *intra-opération*. La répartition des données est une étape cruciale et est en amont du traitement parallèle. Une distribution uniforme des données permet en effet de répartir les accès sur les différents disques, et d'optimiser ainsi le temps d'accès et le débit. Les trois techniques de répartition couramment employées [30][115]:(1) la répartition circulaire, (2) la répartition par hachage, et (3) et la répartition par intervalles.

(a) **Parallélisme inter-requête (*Inter-query parallelism*):**
Le parallélisme inter-requêtes exécute simultanément plusieurs requêtes de manière indépendante. Cette technique vise à partager les ressources disponibles d'une ou plusieurs machines de façon équitable et efficace entre les utilisateurs (ou plus exactement entre les requêtes qu'ils soumettent). Ils existent dans la littérature quelques travaux qui se sont penchés sur cette technique en proposant des approches pour gérer le dispatching des requêtes entre les nœuds de traitement et la synchronisation des accès aux mêmes données [114][146][147][66].

(b) **Parallélisme intra-requête (*Intra-query parallelism*):**
Le parallélisme intra-requête est un mécanisme bien connu pouvant atteindre des performances élevées dans les systèmes de stockages des données. Cette technique consiste à découper la même requête en plusieurs sous-opérations pouvant s'exécuter simultanément. Deux formes de parallélisme intra-requête existent: *Parallélisme inter-opérateur* et *intra-opérateur* (Figure 2.9).

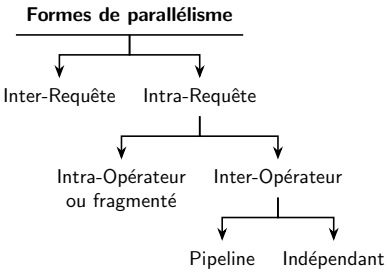
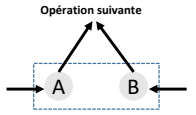
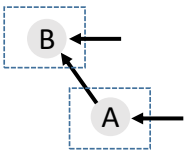


FIGURE 2.9 – Différentes formes de parallélisme.

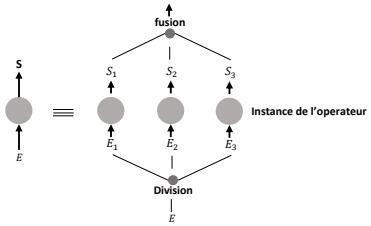
(i) **Parallélisme inter-opérateur:**
Cette forme consiste à exécuter de façon occurrente plusieurs opérations d'une même requête. Il existe deux approches de parallélisme inter-opérateur (Figure 2.9) le parallélisme *inter-opérateur indépendant* et Le parallélisme *inter-opérateur dépendant* (aussi appelé pipeline). Le parallélisme inter-opérateur indépendant consiste à exécuter en parallèle deux opérations du plan de la requête de manière indépendante. La figure 2.10a illustre cette forme de parallélisme entres les opérations A et B. Le parallélisme inter-opérateur en pipeline consiste à exécuter deux opérations du plan ayant un lien de communication directe dite productrice-consommatrice sans la matérialisation des données intermédiaires. La figure 2.10b illustre cette forme de parallélisme entre les opérations A et B.



(a) inter-opérateur indépendant



(b) inter-opérateur pipeline



(c) Intra-opérateur

FIGURE 2.10 – Exemple de parallélisme

(ii) **Parallélisme intra-opérateur:** Le parallélisme intra-opérateur consiste à traiter parallèlement la même opération repartie entre différents nœuds sur des fragments de données obtenues grâce à une méthode de répartition, le nombre de nœuds définit le degré de parallélisme (***DdP***). La figure 2.10c illustre un exemple du parallélisme intra-opération avec une opération de sélection sur une relation E. Le degré de parallélisme fixé à 3.

La génération d'un plan parallèle doit être à la fois correcte, efficace et inclure des opérations de contrôle pour synchroniser les accès aux données et détecter les terminaisons des opérations[198]. Le système doit aussi gérer les difficultés de mise en œuvre du parallélisme telles que (2) le contrôle de l'exécution parallèle, (3) la fragmentation, (4) la reconstitution, (5) l'interférence, (6) la répartition de la charge de travail, (7) le degré de parallélisme.

2.2.2.2 STRs dans les systèmes de stockage distribués

Dans un SSD distribué, le traitement de la requête nécessite la prise en compte du coût de la communication entre les noeuds (ou sites). Le coût de la communication est le goulot d'étranglement dans cette architecture, la fonction principale du STR est de réduire aussi que possible le transfert des données entre les sites. La stratégie globale de traitement d'une requête sur un système distribué est la décomposition de la requête en sous-requêtes, puis d'évaluer chaque sous-requête à proximité des données, contrairement à la stratégie qui consisterait à collecter toutes les données requises sur une seule station qui finalement exécuterait entièrement la requête [21]. Les sous-requêtes générées peuvent aussi être traitées en parallèle sur différentes stations. L'optimiseur local doit être capable de minimiser le temps de réponse ou la consommation de ressources (par exemple, la consommation énergétique au niveau de chaque station) comme dans les systèmes centralisés. La figure 2.11 illustre le processus de traitement des requêtes sur une architecture distribuée [131]. Sur la figure 2.11, nous distinguons quatre étapes principales pour transformer

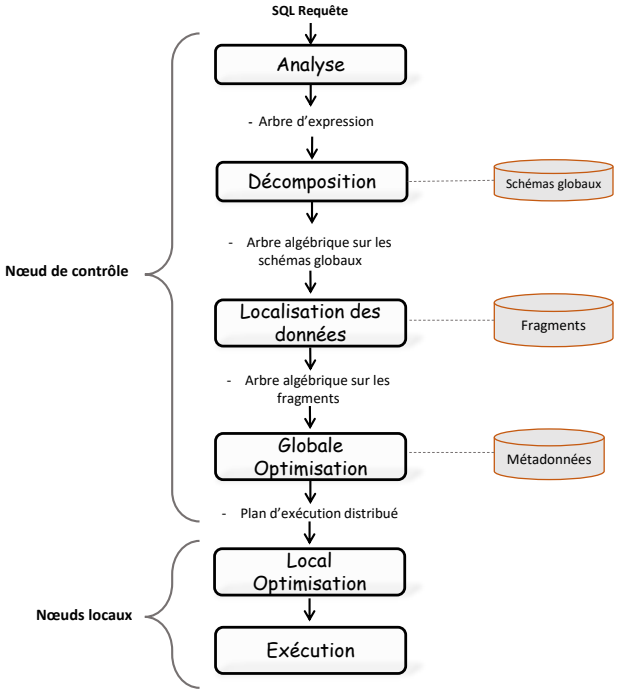


FIGURE 2.11 – Processus de traitement d'une requête dans un système distribué.

une requête écrite en SQL en une séquence optimisée d'opérateurs. Ces étapes assurent les fonctions de décomposition des requêtes, de localisation des données, d'optimisation globale et d'optimisation locale des requêtes puis l'exécution des opérateurs.

La première étape consiste à la décomposition de la requête lors de laquelle, la requête est scannée, analysée et validée avant tout processus de décomposition. L'analyse de la requête permet de détecter puis de rejeter le plus tôt possible les requêtes qui ne respectent pas la syntaxe de formulation. La requête validée est ensuite décomposée en une requête algébrique. Cette traduction est faite en se référant au schéma global des données et ne prend pas en compte la distribution et la réplication réelles de données. Après l'étape de la décomposition, la localisation des fragments de données et l'optimisation globale des requêtes sont les suivantes. La localisation permet de déterminer les fragments qui participeront à la formulation de la requête. L'optimisation globale consiste à sélectionner une stratégie qui se rapproche le plus de l'optimum, obtenue en permutant l'ordonnancement des opérations. Les trois premières étapes sont réalisées sur un site centralisé qui utilise les informations stockées dans le catalogue global du système. Le catalogue global sauvegarde les informations sur les données partagées, les informations sur les placements des fragments. La dernière étape est faite sur les stations locales ayant des fragments impliqués dans la requête. Chaque sous-requête appelée requête locale, est optimisée à l'aide du catalogue local du site puis exécutée ensuite. Les techniques pour l'optimisation locale et l'exécution locale sont similaires à celle utilisées dans les systèmes centralisés.

2.2.2.3 STRs dans les systèmes multi-bases

Nous avons abordé le traitement des requêtes dans les SSDs distribués et centralisés dans les sous-sections précédentes, en fait ces systèmes sont logiquement intégrés et fournissent une unique image du schéma global de la BD, même s'ils sont physiquement repartis. Un système multi-bases de données offre un accès transparent à une collection de sources hétérogènes de données distribuées sur un réseau informatique [131]. En plus de l'hétérogénéité et de la distributivité, les BDs peuvent être autonomes, c'est-à-dire contrôlées et gérées indépendamment du site centralisé. Les différentes étapes impliquées dans le traitement d'une requête sur un système distribué multi-bases sont illustrées sur la figure 2.12.

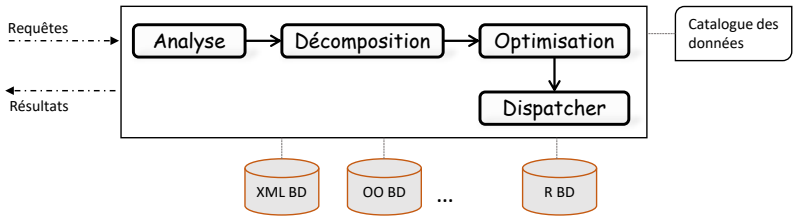


FIGURE 2.12 – Processus de traitement d'une requête dans un système multi-bases.

Comme dans les systèmes distribués, la requête globale est d'abord analysée puis décomposée en une requête représentée sous la forme d'un arbre algébrique. La requête algébrique est composée d'opérateurs primitifs tels que la sélection, projection, jointure, etc. nécessaire pour extraire les données des sources de stockage. Un ensemble de plan d'exécution est construit à partir de la requête algébrique par le générateur de plan. Lorsqu'un plan d'exécution est généré, l'estimateur des coûts fournit une estimation du coût d'exécution du plan en fonction d'un modèle de coût et des statistiques disponibles dans le catalogue global d'information. Un plan qui satisfait les objectifs d'optimisation est ensuite choisi comme plan d'exécution de la requête et envoyé au dispatcher(répartiteur) qui coordonne son exécution dans chaque base participant au traitement de la requête. Le moniteur d'exécution collecte les statistiques sur l'exécution des sous-requêtes et les envoie au gestionnaire de statistiques, qui mettra à jour le catalogue d'information global si nécessaire [107].

Les principaux challenges lors du traitement des requêtes dans les systèmes multi-bases sont : l'hétérogénéité des modèles de coût, l'hétérogénéité de l'optimisation des requêtes et l'adaptabilité du traitement des requêtes pour faire face à la forte variation de l'environnement (défaillances, retards imprévisibles, etc.) [14].

L'hétérogénéité des modèles de coût, réfère au problème d'exploitation des modèles de coût et à la difficulté pour extraire les valeurs des paramètres du modèle à partir des différentes sources de données. Ces valeurs sont importantes pour estimer les coûts des plans physiques générés par l'optimiseur des requêtes multi-bases. En plus de l'hétérogénéité des modèles de coût, l'optimisation des requêtes dans les systèmes multi-bases doit tenir compte de l'hétérogénéité de la capacité de traitement des sources de données. Par exemple, un site local peut ne prendre en charge que des opérations de sélection simple, tandis qu'un autre peut prendre en charge des requêtes complexes impliquant des opérations de jointure et d'agrégation. L'adaptabilité du traitement des requêtes est une forme de traitement rétroactive entre l'environnement d'exécution et l'optimiseur global de requêtes afin de réagir aux variations imprévues des conditions d'exécution.

2.2.2.4 STRs dans les systèmes multi-stores

Les systèmes multi-stockages (multi-stores) permettent la gestion intégrée d'un ensemble de magasins de données (Data-store) en nuages (cloud) en utilisant un framework de traitement des données tel que MapReduce, Spark. Dans [14], les auteurs classifient les systèmes multi-stores en trois catégories sur la base du niveau de couplage des data-stores sous-jacentes: faiblement couplé, fortement couplé et hybride.

Pour traiter une requête sur un système multi-stores, deux modules principaux sont identifiés: un module de traitement de la requête et un module d'adaptateur (Wrapper) dédié pour chaque Data-store. Chaque module possède son catalogue d'information. Les étapes de traitement d'une requête comme illustrées sur la figure 2.13 sont les suivantes: (a) Analyser la requête et la traduire en sous-requêtes (une par data-store), chacune exprimée dans un langage commun, (b) Envoyer les sous-requêtes aux wrappers

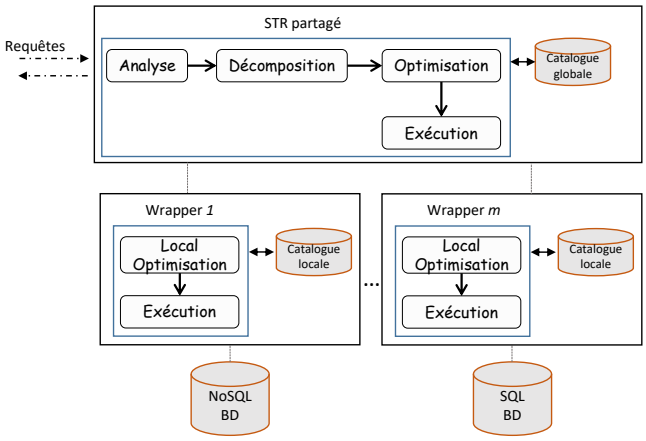


FIGURE 2.13 – Processus de traitement d'une requête dans un système multi-store.

impliqués dans le traitement de la requête. La réception de la sous-requête par les wrappers enclenchera son exécution dans les data-stores pour extraire les données puis les résultats finaux sont mis dans un format commun, (c) fusionner les résultats des wrappers impliqués dans le traitement. Nous décrivons ci-dessous HadoopDB, un exemple des systèmes multi-stores classé comme fortement couplé.

(i) **HadoopDB**

HadoopDB permet de connecter plusieurs systèmes de BDs à un unique nœud en utilisant le framework Hadoop [1]. Il offre une grande flexibilité aux pannes et la possibilité de fonctionner dans des environnements hétérogènes en héritant de l'implémentation de l'ordonnancement et du contrôle des tâches de Hadoop. Les requêtes sont parallélisées entre les nœuds en utilisant le modèle de traitement MapReduce [3].

HadoopDB élargit l'architecture du système Hadoop avec quatre composants: (1) connecteur à la BD,(2) catalogue d'information, (3) chargeur de données et (4) un planificateur SQL-MapReduce-SQL (SMS). Le connecteur sert d'interface entre les systèmes de stockage de données résidant sur les nœuds du cluster et les Task Trackers en utilisant des drivers JDBC [2]. Le catalogue HadoopDB stocke ses méta-informations sous forme de fichier XML [3] dans des formats HDFS [4]. Le chargeur des données a pour fonction de (a) fragmenter les données à partir d'une clé de fragmentation, (b) diviser les données d'un nœud en plusieurs fragments plus petits et (c) de charger massivement les données d'un nœud. Le planificateur SMS étend le module Hive, un composant d'Hadoop. Hive propose un langage basé sur le SQL (norme ANSI-92) appelé HiveQL [5], utilisé pour formuler des requêtes sur les données stockées dans les formats HDFS. Hive transforme une requête écrite en HiveQL en job MapReduce, qui est soumis au JobTracker pour exécution.

Le traitement des requêtes dans un système HadoopDB est simple (figure 2.14). Le planificateur SMS a pour fonction de traduire puis d'optimiser l'exécution de la requête. Le planificateur SMS décompose une requête HiveQL en un plan d'opérateurs relationnels. L'optimiseur restructure le plan de requête pour créer un plan plus optimisé. L'une des principales fonctions de l'optimiseur consiste à diviser le plan en des tâches MapReduce. L'optimiseur de Hive utilise les règles algébriques pour optimiser le plan, il n'utilise pas une approche d'optimisation basée sur les modèles de coût.

Le processus d'exécution d'une requête reste fortement similaire d'une architecture à une autre architecture. Les tâches effectuées en amont des étapes d'optimisation et de traitement dépendent de l'architecture de déploiement sinon toutes les étapes abordées dans les systèmes centralisés sont répétées sur les nœuds locaux dans les architectures repartis (distribuées, multi-bases, multi-stores).

1. Hadoop est un framework libre et open source écrit en Java destiné à faciliter la création d'applications distribuées
2. Java Database Connectivity
3. Extensible Markup Language
4. Hadoop Distributed File System
5. Hive Query Language

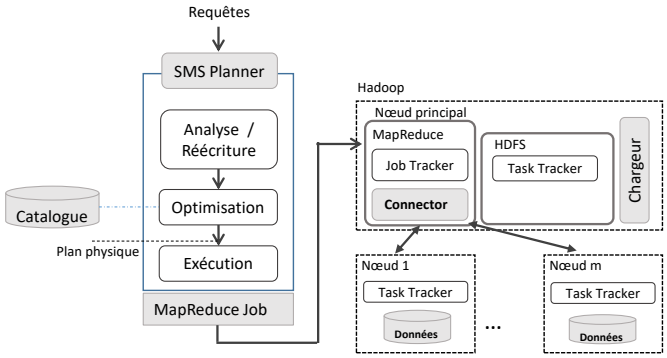


FIGURE 2.14 – Processus de traitement d'une requête dans un système HadoopDB.

2.3 Modèle de coût

Un modèle de coût dans les SGBDs est généralement caractérisé par trois composants [111]: les coûts logiques, les coûts algorithmiques et les coûts physiques.

La composante logique est basée sur la quantification du volume des données à traiter par les opérateurs de la requête. Pour chaque opérateur, trois types de volumes de données peuvent être distingués: les données en entrée, les données de sortie et les données temporaires. Le volume des données est couramment mesuré en cardinalité, c'est-à-dire le nombre d'enregistrement (ou de ligne) dans la table. D'autres unités telles que le nombre de blocs d'E/S, le nombre de pages ou la taille totale en octets sont souvent utilisées. Généralement, les SGBDs mémorisent des statistiques sur les données stockées dans les BDs. Ces statistiques sont entre autres les cardinalités de chaque table, le nombre de valeurs distinctes par colonne, etc. Une pléthore de techniques d'approximation du volume des données a été largement étudiée dans la littérature [75][140][51]. Les coûts algorithmiques étendent les coûts logiques en tenant compte des propriétés des algorithmes. Ils dépendent de la cardinalité des données définies en entrées. Le coût physique combine le coût algorithmique avec une description matérielle abstraite pour dériver les différents facteurs de coût en termes de temps, et donc le temps total d'exécution. Une description matérielle comprend généralement des informations telles que la vitesse du processeur, la latence des E/S, la bande passante des E/S et la bande passante du réseau. Les coûts physiques sont fortement liés au matériel. On distingue différents facteurs de coût physique:

— **CPU:** représente le coût de traitement des opérations sur les enregistrements en mémoire pendant l'exécution de la requête. Ces opérations comprennent la recherche, le tri, la jointure.

— **Mémoire cache:** Il s'agit des coûts de lecture ou d'écriture des données dans la mémoire cache. En effet, Les architectures informatiques modernes disposent d'un système de mémoire hiérarchisée, comme le montre la figure 2.15. Alors que la puissance des processeurs (CPU) n'a cessé de croître au fil du temps, la latence d'accès à la mémoire centrale (DRAM) n'a guère progressé. Pour réduire cet écart de performance exponentielle entre la vitesse du processeur et la latence d'accès à la mémoire, des mémoires caches ont été introduites, composées de puces SRAM (Static Random Access Memory pour Mémoire statique à accès aléatoire en français). Le principe de fonctionnement de toutes les mémoires caches est la « localité de référence », c'est-à-dire l'hypothèse qu'à tout moment le CPU, respectivement le programme, accède de manière répété à une quantité limitée de données qui se maintient dans le cache. Seul le premier accès est « lent », car les données doivent être chargées à partir de la mémoire principale. Cela est qualifié de défaut de cache (cache miss en anglais). Les accès suivants (aux mêmes adresses de données) sont ensuite « rapides » car les données sont déjà disponibles dans le cache. Cela est qualifié de succès de cache (cache hit). Les mémoires caches sont souvent organisées en plusieurs niveaux en cascade entre la mémoire principale et le CPU.

— **Mémoire centrale:** Comme pour la mémoire cache, il s'agit des coûts de lecture ou d'écriture des données dans la mémoire principale. Ces données peuvent être des résultats intermédiaires ou toutes autres données temporaires produites/utilisées lors de l'exécution des opérations sur la BD.

— **Disque E/S:** Il s'agit du coût de la recherche, de la lecture et de l'écriture des blocs de données qui résident sur un dispositif de stockage secondaire, principalement le disque.

— **Réseau de communication:** Les coûts de communication comprennent en outre tous les coûts

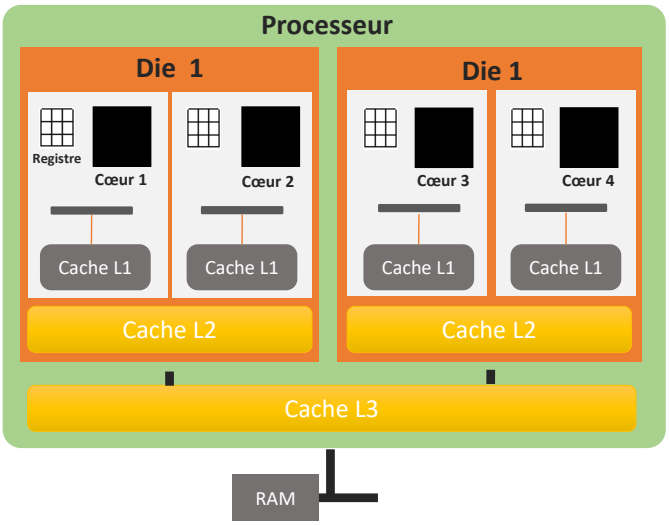


FIGURE 2.15 – Exemple d'architecture Multi-coeur.

d'envoi des (sous-)requêtes et/ou des résultats (intermédiaires) entre les différents nœuds qui participent à l'exécution de la requête.

2.4 Les paramètres sensibles à la dimension énergétique

Nous présentons dans cette section, quelques facteurs clés qui peuvent influencer la consommation énergétique du système informatique dans les SSDs lors du traitement des requêtes. Plusieurs facteurs peuvent impacter l'énergie totale consommée par le système lors du traitement de la requête, ils peuvent provenir des fluctuations du type de la requête, du type des SSDs, des caractéristiques matérielles où de la plateforme de déploiement. De plus, dans un SSD, la puissance dissipée par les serveurs est affectée par des facteurs externes, tels que la température ambiante (conditions météorologiques). La figure 2.16 résume l'ensemble des facteurs clés pouvant impacter l'énergie totale consommée par les systèmes dans les SSDs.

2.4.1 Types de la Requête

Une requête inclue comme informations sa forme (INSERT, DELETE, SELECT, UPDATE, etc.), les tables, les attributs, les opérations (sélection, jointure, agrégation, etc.). Les opérations impliquées dans l'exécution de la requête ont un impact sur la variation de la puissance consommée, car elles peuvent être implémentées de différentes manières avec des méthodes d'accès aux données variées (index, séquentiel, aléatoire, etc.). Par exemple, dans [68], les auteurs étudient puis comparent la consommation de différentes implémentations algorithmiques des opérateurs de trie et de jointure. Ils concluent que la consommation énergétique du tri par insertion et la jointure par fusion sont plus optimales que le tri rapide (ou pivot) et la jointure par boucle imbriquée respectivement.

2.4.2 Types des SSDs

Le type des SSDs réfère aux modèles et structures utilisées pour organiser et stocker physiquement les données sur les supports de sauvegarde. Il inclut également les techniques pour extraire les données et les stratégies d'optimisation. De nombreux types de SSD ont vu le jour au fil des années. Le type des SSDs impacte de manière significative l'énergie totale consommée par le système. Par exemple dans [109][8][192], étant donné la même charge de travail et sur la même configuration, les auteurs concluent que la consommation d'un type de SSD peut différer de manière significative comparée à un autre (MongoDB vs MySQL). Les nouveaux types des SSDs sont plus en plus verts car ils intègrent des techniques et des méthodes plus respectueuses de la dimension énergétique.

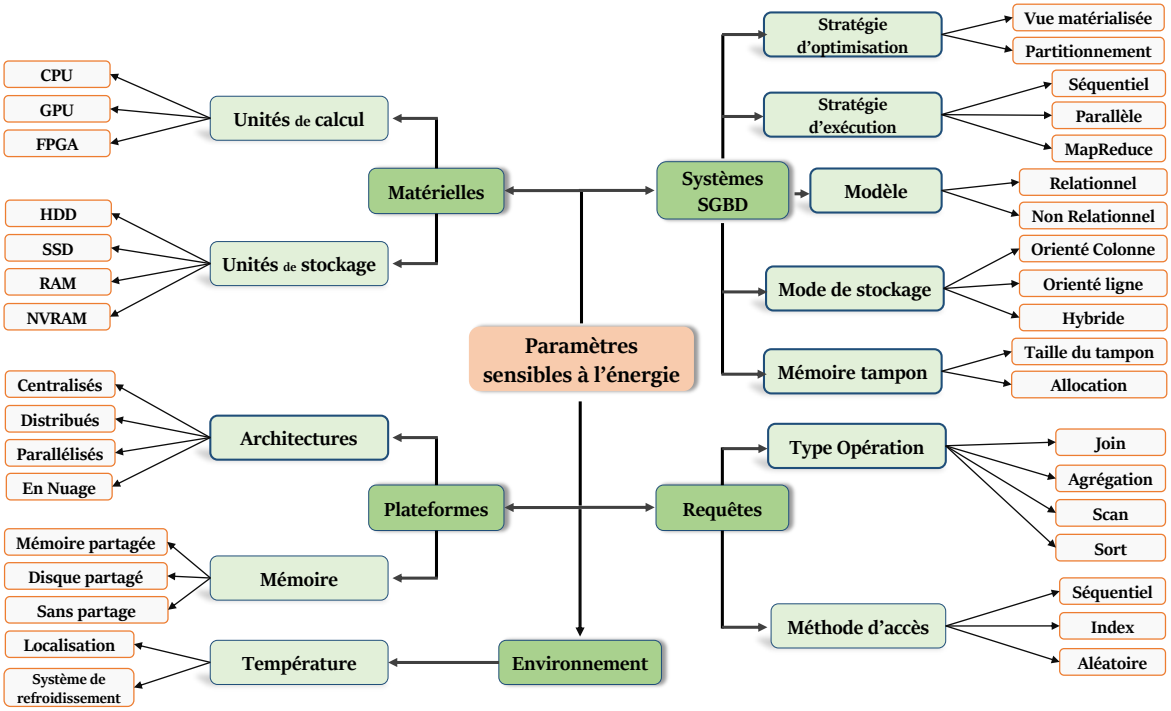


FIGURE 2.16 – Facteurs impactant la consommation énergétique dans les SSDs.

2.4.3 Caractéristiques matérielles

Les caractéristiques matérielles incluent en plus des composants informatiques tels que serveurs, routeurs, ordinateurs, dispositifs de stockage, équipement de télécommunications, ainsi que les systèmes de sécurité, etc. leur forme, le type et même leur taille. La consommation globale du système est influencée par la charge de travail (taux d'utilisation) et les composants électroniques qui le composent (processeurs, mémoire, unité de traitement graphique, etc.). La puissance active des composants (ou des systèmes) dépend fortement des techniques et des technologies utilisées lors de leur conception. L'énergie consommée par un processeur peut varier significativement d'un type à un autre appartenant à la même famille. Par exemple, la puissance consommée par le processeur Xeon E5-2680 est comparée avec celle du Xeon E5-2660 fonctionnant sur la même configuration en utilisant différents benchmarks. En activant le turbo boost, l'écart entre les puissances consommées par les deux processeurs n'est pas si considérable à cause des techniques d'optimisation énergétique comme l'ajustement dynamique de la puissance. En revanche en désactivant le turbo boost, l'écart énergétique croit énormément [179]. Multiples facteurs peuvent être à l'origine de ces écarts énergétiques, dans [159], les auteurs identifient que certains facteurs architecturaux comme le style logique, la conception logique, le dimensionnement des circuits et des fils, le placement et la cartographie des circuits électroniques, contribuent à la consommation d'énergie des composants. Les composants, les plus énergivores ou très sensibles à l'énergie dans les SSDs sont les unités de traitement et les périphériques de stockage dans une architecture centralisée.

2.4.4 Configurations architecturales

Il existe plusieurs types de plateforme ayant pour but de fournir des moyens adéquats pour traiter les données. Le choix d'une configuration architecturale destinée à supporter la BD doit être guidé par plusieurs facteurs (objectifs, budgets, sécuritaire, etc.) puisqu'à mesure que les données croissent, la complexité des applications qui ont besoin de manipuler ces informations pour en déduire de nouvelles connaissances augmente également. L'énergie totale résultante lors du traitement des requêtes est fortement influencée par l'architecture informatique sous-jacente sur laquelle le SSD fonctionne, en particulier par des aspects tels que la mise en réseau, le parallélisme et la distributivité [166].

Nombreuses études ont analysé la performance et l'efficacité énergétique des plateformes de configuration, par exemple Francesquini et al.[42] ont mené une étude de comparaison entre quatre plateformes de

configuration (Intel Xeon E5, SGI Altix UV 2000, Samsung Exynos 5, and Kalray MPPA-256) en utilisant trois différents algorithmes (le problème du voyageur de commerce, Le partitionnement en k-moyennes, La simulation des ondes de propagation sismiques (Ondes3D)). Leurs résultats montrent d'importantes variations énergétiques entre les différentes plateformes pour le même algorithme. Pour chaque algorithme, l'architecture MPPA-256 présenterait la meilleure solution énergétique en consommant moins d'énergie que les autres plateformes (optimisant jusqu'à 6,9 fois comparée à celle de Exynos 5).

2.4.5 Influences environnementales

Au fil des années, la charge de travail était considérée comme la caractéristique principale qui faisait varier l'énergie totale consommée par les composants (CPU, mémoires, routeur, etc.). Cependant certaines études mettent de plus en plus l'accent sur d'autres facteurs externes contributeurs tels que la température ambiante de l'environnement [129][136], l'architecture du bâtiment [136], l'âge des composants électroniques[33], etc. Patterson et al.[136] ont conclu que l'énergie consommée par le serveur est fortement influencée par la température ambiante de deux manières possibles: (1) à travers des composants très sensible à la température (CPU, mémoire) et (2) travers les systèmes de refroidissement interne.

2.5 Conclusion

Dans ce chapitre, nous avons donné un aperçu sur les systèmes de traitement de requêtes (STRs) dans les SSDs, en nous focalisant plus sur le modèle relationnel et ses architectures de déploiement. Nous avons commencé par donner une vue générique des systèmes de traitement des requêtes (STRs), puis nous avons décrit les différentes étapes de traitement des requêtes en fonction de leur architecture de déploiement (centralisée, distribuée) et de leur intégration avec d'autres modèles (multi-bases, multi-stores). Cette étude révèle que malgré quelques étapes de plus dans certaines configurations, les principales étapes restent similaires. L'audit des étapes nous a permis de comprendre les différentes opportunités qu'on pourrait exploiter pour réduire la consommation énergétique du système lors du traitement des requêtes. Dans toutes les étapes de traitement d'une requête, l'optimisation est l'une des phases la plus complexe et la plus coûteuse.

Puis, dans la section suivante, nous avons abordé le principe fondamental des modèles de coût en décrivant ses composants et en présentant les paramètres clés qu'il utilise pour estimer le coût des requêtes lors de l'étape d'optimisation. L'objectif de ce travail est d'accroître l'efficacité énergétique du système lors du traitement de la requête en exploitant un modèle de coût précis, adéquat qui prend en charge les paramètres matériels et logiciels. Pour ce faire, nous devons identifier les principaux facteurs qui peuvent influencer la consommation énergétique du système lors du traitement des requêtes. C'est ainsi que dans la dernière section, nous avons étudié les paramètres sensibles à l'énergie lors du traitement de la requête dans les SSDs.

Au cours de cette étude, nous avons pu comprendre qu'il existe de nombreuses opportunités permettant de réduire la consommation énergétique d'un système lors du traitement de la requête, cela inclut la conception d'un optimiseur éco-énergétique. L'optimisation énergétique par le biais d'un optimiseur éco–énergétique conduit à la proposition d'un modèle énergétique précis pouvant être influencés par de nombreux facteurs tels que: les paramètres de la requête, les paramètres du SSD, la configuration matérielle et l'architecture de déploiement et la technique d'intégration de ce modèle.

Dans le chapitre suivant, nous présentons les concepts fondamentaux de l'énergie et discutons des problèmes généraux de l'efficacité énergétique. Puis nous donnons un état de l'art sur les travaux qui optimisent la consommation énergétique dans les SSDs. Ces travaux sont principalement basés sur la gestion efficiente des facteurs qui influencent la consommation énergétique du système.

CHAPITRE 3

Gestion de l'énergie dans les Systèmes de Traitements des Requêtes

Car il donne à l'homme qui lui est agréable la sagesse, la science et la joie; mais il donne au pécheur le soin de recueillir et d'amasser, afin de donner à celui qui est agréable à Dieu. C'est encore là une vanité et la poursuite du vent

— Ecclésiaste 2.26

Sommaire

3.1 Introduction 49
3.2 Préliminaire 49
3.2.1 Concept de l'énergie 49
3.2.2 Efficacité énergétique 50
3.2.3 Motivations 50
3.2.3.1 Problème lié à la consommation excessive de l'énergie 50
3.2.3.2 Bénéfice de l'efficacité énergétique 51
3.2.4 Méthode d'évaluation de l'EE 51
3.2.4.1 Métriques énergétiques 51
3.2.4.2 Modèles énergétiques 52
3.3 Taxonomie des Techniques éco-énergétiques 55
3.3.1 Solutions matérielles 55
3.3.1.1 Technique de l'ajustement dynamique de la tension (DVFS) . 56
3.3.1.2 Désactivation Dynamique des Composants 57
3.3.1.3 Co-traitements 58
3.3.2 Solutions logicielles 58
3.3.2.1 Technique de virtualisation 59
3.3.2.2 Technique de l'infonuagique (Cloud Computing) 59
3.3.3 Gestion de la température 60
3.3.4 Standards et Conduite 60
3.4 Approches d'EE dans les STRs 61
3.4.1 Approches Orientées Matérielles 61
3.4.1.1 Les unités de traitement 61
3.4.1.2 Technique de Co-traitement 63
3.4.1.3 Les unités de stockage 63
3.4.2 Approches Orientées Logicielles 64
3.4.2.1 Traitement éco-énergetique 64
3.4.2.2 Conception physique éco-énergetique 67
3.5 Conclusion 68

Ce chapitre présente un état des lieux des principaux travaux traitant la problématique de l'efficacité énergétique dans les systèmes informatiques, en général, et les SSDs en particulier. Nous commençons par donner les différentes métriques pour évaluer l'efficacité énergétique après avoir rappeler le concept de base sur l'énergie et sur l'efficacité énergétique. Ensuite, nous présenterons une taxonomie des récentes techniques de réduction de la consommation énergétique au niveau des SSDs en nous focalisant plus sur la gestion de l'énergie dans les STRs. Ces techniques sont axées sur les approches matérielles, les approches logicielles (virtualisation, délocalisation des traitements, gestion de compromis entre l'énergie et la performance) et les solutions environnementales (énergie renouvelables).

3.1 Introduction

Les infrastructures numériques sont parfois utilisées comme une solution éco-énergétique, mais néanmoins elles ne sont pas en marge dans l'exploitation des ressources naturelles et de l'énergie. L'expansion du monde numérique se combine avec une forte augmentation de l'empreinte énergétique considérant l'énergie nécessaire à leur production dans les usines, leur utilisation par les utilisateurs et leur destruction. Ce phénomène est exécrable dans les infrastructures dédiées aux calculs intenses telles que les clusters et les centres de données. On estime que la consommation énergétique des centres de données a augmenté de 56% entre 2005 et 2010, et qu'en 2014, les centres de données américaines ont à eux seuls consommé jusqu'en environ 70 milliards de kilowattheures, soit environ 1,8% de la consommation globale [167]. Toutefois, pour répondre aux problèmes écologiques tout en continuant à fournir la demande de service en matière de gestion des données, les technologies ont évolués vers des moyens de préservation de l'énergie connue sous le nom de solutions éco-énergétiques. C'est ainsi que l'approche standard qui consistait à améliorer les performances des systèmes informatiques (minimiser le temps de réponse lors du traitement des données) a convergé vers la satisfaction de deux objectifs non-fonctionnelles (NF), cruciales et conflictuelles à savoir: (1) un traitement rapide du déluge des données issues des sources d'entreprise, des réseaux sociaux, de l'Internet des objets, etc. et (2) une consommation d'énergie optimale sur ces SSDs pour contribuer aux objectifs écologiques fixés pour sauver notre planète.

De nos jours, l'efficacité énergétique devient de plus en plus critique voir même une contrainte de vente dans les systèmes informatiques. De nombreux efforts ont été fait à différents niveaux comprenant, la sélection des composants chimique éco-énergétique, la conception optimale des circuits logiques, et la conception efficiente des algorithmes ainsi que la définition des techniques de gestion de compromis entre l'énergie et la performance, etc. dans le seul but de maximiser l'efficacité énergétique. Ce chapitre traite les différents moyens explorés pour réduire la consommation énergétique dans les systèmes informatiques. Nous évoquerons les travaux récents portant sur l'efficacité énergétique au niveau des composants matériels, des logiciels de base (système d'exploitation). L'objectif principal de ce travail est de donner un aperçu récent des différents progrès de la recherche sur l'efficacité énergétique dans les SSDs, principalement dans les systèmes de traitement des requêtes (STRs).

3.2 Préliminaire

Afin de mieux appréhender le problème de l'efficacité énergétique, nous commençons par décrire dans cette section quelques notions fondamentales sur l'énergie et sur la puissance puis nous discutons des conséquences néfastes d'une consommation excessive de l'énergie. Puis nous introduisons les métriques et méthodes les plus utilisées pour évaluer et comparer l'efficacité énergétique dans les SSDs.

3.2.1 Concept de l'énergie

Parfois La puissance et l'énergie sont utilisées de manière interchangeable bien qu'elles soient utilisées pour quantifier la même grandeur, elles sont tout de même différentes. L'énergie est définie comme la capacité d'un système à réaliser un travail. Elle se mesure en Joules (J) alors que la puissance se quantifie en Watts(W) et elle est définit comme la quantité d'énergie consommée par unité de temps par le système. La puissance reflète la vitesse à laquelle un travail est fourni ou la dérivée du travail au fil du temps. Le travail est lié à la quantité d'énergie transférée a un système par une force. L'énergie et le travail ont la même unité de mesure. *Formellement, l'énergie et la puissance peuvent être définies comme suit:*

$$P(t) = d/dtE(t), \quad E(t) = \int_0^t p(\tau)d\tau \tag{3.1}$$

où P, t et E définissent respectivement la puissance, le temps et l'énergie. En mesurant l'énergie et le temps qu'il faut pour un programme de s'exécuter du début jusqu'à la fin, la puissance moyenne est calculée par le ratio des deux valeurs. La réduction de la puissance consommée ne conduit pas nécessairement à une réduction de l'énergie totale consommée. En électronique, la puissance électrique d'un système peut être divisée en deux catégories : (a) la puissance statique (aussi appelée inactive ou passive) et (b) la puissance dynamique (aussi appelée puissance active ou de commutation) [48].

1. **La puissance statique (inactive):** Elle représente la puissance consommée par les composants du système à l'état inactif. Elle provient des courants de fuite (*Power leakage*) et se définit comme

étant la puissance électrique qui s'accumule au niveau des isolateurs et traverse les conducteurs qui n'étaient pas destinés à cela. Néanmoins cette puissance reste inférieure au courant qui traverse le système lorsqu'il est en position active.

2. **Puissance dynamique (active)**: La puissance dynamique (*Power dynamic*) est la consommation d'énergie requise pour que le système accomplisse ses fonctions. En d'autres termes, c'est la puissance nécessaire pour charger et décharger les nœuds dans un circuit. Les circuits à capacité commutée sont la principale cause de la consommation d'énergie active.

Au fil des années, le concept d'énergie intelligente est devenu une préoccupation majeure des politiques énergétiques et climatiques. La question fait aujourd'hui un grand débat au sein de la société. Elle concerne notamment les secteurs d'activités suivants: le transport, le bâtiment et l'industrie. Le concept de l'énergie intelligente passe par la recherche de la moindre intensité énergétique (à service égal), une « utilisation rationnelle de l'énergie », des processus et outils plus efficaces c'est-à-dire moins énergivores. Pour designer cet état de fonctionnement, on parle d'efficacité énergétique ou efficience énergétique.

3.2.2 Efficacité énergétique

L'efficacité énergétique évoque l'utilisation optimale de l'énergie par un système pour rendre le même service. L'efficacité énergétique est importante, car elle permet d'une part de faire plus de travail avec la même quantité énergétique et d'autre part de contribuer à la préservation de l'environnement.

L'efficacité énergétique (EE) se définie par le rapport de la performance, mesurée en taux de travail effectué (TE), sur la quantité d'énergie utilisée [177].

$$EE = \frac{TE}{E} = \frac{TE}{P \times t} \tag{3.2}$$

On peut maximiser l'efficacité énergétique en appliquant les techniques de gestion statique de la puissance (SPM), et/ou les techniques de gestion dynamique de la puissance (DPM). Partant de l'équation 3.2, dans un système informatique, l'efficacité énergétique se joue avec l'ajustement des deux paramètres à savoir la performance et l'énergie consommée.

Deux concepts de puissance électrique doivent être pris en considération lors de l'évaluation de l'efficacité énergétique dans un système: la puissance moyenne représentant la puissance moyenne consommée au cours de l'exécution d'une charge de travail, ou le pic représentant la puissance maximale (peak power). Dans cette thèse, nous considérons la puissance moyenne.

3.2.3 Motivations

Dans cette section, nous donnons quelques grandes lignes de motivations autour des études d'efficacité énergétique.

3.2.3.1 Problème lié à la consommation excessive de l'énergie

L'énergie est à la fois une solution et un problème pour le développement durable. Elle contribue au développement économique, à la réduction de la pauvreté, à l'éducation et à l'amélioration générale de la qualité de vie et de celle de l'humanité mais elle est aussi l'une des principales causes de la pollution de l'air et d'autres nuisances sur la santé humaine et sur l'environnement. Les progrès technologiques ont radicalement changé le monde de diverses manières. Cependant, cela a également conduit aux développements de nombreux problèmes environnementaux, qui menacent l'homme et la nature. La sécurité énergétique, la croissance économique et la protection de l'environnement sont les moteurs de la politique énergétique nationale de tous les pays du monde (appelé *syndrome du trilemme*) [160]. À mesure que la population mondiale croisse, beaucoup plus rapidement que la moyenne estimée à 2%, le besoin en énergie est de plus en plus exacerbé (voir Figure 3.1). L'amélioration du mode de vie et la demande d'énergie augmentent ensemble; la riche économie industrialisée, qui contient 25% de la population mondiale, consomme 75% de l'approvisionnement énergétique mondial. La consommation d'énergie mondiale actuelle est estimée à 22×10^9 kWh par an. Cette consommation équivaut environ à une émission de 6.6×10^9 tonnes de dioxyde de carbone(CO_2) dans l'atmosphère [128]. Le CO_2 est un énorme pourvoyeur de gaz à effet de serre(GES). L'organisation de coopération et de développement économique(OCDE) prévient que compte tenu des tendances actuelles, les émissions liées à l'énergie augmenteront de 70% d'ici 2050. Cela peut accélérer les conséquences négatives du changement climatique, notamment des températures plus élevées et une augmentation de la fréquence des événements extrêmes.

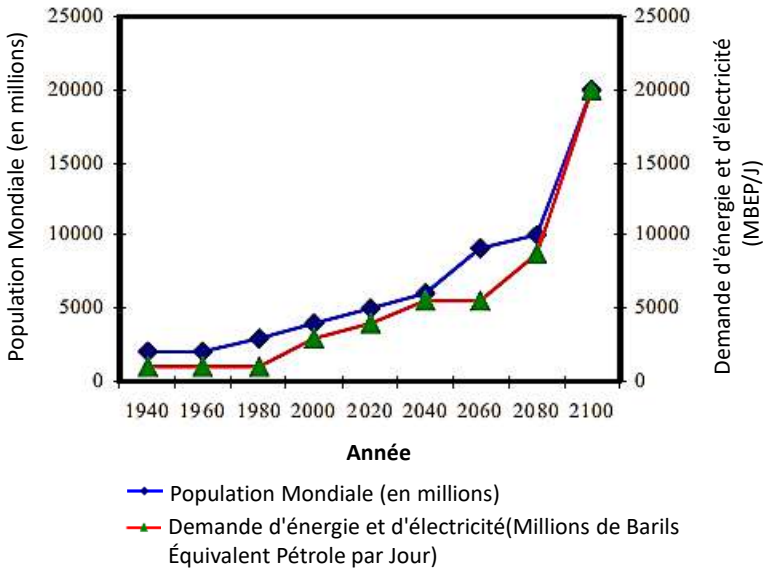


FIGURE 3.1 – Estimation annuelle de la population mondiale et de la demande énergétique [128]

Les principaux problèmes environnementaux sont la pollution de l'eau, le rayonnement et la radioactivité, l'élimination des déchets solides, pluies acides, l'appauvrissement de la couche d'ozone stratosphérique et le réchauffement climatique (effet de serre, changement climatique) [128]. Plus de la moitié de la population mondiale vit à moins de 60 km de la mer. Les gens peuvent être forcés à se déplacer, ce qui augmenterait des troubles mentaux et même favoriser la propagation des épidémies.

3.2.3.2 Bénéfice de l'efficacité énergétique

Nous avons brièvement évoqué dans la sous-section 3.2.3.1 les conséquences liées à la sur-consommation d'énergie sur notre environnement et sur notre santé. L'efficacité énergétique est l'un des moyens les plus rapides et les plus rentables pour économiser de l'argent, réduire les émissions de gaz à effet de serre et répondre à la demande croissante d'énergie. Il peut également avoir de nombreux autres avantages socio-économiques et sanitaires, tels que des maisons plus chaudes et plus saines, longévité des biens et des personnes, des factures d'électricité et des coûts de fonctionnement de l'entreprise moins élevés et indirectement la promotion des emplois.

La réduction de la consommation d'énergie dans les systèmes garantisse la longévité de ces composants électroniques. En fait, la dissipation d'énergie a un effet néfaste sur la fiabilité des circuits électroniques [119]. Le dysfonctionnement ainsi que la détérioration des composants sont causés par de nombreux facteurs, notamment par une forte élévation de la température, une surtension aux bornes des dipôles électriques, etc. Le taux de défaillance des équipements augmente lorsque sa température dépasse 15° Celsius [28].

3.2.4 Méthode d'évaluation de l'EE

Dans l'objectif de pouvoir quantifier différentes approches éco-énergétiques, diverses métriques et outils(benchmarks) ont été proposées afin d'assister les administrateurs dans le processus d'évaluation de l'EE et de comprendre les impacts de leur décision. Dans cette section, nous évoquerons de manière très brève certaines métriques de l'EE utilisée dans les SSDs. Puis nous présenterons les différentes approches pour modéliser l'énergie dynamique consommée par un système, nécessaire lors des perspectives d'amélioration de l'efficacité énergétique ou de la conception des outils des bancs d'essai.

3.2.4.1 Métriques énergétiques

Plusieurs métriques ont été proposées et utilisées dans la littérature pour évaluer l'efficacité énergétique des SSDs. Ces métriques sont généralement catégorisées en des métriques fondées: (1) sur l'exploitation des ressources du système, (2) sur l'énergie totale consommée, (3) sur la chaleur dégagée par le système lors de

son fonctionnement [27]. Ci-dessous, nous en donnons quelques exemples de métriques avec leurs définitions.

(i) **Énergie totale consommée (ETC)**
L'énergie totale consommée réfère à la puissance totale consommée par le SSD pendant un laps de temps. Elle est généralement mesurée en watts par heure (W/h) et se définit ainsi:

$$ETC = \int_{T_1}^{T_2} PC(t)\,\mathrm{d}t$$

Où ETC correspond à l'énergie totale consommée entre T_1 et T_2, et PC définit la puissance consommée.

(ii) **Efficacité de l'infrastructure des SSDs (DCiE)**
Green-Grid (une organisation à but non lucratif de professionnels) a proposé la métrique $DCiE$ pour déterminer l'efficacité énergétique d'un SSD. Elle est exprimée en pourcentage du ratio de l'énergie consommée par les équipements informatiques à l'énergie totale consommée par l'infrastructure. L'énergie consommée par l'infrastructure est mesurée directement du compteur du fournisseur de l'énergie.

$$DCiE = \frac{EnergieEquipementIT}{EnergieInfrastructure} \times 100$$

(iii) **Power Usage Effectiveness (PUE)**
La métrique PUE est définie simplement par la réciproque de la métrique de DCiE. Elle fut introduite par Green-Grid en 2006 [7] et reste la métrique la plus adoptée dans les SSDs. La PUE est utilisée en deux étapes:(1) lors de la conception initiale du projet pour évaluer le potentiel du rendement énergétique puis (2) après la conception pour surveiller et améliorer l'efficacité énergétique afin d'aider les administrateurs à amoindrir les coûts énergétiques. Une valeur idéale de PUE serait 1, ce qui signifierait que 100% de l'électricité fournie au compteur est utilisée par les équipements informatiques. Cependant, la valeur 1 est irréalisable, car l'énergie est aussi nécessaire pour faire fonctionner d'autres équipements tels que les systèmes de refroidissement, l'éclairage, etc.

(iv) **Space, Watts, and Performance (SWaP)**
SWaP (Espace, puissance électrique et performance) est une métrique définie par Sun Microsystems pour les SSDs, prenant en compte ces trois paramètres:

$$SWaP = \frac{Performance}{Space \times Power}$$

La performance est mesurée avec un benchmark approprié, et l'espace est la taille de l'ordinateur. SWaP est flexible car il peut facilement être déployer sur n'importe quel environnement de SSDs en utilisant des paramètres standards définis par l'utilisateur.

3.2.4.2 Modèles énergétiques

Modéliser la consommation d'énergie réelle au niveau de l'ensemble du système ou au niveau de ces composants individuels (processeur, mémoire, disque, carte graphique, réseau, etc.) n'est pas une tâche commode, car elle dépend de nombreux facteurs, tels que la charge de travail, l'équilibre du système et les paramètres environnementaux (voir figure 2.16). Les modèles énergétiques pour les systèmes informatiques peuvent être représentés sous la forme d'équations, de modèles graphiques, de règles, d'arbres de décision, d'ensembles d'exemples représentatifs, de réseaux de neurones, etc. Le choix de la représentation affecte la précision du modèle, ainsi que leur interprétation par les personnes. Pour des fins d'amélioration de l'efficacité énergétique, le modèle doit être précis, rapide, générique, portable, simple, non intrusif et à faible surcharge [29]. Les modèles peuvent être utilisés pour plusieurs objectifs, par exemple pour des études de prévision des tendances de l'efficacité énergétique ou pour des besoins d'optimisation de la consommation d'énergie.

Des efforts considérables ont été déployés aussi bien dans le milieu académique qu'industriel en ce qui concerne les métriques de comparaison et les techniques de modélisation énergétique. Ces efforts peuvent être regroupés en quatre grandes catégories [180]: modèle de simulation, modèle analytique détaillé, modèle en boîte noire et les benchmarks.

(i) **Approches basées sur la simulation**
La modélisation basée sur la simulation fut beaucoup utilisée pour prédire et calibrer la consommation énergétique de plusieurs entités de notre vie comme les bâtiments, les véhicules, etc. [183]. Compte tenu de la complexité pour extraire les informations détaillées du fonctionnement des nombreux composants (matériel, les logiciels, les applications et les environnements externes) caractérisant le système, les approches fondées sur la simulation ont été développées pour amoindrir cette complexité en modélisant chaque composant de manière isolée plutôt qu'en un ensemble [180]. Cette approche est peu coûteuse mais elle n'offre pas une portabilité adéquate car elle repose sur des connaissances spécifiques à chaque composant. Cette catégorie de modélisation inclue les outils comme: *Wattch* proposé par Brooks et al. en 2000 [17]: un framework pour l'analyse et l'optimisation de la puissance des microprocesseurs, *Tempo Simulator* proposé par Shafi et al. [164]: un framework pour simuler la consommation d'énergie du système embarqué PowerPC 405GP, et *SoftWatt Simulator* proposé par Gurumurthi et al. [59]: un outil qui modélise la consommation énergétique du processeur, des mémoires et du sous-système de stockage des données puis quantifie l'énergie nécessaire au fonctionnement des applications et du système d'exploitation.

(ii) **Approches Analytiques**
L'objectif des approches analytiques est de modéliser un système comme un ensemble d'équations mathématiques (appelés modèles de coût) qui spécifient les relations entre les paramètres et leurs valeurs associées en fonction du temps, de l'espace et/ou d'autres paramètres du système [43]. Cette approche nécessite une compréhension détaillée du fonctionnement interne des systèmes. Par exemple dans les BDs, les modèles de coût jouent un rôle crucial dans l'optimisation des requêtes pour obtenir un plan d'exécution efficace. Ils se composent de deux parties: une composante logique et une composante physique [111]. Par exemple, l'équation mathématique suivante peut être utilisée pour estimer le coût de traitement (CT) d'une tâche donnée (ts) sur un système informatique :

$$CT(ts) = \alpha \times I^{ts}_{cpu} + \beta \times I^{ts}_{cache} + \gamma \times I^{ts}_{mem} + \delta \times I^{ts}_{io} + \sigma \times I^{ts}_{com} + \epsilon^{ts}$$

Où I^{ts}_{cpu}, I^{ts}_{cache}, I^{ts}_{mem}, I^{ts}_{io}, I^{ts}_{com}, et ϵ^{ts} représentent respectivement, le nombre d'instructions du processeur, les défauts de cache du processeur, l'utilisation de la mémoire, le taux d'E/S du disque, la quantité des données transférées sur le réseau et le cumul des erreurs d'estimation. Les constantes α, β, γ, δ, et σ sont obtenues à partir de l'apprentissage automatiques en utilisant les techniques de machine learning [117].

(iii) **Approches à boite noire**
Cette approche permet de construire des modèles de consommation énergétique sans connaître les spécifications internes du principe de fonctionnement des systèmes. Des techniques basées sur l'analyse des données historiées sont utilisées pour construire des modèles de prédiction en suivant principalement quatre étapes (figure 3.2): extraction des paramètres, construire des modèles, validation puis exploitation dans des études de prévisions [29]. Une variété de modèles basés sur cette approche a été proposée, généralement dans l'objectif d'accroître l'efficacité énergétique sur un composant ou une machine particulière [149]. En exemple les travaux d'Economou et al. [38] ont proposé un modèle énergétique en corrélant les mesures de la puissance énergétique avec les taux d'utilisation des composants du système et Lang et al.[98] ont proposé un modèle de coût pour la consommation d'énergie d'un cluster MapReduce, en corrélant les caractéristiques de la charge de travail et les spécifications de la configuration sous adjacentes.

(iv) **Benchmarking**
Les Benchmarks énergétiques permettent d'estimer la consommation énergétique des composants du système ou du système tout entier, très généralement proposés par des organismes gouvernementaux ou des consortiums d'industrie dans le but de maximiser le rendement de l'efficacité énergétique. Parmi les plus connus, nous citons: Transaction Processing Performance Council (TPC), Standard Performance Evaluation Corporation (SPEC) et Storage Performance Council (SPC). Ils ont pour objectifs principaux de (i) de définir des normes pour comparer les meilleures techniques de l'EE, (ii) d'exploiter les potentiels d'amélioration de la gestion dynamique de l'énergie, et (iii) de réaliser des optimisations énergétiques [149]. Cette section donne une description détaillée de quelques benchmarks connus dans la littérature(voir figure 3.3).

— ***Transaction Processing Performance Council (TPC):*** Le Transaction Processing Performance Council (TPC) est un organisme à but non lucratif fondé en 1988 dans le but de définir des benchmarks pour les BDs. Le TPC offre TPC-C et TPC-E pour mesurer les performances des systèmes transactionnels en ligne (OLTP) et TPC-H pour mesurer les performances des systèmes d'aide à la décision. La spécification TPC-Energy fut proposée en 2007 pour étendre les benchmarks

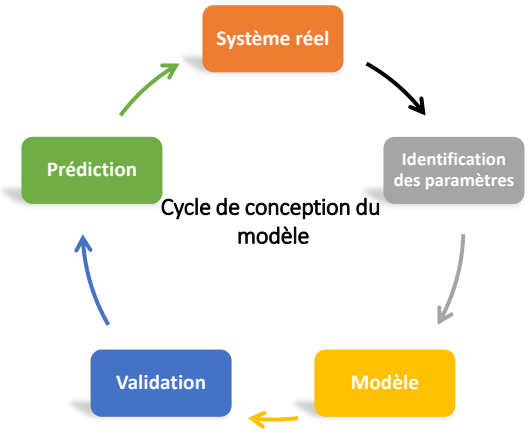


FIGURE 3.2 – Approche générale pour construire et utiliser un modèle d'énergie

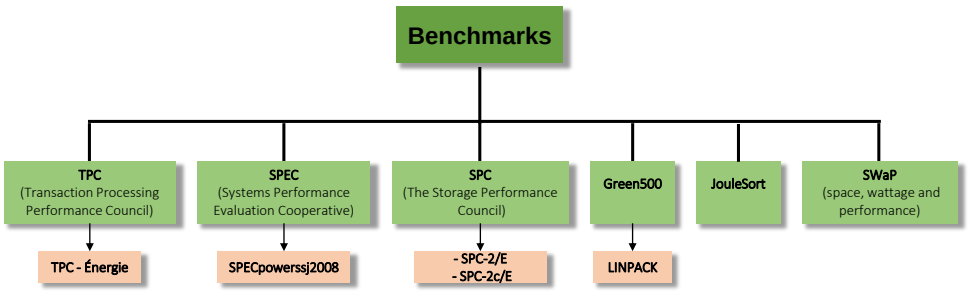


FIGURE 3.3 – Aperçu de quelques benchmarks énergétiques disponible dans la littérature

TPC existants en prenant en compte les mesures énergétiques, afin d'assister les utilisateurs finaux à identifier les équipements éco-énergétiques qui répondent à la fois à leurs exigences informatiques et budgétaires.

— ***Systems Performance Evaluation Cooperative (SPEC):*** SPEC a été fondée en octobre 1988 par Apollo, Hewlett-Packard, MIPS et Sun Microsystems. Il s'agit d'une coopération à but non lucratif créée pour établir, maintenir et approuver un ensemble standardisé de benchmarks pertinents pouvant être appliquées à la dernière génération d'ordinateurs de haute performance, y compris des benchmarks gourmands en processeur, des benchmarks pour mesurer les performances graphiques, etc. SPEC a publié le benchmark SPECpowerssj2008, le premier benchmark industriel pour mesurer les performances et la consommation d'énergie des systèmes en utilisant des niveaux de charge graduelles. Plus précisément, le benchmark mesure les performances et l'énergie de 11 niveaux de charge allant de 0% à 100% en exploitant toute la capacité du serveur pour traiter les transactions commerciales avec une application Java [71] puis agrège les mesures par la moyenne géométrique pour former un seul résultat. Le benchmark exerce les CPU, les caches, la hiérarchie de mémoire sur plusieurs niveaux de charge. Le benchmark fonctionne sur une grande variété de systèmes d'exploitation et d'architectures matérielles avec la prise en charge des systèmes distribués.

— ***The Storage Performance Council (SPC):*** Le Storage Performance Council(SPC) plus focalisé sur l'évaluation des composants de stockage, a défini en 2011 des extensions de ces benchmarks SPC Benchmark 2(SPC-2) et SPC Benchmark 2C(SPC-2C) pour inclure la mesure et le reporting de la consommation d'énergie en plus des performances de stockage pour les applications séquentielles. Ces extensions furent nommées SPC Benchmark 2/Energy(SPC-2/E) et SPC Benchmark 2C/Energy (SPC-2C/E) respectivement.

— ***JouleSort:*** JouleSort est un benchmark centrée sur les E/S qui mesure l'efficacité énergétique des systèmes à leur maximum d'utilisation. Il fut proposé par Rivoire et al.[149] en 2007 pour évaluer l'efficacité énergétique de différents algorithmes de tri. Le benchmark JouleSort est un benchmark de tri, une extension du Sort Benchmark [1], qui est utilisé pour mesurer la performance

1. http://sortbenchmark.org/

et le coût-performance des systèmes informatiques.

3.3 Taxonomie des Techniques éco-énergétiques

De nombreux efforts ont été menés pour améliorer l'EE dans les systèmes informatiques. Les techniques d'EE dans les systèmes sont divisées en deux grandes parties: (1)Gestion statique de l'énergie(GSE) et (2)Gestion dynamique de l'énergie (GDE). La gestion statique cherche à améliorer l'EE lors de la conception des composants, ces techniques s'appuient sur l'optimisation des schémas logiques des circuits électroniques, sur la miniaturisation des composants alors que celles de la gestion dynamique cherchent à réguler la consommation énergétique lors du fonctionnent du système [108]. Dans notre travail, nous nous sommes focalisés sur les techniques basées sur la gestion dynamique de la consommation énergétique que nous avons divisé en quatre grandes catégories (figure 3.6) sur la base de leur niveau de déploiement: (1) solutions matérielles, (2) solutions logicielles, (3) gestion de l'environnement et (4) les normes de conduite. Nous décrivons les efforts investis dans la quête de l'efficacité énergétique dans chacune des catégories puis nous abordons plus en détails les techniques d'optimisations énergétiques dans les STRs. La Figure 3.4 illustre l'interdépendance des différents niveaux qui ont un impact sur la consommation d'énergie des systèmes informatiques.

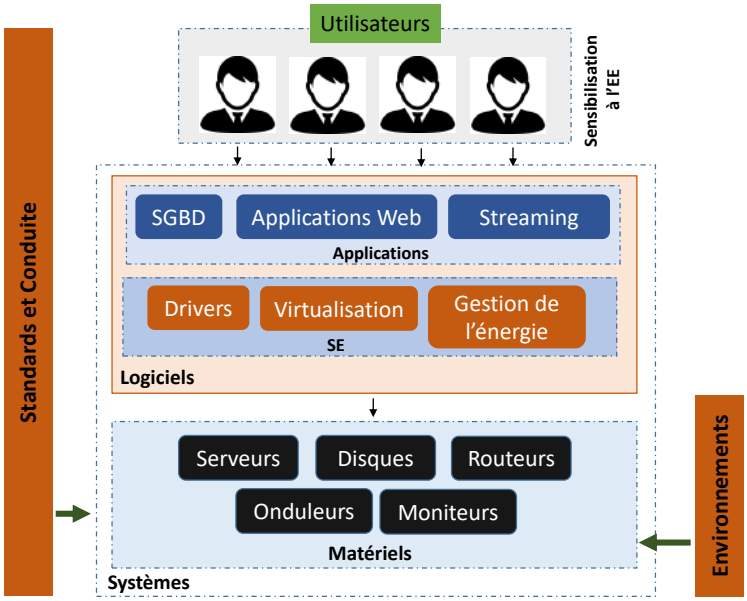


FIGURE 3.4 – Interdépendance des différents niveaux ayant un impact sur la consommation énergétique des systèmes informatiques

3.3.1 Solutions matérielles

Les solutions appliquées au niveau matériel et micro-architectural peuvent être scindées en trois alternatives:(a) Désactivation Dynamique des Composants (DDC), (b)l'Ajustement Dynamique de la Performance (ADP) et (c) la combinaison des unités de traitement (co-processing) [12]. Certains composants informatiques n'ont pas la capacité d'ajuster automatiquement leurs performances, ces derniers doivent être complètement désactivés lors des périodes d'inactivités pour réduire leur consommation énergétique. Cette approche s'appelle la désactivation dynamique des composants. L'ajustement dynamique des performances correspond à l'ajustement dynamique de la tension ($DVFS$ pour Dynamic Voltage and Frequency Scaling). Les deux premières alternatives sont applicables aussi bien au fonctionnement d'un serveur qu'aux équipements du réseau informatique. Nous présenterons dans cette section, l'état de l'art des récentes études basées sur les solutions matérielles classifiées selon les trois alternatives.

3.3.1.1 Technique de l'ajustement dynamique de la tension (DVFS)

La puissance dynamique dissipée par unité de temps par un circuit dépend fortement de la tension et de la fréquence en partant de sa définition mathématique formulée par l'équation:

$$Puissance_{Dynamique} = P_{statique} + C \times f \times V^2$$

Où C est la capacité du circuit en farads (F), f est la fréquence, en hertz (Hz) et V la tension de l'alimentation en volt (V) [48]. Les techniques DVFS exploitent en fait la corrélation existante entre la consommation d'énergie ($Puissance_{Dynamique}$), la tension V fournie et la fréquence de fonctionnement f, pour gérer de manière dynamique la consommation des processeurs multi-cœurs, des mémoires DRAM [2] et d'autres composants dans les SSDs. En effet, en ajustant intelligemment la fréquence, la tension de l'alimentation requise peut être réduite, ce qui conduirait à des économies d'énergie importantes en raison de la corrélation entre les grandeurs physiques.

La technique de DVFS est largement appliquée dans les approches d'efficacité énergétique des composants micro-architecturaux(CPU, mémoire, etc.), dans le traitement éco-énergétique des paquets du réseau d'interconnexion, dans le problème de planification de l'exécution des tâches en vue d'optimiser la consommation d'énergie tout en satisfaisant la qualité des services requises. Ces travaux incluent [69],[168],[138],[32], [124], [60] ,[193], [105], [124].

Dans [194], les auteurs proposent Swan (two-Step poWer mAnagement for distributed search eNgines), un outil de gestion de l'énergie pour les moteurs de recherche distribués basé sur la technique DVFS dans le but de réaliser des économies d'énergie. Pour ce fait, ils commencent par définir un modèle de régression linéaire pour estimer le temps d'exécution de chaque requête en fonction de la longueur de la liste des mots clés caractérisant la recherche. Pour chaque requête, Swan après avoir estimé le temps d'exécution de la requête, sélectionne une fréquence initiale (la plus basse) pour optimiser la consommation énergétique, puis ajuste la fréquence du processeur de manière appropriée au moment opportun lors de l'exécution de la requête pour respecter les délais d'exécution requis. Les expérimentations sur un système multi-cœurs de l'implémentation de l'algorithme Swan dans le moteur de recherche Solr [3] atteignent jusqu'à 39% d'économie d'énergie du CPU.

Dans le même objectif que [194], les auteurs de [70] utilisent un algorithme qui adapte automatiquement la variation de la tension et de la fréquence du processeur (CPU) tout en maintenant le niveau de la perte de performance limitée. Cet algorithme utilise un modèle linéaire corrélant l'intensité appliquée aux bornes des circuits au temps total d'exécution des tâches afin de calculer la fréquence du CPU la plus basse permettant de faire des optimisations énergétiques. L'évaluation des performances de leur algorithme sur les systèmes mono-processeurs et multiprocesseurs révèle que le système peut optimiser l'énergie jusqu'à 20% et 25% de l'énergie consommée par le processeur en utilisant des benchmarks séquentiels et parallèles respectivement, avec une dégradation de la performance ente 3% et 5%. Dans [116], les auteurs analysent la manière optimale de co-programmer les tâches sur un processeur multi-cœurs en tenant compte des critères de sélection des fréquences et de gestion des conflits. L'ordonnancement des tâches permet d'éviter les conflits d'accès aux ressources partagées et facilite la sélection optimale des fréquences pour chaque tâche. Sur la base de cette analyse, les auteurs proposent une stratégie d'ordonnancement qui trie les tâches dans la file d'attente de chaque cœur en fonction de leur taux d'utilisation de la mémoire afin de séparer les tâches gourmandes en mémoire avec les tâches gourmandes en traitement. Une heuristique, qui ajuste le niveau de la fréquence lorsque seules des tâches gourmandes en mémoire sont planifiées, est mise en œuvre afin de jouer sur la consommation globale du système. De même les auteurs dans [76] utilisent la technique de la DVFS, en développant un ordonnanceur de tâches qui en fonction d'un modèle de coût énergétique réorganise l'exécution des tâches en ajustant dynamiquement la fréquence du processeur. Le modèle utilisé pour la prédiction de la consommation énergétique utilise la régression linéaire puis la technique des forêts aléatoires pour déterminer les valeurs des constantes énergétiques. Dans [5], les auteurs proposent une approche de l'ajustement dynamique de la tension pour le contrôle de la consommation énergétique dans les architectures multiprocesseurs. Il utilise des politiques de de rétroaction linéaire et non linéaire pour adapter les fréquences en utilisant le taux d'utilisation de la mémoire tampon pour les flux entrants. Dans l'approche linéaire, la même stratégie de rétroaction est itérée en séquence à chaque étape en utilisant des fonctions linéaires. Les techniques non linéaires utilisent des fonctions de rétroaction non linéaires pour contrôler le fonctionnement du système.

2. Dynamic Random Access Memory
3. `https://lucene.apache.org/solr/`

L'application de la technique DVFS sur un processeur multi-cœur est une tâche complexe. Elle est souvent simplifiée en forçant chaque nœud d'un package à fonctionner à la même fréquence et à la même tension [100]. Appliquer la même tension pour tous les cœurs (DVFS global) conduit à une forte dégradation de l'efficacité énergétique. Pour pallier à cette limitation, des architectures avec DVFS globales et DVFS par-cœur ont été proposées avec plusieurs VFI (Voltage Frequency Islands). Sur ces plates-formes, les cœurs d'un même package partagent la même tension et la même fréquence, mais différents packages peuvent être exécutés à différentes tensions et fréquences [133]. Dans [158], les auteurs proposent un framework qui utilise la technique du DVFS pour répondre aux contraintes d'équité de fonctionnement des unités de traitement en fournissant un ordonnanceur éco énergétique. L'implémentation du framework sur un processeur multi-cœurs hétérogène conduirait à des résultats améliorant considérablement l'efficacité énergétique et l'équité par rapport à l'ordonnanceur standard exploité au niveau du système Linux.

Les auteurs dans [6] décrivent et analysent deux méthodes pour améliorer l'efficacité énergétique de la mémoire centrale (DRAM [4]) en exploitant les modes d'ajustement dynamique de l'énergie au niveau des barrettes de la DRAM. Les deux méthodes visent à prolonger l'inactivité des ranks (rangée des barrettes) en réduisant leurs demandes d'accès. La première méthode nommée Rank-Aware REplacement(RARE) suggère une politique de remplacement du dernier niveau de cache (LLC pour Last Level Cache), qui empêche le remplacement des blocs de mappage aux ranks prioritaires, réduisant ainsi les accès globaux à ces ranks. La seconde méthode Rank-Aware Write Buffer (RAWB) propose que les tampons retiennent les demandes allant vers certaines barrettes puis les envoient en mode groupé (bach) lorsque la DRAM est en mode pleine utilisation, prolongeant ainsi les temps d'inactivité des ranks. [64] propose CROW pour Copy-Row DRAM, un algorithme qui partitionne chaque zone mémoire de la RAM en deux régions (lignes régulières et lignes de copie) puis active un contrôle indépendant sur les lignes de chaque région. Ils utilisent CROW pour concevoir deux nouveaux mécanismes, CROW-cache et CROW-ref, qui améliorent les performances de la DRAM et l'efficacité énergétique. CROW-cache utilise une ligne de copie pour dupliquer une ligne régulière et active simultanément une ligne régulière avec sa ligne de copie dupliquée pour réduire la latence d'activation de la DRAM. CROW-ref remappe les lignes régulières faibles en rétention vers les lignes de copie forte, réduisant ainsi le taux d'actualisation de la DRAM. La flexibilité de CROW permet d'utiliser simultanément CROW-cache et CROW-ref pour une rapidité de $20,0\%$ et une optimisation énergétique de $22,3\%$ par rapport à la DRAM conventionnelle.

3.3.1.2 Désactivation Dynamique des Composants

Les architectures récentes offrent généralement la capacité d'opérer dans différents modes de fonctionnement permettant ainsi de réduire la consommation énergétique du système. Ces modes de fonctionnement sont gérés automatiquement par le système d'exploitation via l'interface avancée de configuration et de gestion de l'énergie (advanced configuration and power interface - ACPI) qui définit les états de performance du processeur(états P) et les états d'inactivités(états C) [67].

La désactivation des composants peut être vue comme un cas spécial de DVFS. Certains travaux explorent ces alternatives pour faire des économies d'énergie comme dans [101] [135]. Les techniques et algorithmes d'optimisation utilisés dans [101] [135] mettent l'accent surtout sur les moyens d'éteindre entièrement la mémoire ou de mettre une partie de ces composants en mode inactif, pendant l'exécution des processus. Dans [196] les auteurs exploitent une approche d'efficacité énergétique basée sur le mécanisme de pré-récupération/mise en cache. Ce mécanisme regroupe les données fréquemment accédées (appelées données populaires) sur des nœuds chauds en conservant les données impopulaires sur des nœuds froids. Ainsi, le SSD offre des économies d'énergie en ajustant par la technique DVFS la consommation énergétique des nœuds froids stockant les données impopulaires. Dans [50], les auteurs ont proposé une méthode de rafraîchissement adaptative pour contrôler la mémoire dans l'objectif de réduire la consommation d'énergie. Dans [22], l'effet de la puissance et des performances a été analysé en réduisant la capacité de la mémoire cache. La réduction de sa taille lors de l'exécution de certaines applications a permis une optimisation significative de l'énergie d'un ordre de $3,17\%$ et $24,16\%$ respectivement en énergie dynamique et statique, tout en dégradant les performances et augmentant les défauts de caches (Cache misses). D'autres technologies émergentes telles que la mémoire à changement de phase (PCM) [195], le couple de transfert de spin [181] et la memristance [125] ont également été proposées comme alternatives pour améliorer l'efficacité énergétique par rapport à la mémoire principale traditionnelle. Les moyens de stockage économe disponibles aujourd'hui sont les disques SSD (Solid State Drives). Les disques durs SSD

4. Dynamic Random Access Memory

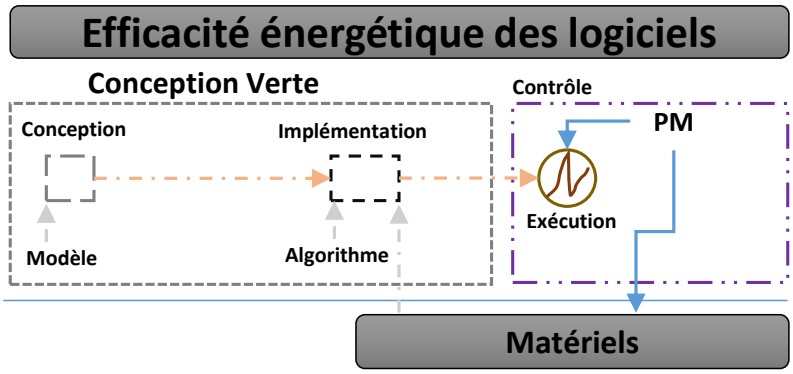


FIGURE 3.5 – Vue abstraite de l'efficacité énergétique des logiciels.PM *pour Power Management en français gestion de l'énergie*

ont des propriétés proportionnelles à l'énergie car l'énergie utilisée est proportionnelle aux opérations d'E/S par seconde [170].

3.3.1.3 Co-traitements

Dans les récentes générations des systèmes, l'amélioration de l'efficacité énergétique ne se limite plus à l'exploitation de l'ajustement dynamique de la fréquence, mais aussi en parallélisant les unités de traitement (architectures hybrides) pour permettre le co-processing et accroitre l'efficacité énergétique. Le calcul hétérogène en combinant les processeurs conventionnels(CPU) avec les architectures *non conventionnelles* telles que les GPUs, les FPGAs [5], les ASICs [6] et les DSPs [7] offre aujourd'hui une grande promesse pour améliorer les performances des applications et l'EE [119]. Dans [175], les auteurs comparent les performances et l'EE entre le CPU et le GPU pour plusieurs applications. Ils ont conclu que GPU offre un avantage significatif en termes de performances par rapport au processeur. Dans [41], les auteurs évaluent les performances et la consommation énergétique des applications sur les architectures $FPGA$, $GPUs$ et $Multi-coeurs$. Ils ont constaté que l'architecture $FPGA$ offre des performances beaucoup plus rapides dans la plupart des cas d'utilisations. Ils ont également conclu que les $FPGA$ offrent une meilleure EE dans presque toutes les situations.

L'approche matérielle explorée dans cette section pour réduire la consommation énergétique du système tire profit des techniques de l'ajustement dynamique de la tension (DVFS) et du co-traitement (hétérogénéité des unités de traitement). Il est très difficile d'améliorer la performance et l'EE grâce à cette approche sans alterner sur l'efficience de l'une ou de l'autre car elles sont fortement dépendantes. Les solutions matérielles ne constituent qu'une partie des approches pour augmenter l'efficacité énergétique, néanmoins celles-ci peuvent rester vaines lorsque les autres approches connexes (logicielles, environnementales) ne sont pas bien adaptées à l'EE.

3.3.2 Solutions logicielles

Les techniques de l'EE appliquées au niveau matériel peuvent être entachées si la couche logicielle n'est pas conçue (codée) avec des perspectives de l'efficience énergétique. En effet des efforts peuvent être faits pendant la phase de conception des logiciels pour intégrer la dimension énergétique dans le but de réduire sa consommation énergétique durant son exécution (figure 3.5). Les techniques d'EE lors de la conception et l'implémentation consistent à utiliser des modèles de conception qui intègrent la dimension énergétique. Utiliser les outils de contrôle permettant d'adapter dynamiquement les performances du système en fonction des demandes des utilisateurs lors de l'exécution du logiciel. La gestion de l'énergie au niveau logiciel peut se faire à deux niveaux soit au niveau système d'exploitation (SE) ou au niveau applicatif. Les techniques de gestion dynamiques de l'énergie au niveau logiciel utilisent l'interface avancée de configuration et de gestion de l'énergie (Advanced Configuration and Power Interface - APCI) [12]

5. Field-Programmable Gate Array, pour Réseau de Portes Programmables en français
6. Application-Specific Integrated Circuit pour Circuit Intégré propre à une Application en français
7. Digital Signal Processor pour Processeur de Signal Numérique

, proposée comme norme industrielle par Intel, Microsoft et Toshiba, pour contrôler la consommation énergétique des différents composants du système. Les récentes approches de gestion de l'énergie au niveau logiciel se concentrent principalement sur la gestion de l'utilisation efficace et optimale des ressources informatiques dans le but de réduire le nombre de composants requis pour faire tourner les SSDs tout en maximisant la qualité du service. Cette section décrit certaines approches couramment utilisées pour la gestion de ressources au sein des SSDs pour améliorer l'EE au niveau logiciel.

3.3.2.1 Technique de virtualisation

La virtualisation est une technique largement connue et appliquée pour améliorer l'EE des systèmes informatiques. La technologie de virtualisation conduit à la réduction de la consommation énergétique en assemblant plusieurs machines physiques(MPs) sur un seul serveur par la création des machines virtuelles (MVs). De cette façon, de nombreuses MVs peuvent être exécutées indépendamment et les MPs non utilisées peuvent être désactivées ou mises en mode de veille prolongée [165].

Dans [123], le VirtualPower est proposé - un système de gestion de l'énergie dans les environnements virtualisés qui exploitent à la fois l'ajustement dynamique de l'énergie et les méthodes logicielles pour contrôler la consommation d'énergie des plates-formes sous-jacentes. La gestion de l'énergie prend en compte les politiques de gestion indépendantes des MVs installées et les coordonne pour atteindre les objectifs globaux souhaités. Dans [144], les auteurs proposent une heuristique d'allocation nommée EPOBF (pour Energy-aware and Performance-per-watt Oriented Bestfit) qui utilise la métrique de performance par watt pour choisir la meilleure configuration éco-énergétique parmi les machines disponibles pour mapper chaque MVs. Les simulations expérimentales montrent que 35% de la consommation totale d'énergie peut être réduite par rapport à l'heuristique d'allocation des MVs conventionnelles.

3.3.2.2 Technique de l'infonuagique (Cloud Computing)

Le cloud computing a totalement bouleversé l'approche conventionnelle qui consistait à disposer de ses propres ressources matérielles pour faire du calcul informatique. La flexibilité et la facilité d'acquérir et de libérer des ressources sur demande présente d'énormes avantages pour les administrateurs. Ce paradigme est fondé sur l'utilisation des technologies comme la virtualisation, le traitement distribué pour offrir des services de location de ressources aux utilisateurs. Le cloud computing peut être un moyen pour optimiser l'EE et par là aider à réduire les émissions de gaz à effet de serre [85]. Hormis le bénéfice de l'EE, le cloud computing possède d'autres avantages comme : (a) utilisation accrue des ressources, (b) indépendance de l'emplacement - Les MVs peuvent être déplacées vers un endroit où l'énergie est moins chère (ou plus verte), (c) adaptation de l'utilisation des ressources aux besoins de l'utilisateur [12].

Plusieurs recherches se sont également focalisées sur certaines actions potentielles pouvant boostées l'EE au sein des techniques basées sur le cloud computing. On peut distinguer principalement deux approches basées: (1) sur l'optimisation du fonctionnement des machines destinées à héberger les MVs, ou (2) sur la consolidation des MVs dans un même cluster. Alors que la première approche vise à réduire la consommation d'énergie de chaque serveur en utilisant la technique du DVFS, la seconde tente de migrer les MVs sur un nombre minimal de machines [19]. La consolidation est une technique permettant de migrer les VMs entre les serveurs dans le but soit: (1) de libérer les serveurs pour la maintenance ou pour les éteindre, (2) de réorganiser dynamiquement les VMs en fonction de la manière dont elles utilisent les ressources. Pour effectuer efficacement les migrations de MV entres les clusters, il existe principalement deux défis : (1) Optimiser le placement des MVs et (2) Minimiser les délais de placement et la consommation énergétique [182]. Il existe une panoplie de méthodes ayant pour but d'optimiser le placement des MV et leur coût énergétique, parmi elles, nous référençons les travaux suivants :[45],[40],[88],[25].

Plus récemment, un nouveau paradigme connu sous le nom de l'informatique de périphérie (edge computing) a émergé pour augmenter l'efficience dans le traitement. C'est est une approche d'optimisation employée dans le cloud computing qui consiste à traiter les données à la périphérie du réseau, près de la source des données. De cette manière la latence de communication est réduite entre les nœuds de traitement [79]. Ce nouveau type de calcul est également sensible à l'énergie. Une étude récente sur l'informatique de périphérie sensible à l'énergie est donnée dans [79], dans laquelle les auteurs passent en revue les solutions d'EE partant des solutions matérielles vers les solutions logicielles.

3.3.3 Gestion de la température

L'environnement de déploiement du système est essentiel au bon fonctionnement de ces composants. La chaleur dégagée par les systèmes conduit à l'augmentation de la température, qui doit être réajustée pour éviter les défaillances matérielles. Plus la quantité de chaleur libérée est élevée, plus est l'énergie consommée par les systèmes de refroidissement. Dans l'objectif de réduire la consommation d'énergie d'un SSD, non seulement les unités de calcul sont importantes, mais aussi l'infrastructure de déploiement, y compris les systèmes de refroidissement, la position géographique, etc., parce que la température ambiante autour du système affecte sa consommation globale [150]. La température environnementale recommandée par ASHRAE [8] pour les SSDs de classe A1 à A2 est comprise entre 18° et 27° Celsius (64° à 81° Farenheit). De nos jours, les techniques de refroidissement des SSDs sont devenues un domaine plus attractif soulevant plusieurs défis. Les chercheurs et les industriels ont développés de nouvelles technologies de refroidissement et ont montré leur efficacité en terme d'économie d'énergie [61]. Le survey dans [37] classe l'efficacité énergétique dans les systèmes de refroidissement en trois catégories (1) refroidissement par air, (2) refroidissement par liquide et (3) refroidissement en deux phases. Les systèmes traditionnels de refroidissement par air ne s'adaptent pas aux objectifs globaux qu'est l'efficacité énergétique, ils sont de plus en plus remplacés par des systèmes de refroidissement à base d'eau. Plusieurs entreprises ont commencé à utiliser ces alternatives de refroidissement par liquide.

3.3.4 Standards et Conduite

En réponse à la surconsommation d'énergie, les gouvernements et quelques entreprises ont décidé d'imposer des règles et des standards de conduite dans l'objectif de réduire les émissions de gaz à effet de serre et atténuer les impacts économique et sécuritaire de l'approvisionnement énergétique. La Commission européenne (CE) a créé le Code de Conduite pour améliorer l'efficacité énergétique en 2008. Comme la CE, le ministère américain de l'énergie a le programme ENERGY STAR, qui propose des lignes directrices pour promouvoir les économies d'énergie aux États-Unis.

Outre le Code de conduite pour surveiller l'efficacité énergétique, plusieurs normes d'efficacité énergétique ont été imposées pour contrôler la fabrication et l'utilisation des équipements. En voici quelques-unes:

— *Directive sur la gestion des déchets des équipements électriques et électroniques:* La CE a introduit la directive WEEE [9] en 2002 pour contrôler les impacts environnementaux des équipements électriques et électroniques non désirés à la fin de leur cycle de vie.

— *Directive de l'Union européenne sur l'efficacité énergétique:* Cette directive introduit des mesures juridiquement contraignantes pour encourager les efforts visant à utiliser l'énergie de manière plus efficace dans toutes les étapes de la production à l'approvisionnement.

— *Normes d'Efficacité Minimales Obligatoires(NEMO):* Élaborées par l'Institut National Chinois des Normes (CNIS), la norme d'efficacité énergétique obligatoire a pour but de réguler la conception et le fonctionnement de certains équipements tels que les appareils ménagers, les appareils d'éclairage et les équipements de chauffage et de refroidissement résidentiels.

Dans cette section, nous avons présenté les travaux de recherche qui traitent la gestion de l'efficacité énergétique dans les SSDs et son environnement d'exploitation. La taxonomie utilisée pour classer les travaux de recherches est présentée sur la figure 3.6.

8. American Society of Heating, Refrigeration and Air Conditioning
9. `https://ec.europa.eu/environment/waste/weee/index_en.htm`

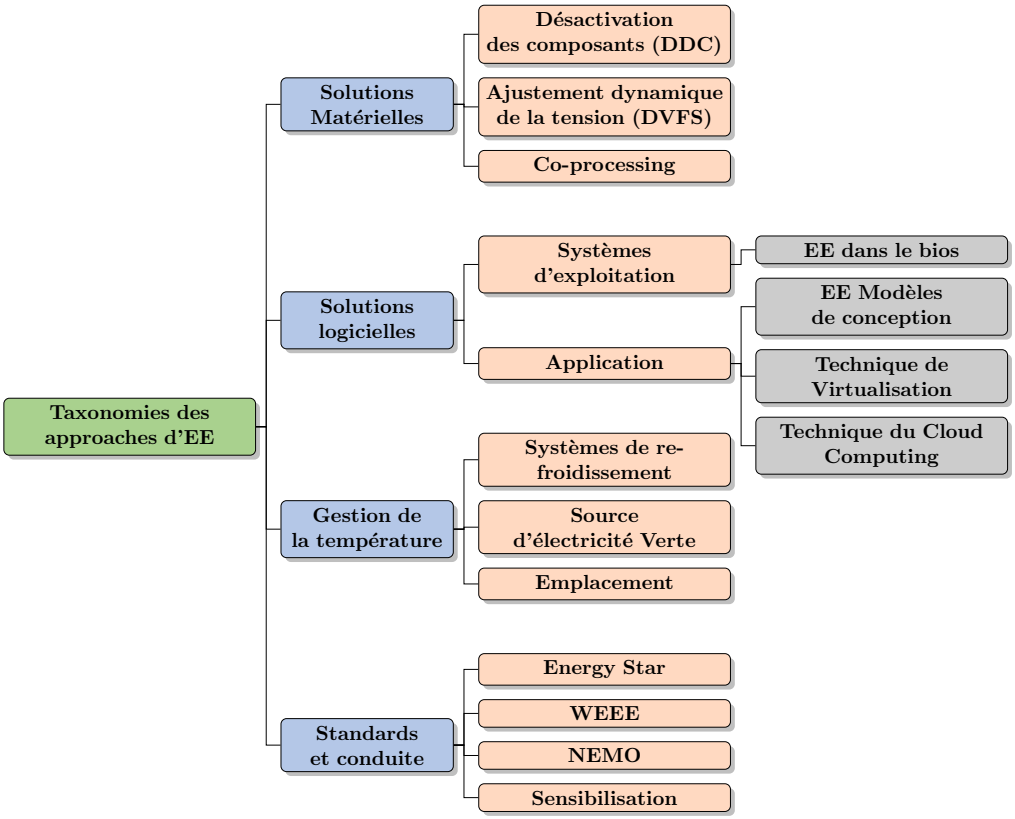


FIGURE 3.6 – Taxonomies des approches d'EE dans les systèmes informatiques.

3.4 Approches d'EE dans les STRs

L'objectif primaire des SGBDs traditionnelles était orienté uniquement vers la performance, c'est-à-dire comment exécuter aussi rapidement que possible une requête sans tenir compte de la dimension énergétique. Avec l'avènement des besoins écologiques, cet objectif migre de plus en plus vers la satisfaction des deux contraintes (performance et énergétique). La gestion de l'énergie dans les SSDs est devenue un sujet de recherche urgent ces dernières années pour la communauté des BDs. Les techniques de gestion de l'énergie dans les BDs sont axées sur deux groupes: matériels et les logiciels. Dans les sections suivantes, nous détaillons les principaux travaux de chaque approche en nous focalisant principalement sur les travaux traitant l'efficacité énergétique dans les systèmes de traitement des requêtes (STRs).

3.4.1 Approches Orientées Matérielles

Cette approche vise à exploiter les capacités offertes par les nouvelles architectures (comme l'ajustement dynamique de la tension, la désactivation des composants pendant les périodes d'inactivité, etc.) lors de l'exécution de la requête pour réduire la consommation énergétique du système. Nous divisons les travaux de cette approche en trois catégories sur la base des composants qu'ils exercent : (1) les unités de traitement, le Co-traitement, les unités de stockage. Nous décrivons pour chaque catégorie les principaux travaux dans les sections suivantes. La Figure 3.7 représente une classification des approches d'EE orientées matérielles dans les STRs.

3.4.1.1 Les unités de traitement

Partant du fait que le processeur, est la composante la plus gourmande en énergie, de multiples études ont proposé des approches basées sur la technique du DVFS pour réguler sa consommation énergétique. Les études de [97],[178],[189] ajustent le niveau de la fréquence du système en fonction du débit et des caractéristiques de la charge de travail fournie. Cette approche réduit la consommation du système en acceptant une diminution de la performance d'exécution des requêtes. Dans le but d'améliorer l'EE du traitement des requêtes transactionnelles dans un environnement multi-cœurs, les auteurs dans [65] ont

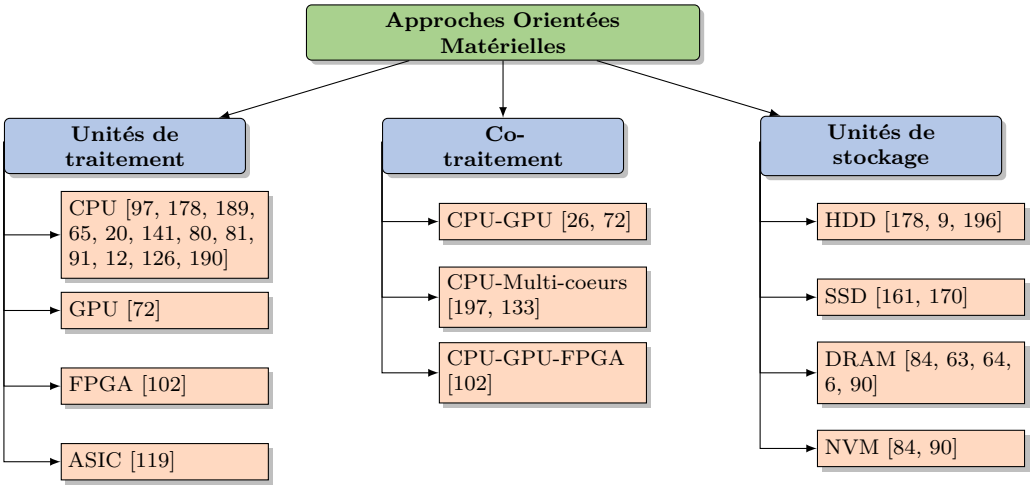


FIGURE 3.7 – Classification des approches d'EE orientées matérielles.

développé une application qui ajuste dynamiquement la fréquence de fonctionnement des cœurs sur la base du délai de réponse acceptable par l'utilisateur. L'application utilise une fonction de compromis entre l'énergie consommée et le délai fixé comme un accord de service (SLA). Les études expérimentales conduisent à une réduction de 7,6% de la consommation d'énergie en utilisant les requêtes du benchmark TPC-C. Des études expérimentales ont été conduites dans [126], pour évaluer l'impact de la réduction de la fréquence d'horloge du processeur sur l'efficacité énergétique de l'implémentation algorithmique de certains opérateurs relationnels tels que le balayage séquentiel, les agrégations et la jointure. Partant de l'analyse des résultats obtenus, les auteurs ont pu conclure que la réduction de la fréquence améliore de façon significative l'efficacité énergétique du processeur pour les requêtes gourmandes en mémoire. Dans le contexte de la recherche des données sur le web, les auteurs dans [20] ont développé un outil de planification prédictive en ligne des économies d'énergie nommée PESOS (Predictive Energy Saving Online Scheduling) pour sélectionner la fréquence du CPU la plus appropriée pour traiter une requête. Cet outil vise à traiter les requêtes dans un délai imparti et de tirer profit des caractéristiques de planification de l'ordonnanceur pour réduire la consommation d'énergie d'un nœud de traitement de la requête. PESOS prend ces décisions d'ajustement à partir des prédicteurs qui estiment le volume de traitement et le temps de traitement d'une requête avant son exécution. Dans la même approche les auteurs dans [141] proposent APESOS (Advanced Predictive Energy Saving Online Scheduling Algorithm), une extension des travaux dans [20] dans le but de sélectionner la fréquence CPU la plus adaptée pour traiter une requête. Dans [80], les auteurs ont développé une approche de la théorie du contrôle pour traiter une seule transaction en temps réel exécutée simultanément avec d'autres transactions interférentes mais pas en temps réel dans un système de base de données intégré. Les économies d'énergie sont faites en utilisant la technique du DVFS et la suppression des données du capteur dans la boucle de rétroaction. Les mêmes auteurs dans [81], ont combiné les avantages de la technique du DVFS et l'adaptation dynamique de la cohérence temporelle des données (QoD) pour atteindre une efficacité énergétique élevée et la rapidité lors du traitement des transactions. Les auteurs dans [91] ont développé une technique nommée *POLARIS*, qui ajuste dynamiquement la fréquence du processeur en fonction de l'ensemble des transactions en cours d'exécution ou en attente d'exécution. Après avoir estimé les temps d'exécution des transactions à différents niveaux de fréquence du processeur, POLARIS choisit la fréquence la mieux adaptée dans un panel de configuration fixe dans lequel le processeur peut fonctionner sans trop atténuer sur les délais d'exécution impartis. Dans [190], les auteurs ont proposé de réduire la consommation énergétique du CPU lors de l'exécution des requêtes en partant d'un coût énergétique défini sur les micro-opérations du CPU (cache niveau L1, Cache niveau L2, Cache niveau L3, mémoire centrale) et de l'ajustement dynamique de la fréquence (DVFS). Ils développent une méthodologie pour évaluer avec précision le coût énergétique des différentes micro-opérations du CPU en concevant des micro-benchmarks qui utilisent des charges de travail incluant uniquement des requêtes de lecture. Les expérimentations menées sur trois SSDs (PostgreSQL, SQLite et MySQL) révèlent que le chargement/stockage du cache L1 constitue un goulot d'étranglement énergétique. Partant de cette conclusion, ils proposent une architecture CPU L1 personnalisée et économe en énergie en sélectionnant le niveau DVFS le mieux adapté. Les résultats expérimentaux montrent que 60% d'économies énergétiques peuvent être réalisées.

Malgré que les techniques mentionnées ci-dessus permettent d'atteindre une efficacité énergétique, il faut néanmoins comprendre que si les transitions entre les états de fonctionnement ne sont pas correctement ajustées, ces techniques peuvent souffrir d'un sur-coût lié aux transitions qui entraînent généralement une consommation d'énergie supplémentaire et des retards causés par la ré-initialisation des composants [12]. Donc par conséquent si les transitions sont fréquentes, elles peuvent atténuer l'efficacité énergétique.

3.4.1.2 Technique de Co-traitement

Le paradigme de co-traitement des requêtes consiste à exploiter les progrès matériels ainsi qu'une conception optimisée des algorithmes pour améliorer les performances de traitement des requêtes et l'EE. Ce paradigme profite des avantages de chacun des dispositifs de traitement incorporés, il est de plus en plus utilisé par les SGBDs surtout les BDs in-memory (en mémoire centrale) pour traiter les opérations de jointure [26]. Les résultats de comparaison entre un jeu d'instruction CPU-GPU discret et un jeu CPU-GPU couplé montrent que la consommation moyenne d'énergie de l'approche discrète est entre 36% et 44% supérieures à ceux de l'architecture couplée [72]. Dans la même tendance, les auteurs dans [26] ont évalué et analysé expérimentalement les performances et la consommation d'énergie du co-traitement des requêtes sur les architectures embarquées CPU-GPU. Pour faciliter l'évaluation et l'optimisation de l'énergie des SGBDs sur un système multi-cœur, un ensemble d'outils appelé EDOM (Energy Efficiency of Database Operations on Multicore Servers) a été proposée en [197]. Les deux principaux composants d'EDOM sont les outils du benchmark et un gestionnaire multi-cœur pour améliorer l'efficacité énergétique des SGBDs. Les outils du benchmarking comprennent trois modules: un générateur de charge de travail, un pilote d'essai, et un gestionnaire d'énergie. Dans [102] un mécanisme de couplage logiciel et matériel est étudié pour minimiser le coût de l'énergie tout en satisfaisant la contrainte de délai d'exécution au sein des SSDs. Sur le plan matériel, une architecture matérielle hétérogène $CPU/GPU/FPGA$ compatible à la technique de DVFS est construite. En adaptant la tension de fonctionnement des processeurs par le biais du régulateur DVFS ou en attribuant les tâches aux nouveaux processeurs, les paramètres dépendants du matériel dans le modèle énergétique peuvent être modifiés pour améliorer l'efficacité énergétique. Dans l'aspect logiciel, ils utilisent un algorithme Q-learning pour rechercher la solution optimale au problème d'ordonnancement. Les résultats obtenus après une série d'expérimentation révèlent de meilleures performances et des optimisations énergétiques.

3.4.1.3 Les unités de stockage

Les approches utilisées au sein des dispositifs de stockage tentent de trouver un bon niveau de consolidation de la charge, d'améliorer les techniques de mise en cache et de pré-chargement et d'optimiser les modes d'énergie dans les réseaux de disques pour minimiser la dissipation énergétique[178]. La technique de placement dynamique des données dans des fragments selon leur fréquence d'accès par les requêtes fut proposée par [178]. Les économies d'énergie sont faites par cette technique en adaptant dynamiquement le niveau de la fréquence des nœuds de fragments les moins accédés. De manière similaire, les auteurs dans [9] proposent un modèle d'ajustement dynamique de la fréquence qui prend des décisions en temps réel sur la commutation des disques en modes de faible puissance. Le modèle est intégré au fonctionnement d'un SGBD et est utilisé pour obtenir des compromis optimaux entre la consommation d'énergie et le temps de réponse de la requête. Les expérimentations révèlent jusqu'à 60% d'économies d'énergie. Dans [161], la performance et la consommation d'énergie des supports de stockages sont analysées et comparées en utilisant des bancs d'essai à forte intensité d'entrées/sorties(E/S). Ils concluent que les supports de stockage de type SSD [10] offrent des meilleures performances en temps d'exécution des requêtes qu'en efficacité énergétique.

L'avènement des BDs orientées colonnes et des BDs résidentes en mémoire centrale font que la mémoire dynamique devient de plus en plus un grand consommateur d'énergie. De nombreux travaux de recherches ont abordé l'efficacité énergétique de la mémoire centrale lors du traitement des requêtes. Les travaux dans [84] présentent une caractérisation empirique de la consommation d'énergie de la mémoire centrale dans les SSDs, tant pour les requêtes analytiques que transactionnelles. Les résultats rapportés indiquent que l'optimisation de l'énergie de la mémoire ne sera efficace que si elle peut réduire la puissance statique grâce à une utilisation plus agressive des états d'inactivité de la mémoire. Les travaux dans [63] proposent une architecture de mémoire hybride composée à la fois de la mémoire dynamique à accès aléatoire(DRAM)

10. Solid State Drive

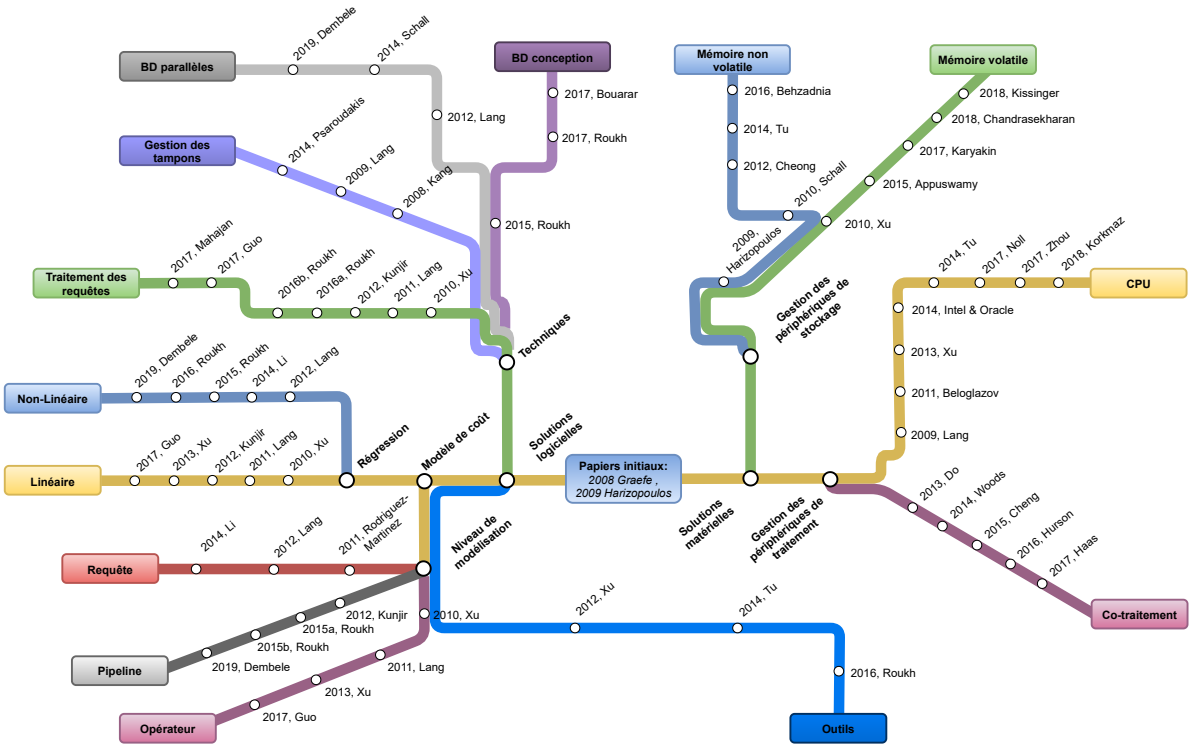


FIGURE 3.8 – Vue d'ensemble de l'intégration de l'efficacité énergétique dans les BDs

et de la mémoire non Volatile (NVM) pour réduire la consommation d'énergie de la BD. Pour cela, ils utilisent une politique de gestion des données au niveau applicatif qui décide de placer des données sur la mémoire DRAM ou NVM en fonction de l'état du système. Plus récemment dans [90] les auteurs ont développé un outil nommé ECL (Energy – Control - Loop), une approche intégrée dans le fonctionnement du SGBD pour le contrôle adaptatif de l'énergie consommée sur les systèmes en mémoire. ECL vise à optimiser activement l'efficacité énergétique et la performance du SGBD tout en obéissant aux délais imparti pour le traitement des requêtes. L'outil ECL repose sur des modèles de coût énergétique adaptatifs définit en fonction de la charge de travail au moment de l'exécution pour réguler dynamiquement la consommation du système.

3.4.2 Approches Orientées Logicielles

En plus des solutions de gestion de l'énergie évoquées dans les sections 3.3.1, 3.3.2,3.4.1, il est également possible de gérer la consommation énergétique à un niveau plus spécifique interne au SGBD comme recommandé dans les deux papiers phares [54] [62] initiant le début de l'intégration de l'énergie dans la conception et l'exploitation des BDs (figure 3.8). Ces recommandations incitent de revoir les techniques d'optimisation des requêtes, les algorithmes d'ordonnancement, la conception physique et les techniques de mise à jour des BDs en tenant compte de l'énergie. Les travaux de cette approche se caractérisent principalement par la (1) définition d'un modèle de coût énergétique, capable d'estimer la consommation énergétique d'une charge de requête et (2) la proposition d'une approche d'incorporation du modèle de coût dans la phase d'optimisation. La Figure 3.9 représente une classification des approches d'EE orientées logicielles dans les STRs.

3.4.2.1 Traitement éco-énergetique

Dans [184][185][186][187][188], les auteurs proposent un modèle de coût énergétique en partant du modèle de coût utilisé dans le système PostgreSQL. Le modèle est défini pour une seule requête qui s'exécute d'une manière isolée sans parallélisme (inter ou intra requête). Pour chaque opération de base, ils définissent un modèle spécifique qui estime sa puissance en fonction du nombre de tuples traités par le processeur et le nombre de page lu/écrit du disque. Les valeurs paramétriques du modèle sont calculées

en utilisant la technique de la régression linéaire simple multivariée. La formule générale pour calculer l'énergie d'un opérateur x est:

$$\hat{E}_x = W_{cpu} \times N_{tuples}^m \times N_{tuples} + \bar{W}_{E/S} \times N_{pages} \tag{3.3}$$

Où W_{cpu} and $\bar{W}_{E/S}$ sont les coefficients énergétiques d'un tuple traité par le CPU (N_{tuples}), et d'une page lue/écrite en mémoire (N_{pages}) respectivement, et m est un coefficient du modèle pour la consommation d'énergie du CPU.

Dans [103], les auteurs ont proposé un modèle énergétique pour estimer la consommation d'énergie d'une requête en utilisant le produit du nombre de lignes indexées par le CPU et le nombre de colonnes récupérés. Ils construisent un modèle de puissance pour chaque opération de base puis estime le coût du plan en fonction du coût des opérations de base. Le tableau 3.1 indique les opérations de base qu'ils mesurent avec la fonction énergétique.

Tableau 3.1 – Fonction énergétique pour les opérations de base

Opérateurs	*Modèle de coût énergétique*
Seq Scan	$F_1(cT) + f_1(N) + C$
Index Scan	$F_2(cT) + f_2(N) + C$
Bitmap Scan	$n(F_2(cT) + f_2(N)) + C$
Sort	$F_3(cT) + f_3(N) + C$

Où T représente le nombre de lignes à récupérer; c représente le nombre de colonnes à récupérer; N représente le nombre de page lu/écrit du disque; la fonction F_i représente la consommation d'énergie du CPU; la fonction f_i représente la consommation d'énergie du disque; C (une constante) représente la consommation d'énergie lorsque le système est inactif.

En extrayant les caractéristiques communes des requêtes (comme le facteur de sélectivité et la cardinalité), les auteurs dans [151] propose de modéliser l'énergie et la puissance pic des opérations d'une requête dans un environnement cluster. Ils utilisent la technique de régression linéaire multiple pour calculer les paramètres du modèle de prédiction de la consommation énergétique des requêtes de sélection simples sur les relations individuelles. Par exemple, en considérant une requête de sélection Q avec une table R, le modèle proposé pour estimer la consommation énergétique de Q est:

$$\begin{aligned} E_Q = &\beta_0 + \beta_1\ S + \beta_2\ SF_p(R)\ S + \beta_3\ |R| \\ &SF_p(R) + \beta_4\ |R| < R > SF_p(R) \end{aligned} \tag{3.4}$$

Où les β_i sont les coefficients de régression et $|R|$, $< R >$, $SF_p(R)$ et S sont respectivement: la cardinalité de la relation R, le nombre de colonnes de R, la sélectivité du prédicat p dans R et le nombre de serveurs utilisés.

Dans [93], les auteurs proposent de modéliser la puissance pic d'une requête segmentée en un ensemble de pipelines délimités par des opérateurs bloquants. Pour chaque pipeline, une fonction mathématique est définie, qui prend en entrée les taux et les tailles des données circulant entre les opérations du pipeline et donne en sortie une estimation de l'énergie pic consommée. Pour calculer les paramètres du modèle, ils utilisent la technique de la régression multiple affine par morceaux. Pour faire des économies d'énergie sur le plan d'exécution, les sources de consommation pic d'énergie pour une requête isolée sont remplacées dans la mesure du possible par des pipelines qui consomment moins d'énergie. En exemple, pour un nœud B (balayage séquentiel), le débit est calculé comme:

$$Rate_D = \frac{EntreB}{TempsDisque_B} \tag{3.5}$$

Où $EntreB$ indique la taille des données extraites du disque par B et $TempsDisque$ est le temps de transfert du disque.

En utilisant les informations extraites du plan d'exécution d'une requête, les auteurs dans [52][57] développent un modèle énergétique qui permet d'estimer la consommation d'énergie de chaque opérateur.

Pour le coût de chaque opérateur, en plus de considérer le coût du CPU et du disque, ils prennent en compte les caractéristiques de la plateforme d'exécution et l'état du SGBD en développant un module spécifique qui permet d'ajuster automatiquement les variables utilisées dans les modèles. Les modèles de coûts (C_t) proposés pour estimer le coût énergétique des opérateurs de balayage séquentiel et Bitmap Heap sont donnés dans le tableau 3.2:

Tableau 3.2 – Coût énergétique pour les deux opérateurs

Opérateurs	**Modèle de coût énergétique**
Seq Scan	$n_{seq} * f_{seq} + n_{tup} * f_{tup}$
Bitmap Heap Scan	$(n_{idx} * f_{idx} + n_{seq} * f_{seq}) + (n_{tup} * f_{tup} + n_{seq} * f_{seq})$

Où f_{seq}, f_{tup}, f_{idx} représentent respectivement le coût de récupération d'une page, le coût CPU pour traiter un tuple, le coût CPU pour traiter un tuple indexé et n est le nombre d'enregistrement en sortie.

Le travail de [96] propose un modèle pour le traitement éco-énergétique des requêtes. Ils étendent les plans d'exécution produits par l'optimiseur avec une prédiction de la consommation d'énergie pour certains opérateurs de la BD comme la sélection, la projection et la jointure en utilisant la technique de régression linéaire. le modèle proposé pour estimer la puissance moyenne du système pendant l'exécution d'une requête isolée est définie sur trois paramètres : (1) le temps T, (2) caractéristique de la requête Q : nombre d'instructions de processeur (I_Q), lecture de page disque (R_Q), écriture de page disque (W_Q) et nombre d'accès au mémoire (M_Q) et (3) caractéristique du système : CPU (C_{cpu}), lecture disque (C_R), écriture disque (C_W), accès mémoire (C_{mm}), reste du système (Cautres). Le modèle est défini comme suit:

$$P_{moy} = C_{cpu} \times \frac{I_Q}{T} + C_R \times \frac{R_Q}{T} + C_W \times \frac{W_Q}{T} + C_{mm} \times \frac{M_Q}{T} + C_{autres} \tag{3.6}$$

Dans [154][152], les auteurs ont proposé un modèle de coût pour prédire la consommation d'énergie des requêtes dans une BD centralisée. Le modèle proposé repose sur la segmentation du plan de la requête en un ensemble de pipelines. Le modèle proposé est construit en utilisant le coût du CPU et du coût d'E/S nécessaire pour évaluer les opérations appartenant au pipeline en utilisant la technique de la régression non linéaire (régression polynomiale). Plus formellement, soit p le nombre de pipelines de la requête $Q : PL_1; PL_2; ...; PL_p$, la Puissance de la requête (Q) est donnée par l'équation suivante:

$$Puissance(Q) = \frac{\sum_{j=1}^{p} Puissance(PL_j) * Temps(PL_j)}{Temps(Q)} \tag{3.7}$$

Où Temps(PL_j) et Temps(Q) représentent respectivement, le temps d'exécution du pipeline j et de la requête Q. Le coût d'un pipeline j doit être décomposé car il peut impliquer plusieurs opérations algébriques. Soit n_j le nombre de ces opérations ($OP_1; OP_2; ...; OP_n$). Le coût énergétique d'un pipeline j ($puissance(PL_j)$) est donné par l'équation suivante:

$$Puissance(PL_j) = \beta_{cpu} \times \sum_{k=1}^{n_j} CPU\_COST_k + \beta_{io} \times \sum_{k=1}^{n_j} ES\_COST_k \tag{3.8}$$

Où β_{cpu} et β_{io} sont les paramètres du modèle et le ES_COST est le nombre prévu d'entrées/sorties nécessaires pour exécuter un opérateur spécifique. Le CPU_COST est le nombre prévu d'instructions CPU que le SGBD a besoin pour exécuter un opérateur spécifique.

Une étude approfondie sur l'effet de la taille de la mémoire centrale et des trois structures de mémoires caches dans un SGBD ORACLE (Database Buffer Cache, Dictionary Cache et Library Cache) en utilisant différentes requêtes est donnée dans [58]. Sur la base de cette étude, les auteurs ont proposé un modèle de coût énergétique en prenant en considération le coût énergétique de l'utilisation de la mémoire centrale. Le modèle proposé s'écrit de la forme suivante:

$$\begin{aligned} E =& N_{cpu}(W_{cpu} \times T_{cpu}) + K_m(W_m \times T_m) \\ &+ K_s(W_s \times T_s) + K(W_k \times T_k) \end{aligned} \tag{3.9}$$

Où W_{cpu} est le coefficient de la puissance de chaque 10000 instructions traitées par le CPU. N_{cpu} est le nombre d'instructions traitées par le CPU, et il est compté par dix mille. T_{cpu} est le temps du processeur pour traiter toutes les dix mille instructions. K_s est le nombre d'opérations faisant une seule lecture disque, K_m est le nombre d'opérations faisant multiples lectures disque, et K est le nombre de blocs de données accessibles par mémoire. Le modèle proposé utilise une technique de régression linéaire simple pour obtenir les valeurs des coefficients du modèle.

Les modèles de coûts proposés pour la prédiction de l'énergie sont ensuite utilisés dans diverses techniques pour optimiser la consommation énergétique des BDs. Les auteurs dans [97] ont proposé le mécanisme QED (Improved Query Energy-efficiency by Introducing Explicit Delays), une alternative pour améliorer l'EE des requêtes en introduisant des retards explicites, en exploitant les caractéristiques communes dans la charge de la requête. Le retard explicite permet de grouper les requêtes dans une file d'attente en mémoire tampon puis de les exécuter en lot par des techniques d'optimisation multi-requêtes. Les travaux dans [96], [185], [153], [58], [143] proposent tous une approche de compromis entre l'énergie et la performance en incorporant leur modèle dans l'optimiseur du SGBD. Ils choisissent des plans de requêtes qui réduisent la consommation d'énergie et répondent aux objectifs de performance en utilisant une variable de pondération qui indique l'importance relative des fonctions de coût énergétique et de performance.

3.4.2.2 Conception physique éco-énergetique

La gestion efficiente de la conception physique peut considérablement améliorer l'efficacité énergétique en spécifiant de manière optimale les paramètres de configuration c'est-à-dire en indexant uniquement les attributs les plus pertinents, en appliquant les techniques de partitionnement adéquates, en identifiant les vues pertinentes à matérialiser, en utilisant les bons algorithmes de compression [54]. La compression des données est une approche largement appliquée dans le contexte des BDs en mémoire ou orientée colonne vu le bénéfice qu'elle apporte en comparant l'énergie supplémentaire consommée par le CPU pendant les étapes de compression et de décompression avec l'énergie économisée dans les sous-systèmes d'entrées/sorties en raison de la diminution du volume des données.

Dans [177], les auteurs après avoir définis un modèle énergétique caractérisant le coût des opérateurs de base d'une requête, expérimentent sur deux SGBDs (PostgreSQL et un SGBD commercial) en variant les paramètres de la requête et les algorithmes de compression. Après l'étude des effets de la compression des données sur l'énergie totale consommée par les systèmes en variant le taux de compression, il constate que la compression des données conduit à une meilleure performance et une meilleure efficacité énergétique. [155], les auteurs ont proposé un algorithme multi-objectif pour la sélection des vues matérialisées sous la contrainte de la dimension énergétique. Pour cela, ils ont utilisé un algorithme génétique pour générer un ensemble de solutions et un modèle de coût énergétique pour sélectionner les vues qui optimisent l'énergie et la performance. Les études expérimentales de cette approche conduisent à des économies d'énergie allant jusqu'à 80%. Les auteurs dans [82] proposent une technique de gestion de la mémoire tampon pour les BDs en temps réel sur une mémoire réinscriptible où les lectures et les écritures ont des coûts énergétiques différents. En ajustant dynamiquement la taille des pools de la mémoire tampon par la technique du contrôle de rétroaction, le système réduit l'utilisation d'énergie tout en respectant les exigences de temps de réponse.

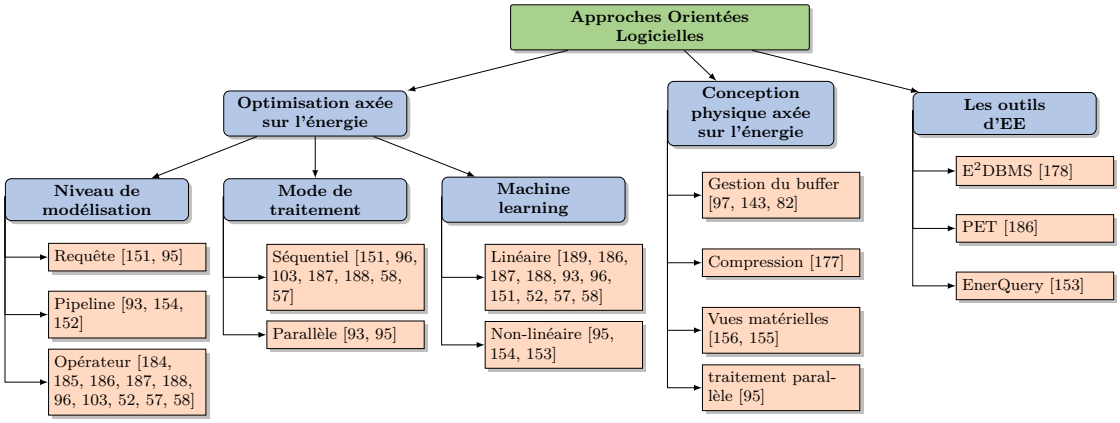


FIGURE 3.9 – Classification des approches d'EE orientées Logicielles.

3.5 Conclusion

Dans ce chapitre, nous avons souligné la nécessité de développer des techniques de gestion de l'énergie dans les SSDs. Nous avons examiné plusieurs techniques qui ont été proposées pour minimiser la consommation énergétique dans les SSDs d'une manière générale puis de manière plus spécifique dans les STRs en les classifiant en fonction de leurs caractéristiques (figures 3.9, et 3.7). Les solutions matérielles sont simples à comprendre mais très difficiles à appliquer à cause du fait qu'elles sont très spécifiques aux composants. Généralement ces solutions ne sont pas portables et nécessitent que les composants aient la capacité de nouvelles technologies telles que la variable DVFS. Pour contourner les difficultés dans les techniques matérielles, les solutions logicielles ont été proposées. Elles peuvent se révéler très coûteuses à cause de l'analyse des connaissances détaillées sur les éléments internes du système, tels que la charge de requêtes, les plans d'exécution, les statistiques dans le cas d'une application de BD, etc.

Cet état de l'art a permis de rappeler les techniques de réduction de la consommation dynamique de l'énergie dans les STRs, allant du niveau des composants matériels au niveau logiciel dans l'optimiseur du système de traitement des requêtes. Nous avons découpé l'approche logicielle en deux groupes: (1) les techniques utilisées lors du traitement de la requête et (2) les techniques incorporées dans la conception physique. L'élément fondamental dans chacune de ces techniques reste la définition d'un modèle de coût mathématique qui estime la consommation énergétique du système influencé par plusieurs facteurs (figure 2.16). Cependant avec des architectures toujours plus complexes, variables et évolutifs, les modèles de coût énergétique sont difficilement portables, et nécessitent de prendre en compte de nouveaux paramètres qui influencent la consommation énergétique du système lors du traitement des requêtes. De plus avec l'avènement des systèmes multi-cœurs, l'impression de plusieurs processeurs sur une seule carte électronique, nombreux sont les concepteurs de BD qui proposent un mode de traitement intra-parallèle pour une seule requête pour accroître les performances d'exécution.

Notre objectif est de proposer une méthodologie claire et précise pour la conception des modèles de coût énergétique lors du traitement des requêtes en tenant compte de tous les paramètres pertinents, puis d'évaluer de manière précise avec les modèles définis la consommation énergétique du système, avant de les exploiter dans des techniques de réduction de la consommation énergétique. La méthodologie, le développement et l'implémentation de nos modèles de coût énergétique sont détaillés dans le chapitre suivant. Les validations de nos modèles de coût en évaluant les erreurs d'estimation y seront également abordées.

Troisième partie

Contributions

CHAPITRE 4

Méthodologie de conception de nos McE et Formulation

En effet, c'est l'Eternel qui donne la sagesse, c'est de sa bouche que sortent la connaissance et l'intelligence.

— Proverbe, *2:6*

Sommaire

4.1 Introduction 73
4.2 Modèles de coût énergétique (McE) dans les STRs 73
4.2.1 Niveau de modélisation 73
4.2.2 Mode d'exécution des requêtes 73
4.3 Méthodologie de conception des McE 74
4.3.1 Audit des SGBDs 75
4.3.2 Identification des paramètres 75
4.3.3 Formalisation du modèle 76
4.3.4 Validation du modèle 76
4.3.5 Exploitation 76
4.4 Audit des systèmes SSDs étudiés 76
4.4.1 PostgreSQL 77
4.4.1.1 Architecture 77
4.4.1.2 Exécution de la requête 77
4.4.2 MonetDB 78
4.4.2.1 Architecture 79
4.4.2.2 Exécution de la requête 79
4.4.3 Hyrise 79
4.4.3.1 Architecture 80
4.4.3.2 Exécution de la requête 80
4.5 Analyse et études comparatives des SSDs étudiés 81
4.5.1 Environnement de l'étude 81
4.5.2 Évaluation des résultats 82
4.5.2.1 Temps et énergie consommée lors du chargement des données . 82
4.5.2.2 Temps d'exécution et énergie consommée lors du traitement des requêtes 82
4.5.3 Identification des paramètres du modèle énergétique 84
4.5.3.1 Stratégie d'exécution d'une requête 84
4.5.3.2 Effet du mode d'exécution 86
4.5.3.3 Effet de la taille du BD 87
4.6 Conception des McE 88

4.6.1 Hypothèses . . . 88
4.6.2 Formulation . . . 89
4.6.2.1 Modélisation des pipelines (PostgreSQL) . . . 89
4.6.2.2 Modélisation des opérateurs (MonetDB & Hyrise) . . . 91
4.6.3 Machine learning . . . 92
4.6.3.1 Les techniques régressives . . . 92
4.6.3.2 Les techniques cognitives . . . 94
4.6.4 Valeurs des paramètres . . . 97
4.6.4.1 Configuration matérielle . . . 97
4.6.4.2 Données d'apprentissage . . . 97
4.6.4.3 Application avec le logiciel R . . . 98
4.7 Validation des modèles . . . 106
4.7.1 Méthode d'évaluation . . . 106
4.7.2 Jeux de données . . . 106
4.7.3 Erreurs d'estimation . . . 107
4.7.3.1 Évaluation sur PostgreSQL . . . 107
4.7.3.2 Évaluation sur MonetDB . . . 109
4.7.3.3 Évaluation sur Hyrise . . . 110
4.7.4 Discussion . . . 112
4.8 Conclusion . . . 116

Dans ce chapitre, nous abordons le problème de la modélisation des coûts énergétiques des opérateurs constituant le plan d'exécution d'une requête donnée. Nous commençons par donner une démarche de construction générique applicable à tous les SGBDs, à savoir (1) l'audit des systèmes conduisant à l'identification des paramètres sensibles à l'énergie, à la détermination du niveau de modélisation adéquat; (2) la formulation du modèle de coût énergétique en appliquant les techniques d'apprentissage automatique estimer les valeurs des constantes énergétiques; (3) la validation du modèle dérivé en évaluant les erreurs d'estimation pour prouver l'efficacité de notre approche.

Partant de cette méthodologie, nous proposons quatre modèles de coût: deux (2) modèles pour les requêtes exécutées en mode parallèle sur le système PostgreSQL, un modèle énergétique pour les systèmes orientés colonnes (MonetDB) et un modèle pour les systèmes en mémoire (Hyrise). Cette étude comporte plusieurs résultats expérimentaux commençant par une étude comparative de la consommation énergétique des SSDs étudiés à la validation de nos modèles de coût proposés contre différents benchmarks.

4.1 Introduction

Les modèles de coût énergétique précis et robustes sont cruciaux pour de nombreux projets ayant pour objectif d'améliorer l'efficacité énergétique des équipements informatiques. Comme discuté dans le chapitre 3, l'énergie peut être optimisée à différents niveaux: matériel, système d'exploitation (SE). En effet pour optimiser la consommation énergétique dans la conception des BDs, la définition d'un modèle estimant le coût énergétique des opérations est une étape fondamentale et constitue même la première étape dans toutes les techniques d'optimisation orientées logicielles. Nous avons exploré dans le chapitre précèdent les différents modèles proposés dans l'état de l'art pour estimer la consommation énergétique des STRs. Nous avons remarqué que ces approches ne proposent pas une méthodologie détaillée et claire pour la conception des modèles de coût énergétique. En plus avec des architectures toujours plus complexes et variables, ces modèles de coût énergétique sont inappropriés et difficilement portables. Nous proposons dans cette section une méthodologie claire pour la conception des modèles de coût énergétique lors du traitement des requêtes en tenant compte de tous les paramètres pertinents.

4.2 Modèles de coût énergétique (McE) dans les STRs

Outre les besoins de prédiction de la consommation d'énergie dans le but de maximiser l'efficacité énergétique, les utilisateurs et les opérateurs des systèmes ont besoin de connaître et de comprendre comment leurs modes d'utilisation affectent la consommation énergétique des systèmes. Pour évaluer le gain de l'efficacité énergétique produit dans les choix de conception des logiciels ou du fonctionnement d'une partie des composants physiques, il est préférable de déployer un modèle d'estimation conçu à cet effet. Cependant, les modèles développés doivent être légers pour ne pas interférer avec l'énergie consommée par les systèmes qu'ils tentent d'estimer. Le modèle de coût devra :

— Être suffisamment précis en incorporant tous les paramètres pertinents relatif au coût;

— Être assez simple pour être facilement compris et permettre son utilisation et son exploitation dans les techniques d'optimisation.

4.2.1 Niveau de modélisation

La modélisation d'une requête peut se faire à trois niveaux d'abstraction. Le niveau de modélisation retenu peut impacter la précision du modèle. Lors du traitement d'une requête, un ensemble d'opérations s'exécutent et interagissent avec diverses relations. La manière la plus évidente d'évaluer les opérations consiste simplement à évaluer une opération à la fois dans un ordre séquentiel. Le résultat de chaque opérateur individuel doit être écrit sur la mémoire pour être utilisé comme entrée dans une autre opération. Une autre approche consiste à grouper les opérateurs en pipelines, de cette manière l'exécution des opérateurs à l'intérieur du pipeline est entrelacée plutôt qu'en séquentielle. Dans ce scénario, aucune sauvegarde temporaire n'est créée. La modélisation pourrait donc se faire soit au niveau opérateur (niveau le plus bas), ou au niveau pipeline (niveau intermédiaire) ou au niveau requête (niveau le plus haut). La figure 4.1 illustre les différents niveaux de modélisation.

— **Niveau opérateur:** Il s'agit de dégager les caractéristiques d'exécution de chaque opérateur de manière individuel. Cette approche est complexe car elle demande la compréhension de l'implémentation algorithme de chaque opérateur.

— **Niveau pipeline:** La modélisation à ce niveau sacrifie une certaine précision par rapport à la modélisation au niveau opérateur. Il s'agit de dégager les caractéristiques communes aux opérations qui s'exécutent de manière entrelacée dans le pipeline.

— **Niveau requête:** Il s'agit de dégager les caractéristiques de la requête, ces caractéristiques concernent globalement les estimations de l'optimiseur telles que les cardinalités et les coûts d'exécution des plans ainsi que le nombre d'occurrence de chaque type d'opérateur dans la requête.

4.2.2 Mode d'exécution des requêtes

Les SGBDs modernes fournissent différents modes d'exécution des requêtes, la plupart d'eux permettent le mode d'exécution inter-requête et le mode d'exécution isolé séquentiel. Le parallélisme inter-requêtes permet à plusieurs requêtes de plusieurs utilisateurs connectés à la BD de s'exécuter de manière concurrente. Avec le mode isolé séquentiel, les SGBDs choisissent un plan d'exécution dit séquentiel pour exécuter

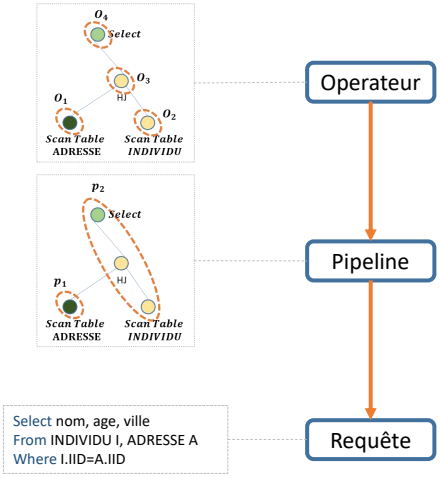


FIGURE 4.1 – Différents niveaux de modélisation.

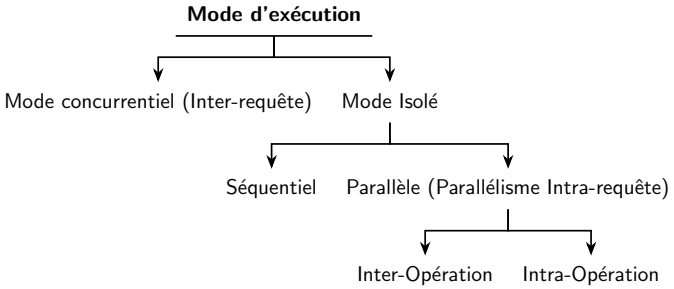


FIGURE 4.2 – Mode d'exécution des requêtes.

une seule requête sans parallélisme intra-opération. Les architectures modernes offrent un modèle de fonctionnement multi-cœurs, ces ordinateurs sont équipés de processeurs très puissant pouvant avoir jusqu'à 20 cœurs ou plus. L'avènement de ces configurations a obligé les concepteurs de SGBDs à quitter les modèles traditionnels qui consistaient à générer un plan séquentiel pour le traitement d'une requête à un mode d'exécution isolé parallèle (parallélisme intra-requête). Dans ce mode d'exécution, l'optimiseur génère un plan dit parallèle dans lequel les opérateurs sont divisés en plusieurs sous-opérations plus petites qui s'exécutent simultanément sur différentes unités de traitement sur une partie des données. La figure 4.2 illustre les différents modes d'exécution d'une requête.

— **Mode isolé séquentiel:** Une seule requête s'exécute de manière indépendante sur une instance de la BD.

— **Mode isolé parallèle(Parallélisme intra-requête):** une seule s'exécute en mode parallèle sur une instance de la BD en exploitant le parallélisme interopération ou/et intra-opération.

— **Mode Concurrent(Parallélisme inter-requête):** Plusieurs requêtes peuvent s'exécuter simultanément sur la même instance de données.

4.3 Méthodologie de conception des McE

Dans cette section, nous présentons la démarche suivie pour développer nos différents modèles de coût qui estiment la consommation d'énergie d'une seule requête SQL exécutée en mode isolé (séquentiel ou parallèle) sur une instance de la BD sur des architectures différentes. Notre approche s'appuie sur l'analyse des observations et des expérimentations menées à partir des charges de travail (un ensemble de requêtes) sur des quantités variables de données afin de repérer des régularités nous permettant de dégager les paramètres les plus pertinents.

Le développement d'un modèle de coût énergétique peut être fastidieux et long car le processus nécessite une compréhension approfondie de nombreux aspects tel que: les caractéristiques des systèmes de stockage (le modèle de données, la structure de stockage, le mode d'exécution), la plateforme de

déploiement (le type du processeur, type de périphérique de stockage), les techniques d'apprentissage automatique (modèles régressifs, modèles cognitifs), la métrique évaluée (le temps, l'énergie), etc. (figure 4.3). Le processus de conception d'un modèle énergétique doit suivre des étapes successives bien définies

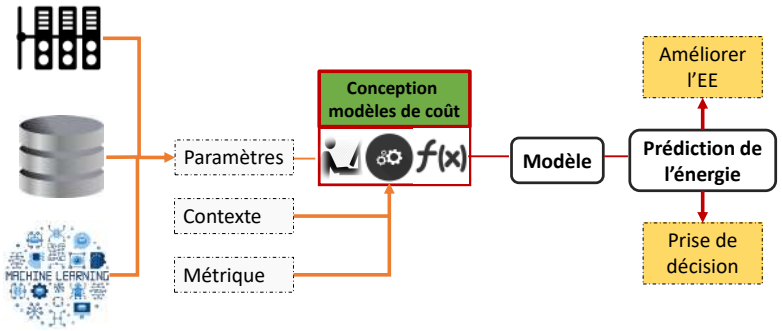


FIGURE 4.3 – Les différents aspects lors de la phase de conception du modèle.

dans le but de produire un modèle de coût précis puisque ce dernier est utilisé pour des fins d'efficacité énergétique, d'estimateur énergétique ou d'orientation dans les prises de décision. Notre méthodologie est fondée principalement sur les étapes suivantes (figure 4.4): Audit du SGBD, identification des paramètres, construction du modèle, validation du modèle, et déploiement.

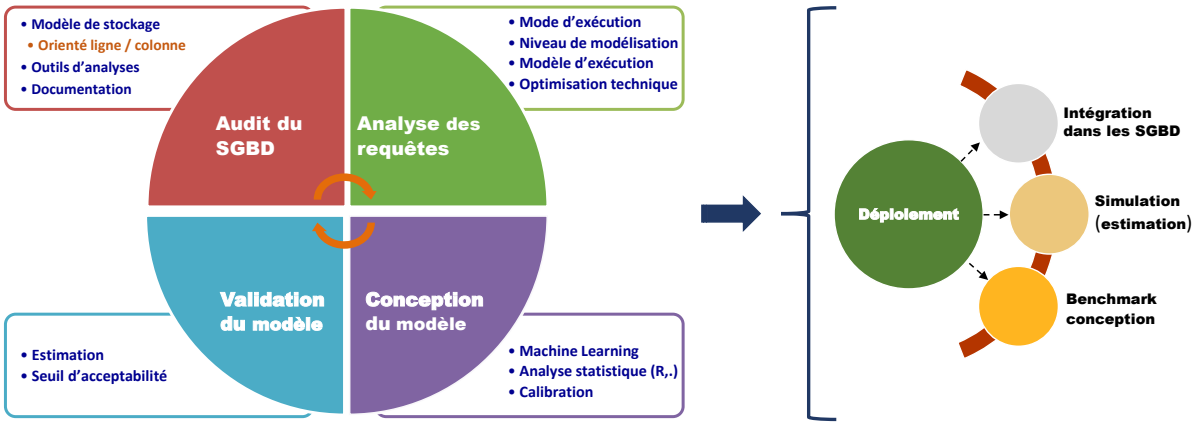


FIGURE 4.4 – Les différentes étapes du processus de développement d'un modèle.

4.3.1 Audit des SGBDs

Le mot audit ici ne reflète pas son utilisation conventionnelle dans le monde des BDs, il fait référence à l'analyse du principe de fonctionnement des SSDs afin de dégager ces principales caractéristiques telle que son modèle de traitement, son modèle de stockage, etc. Il s'agit de comprendre comment la configuration logique du SGBD exploite les caractéristiques physiques (hardware) du serveur: le processeur, la mémoire RAM, disques, etc.

4.3.2 Identification des paramètres

L'identifiabilité est un processus qui permet de déterminer les paramètres exerçant une influence sur le coût du modèle de façon à obtenir une représentation satisfaisante du système. La sélection des paramètres s'appuie particulièrement sur l'analyse des études expérimentales en comparant les entrées /sorties du système. Pour la modélisation de l'énergie, la consommation énergétique de chaque composant doit être évaluée et comparée afin d'identifier les composants les plus énergivores. Cette phase est complexe car elle affecte considérable la précision du modèle attendu. Plus formellement, on note ceci par l'équation suivante:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{y}, \mathbf{z})$$

Où f représente l'état du système. La variable x exprime la configuration matérielle retenue, y représente le SGBD étudié et z représente la charge de travail. En sortie, nous avions un ensemble de paramètres

($\dot{x}$) qui sont responsables de la consommation énergétique. Dans la section 2.4, nous avons répertorié un ensemble de facteurs pouvant influencer la consommation énergétique des SSDs. Le but de cette étape est de dégager ceux (pertinents) qui sont responsables (les plus énergivores) de la consommation énergétique dans le SGBD.

4.3.3 Formalisation du modèle

Cette étape consiste à utiliser les paramètres influents retenus pour choisir un formalisme dans lequel le modèle de coût énergétique sera représenté. Le choix du formalisme est fait en s'appuyant sur des techniques d'apprentissage (statistiques) telles que les techniques régressives, les techniques cognitives, etc. Le principal défi dans la modélisation énergétique est qu'il est impossible d'estimer l'énergie consommée par chaque composant directement de manière isolé. Par conséquent les méthodes analytiques ou de simulation peuvent ne pas s'avérer très efficace et précis dans de tels cas comparé aux approches cumulées à boite noires fondées sur l'apprentissage automatique. Plus formellement, le formalisme générique d'un modèle de coût énergétique peut s'écrire par l'équation suivante:

$$\mathcal{M}(\dot{\mathbf{x}}) = \sum_{\mathbf{i=1}}^{\mathbf{n}} (\beta_{\mathbf{i}} \oplus \mathbf{P_i})$$

Où $\mathcal{M}(\dot{x})$ représente le modèle de coût énergétique attendu, P_i représente le $i^{ème}$ paramètre, β_i le coût énergétique du $i^{ème}$ paramètre, n le nombre de paramètre et $\oplus$ représente la combinaison entre les paramètres.

4.3.4 Validation du modèle

Cette étape est cruciale dans le processus de conception des modèles de coût énergétique car elle permet d'évaluer la précision du modèle obtenu en démontrant son exactitude par rapport à une utilisation donnée. Il existe différentes approches pour évaluer l'efficacité d'un modèle de coût. Elles peuvent être définies sur la base: (a) de l'intuition d'un expert (inspection visuelle); (b) de la comparaison entre les mesures prédites par le modèle et les mesures réelles du système; et (c) des résultats théoriques/analyse [78]. Parmi ces trois approches, l'approche de la comparaison des mesures (système réel vs modèle) est le moyen le plus précis pour valider un modèle. Dans notre travail nous adoptons cette approche de validation pour évaluer la précision et l'exactitude de nos modelés proposés. L'objectif ultime de la validation du modèle est de confirmer l'utilisabilité d'un modèle dans des études de prédiction, d'efficacité énergétique, de définition de bancs d'essai, etc. De manière formelle, elle peut être décrite comme suite:

$$\mathcal{V}(\mathcal{M}(\dot{\mathbf{x}}), \mathbf{x}^{'}, \mathbf{y}^{'}, \mathbf{z}^{'}) = \alpha, \text{si} \begin{cases} \alpha < Seuil_{Erreur}, & Modèle\ Validé. \\ sinon, & Modèle\ non\ Validé \end{cases}$$

Où $\mathcal{V}$ définit la méthode utilisée pour valider le modèle. Elle prend en entrée le modèle de coût construit $\mathcal{M}(\dot{x})$, la configuration matérielle notée par $x^{'}$, $y^{'}$ représente le SGBD et $z^{'}$ représente la charge de travail. α représente la somme des erreurs d'estimation. Selon les objectifs fixés, si cette valeur(α) est inférieure à une valeur dite seuil des erreurs ($Seuil_{Erreur}$) d'estimation, le modèle est accepté sinon une phase de calibration des valeurs des paramètres commence.

4.3.5 Exploitation

Après l'étape de la validation du modèle, il est possible de l'exploiter dans des études de prédiction des tendances de l'efficacité énergétique, de prédiction de la consommation énergétique des requêtes, dans la conception des outils de benchmarks, etc. Ces prédictions peuvent alors être utilisées pour améliorer l'efficacité énergétique des SSDs, par exemple en incorporant le modèle dans les techniques telles que l'ajustement dynamique de la fréquence (DVFS), dans le traitement des requêtes au niveau des STRs, dans la conception physique, ou encore dans les techniques de désactivation des composants inutilisés, etc.

4.4 Audit des systèmes SSDs étudiés

Notre travail a requis l'utilisation des SGBDs)réels pour conduire des études analytiques afin de construire nos modèles de coût et des études expérimentales pour valider les modèles proposés. Pour ce

faire, nous avons étudié le fondement de trois systèmes relationnels différents. Il s'agit de *PostgreSQL-DB*[1], *Monet-DB*[2] *et Hyrise-DB*[3]. Nous avons opté pour ces systèmes car ils sont très largement utilisés et libres (open sources). Ils offrent également un mode de traitement parallèle des requêtes et diffèrent dans leur modèle de stockage. Le fait que les détails architecturaux internes et le code source de ces systèmes (SGBDs) sont accessibles, cela nous a permis de comprendre l'interaction complexe entre le fonctionnement de ces systèmes et le matériel sur lequel ils fonctionnent. Nous pouvons aussi ajouter ou modifier le code interne pour obtenir des informations nécessaires lors de la conception et de la validation de nos modèles. Dans cette section, nous donner une introduction concise sur chacun des trois systèmes (*PostgreSQL, MonetDB, Hyrise*), en mettant en avant sur les caractéristiques essentielles qui sont importantes dans le contexte de notre travail.

4.4.1 PostgreSQL

PostgreSQL est un SGBD relationnel objet (ORDBMS) dont le modèle de stockage est Orienté-Ligne (O-L). Développé à l'université de Californie, PostgreSQL est un outil libre disponible selon les termes d'une licence de type BSD (Berkeley Software Distribution License). Il supporte une grande partie du standard SQL tout en offrant de nombreuses fonctionnalités modernes: requêtes complexes, clés étrangères, triggers, vues modifiables, intégrité transactionnelle, contrôle des versions concurrentes (MVCC, acronyme de « Multi Version Concurrency Control »). De plus, il peut être étendu par l'utilisateur de multiples façons, en ajoutant, par exemple: de nouveaux types de données, de nouvelles fonctions, de nouveaux opérateurs, de nouvelles fonctions d'agrégat, de nouvelles méthodes d'indexage, de nouveaux langages de procédure [55].

4.4.1.1 Architecture

PostgreSQL implémente un système de stockage orienté ligne, où les attributs d'un enregistrement (ou tuple) sont placés de manière contigus dans la mémoire de stockage. Avec cette architecture de stockage, le système peut récupérer rapidement les enregistrements d'une ligne avec un minimum d'opérations (entrées-sorties) sur le disque. PostgreSQL utilise un modèle client/serveur dans lequel chaque processus client sera traité par un processus serveur. Une session PostgreSQL comprend:

— Un processus serveur, qui gère les fichiers de la BD, accepte les connexions à la BD à partir d'applications clientes et effectue des actions sur la BD à la demande des clients. Ce processus est appelé POSTGRES.

— Un processus client (front-end), qui demande d'effectuer des opérations sur la BD. Les applications clientes peuvent être de nature très diverse: un client peut être un outil orienté texte, une application graphique ou une application web.

Le serveur (*Postmaster*) écoute sur le port réseau défini par le système pour détecter une nouvelle demande de connexion des utilisateurs. Pour chaque utilisateur, il crée un nouveau processus (*forks*). Le processus client communique avec le nouveau processus crée côté serveur sans intervention du processus POSTGRES initial. Ainsi, le processus maître (Master) reste en position d'écoute en attendant d'autres connexions (utilisateurs), tandis que les processus clients et processus serveurs associés travaillent étroitement pour récupérer les données. Pour plus d'information sur le fonctionnement client/serveur referez-vous à la documentation [55]. La figure 4.5 présente une architecture du système PostgreSQL.

4.4.1.2 Exécution de la requête

Le processus d'exécution des requêtes dans PostgreSQL suit les étapes classiques évoquées dans la Section 2.2.2.1. L'exécuteur traite le plan physique optimisé de la requête, récupère les tuples en évaluant les opérations requises (balayage séquentiels, tris, jointure, etc.). La stratégie d'exécution des requêtes est basée sur une approche en pipeline. Le mode d'exécution intra-parallèle a été introduit dans la version 9.6 du système. Pour ce fait, le système associe un groupe workers pour le traitement de la requête. Lorsque l'optimiseur sur la base des statistiques constate que le mode intra-parallèle est la stratégie la plus rapide pour une requête particulière, il crée alors un plan d'exécution dit parallèle qui inclut un nœud *Gather* ou un nœud $Gather - Merge$. Le nombre de «background-workers» peut

1. `https://www.postgresql.org/`
2. `https://www.monetdb.org/`
3. `https://hpi.de/plattner/projects/hyrise.html`

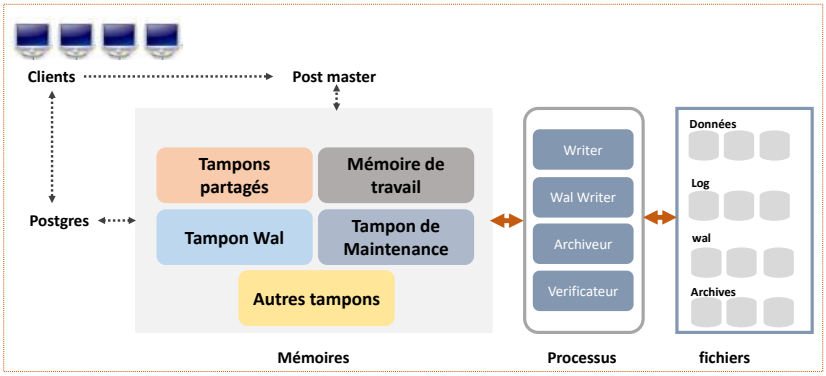


FIGURE 4.5 – Architecture du système PostgreSQL.

être définit par les paramètres *max_worker_processes* et *max_parallel_workers*. Par exemple, la commande *set max_parallel_workers_per_gather* = 3 permet de fixer le nombre de Workers à 3. Pour traiter ou forcer l'optimiseur à générer un plan d'exécution dit séquentiel (série), la variable *max_parallel_workers_per_gather* doit être fixée à 0. La figure 4.6 illustre le plan parallélisé de la requête Q_i (2.2.2.1) obtenue par la commande *Explain*. Les nœuds Data Send(DS), Data Receive(DR) et Gather sont ajoutés pour implémenter les opérations de distribution et de fusion dans un plan parallèle comme illustré sur la figure 4.6.

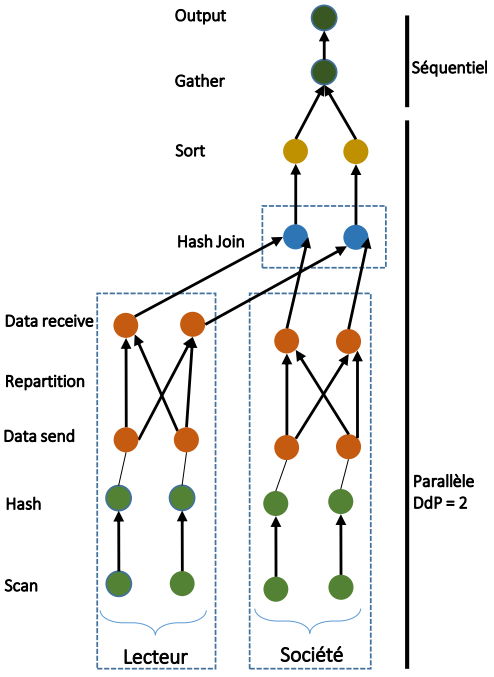


FIGURE 4.6 – Plan parallèle de la requête.

4.4.2 MonetDB

MonetDB est un système open source, orienté-colonne (O-C), développée par Centrum Wiskunde & Informatica(CWI) dans les années 1994. Les SSDs orientés colonnes (column-stores) stockent toutes les valeurs d'une colonne ensemble, puis les valeurs de la colonne suivante, et ainsi de suite de manière contiguë et compressée. MonetDB est conçu dans le but d'atteindre des performances élevées pour des requêtes gourmandes en ressources, telles que celles créées pour le traitement analytique en ligne (OLAP) ou les applications de datamining. MonetDB utilise un modèle de stockage décomposé (Decomposed Storage Model - DSM), qui stocke chaque colonne de la relation dans une table binaire séparée, appelée table d'association binaire (Binary Association Table - *BAT*) [13]. MonetDB a apporté de nouvelles innovations en modifiant les couches inférieures des SGBDs classiques: (a) un modèle de stockage basé sur la fragmentation verticale, (b) une architecture d'exécution des requêtes adaptée aux systèmes multi-cœurs ce qui explique le pourquoi des meilleures performances de MonetDB comparée aux autres SGBDs classiques

réalisant le même travail. Il fut l'un des premiers systèmes à adapter son optimiseur de requêtes pour tenir compte des mémoires caches du CPU et permet aussi la création automatique et auto-adaptation des index [73].

4.4.2.1 Architecture

L'architecture de MonetDB est composée de trois couches, chacune ayant son propre système d'optimiseurs. La couche supérieure sert à fournir une interface SQL, la couche intermédiaire contient les optimiseurs pour le langage d'assemblage MonetDB (MAL), et la couche inférieure est le noyau de la BD qui permet d'accéder aux tables d'association binaires.

Le front-end est la couche supérieure, elle sert d'interface entre les utilisateurs et le système sous-jacent. Le front-end permet de recueillir les requêtes des utilisateurs exprimées en SQL, ou SciQL [4] ou SPARQL [5]. Les requêtes sont analysées et optimisées en fonction des représentations spécifiques propre à chaque domaine, comme l'algèbre relationnelle pour SQL. Les plans logiques générés sont traduits en instructions *MAL* (MonetDB Assembly Language), qui sont transmises à la couche suivante. La couche intermédiaire ou back-end consiste en une collection de modules d'optimisation, qui sont assemblés dans un pipeline d'optimisation. Chaque optimiseur prend le plan MAL et le transforme en un plan plus efficace, éventuellement argumenté par des directives de gestion des ressources et des flux de contrôle. La couche inférieure est le noyau de la BD, qui donne accès aux données stockées dans les tables d'association binaires (BAT). Le modèle de stockage représente les tables relationnelles en utilisant la fragmentation verticale, en stockant chaque colonne dans une table (OID, value) séparée. La colonne de gauche (object-identifier - OID) est appelée la tête, tandis que la colonne de droite (valeur) est la queue. La figure 4.8a illustre l'architecture interne de MonetDB.

4.4.2.2 Exécution de la requête

Le noyau MonetDB est une machine abstraite, programmée en langage assembleur MonetDB (MAL). Chaque opérateur relationnel du plan est traduit en algèbre BAT puis compilé en instructions MAL. Le modèle utilise une approche basée sur une «opération-à-la-fois» (operator-at-a-time) pour exécuter les programmes MAL.

Pour illustrer un exemple d'algèbre relationnelle produite par MonetDB, nous prenons la requête Q_i de la section 2.2.2.1. Les figures 4.7a et 4.7b illustrent la traduction de la requête SQL en algèbre relationnelle puis en instructions MAL produites par le système MonetDB respectivement.

Pour le mode d'exécution parallèle, le système génère premièrement un plan d'exécution séquentiel puis ajoute lors de la phase d'optimisation les instructions pour l'intra-parallélisme. Les opérateurs MAL sont marqués comme étant soit "bloquants", soit "parallélisables" [145]. MonetDB utilise autant de cœurs que possible pour exploiter le mode inter-parallélisme et intra-parallélisme dans le but d'accroître les performances. La commande $gdk\_nr\_threads =< nombre >$ peut être utilisé pour limiter le nombre de nœud (cœurs) affectés à chaque session.

4.4.3 Hyrise

Hyrise [6] a été présenté pour la première fois en 2010, développé par *Hasso-Plattner-Institute* en Allemagne sous la direction du professeur Hasso Plattner afin d'introduire le concept du modèle de stockage hybride (Orienté-Ligne et Orienté-Colonne) pour les BDs en mémoire (in-memory). Le système Hyrise est une nouvelle implémentation basée sur le modèle relationnel, qui scinde verticalement puis horizontalement de manière automatique les tables de données en des petites partitions, de largeurs variables en fonction de la façon dont les colonnes de la table sont invoquées dans les requêtes [56][35]. Le système vise à maximiser les performances de la mémoire cache pour les requêtes analytiques (de type OLAP) et transactionnelles (de type OLTP) en dynamisant la structure de conception du stockage des données. L'objectif des concepteurs du système Hyrise est de fournir une plateforme de gestion des données en mémoire flexible, permettant de mener des expérimentations afin de valider les nouveaux concepts autour de la gestion des données en mémoire.

4. `https://projects.cwi.nl/scilens/Resources/SciQL.html`
5. `https://www.w3.org/TR/rdf-sparql-query/`
6. `https://hpi.de/plattner/projects/hyrise.html`

(a) Algèbre relationnelle.

```
| rel
1 | project (
2 | | project (
3 | | | join (
4 | | | | select (
5 | | | | | table(sys.lecteur) [ "lecteur"."lid" NOT NULL HASHCOL  as "l"."lid", "lecteur"."nom" as "l"."nom", "lecteur"."email" as "l"."email", "lecte
6 : ur"."ville" as "l"."ville" ] COUNT
7 | | | | ) [ "l"."ville" = varchar(50) "Poitiers" ],
8 | | | | select (
9 | | | | | table(sys.société) [ "société"."sid" NOT NULL HASHCOL  as "s"."sid", "société"."nom" as "s"."nom" ] COUNT
10| | | | ) [ "s"."nom" = varchar(50) "ABAN" ]
11| | | ) [ "l"."lid" NOT NULL HASHCOL  = "s"."sid" NOT NULL HASHCOL  ]
12| | ) [ "l"."nom", "l"."email" ]
13| ) [ "l"."nom", "l"."email" ] [ "l"."nom" ASC, "l"."email" NULLS LAST ]
```

(b) Instructions MAL générées.

```
| mal
1 | function user.s4_0():void;
2 |   X_3:void := querylog.define("explain select l.nom, l.email\nfrom lecteur l, société s\nwhere l.lid = s.sid and\ns.nom = \\'ABAN\\' and\nl.ville
3 :            = \\'Poitiers\\'\norder by l.nom, l.email desc;":str, "default_pipe":str, 67:int);
4 |   barrier X_168:bit := language.dataflow();
5 |     X_102:bat[:str] := bat.pack("sys.l":str, "sys.l":str);
6 |     X_103:bat[:str] := bat.pack("nom":str, "email":str);
7 |     X_104:bat[:str] := bat.pack("varchar":str, "varchar":str);
8 |     X_105:bat[:int] := bat.pack(50:int, 50:int);
9 |     X_106:bat[:int] := bat.pack(0:int, 0:int);
10|      X_99:bat[:str] := bat.new(nil:str);
11|     X_100:bat[:str] := bat.new(nil:str);
12|   exit X_168:bit;
13|     sql.resultSet(X_102:bat[:str], X_103:bat[:str], X_104:bat[:str], X_105:bat[:int], X_106:bat[:int], X_99:bat[:str], X_100:bat[:str]);
14|   end user.s4_0;
```

FIGURE 4.7 – Algèbre relationnelle et instructions MAL générées.

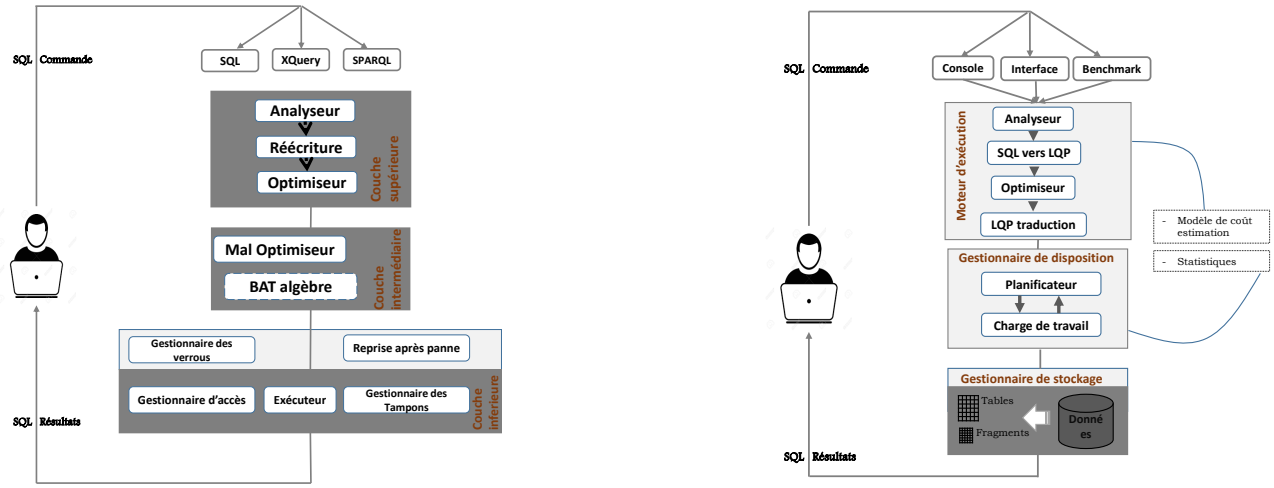
4.4.3.1 Architecture

L'architecture générale du système *Hyrise* est composée de trois composants principaux : un gestionnaire de stockage, un moteur d'exécution des requêtes et un gestionnaire de disposition (layout) illustrée sur la figure 4.8b. Le gestionnaire de stockage est responsable de la création et de la maintenance des conteneurs hybrides qui stockent les données. Le stockage est basé sur le partitionnement vertical des tables relationnelles en un ensemble disjoint d'attributs. Chaque partition est représentée en interne comme un conteneur. Les conteneurs sont stockés dans un seul bloc compressé dans la mémoire principale (RAM) et peuvent être fusionnés après décompression pour reconstruire la table relationnelle initiale. Le gestionnaire de disposition analyse une charge de requête fournie et suggère la meilleure disposition possible (partitionnement) au gestionnaire de stockage.

4.4.3.2 Exécution de la requête

L'exécuteur des requêtes reçoit les requêtes des utilisateurs, crée un plan physique pour chaque requête et les exécute en appelant le gestionnaire de stockage. Plus précieusement dans le moteur de traitement des requêtes, le *SQLParser* traduit la chaîne SQL en un arbre syntaxique abstrait exprimé sous forme de structures de données $C++$. Ensuite, le traducteur «SQL vers $LQP(Logic-Query-Plan)$» modifie le niveau d'abstraction en créant un plan de requête logique (graphe acyclique orienté) dont les nœuds représentent les opérations de l'algèbre relationnelle. Lors de la phase d'optimisation, une série de règles est appliqué au LQP, qui le transforme en une version plus efficace et sémantiquement équivalente (section 2.2.2.1). Après l'optimisation, le LQP est traduit en un plan physique de requête (PQP) par le module de traducteur «LQP vers PQP».

Par contre si le traducteur (compilateur) juste-à-temps (Just-In-Time - JIT) est activé, le traducteur «LQP vers PQP» n'est plus chargé de la traduction, mais le traducteur «$LQPJIT-aware$», qui crée des opérateurs JIT dont le code est personnalisé en utilisant les informations d'exécution [36]. Le moteur JIT de Hyrise est basé sur trois composants: le JIT-Repository, qui contient les opérateurs de la table JIT; le traducteur JIT-Aware LQP, qui traduit les plans des requêtes logiques en une chaîne d'opérateurs de la table JIT; et le générateur de code JIT, qui optimise le code des opérateurs et les fusionne en une seule boucle. La version actuelle de Hyrise (V2) implémente les opérations de la projection, la sélection, le tri et le regroupement, et supporte la matérialisation précoce et tardive [56].



(a) Architecture du système MonetDB

(b) Architecture du système hyrise.

FIGURE 4.8 – Architecture de fonctionnement interne des systèmes.

4.5 Analyse et études comparatives des SSDs étudiés

Cette section présente une étude expérimentale comparant nos trois SGBDs (PostgreSQL, MonetDB, Hyrise) étudiés en termes de temps d'exécution, de la consommation énergétique et du nombre de défaut caches (Caches misses) lors de l'exécution des requêtes. Nous évaluons également le temps de chargement des données dans le système et l'énergie consommée. Ces expérimentations sont menées à l'aide des dix (10) premières requêtes de Sélection-Projection-Jointure (SPJ) du benchmark TPC-H. Le système physique sur lequel nous effectuons ces évaluations est décrit à la section 4.5.1. La section 4.5.2 présente les résultats de la consommation d'énergie et l'évaluation de la performance.

4.5.1 Environnement de l'étude

Serveur: Nos expérimentations sont menées sur un Dell Précision T1700 avec une carte mère Dell 073MMW (Socket 1150 LGA), un processeur Intel Core i7 4770 de 3,40 GHz (1 CPU – 4 Cœurs - 8 Threads), 16 Go [7] RAM DDR3 mémoire principale, carte graphique NVIDIA, ATA Disque Western Digital (WD) 500 Go. L'enveloppe thermique, ou TDP (Thermal Design Power) est de 84,0 W watts. Le serveur a la capacité d'ajuster automatiquement sa consommation énergétique en utilisant la technique de DVFS. Ainsi, un nœud cœur inutilisé est mis en mode d'économie d'énergie dynamiquement. Un descriptif des valeurs maximales de la consommation statique et dynamique de certains composants sont donnés dans le tableau 4.1.

		Puissance (Watts)	
	Composants	**Statique (Min)**	**Dynamique (Max)**
Serveur	Intel Core i7-4770	0.8 W	11.5 W
	CPU Package	5.9 W	19.7 W
	Mémoire RAM: 2x4Go DDR3@ 798MHz	1.6 W	4.6 W
	Disque HDD-WD	3.6 W	7.6 W

TABLEAU 4.1 – Énergie statique et dynamique de certains composants du système.

SE and SGBDs: Pour toutes nos expérimentations, le serveur utilise comme système d'exploitation Ubuntu 18.04 bionic (noyau 5.0.0-27-générique) avec les trois systèmes SGBDs installés: PostgreSQL (version 10.10), MonetDB (version 11.33.11) et Hyrise (version V2).

Benchmarks: Nous utilisons le benchmark TPC-H, conçu pour illustrer des entrepôts de données de grands volumes de données en exécutant des requêtes d'une très grande complexité. Il est composé de huit tables (PART, PARTSUPP, REGIONAL, FOURNISSEUR, CLIENT, LIGNE, NATION, et

7. Giga Octet

COMMANDES). Nous générons des données à différents facteurs d'échelle (*Scale Factor - SF*). Nous avons opté pour les benchmarks TPC parce qu'ils sont largement connus et possède de nombreuses années d'expériences dans la définition des transactions et des benchmarks des BDs.

Configuration: Après la génération des données du benchmark TPC-H dans les fichiers textuels de type CSV (valeurs séparées par des virgules), nous avons utilisé la commande COPY de SQL pour charger les données dans les différents SGBDs. Nous avons collecté les statistiques sur le temps d'exécution et sur la consommation énergétique lors du chargement des données et lors de l'exécution de chaque requête dans les différents SGBDs. Pour la mesure de l'énergie, nous avons utilisé un équipement externe Wattmètre appelé « *Watts Up Pro* [8]» placée entre la source d'alimentation électrique et le serveur des BDs. Pour le dénombrement des défauts de caches, nous utilisons les outils Cachegrind [9], Processor Counter Monitor(PCM) [10] et PAPI [11] dans un programme *Java* que nous avons développé.

4.5.2 Évaluation des résultats

Nous effectuons plusieurs expérimentations pour évaluer le temps de chargement des données, le temps d'exécution des requêtes et l'énergie consommée lors du traitement des requêtes sur le système dont la configuration est décrite ci-dessus. Nous présentons en détails dans cette section nos résultats d'expérimentation.

4.5.2.1 Temps et énergie consommée lors du chargement des données

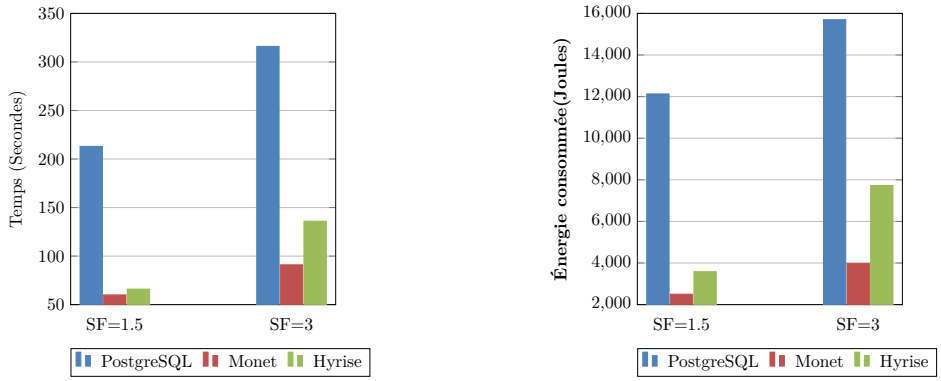
On analyse le comportement de PostgreSQL, MonetDB et Hyrise lors du chargement de deux jeux de données de taille $1,5$ Go ($SF = 1,5$) et 3 Go ($SF = 3$). La figure 4.9a représente le temps de chargement des données des deux jeux de données dans chaque système. Partant de l'analyse de cette figure, mous constatons que le temps de chargement augmente pour chaque système à mesure que la taille du jeu de données augmente. MonetDB surpasse largement les deux autres systèmes SGBDs dans les deux cas; lorsqu'on considère le jeu de données de 3Go, MonetDB est de $3,5$ fois plus rapide que PostgreSQL et $1,5$ fois plus rapide que celui du système Hyrise. Après le système MonetDB, Hyrise révèle de meilleure performance comparée à celle de PostgreSQL. PostgreSQL affiche les pires performances de chargement, la raison en est que les systèmes orientés-lignes nécessitent en plus des données à écrire, l'écriture des informations auxiliaires dans chaque ligne (par exemple, l'en-tête) en mémoire. L'augmentation du temps de chargement des données dans le système Hyrise comparé à celui de MonetDB est due au traitement supplémentaire requis pour la compression des données. Hyrise applique un taux de compression plus élevé que le système MonetDB car ce dernier ne compresse que les valeurs de type chaîne de caractères. Comme pour le temps de chargement des données, l'énergie consommée lors du chargement de chaque jeu de données est illustrée sur la figure 4.9b. L'énergie est une quantité physique influencée par le temps qui s'écoule. Sur la figure 4.9b, nous remarquons que l'énergie consommée par PostgreSQL est plus considérable que celle de MonetDB et Hyrise.

4.5.2.2 Temps d'exécution et énergie consommée lors du traitement des requêtes

Dans cette expérience, nous évaluons le temps d'exécution et l'énergie consommée dans les systèmes PostgreSQL et MonetDB lors du traitement des requêtes sur deux jeux de données: 30 Go et 100 Go. Ensuite, nous analysons avec un jeu de données de taille 5 Go le comportement des trois systèmes (PostgreSQL, monetDB, hyrise). Nous collectons puis analysons le temps d'exécution de chaque requête SPJ et sa consommation énergétique. Les figures 4.10b et 4.10c résument le temps d'exécution des requêtes sur les systèmes PostgreSQL et MonetDB. La figure 4.10a représente le temps d'exécution des trois systèmes avec un jeu de données de 5 Go.

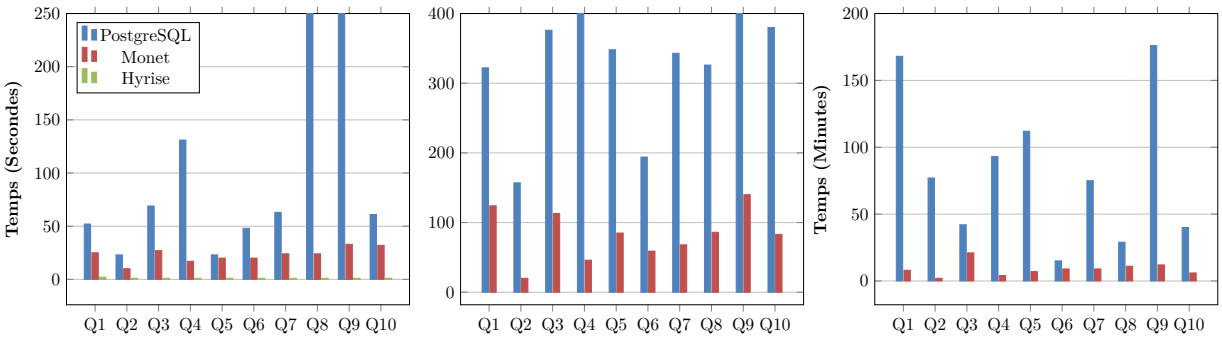
Ces résultats montrent que les performances de Hyrise surpassent celles obtenues par PostgreSQL et MonetDB sur le jeu de données de 5 Go pour toutes les 10 premières requêtes. Nous remarquons que les temps d'exécution sur le système Hyrise de la plupart des requêtes dépassent largement ceux de PostgreSQL. Ces écarts de performance constaté au niveau du système PostgreSQL sur les figures 4.10a,4.10b et 4.10c peuvent s'expliquer par le fait qu'il accumule un grand nombre de défauts de caches

8. `http://www.energyalternatives.ca/pdf/wattsup_TTW.pdf`
9. `https://valgrind.org/docs/manual/cg-manual.html`
10. `https://github.com/opcm?tab=repositories`
11. `https://icl.utk.edu/papi/`



(a) Temps de chargement (b) Consommation énergétique

FIGURE 4.9 – Temps et énergie consommée lors du chargement des données.

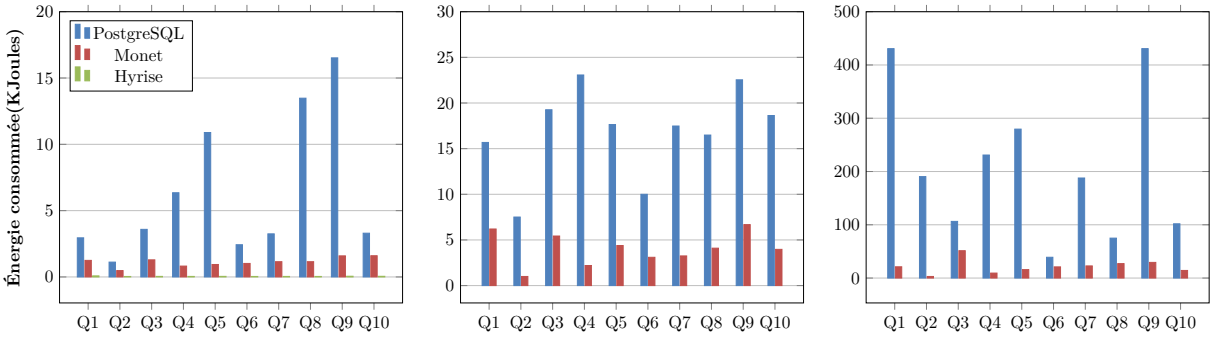


(a) SF=5 Go (b) SF=30 Go (c) SF=100 Go

FIGURE 4.10 – Comparaison du temps d'exécution des requêtes $Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}$ du benchmark TPC-H en mode parallèle.

(pages) lors du traitement des requêtes. En fait les systèmes orientés-Lignes doivent analyser et utiliser l'ensemble des n-tuples (enregistrements) plutôt que de se limiter aux valeurs de colonnes uniquement nécessaires dans les requêtes. Par conséquent, ces lignes entières plus l'arbre d'index (B+) construit ne peuvent pas résider assez longtemps dans la mémoire principale ou dans les mémoires caches. elles doivent être réécrites sur le disque, ce qui conduit à de nombreux entrées-sorties (E/S) disques. Pour les systèmes orientés-colonnes comme MonetDB, seules les valeurs des colonnes uniquement nécessaires pour répondre aux requêtes sont chargées. Dans les systèmes hybrides comme Hyrise, les avantages des systèmes de stockage orienté-colonne et orienté-ligne sont combinés grâce l'auto-adaptabilité qu'est offert par le gestionnaire de stockage. Ils sont conçu dans l'objectif de bien s'adapter avec l'exécution parallèle sur les processeurs multi-cœurs.

Pour comparer la consommation d'énergie sur nos trois systèmes lors de l'exécution des requêtes, nous mesurons la puissance dissipée par seconde à l'aide d'un appareil appelé wattmètre. À la fin de ces mesures, nous calculons l'énergie moyenne consommée par chaque requête. La figure 4.11a illustre la consommation d'énergie des trois SGBDs sur un jeu de 5 Go. Hyrise est plus efficiente que PostgreSQL et MonetDB à cause de la performance d'exécution obtenue grâce à son gestionnaire de stockage. Hyrise accède directement aux données compressées dans la mémoire principale et non sur les mémoires secondaires, ce qui accélère également le traitement de la requête. L'énergie consommée par les systèmes MonetDB et PostgreSQL pendant l'exécution des requêtes sur un jeu de 30 Go et 100 Go sont illustrés sur les figures 4.11b et 4.11c. La consommation d'énergie du système MonetDB est optimale que celle de PostgreSQL. La forte consommation de PostgreSQL peut être expliquée par son modèle de stockage (orienté-ligne). MonetDB utilise des techniques de compression pour réduire le coût d'accès aux données. Cela affecte directement les performances des requêtes en raison de la diminution des demandes d'E/S et des erreurs de défauts de page. Pour comprendre ce qui cause la forte consommation de PostgreSQL, nous avons calculé puis comparé les défauts de cache du dernier niveau de cache (Last Level Cache - LLC) et les défauts de cache de niveau 2 (L2) des trois systèmes en utilisant les trois premières requêtes du benchmark TPC-H sur le jeu de données de 5 Go. La figure 4.12 représente le nombre de défauts de cache obtenu pour les



(a) SF=5Go (b) SF=30Go (c) SF=100Go

FIGURE 4.11 – Consommation d'énergie moyenne des requêtes $Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}$ du benchmark TPC-H.

différents systèmes. Comme illustré sur la figure, le nombre de défauts de cache du système PostgreSQL dépasse considérablement ceux obtenus pour les systèmes MonetDB et Hyrise.

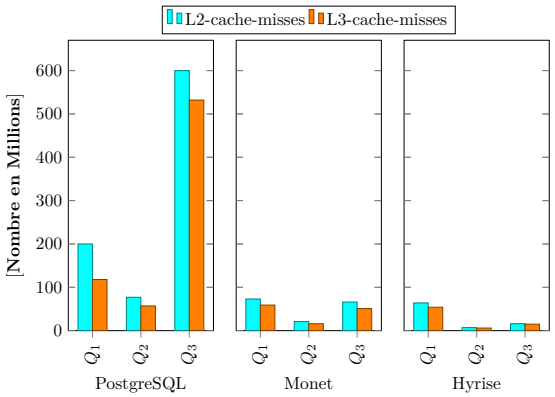


FIGURE 4.12 – L3 et L2 défauts de cache comptés dans Q_1, Q_2, Q_3 du TPC-H.

4.5.3 Identification des paramètres du modèle énergétique

Un paramètre est dit pertinent lorsqu'il exerce une influence sur la consommation énergétique du système. En fait, dans cette section nous essayons à l'aide d'une série d'expérimentation d'évaluer la pertinence de certains paramètres sur la consommation énergétique des composants lors de l'exécution des requêtes. Lors cette expérimentation, plusieurs requêtes SPJ complexes et simples sont exécutées en mode séquentiel et parallèle en variant le degré de parallélisme et la taille des données.

4.5.3.1 Stratégie d'exécution d'une requête

L'objectif des STRs consiste à trouver une stratégie d'exécution la plus proche de l'optimum. La stratégie d'exécution d'une requête est donnée schématiquement par une structure arborescente dans laquelle chaque nœud correspond à un opérateur de l'algèbre relationnel et les arcs aux flux de dépendance avec les nœuds adjacents et descendants. Le moteur d'exécution utilise alors cette stratégie formulée par l'optimiseur pour extraire les données correspondantes et les envoyer vers le client.

PostgreSQL utilise une stratégie d'exécution basée sur une approche en *pipelinage (pipeline)*. Cette approche consiste à ne pas écrire aussi que possible les structures de données ainsi que les tuples produits par un opérateur intermédiaire dans des fichiers temporaires mais à les consommer immédiatement dans un opérateur adjacent. Étant donné un plan d'exécution, il est nécessaire de le segmenter en un ensemble de pipeline délimité par des opérateurs bloquants. Un opérateur est dit bloquant s'il ne peut produire aucun tuple sans consommer toutes ses entrées (exemple de l'opérateur de tri).

TABLEAU 4.2 – Plan d'exécution généré par EXPLAIN (PostgreSQL)

	Query Plan
1	Finalize Aggregate (cost=365007.57..365007.58 rows=1 width=8)
2	− > Gather (cost=365007.36..365007.57 rows=2 width=8)
3	Workers Planned:3
4	− > Partial Aggregate (cost=364007.36..364007.37 rows=1 width=0)
5	− > Parallel seq scan on customers (cost=0.00..356412.67 rows=3037876 width=0)
6	− > Filter: (Ville = '#London'::bpchar)

À cela s'ajoute le mode parallèle plus précisément le parallélisme intra-opération qu'offre le système PostgreSQL. Ce mode parallélise si possible l'exécution de chaque opération dans la requête. Le processus scinde les données en des plus petites quantités à l'aide d'une technique de la Fragmentation Horizontale (FH) au niveau physique puis parallélise l'exécution de la même opération en utilisant les unités de traitement (cœurs) disponibles. Ensuite, les résultats partiaux issus de chaque nœud de traitement (cœurs) sont combinés en un résultat final ou partiel (pour une autre opération). La figure 4.13 illustre le plan parallélisé de la requête `select sum(sales) from customers where customers.city='London';` dont le plan d'exécution est donné dans le tableau 4.2. Le dégrée de parallélisme (DdP) étant fixé 3, le moteur utilise trois workers ($CPU - coeurs$) pour lire les tuples de la table $CUSTOMER$ (*Parallel Seq Scan*), puis le nœud de fusion *Gather merge* [12] collecte les résultats partiaux de Chaque nœud.

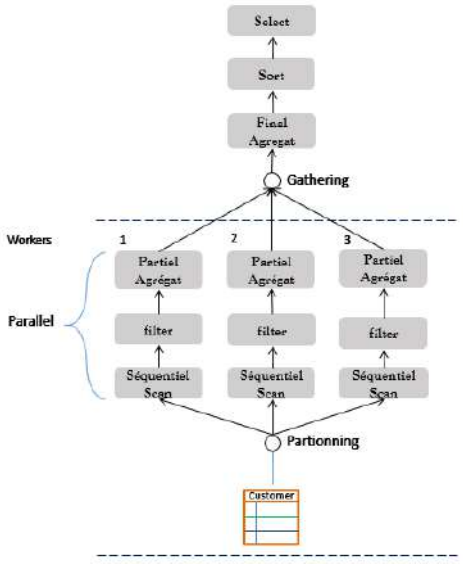


FIGURE 4.13 – Représentation du plan d'exécution en mode parallélisée

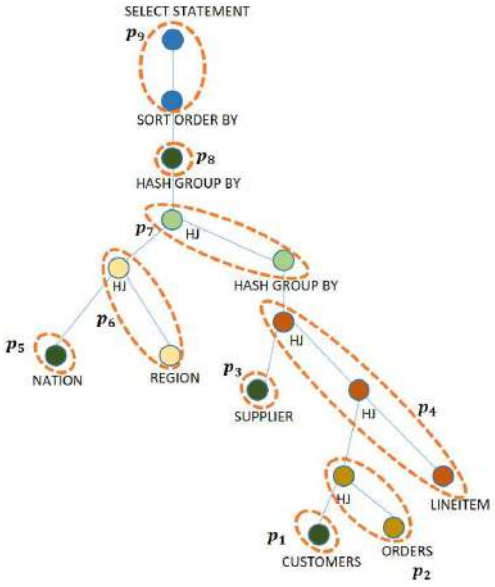


FIGURE 4.14 – La segmentation du plan d'exécution en pipelines

Pour déterminer le niveau de modélisation de la consommation d'énergie, nous analysons le comportement de la consommation énergétique de certaines requêtes du benchmark TPC-H pour comprendre comment la puissance énergétique évolue au fur de l'exécution des opérations de la requête en se basant sur la stratégie d'exécution adoptée par le système. Prenons par exemple la requête Q_5 du benchmark TPC-H donnée sur la figure 4.16. La figure 4.14 illustre la segmentation du plan d'exécution de la Q_5 en neuf (9) pipelines. La répartition de la consommation d'énergie au fur de l'exécution de la requête Q_5 suivant les différents pipelines est donnée sur la figure 4.15. Les opérations sont ordonnées sur la base des informations du module de surveillance SQL temps réel intégré dans les SGBD [13]. La forte consommation énergétique (pic) se fait au début de l'exécution de la requête dans le premier pipeline(P_1). Les pipelines P_5, P_6, P_7, P_8 prennent peu de temps pour terminer leur exécution. Le pic énergétique au niveau du pipeline P_1 s'explique par l'initialisation du parallèlisme (initialisation et lancement des workers). Nous remarquons qu'au cours de l'exécution d'un pipeline, sa consommation énergétique a tendance à être

12. Nœud de rassemblement des données pour les opérations parallélisées
13. `https://www.oracle.com/technetwork/database/manageability/owp-sql-monitoring-128746.pdf`

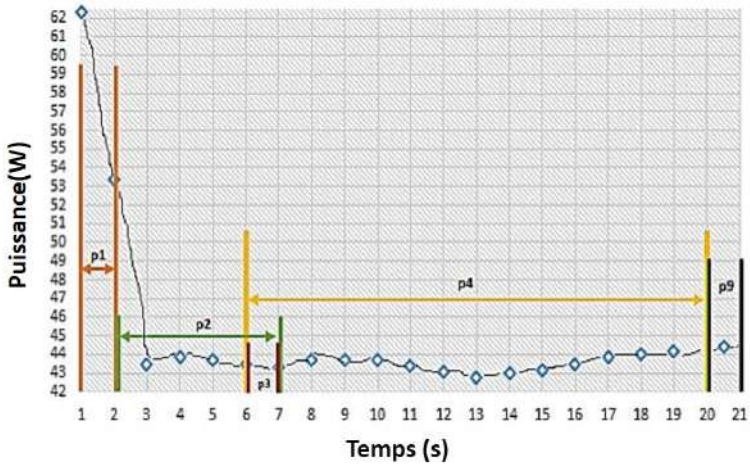


FIGURE 4.15 – Séquençage des pipelines et répartition de la consommation d'énergie au fur de l'exécution de la requête Q5 du TPC-H benchmark

```
1  SELECT n_name, sum(l_extendedprice * (1 - l_discount)) as revenue
2  FROM customer, orders, lineitem, supplier, nation, region
3  WHERE  c_custkey = o_custkey AND l_orderkey = o_orderkey
4  AND l_suppkey = s_suppkey AND c_nationkey = s_nationkey
5  AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
6  AND r_name = 'ASIA' AND o_orderdate >= date '1994-01-01'
7  AND o_orderdate < date '1994-01-01' + interval '1' year
8  GROUP BY n_name ORDER BY revenue desc;
```

FIGURE 4.16 – Requête Q_5 du bechmark TPC-H

stable sauf dans le premier et deuxième pipeline qui s'explique par la mise en place du parallèlisme. Lorsqu'un pipeline possède une opération parallélisée, sa consommation énergétique tend à se stabiliser à la fin de son traitement car le début de traitement est impacté par les initialisations des workers ou par la collecte des résultats partiaux (Exemple: P_2 sur la figure 4.15). Par conséquent les conclusions faites dans les travaux de [157] concordent bien avec un traitement de la requête en mode isolé séquentiel mais ne se généralise pas au traitement en mode parallélisé intra-opération. Pour définir un modèle plus précis, le niveau de modélisation doit tenir compte de cet aspect du mode parallèle. Pour PostgreSQL, nous utilisons un niveau de modélisation basé sur le pipeline et pour les systèmes MonetDB et Hyrise, nous adoptons une approche basée sur les opérateurs.

4.5.3.2 Effet du mode d'exécution

Les moteurs des BDs peuvent traiter les requêtes en mode séquentiel où une seule requête s'exécute de manière séquentielle sur la BD, ou en mode parallèle où une seule requête s'exécute en mode parallèle (intra-requête), ou en mode batch où plusieurs requêtes s'exécutent simultanément (parallélisme inter-requête). Nous étudions à l'aide des requêtes $Q_1 - Q_{13}$ du benchmark du SSB l'effet du mode d'exécution (séquentiel et parallèle) sur la consommation énergétique du système PostgreSQL lors de l'exécution des requêtes sur un jeu de données de taille de 50 Go. Puis nous faisons de même sur MonetDB avec les 10 premières requêtes du benchmark TPC-H sur un jeu de 10Go. Pour le mode d'exécution en mode parallèle, nous varions le DdP de 2 à 4. Lors de l'exécution de chacune des 13 et 10 requêtes de Sélection-Projection-Jointure (SPJ), nous mesurons à l'aide d'un ampèremètre l'énergie totale consommée. À partir de ces valeurs énergétiques, nous calculons la puissance minimale, la puissance moyenne et la puissance pic pour chacune des requêtes. Les figures 4.17 et 4.18 représentent respectivement la variation de la consommation énergétique du système suivant les deux modes d'exécution sur les systèmes PostgreSQL et MonetDB.

L'analyse des résultats représentés sur la figure 4.17, montre que la puissance moyenne des requêtes en mode séquentiel est optimale que celui du mode parallèle. Puis en comparant la puissance pic et la puissance minimale de chaque mode, le mode séquentiel tend à consommer moins d'énergie comparé

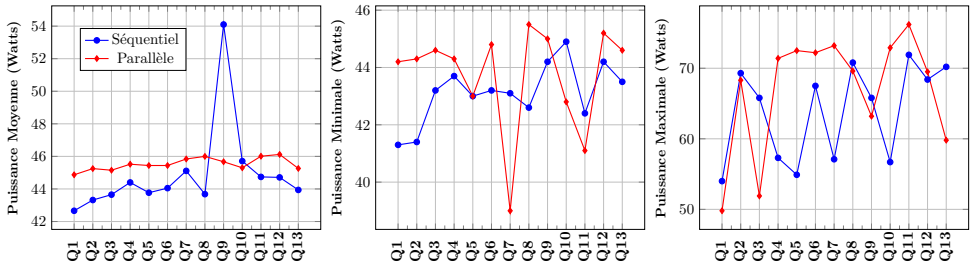


FIGURE 4.17 – la variation de la consommation énergétique du système suivant les deux modes d'exécution sur 10 requêtes (PostgreSQL)

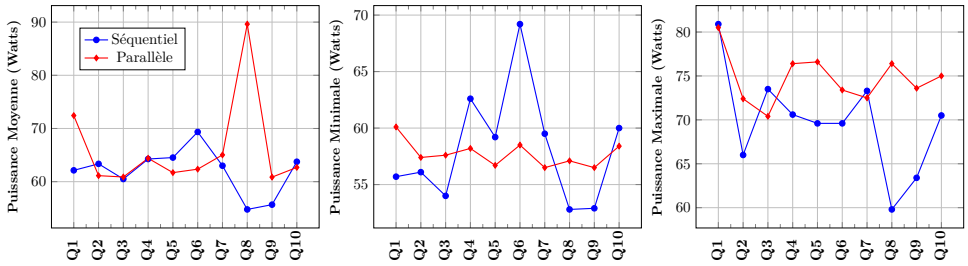


FIGURE 4.18 – la variation de la consommation énergétique du système suivant les deux modes d'exécution sur 10 requêtes (MonetDB)

au mode parallélisé également. Cela s'explique par le fait qu'en mode parallèle beaucoup ressources du système sont mobilisées pour exécuter la requête par contre l'énergie totale consommée pour traiter les requêtes en mode parallèle est plus efficiente (figure 4.19).

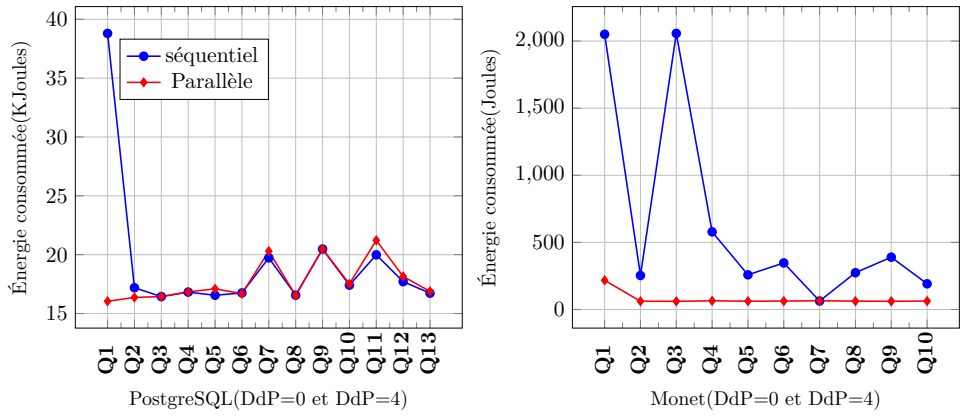


FIGURE 4.19 – Énergie totale consommée lors de l'exécution de chaque requête dans les deux modes de traitement

L'augmentation du degré de parallélisme implique l'invocation de plus de ressources. Ceci affecte par conséquent la consommation totale du système pendant l'exécution des requêtes. Ainsi le mode d'exécution de la requête impacte la consommation énergétique du système et par conséquent, cet aspect doit être inclus dans la modélisation énergétique.

4.5.3.3 Effet de la taille du BD

Pour évaluer l'effet de la taille du BD avec la variation du degré de parallélisme(DdP), nous menons une série d'expérimentation à l'aide des 10 premières requêtes du benchmark TPC-H sur trois jeux de données de taille 10Go, 50Go, 100Go. Sur chaque jeu de données, les requêtes sont exécutées en variant le DdP de 2 à 4. L'énergie totale consommée sur chacun des jeux de données est représentée sur la figure 4.20. Nous remarquons que les consommations énergétiques croissent lorsque nous augmentons la taille de la BD aussi bien pour le DdP fixé à 2 qu'à celui fixé à 4 sur le système PostgreSQL.

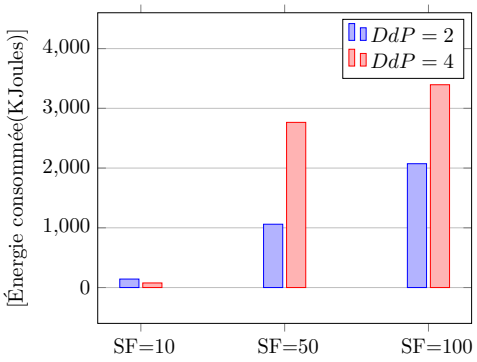


FIGURE 4.20 – Consommation totale du système lors de l'exécution des requêtes

En conclusion l'énergie totale consommée croit lorsque nous augmentons la taille de la BD. Malgré que le système soit plus chargé, il peut arriver qu'en augmentant le degré de parallélisme sur le même jeu de données, l'énergie totale consommée diminue car le système prend moins de temps pour traiter les requêtes (cas du SF=10 Go).

La sélection des paramètres pertinents pour un modèle énergétique n'est pas une tâche facile, l'estimation du coût d'exécution d'un opérateur dans les optimiseurs inclut les paramètres suivants: la cardinalité, la longueur moyenne des lignes et le nombre de partitions, la taille du bloc. Ces paramètres sont ensuite combinés pour créer d'autres paramètres dérivés afin de capturer l'utilisation des ressources physiques (CPU, Disk E/S, etc.) par les implémentations algorithmiques des opérateurs constituant le plan de la requête. Les ressources CPU, E/S mémoires, canaux de communication, etc. sont les principaux composants qui consomment l'énergie lors de l'évaluation des opérateurs. Ainsi le coût du CPU, le coût des E/S des mémoires (RAM, Caches) sont des paramètres pertinents pour la modélisation du coût énergétique. Nous donnerons plus de détails sur l'implémentation algorithmique des coûts d'E/S et de CPU de chaque opérateur dans l'annexe A.

4.6 Conception des McE

Dans cette section, après avoir évoqué nos hypothèses de modélisation nous construisons nos modèles de coût en combinant les différents paramètres identifiés. Nous commençons par donner un formalisme pour chacun de nos modèles sur lesquels nous appliquons ensuite une des techniques de l'apprentissage automatique pour déterminer les valeurs des constantes énergétiques. Dans notre étude, pour estimer les valeurs de ces paramètres nous avons utilsé trois techniques d'apprentissage présentée dans la section 4.6.3, et pour chacune d'elle nous menons des études expérimentales pour valider nos résultats en utilisant différents benchmarks.

4.6.1 Hypothèses

Les modèles exposés ci-dessous reposent sur un ensemble d'hypothèses que nous fixons afin d'atténuer la complexité du problème traité. Ces hypothèses nous permettent de définir les limites de notre environnement de travail. Elles sont énumérées dans les items suivants:

(i) Nous menons nos expérimentations dans un environnement centralisé c'est-à-dire que la BD est hébergée sur une seule machine. Dans un environnement centralisé, le coût de la communication n'est pas pris en compte.

(ii) La station de travail est une machine multi-cœur. Nous ne comptons pas le coût de la communication entre les cœurs lors du traitement de la requête en mode parallèle.

(iii) En plus du traitement inter-requêtes, nous considérons que le système permet le traitement d'une requête en mode parallèle (intra-requête). PostgreSQL fournit cette particularité à partir de sa version 9.6. Le traitement des requêtes en mode parallèle est activé par défaut depuis sa version 10. PostgreSQL permet la modification de la valeur du *DdP* en utilisant la commande *max_parallel_workers_per_gather*, sinon par défaut cette valeur est fixée à 2. MonetDB offre la même particularité, pour définir le *DdP* la variable *gdk_nr_threads* est utilisée.

(iv) Toutes les requêtes utilisées dans cette étude sont de la forme Sélection-Projection-Jointure (SPJ) ou de type d'interrogation des données pouvant être transactionnelles ou analytiques. Les requêtes de définition des données et de manipulation des données ne sont pas prises en compte dans la conception de nos modèles.

(v) Les requêtes sont exécutées de façon isolées en mode séquentiel ou parallèle(intra-requête).

4.6.2 Formulation

Nous présentons dans cette section une variété de formalisme de modèles proposés indépendamment des études expérimentations. Pour chaque système SGBD, nous proposons un modèle qui résulte de la combinaison des ressources physiques invoquées (CPU, Mémoire, Disque) lors du traitement de la requête. Pour le système PostgreSQL, nous proposons deux modèles: le premier proposé en l'extension des travaux de [157] et le second sur la base de l'utilisation de nouveaux paramètres énergétiques. Pour faciliter la lecture de la section, nous classifions nos modèles dans deux catégories sur la base de leur niveau de modélisation à savoir: *niveau pipeline* et *niveau opérateur*.

4.6.2.1 Modélisation des pipelines (PostgreSQL)

Le moteur d'exécution du STR prend le plan créé par l'optimiseur et le traite de façon récursive pour récupérer l'ensemble des tuples requis dans un mécanisme de pipeline. La délimitation des pipelines constituant le plan se fait à la suite des opérateurs dits bloquants [24]. Le pipeline se compose d'un ensemble d'opérateurs élémentaires qui mobilisent les ressources du système (CPU, Mémoire, Communication, etc.) pour réaliser ces tâches. Le coût des opérateurs composant le pipeline détermine le «coût du pipeline», qui est à la base de la puissance dynamique dissipée par le système.

Lors du traitement des opérateurs, (1) le processeur exécute un ensemble d'instruction fortement lié à l'implémentation de l'opérateur et à la cardinalité des données en entrées, ce paramètre est appelé *«coût du CPU»* dénoté par C_{cpu} et (2) un certain nombre de pages sont chargées réciproquement de la mémoire secondaire (disque) vers la mémoire primaire (RAM) et de la mémoire RAM vers les mémoires caches du CPU, ce paramètre est appelé *«coût d'entrées-sorties (E/S) mémoire»* dénoté par $C_{E/S}$ pour le disque et C_{mem} pour la mémoire RAM. Ainsi pour une requête (Q) composée de K_0 pipelines noté $\{PL_1, PL_2, \dots PL_{k_0}\}$, la forme générique pour estimer le coût de la puissance dissipée est donnée dans l'équation suivante:

$$Puissance(Q) = \frac{\sum_{i=1}^{k} Puissance(PL_i) * Temps(PL_i)}{Temps(Q)} \tag{4.1}$$

où $Temps(Q)$, $Time(PL_i)$ représentent respectivement, le temps d'exécution de la requête Q et le temps d'exécution du pipeline PL_i.

Notre premier modèle proposé pour PostgreSQL part des travaux de [157], où nous avons étendu leur modèle en prenant en compte le mode parallèle introduit par PostgreSQL tout en ayant la capacité de prédire la puissance des requêtes exécutées en mode séquentiel. Pour ce faire, nous divisons les pipelines du plan de la requête Q en deux groupes de pipelines dénotés par: (1) pipeline séquentiel (PL) dont le DdP est fixé à 0 ($DdP = 0$) et (2) pipeline parallèle (PPL) avec un DdP fixé à DdP. Un pipeline dit séquentiel lorsqu'il ne possède pas d'opérations traitées en mode parallèle (parallélisme intra-opération). L'énergie du plan parallèle de la requête Q peut se formuler comme suite:

$$Energie(Q, DdP) = \sum_{i=1}^{n_0} Energie(PL_i, 1) + \sum_{i=1}^{m_0} Energie(PPL_i, DdP) \tag{4.2}$$

Où DdP définit le degré de parallélisme et $m_0 + n_0$ correspond au nombre de pipelines dans le plan de la requête ($m_0 + n_0 = k_0$).

La puissance dissipée lors du traitement du pipeline séquentiel est la combinaison de la puissance des ressources physiques identifiées. Le modèle que nous étendons de [157] utilise les paramètres CPU et les E/S disques. La puissance pour un pipeline séquentiel s'écrit donc par l'équation 4.3:

$$Puissance(PL_i, 1) = W_{CPU} * \sum_{k=1}^{n} C_{CPU_k} \oplus W_{E/S} * \sum_{k=1}^{n} C_{E/S_k} \tag{4.3}$$

où W_{CPU} et $W_{E/S}$ définissent les unités énergétiques pour une instruction du CPU et une opération de lecture ou d'écriture sur le disque respectivement. Le C_{CPU_k} est le nombre d'instructions exécutées par le CPU. C_{E/S_k} est le nombre d'accès au disque soit en lecture ou en écriture. La variable n définit le nombre d'opérateurs dans le pipeline i et k est l'indice de sommation.$\oplus$ exprime la corrélation entre les paramètres.

Pour estimer la puissance d'un pipeline parallèle, nous introduisons un paramètre nommé facteur énergétique dénoté par fc. Ce paramètre exprime la différence énergétique du CPU entre la puissance consommée lors du traitement d'une requête en mode parallèle par rapport au traitement en mode séquentiel. Il est estimé en fonction du DdP défini par l'utilisateur pour traiter la requête. La formule décrite dans l'équation 4.4 permet d'estimer ce paramètre. P_n exprime la puissance moyenne consommée par les n cœurs du CPU pour traiter le plan parallèle de la requête ($DdP = n$) et P_0 la puissance moyenne consommée par un seul cœur pour traiter le plan en mode séquentiel de la même requête ($DdP = 0$).

$$fc_n = \frac{P_n - P_0}{P_0} \tag{4.4}$$

La puissance dissipée lors du traitement d'un pipeline parallèle est donnée par l'équation 4.5 en introduisant le facteur énergétique dans le formalisme.

$$Puissance(PPL_i, DoP) = (fc_{DoP} + 1) \times W_{CPU} \times \sum_{k=1}^{n} C_{CPU_k} \oplus W_{E/S} \times \sum_{k=1}^{n} C_{E/S_k} \tag{4.5}$$

L'optimiseur utilise les données statistiques sur les relations pour estimer les coûts CPU et E/S des opérateurs puis convertis ces coûts en temps d'exécution sur la base des paramètres du système tel que la vitesse du processeur et la vitesse du transfert des données. Nos modèles se basent sur la même approche en ayant les valeurs des coûts CPU et E/S à partir du système, nous appliquons les techniques de l'apprentissage automatique pour déduire les valeurs des unités énergétiques. Comme pour l'énergie, nous utiliserons la même approche pour estimer le temps des pipelines séquentiel et parallèles. En mode séquentiel, le temps d'exécution est obtenu par la somme du temps de chacun des pipelines. Plus formellement, nous donnons dans l'équation 4.6 la formule qui calcule le temps d'exécution d'un plan parallèle avec n pipelines séquentiels et m pipelines parallèles. Dans la suite de ce travail, nous nous concentrons plus sur l'estimation de la puissance moyenne, nombreux sont les travaux dans la littérature qui traitent la problématique de la performance.

$$Temps(Q, DdP) = \sum_{i=1}^{n} Temps(PL_i, 1) + \frac{1}{DdP} * \sum_{j=1}^{m} Temps(PPL_j, 1) \tag{4.6}$$

En effet sur les architectures modernes la capacité de la mémoire centrale est de plus en plus conséquente, les systèmes SGBDs utilise une partie de cette mémoire connue sous le nom de «buffer ou mémoire cache» pour stocker les données sous forme de pages et de pouvoir les réutiliser pour d'autres requêtes. La mémoire cache en tant que pont entre le CPU et les E/S disques, a une influence significative sur la performance et la consommation énergétique des requêtes. Elle a également un impact direct sur la consommation des ressources du CPU et du disque. Afin de vérifier l'impact de la mémoire cache sur les performances et la consommation d'énergétique, en partant de la configuration décrite dans la section 4.5.1, nous exécutons les sept(7) premières requêtes du benchmark TPC-H alors que la mémoire cache est vide (*Scénario*1) puis nous les ré-exécutons en présence des données dans la mémoire cache (*Scénario*2). La figure 4.21 illustre les temps d'exécution et les consommations énergétiques de sept (7) requêtes dans les deux scénarios. Nous constatons que les temps de réponse et les consommations énergétiques dans le *Scénario*1 sont supérieurs à ceux du *Scénario*2.

Partant de cette conclusion, nous proposons pour PostgreSQL un deuxième modèle en considérant la source de chargement des données lors du traitement de la requête dans le but d'améliorer l'exactitude de la précision du modèle. La puissance dissipée lors du traitement de la requête devient donc la combinaison

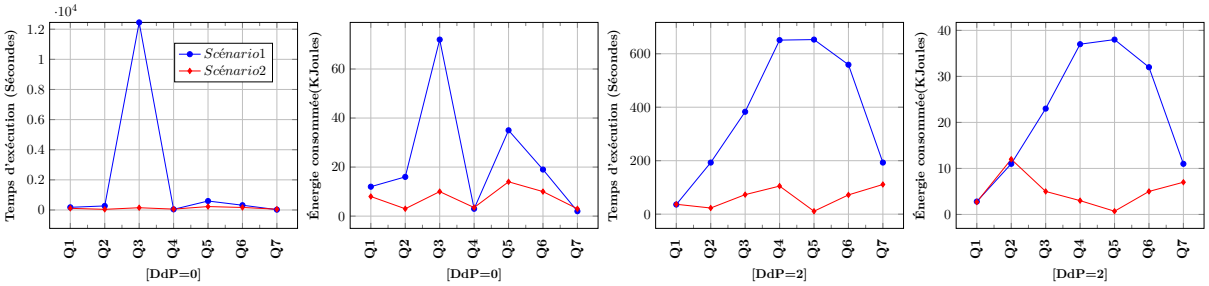


FIGURE 4.21 – Comparaison du temps d'exécution et consommation énergétique des requêtes avec/et sans données dans la mémoire cache.

de la puissance des ressources du CPU, de la mémoire principale et du disque. Elle s'écrit de manière formelle comme :

$$Puissance(PL_i) = W_{CPU} * \sum_{k=1}^{n} C_{CPU_k} \oplus W_{mem} * \sum_{k=1}^{n} C_{mem_k} \oplus W_{E/S} * \sum_{k=1}^{n} C_{E/S_u} \tag{4.7}$$

où W_{CPU}, W_{mem} et $W_{E/S}$ définissent les unités énergétiques pour une instruction du CPU, une opération de lecture ou d'écriture sur disque et une opération de lecture ou d'écriture en mémoire cache respectivement. Le C_{CPU_k} est le nombre d'instructions exécutées par le CPU. C_{E/S_k} est le nombre d'accès au disque soit en lecture ou en écriture. C_{mem_u} est le nombre d'accès à la mémoire principale soit en lecture ou en écriture. La variable n définit le nombre d'opérateurs dans le pipeline i et k est l'indice de sommation.$\oplus$ exprime la corrélation entre les paramètres.

4.6.2.2 Modélisation des opérateurs (MonetDB & Hyrise)

Pour MonetDB nous adoptons une approche de modélisation basée sur les opérateurs. Le plan étant constitué d'un ensemble d'opérateur, pour chaque opérateur il s'agit de combiner les principaux facteurs responsables de la consommation énergétique lors du traitement de la requête. Ainsi étant donné une requête Q constitué de k_0 opérateurs dans le plan noté $\{OP_1, OP_2, OP_3, \ldots OP_{k_0}\}$, l'énergie moyenne consommée par Q est décrit par l'équation 4.8.

$$Puissance(Q) = \frac{\sum_{j=1}^{k_0} Puissance(OP_j) * Temps(OP_j)}{Temps(Q)} \tag{4.8}$$

où $Temps(Q)$, $Temps(OP_j)$ représentent respectivement le temps d'exécution de la requête Q et le temps d'exécution de l'opérateur OP_j. La puissance dissipée lors du traitement d'un opérateur est la combinaison de la puissance consommée par les principales ressources identifiées. La formule est donnée par l'équation 4.9:

$$Puissance(OP_j) = W_{CPU} * C_{CPU_j} \oplus W_{mem} * C_{mem_j} \oplus W_{E/S} * C_{E/S_j} \tag{4.9}$$

Avec la baisse du prix des barrettes mémoires et l'augmentation de leurs capacités, la taille de la mémoire principale est de plus en plus importante. Cette caractéristique a conduit à l'avènement des systèmes dans lequel le traitement de la requête se fait totalement à partir de la mémoire centrale afin d'éviter les accès coûteux au disque. Dans les configurations modernes des CPU, pour diminuer la latence de chargement des données de la mémoire centrale vers les registres, de nouvelles mémoires caches dynamiques ont été introduite généralement organisées en trois niveaux $L1$, $L2$, $L3$ comme illustré sur la figure 2.15. Sur ces systèmes, le CPU commence par chercher la page sur la mémoire cache la plus proche (L1D cache). Si la page est trouvée dans L1D alors elle est chargée vers le registre sinon c'est un défaut de cache (L1D cache misses) et le CPU va chercher les données dans le cache de niveau suivant (L2D cache). À ce niveau aussi, il y a deux cas: soit la page est trouvée et elle est chargée vers la cache L1D ou c'est défaut de cache (L2D cache misses) et L3D cache commence. Le processus se répète ainsi jusqu'à retrouver la page recherchée. Le coût d'accès aux différentes mémoires caches n'est pas uniformes, il dépend du niveau de la mémoire cache dans laquelle la donnée a été trouvée [111]. En conséquence sur ces architectures, la mémoire principale (RAM) et les mémoires caches (L1D, L2D, L3D) sont des

paramètres pertinents pour modéliser la consommation énergétique.

Pour le système Hyrise, nous adoptons une approche de modélisation basée sur les opérateurs. Dans ce système, il n'y a pas de coût E/S sur disque par contre nous modélisons les E/S de la mémoire principale (RAM) et les mémoires caches en estimant le nombre de défauts cache à chaque niveau. Hyrise invoque un opérateur «GetTable» au début de l'exécution de la requête. Ce dernier est responsable de charger le nom de la table et les blocs de données de la mémoire principale vers les caches du CPU. Ainsi, pour modéliser la consommation d'énergie d'un opérateur dans le plan, nous utilisons la combinaison de la consommation d'énergie des principales ressources identifiées (RAM, L1D, L2D, L3D). Pour une requête donnée (Q) constitué de k_0 opérations notées $\{OP_1, OP_2, OP_3, \dots OP_{k_0}\}$, sa puissance est estimée comme suit:

$$Puissance(Q) = \frac{\sum_{j=1}^{k} Puissance(OP_j) * Temps(OP_j)}{Temps(Q)} \tag{4.10}$$

où $Temps(Q)$, $Temps(OP_j)$ représentent respectivement le temps d'exécution de la requête Q et le temps d'exécution de l'opérateur OP_j. La formule de la puissance associée à un opérateur est donnée par l'équation 4.11:

$$Puissance(OP_j) = W_{CPU} * C_{CPU_j} \oplus W_{L2D} * C_{L2D_j} \oplus W_{L3D} * C_{L3D_j} \oplus W_{mem} * C_{mem_j} \tag{4.11}$$

où W_{L2D} , W_{L3D}, W_{mem} et W_{CPU} sont les paramètres du modèle. W_{L2D} et W_{L3D} définissent les unités énergétiques pour un défaut de cache respectivement au niveau de L2D et L3D. C_{L2D_j} définit le nombre de défauts de cache cumulé au niveau de $L1D$, C_{L3D_j} est le nombre de défaut de cache cumulé au niveau de $L2D$ et C_{mem_j} le nombre de défaut de cache cumulé au niveau de $L3D$.

4.6.3 Machine learning

Dans le contexte de la modélisation énergétique, les modèles d'apprentissage automatique sont largement utilisés pour la prédiction ou l'estimation de la consommation énergétique des systèmes. Ils permettent d'estimer de façon automatique à partir des techniques d'exploration des données les paramètres du modèle. Le concept de l'apprentissage automatique regroupe plusieurs types de modèle qui peuvent être régressifs, cognitifs, analogiques, etc. Dans cette sous-section, les techniques régressives et cognitives sont décrites.

4.6.3.1 Les techniques régressives

Les modèles de régression sont une approche statistique de modélisation des relations entre différentes variables quantitatives (dépendantes et indépendantes) afin qu'une variable dite *expliquée* puisse être prédite à partir des autres [120]. Nous présentons très brièvement les notions de bases sur les régressions linéaires et non linéaires dans cette section.

(i) **Régression linéaire :**
La régression linéaire (simple ou multiple) chercher à expliquer une variable Y à partir d'une variable X (ou plusieurs variables $X_1 \dots, X_n$ dans le cas de la régression multiple). La variable Y est appelée variable dépendante, ou variable à expliquer et la variable X (ou les variables $X_1, \dots, X_n$) est(sont) appelée(s) variable(s) indépendante(s), ou variable(s) explicative(s). Le modèle de la régression linéaire (voir figure 4.22) exprime une équation censée représenter la corrélation entre les variables. Il s'écrit sous la forme: $Y = f(X) + \epsilon$, *plus précisément:*

$$\begin{cases} \boldsymbol{y_i = \beta_0 + \beta_1 \times x_i + \epsilon_i}, & \textit{Pour les modèles linéaires simples} \\ \boldsymbol{y_i = \beta_0 + \beta_1 \times x_{i,1} + \dots + \beta_k \times x_{i,k} + \epsilon_i}, & \textit{Pour les modèles linéaires multiples} \\ \text{avec}, & \boldsymbol{i \in \{1, 2, \dots, n\}} \end{cases}$$

Où $f(x)$ est une fonction sur les variables explicatives, le terme ϵ_i représente la perturbation, ou la marge d'erreur ou l'imprécision du modèle. $\beta_0, \beta_1, \dots, \beta_k$ sont les paramètres/coefficients du modèle que l'on veut estimer.

— **Estimation des paramètres par le critère des moindres carrés:** Nous considérons que la nature de la liaison entre les deux variables est linéaire simple afin d'illustrer le critère des moindres

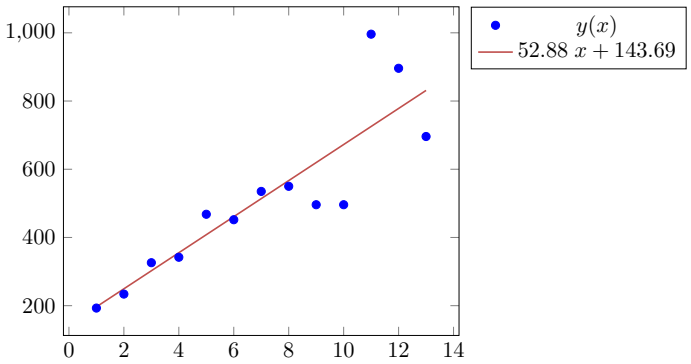


FIGURE 4.22 – Exemple graphique de la corrélation entre deux variables

carrés pour estimer les valeurs des paramètres. L'estimation des paramètres est obtenue par la minimisation de la somme des carrés des écarts entre les observations et les estimations du modèle. Pour chaque cas d'observations (x_i, y_i) des n points d'observations, la méthode des moindres carrés considère l'écart de y_i par rapport à sa valeur attendue. Cet écart est calculé par: $\epsilon_i = [y_i - (\beta_0 + \beta_1 \times x_i)]$. Ces termes ϵ_i sont appelés aussi résidus. Le critère des moindres carrés consiste à déterminer les valeurs des coefficients β_0 et β_1 de tels sorte que la somme des carrés des résidus soit le plus proche de la valeur 0. Ce critère s'écrit:

$$\sum_{i=1}^{n} \epsilon_i^2 = \sum_{i=1}^{n} [y_i - (\beta_0 + \beta_1 \times x_i)]^2$$

Les estimateurs des coefficients β_0 et β_1 noté b_0 et b_1 respectivement, qui minimisent le critère des moindres carrés pour les observations $(x_1, y_1), (x_1, y_2), \ldots, (x_n, y_n)$ sont donnés par les formules suivantes [94].

$$b_0 = \frac{1}{n}(\sum_{i=1}^{n} y_i - b_1 \sum_{i=1}^{n} x_i) = \bar{Y} - b_1 \bar{X} \ \text{ et } \ b_1 = \frac{\sum_1^n (x_i - \bar{X})(y_i - \bar{Y})}{\sum_1^n (x_i - \bar{X})^2}$$

avec: $\bar{X} = \frac{1}{n}\sum_1^n x_i$ et $\bar{Y} = \frac{1}{n}\sum_1^n y_i$. En plus de l'estimation des coefficients β_0, β_1, les estimations d'intervalles de confiance de ces paramètres sont déduites. Ces intervalles de confiance permettent de mesurer la qualité globale de la droite de régression.

(ii) **Régression polynomiale (NLR):**
La régression polynomiale est une forme d'analyse dans laquelle la relation entre la variable indépendante Xi et la variable dépendante Yi est modélisée comme un polynôme de degré k. La méthode consiste à élever la variable explicative Xi en différents degré de puissance dans l'équation de la régression. Le modèle de régression polynomiale à une variable peut être exprimé comme:

$$\mathbf{y_i} = \beta_\mathbf{0} + \beta_\mathbf{1} \times \mathbf{x_i} + \beta_\mathbf{2} \times \mathbf{x_i^2} + \cdots + \beta_\mathbf{k} \times \mathbf{x_i^k} + \epsilon_\mathbf{i}$$

Où k est le degré du polynôme (appelé aussi ordre) et ϵ_i représente les résidus. Les modèles polynomiaux sont des cas particuliers de la régression linéaire. Ils peuvent être écrit sous la forme de régression linéaire multiple avec k variables de régression:

$$y_i = \beta_0 + \beta_1 \times x_1 + \beta_2 \times x_2 + \cdots + \beta_k \times x_k + \epsilon_i$$

avec $x_i = x_i^i$. L'équation peut être exprimée sous forme matricielle comme pour n observations:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \ldots & x_{1k} \\ x_{21} & x_{22} & \ldots & x_{2k} \\ \vdots & \ldots & & \ldots \\ y_{n1} & x_{n2} & \ldots & x_{nk} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

Où y_i est le vecteur d'observation, x_{ii} est le tableau vectoriel ou les variables, β_i est le vecteur des paramètres, ϵ_i est le vecteur d'erreur.

— **Estimation des paramètres:** Les paramètres de la régression polynomiale peuvent être estimés en utilisant la méthode des moindres carrés. La méthode des moindres carrés vise à minimiser la variance entre les valeurs estimées à partir du polynôme et les valeurs attendues de l'ensemble des données. En admettant que l'équation de régression ajustée est:

$$\hat{y_i} = \hat{\beta_0} + \hat{\beta_1} \times x_i + \hat{\beta_2} \times x_i^2 + \cdots + \hat{\beta_k} \times x_i^k$$

La moyenne des carrés des erreurs (MCE) est la moyenne arithmétique des carrés des écarts entre les prévisions du modèle et les observations. Elle définit par l'équation :

$$MCE = \frac{\sum_1^n (y_i - \hat{y_i})^2}{n - (k+1)}$$

où y_i est la valeur observée et $\hat{y_i}$ est la valeur ajustée de la variable dépendante pour le $i_{ème}$ cas. MCE est une mesure de la qualité de l'ajustement de la régression aux données. La racine de l'erreur quadratique moyenne ($RMSE$) définit par la racine carrée de MCE ($RMSE = \sqrt{MCE}$) est un estimateur non biaisé de l'écart type σ. MCE et $RMSE$ sont des indicateurs sur les erreurs d'estimation de la régression mais ne donnent pas d'indication sur la contribution de chaque composant d'erreur séparément [130]. Le coefficient de détermination, noté R^2 mesure l'adéquation du modèle de régression aux données d'observation. Il est formulé comme:

$$R^2 = 1 - \frac{\sum_1^n (y_i - \hat{y_i})^2}{\sum_1^n (y_i - \bar{y})^2}$$

où y est la moyenne arithmétique de la variable Y. La valeur de R^2 est toujours comprise entre 0 et 1 ($0 \leq R^2 \leq 1$). Une valeur de R^2 supérieure ou égale à $0,9$ est très bonne, une valeur supérieure à $0,8$ est bonne, et une valeur de $0,6$ ou plus peut être satisfaisante dans certaines applications, bien que nous devions être conscients du fait que, dans de tels cas, les erreurs de prédiction peuvent être relativement élevées. Le R^2 ajusté est une version modifiée du R^2, il est ajusté pour tenir compte du nombre de prédicteurs dans le modèle. Il est calculé par[130]:

$$R^2_{Ajusté} = R^2 - \frac{(1 - R^2) \times k}{n - (k+1)}$$

4.6.3.2 Les techniques cognitives

Le modèle cognitif englobe tous les modèles de cognition, allant d'aspects cognitifs très spécifiques et isolés, applicables uniquement dans des situations spécifiques, à des modèles plus complets et généralisables. Les modèles cognitifs peuvent faire des prévisions sur la façon dont de multiples aspects ou variables interagissent et produisent le comportement observé dans les études empiriques [142]. Nous présentons dans cette section les modèles des réseaux de neurones et des forêts aléatoires

(i) **Les réseaux de neurones artificiels (RNA):**
Un réseau de neurone (*Artificial Neural Network :ANN*) est un paradigme de traitement d'information qui s'inspire de la façon dont le système nerveux règle le fonctionnement du corps humain, plus précisément sur la manière dont le cerveau traite l'information. L'élément clé de ce paradigme est la nouvelle structure du système de traitement de l'information. Il est composé d'un grand nombre d'éléments de traitement (neurones) hautement interconnectés et pondérés travaillant en complicité pour résoudre un problème spécifique. Les interconnexions pondérées représentent la force des connections synaptiques reliant les neurones entre eux. Les réseaux de neurones acquièrent des connaissances à partir des exemples (expériences ou vécus), puis sur la base de cet apprentissage peuvent prendre des décisions ou faire des prédictions sur de nouvelles données [176][127]. Les RNA connaissent une vive progression au sein des domaines touchant la reconnaissance vocale, l'analyse d'image, le contrôle adaptatif, etc.

Le type le plus classique des réseaux de neurones est composé de trois couches: d'une couche d'entrée, d'une ou plusieurs couche(s) intermédiaires dites cachées et d'une couche de sortie comme illustré dans la figure 4.23.

Un neurone peut possèder plusieurs entrées $x_1, \ldots, x_m$. Chaque lien de connexion de l'entrée vers l'unité de traitement est affecté par un poids synaptique noté respectivement $w_1, w_2, \ldots, w_m$. Un signal x_j à l'entrée du synapse ou connection j, connecté au neurone k, est multiplié par le poids synaptique w_{kj}. Une fonction d'activation(f_a) est utilisée pour déterminer si le neurone doit être activé ou non. Elle est définie par la formule ci-dessous [113].

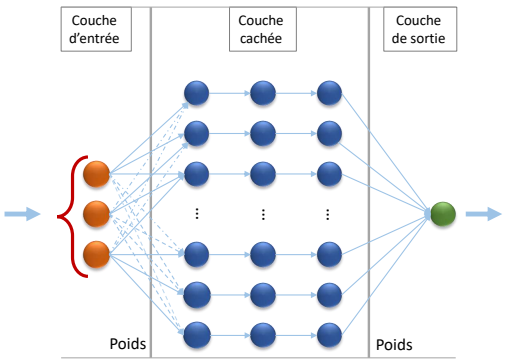


FIGURE 4.23 – Structure classique d'un réseau de neurone artificiel

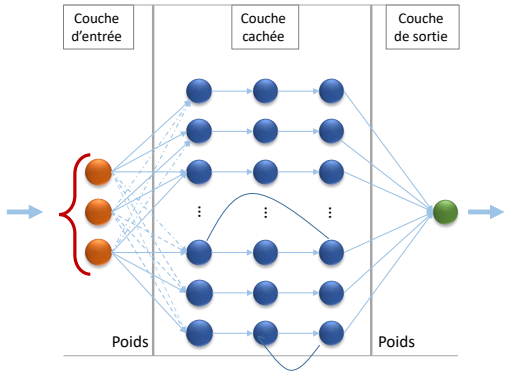


FIGURE 4.24 – Réseau de neurones récurrent

$$f_a(x) = a(\sum_{j=0}^{m} w_j x_j)$$

Où w_0 est considéré comme le biais de l'estimation et $x_0 = 1$. La figure 4.25 illustre l'anatomie d'un neurone artificiel.

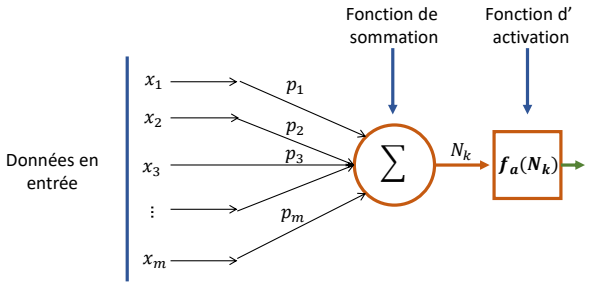


FIGURE 4.25 – Anatomie d'un neurone artificiel

La disposition des neurones dans un réseau connu aussi sous le nom de topologie neuronal permet de déduire le type d'architecture utilisé. Le type d'architecture utilisé dans les RNA est étroitement lié à l'algorithme d'apprentissage. Les architectures des RNA sont classées dans trois catégories: (*i*) les réseaux à propagation-avant monocouche, (*ii*) les réseaux à propagation-avant multicouche, et (*iii*) les réseaux récurrents [127].

Les RNA à propagation-avant (*Feed-Forward Neural Networks-FFNN*) sont abondamment utilisés dans de nombreuses applications pratiques car il fut le premier modèle conçu et le plus simple. Dans ce modèle, les flux d'information circulent dans une seule direction des nœuds d'entrée vers les nœuds de sortie sans créer de boucles (figure 4.23). Les réseaux de neurones récurrents (*Feed-Back Neural Network - FBNN*) sont des réseaux cycliques c'est-à-dire qui contiennent au moins une connexion de rétro-action (figure 4.24).

Parmi les principaux réseaux basés sur l'architectures multicouches à propagation-avant, on trouve le Perceptron multicouche (MLP) et la fonction de base radiale (RBF). Le perceptron multicouche est un modèle bien connue, sur lequel l'algorithme d'apprentissage Back-propagation [14] est implémenté. MLP est adapté pour résoudre des problèmes non linéaires et stochastiques.

Il existe deux classes de techniques d'apprentissage des réseaux de neurones: supervisée, non supervisée. Le mécanisme de l'apprentissage supervisé consiste à forcer le réseau à converger vers un état final précis en lui fournissant les résultats souhaités, et dans le mode non supervisé, le réseau apprend de lui-même en donnant un sens aux données de l'entrée [110]. L'algorithme de rétro-propagation(BP) est l'un des algorithmes d'apprentissage supervisés le plus utilisé dans les réseaux de neurones. L'algorithme BP est répété sur toutes les données en entrée (ou données d'apprentissage) jusqu'à ce que l'erreur de réseau

14. `https://en.wikipedia.org/wiki/Backpropagation`

converge vers un seuil minimum; cette erreur est appelée erreur quadratique moyenne (Root Mean-Squared Error - RMSE). Elle est calculée par la formule suivante [87][127]:

$$RMSE = \sum_{k=1}^{m} (R_k - E_k)^2$$

où $RMSE$ est l'erreur, R_k est la valeur réelle souhaitée, et E_k la valeur de sortie du réseau. Dans notre expérimentation, nous utilisons le perceptron multicouche sur lequel l'algorithme BP est implémenté pour apprendre nos modèles d'estimation de la consommation énergétique du système à partir des données collectées, les détails de nos paramétrages sont décrits dans les sections suivantes.

(ii) **Forêts aléatoires:**
Initialement introduit par *Leo Breiman* en 2001 [16], les forêts aléatoires (plus communément appelé Random Forest en anglais) est une combinaison (agrégation) d'arbres prédicteurs ou classifieur dans laquelle chaque arbre est construit sur un unique sous-ensemble de données avec la sélection d'une variable aléatoire. La méthode est dite CART (Classification And Regression Tree). Il s'agit d'un algorithme d'apprentissage non paramétrique utilisable à la fois en régression et en classification, cet algorithme a démontré des améliorations significatives en terme de précision de la prédiction et de la classification dans des problèmes complexes incluant les situations de non linéarité. La figure 4.26 illustre un exemple de la structure des Random Forest.

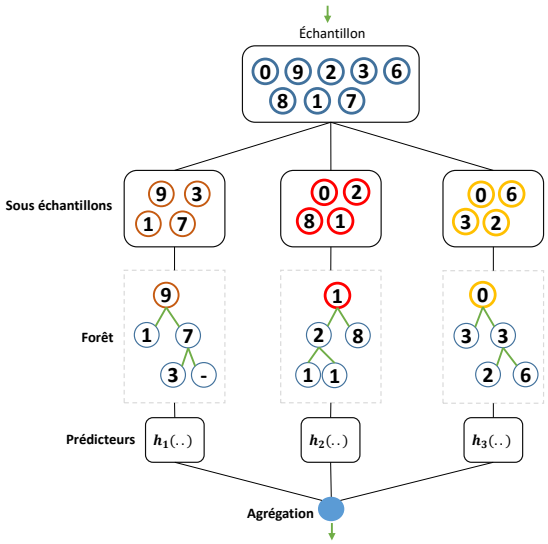


FIGURE 4.26 – Une illustration de la structure des Random Forest

L'objectif principal de notre étude est de prédire la consommation d'énergie d'une requête isolée, alors dans ce qui suit nous nous attarderons plus sur les forêts aléatoires de régression (FAR), pour plus de détails sur l'approche, veuillez vous référer aux travaux suivants [16] [106]. La forêt aléatoire de régression est une méthode de régression qui consiste en un ensemble de collection de prédicteurs d'arbres $\{h_1(X), h_2(X), \ldots, h_k(X)\}$, où la fonction h_i est le prédicteur construit sur X, X représente le vecteur d'entrée des variables observées de longueur n. La valeur finale prédite (soit y_i) pour une observation x_i est obtenue en calculant la moyenne des valeurs estimées par tous les prédicteurs de l'arbre [162]:

$$y_i = \frac{1}{k}(\sum_{j=1}^{k} h(x_i))$$

L'apprentissage des forêts aléatoires est réalisé via la technique générale du *bootstrap aggregating*, ou *bagging* [15]. Le mot Bagging est la contraction des mots Bootstrap et Aggregating. Généralement, on note trois paramètres dont les valeurs doivent être définies avant le début processus d'apprentissage du modèle à savoir:

15. `https://en.wikipedia.org/wiki/Bootstrap_aggregating`

(i) **ntree:** le nombre d'arbre dans la forêt,

(ii) **mtry:** le nombre de variables sélectionnées aléatoirement à chaque nœud,

(iii) **nodesize:** le nombre minimum d'observation en dessous duquel on ne subdivise plus un nœud.

Pour entraîner un modèle des forêts aléatoires, il faut au moins deux principales étapes, (i) la première étape consiste à créer A_n sous-échantillons de taille m en sélectionnant aléatoirement avec remise m observations parmi les données d'observation. Un processus de bootstrap est utilisé à cet effet. Certaines observations peuvent être présentes plusieurs fois dans les échantillons formés. (ii) la deuxième étape construit un arbre de décision pour chaque sous échantillon issue de l'étape précédente. À chaque nœud, on tire aléatoirement un nombre #m# de variables, et on cherche la meilleure coupure uniquement suivant les *mtry* variables sélectionnées parmi l'ensemble des variables disponibles. Ce hyper-paramètre du modèle est défini avant le début de l'entraînement du modèle. Pour la régression, la valeur finale prédite est obtenue par l'agrégation (moyenne) des résultats de tous les arbres [49]. Cette étape est répétée jusqu'à ce que le nombre d'arbre défini (ntree) soit atteint sur la base des sous échantillons d'observation.

L'algorithme des forêts aléatoires inclue une méthode pour évaluer l'importance des variables du modèle. Il permet également de calculer les erreurs d'estimation dans le but d'examiner la précision du modèle. Cette erreur est appelée $Out-Of-Bag(OOB)$. Elle se calcule ainsi : soit (x_i, y_i) une observation de l'échantillon d'apprentissage X, soit $\hat{y}_i$ la valeur d'agrégation construite à partir des valeurs de prédiction des arbres prédicteurs: $h_k(X)$ de x_i, et supposons que cette observation est $Out-of-bag(OOB)$ c'est-à-dire n'existe pas dans les arbres de construits. L'erreur commise sur cette observation est $(\hat{y}_i - y_i)^2$. En appliquant ce procédé sur l'ensemble des données d'observation, l'erreur quadratique moyenne (ou erreur OBB) est donnée ainsi:

$$OOB = \frac{1}{n}\sum_{j=1}^{n}(\hat{y_i} - y_i)^2$$

4.6.4 Valeurs des paramètres

Dans cette section, nous donnons les étapes suivies pour identifier les valeurs des paramètres de nos modèles. Nous présentons les caractéristiques des jeux de données qui ont été utilisés et les paramètres de l'architecture physique sur lequel sont configurés les SSDs. Nous décrivons également le processus de collecte des statistiques des requêtes d'apprentissage lors de leur traitement.

4.6.4.1 Configuration matérielle

Nous avons utilisé un serveur ayant un processeur Intel (R) core (TM) i7- 4470 de 3.40 GHz (3.90 GHz en turbo), une mémoire RAM de 16 Go et un disque dur SATA de 500 Go. Le CPU possède 4 cœurs (L1D cache : 4x32 Ko, L2D cache : 4x256 ko, L3D cache : 8 Mo) et l'enveloppe thermique, ou TDP (Thermal Design Power) est de 84,0 W. Pour mesurer la consommation réelle du système, nous avons utilisé l'ampèremètre « Watts Up Pro ». Il a été directement connecté à la prise d'alimentation murale qui à son tour alimente le serveur. Les valeurs mesurées par l'équipement sont enregistrées dans un fichier texte sur une machine de contrôle au fur de l'exécution des requêtes. La topologie de notre environnement de travail est présentée dans la figure 4.27.

Nous notons que l'outil de contrôle dynamique de la fréquence (DVFS) sur Intel appelé EIST (Enhanced Intel SpeedStep Technology) pour améliorer l'efficacité énergétique n'est pas appliqué dans nos expérimentation. Cette technique peut entraîner des erreurs dans la mesure de l'énergie. Nous l'avons désactivé dans le bios du système.

4.6.4.2 Données d'apprentissage

Dans la phase d'apprentissage, nous avons utilisé des jeux de données issus du benchmark TPC-H. Le benchmark modélise la gestion des Produits – Commandes - Fournisseurs contenant 2 tables de faits et 6 tables de dimensions (Figure 4.29). Il possède un outil QGEN qui génère une charge de 22 requêtes décisionnelles caractérisées par un large volume de données et un degré élevé de complexité dont les valeurs de certaines variables ne sont pas définies au préalable. Nous générons des données à différents facteurs d'échelle: 1 Go, 2 Go, 3 Go, 5 Go, 10 Go, 30 Go. Pour l'apprentissage des modèles de PostgreSQL et MonetDB, nous avons étudiés les caractéristiques de soixante-dix (70) requêtes (Annexe B) de Sélection-Projection-Jointure (SPJ). Pour le système Hyrise, nous avons utilisé le même nombre

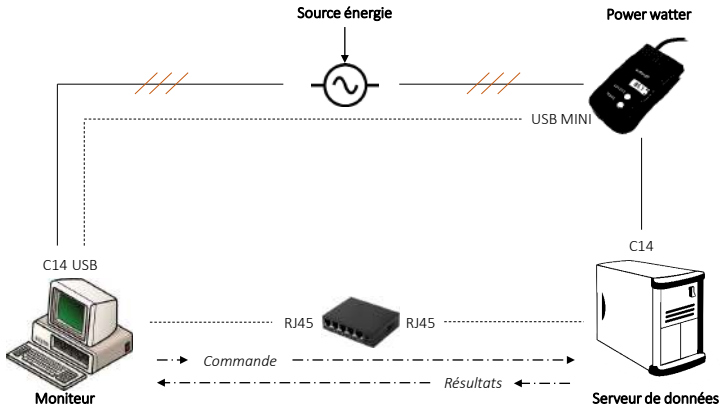


FIGURE 4.27 – Architecture de l'environnement de l'expérimentation.

de requêtes sur des tailles (facteur d'échelle) plus petite à cause des limitations de la mémoire RAM. Ces requêtes sont composées de simples à complexes impliquant des opérations gourmandes en CPU et nécessitant des E/S importantes. Après le peuplement des instances de la BD, pour chaque requête, nous mesurons sa consommation énergétique et nous reprenons le processus plusieurs fois pour nous assurer de la qualité des mesures et des statistiques collectées. Pour la collecte des statistiques, nous avons utilisé les

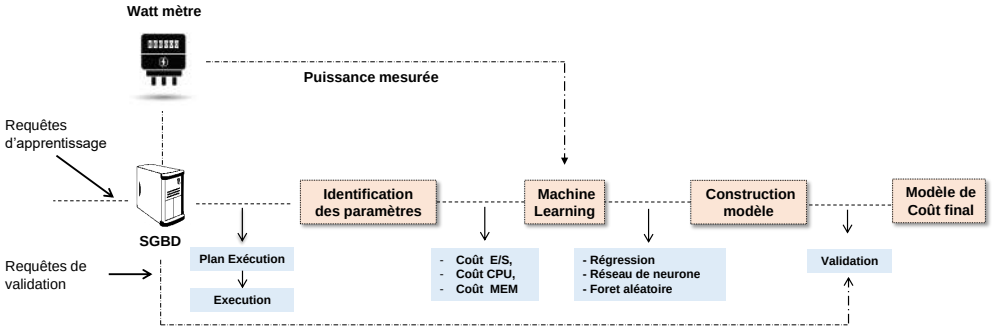


FIGURE 4.28 – Processus d'apprentissage automatique.

outils d'analyse disponible au sein des systèmes comme *Explain* sur PostgreSQL, *Trace* sur MonetDB. Sur MonetDB et sur Hyrise pour la collecte des données du nombre d'instructions exécutées et les défauts de caches, nous utilisons des outils externes aux systèmes mais intégrés dans les applications et script automatisés que nous avons développées. Nous avons utilisé Cachegrind [16] et PAPI [17]. La figure 4.28 résume le processus d'apprentissage de nos modèles de coût.

4.6.4.3 Application avec le logiciel R

Comme mentionné ci-dessus, les valeurs de certains paramètres des modèles sont estimées à partir de l'apprentissage automatique. Après avoir fait une série d'observation pendant lesquelles nos requêtes choisies pour l'apprentissage sont exécutées l'une après l'autre, tout en collectant la consommation d'énergie et les statistiques d'exécution de chacune d'elle. Nous utilisons le langage R (version 3.5.2) pour déterminer les valeurs des paramètres de nos modèles en utilisant les différents algorithmes décrits dans la Section 4.6.3.

R est un langage de programmation et un logiciel gratuit destiné à l'analyse statistique et à la science des données [74]. R dispose d'un très grand nombre de fonctions fournies par les contributeurs de manière volontaire pour les analyses statistiques et les graphiques. R possède des bibliothèques pour la modélisation polynomiale, linaire, réseau de neurones artificiel et les forêts aléatoires que nous allons

16. https://valgrind.org/docs/manual/cg-manual.html
17. `https://icl.utk.edu/papi/`

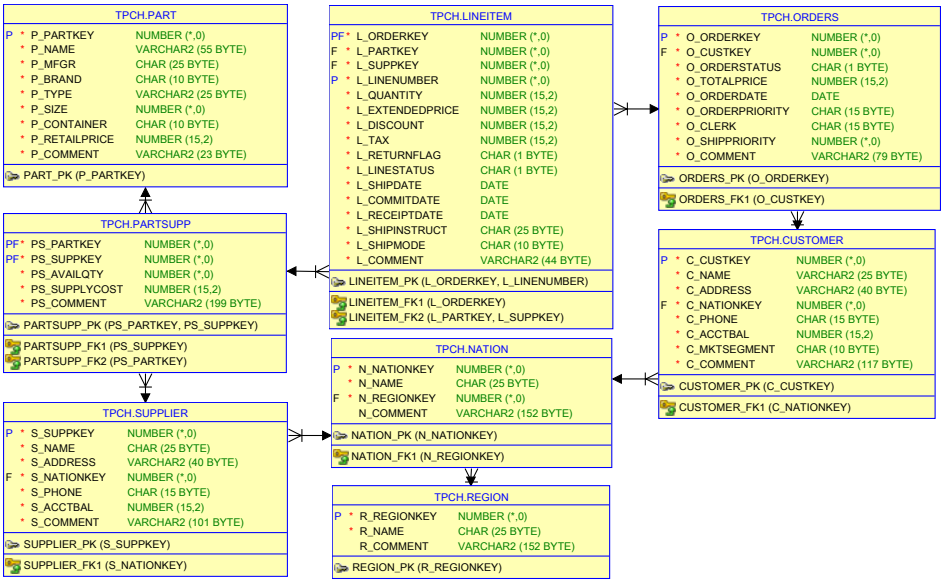


FIGURE 4.29 – Schéma du benchmark TPC-H.

appliquer sur nos modèles. Nous décrivons ci-dessous les valeurs de nos modèles en les regroupant par technique d'apprentissage.

(i) **Regression polynomiale:**
Nous avons commencé par appliquer une régression linéaire simple sur nos modèles, nous nous sommes rendus compte que cette technique ne donne pas de meilleure estimation de la puissance du fait que les données ne sont pas fortement corrélées linéairement avec la puissance. Par conséquent nous avons opté pour la régression polynomiale de degré p.

— **PostgreSQL:** Avec le premier modèle, partant de nos expérimentations le degré fixé à 2 nous donne des erreurs d'estimation non considérables de la puissance (la somme des carrés des résidus est faible). Nous avons utilisé la régression linéaire simple pour trouver la valeur du paramètre fc_{DdP} et la régression polynomiale pour déterminer les valeurs des unités énergétiques W_{CPU}, $W_{E/S}$. En appliquant la technique de la régression linéaire sur l'équation 4.4, le paramètre «facteur énergétique» s'écrit en fonction du degré de parallélisme comme suite:

$$fc_{DoP} = \alpha * DdP + \beta + \epsilon \tag{4.12}$$

où DoP est le degré de parallélisme et ϵ représente l'erreur d'estimation. Les paramètres α et β sont des coefficients de régression. L'équation 4.13 décrit l'équation du modèle linéaire dans 4.12 avec les valeurs des coefficients de régression.

$$fc_{DdP} = 0.0552 * DdP + 0.00783 + \epsilon \tag{4.13}$$

De la même manière, en appliquant la régression polynomiale de degré 2 sur l'équation 4.3 et 4.5, les coûts de la puissance moyenne du pipeline séquentiel et parallèle deviennent respectivement:

$$\begin{aligned} P(PL_i, 1) &= \beta_1 * C_{E/S} + \beta_2 * C_{CPU} + \beta_3 * C_{E/S}^2 + \beta_4 * C_{CPU}^2 + \beta_5 * C_{E/S} * C_{CPU} \\ &+ \beta_0 + \epsilon \end{aligned} \tag{4.14}$$

$$\begin{aligned} P(PPL_i, DdP) &= \beta_1 * C_{E/S} + \beta_2 * C_{CPU} * fc_{DdP} + \beta_3 * C_{E/S}^2 + \beta_4 * C_{CPU}^2 * fc_{DdP} \\ &+ \beta_5 * C_{E/S} * C_{CPU} * fc_{DdP} + \beta_0 + \epsilon \end{aligned} \tag{4.15}$$

où C_{CPU}, $C_{E/S}$ sont respectivement le coût du processeur et le coût des E/S du pipeline. Ces valeurs sont calculées sur la base des statistiques sauvegardées sur la BD dans le SGBD, ϵ représente

les erreurs de mesure et $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5$ sont des coefficients de régression. L'équation 4.16 décrit l'équation du modèle polynomial dans 4.15 avec les valeurs des coefficients de régression.

$$\begin{aligned} P(PPL_i, DdP) &= 39.91 - 4.58 * 10^{-6} * C_{E/S} + 2.64 * 10^{-12} * C_{E/S}^2 \\ &+ 9.02 * 10^{-6} * C_{CPU} * fc_{DdP} + 8.47 * 10^{-13} * C_{CPU}^2 * fc_{DoP} \\ &- 4.07 * 10^{-12} * C_{E/S} * C_{CPU} * fcDoP \end{aligned} \tag{4.16}$$

Avec le second modèle, sur la base de nos expérimentations nous avons fixé le degré à 2. Après application de la régression polynomiale dans R, l'équation 4.7 décrivant le coût de la puissance moyenne d'un pipeline se formule ainsi:

$$\begin{aligned} P(PL_i) &= \beta_1 * C_{CPU} + \beta_2 * C_{mem} + \beta_3 * C_{E/S} + \beta_4 * C_{CPU} * C_{mem} + \beta_5 * C_{mem} * C_{E/S} \\ &+ \beta_6 * C_{CPU} * C_{E/S} + \beta_7 * C_{CPU}^2 + \beta_8 * C_{mem}^2 + \beta_9 * C_{E/S}^2 + \beta_0 + \epsilon \end{aligned} \tag{4.17}$$

où $C_{CPU}, C_{mem}, C_{E/S}$ sont des paramètres dont les valeurs sont estimées sur la base des statistiques du SGBD, *epsilon* représente les erreurs de mesure et $\beta_1, \therefore, \beta_9$ sont des coefficients de la régression polynomiale. L'équation 4.18 décrit l'équation du modèle polynomial dans 4.17 avec les valeurs des coefficients de régression.

$$\begin{aligned} P(PL_j) &= 2.07 * 10^{-7} * C_{E/S} + 5.78 * 10^{-8} * C_{mem} + 1.07 * 10^{-6} * C_{CPU} \\ &- 6.17 * 10^{-15} * C_{CPU} * C_{mem} + 1.03 * 10^{-14} * C_{mem} * C_{E/S} \\ &- 8.46 * 10^{-14} * C_{CPU} * C_{E/S} + 1.60 * 10^{-13} * C_{CPU}^2 + \\ &- 3.10 * 10^{-15} * C_{mem}^2 - 8.56 * 10^{-16} * C_{E/S}^2 + 41,9 \end{aligned} \tag{4.18}$$

— **MonetDB:** Pour MonetDB, le degré 3 est utilisé pour trouver les valeurs des paramètres. L'équation 4.9 de la puissance d'une opération s'écrit de la manière suivante:

$$\begin{aligned} P(OP_j) &= \beta_1 * C_{CPU} + \beta_2 * C_{mem} + \beta_3 * C_{E/S} + \beta_4 * C_{CPU} * C_{mem} \\ &+ \beta_5 * C_{mem} * C_{E/S} + \beta_6 * C_{CPU} * C_{E/S} + \beta_7 * C_{CPU} * C_{mem} * C_{E/S} \\ &+ \beta_8 * C_{CPU}^2 + \beta_9 * C_{mem}^2 + \beta_{10} * C_{E/S}^2 + \beta_{11} * C_{CPU}^2 * C_{mem} \\ &+ \beta_{12} * C_{CPU} * C_{mem}^2 + \beta_{13} * C_{CPU}^2 * C_{E/S} + \beta_{14} * C_{mem}^2 * C_{E/S} \\ &+ \beta_{15} * C_{CPU} * C_{E/S}^2 + \beta_{16} * C_{mem} * C_{E/S}^2 \beta_{17} * C_{CPU}^3 + \\ &+ \beta_{18} * C_{mem}^3 + \beta_{19} * C_{E/S}^3 \beta_0 + \epsilon \end{aligned} \tag{4.19}$$

où $C_{CPU}, C_{mem}, C_{E/S}$ sont des paramètres dont les valeurs sont estimées sur la base des statistiques du SGBD et les outils de mesure de performances (Processor Counter Monitor) comme PAPI, *epsilon* représente les erreurs de mesure et $\beta_1, \therefore, \beta_{19}$ sont des coefficients de la régression polynomiale. L'équation 4.20 décrit l'équation du modèle polynomial dans 4.19 avec les valeurs des coefficients de régression.

$$\begin{aligned} P(OP_i) &= 1.09 * 10^{-5} * C_{CPU} - 1.26 * 10^{-2} * C_{mem} - 4.64 * 10^{-7} * C_{E/S} \\ &- 9.26 * 10^{-9} * C_{CPU} * C_{mem} - 7.45 * 10^{-11} * C_{mem} * C_{E/S} \\ &+ 9.19 * 10^{-13} * C_{CPU} * C_{E/S} + 4.47 * 10^{-16} * C_{cpu} * C_{mem} * C_{E/S} \\ &+ 9.65 * 10^{-12} * C_{CPU}^2 + 3.08 * 10^{-6} * C_{mem}^2 + 9.50 * 10^{-15} * C_{E/S}^2 \\ &+ 8.58 * 10^{-16} * C_{CPU}^2 * C_{mem} + 9.11 * 10^{-14} * C_{CPU} * C_{mem}^2 \\ &- 1.08 * 10^{-18} * C_{CPU}^2 * C_{E/S} - 5.09 * 10^{-14} * C_{mem}^2 * C_{E/S} \\ &- 2.09 * 10^{-20} * C_{CPU} * C_{E/S}^2 + 2.77 * 10^{-18} * C_{mem} * C_{E/S}^2 \\ &- 1.30 * 10^{-18} * C_{CPU}^3 - 1.18 * 10^{-10} * C_{mem}^3 - 3.98 * 10^{-23} * C_{E/S}^3 \\ &+ 62.7 \end{aligned} \tag{4.20}$$

— **Hyrise:** Pour Hyrise, nous avons utilisé l'ordre 2 pour trouver les valeurs des paramètres. L'équation

4.11 de la puissance d'une opération s'écrit de la manière suivante:

$$\begin{aligned} P(OP_i) &= \beta_1 * C_{CPU} + \beta_2 * C_{CPU}^2 + \beta_3 * C_{L2D} \\ &+ \beta_4 * C_{CPU} * C_{L2D} + \beta_5 * C_{L2D}^2 + \beta_6 * C_{L3D} \\ &+ \beta_7 * C_{CPU} * C_{L3D} + \beta_8 * C_{L2D} * C_{L3D} + \beta_9 * C_{L3D}^2 \\ &+ \beta_{10} * C_{mem} + \beta_{11} * C_{CPU} * C_{mem} - \beta_{12} * C_{L2D} * C_{mem} \\ &+ \beta_{13} * C_{L3} * C_{mem} + \beta_{14} * C_{mem}^2 + \beta_0 \end{aligned} \quad (4.21)$$

où C_{CPU}, C_{mem}, $C_{E/S}$, C_{L2D} , C_{L3D} sont des paramètres dont les valeurs sont estimées par les outils de mesure de performances (Processor Counter Monitor) comme PAPI, *epsilon* représente les erreurs de mesure et $\beta_1, \therefore, \beta_{19}$ sont des coefficients de la régression polynomiale. L'équation 4.22 décrit l'équation du modèle polynomial dans 4.21 avec les valeurs des coefficients de régression.

$$\begin{aligned} P(OP_i) &= -4.25 * 10^{-10} * C_{CPU} + 3.50 * 10^{-20} * C_{CPU}^2 - 2,80 * 10^{-7} * C_{L2D} \\ &- 1.85 * 10^{-17} * C_{CPU} * C_{L2D} + 7.25 * 10^{-15} * C_{L2D}^2 + 8.03 * 10^{-6} * C_{L3D} \\ &+ 1.96 * 10^{-16} * C_{CPU} * C_{L3D} - 1.34 * 10^{-13} * C_{L2D} * C_{L3D} + 9.71 * 10^{-14} * C_{L3D}^2 \\ &+ 5.58 * 10^{-7} * C_{mem} + 2.32 * 10^{-17} * C_{CPU} * C_{mem} - 2.42 * 10^{-14} * C_{L2D} * C_{mem} \\ &+ 2.22 * 10^{-13} * C_{L3} * C_{mem} + 2.09 * 10^{-14} * C_{mem}^2 + 39,3 \end{aligned} \quad (4.22)$$

Pour vérifier l'adéquation de nos modèles de régression aux données d'apprentissage, nous calculons le coefficient de détermination noté par R^2 à partir du logiciel R. Le coefficient de corrélation est une quantité comprise entre -1 et 1 et indique la qualité de la mesure des variables dépendantes à partir des variables indépendantes [121]. Dans le tableau 4.3, nous donnons pour chaque modèle son coefficient de détermination indiqué par R^2, son R ajusté (Adjusted R-squared) et la racine carré de l'erreur quadratique moyenne (MSE).

Techniques	**Variables**	**Systèmes**			
		PostgreSQL		**MonetDB**	**Hyrise**
		1	**2**		
NLR	Multiple R-squared	0,66	0.77	0.64	0.75
	Adjusted R-squared	0,62	0.75	0.61	0.73
	$\sqrt{MSE}$	4,81	2,71	6,22	2,52
RNA	RMSE	1,94	5,57	9,03	4,05
	MAPE	3,38	3,68	6,81	2,28
FAR	R-squared	0,72	0.88	0.89	0.67
	RMSE	4,55	1.82	3.26	2.65
	MAPE	3,10	0,73	2,29	1,40

TABLEAU 4.3 – Analyse statistique de l'erreur de prédiction obtenue avec les différentes approches.

(ii) **Réseaux de neurones:**
Le modèle FFNN utilisé dans cette étude pour la prédiction de la puissance moyenne des requêtes est un réseau de neurone à trois couches composé d'une couche d'entrée, d'une couche de sortie et d'une couche cachée. Chaque couche comprend un certain nombre de nœuds. Chaque nœud reçoit les valeurs des nœuds de la couche précédente, puis détermine sa propre valeur et la transmet aux nœuds de la couche suivante. Ce processus est répété jusqu'à ce que les nœuds de la couche de sortie produisent des valeurs proches des données d'apprentissage. Pour la précision du modèle, il est essentiel parfois de varier les valeurs de certains hyper-paramètres comme le nombre de couche cachée, le nombre de nœuds dans chaque couche cachée, la fonction d'activation, l'algorithme d'apprentissage et le nombre de répétition.

— **PostgreSQL:** Pour PostgreSQL, nous utilisons un algorithme à propagation-avant pour entraîner le modèle. Cet algorithme est basé sur la méthode de descente du gradient qui met à jour les poids itérativement jusqu'à ce qu'il parvienne à une erreur de sortie considérée comme négligeable entre les valeurs cibles du réseau et les valeurs d'apprentissage. La fonction d'activation (de transmission) utilisée entre les neurones dans la couche cachée est une fonction logistique (logsig). Le critère d'arrêt définit dans la fonction d'erreur est $0,08$ et le nombre maximal d'itérations est fixé à 50000. Le nombre de neurones définit dans les couches d'entrée et de sortie sont respectivement 3 et 1 pour le deuxième modèle. Pour le premier modèle, ils sont définit à 2 et 1. Après plusieurs expérimentations en variant les hyper-paramètres d'apprentissage (Figure 4.30), nous avons constaté que la structure du réseau mena aux meilleurs résultats correspond à la structure $3-4-3-1$ comme illustré sur la figure 4.31, avec 3 neurones dans la couche d'entrée, $4-3$ dans la couche cachée et 1 dans la couche de sortie. Le tableau 4.4 résume les valeurs des arguments utilisés pour construire nos modèles.

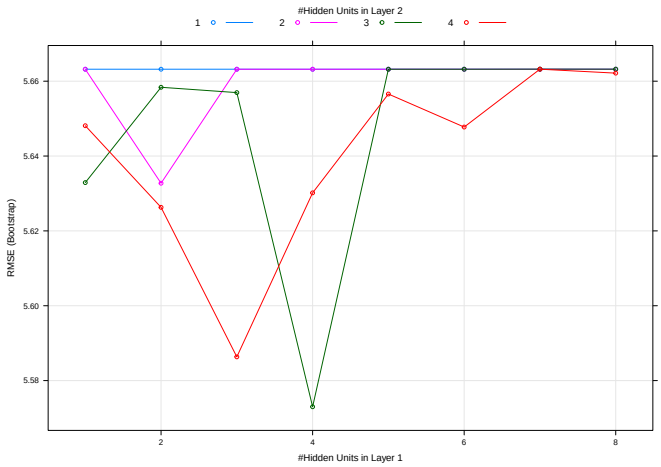


FIGURE 4.30 – Paramètres d'apprentissage de la structure neuronale (PostgreSQL)

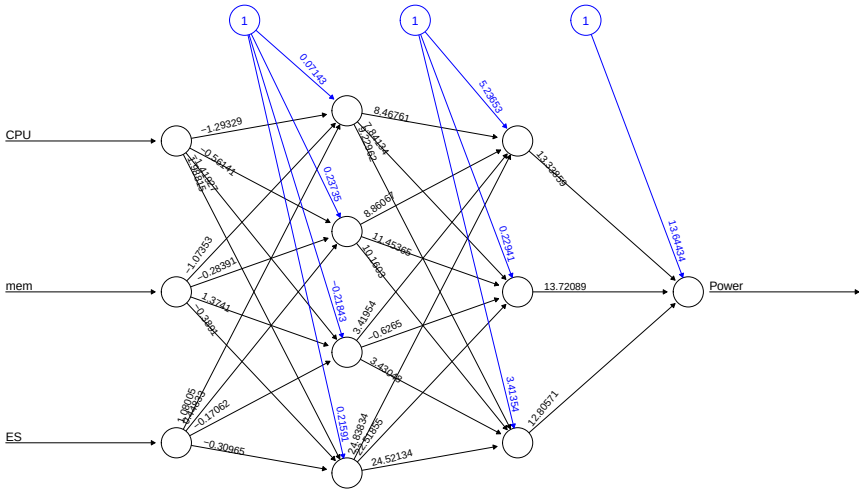


FIGURE 4.31 – Structure neuronale du modèle comprenant les poids synaptiques (PostgreSQL)

— **MonetDB:** Pour MonetDB, nous utilisons la même configuration que celle décrite au niveau du cas de PostgreSQL. Le critère d'arrêt de la fonction d'erreur est fixé à $0,08$ et 50000 pour le nombre maximal d'itérations. Le nombre de neurones dans les couches d'entrée et de sortie sont fixés à 3 et 1 respectivement. Après un certain nombre d'essais

Arguments	Valeurs			
	PostgreSQL		MonetDB	Hyrise
	1	2		
Structure neuronale	2-4-1	3-4-3-1	3-6-1	4-8-1
Nombre couche cachée	1	2	1	1
Nombre d'étape	100000	50000		
Algorithme	RProp-	RProp+		
Fonction d'erreur	SSE			
Fonction d'activation	Logistic			
Lifesign	Minimal			
Seuil d'arrêt	0,02	0,08		

TABLEAU 4.4 – Paramétrage des arguments du réseau de neurone artificiel lors des processus d'apprentissage.

(Figure 4.32), nous remarquons que le nombre de neurones fixé à 6 dans la couche cachée conduit le réseau neuronal aux meilleurs résultats. Ainsi, le nombre de neurones dans chaque couche du réseau est de $3-6-1$ comme illustré sur la figure 4.33. Le tableau 4.4 résume les valeurs des arguments utilisés pour construire le modèle de MonetDB.

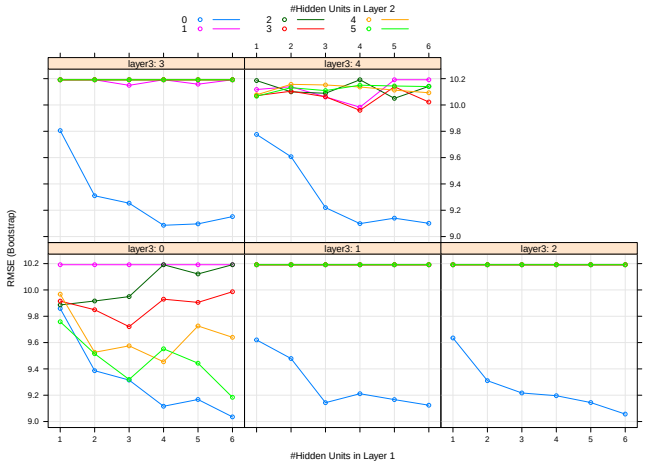


FIGURE 4.32 – Paramètres d'apprentissage de la structure neuronale (MonetDB)

— **Hyrise:** Pour Hyrise, Le critère d'arrêt de la fonction d'erreur a été fixé à $0,08$ et le nombre maximum d'itérations à 50000. Le nombre de neurones dans les couches d'entrée et de sortie ont été fixé à 4 et 1 respectivement. Après une série d'expérimentation (Figure 4.34), il a été constaté que la structure $4-8-1$ du réseau neuronal conduit aux meilleurs résultats, avec 8 le nombre de neurones dans la couche intermédiaire. La figure 4.35 illustre la structure neuronale retenue. Le tableau 4.4 résume les valeurs des arguments utilisés pour construire le modèle de Hyrise.

L'adéquation de nos modèles des réseaux de neurones est vérifiée en mesurant l'erreur quadratique moyenne (Root Mean-Squared Error - RMSE) et le pourcentage de l'erreur absolue moyenne(Mean Absolute Percentage Erro - MAPE). Le tableau 4.3 présente pour chaque modèle les valeurs des coefficients cités ci-dessous.

(iii) **Forêts aléatoires:**
Une forêt aléatoire de régression est un ensemble d'arbre de régression construit à partir d'un échantillon d'apprentissage en utilisant l'algorithme CART. Les branches de chaque arbre sont

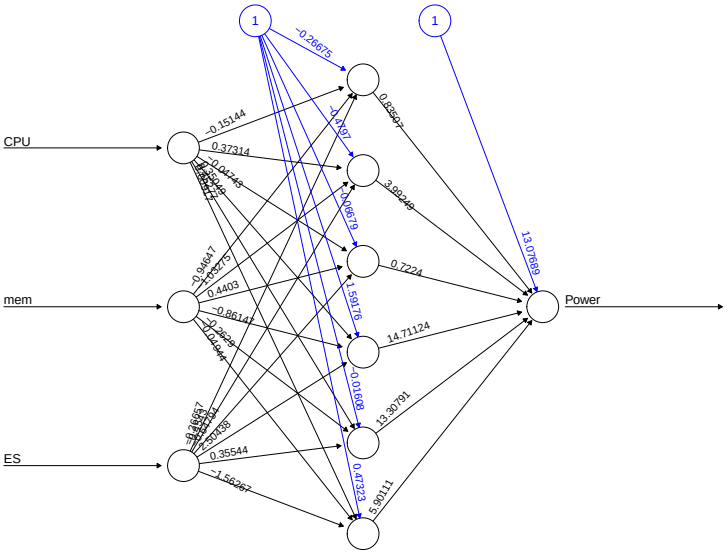


FIGURE 4.33 – Structure neuronale du modèle comprenant les poids synaptiques (MonetDB)

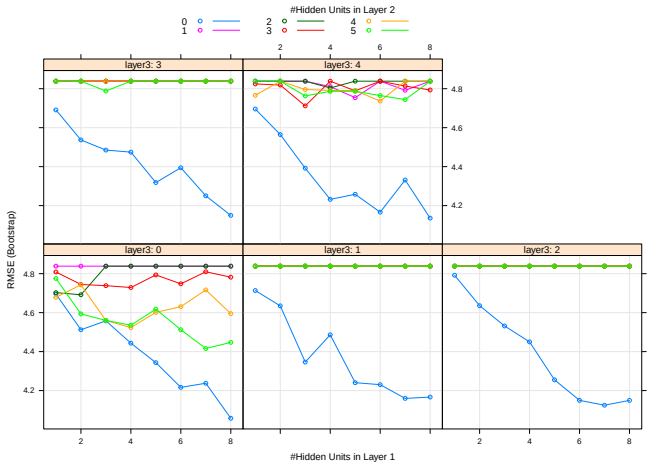


FIGURE 4.34 – Paramètres d'apprentissage de la structure neuronale (Hyrise)

subdivisées tant que le nombre minimum d'observation dans chaque feuille est supérieur à une des valeurs à prédire. Pour la sélection des valeurs adéquates des hyper-paramètres du modèle de la forêt aléatoire (*ntree*, *mtry* et *nodesize*), nous avons utilisé une approche itérative en variant un à un les valeurs des paramètres dans des plages de valeurs définies. La valeur optimale de l'hyper-paramètre *mtry* est sélectionné dans la plage de $\{1, 2, 3, 4\}$, et celle de *ntree* dans une marge de $[10 - 500]$. Les valeurs des paramètres qui optimisent les erreurs d'estimations sont sélectionnées comme optimaux à la fin du processus d'apprentissage. Nous construisons nos modèles en utilisant le package de *#RandomForest#*.

— **PostgreSQL:** Le modèle a été entraîné en utilisant toutes les variables pertinentes identifiées. La détermination des valeurs optimales des paramètres est effectuée à l'aide du langage R en changeant itérativement une à une les valeurs des paramètres. Après plusieurs itérations, les valeurs optimales obtenues pour le premier modèle furent 2, 8, 14 et 2, 500, 14 pour le second respectivement pour mtry, ntree et nodesize. La figure 4.36a illustre l'analyse de l'évolution des valeurs du paramètre *#mtry#* à l'issu de l'apprentissage automatique pour le système PosgreSQL.

— **MonetDB:** Pour les données d'apprentissage collectées pour MonetDB, la valeur optimale est atteinte à 30 arbres (ntree). La valeur optimale pour le paramètre mtry est 2 et

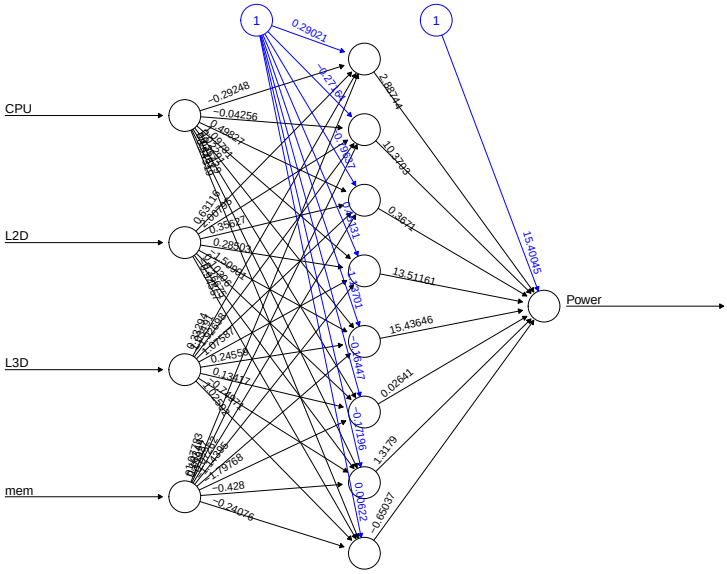
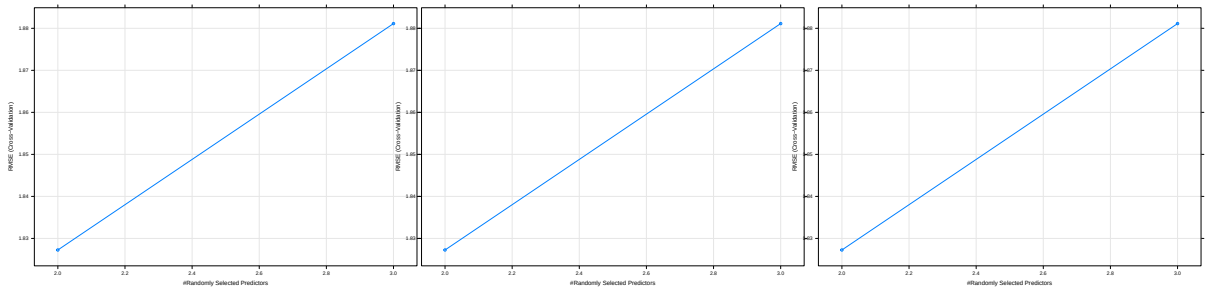


FIGURE 4.35 – Structure neuronale du modèle comprenant les poids synaptiques (Hyrise)



(a) PostgreSQL (b) MonetDB (c) Hyrise

FIGURE 4.36 – Paramètre *#mtry#* lors du processus d'apprentissage du modèle.

10 pour nodesize. Lorsque la valeur de ntree est trop petite, les résultats se détériorent considérablement et lorsqu'elle augmente au-dessus de l'optimum, les résultats ne s'améliorent pas significativement mais le temps de calcul processeur s'augmente. La figure de 4.36b illustre l'analyse de l'évolution des valeurs du paramètre *#mtry#* à l'issu de l'apprentissage automatique pour le système MonetDB.

— **Hyrise:** Pour le système Hyrise, la valeur optimale pour le paramètre ntree est 500 (valeur par défaut) et 2 pour mtry. La figure de 4.36c illustre l'analyse de l'évolution des valeurs du paramètre *#mtry#* à l'issu de l'apprentissage automatique pour le système Hyrise.

Le tableau 4.5 résume les valeurs des hyper-paramètres utilisés pour construire nos modèles. Pour évaluer la qualité de nos modèles, nous tenons compte de l'adéquation des données d'apprentissage en calculant le coefficient de détermination. Le tableau 4.3 résume les statistiques des résultats d'apprentissage. Nous donnons pour chaque modèle le coefficient de corrélation indiqué par R^2 , et l'erreur de validation out-of-bag(OOB).

Arguments	Valeurs			
	PostgreSQL		MonetDB	Hyrise
	1	2		
Mtry	2	2	2	2
Ntree	8	500	30	500
Nodesize	14	14	10	14

Tableau 4.5 – Paramétrage des arguments de l'algorithme des forêts aléatoires lors des processus d'apprentissage.

4.7 Validation des modèles

Dans cette section, nous présentons les résultats expérimentaux évaluant la précision de nos modèles proposés pour les différents systèmes de stockage. Nous décrivons tout d'abord la méthodologie utilisée pour évaluer la qualité des modèles proposés et les jeux de données utilisés. Puis pour chaque modèle nous évaluons sa performance de prédiction avec chacune des trois techniques d'apprentissage (Non linéaire, Réseau de neurone, forêt aléatoire).

4.7.1 Méthode d'évaluation

La méthodologie adoptée pour évaluer la qualité de nos modèles consiste à la détermination de l'erreur de prédiction calculée par la différence entre la consommation d'énergie mesurée (watt mètre) et la consommation d'énergie estimée en pourcentage. En supposant que le coût de l'énergie estimé par notre modèle pour une requête Q_i est noté ($E_{Estimee}$) et le coût de l'énergie réel noté par ($E_{Mesuree}$) mesuré à l'aide d'un équipement physique, l'erreur de prédiction en pourcentage ($Erreur$) est calculé par la formule suivante pour une seule requête:

$$Erreur_{Qi} = \frac{|E_{Mesuree} - E_{Estimee}|}{E_{Mesuree}} \times 100\% \tag{4.23}$$

Pour quantifier la précision du modèle sur un ensemble de requête, nous calculons la moyenne des erreurs ($Erreur_{Moy}$) de prédictions formulée comme suite:

$$Erreur_{Moy} = \frac{1}{n} \times \sum_{i=1}^{n} Erreur_{Qi} \tag{4.24}$$

Pour comparer l'efficacité des techniques d'apprentissage, nous nous referons à la moyenne des erreurs de prédiction sur l'ensemble des requêtes.

4.7.2 Jeux de données

En plus du jeu de données et les requêtes du benchmark TPC-H dont le schéma est décrit sur la figure 4.29, nous utilisons deux jeux de données et les requêtes issues des benchmarks SSB et TPC-DS.

TPC-DS est le standard de référence développé par TPC pour pallier aux lacunes du benchmark TPC-H lors des tests sur des systèmes d'aide à la décision (SDD) beaucoup plus complexes. Le schéma du TPC-DS modélise le processus de ventes et de retours des produits pour une organisation qui utilise trois points de vente: les magasins, les catalogues et Internet. Le schéma comprend 7 tables de faits et 17 tables de dimensions. Il s'agit d'un schéma en constellation. Le tableau 4.6 présente quelques statistiques sur les caractéristiques des tables du schéma TPC-DS. La charge de requête fournit par TPC-DS est composée de 99 requêtes, divisée en quatre catégories de requêtes: requêtes de rapports, requêtes décisionnelles ad-hoc, requêtes traitement analytique en ligne (On-Line Analytical Processing, OLAP), requêtes d'exploration de données [122].

Le SSB est un benchmark des entrepôts de données dérivé de TPC-H. Contrairement au TPC-H, il utilise un modèle conceptuel multidimensionnel en étoile composé d'une table de fait et de 4 tables de dimensions. Il contient moins de requêtes que TPC-H et n'a pas des exigences strictes en terme de formes de calibrage (tuning) autorisées et interdites [132].

Nombre de table de fait	7	
Nombre de table de dimension	17	
Nombre de colonnes	minimum	3
	maximum	34
	Moyenne	18
Nombre de clés étrangères	104	
Taille des attributs (Octets)	minimum	16
	maximum	317
	moyenne	136

Tableau 4.6 – Statistiques sur les caractéristiques du schéma TPC-DS.

4.7.3 Erreurs d'estimation

Dans cette section, nous présentons les erreurs d'estimation obtenues pour chaque système lors de l'évaluation de nos modèles sur un ensemble de données et une charge de requête bien définie.

4.7.3.1 Évaluation sur PostgreSQL

Nous exploitons nos modèles énergétiques décrit dans la sous-section 4.6.4.3 (équations 4.16, 4.18) pour estimer la puissance énergétique des requêtes de Selection-Jointure-Projection(SPJ) sur des jeux de données de taille différentes. Puis nous comparons l'énergie estimée par nos modèles de coût avec l'énergie réelle consommée par le serveur de BD pendant le traitement des requêtes en utilisant notre méthodologie d'évaluation.

4.7.3.1.1 Résultats avec TPC-H

Les tableaux 4.7 et 4.7c résument les erreurs d'estimation obtenues avec le premier modèle proposé sur un jeu de données de taille 10 Go. En appliquant la technique de la Régression Non Linéaire (NLR), l'erreur d'estimation de chacune des 22 requêtes est donnée dans le tableau 4.7a en fixant le degré de parallélisme à 2 puis à 4 respectivement. Les moyennes d'erreurs sont respectivement 9,85% et 12,16% pour le degré de parallélisme fixé à 2 puis à 4. En utilisant la technique des Réseaux de Neurones Artificiels (RNA) dont les résultats sont décrits dans le tableau 4.7b, les moyennes d'erreurs sont respectivement 7,89% et 9,63% pour le degré 2 et le degré 4. Nous obtenions respectivement 12,90% et 13,61% pour le degré 2 et le degré 4 comme moyenne d'estimation en utilisant la technique des forêts aléatoires de régression (FAR) dont les résultats sont donnés dans le tableau 4.7c.

La figure 4.37a illustre les moyennes d'erreurs des trois techniques sur des degrés de parallélisme fixés à 2 et à 4. Comme le montre la figure, la technique des réseaux de neurones (RNA) se révèle être la plus adéquate en faisant moins d'erreur d'estimation que la technique de forêt aléatoire et celle de la régression non linéaire (NLR) dans ces premières expérimentations.

Le tableau 4.8 présente les taux d'erreur d'estimation obtenus en exploitant le second modèle décrit dans 4.18. Comme nous pouvons le constater à partir du sous-tableau 4.8a, l'erreur moyenne en utilisant la technique de la régression non linéaire est $6,69\%$ sur le jeu de données de 10 Go et $11,58\%$ sur 50 Go, et l'erreur maximale est inférieur à 25% dans les deux cas. L'erreur moyenne est de $10,47\%$ (resp. $10,65\%$) et l'erreur maximale est de $19,50\%$ (resp. 18%) en utilisant la technique des réseaux de neurones (sous-table 4.8b) sur un jeu de données de 10 Go (resp. 50 Go). De même en utilisant la technique de forêt aléatoire (sous-table 4.8c), l'erreur moyenne est de $6,82\%$ (resp. $14,79\%$) sur le jeu de données de 10 Go (resp. 50 Go).

Pour résumer, nous présentons sur la figure 4.37 les erreurs moyennes obtenues sur chaque facteur d'échelle en utilisant les trois techniques d'apprentissage. Comme le montre la figure, en comparant les erreurs moyennes obtenues des trois techniques d'apprentissage, nous constatons que la technique des réseaux de neurone estime mieux la puissance que la technique de la régression non linéaire et celle de la technique de forêt aléatoire.

TABLEAU 4.7 – Taux d'erreurs d'estimation d'énergie des requêtes en mode parallèle (PostgreSQL - TPC-H)

(a) *Modèle NLR*

SF=10 Go					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	DdP=2	*DdP=4*		*DdP=2*	*DdP=4*
1	11,39%	14,32%	12	6,38%	10,65%
2	16,31%	18,16%	13	15,20%	20,86%
3	0,41%	3,69%	14	6,17%	8,91%
4	15,18%	14,18%	15	2,16%	15,55%
5	7,57%	9,11%	16	19,72%	20,18%
6	5,83%	8,70%	17	17,25%	17,34%
7	6,99%	7,02%	18	11,17%	12,76%
8	4,19%	2,24%	19	16,55%	5,82%
9	2,01%	5,74%	20	11,10%	9,45%
10	1,71%	2,40%	21	-	-
11	15,30%	24,30%	22	14,19%	24,04%

(b) *Modèle RNA*

SF=10 Go					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	DdP=2	*DdP=4*		*DdP=2*	*DdP=4*
1	5,39%	12,79%	12	4,41%	5,31%
2	19,12%	20,91%	13	22,53%	12,91%
3	1,37%	3,56%	14	1,58%	3,07%
4	15,67%	14,35%	15	11,13%	26,60%
5	5,64%	6,13%	16	5,06%	5,61%
6	2,46%	3,09%	17	2,62%	3,25%
7	5,98%	4,92%	18	9,47%	10,88%
8	3,18%	4,48%	19	12,13%	3,25%
9	8,41%	11,68%	20	3,78%	2,58%
10	3,82%	3,71%	21	-	-
11	11,63%	22,71%	22	10,39%	20,54%

(c) Modèle FAR

SF=10 Go					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	DdP=2	*DdP=4*		*DdP=2*	*DdP=4*
1	18,40%	7,26%	12	7,31%	9,26%
2	19,60%	21,38%	13	10,65%	31,02%
3	1,95%	5,07%	14	4,57%	6,02%
4	13,13%	10,91%	15	18,97%	1,73%
5	8,51%	8,98%	16	20,93%	21,38%
6	5,43%	6,04%	17	37,38%	33,11%
7	8,84%	7,81%	18	29,31%	26,25%
8	6,12%	1,30%	19	11,13%	6,19%
9	5,42%	29,86%	20	10,26%	4,74%
10	5,66%	0,16%	21	-	-
11	14,31%	24,43%	22	13,11%	22,94%

4.7.3.1.2 Résultats avec TPC-DS

Pour vérifier la portabilité du modèle, nous utilisons un nouveau schéma de BD (TPC-DS) plus complexe que le benchmark TPC-H pour évaluer les erreurs d'estimation d'une nouvelle charge de requête. Nous sélectionnons parmi les 99 requêtes vingt (20) requêtes gourmandes en CPU ou en E/S (Annexe B). Le tableau 4.9d montre les résultats expérimentaux en utilisant la technique de la régression non linéaire, la technique des réseaux de neurones et la technique de forêt aléatoire sur une BD de 20 Go.

On constate que la moyenne d'erreurs du modèle pour chaque technique est inférieur 15% et l'erreur maximale est 30% en utilisant la technique de la régression non linéaire. Nous observons également que les techniques de forêts aléatoires et des réseaux de neurone font moins d'erreurs d'estimation que la technique de la régression non linéaire.

4.7.3.1.3 Résultats avec SSB

Toujours dans le même objectif, nous utilisons un nouveau schéma de BD (SSB) moins complexe que le benchmark TPC-H en raison de la taille de son schéma pour évaluer les erreurs d'estimation d'une nouvelle charge de requête. Nous utilisons les 13 requêtes standards du benchmark SSB. Le tableau 4.9 présente les résultats expérimentaux en utilisant la technique de la régression non linéaire, la technique des réseaux de neurones et la technique de forêts aléatoires sur une BD de taille 50 Go et 100 Go.

TABLEAU 4.8 – Taux d'erreurs d'estimation d'énergie des requêtes en mode parallèle (PostgreSQL - TPC-H)

(a) *Modèle NLR*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=10	*SF=50*		*SF=10*	*SF=50*
1	4,02%	10,80%	12	7,89%	14,35%
2	0,75%	6,00%	13	21,73%	3,96%
3	3,47%	18,97%	14	2,08%	7,83%
4	6,58%	5,89%	15	7,94%	20,78%
5	10,73%	10,35%	16	13,66%	13,41%
6	5,62%	6,96%	17	-	-
7	6,50%	10,73%	18	4,78%	25,76%
8	2,36%	10,89%	19	3,65%	14,96%
9	4,45%	12,15%	20	-	-
10	3,88%	8,28%	21	-	-
11	7,43%	6,25%	22	9,53%	11,61%

(b) *Modèle RNA*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=10	*SF=50*		*SF=10*	*SF=50*
1	4,40%	12,22%	12	16,42%	9,48%
2	11,33%	12,75%	13	19,50%	4,75%
3	10,51%	13,92%	14	10,87%	9,24%
4	16,28%	11,07%	15	7,32%	18,00%
5	18,13%	12,01%	16	4,96%	8,36%
6	14,55%	6,41%	17	-	-
7	13,91%	10,65%	18	0,09%	13,12%
8	4,11%	7,08%	19	6,28%	7,17%
9	10,12%	10,81%	20	-	-%
10	11,64%	11,33%	21	-	-
11	17,42%	9,55%	22	1,11%	14,42%

(c) *Modèle FAR*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=10	*SF=50*		*SF=10*	*SF=50*
1	0,11%	15,10%	12	1,35%	13,50%
2	6,40%	7,85%	13	21,10%	1,38%
3	4,39%	22,24%	14	6,99%	11,91%
4	11,97%	7,75%	15	5,87%	5,83%
5	12,96%	21,34%	16	9,17%	12,03%
6	1,27%	9,26%	17	-	-
7	4,29%	19,02%	18	5,18%	47,10%
8	6,25%	11,09%	19	6,91%	12,70%
9	3,92%	23,50%	20	-	-%
10	3,54%	18,11%	21	-	-
11	12,47%	5,87%	22	5,43%	15,49%

(d) *Modèle NLR, RNA et FAR*

TPC-H (SF=100Go)								
Q_i	*Erreur(%)*			Q_i	*Erreur(%)*			
	NLR	*RNA*	*FAR*		*NLR*	*RNA*	*FAR*	
1	15,73%	9,36%	17,56%	12	28,85%	10,61%	43,83%	
2	18,61%	13,02%	15,90%	13	56, 93%	3,65%	4,76%	
3	25,66%	10,09%	43,15%	14	14,26%	9,83%	21,46%	
4	55, 94%	3,73%	25,16%	15	45, 01%	20,92%	2,82%	
5	2,07%	11,76%	45,32%	16	0,76%	1,63%	3,86%	
6	16,75%	5,92%	13,95%	17	-	-	-	
7	2,96%	11,84%	45,43%	18	7,92%	11,25%	44,66%	
8	20,97%	7,60%	39,91%	19	2,84%	23,26%	1,16%	
9	1,51%	14,00%	48,24%	20	-	-	-	
10	7,39%	9,57%	42,47%	21	-	-	-	
11	9,42%	5,82%	10,53%	22	19,58%	13,45%	19,74%	

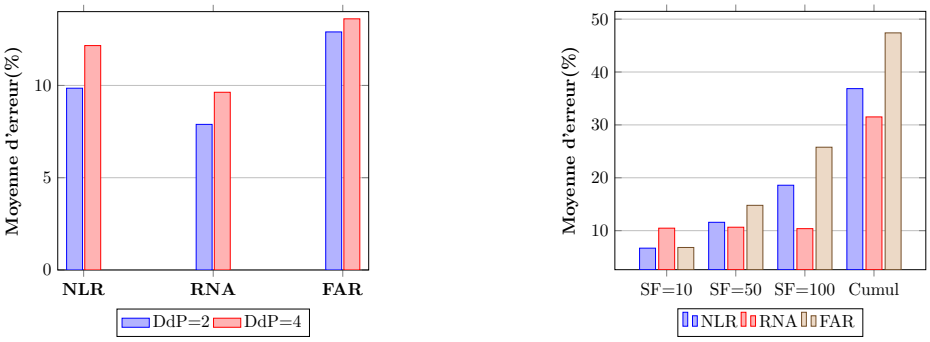
L'erreur moyenne est 12.98% et la maximale interprétable fait 18,56% en utilisant la technique de la régression non linéaire sur le jeu de donne de 100 Go. Elle fait 7,76% et le maximal est 13,11% avec la technique des réseaux de neurones. Ici également, les techniques de forêt aléatoire et des réseaux de neurone sont plus consistants que celle de la régression non linéaire car elles permettent de donner des valeurs d'estimations interprétables même sur de grandes quantités de données.

4.7.3.2 Évaluation sur MonetDB

Comme pour le système PostgreSQL, nous évaluons le modèle énergétique décrit dans l'équation 4.9 pour estimer la puissance énergétique des requêtes sur différents jeux de données. Pour chaque benchmark, nous donnons les détails des résultats expérimentaux obtenus en utilisant les trois techniques d'apprentissage (régression non linéaire, réseaux de neurone et la forêt aléatoire).

4.7.3.2.1 Résultats avec TPC-H

Dans le tableau 4.10, nous donnons les taux d'erreurs d'estimation obtenues en utilisant les trois techniques d'apprentissage. L'erreur moyenne interprétable est de $8,99\%$ (resp. $18,68\%$) et l'erreur maximale est de $22,09\%$ (resp. 39%) en utilisant la technique de la régression non linaire (sous-table 4.10a) sur un jeu de données de 100 Go (resp. 200 Go). En utilisant la technique des réseaux de neurone (sous-table 4.10b), l'erreur moyenne est de $7,17\%$ (resp. $14,73\%$) et l'erreur maximale est de $11,56\%$



(a) le premier modèle proposée.

(b) le deuxième modèle proposée

FIGURE 4.37 – La moyenne d'erreur d'estimation des trois techniques en utilisant le benchmark TPC-H (PostgreSQL).

(resp. 34%) sur un jeu de données de 100 Go (resp. 200 Go). Avec la technique des forêts aléatoires, l'erreur moyenne est $4,9\%$ (resp. $14,02\%$) et l'erreur maximale est de $10,97\%$ (resp. 35%) sur 100 Go (resp. 200 Go). Sur le jeu de 200 Go, le modèle de la régression non linéaire donne des valeurs non interprétables sur 5 requêtes (Q_1,Q_7,Q_{15},Q_{19},Q_{20}), cela est probablement dû à un sous apprentissage du modèle de régression car lors de la phase d'apprentissage la taille maximale des données utilisées était 30 Go. Néanmoins, avec les deux autres techniques nous n'avons pas ce problème vu qu'elles restent borner dans un intervalle et qu'elles sont plus consistantes. En comparant les erreurs moyennes obtenues des trois techniques d'apprentissage, nous constatons que les techniques de forêt aléatoire et des réseaux de neurones estiment mieux la puissance énergétique que la technique de la régression non linéaire sur chacun des deux jeux de données du benchmark TPC-H.

4.7.3.2.2 Résultats avec TPC-DS

Avec 21 requêtes sélectionnées parmi les 99 requêtes du benchmark TPC-DS, nous évaluons nos techniques sur ce schéma un peu plus complexe. Nous présentons dans le tableau 4.10d les résultats expérimentaux obtenues sur une taille de données de 50 Go. On constate que la moyenne d'erreur du modèle pour chaque technique est inférieur 13% et l'erreur maximale est de 17,88% en utilisant la technique de forêt aléatoire. Cette expérience révèle la robustesse de notre modèle de coût face à un nouveau schéma plus complexe. Nous constatons en comparant la moyenne d'erreur des trois techniques que la technique des forêts aléatoires est la plus stable suivi par la technique des réseaux de neurone.

4.7.3.2.3 Résultats avec SSB

Dans ce test, nous utilisons un schéma moins complexe pour vérifier la portabilité du modèle sur deux BDs de taille 100 Go et 200 Go respectivement. Nous exécutons les treize (13) requêtes du benchmark SSB sur les deux facteurs de données. Les résultats expérimentaux sont présentés dans le tableau 4.11.

L'expérience montre que l'erreur d'estimation moyenne la plus faible est de 8,64% sur le jeu de 100 Go et 3,59% sur celui de 200 Go, toutes obtenues avec la technique des réseaux de neurones. L'erreur maximale commise par les deux techniques: la régression non linéaire et la forêt aléatoire (31,94% et 34,04 respectivement) est observée au niveau de la requête Q_{10} sur le jeu de 200 Go. Cette marge est dû à une erreur d'estimation du nombre de lecture disque (entrées-sorties) comptabilisé par le système. En analysant les données collectées par l'outil *#TRACE#* de MonetDB, nous constatons que le système ne compte aucunes lectures disques et ne fait que 1384 écritures ceci ne reflète les valeurs réelles attendues avec une telle quantité de données. Sans tenir compte de la requête Q_{10}, l'erreur maximale est 17,45% avec la technique de la régression non linéaire, 5,52% avec la technique des réseaux de neurone et 4,92% en utilisant la technique des forêts aléatoires. Ainsi, notre modèle peut également faire à ce cas de figure où le schéma est moins complexe.

4.7.3.3 Évaluation sur Hyrise

Pour Hyrise également nous évaluons notre modèle énergétique décrit dans l'équation 4.9 sur différents jeux de données de taille non conséquente à cause de la limitation matérielle (Capacité de la mémoire RAM). Pour chaque benchmark, nous donnons les détails des résultats expérimentaux résultant des trois techniques d'apprentissage.

TABLEAU 4.9 – Taux d'erreurs d'estimation d'énergie des requêtes (PostgreSQL - SSB/TPC-DS)

(a) *Modèle NLR*

SSB (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=50	*SF=100*		*SF=50*	*SF=100*
1	2,64%	18,56%	8	0,35%	9,71%
2	4,53%	16,77%	9	1,38%	16,83%
3	5,73%	17,24%	10	1,43%	-
4	0,01%	11,70%	11	0,15%	12,92%
5	0,31%	10,61%	12	0,01%	8,74%
6	0,05%	10,81%	13	0,17%	6,60%
7	0,42%	15,26%			

(b) *Modèle RNA*

SSB(SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=50	*SF=100*		*SF=50*	*SF=100*
1	4,30%	10,99%	8	3,80%	7,00%
2	5,42%	7,88%	9	4,72%	7,32%
3	6,68%	8,65%	10	4,88%	13,11%
4	3,53%	5,37%	11	2,95%	6,67%
5	3,83%	7,23%	12	3,51%	5,68%
6	4,24%	7,58%	13	4,50%	6,99%
7	2,89%	6,45%			

(c) *Modèle FAR*

SSB (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=50	*SF=100*		*SF=50*	*SF=100*
1	6,38%	17,09%	8	0,46%	21,29%
2	7,53%	13,94%	9	1,36%	15,79%
3	8,81%	14,66%	10	1,51%	26,24
4	0,20%	16,26%	11	5,38%	36,72%
5	0,49%	18,22%	12	5,79%	35,62%
6	0,89%	17,44%	13	1,10%	21,48%
7	5,18%	36,43%			

(d) *Modèle NLR, RNA, FAR*

TPC-DS (SF=20Go)						
Q_i	*Erreur(%)*			Q_i	*Erreur(%)*		
	NLR	*RNA*	*FAR*		*NLR*	*RNA*	*FAR*
1	7,09%	18,78%	13,53%	11	-	10,81%	17,88%
2	11,12%	20,78%	18,89%	12	7,85%	11,84%	11,29%
3	31,87%	26,47%	29,98%	13	23,93%	16,18%	13,95%
4	-	1,29%	23,66 %	14	-	8,59%	13,57%
5	2,62%	12,99%	8,09%	15	14,54%	5,10%	9,30%
6	2,72%	8,83%	1,57%	16	1,93%	5,03%	5,61
7	1,61%	7,43%	2,92%	17	1,38%	6,72%	5,45%
8	2,24%	4,55%	5,24%	18	9,47%	2,02%	13,21%
9	1,00%	8,14%	3,62%	19	1,96%	12,96%	7,96
10	23,12%	14,74%	18,51%	20	5,20%	16,81%	11,64

4.7.3.3.1 Résultats avec TPC-H

Dans cette expérimentation, nous utilisons le benchmark TPC-H avec un facteur d'échelle de 6 Go et 5 Go. Les résultats de l'exécution des requêtes pour chaque technique sont présentés dans le tableau 4.12 pour les deux facteurs de données.

Les résultats expérimentaux sont acceptables, comme nous pouvons le constater à partir du sous-tableau 4.12a, l'erreur moyenne est 1,32% et 1,12% sur les deux jeux de données (6 Go et 5 Go respectivement) en utilisant la technique de la régression non linéaire et l'erreur d'estimation maximale faite est inférieure à 11% dans les deux cas. En utilisant la technique des réseaux de neurone, l'erreur moyenne est 2,41% et 3,53% sur 6 Go et 5 Go respectivement et l'erreur maximale observée est inférieure à 14% (sous-tableau 4.12b). L'erreur moyenne est 1,34% et 1,13% sur 6 Go et 5 Go respectivement et l'erreur maximale est inférieure à 11% en adoptant la technique des forêts aléatoires (sous-tableau 4.12c). Ces résultats expérimentaux révèlent la stabilité et la précision d'estimation de notre modèle avec le benchmark TPC-H sur des quantités de données non conséquentes. En comparant les moyennes des erreurs d'estimation des trois techniques, nous constatons que la technique des forêts aléatoires et celle de la régression non linaire sont plus précises que la technique des réseaux de neurones sur les deux jeux de données.

4.7.3.3.2 Résultats avec TPC-DS

Pour évaluer notre modèle avec le benchmark TPC-DS, nous avons développé un outil en Java pour réécrire les données générées afin de pouvoir le charger dans le système Hyrise. Après la réécriture de certaines requêtes, nous avons exécuté 11 requêtes parmi les 99 requêtes du benchmark TPC-DS. Les résultats expérimentaux en utilisant nos techniques d'apprentissage sur une taille de 10 Go de données sont donnés dans le tableau 4.13.

La moyenne d'erreur minimale observée est 1,95% faite en utilisant la technique de forêt aléatoire et l'erreur maximale est inférieure à 15% pour toutes les techniques. Cette expérience révèle la stabilité de

TABLEAU 4.10 – Taux d'erreurs d'estimation d'énergie des requêtes (Monet - TPC-H)

(a) *Modèle NLR*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=100	*SF=200*		*SF=100*	*SF=200*
1	-	-	12	2,67%	2,54%
2	10,90%	20,92%	13	6,41%	39,67%
3	13,55%	17,53%	14	12,16%	2,09%
4	3,12%	9,51%	15	-	-
5	0,34%	19,65%	16	-	31,72%
6	9,89%		17	14,42	27,13%
7	3,89%	-	18	1,63%	
8	16,20%	10,44%	19	5,57%	-
9	3,71%	22,03%	20	22,09%	-
10	17,09%	3,28%	21	0,42%	28,76%
11	9,99%	18,75%	22	16,79%	28,43%

(b) *Modèle RNA*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=100	*SF=200*		*SF=100*	*SF=200*
1	0,87%	14,52%	12	8,89%	2,22%
2	11,56%	9,66%	13	4,70%	34,43%
3	10,97%	7,43%	14	9,87%	0,31%
4	8,33%	10,67%	15	4,58%	10,62
5	7,13%	19,71%	16	5,88%	17,14%
6	10,89%		17	7,70%	11,28%
7	6,96%	5,94%	18	5,53%	
8	8,79%	0,25%	19	8,63%	6,61
9	5,91%	22,01%	20	6,09%	31,70
10	6,61%	1,01%	21	5,77%	30,77%
11	10,02%	28,94%	22	2,19%	29,43%

(c) *Modèle FAR*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=100	*SF=200*		*SF=100*	*SF=200*
1	4,49%	12,73%	12	10,77%	0,63%
2	5,69%	14,64%	13	3,05%	35,65%
3	5,86%	8,79%	14	10,80%	0,83%
4	5,46%	12,08%	15	0,67%	8,72%
5	4,67%	11,85%	16	9,53%	19,91%
6	5,47%		17	6,54%	12,60%
7	0,76%	1,97%	18	2,97%	
8	3,65%	0,49%	19	6,07%	2,68%
9	0,64%	23,26%	20	1,53%	-
10	1,70%	1,25%	21	1,11%	27,34%
11	10,97%	7,78%	22	5,49%	18,86%

(d) *Modèle NLR, RNA et FAR*

TPC-DS (SF=50Go)								
Q_i	*Erreur(%)*			Q_i	*Erreur(%)*		
	NLR	*RNA*	*FAR*		*NLR*	*RNA*	*FAR*
1	-	17,02%	1,43%	12	6,10%	1,49%	0,90%
2	11,11%	6,29%	3,17%	13	0,62%	8,80%	5,62%
3	13,68%	1,87%	2,57%	14	8,76%	29,84%	4,80%
4	4,25%	4,39%	1,41%	15	7,49%	13,06%	17,88%
5	15,14%	14,25%	1,22%	16	6,91%	7,60%	2,14%
6	22,27%	0,48%	0,21%	17	20,73%	23,46%	2,95%
7	3,95%	11,55%	8,05%	18	3,01%	18,25%	0,03%
8	5,57%	4,67%	3,92%	19	25,40%	20,64%	7,35%
9	16,86%	22,47%	2,89%	20	14,02%	22,67%	15,41%
10	20,90%	15,47%	9,76%	21	4,93%	9,62%	8,67%
11	2,32%	10,14%	10,16%				

notre modèle de coût face à un nouveau schéma plus complexe que le TPC-H. Ici également comme dans le cas du benchmark TPC-H, la technique des forêts aléatoires (FAR) se révèle être la plus précise suivie par la technique de régression non linéaire (NLR).

4.7.3.3.3 Résultats avec SSB

Nous évaluons notre modèle avec le benchmark SSB en exécutant les treize requêtes sur un jeu de 10 Go. Les résultats expérimentaux sont présentés dans le tableau 4.13a.

À la suite de l'analyse de ces résultats, l'erreur moyenne la plus faible est 3,99% faite en utilisant la technique des forêts aléatoires et l'erreur maximale commise est 25,57% en utilisant la technique des réseaux de neurone. La technique des forêts aléatoires et celle de la régression non linéaire estiment mieux la puissance énergétique que la technique des réseaux de neurone dans cette série d'expérimentation de 10 Go de données du benchmark SSB.

4.7.4 Discussion

Nous résumons sur la figure 4.38 les erreurs moyennes obtenues à partir des expérimentations menées sur les trois SSDs à l'aide des trois techniques d'apprentissage à savoir la régression non linéaire, les réseaux de neurone et la forêt aléatoire. Nous obtenions à l'issu de ces expérimentations dans la majorité des cas que l'énergie estimée est très proche des valeurs réelles mesurées à l'aide d'un power-mètre sur

TABLEAU 4.11 – Taux d'erreurs d'estimation d'énergie des requêtes (Monet - SSB)

(a) *Modèle NLR*

SSB (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=100	*SF=200*		*SF=100*	*SF=200*
1	-	-	8	12,94%	-
2	29,15%	17,45%	9	0,87%	5,29%
3	18,76%	2,12%	10	18,87%	31,94%
4	2,27%	2,21%	11	0,59%	0,67%
5	21,40%	3,30%	12	4,05%	1,02%
6	9,08%	1,09	13	-	-
7	9,65%	1,51%			

(b) *Modèle RNA*

SSB (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=100	*SF=200*		*SF=100*	*SF=200*
1	0,98%	5,52%	8	9,04%	0,66%
2	10,51%	0,12%	9	4,15%	1,82%
3	8,57%	1,11%	10	8,53%	21,49%
4	5,55%	2,58%	11	7,13%	1,76%
5	17,70%	1,13%	12	5,32%	3,04%
6	23,64%	3,17%	13	2,60%	2,91%
7	8,57%	1,33%			

(c) *Modèle FAR*

SSB (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF=100	*SF=200*		*SF=100*	*SF=200*
1	2,70%	1,50%	8	8,64%	0,51%
2	25,07%	0,09%	9	4,18%	1,67%
3	26,53%	1,26%	10	25,70%	34,05%
4	6,55%	3,79%	11	7,41%	0,18%
5	22,01%	2,47%	12	5,39%	4,92%
6	17,03%	2,55%	13	2,52%	2,82%
7	8,09%	1,75%			

les benchmarks TPC-H, TPC-DS et SSB. Ces résultats expérimentaux démontrent la stabilité et la précision de nos modèles à être exploitable dans des études d'optimisation ou d'estimation énergétiques. En analysant la précision des valeurs estimées par nos modèles en variant la méthode de machine learning, nous constatons que l'erreur d'estimation moyenne de la puissance faite par la technique de forêt aléatoire (FAR) et celle de la régression non linéaire (NRL) sont plus performantes lorsque la quantité des données n'est pas conséquente (cas du système Hyrise et des autres systèmes sur des jeux de données de taille faible), tandis que les réseaux de neurones sont plus précis lorsque les données augmentent et cela a tendance à s'améliorer. Les avantages des forêts aléatoires et de la régression non linéaire sont qu'ils sont plus simples à entraîner et à optimiser en fonction de leurs hyper-paramètres [134]. De plus leur coût de calcul et leur temps d'apprentissage sont relativement faibles alors que les réseaux de neurones ont généralement besoin de plus de données et de temps. Néanmoins, en comparant le cumul des erreurs d'estimation des requêtes sur les différents benchmarks, la moyenne d'erreur obtenue avec la technique des réseaux de neurone et la technique des forêts aléatoires offre une plus grande précision que celle de la régression non linéaire en considérant toutes les expérimentations menées.

La modélisation de la consommation énergétique est le sous-bassement dans le processus de conception des SSDs éco-énergétiques. Les modèles de machine learning utilisés dans la littérature pour prédire la consommation d'énergie des requêtes dans les BDs sont principalement divisés entre les techniques de la régression linéaire et de la régression non linéaire. Notre Framework offre une nouvelle approche en plus de tenir compte de nouveaux paramètres pertinents, nous utilisons de nouvelles techniques de machine learning pour entraîner nos modèles de prédiction. En comparant les erreurs moyennes, nous avons constaté que la technique des réseaux de neurones et celle des forêts aléatoires peuvent offrir de meilleures performances.

TABLEAU 4.12 – Taux d'erreurs d'estimation d'énergie des requêtes (Hyrise - TPC-H)

(a) *Modèle NLR*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF= 6	*SF= 5*		*SF= 6*	*SF= 5*
1	3,50%	0,68%	12	0,64%	0,50%
2	0,33%	0,11%	13	0,57%	1,11%
3	0,23%	0,06%	14	0,47%	0,44%
4	0,63%	0,18%	15	5,48%	0,12%
5	0,16%	0,06%	16	2,09%	4,68%
6	0,01%	0,59%	17	5,74%	0,93%
7	0,17%	0,26%	18	4,21%	0,02%
8	0,26%	0,25%	19	0,31%	1,04%
9	1,37%	1,33%	20	0,75%	0,56%
10	0,25%	0,25%	21	0,37%	0,61%
11	0,68%	0,24%	22	0,91%	10,60%

(b) *Modèle RNA*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF= 6	*SF= 5*		*SF= 6*	*SF= 5*
1	8,29%	11,46%	12	0,20%	0,93%
2	0,77%	0,54%	13	0,63%	11,01%
3	0,67%	0,37%	14	0,03%	0,87%
4	1,07%	0,61%	15	6,07%	0,30%
5	0,26%	0,37%	16	5,47%	13,07%
6	0,44%	1,02%	17	6,14%	0,49%
7	0,26%	0,69%	18	7,47%	12,22%
8	0,17%	0,68%	19	0,74%	0,60%
9	10,67%	10,70%	20	1,18%	0,12%
10	0,17%	0,68%	21	0,79%	0,17%
11	0,25%	0,67%	22	1,33%	10,12%

(c) *Modèle FAR*

TPC-H (SF en Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	SF= 6	*SF= 5*		*SF= 6*	*SF= 5*
1	3,31%	0,38%	12	0,85%	0,37%
2	0,39%	0,22%	13	1,68%	0,88%
3	1,15%	0,11%	14	0,47%	0,31%
4	0,86%	0,32%	15	5,45%	0,48%
5	0,14%	0,05%	16	2,65%	4,35%
6	0,34%	0,25%	17	5,73%	0,02%
7	0,19%	0,20%	18	2,25%	2,32%
8	0,82%	0,02%	19	0,01%	1,06%
9	0,88%	0,92%	20	0,34%	0,89%
10	0,25%	0,14%	21	0,03%	0,90%
11	0,82%	0,11%	22	0,94%	10,61%

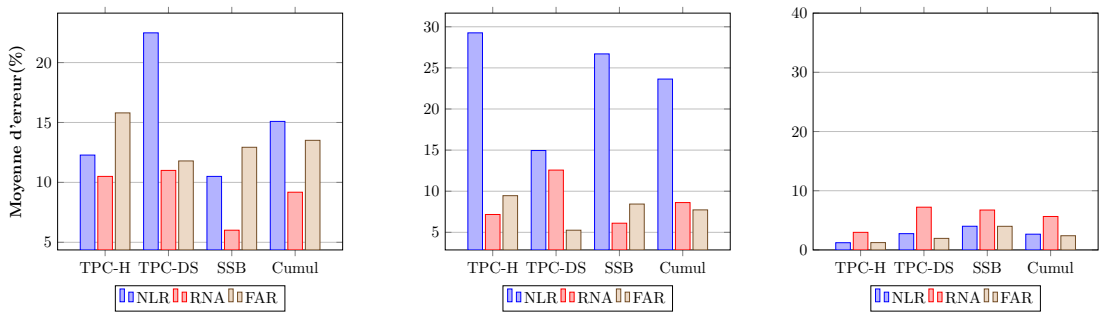
TABLEAU 4.13 – Taux d'erreurs d'estimation d'énergie des requêtes (Hyrise - SSB/TPC-DS)

(a) *Benchmark SSB*

SF=10 Go			
Q_i	*Erreur(%)*		
	NLR	*RNA*	*FAR*
1	1,84%	1,91%	2,19%
2	2,18%	2,61%	3,66%
3	2,45%	2,87%	2,58%
4	3,49%	8,27%	1,27%
5	3,31%	8,53%	3,19%
6	3,04%	3,46%	3,46%
7	5,42%	6,13%	3,81%
8	3,04%	8,65%	0,74%
9	2,77%	2,84%	2,81%
10	2,75%	3,17%	4,14%
11	5,40%	6,19%	5,81%
12	4,30%	7,40%	2,84%
13	12,07%	25,57%	15,33%

(b) *Benchmark TPC-DS*

SF=10 Go			
Q_i	*Erreur(%)*		
	NLR	*RNA*	*FAR*
1	0,94%	0,87%	1,03%
2	0,64%	12,85%	2,91%
3	0,49%	12,80%	0,81%
4	14,85%	12,44%	3,76%
5	1,74%	2,18%	1,62%
6	0,04%	12,16%	2,28%
7	0,34%	12,30%	3,64%
8	0,02%	0,46%	0,13%
9	10,07%	11,80%	3,18%
10	0,42%	0,48%	0,79%
11	0,65%	1,09%	1,34%



(a) Évaluation du modèle de PostgreSQL

(b) Évaluation du modèle de MonetDB

(c) Évaluation du modèle de Hyrise

FIGURE 4.38 – La moyenne d'erreur d'estimation des trois systèmes.

4.8 Conclusion

Dans ce chapitre, nous avons décrit notre méthodologie de construction d'un modèle de coût pour estimer la consommation d'énergie lors de l'exécution d'une requête SQL. Notre proposition s'articule autour de quatre grandes étapes : (1) un audit approfondi permettant de comprendre le fonctionnement du STR dans le SGBD cible, (2) identification des paramètres pertinents sensibles à l'énergie, (3) élaboration d'un modèle de coût mathématique pour estimer l'énergie consommée lors de l'exécution d'une requête et (4) validation du modèle en évaluant les erreurs d'estimation. À la suite des modèles proposés en suivant notre méthodologie de conception, nous menons une série d'expérimentation pour démontrer l'exactitude et l'efficacité de nos propositions pour chaque système (PostgreSQL, MonetDB et Hyrise) sur des BDs réelles, en exécutant un ensemble de requêtes générées à partir des benchmarks TPC-H, TPC-DS et SSB. Nous avons également varié la technique de machine learning pour juger de leur impact sur la qualité du modèle final retenu, pour ce fait nous avons utilisé trois techniques soit la régression non linéaire, les réseaux de neurones et la forêt aléatoire. Nos résultats expérimentaux révèlent que les puissances estimées sont très proches des valeurs réelles mesurées pour chacune des différentes techniques d'apprentissage sur les différents systèmes. En comparant la moyenne des erreurs d'estimation obtenues à l'aide des trois techniques d'apprentissage, nous constatons que la technique des réseaux de neurone donne de meilleure prédiction que la technique de forêt aléatoire et celle de la régression non linéaire sur le système MonetDB et PostgreSQL. Sur le système Hyrise, la technique de forêt aléatoire se révèle être la meilleure.

CHAPITRE 5

Frameworks pour le déploiement de nos modèles de coût énergétique

Where there is discord, may we bring harmony. Where there is error, may we bring truth. Where there is doubt, may we bring faith. And where there is despair, may we bring hope.

— Margaret Thatcher

Sommaire

5.1 Introduction 119
5.2 Optimisation énergétique dans le module de traitement des requêtes 119
5.2.1 Influence des différentes étapes de traitement 119
5.2.1.1 Analyse syntaxique 119
5.2.1.2 Réécriture 119
5.2.1.3 Plan d'exécution 120
5.2.1.4 Exécution 120
5.2.2 Intégration avec PostgreSQL 121
5.2.2.1 Modèle de coût 121
5.2.2.2 Plan d'évaluation 123
5.2.2.3 Architecture fonctionnelle 123
5.2.2.4 Résultats expérimentaux 126
5.2.3 Intégration avec MonetDB 128
5.2.3.1 Modèle de coût 129
5.2.3.2 Architecture fonctionnelle 130
5.2.3.3 Résultats expérimentaux 131
5.3 Estimation dynamique de l'énergie lors du traitement des requêtes (Simulation) 132
5.3.1 Architecture fonctionnelle 133
5.3.2 Résultat expérimentaux 133
5.4 Conclusion 135

Dans ce chapitre, nous introduisons nos outils implémentés au-dessus des trois SSDs relationnels (PostgreSQL, MonetDB et Hyrise) dans le but de proposer des systèmes écologiques (vert) intégrant la dimension énergétique dans le processus de traitement des requêtes. Après avoir analysé le module de traitement des requêtes de chaque système afin d'identifier les points d'exploitation possible de la dimension énergétique, nous proposons trois Frameworks (un pour chaque système) qui intègrent nos modèles de coût dans le processus de traitement des requêtes.

5.1 Introduction

En utilisant les modèles de coût développés, nous explorons expérimentalement dans ce chapitre les points de conservation énergétique identifiés lors de la phase d'audit des SSDs étudiés dans le chapitre précèdent. L'exploration de ces points d'intégration énergétique est conduite en ayant deux objectifs principaux: (i) sélection des plans d'exécution dont le coût énergétique est moindre lors de l'optimisation de la requête; et (ii) quantification du coût énergétique d'une charge de requête afin de sensibiliser les utilisateurs de l'impact de leur choix de conception sur l'énergie totale consommée. Pour ce faire, nous présentons en détail pour chaque étape du processus de traitement, son influence sur l'énergie totale consommée puis nous donnons les descriptions de certains fichiers appartenant aux noyaux (codes sources) des SSDs étudiés qu'il faudrait modifier afin d'intégrer la dimension énergétique. Compte tenu de la variété de nos SSDs étudiés, nous proposons trois Frameworks (un pour chaque système) qui intègrent nos modèles de coût dans le processus de traitement des requêtes. Pour le système PostgreSQL, nous implémentons l'outil «GreenPipeline» inspiré des travaux de [156] [186] développé au-dessus de la version offrant le mode intra-parallèle. Nous proposons également deux outils «EcoQMPro» et «SimHyriseEco» construit respectivement au-dessus des systèmes MonetDB et Hyrise afin d'estimer dynamiquement en ligne la consommation énergétique d'une charge de requête donnée. Nous évaluons nos différents frameworks pour quantifier nos économies énergétiques et de puissances à l'aide des différentes charges de travail générées à partir des benchmarks TPC-H et SSB.

5.2 Optimisation énergétique dans le module de traitement des requêtes

Dans le but de concevoir un module de traitement éco-énergétique, les différentes étapes de traitement d'une requête ont été explorées pour des études de conservation de la puissance électrique. Pour mémoire, le module de traitement de requête est principalement constitué de quatre étapes comme illustré sur la figure 2.3, nous discutons dans cette section l'influence énergétique de chacune des étapes sur la consommation énergétique globale.

5.2.1 Influence des différentes étapes de traitement

5.2.1.1 Analyse syntaxique

Lors de cette phase, le système après avoir vérifié que la requête formulée est correcte, construit un « arbre de syntaxe ». L'arbre syntaxique est ensuite transformé en une structure de données appelée « arbre de requête ». Cette étape du module de traitement se termine assez rapidement et n'invoque pas une consommation accrue des ressources du système. Par conséquent cette étape ne constitue pas un point qui provoque une consommation excessive de l'énergie toute fois des techniques d'optimisation énergétique basées sur l'analyse de la complexité algorithmique peuvent faire l'objet des études. Les fonctions responsables de l'analyse syntaxique et de la transformation dans PostgreSQL sont « raw_-parser() » et « parse_analyze() ». Dans MonetDB, elles sont réalisées par les fonctions « sqlparse() », « query_create() » et « rel_semantic() ».

5.2.1.2 Réécriture

Le module de la réécriture consiste à appliquer les règles de transformation algébrique sur « l'arbre de requête » en un autre « arbre » équivalent mais plus « optimal ». Cette étape de simplification influence considérablement la consommation des ressources lorsque la requête fournie est assez complexe. Elle est disponible dans l'immense majorité des moteurs de BD et leur application dans la chaîne de traitement se termine rapidement car il s'agit de changer l'ordre d'exécution de certaines opérations. La fonction de réécriture est nommée « QueryRewrite() » définie dans le fichier «rewriteHandler.c» dans le cas de PostgreSQL. Pour MonetDB, les fonctions assurant cette tâche sont définies dans le fichier « rel_optimizer.c». Pour Hyrise, tous les fichiers concernant l'application des règles de transformation sont dans le dossier « /src/lib/optimizer/strategy ».

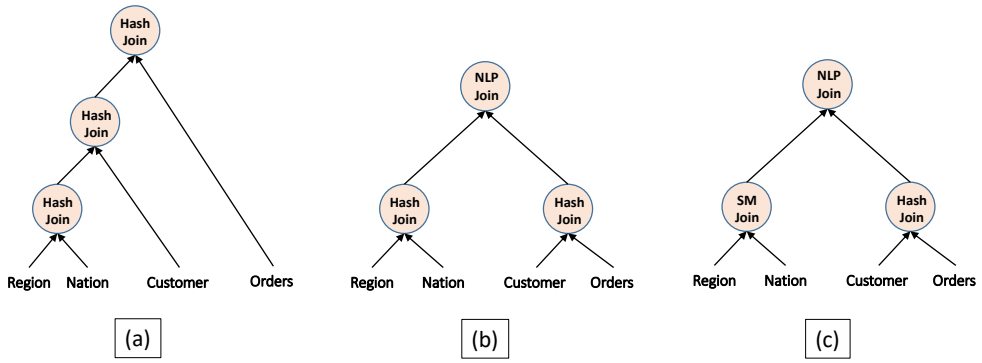


FIGURE 5.1 – Différents Plans d'évaluation d'une requête conduisant au même ensemble de données.

5.2.1.3 Plan d'exécution

Il incombe à l'optimiseur de trouver un plan ayant une performance acceptable en utilisant des techniques comme celle basée sur les modèles de coût dans le cas de PostgreSQL. Les SGBD traditionnels utilisent un modèle de coût orienté performance pour estimer le coût de chaque plan, et en choisisse celui qui est le plus meilleur.La figure 5.1 montre par exemple trois plans d'exécution différents d'une requête du benchmark TPC-H. Chaque plan sur la figure 5.1 représente une approche d'évaluation de cette requête. Pour évaluer la performance de chaque plan, des fonctions de coût ont été définis pour chaque opérateur SQL. La formule générale pour estimer le coût d'un opérateur « op » peut être exprimée comme:

$$C_{op} = \alpha \times C_{CPU} \oplus \beta \times C_{memoire} \oplus \gamma \times C_{reseau}.$$

Où α, β, γ représentent les coefficients utilisés pour convertir les estimations en une l'unité de grandeur souhaitée (par exemple le temps, l'énergie, coût monétaire). $\oplus$ désigne le type de relation entre les paramètres. Typiquement, le coût de l'accès à la mémoire disque dans les SGBD traditionnels est prédominant et se calcule en comptant le nombre de page lu (et écrit) et le nombre de recherche. Dans l'optimiseur de PostgreSQL, le coût d'une l'opération de la requête est estimé comme suit:

$$C_{op} = ((\alpha|\beta|\lambda) + \phi) \times N_{tuples} + (\rho|\mu)N_{pages}$$

Où N_{tuples} représentent le nombre de tuples et N_{pages} le nombre de page nécessaire pour exécuter l'opération « op ». Les descriptions et les valeurs par défaut des coefficients (α,β,λ, phi, ρ, μ) des coûts de performance sont donnés dans le tableau 5.1. Le coût du plan de la requête est estimé en cumulant le coût de chaque opérateur du plan.

Variables	Description	coût(valeur)
α	Coût CPU pour traiter un tuple	0.01
β	Coût CPU pour traiter un tuple via un accès index	0.005
λ	Coût CPU pour traiter un tuple Parallèle	0.1
phi	Coût CPU pour effectuer une opération	0.0025
ρ	Coût d'E/S séquentiel à une page	1.0
μ	Coût d'E/S aléatoire à une page	4.0

TABLEAU 5.1 – Valeurs des paramètres du modèle de coût de PostgreSQL.

Ce mécanisme de sélection du plan d'exécution basé sur la performance peut être entendu pour prendre en compte la contrainte énergétique dans le but de réaliser des optimisations énergétiques. Pour atteindre cet objectif, l'optimiseur doit pouvoir intégrer et estimer le coût énergétique de chaque plan. En se basant sur nos modèles de coût énergétique, un modèle d'évaluation des plans des requêtes est conçu pour permettre à l'optimiseur de sélectionner un plan qui répond aux besoins spécifiques de l'utilisateur (Temps vs Énergie).

5.2.1.4 Exécution

La dernière étape du module de traitement consiste à l'exécution du plan crée par le « planificateur/optimiseur » en utilisant un mode d'exécution généralement parallèle (interopération/ interopération)

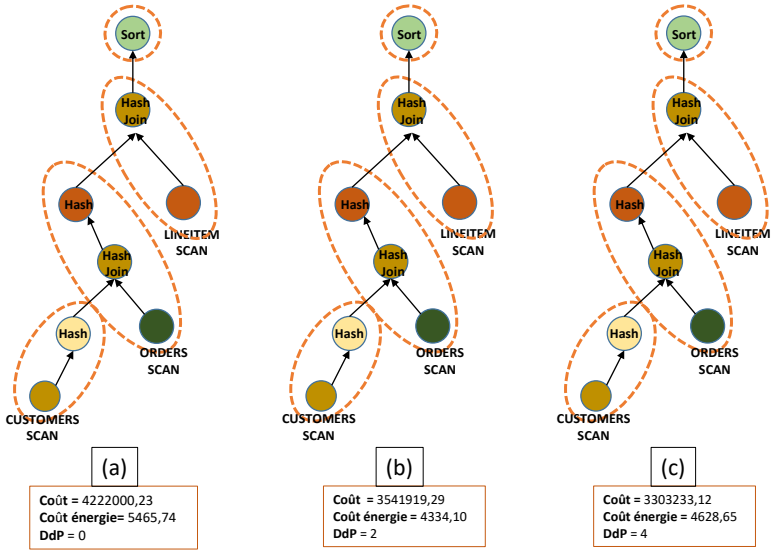


FIGURE 5.2 – Coût des différents plans d'exécution de la requête Q_3 en variant le degré de Parallélisme de 0 à 4.

sur les récentes architectures matérielles. Le mode de traitement impacte considérablement la consommation énergétique des plans d'exécution. Nous illustrons cet impact en considérant par exemple le plan d'exécution de la requête Q_3 du benchmark TPC-H. Comme le montre la figure 5.2, en variant la valeur du degré de parallélisme de 0 à 4, nous constatons que la valeur du DdP influence le coût d'exécution estimé par l'optimiseur et de l'énergie.

5.2.2 Intégration avec PostgreSQL

Cette section présente les composants et fonctionnalités importants du Framework *GreenPipeline*. Notre Framework est constitué de trois parties principales, à savoir l'extension du module de traitement des requêtes pour intégrer le modèle de coût estimant l'énergie des opérations du plan, le modèle d'évaluation des plans pour permettre à l'optimiseur de sélectionner un plan qui répond aux besoins spécifiques de l'utilisateur et une interface graphique pour la configuration et la visualisation des paramètres.

5.2.2.1 Modèle de coût

Partant de nos modèles définis dans le chapitre 4 pour estimer le coût énergétique du plan. Pour une requête SQL donnée, l'optimiseur de requête est responsable de l'estimation des coûts d'E/S et de CPU pour chaque opérateur algébrique de base. Notre processus d'intégration du modèle de coût énergétique dans le module du traitement d'une requête est implémenté en suivant le même mécanisme que celui de la performance. Les paramètres sont d'abord estimés puis convertis en énergie par des unités énergétiques pour chaque opérateur en utilisant nos modèles de coût.

(i) **Balayage séquentiel:** il recherche dans toutes les page de la relation correspondante l'ensemble des tuples qui satisfont un prédicat donné. Le coût d'E/S est égal au nombre de pages dans la relation (p) et le coût CPU est égal au nombre de tuples (n). Ainsi, l'estimation du coût énergétique partant de nos modeles est: $W_1 \times n \oplus W_2 \times p$, avec $\oplus$ la combinaison entre les deux grandeurs, W_1 et W_2 sont des constantes énergétiques.

(ii) **Parours par index:** il est similaire au balayage séquentiel sauf qu'il utilise un index (généralement sous forme d'arbre B) pour réduire le nombre de tuples à lire. Le nombre de tuples est déterminé par le nombre de tuples dans l'index multiplié par l'index de selectivité.

(iii) **Parcours par bitmap:** il recherche le fichier index en utilisant son index binaire, qui est basé sur des tableaux de bits (communément appelés bitmaps) de colonne [11, 10]. Puis le résultat est trié en utilisant une structure binaire. Le parcours par bitmap est basé sur le parcours par index, les coûts d'E/S et de CPU sont donc calculés de la même manière.

(iv) **Parcours séquentiel parallèle:** Le parcours séquentiel parallèle est similaire au balayage séquentiel à part que les pages de la table seront divisées entre les workers (cœurs) participant au parcours. Les pages sont retournées une à la fois, afin que l'accès à la table reste séquentiel. Le coût du CPU d'un workers est

$\frac{n}{(DdP+(1-leader_{contribution}\times DdP))}$, avec n le numbre de tuple dans la table et DdP le nombre de workers (degré de parallelisme).

(v) **Parcours de bitmap parallèle:** Lors de ce parcours un processus est choisi pour être le master. Ce dernier effectue le parcours d'un ou plusieurs index et crée un bitmap indiquant les pages de la table devant être lues. Ces pages sont alors divisées entre les workers participant au parcours. En d'autres termes, le parcours de la table est effectué en parallèle, mais le parcours d'index associé ne l'est pas.

(vi) **Parcours d'index parallèle:** Lors ce parcours les workers participant au parcours se relaient pour lire les données depuis l'index. Chaque processus peut demander une seule page de l'index à la fois puis la parcours pendant ce moment les autres processus peuvent être occupés à d'autres tâches. Les résultats d'un parcours d'index parallèle sont retournés triés au sein de chaque worker.

(vii) **Tri:** Le tri est à la fois une opération gourmande en CPU et en mémoire à cause de la nécessité de trier les données en plusieurs étapes. Si le tri peut se faire en mémoire sans nécessité l'écriture des données alors PostgreSQL utilise l'algorithme de tri rapide, où le coût d'E/S est nul. Si non, le coût de matérialisation et la relecture des données sont ajoutés au coût d'E/S. Le coût CPU est log(n), avec n le nombre de tuple de la relation.

(viii) **Agrégation:** Les fonctions d'agrégation calculent une valeur unique à partir d'un ensemble de valeurs en entrée. Pour cela une table de hachage temporaire est utilisée pour grouper les tuples avant d'appliquer la fonction d'agrégation. Le coût CPU est le nombre de tuples à grouper.

(ix) **Jointure séquentiel:** Généralement, les opérations de jointure impliquent seulement deux tables à la fois. Si une requête a plus de deux tables à joindre alors la jointure se réalise séquentiellement: deux tables, puis le résultat intermédiaire avec la table suivante. Pour toute jointure (table d'origine ou temporaire), la performance et la consommation d'énergie dépendent fortement de l'algorithme de jointure utilisé.

- **Jointure par boucle imbriquée:** Cette implémentation joint deux tables dite table interne (inner) et table externe (outer). La relation externe doit être la plus petite afin de réduire le nombre d'itérations. Le coût d'E/S dans ce type de jointure est la somme du coût de la relation inner et outer, le coût CPU est la multiplication de nombres de tuples dans chaque relation.
- **Jointure par hachage:** Cet algorithme comprend deux phases, une première phase où les données sont partitionnées en des fragments en utilisant la même fonction de hachage et une autre phase pour la jointure. Le coût d'E/S est le coût de lecture des deux relations, et le coût CPU est la somme des coûts lors de la construction de la table de hachage plus le coût de comparaison des tuples qui satisfont la clause de jointure.
- **Jointure par tri-fusion:** Cette implémentation est utilisée uniquement lorsque les deux relations sont triées sur l'attribut de jointure. Le coût d'E/S est la somme du coût de la relation inner et outer, et le coût CPU est la somme des coûts pour trier les tuples des deux relations.

Les formules estimant les coûts des E/S et du CPU pour chaque opération basique sont données dans le tableau 5.2 avec la description des symboles dans le tableau 5.3.

Paramètre	Coût d'E/S	Coût de CPU
Balayage séquentiel	p_{seq}	t_{seq}
Parcours par index	p_{index}	$t_{index} \times sel$
Parcours par bitmap	p_bitmap	$t_bitmap \times sel$
Jointure par boucle imbriquée	$p_{outer} + p_{inner}$	$t_{outer}.t_{inner}$
Jointure par tri-fusion	$p_{outer} + p_{inner}$	$t_{sort(outer)} + t_{sort(inner)}$
Jointure par hachage	$p_{outer} + p_{inner}$	$t_{outer} \times n_{hash} + t_{inner} \times p_{hash}$
Tri	$p_{sort}; s < m; 0 sinon$	$t_{sort} \times \log_2 t_sort$
Agrégation	0	t_{agg}
Groupement	0	$t_{group} \times n_{group}$

Tableau 5.2 – Description des paramètres du modèle de coût.

Pour l'estimation du nombre de blocs lus à partir de la mémoire principale (mémoire cache) lors du traitement de la requête, nous nous referons aux approches proposés par [111] pour la conception des modèles de coût dans les BDs en mémoire. Ils modélisent le coût de la mémoire en

Paramètres	Description
m	taille de la mémoire tampon pour l'opération de tri
$block$	taille d'une page dans SGBD
T_i	la taille de la table i
t_i	nombre de tuples d'entrée pour l'opérateur i
p_i	nombre de pages d'entrée pour l'opérateur i
sel	sélectivité d'index
s	la taille de la relation d'entrée pour l'opérateur de tri
n_{hash}	nombre de clauses de hachage en phase de construction
n_{hash}	nombre de partitions de hachage en phase de sondage
$p_{outer/inter}$	nombre de pages pour l'opérateur de jointure
$t_{outer/inter}$	nombre de tuples pour l'opérateur de jointure
n_{group}	nombre de colonnes de groupement

TABLEAU 5.3 – Description des paramètres du modèle de coût.

estimant le nombre de défauts caches multipliés par la latence d'accès à la mémoire. En supposant que R décrit une relation constituée de $R.p$ pages chacune ayant une taille de $R.w$, la taille de la relation R est alors dénotée par $||R|| = R.p \times R.w$. Soient C_i et B_i la capacité de la mémoire ou du cache i et la taille du bloc de cache. Le nombre de pages pouvant tenir en mémoire est exprimée par $|C|_p = \frac{C_i}{R.w}$ et le nombre de ligne de cache pour mémoriser l'ensemble des pages de R est : $\frac{||R||}{B_i}$. Nous exploitons ces formules pour estimer le nombre maximum de blocs qui peuvent résider dans la mémoire cache pour une relation donnée lors de déploiement de nos modèles.

5.2.2.2 Plan d'évaluation

Le modèle d'évaluation des plans fournit une approche pour évaluer la performance des plans d'une requête. À chaque niveau, les plans possibles sont évalués et les meilleurs plans sont conservés. Au lieu de choisir un plan avec une performance optimale, un modèle de coût énergétique peut être utilisé pour choisir celui qui économise l'énergie ou une méthode d'évaluation pour définir un seuil de compromis entre l'énergie et le temps. Le compromis entre le temps et l'énergie est fait sur la base d'un critère qui reflète la supériorité de la performance (T) ou du coût énergétique (P). Ainsi le modèle d'évaluation permettant de trouver un tel plan peut être formulé comme:

$$C_{plan} = \alpha \times log(T) + (1 - \alpha) \times log(P). \tag{5.1}$$

C_{plan} représente le coût d'un plan. α est un coefficient qui reflète le niveau de compromis entre les deux grandeurs T et P. La valeur du coefficient varie entre 0 à 1. Avec ce modèle d'évaluation, il est alors possible de sélectionner des plans ayant des objectifs d'optimisation différents en régulant la valeur du critère α.

5.2.2.3 Architecture fonctionnelle

Dans cette section, nous présentons la conception de l'outil « GreePipeline ». Cet outil propose un SSD écologique (vert) dont l'objectif est de considérer la dimension énergétique dans le processus de traitement des requêtes. Notre outil est composé de deux grandes parties indépendantes : une partie basée sur les modifications apportées dans le SGBD afin d'incorporer nos modèles de coût et le modèle d'évaluation des plans d'exécution et une partie interface graphique (GUI) permettant de définir les valeurs de certains paramètres et de visualiser en temps réel la consommation énergétique des requêtes. Cet outil a été développé dans un environnement $C++(Version1)$ puis en $Java(Version2)$. La $Version1$ est l'extension de l'outil proposé par [156] afin d'incorporer la notion de l'intra-parallélisme dans les plans générés et la mise à jour du modèle de coût. Le workflow de notre Framework est décrit sur la figure 5.3.

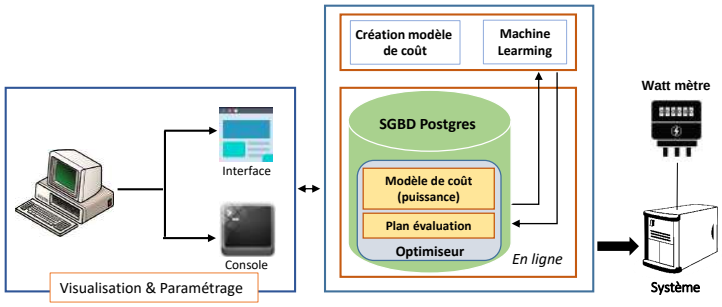


FIGURE 5.3 – Workflow du Framework GreenPipeline.

(a) **Modification dans PostgreSQL:**
Nous avons apporté des modifications sur certains fichiers du code source de PostgreSQL afin d'incorporer notre modèle de coût énergétique, le modèle d'évaluation des plans et quelques instructions permettant l'affichage de l'énergie dans le plan d'exécution. Ces modifications ont concerné les fichiers suivants:

— *«src/backend/optimizer/path/costsize.c:»* Ce fichier contient l'ensemble des fonctions de calcul d'estimation des coûts de chaque opérateur algébrique. Nous récupérons les coûts d'E/S et de CPU de chaque opérateur puis estimons sa puissance en utilisant une des trois approches (régression non linéaire, réseau de neurone, forêt aléatoire) qui implémente notre modèle de coût (par défaut la fonction count_power_cost_NLR() qui applique l'équation de la régression non linéaire est utilisée). Nous donnons sur la figure 5.4 le code de la fonction count_power_cost_NLR(), les B_i sont les coefficients de régression et DdP est le degré de parallélisme.

```
1
2         Cost
3         count_power_cost_NLR(double io_cost, double cpu_cost,int wh)
4         {
5           // io_cost : Cout du DISK
6           // cpu_cost: Cout du CPU
7           // wh: Degre de parallelisme
8
9           Cost power_cost = 0.0;
10          double fcd=wh*0.0552+0.00783;
11
12          if(wh!=0){
13            power_cost = B0
14            + pow(io_cost, 1) * pow(cpu_cost, 0) * B1 + pow(io_cost, 2) * pow(cpu_cost, 0) * B3
15            + (pow(io_cost, 0) * pow(cpu_cost, 1) * B2 + pow(io_cost, 0) * pow(cpu_cost, 1) * fcd * B2)
16            + (pow(io_cost, 0) * pow(cpu_cost, 2) * B4 + pow(io_cost, 0) * pow(cpu_cost, 2) * fcd * B4)
17            + (pow(io_cost, 1) * pow(cpu_cost, 1) * B5 + pow(io_cost, 1) * pow(cpu_cost, 1) * fcd * B5);
18          }
19          else {
20            power_cost = B0
21            + pow(io_cost, 1) * pow(cpu_cost, 0) * B1 + pow(io_cost, 0)* pow(cpu_cost, 1) * B2
22            + pow(io_cost, 2) * pow(cpu_cost, 0) * B3 + pow(io_cost, 0)* pow(cpu_cost, 2) * B4
23            + pow(io_cost, 1) * pow(cpu_cost, 1) * B5;
24
25          }
26          return power_cost;
27        }
```

FIGURE 5.4 – Vue des instructions de la fonction *count_power_cost_NLR()*.

— *«src/backend/optimizer/util/pathnode.c:»* Après que les coûts des opérateurs soient estimés, les fonctions définies dans ce fichier sont utilisées afin de comparer le coût d'exécution des plans. Nous modifions les fonctions compare_fractional_path_costs() et compare_path_costs() pour que la sélection du plan se fasse sur la base de notre modèle d'évaluation cumulant les deux contraintes (temporaire et énergétique).

— *«src/backend/commands/explain.c:»* Ce fichier englobe les fonctions permettant d'afficher le plan d'exécution associé à une requête en annotant les informations sur les coûts de la performance initiale, totale, la cardinalité (rows) et la taille des lignes en octets (width) de chaque opérateur dans le plan. Nous modifions ces instructions pour inclure l'affichage de la puissance de chaque opérateur.

Nous avons également apportés quelques modifications mais très mineures sur d'autres fichiers tel que: «postgresql.conf.sample», «guc.c», «allpaths.c», «copyfuncs.c», «createplan.c», «outfunc.c», «subselect.c». Dans le fichier de configuration «postgresql.conf.sample», nous avons principalement ajouté le critère de compromis en déclarant deux variables *time_wt* et *power_wt* initialisées à 1 et à 0 respectivement.

(b) **GUI:**
Le module GUI permet à l'utilisateur de définir les valeurs de certains paramètres, de visualiser la consommation énergétique de la requête en temps réel, de visualiser les plans d'exécution et de voir la comparaison des valeurs de prédiction de l'énergie en utilisant les trois techniques d'apprentissage. Elle n'est pas obligatoire pour utiliser le système sous-jacent, elle sert simplement à montrer les différentes interactions possibles avec ce dernier. Le module permet de faire les tâches suivantes:

— Configuration des valeurs des paramètres du système
— Visualisation du plan
— Visualisation de la chronologie de la consommation
— Visualisation des valeurs de prédiction de l'énergie
— Visualisation des résultats de la requête.

(a) **Configuration des paramètres:** Cette interface permet d'établir la connexion avec la BD. Dès que la connexion ait été établie, les statistiques sur le système et les méta données de la BD sont affichées dans la zone *«MetaData»*. La partie de gauche permet à l'utilisateur de fournir le chemin d'accès au driver du *Power-mètre* dans la zone *«PowerMeter»* et dans la zone *«Tradeoff»* de spécifier son désir de compromis entre la performance et la puissance. Nous avons ajouté dans la zone *«optimizer tuning»*, des options permettant à l'utilisateur de forcer l'optimiseur d'utiliser des algorithmes spécifiques pour l'exécution de la requête dans le but de pourvoir comparer les consommations énergétiques. Dans la zone *«Machine learning Technique»*, il est possible pour l'utilisateur de cocher sa ou ses techniques d'apprentissage. La figure 5.5 donne un aperçu de l'interface de paramétrage des deux versions ($C++$ en utilisant la bibliothèque Qt et *Java* en utilisant la bibliothèque AWT) que nous avons implémenté.

(b) **Visualisation du plan:** La figure 5.5 donne un aperçu de la fenêtre de préparation des requêtes. Cette fenêtre est initialisée à une simple requête. Les utilisateurs peuvent la modifier pour entrer soit une seule requête SQL ou une charge de requête à exécuter. Par défaut, le parallélisme est activé et le degré de parallélismes est à 2. L'exécution de la requête se déroule et les résultats sont affichés dans une table (widget) lorsque l'utilisateur clique sur le bouton *« run »*. Lors ou avant de lancer l'exécution de la requête, l'utilisateur peut regarder le plan d'exécution annoté avec diverses informations, telles que le coût total, la puissance énergétique, et les coûts d'E/S et de CPU pour chaque opérateur en cliquant sur le bouton *« Explain »*. Pour visualiser le graphe d'exécution non segmenté en pipeline, l'utilisateur doit cocher *« Display Graph »*. En cochant *« split pipeline »*, le graphe est divisé en un ensemble de pipeline dont les opérateurs sont regroupés avec la même couleur.

(c) **Chronologie de la consommation:** Lors du traitement de la requête, il est possible pour l'utilisateur de cliquer sur le bouton *« Power timeline »* pour afficher en temps réel la

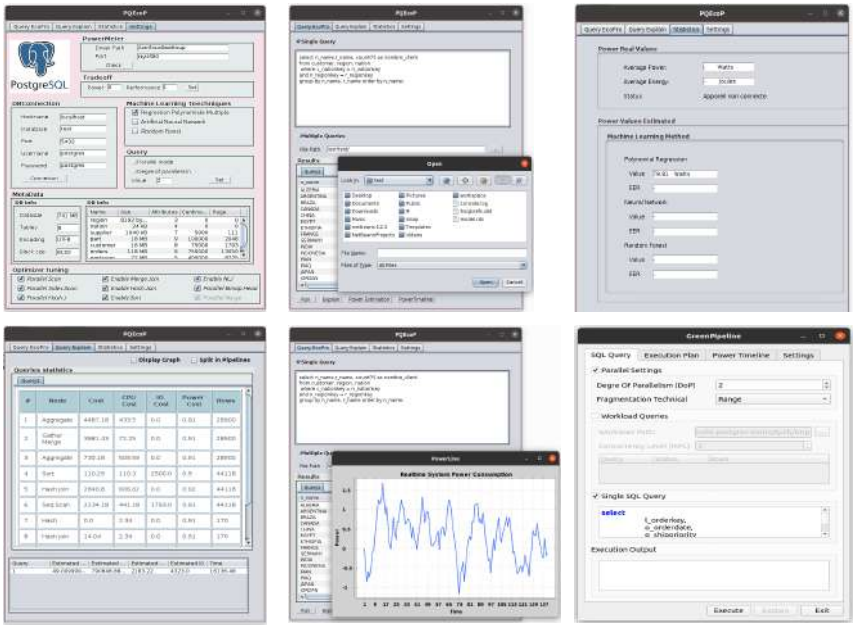


FIGURE 5.5 – Interfaces du framework (configuration des paramètres, de visualisation du plan, Prédiction de l'énergie du plan).

consommation d'énergie capturée par le *Power mètre*. Lorsque le wattmètre est connecté, les valeurs énergétiques sont calculées puis sont comparées à la fin de l'exécution de la requête avec les valeurs d'estimation obtenues à l'aide des trois techniques d'apprentissage. De cette manière, les utilisateurs peuvent vérifier l'adéquation des valeurs de nos modèles de coût par rapport aux valeurs réelles mesurées.

(d) **Prédiction de l'énergie:** La figure 5.5 donne un aperçu de la fenêtre récapitulant la consommation énergétique réelle et prédite après le traitement d'une requête. Cette fenêtre est initialement vide. Après l'exécution de la requête, l'utilisateur peut alors cliquer sur le bouton « Power Estimation », ce qui va ouvrir une fenêtre lui permettant de voir les valeurs estimées des différentes techniques d'apprentissage et l'énergie réelle consommée. L'utilisateur peut ainsi comparer les valeurs des différentes techniques d'apprentissage.

5.2.2.4 Résultats expérimentaux

Nous menons une série d'expérimentations pour évaluer l'efficacité de nos approches. Nous présentons dans cette section nos résultats expérimentaux obtenus après avoir donné l'architecture matérielle et les benchmarks que nous avons utilisés.

(a) **Configuration matérielle:**
Comme mentionné avant, le serveur de BD a été installé avec un système d'exploitation *Ubuntu (Noyau 5.0.0-27)*. Notre architecture d'implémentation ne change pas, elle reste identique à celle décrite dans la section 4.5.1. Nous avons utilisé les benchmarks TPC-H et SSB pour évaluer nos propositions avec différents facteurs d'échelle (1, 5, 10, 30, 50, 100). Dans la relation décrite par l'équation 5.1, les valeurs extrêmes du critère α entraînent la dégradation de l'une des grandeurs dans le choix du plan d'exécution. Pour ne pénaliser aucune grandeur, une approche serait de choisir des valeurs de α compromettante afin d'obtenir des plans d'exécution avec une performance et une consommation énergétique acceptable. Dans notre expérimentation, nous avons choisi d'étudier trois cas:

— $\alpha = 1$ *pour* $time\_wt = 1$ *et* $power\_wt = 0$*:* cette configuration indique que la performance est l'objectif d'optimisation primaire lors du traitement de la requête.

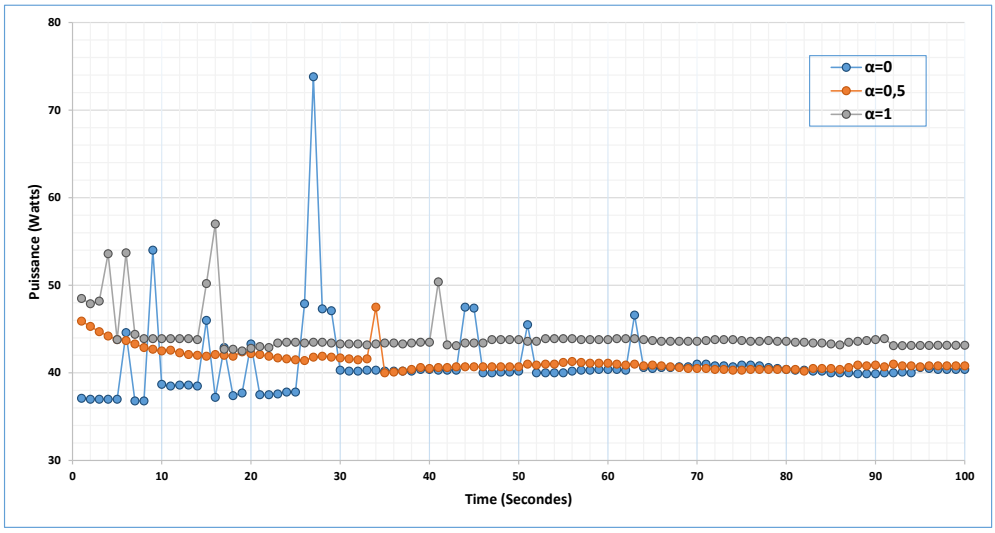


FIGURE 5.6 – Variations de la consommation énergétique capturée en temps réel.

— $\alpha = 0$ *pour* $time\_wt = 0$ *et* $power\_wt = 1$*:* cette configuration indique que l'énergie est l'objectif primaire d'optimisation lors du traitement de la requête.
— $\alpha = 0,5$ *pour* $time\_wt = 0,5$ *et* $power\_wt = 0,5$*:* cette configuration cherche l'équilibre entre les deux grandeurs. Un plan d'exécution qui satisfait la contrainte énergétique sans trop dégrader la performance est sélectionné.

(b) **Résultats:**
En plus de varier le facteur d'échelle (SF) sur chaque benchmark, nous exécutions les requêtes du benchmark TPC-H et SSB sur chaque configuration décrite ci-dessus ($\alpha = 1$, $\alpha = 0$, $\alpha = 0,5$). Pour chacune des configurations, nous collectons les valeurs de puissances et les statistiques d'exécution de chaque requête. Sur la base des résultats collectés à partir de l'optimiseur nous représentons sur la figure 5.6 les variations de la consommation énergétique capturée en temps réel lors de l'exécution d'une requête dans les 100 premières secondes. Partant de cette figure, nous pouvons remarquer en comparant la variation de la consommation d'énergie que la configuration $\alpha = 1$ est sans surprise celle qui consomme le plus d'énergie du fait qu'elle est orienté performance. Pour chaque facteur d'échelle, nous résumons dans le tableau 5.4, la moyenne d'énergie dissipée et le temps qu'il faut pour traiter toutes les requêtes sur chaque configuration. Nous récapitulons également dans ce tableau l'énergie économisée, la variation de la puissance et la dégradation des performances des configurations orientées énergétiques ($\alpha = 0$, $\alpha = 0,5$) comparée à celle qui est orientée performance ($\alpha = 1$). De ce tableau, nous constatons que la configuration $\alpha = 0$ possède nettement une consommation énergétique plus optimale comparée aux autres configurations, soit environ $1,5$ fois d'efficacité énergétique par rapport à la configuration $\alpha = 1$. Par ailleurs la configuration $\alpha = 0$ axée sur l'énergie passe plus de temps pour traiter les requêtes cela se traduit par une dégradation des performances à ce niveau. Cette dégradation des performances est inévitable vu la relation entre les deux grandeurs de même sur la configuration $\alpha = 0,5$, la dégradation des performances est constatée néanmoins nous obtenions une économie d'énergie de $26,4\%$. L'application de notre proposition à travers cette série d'expérimentations relève une économie d'énergie moyenne de $12,5\%$ et $13,9\%$ dans la configuration ($\alpha = 0$) et dans la configuration $\alpha = 0,5$ respectivement.

Afin d'explorer plus en détail le potentiel d'économie d'énergie réalisé dans les deux configurations axées sur la puissance, nous reportons dans le tableau 5.5 la puissance moyenne de l'efficacité énergétique des quinze (15) premières requêtes du benchmark TPC-H. Les requêtes ont

Charge de travail		Compromis	Puissance	Temps	Énergie	Économie		Performance
Jeux	Taille	(α)	(en Watts)	(Secondes)	(KJoules)	Puissance	Énergie	Dégradation
SSB	15 Go	1	73,18	1606,66	117,58	-	-	-
		0	63,95	1682,61	107,60	12,6%	8,5%	-4,7%
		0,5	68,57	1644,63	112,77	6,3%	4,1%	-2,4%
TPC-H	1 Go	1	53,28	30,77	1,64	-	-	-
		0	44,50	35,38	1,57	16,5%	4,0%	-14,9%
		0,5	51,89	35,84	1,86	2,7%	-	-16,4%
TPC-H	10 Go	1	59,55	2337,03	139,17	-	-	-
		0	58,34	3786,19	220,87	2,0%	-	-62,0%
		0,5	46,52	2696,77	125,45	21,8%	9,8%	-15,4%

Tableau 5.4 – Analyse de l'évaluation des plans d'exécution dans les différentes configurations.

été exécutées et analysées sur un jeu de données de taille 1 Go. Comme reporté dans le tableau 5.5, de fortes économies de puissance sont remarquées au niveau des requêtes Q_1, $Q_2 - Q_10$ et $Q_12 - Q_14$. Généralement des économies de puissance conduisent à des économies d'énergie lorsque la dégradation des performances n'est pas conséquente. La moyenne des économies de puissance est de $16,2\%$ pour la configuration $(\alpha = 0)/(\alpha = 1)$ et $2,4\%$ pour celle de $(\alpha = 0,5)/(\alpha = 1)$. Les plus grandes marges d'économies observées dans certaines requêtes peuvent s'expliquer par le fait qu'elles sont composées d'un nombre important d'opérateurs, octroyant ainsi plus d'alternatif à l'optimiseur dans le choix du plan.

Charge de travail		**Puissance (Watts)**			**Économie**	
Benchmarks	Requêtes	$(\alpha = 1)$	$(\alpha = 0,5)$	$(\alpha = 0)$	$(\alpha = 0)/(\alpha = 1)$	$(\alpha = 0,5)/(\alpha = 1)$
TPC-H	1	59,10	41,81	40,67	31,2%	29,2%
	2	59,30	59,67	56,63	4,5%	-0,6%
	3	47,60	47,47	40,22	15,5%	0,3%
	4	50,77	43,30	40,07	21,0%	14,7%
	5	53,55	50,8	39,52	26,2%	5,1%
	6	52,10	50,55	40,67	21,9%	2,9%
	7	52,27	52,20	41,17	21,2%	0,1%
	8	53,12	52,77	43,75	17,6%	0,7%
	9	60,52	58,07	45,37	25,03%	4,04
	10	52,37	53,17	41,12	21,5%	-1,5%
	11	53,22	52,42	63,43	-19,2%	1,5%
	12	51,27	51,45	42,35	17,40%	-0,3%
	13	54,10	59,80	43,30,77	19,9%	-10,5%
	14	52,32	53,10	41,80	20,1%	-1,4%
	15	47,55	51,72	47,45	0,2%	-8,7

Tableau 5.5 – Analyse de la puissance optimisée au niveau des requêtes du benchmark TPC-H.

5.2.3 Intégration avec MonetDB

Nous relatons dans cette section, les fonctionnalités de notre outil «MonetDB Eco-Processing» (MEcoPro) mis en œuvre pour exploiter notre modèle énergétique définit dans la section 4.6.2.2

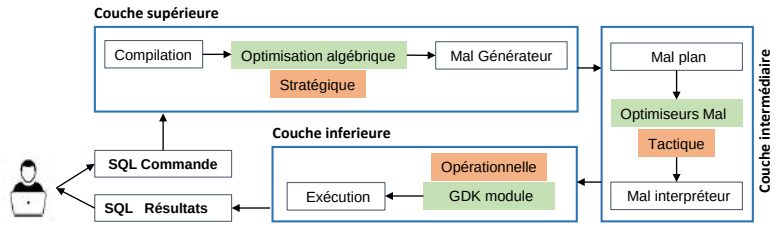


FIGURE 5.7 – L'architecture de MonetDB avec vue sur les optimiseurs.

lors de l'exécution des requêtes sur le système MonetDB. Le Framework est composé de trois modules: un module de visualisation et de paramétrage, un module d'estimation de la puissance et le dernier décrit le système MonetDB.

5.2.3.1 Modèle de coût

Comme dans le cas du système PostgreSQL, il faut trouver une approche pour intégrer notre modèle dans le processus de traitement de la requête dans le système MonetDB. Pour cela, nous commençons par analyser les points d'intégration possible de notre modèle. L'architecture de MonetDB (figure 5.7) possède une couche principale responsable de fournir le modèle de données binaire, le moteur de traitement des requêtes, les mécanismes de concurrence et le contrôle des transactions. Au-dessus de cette couche, fonctionne une couche ayant pour objectif de fournir une interface pour supporter différents modèles de données.

En analysant les étapes du processus de traitement (figure 5.7) évoquées dans la section 4.4.2, nous distinguons trois points d'intégration possibles de notre modèle énergétique pouvant jouer sur la consommation énergétique de la requête. Ces trois points se situent au niveau des trois optimiseurs (Stratégique, Tactique, Opérationnelle) que MonetDB utilise pour améliorer la performance des requêtes complexes.

(a) **Stratégique:**
Cette étape d'optimisation est identique à celle de la réécriture que nous avons abordée dans le cas de PostgreSQL. Certes qu'elle impacte considérablement le coût de traitement de la requête lors de son exécution, mais n'offre pas assez d'alternatifs de sélection car il s'agit d'appliquer sur la structure du plan un ensemble de règles connues qui réduisent la taille des données intermédiaires comme par exemple la descente de l'opérateur de sélection dans l'arborescence dans la mesure du possible. L'ordre de jointure des tables est fait sur la base des heuristiques qui réduisent le volume des données produit par une séquence de jointure. Pour le faire, le système MonetDB se base sur leurs coûts de traitement. Nous avons proposé de modifier le fichier « rel_planner.c » dans le répertoire « src\sql\\» afin qu'en plus du coût de traitement de considérer le coût énergétique. Cette modification affecte l'ordre de jointure et n'a pas d'effet sur le choix des implémentations algorithmiques.

(b) **Tactique:**
L'optimiseur tactique travaille sur le code MAL généré et est constitué d'une collection de modules d'optimiseur. Chaque module transforme le code MAL donné en un code plus efficace en ajoutant éventuellement des directives de gestion des ressources. L'optimisation tactique est plus basée sur l'optimisation du langage de programmation que sur l'optimisation classique des requêtes qu'on retrouve dans les BDs orientées modèle de coût. À cette étape, chaque Module d'Optimiseur des Requêtes *« MOR »* dédié à une fonction bien spécifique réécrit le code MAL du plan. Par exemple, le module qui supprime les *coercitions* inutiles qui peuvent résulter d'un générateur de code peu cohérent ou de résolution des appels de fonction. Par exemple: $v := calc.int(23)$; devient une affectation unique sans appel à la fonction $v := 23$ [1]. Les modules sont généralement groupés en pipeline dans des paquets d'optimiseur pour un objectif bien définit. La liste des modules d'optimiseur est défini dans le répertoire : « src\monetdb5 \optimizer ».

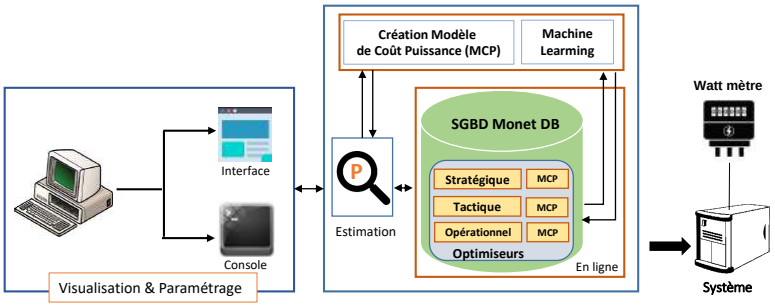


FIGURE 5.8 – Workflow du Framework EcoMPro.

Pour exploiter la dimension énergétique lors de cette phase, une approche serait d'estimer le coût énergétique du code MAL avant et après chaque transformation par un module d'optimiseur énergétique. Si celle-ci optimise l'énergie on la maintient sinon on rapplique au code mal précèdent. Il est également possible de concevoir un optimiseur dédié à l'optimisation énergétique qui réorganise/ ou remplace les instructions sur la base de leur consommation énergétique. Ces approches se révèlent être très coûteuse en terme du volume de modification qu'il faudrait faire. Néanmoins ce sont des approches que nous sommes entrains d'explorer.

(c) **Opérationnelle:**
Au fur de l'exécution des instructions MAL, le contrôle de chacune d'elles est transmis à la *« bibliothèque GDK »* où se déroule le traitement de la requête. Le « GDK » fournit les accès aux structures de données (BAT) ainsi qu'une bibliothèque d'implémentation hautement optimisées des opérateurs algébriques. En raison du paradigme de traitement d'une *« opération à la fois »* (bulk -processing), chaque opérateur a accès à ces données en entrées y compris ces propriétés. Ceci permet au moteur d'exécution « GDK » d'effectuer une optimisation opérationnelle, c'est-à-dire choisir l'implémentation algorithmique la mieux adaptée en fonction des caractéristiques des données en entrée. Par exemple, un opérateur de sélection peut déployer une recherche dichotomique si les données dans le BAT sont triées, un parcours indexé pour les sélections ponctuelles ou encore un balayage séquentiel. De même, pour une jointure le GDK peut effectuer une jointure de fusion-tri si les attributs de jointures sont triés, sinon la jointure par hachage.

Une des approches pour intégrer notre modèle énergétique dans l'optimisation opérationnelle serait de sélectionner l'implémentation algorithmique de certains opérateurs sur la base de leurs coûts énergétiques. Ce choix est fait dans le GDK sur la base des propriétés des données en entrées (nombre de tuple, caractère de trie des attributs,...). Nous sommes entrains d'explorer cette approche en apportant toutes les modifications nécessaires dans les fichiers dans le répertoire: *« src/dgk/ »* principalement dans le fichier *« gdk_join.c »*.

5.2.3.2 Architecture fonctionnelle

Dans cette section, nous présentons la conception de l'outil « EcoMPro ». Dans cette première version, nous utilisons notre modèle pour des fins d'estimation et d'optimisation énergétique en l'implémentant au-dessus du système MonetDB. Notre outil est composé de deux grandes parties indépendantes: une partie basée sur le SGBD et une partie interface graphique (GUI) permettant de définir les valeurs de certains paramètres et de visualiser en temps réel la consommation énergétique des requêtes. Cet outil a été développé en $Java(Version1)$. Le workflow de notre Framework est décrit sur la figure 5.8.

Comme pour PostgreSQL, le module GUI permet à l'utilisateur de définir les valeurs de certains paramètres, de visualiser la consommation énergétique de la requête en temps réel, de visualiser les plans d'exécution et de voir la comparaison des valeurs de prédiction de l'énergie en utilisant les trois techniques d'apprentissage. Le module permet de faire les tâches suivantes:

FIGURE 5.9 – Interface IHM du framework EcoMPro.

— Configuration des valeurs des paramètres du système
— Visualisation du plan
— Visualisation de la chronologie de la consommation
— Visualisation des valeurs de prédiction de l'énergie
— Visualisation des résultats de la requête.

Les fonctionnalités des différentes interfaces étant similaires à celle de PostgreSQL, pour ne pas nous répéter nous donnons sur la figure 5.9 des aperçus de quelques interfaces. Nous avons apporté des modifications sur certains fichiers du code source de MonetDB afin d'incorporer notre modèle de coût.

5.2.3.3 Résultats expérimentaux

Nous menons une série d'expérimentations pour évaluer l'efficacité de nos approches. Nous présentons dans cette section nos résultats expérimentaux, l'architecture matérielle et les bancs d'essai que nous avons utilisés restent inchangé par rapport aux expérimentations avec PostgreSQL.

Pour quantifier la précision de nos estimations, nous avons utilisé la métrique de taux d'erreur évoquée dans la section 4.7.1. Le tableau 5.6 résume les erreurs obtenues, l'erreur moyenne en utilisant la technique de la régression non linéaire est $10,9\%$ sur le jeu de données du TPC-H et $12,65\%$ pour celui de SSB. En utilisant la technique des réseaux de neurones, elle est égale $17,66\%$ sur TPC-H et $14,12\%$ pour SSB.

En déployant notre modèle dans l'optimiseur stratégique du système, nous étudions les caractéristiques des requêtes du banc d'essai TPC-H et SSB afin d'évaluer les avantages de notre approche en termes d'efficacité énergétique. Dans le tableau 5.7, nous reportons la puissance moyenne, l'énergie totale consommée, l'efficacité énergétique des requêtes du banc d'essai TPC-H et SSB. Les requêtes ont été exécutées et analysées sur trois jeux de données de taille 10, 50 et 100 Go. Comme reporté dans le tableau 5.7, La moyenne des économies d'énergie est de $8,03\%$ dans le cas du banc d'essai TPC-H et $2,0\%$ dans celui du SSB. La plus grande marge d'économies d'énergie observées au niveau du banc d'essai TPC-H s'explique par le fait que ces requêtes contiennent un nombre important de séquencement de jointure.

TABLEAU 5.6 – Taux d'erreurs d'estimation de la puissance énergétique des requêtes (MonetDB - TPC-H/SSB)

(a) *Modèle NLR, RNA et FAR (TPC-H)*

TPC-H (SF=10Go)							
Q_i	*Erreur(%)*			Q_i	*Erreur(%)*		
	NLR	*RNA*	*FAR*		*NLR*	*RNA*	*FAR*
1	-	16,15%	12,51%	12	7,98%	25,23%	10,05%
2	35,47%	13,71%	10,69%	13	3,60%	19,55%	0,15%
3	12,49%	12,76%	14,85%	14	0,89%	14,61%	5,66%
4	8,55%	6,76%	20,12%	15	0,77%	17,09%	1,04%
5	2,32%	13,41%	10,46%	16	-	-	-
6	16,02%	30,33%	12,33%	17	15,50%	24,70%	7,30%
7	0,95%	18,20%	2,75%	18	28,09%	21,92%	4,00%
8	6,16%	16,92%	5,39%	19	4,58%	19,85%	0,33%
9	2,49%	13,23%	9,53%	20	16,74%	18,33%	3,75%
10	-	15,53%	12,96%	21	23,79%	20,76%	5,24%
11	3,53%	15,43%	4,41%	22	18,47	16,65	2,59

(b) *Modèle NLR, RNA et FAR (SSB)*

SSB (SF=25Go)							
Q_i	*Erreur(%)*			Q_i	*Erreur(%)*		
	NLR	*RNA*	*FAR*		*NLR*	*RNA*	*FAR*
1	-	13,56%	16,95%	8	-	1,02%	13,58%
2	20,32%	10,62%	25,75%	9	2,59%	10,47%	26,04%
3	8,08%	9,02%	24,55%	10	6,57%	5,98%	20,55%
4	13,90%	21,52%	22,04%	11	10,13%	28,39%	26,31%
5	14,13%	16,51%	32,36%	12	14,07%	20,82%	34,94%
6	4,19%	11,41%	24,60%	13	36,33%	13,52%	29,38%
7	8,87%	20,76%	13,96%				

Charge de travail		Compromis	Puissance	Temps	Énergie	Économie		Performance
Jeux	Taille	(α)	(Watts)	(Secondes)	(KJoules)	de Puissance	d'Énergie	Dégradation
SSB	50 Go	1	68,84	428	29,46			
		0	63,32	1356	85,86	8,0%	-	-
SSB	100 Go	1	60,62	3620	219,43			
		0	58,20	3699	215,29	4,0%	2,0%	-2,1%
TPC-H	10 Go	1	66,08	256	16,91			
		0	61,77	265	16,36	6,5%	3,2%	-3,5%
TPC-H	50 Go	1	59,42	3159	187,71			
		0	58,05	3078	178,68	2,3%	5,0%	2,6%

TABLEAU 5.7 – Analyse de l'évaluation des plans d'exécution dans les différentes configurations (MonetDB).

5.3 Estimation dynamique de l'énergie lors du traitement des requêtes (Simulation)

Dans cette section, nous décrivons l'architecture et les résultats expérimentaux obtenus en utilisant notre outil « *Simulator Query Eco-processing In MemoryDB* » (SimQEcoIM). SimQEcoIM se base sur l'analyse des compteurs de performance matérielle (Hardware performance counters) pour estimer l'énergie dynamique consommée par les SSDs en mémoire pendant le traitement des requêtes. Les compteurs permettent de collecter des informations détaillées sur la performance des composants telles la fréquence d'utilisation des cœurs du processeur, la quantité de mémoire utilisée et les informations relatives à chaque processus. Ils révèlent une quantité considérable d'information en corrélation avec la consommation d'énergie du composant (système). Les compteurs de performance ont pour fonction de surveiller différents événements qui se produisent lorsqu'un processeur exécute des instructions (p. ex., erreurs de prédiction de branchement, défauts de cache). Il existe de nombreux outils pour quantifier les évènements des compteurs de performances tel que PAPI [1], perf_events [2], PCM [3]. Dans notre étude, mous avons

1. `https://icl.utk.edu/papi/`
2. `https://www.kernel.org/doc/html/latest/admin-guide/perf-security.html`
3. `https://github.com/opcm/pcm`

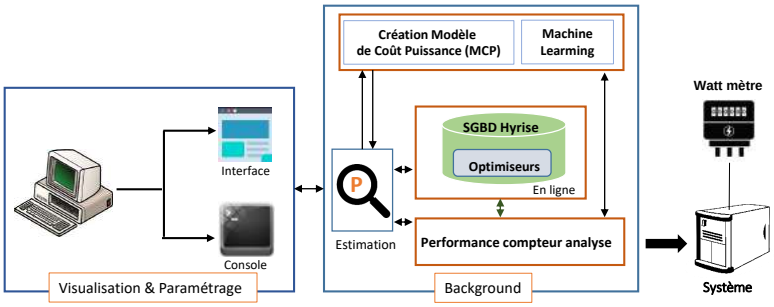


FIGURE 5.10 – Architecture du framework SimQEcoIM.

utilisé PAPI qui fournit une interface et une méthodologie standardisée pour l'utilisation des compteurs de performance matérielle. Les compteurs que nous avons analysés lors de l'exécution des requêtes sont: *Instructions par cycle (IPC)*, *Défaut/Succès cache*, *Instructions retirées*.

Lors de l'exécution d'une requête sur une BD en mémoire, nous capturons les différents événements des compteurs de performance nécessaire à l'estimation du coût énergétique puis nous exploitons le modèle énergétique proposé pour Hyrise pour estimer la consommation d'énergie du système. Notre modèle énergétique a été formalisé et validé en utilisant le système Hyrise dont les résultats expérimentaux sont reportés dans la section 4.7.3.3. Nous avons déployé notre modèle énergétique dans l'objectif de réaliser des estimations dynamiques en ligne de la consommation énergétique pour tout systèmes de BD en mémoire lors du traitement des requêtes. Pour évaluer l'efficacité de notre approche de modélisation en utilisant notre simulateur SimQEcoIM, nous avons mené des études d'estimation sur deux nouveaux systèmes de BD en mémoire nommés: HSQLDB [4] et H2 [5].

5.3.1 Architecture fonctionnelle

L'outil SimQEcoIM a été conçu pour contourner les difficultés liées à l'acquisition et l'exploitation des équipements physiques de mesure d'énergie dans les SSDs en mémoire lors du traitement des requêtes. Il est composé de trois grandes parties: une partie définissant le SGBD en mémoire, une partie interface graphique (GUI) permettant de définir les valeurs de certains paramètres et de visualiser en temps réel la consommation énergétique des requêtes et une partie pour la surveillance et la mesure des valeurs des compteurs de performance. Cet outil a été développé dans un langage de programmation *Java* associé à la bibliothèque *Awt*. Le workflow de notre Framework est décrit sur la figure 5.10.

Les fonctionnalités des différentes interfaces restent similaires à celle de PostgreSQL et MonetDB seulement que les interfaces IHM diffèrent un peu dans le design, nous donnons sur la figure 5.11 des aperçus de quelques interfaces.

5.3.2 Résultat expérimentaux

Nous présentons dans cette section nos résultats expérimentaux, l'architecture matérielle et les bancs d'essai que nous avons utilisés restent inchangé par rapport aux expérimentations avec PostgreSQL. Les requêtes ont été exécutées et analysées sur deux jeux de données de taille 1 Go et 2 Go.

Pour quantifier la précision de nos estimations, nous avons utilisé la même métrique du taux d'erreur évoquée dans la section 4.7.1. Les tableaux 5.8 résument les erreurs d'estimation obtenues pour les systèmes HSQLDB et H2DB. Sur le système HSQLDB, l'erreur moyenne vaut $7,6\%$ sur le jeu SSB de 1 Go (et $17,1\%$ sur le jeu de données de 2 Go) en utilisant le modèle des réseaux

4. `http://hsqldb.org/`
5. `https://www.h2database.com/html/main.html`

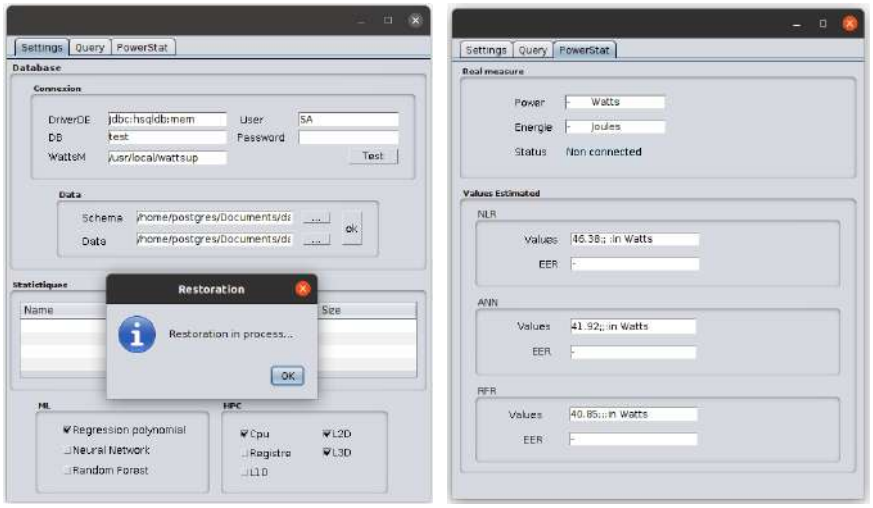


FIGURE 5.11 – Interface IHM du framework SimQEcoIM.

de neurones(RNA). La moyenne d'erreur est 18, 3% sur le jeu SSB de 1 Go et 16, 35% sur celui de 2 Go en utilisant le modèle des réseaux de neurones(RNA) sur le système H2. Notre modèle de la régression non linéaire fait une grande marge d'erreur dans l'estimation de la puissance énergétique dans la majorité des 13 requêtes étudiées, cela s'explique par le fait que les systèmes HSQLDB et H2DB cumulent un nombre important de défaut de caches au niveau des mémoires intermédiaires. Ce cumule impact considérablement la qualité du modèle de la régression non linéaire. Ce dernier a été défini sur la base d'un système orienté colonne. Néanmoins cette marge a pu être gérée par les deux autres approches.

TABLEAU 5.8 – Taux d'erreurs d'estimation de la puissance énergétique des requêtes (HSQL/H2 - SSB)

(a) *Modèle RNA et FAR (HSQLDB)*

SSB (SF=1Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	RNA	*FAR*		*RNA*	*FAR*
1	34,05%	33,50%	6	0,45%	7,10%
2	33,97%	33,42%	7	0,78%	8,43%
3	1,35%	7,93%	8	0,68%	8,34%
4	2,43%	8,94%	9	1,35%	6,46%
5	0,33%	6,98%	10	1,01%	6,77%

SSB (SF=2Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	RNA	*FAR*		*RNA*	*FAR*
1	3,80%	10,22%	6	31,41%	30,70%
2	14,65%	20,35%	7	26,36%	31,27%
3	12,89%	18,70%	8	11,94%	17,81%
4	23,37%	22,37%	9	11,85%	18,65%
5	23,12%	22,48%	10	11,98%	18,77%

(b) *Modèle RNA et FAR (H2DB)*

SSB (SF=1Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	RNA	*FAR*		*RNA*	*FAR*
1	19,94%	25,28%	8	13,53%	18,96%
2	5,15%	1,85%	9	24,19%	29,09%
3	8,50%	1,27%	10	21,08%	26,19%
4	18,46%	23,52%	11	15,34%	16,84%
5	28,70%	33,12%	12	15,59%	8,42%
6	21,50%	26,37%	13	25,59%	26,91%
7	20,65%	24,30%			

TPC-H (SF=2Go)					
Q_i	*Erreur(%)*		Q_i	*Erreur(%)*	
	RNA	*FAR*		*RNA*	*FAR*
1	0,38%	6,30%	8	30,28%	34,79%
2	18,91%	24,32%	9	10,92%	16,62%
3	25,34%	26,65%	10	12,67%	18,33%
4	14,03%	19,60%	11	29,50%	33,92%
5	13,70%	19,05%	12	22,19%	27,23%
6	15,35%	20,99%	13	16,78%	22,33%
7	2,48%	8,99%			

Les moyennes d'erreur obtenues lors des expérimentations avec les deux systèmes révèlent la

précision de notre approche dans des études d'estimation. **Notre simulateur par conséquent peut servir de guide pour détecter les requêtes gourmandes en énergie ou d'outil d'estimation de l'énergie totale consommée par une charge de travail donnée.**

5.4 Conclusion

Dans ce chapitre, nous avons introduit nos outils implémentés au-dessus de trois SSDs relationnels (PostgreSQL, MonetDB et Hyrise) dans le but de proposer des systèmes écologiques(vert) intégrant la dimension énergétique dans le processus de traitement des requêtes. Pour ce faire, nous avons d'abord proposé pour chaque système un modèle de coût énergétique prenant en compte plusieurs paramètres clés afin d'estimer de manière approprié la puissance énergétique du système lors du traitement des requêtes. Puis, nous avons analysé le module de traitement des requêtes de chaque système afin d'identifier les points d'exploitation possible de nos modèles de coût conduisant à une efficacité énergétique. Ainsi pour *PostgreSQL*, nous avons proposé un formalisme d'évaluation des plans d'une requête intégrant notre modèle afin de permettre à l'optimiseur de sélectionner un plan qui répond aux besoins spécifiques de l'utilisateur. Pour *MonetDB*, nous avons exploré trois niveaux d'intégration possibles de notre modèle de coût énergétique et pour *Hyrise*, nous avons utilisé notre modèle pour construire un outil permettant d'estimer la consommation énergétique des requêtes pour différents SGBDs en mémoire. Pour chaque cas, des expérimentations intensives ont été conduite avec différents bancs d'essai pour prouver l'efficacité et l'utilité de nos approches. Notre évaluation révèle que nous pouvons optimiser l'énergie consommée en utilisant une approche de compromis entre la performance et la puissance ou en réduisant le coût de calcul en réordonnant l'ordre de jointure du plan.

Quatrième partie

Conclusion et perspectives

CHAPITRE 6

Conclusion générale et Perspectives

Sommaire

6.1 Conclusion 139
6.1.1 Audit des STRs 140
6.1.2 L'efficacité énergétique dans les systèmes informatiques 140
6.1.3 Modèles de coût énergétique 140
6.1.4 L'utilisation de l'apprentissage automatique 141
6.1.5 Compromis de traitement entre la performance et l'énergie 141
6.1.6 Développement des outils pour le traitement vert des requêtes 141
6.2 Perspectives 142
6.2.1 Extension du modèle de coût 142
6.2.2 Traitement écologique des requêtes 142
6.2.3 Optimisation énergétique dans les systèmes de stockage en mémoire . . 142
6.2.4 Prise en compte de d'autres modèles et des requêtes 143
6.2.5 Vers un simulateur pour le choix d'un SSD adapté à un client 143
6.2.6 Retour d'expérience: Learned Modèles de coût 143
6.2.7 Autres approches d'optimisation énergétique dans les BDs 143

6.1 Conclusion

Ces dernières années, l'efficacité énergétique est devenue l'une des exigences majeures de conception des composants du système informatique, allant d'un simple ordinateur portable aux serveurs infonuagiques dans les SSDs, car au fil des années l'énergie consommée par les composants n'ont guère cesser de croître. Outre les coûts d'exploitation exorbitants, la forte consommation énergétique conduit à des émissions importantes de gaz à effet de serre dans l'environnement responsable du dérèglement climatique.

Pour résoudre ce problème de forte consommation dans les systèmes informatiques et faciliter la transition écologique, des initiatives ont été élaborées permettant de réduire à la fois la consommation excessive et son impact écologique. Ces initiatives s'orientent dans deux axes de direction: (i) développer de nouvelles sources d'énergie propres et renouvelables ou (ii) maximiser l'efficacité énergétique et diminuer le gaspillage énergétique.

Dans cette thèse, nous nous sommes focalisés sur l'efficacité énergétique dans les SSD, plus précisément dans l'intégration de la contrainte énergétique dans les STRs. Ces derniers sont au cœur du fonctionnement des SSDs et constituent son élément le plus énergivore à cause de l'évolution fulgurante des données massives qu'ils doivent stocker, traiter et maintenir. Notre approche pour optimiser la consommation énergétique dans les STRs est d'analyser le processus de traitement des requêtes afin de dégager les points d'intégration possibles de la dimension énergétique puis de proposer une technique d'optimisation dirigée par l'utilisation des modèles de coût énergétique. Ce travail de thèse a conduit à la couverture d'un large éventail des domaines: les

entrepôt de données, l'optimisation des requêtes, le développement des modèles de coût, l'énergie, l'apprentissage automatique, etc. Pour faciliter à la fois la compréhension et la réutilisation de notre approche, nous avons proposé une méthodologie de conception et d'exploitation de nos modèles de coût énergétique dans les STRs. Cette méthodologie porte essentiellement sur les étapes suivantes : (i) l'audit des composants du processeur des requêtes, (ii) l'élaboration d'un état de l'art sur les approches d'efficacité énergétique dans les STRs, (iii) la définition des modèles de coût pour quantifier l'énergie des plans des requêtes, (iv) la proposition des techniques d'exploitation de nos modèles de coût pour des fins d'optimisation et/ou d'estimation énergétique, (v) le déploiement des modèles dans le fonctionnement interne des STRs de différents SDDs.

6.1.1 Audit des STRs

Ce chapitre est primordial dans notre vision de l'étude de l'efficacité énergétique dans les SDD. Il a permis d'identifier les composantes sensibles à l'énergie d'un SDD. Pour satisfaire cet objectif, nous avons mené un état de l'art qui nous a permis de comprendre le fonctionnement global des SSDs et de maîtriser les interactions des éléments qui composent son écosystème. Après avoir identifié les points qu'on pourrait exploiter pour proposer une technique de réduction de la consommation énergétique dans les SSDs lors du processus de traitement des requêtes, nous avons déterminé différents paramètres clés qui peuvent influencer la consommation énergétique du système. Lors de cette étude, nous les avons classifiés en cinq nivaux essentiels: les paramètres ayant trait à la requête, les paramètres du processeur de requêtes, le système matériel, l'architecture de déploiement adopté et les facteurs environnementaux. Cette explicitation nous a permis de délimiter notre environnement de travail et nous a servi lors de la phase de sélection des paramètres pertinents qui caractérisent nos modèles de coût énergétique.

6.1.2 L'efficacité énergétique dans les systèmes informatiques

Dans ce chapitre, nous avons passé en revue les principaux travaux de la littérature en mettant l'accent sur la minimisation de la consommation énergétique dans les SSD d'une manière générale puis de manière plus spécifique dans STRs que nous avons classifiées en trois approches principales: (i) les approches basées sur la gestion de l'énergie des composants matériels, (ii) les approches basées sur les techniques logicielles et l(iii) es approches basées sur les techniques de gestion environnementale et règlementaire.

Nous avons décrit les efforts investis dans la quête de l'efficacité énergétique dans chacune de ces catégories en découpant l'approche logicielle en deux groupes: (1) les techniques utilisées lors du traitement de la requête et (2) les techniques incorporées lors de la conception physique. L'élément fondamental dans chacune de ces techniques reste la définition d'un modèle de coût mathématique qui estime la consommation énergétique du système lors de l'exécution de la requête. Cette étude nous a permis de fournir une feuille de route pour les concepteurs et les étudiants qui souhaitent comprendre les questions liées à la gestion de l'énergie dans le monde des BDs ou de servir de base pour le développement des modèles énergétiques. Nous avons identifié les lacunes des approches existantes par rapport au contexte actuel de l'exploitation des SSDs et nous avons proposé nos démarches pour les améliorer en intégrant de nouveaux paramètres dans les modèles de coûts proposés.

6.1.3 Modèles de coût énergétique

En utilisant les connaissances acquises lors de l'analyse des composants des processeurs de requêtes, nous avons développé des modèles de coût pour estimer la consommation d'énergie du système lors de l'exécution des requêtes. L'objectif principal de ces modèles est d'estimer avec précision les coûts énergétiques des requêtes. Pour chaque type de SSD évoqués dans nos études, nous proposons des modèle sde coût différant dont les paramètres (coûts d'E/S

et CPU) sont soient estimés à partir de l'optimiseur ou en utilisant des outils d'analyse de performance. Nos modèles sont différents de ceux fournis dans la littérature selon deux dimensions: (i) ils intègrent de nouveaux paramètres (E/S mémoire caches) pertinents dans l'étude de la consommation énergétique des requêtes et deuxièmement et (ii) qu'ils sont reproduits sur les nouvelles architectures de traitement et sur les nouveaux types de SSDs.

6.1.4 L'utilisation de l'apprentissage automatique

Après l'analyse des données d'apprentissage, les valeurs de certains paramètres doivent être calculées en utilisant les algorithmes l'apprentissage automatique. cette étude, nous avons utilisé trois algorithmes : l'algorithme de la régression polynomiale multiple, l'algorithme des réseaux de neurones et l'algorithme des forêts aléatoires dans le but de confronter les prédictions et d'évaluer leurs impacts sur la précision de nos modèles de coût. Lors de cette étape, nous avons effectué des tests avec chaque algorithme sur les différentes plateformes (PostgreSQL, MonetDB, Hyrise) en utilisant les jeux de données des benchmarks TPC-H, TPC-DS, SSB. Dans chaque scénario, nous avons évalué l'erreur d'estimation en comparant les coûts réels de l'énergie avec ceux prédits par nos modèles énergétiques. Nos résultats expérimentaux révèlent la précision de nos modèles et aussi que le choix de l'algorithme d'apprentissage peut considérablement impacter sur la qualité du modèle.

6.1.5 Compromis de traitement entre la performance et l'énergie

Dans cette thèse, nous avons proposé un modèle d'évaluation des plans lors de l'exécution d'une requête basé sur le compromis entre la performance et l'énergie. Dans les optimiseurs traditionnels mono-objectifs, ce modèle d'évaluation a pour objectif de sélectionner le plan le plus rapide. Nous avons proposé qu'au lieu de choisir un plan avec une performance optimale, d'utiliser une méthode d'évaluation pour définir un seuil de compromis entre l'énergie et la performance. Ce compromis est fait sur la base d'un critère qui reflète la supériorité de la performance (T) ou du coût énergétique (P).

6.1.6 Développement des outils pour le traitement vert des requêtes

Nous avons proposé différents outils qui intègrent nos modèles de coût validés dans des études d'estimation et de conservation énergétique. Nous avons apporté des modifications dans l'optimiseur des requêtes de PostgreSQL (resp. MonetDB) pour intégrer la dimension énergétique dans le modèle d'évaluation des plans. Pour PostgreSQL, nous avons implémenté « *GreenPipeline* ». Cet outil a été conçu dans le but de proposer un SSD écologiques (vert). Il considère la dimension énergétique dans le processus de traitement des requêtes et permet aux utilisateurs de visualiser les différents plans des requêtes moins énergivores et de pouvoir définir les valeurs de certains paramètres de configuration (par ex. le compromis entre l'énergie et la performance). Pour monetDB, nous avons proposé «MonetDB Eco-Processing» (MEcoPro) et pour Hyrise nous avons proposé un estimateur « SimQEcoIM » développé pour permettre aux administrateurs d'analyser le comportement de la consommation d'énergie des BDs en mémoire lors de l'exécution des requêtes. Ce simulateur donne la possibilité aux utilisateurs de confronter les différents algorithmes d'apprentissage évoqués dans nos études. Nous avons également mené une série d'expérimentation en utilisant nos Frameworks proposés. Partant des résultats obtenus, nos outils se révélent être précis dans les études d'estimations et peuvent conduire à réduire l'énergie consommée par le SSD lors du traitement des requêtes.

6.2 Perspectives

De nombreuses perspectives tant pour les solutions orientées matérielles que logicielles peuvent être envisagées à partir des travaux initiés lors de cette étude. Nous présentons dans cette section celles qui ont plus retenues notre attention.

6.2.1 Extension du modèle de coût

Malgré que nos modèles d'estimation donnent des résultats assez proches des valeurs réelles mesurées à partir du power mètre utilisé, nous remarquons que certaines différences peuvent s'expliquer par le fait que nous n'avons pas tenu compte de la consommation énergétique de certains composants ou de l'influence de certains facteurs. Par conséquent, l'une des perspectives que nous sommes entrains d'étudier actuellement est d'étendre nos modèles proposés afin d'intégrer d'autres paramètres énergétiques liés aux composants physiques tels que: les bus de communication internes, le registre, etc. et à l'environnement tels que : la température ambiante, l'âge des systèmes, etc. Le présent travail a porté sur les systèmes centralisés relationnels, nos modèles peuvent être étendus en intégrant le coût énergétique du flux de transfert des données entre les équipements réseaux (ex. carte réseau) afin de pouvoir les utiliser sur des systèmes distribués ou sur des systèmes NoSQL. Considérant l'impact que peuvent joué les techniques d'apprentissage sur la précision du modèle, l'exploration de d'autres techniques d'apprentissage avancées peuvent se révéler payante en termes de précision d'estimation de la consommation énergétique et en la qualité de sélection du meilleur plan moins énergivore.

6.2.2 Traitement écologique des requêtes

Dans ce travail, nous avons audité les STR de trois SSD appartenant à des classes différentes pour des fins d'efficacité énergétique. Notre approche a considéré la problématique de l'intégration de la contrainte énergétique dans le contexte des SSD centralisés c'est-à-dire sur une seule station de travail. Les résultats expérimentaux ont révélé que nous pouvons optimiser la consommation énergétique lors de l'exécution de certaines requêtes sur les deux systèmes sur lesquels nous avons mené les études d'optimisation. Une manière d'étendre ou d'évaluer la précision de notre approche serait d'intégrer les modèles proposés dans le processeur des requêtes de d'autres SSDs appartenant à la même classe afin de quantifier le gain énergétique. Et aussi, comme discuté dans le chapitre 1, notre approche peut facilement être appliquée dans le module de traitement des requêtes dans le cadre des SSD distribués et les systèmes multi-stores (NoSQL). Pour accroitre l'efficacité énergétique lors du traitement des requêtes, en plus de l'intégration de l'énergie dans le module de traitement des requêtes, il est possible de gérer dynamiquement à l'aide du modèle proposé la sélection de la fréquence optimale d'utilisation du processeur. Il s'agit de diminuer la fréquence d'exploitation du processeur lorsque le coût énergétique de l'opération en cours de traitement n'est pas assez conséquent. Cette approche permet ainsi de tirer profit du gain énergétique obtenu en cumulant les deux approches: matérielle et logicielle.

6.2.3 Optimisation énergétique dans les systèmes de stockage en mémoire

Dans notre travail, nous nous sommes arrêtés à l'analyse de la portabilité de notre modèle sur d'autres systèmes en mémoire en menant des études d'estimation énergétique avec le modèle proposé pour le système Hyrise. Nos résultats expérimentaux attestent la robustesse et la portabilité de notre modèle en utilisant la technique des forêts aléatoires et celle des réseaux de neurones. Il serait intéressant d'étendre ce travail en menant des études d'optimisation énergétique en intégrant la contrainte énergétique dans le processus de traitement des requêtes surtout dans le cas des systèmes mobiles (ex. téléphones). En effet, les téléphones deviennent de plus en plus les terminaux finaux avec lesquels les utilisateurs interagissent avec les services disponibles sur le web. De nombreuses applications utilisent les BDs en mémoire sur les téléphones avant

de swapper les données vers les serveurs distants, il serait donc très bénéfique de revisiter le processeur des requêtes des systèmes en mémoire sur les téléphones compte tenue de leur forte contrainte énergétique.

6.2.4 Prise en compte de d'autres modèles et des requêtes

Dans cette thèse, comme évoqué dans le chapitre 4, nous avons construit et testé nos modèles en considérant des requêtes décisionnelles de type Sélection, Jointure, Agrégation et Projection. La charge de requête utilisée pour l'apprentissage n'inclut pas les requêtes ayant des imbrications de sélection mais lors des validations, certaines requêtes avaient cette forme (TPC-H, TPC-DS). Les conclusions faites dans ce travail se basent uniquement sur les expérimentations menées avec les requêtes de type SPJ (sélection, projection, jointure) alors que nous pouvons trouver d'autres types de requête comme les requêtes récursives (gourmandes en calcul), les requêtes de maintenance. Il serait intéressant dans l'objectif d'être plus générique d'inclure ces types de requêtes dans nos modèles proposés. De même notre étude a porté sur le cas du modèle relationnel, il serait intéressant de revisiter nos propositions pour traiter d'autres modèles de données en pleine expansion, comme les RDF et son langage SPARQL, les BDs graphes, etc.

6.2.5 Vers un simulateur pour le choix d'un SSD adapté à un client

Dans ce travail, notre étude du comportement énergétique de trois types de SSD sur des requêtes de type SPJ a révélé que pour la même charge de travail fournie dans le même environnement, les SSDs peuvent avoir un comportement énergétique très différent. À la suite des outils proposés qui intègrent la dimension énergétique, il serait intéressant de proposer un simulateur capable de guider le choix des entreprises et des enseignants en la sélection d'un SSD vert. En fait ce simulateur doit à partir d'une charge de requête, la BD, et les contraintes être capable en sortie de recommander le SSD le plus économe en énergie à déployer.

6.2.6 Retour d'expérience: Learned Modèles de coût

La conception des modèles de coût énergétique est une tâche complexe et fastidieuse. C'est l'étape la plus longue dans notre méthodologie qui a pris environ 70% avec les étapes de calibration et de validation. Cette lenteur est au temps passé pour définir les formules mathématiques dédiées à chaque métrique énergétique. Chaque métrique nécessite un ensemble important de paramètres appartenant à plusieurs composantes matériels et logiciels. Récemment de nombreux travaux ont poussé l'idée d'utiliser des techniques d'apprentissage automatique pour estimer tous les éléments de l'environnement des modèles de coût des optimiseurs de requêtes tels que les cardinalités [89], l'ordre des jointures [191], les index (connus sous le nom de *learned indexes*) [112], etc. Ces travaux doivent être étudiée d'une manière approfondie afin de les intégrer dans nos propositions.

6.2.7 Autres approches d'optimisation énergétique dans les BDs

Dans ce travail, nous avons étudié le problème de l'intégration de la contrainte énergétique dans le processus de traitement des requêtes. Cependant, similairement au module de traitement des requêtes d'autres phases composant le SGBD peuvent être explorées telles que: la phase de conception physique ou la phase de la modélisation conceptuelle. Pour la phase de conception physique, il s'agirait de considérer le coût énergétique dans le processus de la sélection des structures d'optimisation comme les index, les vues matérialisée, et les schémas de fragmentation. Par exemple le gestionnaire de disposition du système *Hyrise* suggère la meilleure disposition des données au gestionnaire de stockage sur la base de l'analyse des performances d'exécution de la charge des requêtes fournies. Il serait intéressant de suggérer une disposition sur la base de la contrainte énergétique afin d'évaluer le gain énergétique. Pour la phase de la modélisation

conceptuelle, il s'agirait de jouer sur la variabilité des méthodes de conception, des types de données et des contraintes d'intégrité.

Bibliographie

[1] MonetDB kernel description. `https://www.monetdb.org/Documentation/MonetDBInternals`. Accédé: 2021-05-20. (Cité en page 129)

[2] D. Abadi, R. Agrawal, A. Ailamaki, M. Balazinska, P. A. Bernstein, M. J. Carey, S. Chaudhuri, J. Dean, A. Doan, M. J. Franklin, et al. The beckman report on database research. *Communications of the ACM*, 59(2):92–99, 2016. (Cité en page 17)

[3] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. Hadoopdb: an architectural hybrid of mapreduce and dbms technologies for analytical workloads. *Proceedings of the VLDB Endowment*, 2(1):922–933, 2009. (Cité en page 42)

[4] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia-Molina, et al. The claremont report on database research. *ACM Sigmod Record*, 37(3):9–19, 2008. (Cité en page 17)

[5] A. Alimonda, A. Acquaviva, S. Carta, and A. Pisano. A control theoretic approach to run-time energy optimization of pipelined processing in mpsocs. In *Proceedings of the Conference on Design, Automation and Test in Europe, DATE 2006, Munich, Germany, March 6-10, 2006*, pages 876–877, 2006. (Cité en page 56)

[6] A. M. Amin and Z. Chishti. Rank-aware cache replacement and write buffering to improve DRAM energy efficiency. In V. G. Oklobdzija, B. Pangle, N. Chang, N. R. Shanbhag, and C. H. Kim, editors, *Proceedings of the 2010 International Symposium on Low Power Electronics and Design, 2010, Austin, Texas, USA, August 18-20, 2010*, pages 383–388. ACM, 2010. (Cité en page 57), (Cité en page 62)

[7] V. Avelar, D. Azevedo, A. French, and E. N. Power. Pue: a comprehensive examination of the metric. *White paper*, 49, 2012. (Cité en page 52)

[8] B. Bani, F. Khomh, and Y.-G. Guéhéneuc. A study of the energy consumption of databases and cloud patterns. In *International Conference on Service-Oriented Computing*, pages 606–614. Springer, 2016. (Cité en page 44)

[9] P. Behzadnia, W. Yuan, B. Zeng, Y.-C. Tu, and X. Wang. Dynamic power-aware disk storage management in database servers. In *DEXA*, pages 315–325. Springer International Publishing, 2016. (Cité en page 62), (Cité en page 63)

[10] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias. A data mining approach for selecting bitmap join indices. *J. Comput. Sci. Eng.*, 1(2):177–194, 2007. (Cité en page 121)

[11] L. Bellatreche, R. Missaoui, H. Necir, and H. Drias. Selection and pruning algorithms for bitmap index selection problem using data mining. In I. Y. Song, J. Eder, and T. M. Nguyen, editors, *9th International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 221–230. Springer, 2007. (Cité en page 121)

[12] A. Beloglazov, R. Buyya, Y. Lee, and A. Y. Zomaya. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *ArXiv*, abs/1007.0066, 2011. (Cité en page 55), (Cité en page 58), (Cité en page 59), (Cité en page 62), (Cité en page 63)

[13] P. A. Boncz, M. L. Kersten, and S. Manegold. Breaking the memory wall in monetdb. *Commun. ACM*, 51(12):77–85, 2008. (Cité en page 78)

[14] C. Bondiombouy and P. Valduriez. Query processing in multistore systems: an overview. *International Journal of Cloud Computing*, 5(4):309–346, 2016. (Cité en page 41)

[15] L. Bouganim. Exécution parallèle de requêtes relationnelles et équilibrage de charge. (Cité en page 38)

[16] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. (Cité en page 96)

[17] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. *ACM SIGARCH Computer Architecture News*, 28(2):83–94, 2000. (Cité en page 53)

[18] R. E. Brown, R. Brown, E. Masanet, B. Nordman, B. Tschudi, A. Shehabi, J. Stanley, J. Koomey, D. Sartor, P. Chan, et al. Report to congress on server and data center energy efficiency: Public law 109-431. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States), 2007. (Cité en page 15)

[19] D. Bui, Y. Yoon, E. Huh, S. Jun, and S. Lee. Energy efficiency for cloud computing system based on predictive optimization. *J. Parallel Distributed Comput.*, 102:103–114, 2017. (Cité en page 59)

[20] M. Catena and N. Tonellotto. Energy-efficient query processing in web search engines. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1412–1425, July 2017. (Cité en page 62)

[21] S. Ceri and G. Pelagatti. Allocation of operations in distributed database access. *IEEE Transactions on Computers*, pages 119–129, 1982. (Cité en page 40)

[22] S. Chakraborty, D. Deb, D. Buragohain, and H. Kapoor. Cache capacity and its effects on power consumption for tiled chip multi-processors. pages 1–6, 02 2014. (Cité en page 57)

[23] S. V. Chande and M. Sinha. Genetic optimization for the join ordering problem of database queries. In *2011 Annual IEEE India Conference*, pages 1–5. IEEE, 2011. (Cité en page 37)

[24] S. Chaudhuri, V. Narasayya, and R. Ramamurthy. Estimating progress of execution for sql queries. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 803–814, New York, NY, USA, 2004. ACM. (Cité en page 89)

[25] N. S. Chauhan and A. Saxena. A green software development life cycle for cloud computing. *It Professional*, 15(1):28–34, 2013. (Cité en page 59)

[26] X. Cheng, B. He, and C. T. Lau. Energy-efficient query processing on embedded cpu-gpu architectures. In *Proceedings of the 11th International Workshop on Data Management on New Hardware*, page 10. ACM, 2015. (Cité en page 62), (Cité en page 63)

[27] L. Cupertino, G. Da Costa, A. Oleksiak, W. Pia, J.-M. Pierson, J. Salom, L. Siso, P. Stolf, H. Sun, T. Zilio, et al. Energy-efficient, thermal-aware modeling and simulation of data centers: the coolemall approach and evaluation results. *Ad Hoc Networks*, 25:535–553, 2015. (Cité en page 52)

[28] E. R. D. Anderson, J. Dykes. More than an interface — scsi vs. ata. *in 2nd USENIX Conference on File and Storage Technologies (FAST03)*, page 245– 257, 2003. (Cité en page 17), (Cité en page 51)

[29] M. Dayarathna, Y. Wen, and R. Fan. Data center energy consumption modeling: A survey. *IEEE Commun. Surv. Tutorials*, 18(1):732–794, 2016. (Cité en page 52), (Cité en page 53)

[30] D. DeWitt and J. Gray. Parallel database systems: the future of high performance database systems. *Communications of the ACM*, 35(6):85–98, 1992. (Cité en page 38)

[31] S. Dhar. From outsourcing to cloud computing: evolution of it services. *Management Research Review*, 2012. (Cité en page 17)

[32] G. Dhiman, R. Z. Ayoub, and T. Rosing. PDRAM: a hybrid PRAM and DRAM main memory system. In *Proceedings of the 46th Design Automation Conference, DAC 2009, San Francisco, CA, USA, July 26-31, 2009*, pages 664–469. ACM, 2009. (Cité en page 56)

[33] M. E. M. Diouri, O. Glück, L. Lefevre, and J.-C. Mignot. Your cluster is not power homogeneous: Take care when designing green schedulers! In *2013 International Green Computing Conference Proceedings*, pages 1–10. IEEE, 2013. (Cité en page 46)

[34] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Elsevier, 2012. (Cité en page 31), (Cité en page 36)

[35] M. Dreseler, J. Kossmann, M. Boissier, S. Klauck, M. Uflacker, and H. Plattner. Hyrise re-engineered: An extensible database system for research in relational in-memory data management. In *EDBT*, pages 313–324, 2019. (Cité en page 79)

[36] M. Dreseler, J. Kossmann, M. Boissier, S. Klauck, M. Uflacker, and H. Plattner. Hyrise re-engineered: An extensible database system for research in relational in-memory data management. In *EDBT*, 2019. (Cité en page 80)

[37] K. Ebrahimi, G. F. Jones, and A. S. Fleischer. A review of data center cooling technology, operating conditions and the corresponding low-grade waste heat recovery opportunities. *Renewable and Sustainable Energy Reviews*, 31:622 – 63, 2014. (Cité en page 60)

[38] D. Economou, S. Rivoire, C. Kozyrakis, and P. Ranganathan. Full-system power analysis and modeling for server environments. International Symposium on Computer Architecture (IEEE), 2006. (Cité en page 53)

[39] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Pearson, 6th edition, 2015. (Cité en page 18), (Cité en page 32), (Cité en page 34)

[40] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya. Virtual machine consolidation in cloud data centers using aco metaheuristic. In *European conference on parallel processing*, pages 306–317. Springer, 2014. (Cité en page 59)

[41] J. Fowers, G. Brown, P. Cooke, and G. Stitt. A performance and energy comparison of fpgas, gpus, and multicores for sliding-window applications. In *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '12, pages 47–56, New York, NY, USA, 2012. ACM. (Cité en page 58)

[42] E. Francesquini, M. Castro, P. H. Penna, F. Dupros, H. C. Freitas, P. O. Navaux, and J.-F. Méhaut. On the energy efficiency and performance of irregular application executions on multicore, numa and manycore platforms. *Journal of Parallel and Distributed Computing*, 76:32–48, 2015. (Cité en page 45)

[43] S. Friedenthal, A. Moore, and R. Steiner. *A practical guide to SysML: the systems modeling language*. Morgan Kaufmann, 2014. (Cité en page 53)

[44] J. Gama and C. Pinto. Discretization from data streams: applications to histograms and data mining. pages 662–667. ACM, 2006. (Cité en page 37)

[45] Y. Gao, H. Guan, Z. Qi, Y. Hou, and L. Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of computer and system sciences*, 79(8):1230–1242, 2013. (Cité en page 59)

[46] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database systems - the complete book (2. ed.)*. Pearson Education, 2009. (Cité en page 17), (Cité en page 29), (Cité en page 34)

[47] G. Gardarin. *Bases de données*. Eyrolles, 2003. (Cité en page 34)

[48] R. Ge, X. Feng, and K. W. Cameron. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. page 34, 2005. (Cité en page 49), (Cité en page 56)

[49] R. Genuer and J.-M. Poggi. Arbres cart et forêts aléatoires, importance et sélection de variables. 2017. (Cité en page 97)

[50] M. Ghosh and H.-H. S. Lee. Smart refresh: An enhanced memory controller design for reducing energy in conventional and 3d die-stacked drams. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 134–145, Washington, DC, USA, 2007. IEEE Computer Society. (Cité en page 57)

[51] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Transactions on Database Systems (TODS)*, 27(3):261–298, 2002. (Cité en page 43)

[52] R. Gonçalves, J. Saraiva, and O. Belo. Defining energy consumption plans for data querying processes. In *2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, pages 641–647. IEEE, 2014. (Cité en page 65), (Cité en page 68)

[53] G. Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993. (Cité en page 36)

[54] G. Graefe. Database servers tailored to improve energy efficiency. In *Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management*, pages 24–28, 2008. (Cité en page 17), (Cité en page 64), (Cité en page 67)

[55] T. P. G. D. Group. Documentation postgresql 12.2. 2020. (Cité en page 77)

[56] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden. Hyrise: A main memory hybrid storage engine. *Proc. VLDB Endow.*, 4(2), 2010. (Cité en page 79), (Cité en page 80)

[57] M. Guimarães, J. Saraiva, and O. Belo. Some heuristic approaches for reducing energy consumption on database systems. *DBKDA 2016*, page 59, 2016. (Cité en page 65), (Cité en page 68)

[58] B. Guo, J. Yu, B. Liao, D. Yang, and L. Lu. A green framework for dbms based on energy-aware query optimization and energy-efficient query processing. *Journal of Network and Computer Applications*, 84:118–130, 2017. (Cité en page 66), (Cité en page 67), (Cité en page 68)

[59] S. Gurumurthi, A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, and M. Kandemir. Using complete machine simulation for software power estimation: The softwatt approach. In *Proceedings Eighth International Symposium on High Performance Computer Architecture*, pages 141–150. IEEE, 2002. (Cité en page 53)

[60] J. Haj-Yahya, M. Alser, J. Kim, A. G. Yaglikçi, N. Vijaykumar, E. Rotem, and O. Mutlu. Sysscale: Exploiting multi-domain dynamic voltage and frequency scaling for energy efficient mobile processors. *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 227–240, 2020. (Cité en page 56)

[61] A. Hammadi and L. Mhamdi. A survey on architectures and energy efficiency in data center networks. *Computer Communications*, 40:1 – 21, 2014. (Cité en page 60)

[62] S. Harizopoulos, M. Shah, J. Meza, and P. Ranganathan. Energy efficiency: The new holy grail of data management systems research. *arXiv preprint arXiv:0909.1784*, 2009. (Cité en page 17), (Cité en page 64)

[63] A. Hassan, H. Vandierendonck, and D. S. Nikolopoulos. Energy-efficient in-memory data stores on hybrid memory hierarchies. In *Proceedings of the 11th International Workshop on Data Management on New Hardware*, page 1. ACM, 2015. (Cité en page 62), (Cité en page 63)

[64] H. Hassan, M. Patel, J. S. Kim, A. G. Yaglikci, N. Vijaykumar, N. M. Ghiasi, S. Ghose, and O. Mutlu. Crow: A low-cost substrate for improving dram performance, energy efficiency, and reliability. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 129–142, 2019. (Cité en page 57), (Cité en page 62)

[65] Y. Hayamizu, K. Goda, M. Nakano, and M. Kitsuregawa. Application-aware power saving for online transaction processing using dynamic voltage and frequency scaling in a multicore environment. In *International Conference on Architecture of Computing Systems*, pages 50–61, Berlin, Heidelberg, 2011. Springer. (Cité en page 61), (Cité en page 62)

[66] Y. Hirano, T. Satoh, U. Inoue, and K. Teranaka. Load balancing algorithms for parallel database processing on shared memory multiprocessors. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 210–217. IEEE, 1991. (Cité en page 38)

[67] E. J. Hogbin. Acpi: Advanced configuration and power interface, 2015. (Cité en page 57)

[68] H. Höpfner and C. Bunse. Towards an energy aware dbms-energy consumptions of sorting and join algorithms. In *Grundlagen von Datenbanken*, pages 69–73, 2009. (Cité en page 44)

[69] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers*, 56(4):444–458, 2007. (Cité en page 56)

[70] C.-h. Hsu and W.-c. Feng. A power-aware run-time system for high-performance computing. In *SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, pages 1–1. IEEE, 2005. (Cité en page 56)

[71] C.-H. Hsu and S. W. Poole. Measuring server energy proportionality. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, pages 235–240, 2015. (Cité en page 54)

[72] A. Hurson and H. Azad. *Energy Efficiency in Data Centers and Clouds*. Academic Press, 2016. (Cité en page 62), (Cité en page 63)

[73] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. S. Mullender, and M. L. Kersten. Monetdb: Two decades of research in column-oriented database architectures. *IEEE Data Eng. Bull.*, 35:40–45, 2012. (Cité en page 79)

[74] R. Ihaka and R. Gentleman. R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3):299–314, 1996. (Cité en page 98)

[75] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, volume 99, pages 7–10, 1999. (Cité en page 43)

[76] A. Jaiantilal, Y. Jiang, and S. Mishra. Modeling cpu energy consumption for energy efficient scheduling. In *Proceedings of the 1st Workshop on Green Computing*, pages 10–15, 2010. (Cité en page 56)

[77] M. Jarke and J. Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16(2):111–152, 1984. (Cité en page 34)

[78] R. Jauregui and F. Silva. Numerical validation methods. *Numerical analysis—theory and application*, pages 155–174, 2011. (Cité en page 76)

[79] C. Jiang, T. Fan, H. Gao, W. Shi, L. Liu, C. Cérin, and J. Wan. Energy aware edge computing: A survey. *Computer Communications*, 151:556–580, 2020. (Cité en page 59)

[80] W. Kang and J. Chung. Qos management for embedded databases in multicore-based embedded systems. *Mobile Information Systems*, 2015, 2015. (Cité en page 62)

[81] W. Kang and J. Chung. Energy-efficient response time management for embedded databases. *Real-Time Systems*, 53(2):228–253, 2017. (Cité en page 62)

[82] W. Kang, S. H. Son, and J. A. Stankovic. Power-aware data buffer cache management in real-time embedded databases. In *RTCSA*, pages 35–44, 2008. (Cité en page 67), (Cité en page 68)

[83] K. Kant. Data center evolution: A tutorial on state of the art, issues, and challenges. *Computer Networks*, 53(17):2939–2965, 2009. Virtualized Data Centers. (Cité en page 17)

[84] A. Karyakin and K. Salem. An analysis of memory power consumption in database systems. In *Proceedings of the 13th International Workshop on Data Management on New Hardware*, page 2. ACM, 2017. (Cité en page 62), (Cité en page 63)

[85] T. Kaur and I. Chana. Energy efficiency techniques in cloud computing: A survey and taxonomy. *ACM Comput. Surv.*, 48, Oct. 2015. (Cité en page 59)

[86] K. Kennedy. Fast greedy weighted fusion. *Int. J. Parallel Program.*, 29(5):463–491, 2001. (Cité en page 36)

[87] M. Khandelwal and T. Singh. Evaluation of blast-induced ground vibration predictors. *Soil Dynamics and Earthquake Engineering*, 27(2):116–125, 2007. (Cité en page 96)

[88] A. Khosravi, S. K. Garg, and R. Buyya. Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In *European Conference on Parallel Processing*, pages 317–328. Springer, 2013. (Cité en page 59)

[89] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper. Learned cardinalities: Estimating correlated joins with deep learning. *arXiv preprint arXiv:1809.00677*, 2018. (Cité en page 143)

[90] T. Kissinger, D. Habich, and W. Lehner. Adaptive energy-control for in-memory database systems. In *Proceedings of the 2018 International Conference on Management of Data*, pages 351–364. ACM, 2018. (Cité en page 62), (Cité en page 64)

[91] M. Korkmaz, M. Karsten, K. Salem, and S. Salihoglu. Workload-aware cpu performance scaling for transactional database systems. In *Proceedings of the 2018 International Conference on Management of Data*, pages 291–306. ACM, 2018. (Cité en page 62)

[92] D. Kossmann and K. Stocker. Iterative dynamic programming: a new class of query optimization algorithms. *ACM Transactions on Database Systems (TODS)*, 25(1):43–82, 2000. (Cité en page 36)

[93] M. Kunjir, P. K. Birwa, and J. R. Haritsa. Peak power plays in database engines. In *EDBT*, pages 444–455. ACM, 2012. (Cité en page 65), (Cité en page 68)

[94] M. H. Kutner, C. J. Nachtsheim, J. Neter, W. Li, et al. *Applied linear statistical models*, volume 5. McGraw-Hill Irwin New York, 2005. (Cité en page 93)

[95] W. Lang, S. Harizopoulos, J. M. Patel, M. A. Shah, and D. Tsirogiannis. Towards energy-efficient database cluster design. *arXiv preprint arXiv:1208.1933*, 2012. (Cité en page 68)

[96] W. Lang, R. Kandhan, and J. M. Patel. Rethinking query processing for energy efficiency: Slowing down to win the race. *IEEE Data Eng. Bull.*, 34(1):12–23, 2011. (Cité en page 17), (Cité en page 66), (Cité en page 67), (Cité en page 68)

[97] W. Lang and J. Patel. Towards eco-friendly database management systems. *arXiv preprint arXiv:0909.1767*, 2009. (Cité en page 61), (Cité en page 62), (Cité en page 67), (Cité en page 68)

[98] W. Lang and J. M. Patel. Energy management for mapreduce clusters. *Proceedings of the VLDB Endowment*, 3(1-2):129–139, 2010. (Cité en page 53)

[99] R. S. G. Lanzelotte, P. Valduriez, and M. Zaït. On the effectiveness of optimization search strategies for parallel execution spaces. pages 493–504. Morgan Kaufmann, 1993. (Cité en page 36)

[100] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association. (Cité en page 57)

[101] A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis. Power aware page allocation. *ACM Sigplan Notices*, 35(11):105–116, 2000. (Cité en page 57)

[102] X. Liu, P. Liu, L. Hu, C. Zou, and Z. Cheng. Energy-aware task scheduling with time constraint for heterogeneous cloud datacenters. *Concurrency and Computation: Practice and Experience*, 32(18):e5437, 2020. (Cité en page 62), (Cité en page 63)

[103] X. Liu, J. Wang, H. Wang, and H. Gao. Generating power-efficient query execution plan. In *2nd International Conference on Advances in Computer Science and Engineering (CSE 2013)*. Atlantis Press, 2013. (Cité en page 65), (Cité en page 68)

[104] Y. Liu, X. Wei, J. Xiao, Z. Liu, Y. Xu, and Y. Tian. Energy consumption and emission mitigation prediction based on data center traffic and pue for global data centers. *Global Energy Interconnection*, 3(3):272–282, 2020. (Cité en page 15)

[105] P. Llopis, J. G. Blas, F. Isaila, and J. Carretero. Survey of energy-efficient and power-proportional storage systems. *The Computer Journal*, 57(7):1017–1032, 2014. (Cité en page 56)

[106] G. Louppe. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502*, 2014. (Cité en page 96)

[107] H. Lu, B.-C. Ooi, and C.-H. Goh. Multidatabase query optimization: Issues and solutions. In *Proceedings RIDE-IMS93: Third International Workshop on Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, pages 137–143. IEEE, 1993. (Cité en page 41)

[108] Y.-H. Lu and G. De Micheli. Comparing system level power management policies. *IEEE Design & test of Computers*, 18(2):10–19, 2001. (Cité en page 55)

[109] D. Mahajan, C. Blakeney, and Z. Zong. Improving the energy efficiency of relational and nosql databases via query optimizations. *Sustainable Computing: Informatics and Systems*, 22:120–133, 2019. (Cité en page 44)

[110] S. B. Maind, P. Wankar, et al. Research paper on basic of artificial neural network. *International Journal on Recent and Innovation Trends in Computing and Communication*, 2(1):96–100, 2014. (Cité en page 95)

[111] S. Manegold et al. *Understanding, modeling, and improving main-memory database performance*. Universiteit van Amsterdam [Host], 2002. (Cité en page 43), (Cité en page 53), (Cité en page 91), (Cité en page 122)

[112] R. Marcus, A. Kipf, A. van Renen, M. Stoian, S. Misra, A. Kemper, T. Neumann, and T. Kraska. Benchmarking learned indexes. *Proc. VLDB Endow.*, 14(1):1–13, 2020. (Cité en page 143)

[113] J. F. Mas and J. J. Flores. The application of artificial neural networks to the analysis of remotely sensed data. *International Journal of Remote Sensing*, 29(3):617–663, 2008. (Cité en page 94)

[114] M. Mehta and D. J. DeWitt. Dynamic memory allocation for multiple-query workloads. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 1993. (Cité en page 38)

[115] M. Mehta and D. J. DeWitt. Data placement in shared-nothing parallel database systems. *The VLDB Journal*, 6(1):53–72, 1997. (Cité en page 38)

[116] A. Merkel and F. Bellosa. Memory-aware scheduling for energy efficiency on multicore processors. In F. Zhao, editor, *Workshop on Power Aware Computing and Systems, HotPower 2008, December 7, 2008, San Diego, CA, USA, Proceedings*. USENIX Association, 2008. (Cité en page 56)

[117] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell. *Machine learning: An artificial intelligence approach*. Springer, 2013. (Cité en page 53)

[118] M. Mitchell, J. H. Holland, and S. Forrest. When will a genetic algorithm outperform hill climbing. In *Advances in neural information processing systems*, pages 51–58, 1994. (Cité en page 37)

[119] S. Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *CoRR*, abs/1401.0765, 2014. (Cité en page 16), (Cité en page 51), (Cité en page 58), (Cité en page 62)

[120] D. C. Montgomery, E. A. Peck, and G. G. Vining. *Introduction to linear regression analysis*, volume 821. John Wiley & Sons, 2012. (Cité en page 92)

[121] N. J. Nagelkerke et al. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991. (Cité en page 101)

[122] R. O. Nambiar and M. Poess. The making of tpc-ds. In *VLDB*, volume 6, pages 1049–1058, 2006. (Cité en page 106)

[123] R. Nathuji and K. Schwan. Virtualpower: Coordinated power management in virtualized enterprise systems. volume 41, pages 265–278, 01 2007. (Cité en page 59)

[124] N. Nishikawa, M. Nakano, and M. Kitsuregawa. Application sensitive energy management framework for storage systems. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2335–2348, Sep. 2015. (Cité en page 56)

[125] D. Niu, Y. Chen, C. Xu, and Y. Xie. Impact of process variations on emerging memristor. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 877–882. IEEE, 2010. (Cité en page 57)

[126] S. Noll, H. Funke, and J. Teubner. Energy efficiency in main-memory databases. *Datenbank-Spektrum*, 17(3):223–232, 2017. (Cité en page 62)

[127] I. NUNES and H. S. DA SILVA. *Artificial neural networks: a practical course*. Springer, 2018. (Cité en page 94), (Cité en page 95), (Cité en page 96)

[128] A. Omer. Energy use and environmental impacts: A general review. *Journal of Renewable and Sustainable Energy*, 1, 09 2009. (Cité en page 15), (Cité en page 50), (Cité en page 51), (Cité en page 183)

[129] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas. Demystifying energy consumption in grids and clouds. In *International Conference on Green Computing*, pages 335–342. IEEE, 2010. (Cité en page 46)

[130] E. Ostertagová. Modelling using polynomial regression. *Procedia Engineering*, 48:500–506, 2012. (Cité en page 94)

[131] M. T. Özsu and P. Valduriez. *Principles of distributed database systems*, volume 2. Springer, 1999. (Cité en page 40), (Cité en page 41)

[132] P. E. O'Neil, E. J. O'Neil, and X. Chen. The star schema benchmark (ssb). *Pat*, 200(0):50, 2007. (Cité en page 106)

[133] S. Pagani, A. Pathania, M. Shafique, J. Chen, and J. Henkel. Energy efficiency for clustered heterogeneous multicores. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1315–1330, May 2017. (Cité en page 57), (Cité en page 62)

[134] D. S. Palmer, N. M. O'Boyle, R. C. Glen, and J. B. O. Mitchell. Random forest models to predict aqueous solubility. *J. Chem. Inf. Model.*, 47(1):150–158, 2007. (Cité en page 113)

[135] Y. Park, D.-J. Shin, S. K. Park, and K. H. Park. Power-aware memory management for hybrid main memory. In *Next Generation Information Technology (ICNIT), 2011 The 2nd International Conference on*, pages 82–85. IEEE, 2011. (Cité en page 57)

[136] M. K. Patterson. The effect of data center temperature on energy efficiency. In *2008 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, pages 1167–1174. IEEE, 2008. (Cité en page 46)

[137] J. Pezet. Le coronavirus a-t-il eu pour effet de diminuer la pollution? `https://www.liberation.fr/checknews/2020/03/09/le-coronavirus-a-t-il-eu-pour-effet-de-diminuer-la-pollution_1780537`. 9/03/2020 (Consulté le 12/10/2020). (Cité en page 17), (Cité en page 183)

[138] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 89–102, 2001. (Cité en page 56)

[139] M. Poess and R. O. Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results. *Proc. VLDB Endow.*, 1(2):1229–1240, 2008. (Cité en page 15), (Cité en page 17)

[140] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita. Improved histograms for selectivity estimation of range predicates. *ACM Sigmod Record*, 25(2):294–305, 1996. (Cité en page 37), (Cité en page 43)

[141] M. B. Prasad, N. Nimisha, T. Tejaswi, B. Bhavani, and K. Prudhviraju. Enhanced energy query processing in web searching. *Journal of Information and Computational Science*, 10(3), 2020. (Cité en page 62)

[142] S. Prezenski, A. Brechmann, S. Wolff, and N. Russwinkel. A cognitive modeling approach to strategy formation in dynamic decision making. *Frontiers in psychology*, 8:1335, 2017. (Cité en page 94)

[143] I. Psaroudakis and et al. Dynamic fine-grained scheduling for energy-efficient main-memory queries. In *DaMoN*, page 1, 2014. (Cité en page 67), (Cité en page 68)

[144] N. Quang-Hung, N. Thoai, and N. T. Son. Enery efficient allocation of virtual machines in high performance computing cloud. *CoRR*, abs/1310.7801, 2013. (Cité en page 59)

[145] M. Raasveldt. Monetdblite: An embedded analytical database. In G. Das, C. M. Jermaine, and P. A. Bernstein, editors, *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1837–1838. ACM, 2018. (Cité en page 79)

[146] E. Rahm and R. Marek. Analysis of dynamic load balancing strategies for parallel shared nothing database systems. In *VLDB*, pages 182–193, 1993. (Cité en page 38)

[147] E. Rahm and R. Marek. Dynamic multi-resource load balancing in parallel database systems. In *VLDB*, volume 95, pages 11–15, 1995. (Cité en page 38)

[148] R. Ramakrishnan and J. Gehrke. *Database management systems*. McGraw-Hill, 2000. (Cité en page 36)

[149] S. M. Rivoire. *Models and metrics for energy-efficient computer systems*. Stanford University, 2008. (Cité en page 53), (Cité en page 54)

[150] S. M. Rivoire. *Models and metrics for energy-efficient computer systems*. PhD thesis, Stanford University, 2008. (Cité en page 60)

[151] M. Rodriguez-Martinez, H. Valdivia, J. Seguel, and M. Greer. Estimating power/energy consumption in database servers. *Procedia Computer Science*, 6:112–117, 2011. (Cité en page 65), (Cité en page 68)

[152] A. Roukh. Estimating power consumption of batch query workloads. In *International Conference on Model and Data Engineering*, pages 198–212. Springer, 2015. (Cité en page 66), (Cité en page 68)

[153] A. Roukh, L. Bellatreche, , N. Tziritas, and C. Ordonez. Energy-aware query processing on parallel database cluster nodes. In *International Conference on Algorithms and Architectures for Parallel Processing*, pages 260–269. Springer, 2016. (Cité en page 67), (Cité en page 68)

[154] A. Roukh and L. Bellatreche. Eco-processing of olap complex queries. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 229–242. Springer, 2015. (Cité en page 66), (Cité en page 68)

[155] A. Roukh, L. Bellatreche, S. Bouarar, and A. Boukorca. Eco-physic: Eco-physical design initiative for very large databases. *Information Systems*, 68:44–63, 2017. (Cité en page 67), (Cité en page 68)

[156] A. Roukh, L. Bellatreche, A. Boukorca, and S. Bouarar. Eco-dmw: Eco-design methodology for data warehouses. In *Proceedings of the ACM Eighteenth International Workshop on Data Warehousing and OLAP*, pages 1–10, 2015. (Cité en page 68), (Cité en page 119), (Cité en page 123)

[157] A. Roukh, L. Bellatreche, and C. Ordonez. Enerquery: Energy-aware query processing. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2465–2468. ACM, 2016. (Cité en page 16), (Cité en page 86), (Cité en page 89), (Cité en page 183)

[158] B. Salami, H. Noori, and M. Naghibzadeh. Fairness-aware energy efficient scheduling on heterogeneous multi-core processors. *IEEE Transactions on Computers*, pages 1–1, 2020. (Cité en page 57)

[159] V. Saravanan, S. K. Chandran, S. Punnekkat, and D. Kothari. A study on factors influencing power consumption in multithreaded and multicore cpus. *WSEAS Transactions on Computers*, 10(3):93–103, 2011. (Cité en page 45)

[160] Y. Sato. Energy consumption: an environmental problem. *IEEJ Transactions on Electrical and Electronic Engineering*, 2(1):12–16, 2007. (Cité en page 50)

[161] D. Schall, V. Hudlet, and T. Härder. Enhancing energy efficiency of database applications using ssds. In *Proceedings of the Third C* Conference on Computer Science and Software Engineering*, pages 1–9. ACM, 2010. (Cité en page 62), (Cité en page 63)

[162] M. R. Segal. Machine learning benchmarks and random forest regression. 2004. (Cité en page 96)

[163] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34, 1979. (Cité en page 36), (Cité en page 161), (Cité en page 165), (Cité en page 166)

[164] H. Shafi, P. J. Bohrer, J. Phelan, C. A. Rusu, and J. L. Peterson. Design and validation of a performance and power simulator for powerpc systems. *IBM Journal of Research and Development*, 47(5.6):641–651, 2003. (Cité en page 53)

[165] K. Shalini and K. N. Prasanthi. Green computing. *Journal of Telematics and Informatics*, 1(1):1–13, 2013. (Cité en page 59)

[166] L. A. Sharrott. Centralized and distributed information systems: two architecture approaches for the 90s. In *Healthcare information management systems*, pages 306–315. Springer, 1991. (Cité en page 45)

[167] A. Shehabi, S. J. Smith, D. A. Sartor, R. E. Brown, M. Herrlin, J. G. Koomey, E. R. Masanet, N. Horner, I. L. Azevedo, and W. Lintner. United states data center energy usage report. Report, Energy Technology Area, June 2016. (Cité en page 15), (Cité en page 49)

[168] Y. Shin, K. Choi, and T. Sakurai. Power-conscious scheduling for real-time embedded systems design. *VLSI Design*, 12(2):139–150, 2001. (Cité en page 56)

[169] J. Shuja, K. Bilal, S. A. Madani, and S. U. Khan. Data center energy efficient resource scheduling. *Cluster Computing*, 17(4):1265–1277, 2014. (Cité en page 15)

[170] J. Shuja, K. Bilal, S. A. Madani, M. Othman, R. Ranjan, P. Balaji, and S. U. Khan. Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Systems Journal*, 10(2):507–519, June 2016. (Cité en page 58), (Cité en page 62)

[171] A. Silberschatz, H. F. Korth, and S. Sudarshan. *Database System Concepts, Sixth Edition*. McGraw-Hill Book Company, 2011. (Cité en page 29)

[172] A. Silberschatz, M. Stonebraker, and J. Ullman. Database research: achievements and opportunities into the 1st century. *ACM sIGMOD record*, 25(1):52–63, 1996. (Cité en page 30)

[173] J. M. Smith and P. Y.-T. Chang. Optimizing the performance of a relational algebra database interface. *Communications of the ACM*, 18(10):568–579, 1975. (Cité en page 36)

[174] M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal*, 6(3):191–208, 1997. (Cité en page 36)

[175] C. Timm, A. Gelenberg, P. Marwedel, and F. Weichert. Reducing the energy consumption of embedded systems by integrating general purpose gpus. 06 2010. (Cité en page 58)

[176] C.-F. Tsai and J.-W. Wu. Using neural network ensembles for bankruptcy prediction and credit scoring. *Expert systems with applications*, 34(4):2639–2649, 2008. (Cité en page 94)

[177] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah. Analyzing the energy efficiency of a database server. In *sigmod*, pages 231–242, 2010. (Cité en page 50), (Cité en page 67), (Cité en page 68)

[178] Y.-C. Tu, X. Wang, B. Zeng, and Z. Xu. A system for energy-efficient data management. *ACM SIGMOD Record*, 43(1):21–26, 2014. (Cité en page 61), (Cité en page 62), (Cité en page 63), (Cité en page 68)

[179] J. von Kistowski, H. Block, J. Beckett, C. Spradling, K.-D. Lange, and S. Kounev. Variations in cpu power consumption. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, pages 147–158, 2016. (Cité en page 45)

[180] J. Wang, L. Feng, W. Xue, and Z. Song. A survey on energy-efficient data management. *ACM SIGMOD Record*, 40(2):17–23, 2011. (Cité en page 52), (Cité en page 53)

[181] X. Wang, Y. Chen, H. Li, D. Dimitrov, and H. Liu. Spin torque random access memory down to 22 nm technology. *IEEE transactions on magnetics*, 44(11):2479–2482, 2008. (Cité en page 57)

[182] Y. Wang. *Evaluating and modeling the energy impacts of data centers, in terms of hardware/software architecture and associated environment*. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique, 2020. (Cité en page 59)

[183] T. Wilde. *Assessing the Energy Efficiency of High Performance Computing (HPC) Data Centers*. PhD thesis, Technische Universität München, 2018. (Cité en page 53)

[184] Z. Xu. Building a power-aware database management system. In *Proceedings of the Fourth SIGMOD PhD Workshop on Innovative Database Research*, pages 1–6, 2010. (Cité en page 64), (Cité en page 68)

[185] Z. Xu, Y.-C. Tu, and X. Wang. Exploring power-performance tradeoffs in database systems. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 485–496. IEEE, 2010. (Cité en page 64), (Cité en page 67), (Cité en page 68)

[186] Z. Xu, Y.-C. Tu, and X. Wang. Pet: reducing database energy cost via query optimization. *PVLDB*, 5(12):1954–1957, 2012. (Cité en page 64), (Cité en page 68), (Cité en page 119)

[187] Z. Xu, Y.-C. Tu, and X. Wang. Dynamic energy estimation of query plans in database systems. In *2013 IEEE 33rd International Conference on Distributed Computing Systems*, pages 83–92. IEEE, 2013. (Cité en page 64), (Cité en page 68)

[188] Z. Xu, Y.-C. Tu, and X. Wang. Online energy estimation of relational operations in database systems. *IEEE transactions on computers*, 64(11):3223–3236, 2015. (Cité en page 64), (Cité en page 68)

[189] Z. Xu, X. Wang, and Y.-C. Tu. Power-aware throughput control for database management systems. In *ICAC*, pages 315–324, 2013. (Cité en page 61), (Cité en page 62), (Cité en page 68)

[190] C. Yang, Y. Du, Z. Du, and X. Meng. Micro analysis to enable energy-efficient database systems. In *EDBT*, 2020. (Cité en page 62)

[191] X. Yu, G. Li, C. Chai, and N. Tang. Reinforcement learning with tree-lstm for join order selection. In *ICDE*, pages 1297–1308. IEEE, 2020. (Cité en page 143)

[192] A. Zaina, A. Reinhardt, and J. Huchtkoetter. Relational or non-relational? a comparative evaluation of database solutions for energy consumption data. In *Proceedings of the Ninth International Conference on Future Energy Systems*, pages 474–476, 2018. (Cité en page 44)

[193] W. Zang and A. Gordon-Ross. A survey on cache tuning from a power/energy perspective. *ACM Comput. Surv.*, 45(3):32:1–32:49, 2013. (Cité en page 56)

[194] L. Zhou, L. N. Bhuyan, and K. K. Ramakrishnan. *Swan: A Two-Step Power Management for Distributed Search Engines*, page 67–72. 2020. (Cité en page 56)

[195] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ACM SIGARCH computer architecture news*, volume 37, pages 14–23. ACM, 2009. (Cité en page 57)

[196] Y. Zhou, S. Taneja, M. Alghamdi, and X. Qin. Improving energy efficiency of database clusters through prefetching and caching. In *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pages 388–391, May 2018. (Cité en page 57), (Cité en page 62)

[197] Y. Zhou, S. Taneja, X. Qin, W.-S. Ku, and J. Zhang. Edom: Improving energy efficiency of database operations on multicore servers. *Future Generation Computer Systems*, 2017. (Cité en page 62), (Cité en page 63)

[198] M. Ziane, M. Zaït, and P. Borla-Salamet. Parallel query processing with zigzag trees. *VLDB J.*, 2(3):277–301, 1993. (Cité en page 39)

Cinquième partie

Annexes

ANNEXE A

Modèle de coût dans un système de stockage orienté ligne

A.1 Introduction

Dans le modèle relationnel, l'optimiseur traite chaque requête en utilisant un plan exécution constitué d'un ensemble de nœuds. Chaque nœud représente un opérateur algébrique (sélection, jointure, projection), et cet opérateur est annoté par un algorithme d'exécution. Lors du traitement de la requête, l'optimiseur doit être capable de choisir un unique plan. Pour ce faire il utilise une approche qui estime le coût global de l'exécution des plans de la requête à partir des connaissances du coût des opérateurs du plan. L'estimation du coût d'exécution des opérateurs nécessite la connaissance des cardinalités en entrée et en sortie pour les opérateurs parents.

Le coût réel de l'exécution d'une requête est exprimé en unité de temps (par exemple, la seconde) qui correspond au temps consommé entre le début et la fin de l'exécution de la requête. Dans les systèmes traditionnels, ce coût englobe deux paramètres fondamentaux: (1) Le coût des entrées/sorties (ES) qui exprime le temps de chargement des données depuis les dispositifs de stockage vers la mémoire principale. (2) Le coût du processeur (CPU) qui dépend des composants physiques, des entrées/sorties et du nombre d'opérations arithmétiques et logiques.

Cette annexe présente le processus d'estimation des coûts des opérateurs dans un optimiseur traditionnel. Nous commençons par donner les fonctions d'estimation de cardinalité des résultats intermédiaires puis les formules d'estimation du coût de traitement des opérations.

A.2 Estimation de la cardinalité des opérateurs

La cardinalité d'une relation ou d'un résultat intermédiaire fait référence à son nombre de tuples. Pour l'estimer, il est nécessaire de disposer des statistiques sur la distribution des données dans les relations de base. L'optimiseur utilise des outils pour créer ou pour mettre à jour les statistiques de distribution des données dans les BDs, puis applique des fonctions sur ces statistiques en fonction des opérateurs. Les statistiques sont périodiquement recalculées par le SGBD, ou mises à jour par l'administrateur au besoin. Il est important de noter qu'une mauvaise statistique peut induire à la sélection d'un mauvais plan d'exécution.

A.2.1 cardinalité des résultats intermédiaires

L'objectif de l'estimation de la cardinalité n'est pas de donner une information exacte, mais plutôt une estimation qui permet de sélectionner un plan de requête physique considéré comme optimal. Pour une relation donnée ou un résultat intermédiaire R, on note sa cardinalité par $|R|$. Les notations utilisées dans les formules sont décrites dans le tableau A.1. Les formules de coût présentées se basent sur les travaux de Selinger et al. dans [163].

Paramètres	Description
R, S	Deux relations ou résultats intermédiaires
$\|R\|$	Nombre d'enregistrement dans la relation
$\|\|R\|\|$	Nombre de pages utilisé pour stocker la relation
$\|a\|$	Nombre de valeurs distinctes de l'attribut a
max_a	Valeur maximale de l'attribut a dans la relation
min_a	Valeur minimale de l'attribut a dans la relation
$l_{R,a}$	Longueur moyenne de la valeur de l'attribut a dans la relation R
l_R	Longueur moyenne d'un tuple de la relation

Tableau A.1 – Description des paramètres pour l'estimation de cardinalité des opérateurs.

A.2.1.1 Sélection

Soit p un prédicat de sélection appliqué sur une relation R. Nous notons le résultat de cette sélection par $\sigma_p(R)$. Pour estimer la cardinalité de ce résultat, il nous faut calculer la valeur d'un paramètre appelé facteur de sélectivité du prédicat, qui est défini comme le nombre de tuples satisfaisant le prédicat sur le nombre total de tuples dans la base, et sa valeur est dans intervalle $(0, 1)$. Soit f_p le facteur de sélectivité du prédicat p. La cardinalité du résultat de la sélection est définie comme suit:$|\sigma_p(R)| = f_p \times |R|$. Le tableau A.2 résume les différentes formules utilisées pour estimer f_p, A et B désignent des attributs et val_i dénotent des constantes ($i \in N$).

Prédicat (p)	Sélectivité (f_p)
$\neg(p)$	$1 - f_p$
$p_1 \wedge p_2$	$f_{p_1} \times f_{p_2}$
$p_1 \vee p_2$	$f_{p_1} + f_{p_2} - f_{p_1} \times f_{p_2}$
$a = val$	$\frac{1}{\|a\|}$
$a = b$	$\frac{1}{max(\|a\|,\|b\|)}$
$a > val$	$\frac{(max_a - val)}{(max_a - min_a)}$
$a < val$	$\frac{(val - min_a)}{(max_a - min_a)}$
$val_1 \leq a \geq val_2$	$\frac{(val_2 - val_1)}{(max_a - min_a)}$

Tableau A.2 – Formule de calcul de sélectivité des prédicats.

A.2.1.2 Projection

Soit A un attribut ou un ensemble d'attributs dans la relation. La cardinalité de la projection de A sur la relation est la cardinalité de la relation R. Elle se calcule comme: $\pi_A(R) = |R|$.

A.2.1.3 Produit cartésien

La cardinalité de l'évaluation du produit cartésien de deux relations est la multiplication de leurs cardinalités.Elle se formule ainsi: $|R \times S| = |R| \times |S|$.

A.2.1.4 Jointure

Plusieurs formes de jointure existent, dans cette annexe nous considérons uniquement la jointure naturelle entre deux relations $R(a, b)$ et $S(b, c)$ où b est l'attribut de jointure, a et c désignent un ensemble d'attributs. La cardinalité du résultat de jointure de R et S dépend des valeurs de l'attribut b.

— Si b est clé primaire de S et clé étrangère de R, chaque tuple de R joint uniquement un tuple de S, et $|R \bowtie S| = |R|$.

— Si tous les tuples de R et S ont la même valeur de b, donc la cardinalité de jointure équivaut à la cardinalité du produit cartésien de R et S.

— En considerant b_R et b_S comme les attributs de R et S respectivement et r comme un tuple de R et s de S. Si $|b_R| > |b_S|$, alors la valeur b de S est une valeur qui apparaît dans la valeur b de R. Par conséquent, la probabilité que r et s partagent les mêmes valeurs de b est $\frac{1}{|b_R|}$. Aussi, si $|b_R| < |b_S|$, alors la probabilité que r et s partagent les mêmes valeurs de b est $\frac{1}{|b_S|}$. Dans l'opération de jointure, le nombre de comparaison possible des tuples r et s est $|R| \times |S|$, donc la cardinalité de la jointure naturelle est:

$$R \bowtie S = \frac{max(|R|, |S|)}{max(|b_R|, |b_S|)} \tag{A.1}$$

A.2.1.5 Union

La cardinalité de l'union impliquant deux relations est estimée comme:

$$|R \cup S| = max(|R|, |S|) \; ou \; |R \cup S| = |R| + |S| \tag{A.2}$$

A.2.1.6 Intersection

La cardinalité de l'intersection de R et S est comprise entre 0 et la cardinalité minimale des deux relations.Elle s'estime comme suit:

$$R \cap S = \frac{min(|R|, |S|)}{2} \tag{A.3}$$

A.2.1.7 Différence

La cardinalité de la différence de R et S s'exprime par la formule suivante:

$$|R - S| = \frac{(|R| - |S|)}{2} \tag{A.4}$$

A.2.1.8 Agrégation

Le nombre de tuples générés à partir d'une fonction d'agrégation correspond au nombre de groupe formé.

$$|\phi(R) = \frac{|R|}{2}| \tag{A.5}$$

A.2.2 Estimation du coût des opérateurs physiques

Pour estimer le coût d'exécution d'une requête, l'optimiseur peut combiner plusieurs paramètres (disque, processeur, carte réseau). Le choix des paramètres et des pondérations dépendent de plusieurs facteurs tels que: le type de stockage, l'architecture de déploiement et la métrique à optimiser, etc. Sur une architecture locale avec les systèmes de stockage traditionnels, l'optimiseur utilise généralement le coût du processeur (C_{CPU}) et disque ($C_{E/S}$) pour estimer le coût d'exécution d'une requête donnée. Ce coût est fortement dominé par le coût de l'E/S. C'est pourquoi certaines fonctions de coût considèrent uniquement le facteur d'accès au disque.

Le coût d'une requête Q_i suivant son plan physique est donné par la somme des coûts de chaque opération (op) exécutée comme la selection, la jointure:

$$Cout_{Q_i} = \sum_{j=1}^{n} Cout_{op_j} \tag{A.6}$$

La métrique évaluée par les foncions données ci-dessous concerne la performance d'exécution des opérations d'une requête. Pour estimer le coût des opérations, certains paramètres du système de stockage et de la configuration matérielle sont nécessaires, nous les répertorions dans le tableau A.3.

Paramètres	**Description**
PS	Taille de la page
$T_{E/S}$	Temps de transfert du disque d'une page
$\|\|R\|\|$	Nombre de pages utilisé pour stocker la relation
$\|R\|$	nombre de tuples dans R
M	Le nombre de blocs dans la mémoire tampon (principale)
$L_t(R)$	Longueur d'un tuple de la relation R
$L_a(C_R)$	La longueur moyenne de la colonne C dans la relation R
$\|R.c\|$	Nombre de valeurs distinctes de la colonne C dans la relation R
f_{σ_R}	facteur de sélectivité pour la sélection (σ_R) sur la relation R
f_{π_R}	Facteur de sélectivité pour la projection (π_R) sur la relation R
$f_{R\bowtie S}$	Facteur de sélectivité pour la jointure($R \bowtie S$) sur les relations R et S

Tableau A.3 – Description des paramètres pour le calcul du coût des opérateurs.

Le coût $C_{E/S}$ de chaque opération (op_j) est donné soit par le nombre de page chargé soit par le nombre de seconde écoulé, comme suit:

$$C_{E/S}(op_j) = \begin{cases} P_{op_j} & \text{si le coût est représenté par le nombre de pages d' E/S} \\ T_{op_j} & \text{si le coût est représenté par le temps écoulé.} \end{cases} \tag{A.7}$$

Où P_{op_j} est le nombre de page lu ou écrit lors de l'exécution de l'opération op_j, et T_{op_j} est le temps écoulé lors de l'exécution de l'opération op_j. P_{op_j} et T_{op_j} sont définis ainsi:

$$P_{op_j} = P_{entre} + P_{interm} + P_{sortie} \tag{A.8}$$

$$T_{op_j} = T_{entre} + T_{interm} + T_{sortie} \tag{A.9}$$

Où P_{entre} et T_{entre} représentent le nombre de page et le temps qu'il faut respectivement pour lire une (des) relation(s) en entrée. P_{sortie} et T_{sortie} représentent respectivement le nombre de pages et le temps nécessaire pour transmettre la relation de sortie vers l'opérateur parent ou l'écrire sur la mémoire de stockage. P_{interm} et T_{interm} représentent le nombre de pages et le temps pour le stockage des résultats intermédiaires sur le disque.

Les opérateurs unaires nécessitent de lire une relation en entrée et les opérateurs binaires en demandent deux; le coût d'entrée d'un opérateur se formule comme suit:

$$T_{entre} = \begin{cases} P_{entre} \times T_{E/S} & \text{, si l'opérateur est unaire} \\ P_{entre_R} \times T_{E/S} + P_{entre_S} \times T_{E/S} & \text{, si un opérateur binaire sans pipelines.} \\ max(P_{entre_R} \times T_{E/S}, P_{entre} \times T_{E/S}) & \text{, si un opérateur binaire utilisant les pipelines.} \end{cases} \tag{A.10}$$

les coûts T_{interm} et T_{sortie} restent les mêmes pour les opérateurs unaires ou binaires:

$$T_{sortie} = P_{sortie} \times T_{E/S} \tag{A.11}$$

$$T_{interm} = P_{interm} \times T_{E/S} \tag{A.12}$$

Ci-dessous, nous donnons les formules universelles pour le calcul du coût des opérateurs.

A.2.2.1 Coût de la sélection

La sélection sur une relation R permet de réduire horizontalement la taille de la table par un facteur de sélectivité f_{σ_R} et verticalement par un facteur de filtrage $\phi(R, q)$ qui défini la fraction de la taille de q attributs sur la taille d'un tuple dans la relation R:

$$\phi(R, q) = \sum_{cinq} L_a(R.c)/L_t(R) \tag{A.13}$$

Le coût de sortie d'une opération de sélection est défini comme suit :

$$P_{sortie} = f_{\sigma_R} \times ||R|| \times \phi(R, q) \tag{A.14}$$

P_{interm} vaut 0 pour l'opérateur de sélection (aucun résultat intermédiaire). Pour le coût P_{entre}, tous les tuples de R sont balayés séquentiellement, sauf si R est trié selon les attributs de restriction, puis uniquement les tuples satisfaisant la condition de sélection sont retenus. Ainsi, le coût d'entrée peut être calculé comme suit:

$$P_{entre} = \begin{cases} ||R|| & \text{si R n'est pas trié} \\ f_{\sigma_R} \times ||R|| & \text{si R est trié.} \end{cases} \tag{A.15}$$

A.2.2.2 Coût de projection

Lors d'une projection, il n'y a pas d'attributs redondants à supprimer, c'est-à-dire que l'entrée est exactement les attributs à transmettre à l'opérateur suivant.

$$P_{sortie} = f_{\pi_R} \times ||R|| \tag{A.16}$$

Si la projection doit être triée et que la mémoire disponible pour le tri est insuffisante alors les résultats intermédiaires sont stockés sur disque, par conséquent le coût P_{interm} est comme suit:

$$P_{interm} = \begin{cases} 0 & \text{, si } ||R|| \leq (M-1) \text{ ,aucun tri} \\ ||R|| \times \log_{(M-1)}(||R||) & \text{, sinon.} \end{cases} \tag{A.17}$$

A.2.2.3 Coût de jointure

L'opérateur de jointure classique appliqué sur deux relations R et S produit une relation de sortie $R \bowtie S$, dont le coût de sortie peut être évalué comme suite:

$$P_{sortie} = f_{R \bowtie S} \times PS \times \frac{L_t(R \bowtie S)}{L_t(R) \times L_t(S)} \tag{A.18}$$

Les coûts P_{interm} et P_{entre} dépendent de l'implémentation algorithmique de la jointure comme suit:

— **Jointure par boucle imbriquée:** Dans cette implémentation, la relation externe doit être la plus petite afin de réduire le nombre d'itérations [163]. Cependant, si une des relations est déjà chargée dans la mémoire principale, elle est utilisée comme relation interne pour éviter les accès répétés au disque. Soit R et S les relations invoquées dans une opération

de jointure, où R est la relation extérieure, donc $||S|| \leq M \leq ||R||$, le coût P_{entre} est donné ainsi:

$$P_{entre} = \begin{cases} ||R||| + ||S|| & \text{, si sans pipeline} \\ max(||R||, ||S||) & \text{, sinon.} \end{cases} \tag{A.19}$$

Si la relation interne (R) est chargé en mémoire, $P_{intern} = 0$; sinon R doit être stockée sur le disque et être lue à plusieurs reprises pour chaque tuple de la relation (S). Ainsi, le coût P_{intern} peut être défini comme:

$$P_{intern} = \begin{cases} 0 & \text{, si} ||R|| \leq (M - ||S||) \\ |S| \times ||R|| & \text{, sinon.} \end{cases} \tag{A.20}$$

— **Jointure par tri fusion:**: Cette implémentation algorithmique est utilisée uniquement lorsque les deux relations sont triées sur l'attribut de jointure. Les relations sont lues en parallèle. Ainsi P_{interm} vaut nul et P_{entre} est égale au coût de lecture de la longue relation comme suit:

$$P_{entre} = max(||R||, ||S||) \tag{A.21}$$

— **Jointure par hachage:** Cette implémentation cumule deux étapes: une étape de partitionnement des deux relations en p fragments, avec la même fonction de hachage, et une étape de jointure. Nous supposons que la table de hachage est construite pour S et R est la relation extérieure. Le coût de création de la table de hachage h_{table} est le coût de chargement de S et de l'écriture de la table de hachage sur le disque. Ensuite h_{table} et R sont joints suivant une jointure par boucle imbriquée de sorte que h_{table} est la relation interne. Si l'on suppose que f_{hash} est le pourcentage de tuples de S qui dépend du facteur de sélectivité de jointure et de la fonction de hachage, alors le coût d'entrée P_{entre} et intermédiaire P_{interm} sont définis comme suit:

$$||h_{table}|| = f_{hash} \times ||S|| \tag{A.22}$$

$$P_{entre} = \begin{cases} ||R|| + ||S|| + ||h_{table}|| & \text{, si aucun pipeline} \\ max(||R||, ||S||) + ||h_{table}|| & \text{, sinon.} \end{cases} \tag{A.23}$$

$$P_{interm} = \begin{cases} 0 & \text{, si} ||h_{table}|| \leq (M - ||R||) \\ |R| \times ||h_{table}|| & \text{, sinon.} \end{cases} \tag{A.24}$$

A.2.2.4 Coût de tri

Différentes implémentations existent pour le tri et généralement le choix de leur utilisation dépend de la mémoire allouée par le systeme. Si la relation peut entièrement résider en mémoire alors, des implémentations de tri standard telles que le tri rapide peuvent être utilisées. Sinon le tri externe est utilisé. L'idée principale de cet algorithme est de charger des portions de la relation dans la mémoire principale, les trier, puis sauvegarder leur résultat sur le disque. Cela produit des blocs de fichiers triés, qui doivent ensuite être fusionnés afin de créer un unique bloc.

$$P_{entre} = 2 \times ||R|| \tag{A.25}$$

Le nombre d'exécution initial est $\lfloor ||R||/M \rfloor$. Puisque le nombre d'exécutions diminue d'un facteur $(M-1)$ dans chaque passe de fusion, le nombre total de passes de fusion requis est donc [163]:

$$P_{interm} = \left\lfloor \log_{(M-1)}(||R||/M) \right\rfloor \tag{A.26}$$

Chacune de ces passes lit chaque bloc de R une fois et l'écrit une fois, donc le P_{sortie} est:

$$P_{sortie} = \left\lfloor \log_{(M-1)}(||R||/M) \right\rfloor \tag{A.27}$$

A.2.2.5 Coût des opérations ensemblistes

L'implémentation des opérations de l'union, l'intersection et de la différence se font en triant les deux relations, puis en balayant une fois chacune des relations triées pour produire le résultat. Pour $R \cup S$, on ne retient qu'un tuple des tuples dupliqués. Le résultat de $R \cap S$ produits des tuples qui apparaissent dans les deux relations. La différence de $R - S$ produit des tuples de R seulement s'ils sont absents de S.

Le coût de chacune de ces opérations peut se formuler comme $||R|| + ||S||$ définissant le nombre de pages lues si les relations sont triées dans le même ordre, sinon le coût du tri doit être inclus.

A.2.2.6 Coût d'agrégation

L'implémentation des algorithmes d'agrégation utilise soit le tri ou une combinaison d'une fonction de hachage et le tri. L'approche est donc similaire à celle de la jointure par hachage et l'opérateur tri.

ANNEXE B

Benchmarks et les requêtes d'apprentissage

B.1 Introduction

Pour évaluer les approches que nous avons proposées, nous avons utilisé sur une configuration matérielle réelle différents benchmarks avec leur charge de requêtes. Les benchmarks utilisés dans notre travail sont:

- **Star Schema Benchmark (SSB) ; Source = `https://github.com/Kyligence/ssb-kylin`**
- **Transaction Processing Council Ad-hoc/decision support benchmark (TPC-H)**; Source=`http://www.tpc.org/tpch/`
- **Transaction Processing Council for Decision Support benchmark (TPC-DS)** ; Source=`http://www.tpc.org/tpcds/`

Pour construire nos différents modèles énergétiques, nous avons crée une charge de requête composée de soixante-dix (70) requêtes puis nous avons étudié leur caractéristique sur un jeu de donnée de taille variable généré à partir du benchmark TPC-H. Les études de validation de nos modèles ont été réalisées en utilisant les charges de requêtes générées à l'aide de l'outil « qgen »; soient 22 requêtes pour le benchmark TPC-H, 13 pour SSB, et 22 parmi les 99 requêtes du TPC-DS. Nos études d'estimation et d'optimisation énergétique ont été validé en utilisant les mêmes requêtes.

Dans cette annexe, nous présentons les différentes requêtes que nous avons utilisées lors de l'apprentissage du développement de notre modèle de coût énergétique.

B.2 Requêtes d'apprentissage

```
1   -- Query 1
2   SELECT sum(l_quantity), avg(l_quantity),
        sum(l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        avg(l_discount * (l_tax +1)),
        avg(l_quantity / (l_tax +1)),
        sum(l_tax)/sum(l_extendedprice),
        sum(l_quantity * 0.5), avg(l_quantity *
        0.2), sum(o_totalprice),
        avg(o_shippriority), sum(o_totalprice -
        l_extendedprice), avg( o_shippriority *
        l_extendedprice - 1),
        sum(o_shippriority)/sum( o_totalprice),
        count(*)
3   FROM lineitem, orders
4   WHERE l_suppkey <= 100000 and l_suppkey >=
        5000 and o_orderstatus = 'f' and
        l_orderkey = o_orderkey;
5
6   -- Query 2
7   SELECT l_shipmode, l_shipinstruct,
        o_orderstatus
8   FROM lineitem, orders
9   WHERE l_suppkey <= 100000 and l_suppkey >=
        5000 and o_orderstatus = 'f' and
        l_orderkey = o_orderkey
10  GROUP BY l_shipmode, l_shipinstruct,
        o_orderstatus;
11
12  -- Query 3
13  SELECT count(*)
14  FROM lineitem, orders
15  WHERE l_suppkey <= 100000 and l_suppkey >=
        5000 and l_orderkey = o_orderkey;
16
```

```
-- Query 4
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus
FROM lineitem, orders
WHERE l_suppkey <= 1000 and l_orderkey =
    o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus;

-- Query 5
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, count(*)
FROM lineitem, orders
WHERE l_suppkey <= 10000 and l_suppkey >=
    500 and o_orderstatus = 'f' and
    l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus;

-- Query 6
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, sum(l_quantity),
    avg(l_quantity), sum(o_totalprice),
    count(*)
FROM lineitem, orders
WHERE l_suppkey <= 1000 and l_suppkey >= 500
    and l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus;

-- Query 7
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, sum(l_quantity), avg(
    l_quantity), sum(o_totalprice), count(*)
FROM lineitem, orders
WHERE l_suppkey <= 100000 and l_suppkey >=
    5000 and o_orderstatus = 'f' and
    l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus;

-- Query 8
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority,
    sum(l_quantity), avg(l_quantity),
    sum(l_quantity - l_extendedprice),
    sum(l_quantity * l_extendedprice - 1),
    sum(o_totalprice), avg(o_shippriority),
    count(*)
FROM lineitem, orders
WHERE l_suppkey <= 1000 and l_suppkey >= 500
    and l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority;

-- Query 9
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority,
    sum(l_quantity), avg(l_quantity),
    sum(l_quantity - l_extendedprice),
    sum(l_quantity * l_extendedprice - 1),
    sum(o_totalprice), avg(o_shippriority),
    count(*)
FROM lineitem, orders
WHERE l_suppkey <= 100000 and l_suppkey >=
    5000 and o_orderstatus = 'p' and
    l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority;

-- Query 10
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority, sum(
    l_quantity), avg(l_quantity),
    sum(l_quantity - l_extendedprice), sum(
    l_quantity * l_extendedprice - 1),
    avg(l_discount * (l_tax +1)), avg(
    l_quantity / (l_tax +1)),
    sum(o_totalprice), avg(o_shippriority),
    sum(o_totalprice - l_extendedprice),
    avg(o_shippriority *l_extendedprice -
    1), count(*)
FROM lineitem, orders
WHERE l_suppkey <= 1000 and l_suppkey >= 500
    and l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority;

-- Query 11
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority, sum(
    l_quantity), avg(l_quantity),
    sum(l_quantity - l_extendedprice),
    sum(l_quantity * l_extendedprice - 1),
    avg(l_discount * (l_tax +1)),
    avg(l_quantity / (l_tax +1)),
    sum(o_totalprice), avg(o_shippriority),
    sum(o_totalprice - l_extendedprice),
    avg(o_shippriority *l_extendedprice -
    1), count(*)
FROM lineitem, orders
WHERE l_suppkey <= 100000 and l_suppkey >=
    5000 and o_orderstatus = 'p' and
    l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority;

-- Query 12
SELECT l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority,
    sum(l_quantity), avg(l_quantity),
    sum(l_quantity - l_extendedprice),
    sum(l_quantity * l_extendedprice - 1),
    avg(l_discount * (l_tax +1)),
    avg(l_quantity / (l_tax +1)),
    sum(l_tax)/sum(l_extendedprice),
    sum(l_quantity * 0.5), avg(l_quantity *
    0.2), sum(o_totalprice),
    avg(o_shippriority), sum(o_totalprice -
    l_extendedprice), avg(o_shippriority *
    l_extendedprice - 1),
    sum(o_shippriority)/sum(o_totalprice),
    count(*)
FROM lineitem, orders
WHERE l_suppkey <= 1000 and l_suppkey >= 500
    and l_orderkey = o_orderkey
GROUP BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority
ORDER BY l_shipmode, l_shipinstruct,
    o_orderstatus, o_orderpriority;

-- Query 13
```

```
80      SELECT count(*)
81      FROM lineitem, orders
82      WHERE l_suppkey <= 1000 and l_orderkey =
            o_orderkey;
83
84      -- Query 14
85      SELECT l_shipmode, l_shipinstruct,
            o_orderstatus, o_orderpriority, sum(
            l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice), sum(
            l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)), avg(
            l_quantity / (l_tax +1)),
            sum(l_tax)/sum(l_extendedprice),
            sum(l_quantity * 0.5), avg(l_quantity *
            0.2), sum(o_totalprice),
            avg(o_shippriority), sum(o_totalprice -
            l_extendedprice), avg(o_shippriority *
            l_extendedprice - 1),
            sum(o_shippriority)/sum(o_totalprice),
            count(*)
86      FROM lineitem, orders
87      WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 and o_orderstatus = 'p' and
            l_orderkey = o_orderkey
88      GROUP BY l_shipmode, l_shipinstruct,
            o_orderstatus, o_orderpriority
89      ORDER BY l_shipmode, l_shipinstruct,
            o_orderstatus, o_orderpriority;
90
91      -- Query 15
92      SELECT sum(o_totalprice)
93      FROM lineitem, orders
94      WHERE l_suppkey <= 10000 and l_suppkey >=
            500 and l_orderkey = o_orderkey;
95
96      -- Query 16
97      SELECT sum(l_quantity), avg(l_quantity),
            sum(o_totalprice), count(*)
98      FROM lineitem, orders
99      WHERE l_suppkey <= 1000 and l_suppkey >= 500
            and l_orderkey = o_orderkey;
100
101     -- Query 17
102     SELECT sum(l_quantity), avg(l_quantity),
            sum(o_totalprice), count(*)
103     FROM lineitem, orders
104     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 and o_orderstatus = 'f' and
            l_orderkey = o_orderkey;
105
106     -- Query 18
107     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            sum(o_totalprice), avg(
            o_shippriority), count(*)
108     FROM lineitem, orders
109     WHERE l_suppkey <= 1000 and l_suppkey >= 500
            and l_orderkey = o_orderkey;
110
111     -- Query 19
112     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            sum(o_totalprice), avg(
            o_shippriority), count(*)
113     FROM lineitem, orders
114     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 and o_orderstatus = 'f' and
            l_orderkey = o_orderkey;
115
116     -- Query 20
117     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)),
            sum(o_totalprice), avg(o_shippriority),
            sum(o_totalprice - l_extendedprice),
            avg(o_shippriority * l_extendedprice -
            1), count(*)
118     FROM lineitem, orders
119     WHERE l_suppkey <= 1000 and l_suppkey >= 500
            and l_orderkey = o_orderkey;
120
121     -- Query 21
122     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)),
            sum(o_totalprice), avg(o_shippriority),
            sum(o_totalprice - l_extendedprice),
            avg(o_shippriority * l_extendedprice -
            1), count(*)
123     FROM lineitem, orders
124     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 and o_orderstatus = 'f' and
            l_orderkey = o_orderkey;
125
126     -- Query 22
127     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)),
            sum(l_tax)/sum(l_extendedprice), sum(
            l_quantity * 0.5), avg(l_quantity *
            0.2), sum(o_totalprice), avg(
            o_shippriority), sum(o_totalprice -
            l_extendedprice), avg( o_shippriority *
            l_extendedprice - 1),
            sum(o_shippriority)/sum( o_totalprice),
            count(*)
128     FROM lineitem, orders
129     WHERE l_suppkey <= 1000 and l_suppkey >= 500
            and l_orderkey = o_orderkey;
130
131     -- Query 23
132     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)),
            sum(l_tax)/sum(l_extendedprice), sum(
            l_quantity * 0.5), avg(l_quantity *
            0.2), count(*)
133     FROM lineitem
134     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 ;
135
136     -- Query 24
137     SELECT l_shipmode, l_shipinstruct
138     FROM lineitem
139     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000
140     GROUP BY l_shipmode, l_shipinstruct;
141
142     -- Query 25
143     SELECT count(*)
144     FROM lineitem
145     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 ;
146
147     -- Query 26
```

```
    SELECT l_shipmode, l_shipinstruct
    FROM lineitem
    WHERE l_suppkey <= 1000
    GROUP BY l_shipmode, l_shipinstruct;

    -- Query 27
    SELECT l_shipmode, l_shipinstruct, count(*)
    FROM lineitem
    WHERE l_suppkey <= 10000 and l_suppkey >= 500
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 28
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity),
        count(*)
    FROM lineitem
    WHERE l_suppkey <= 1000 and l_suppkey >= 500
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 29
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity),
        count(*)
    FROM lineitem
    WHERE l_suppkey <= 100000 and l_suppkey >=
        5000
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 30
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity), sum(
        l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice -
        1),count(*)
    FROM lineitem
    WHERE l_suppkey <= 1000 and l_suppkey >= 500
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 31
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity), sum(
        l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        count(*)
    FROM lineitem
    WHERE l_suppkey <= 100000 and l_suppkey >=
        5000
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 32
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity), sum(
        l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        avg(l_discount * (l_tax +1)),
        avg(l_quantity / (l_tax +1)), count(*)
    FROM lineitem
    WHERE l_suppkey <= 1000 and l_suppkey >= 500
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 33
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity), sum(
        l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        avg(l_discount * (l_tax +1)),
        avg(l_quantity / (l_tax +1)), count(*)
    FROM lineitem
    WHERE l_suppkey <= 100000 and l_suppkey >=
        5000
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 34
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity),
        sum(l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        avg(l_discount * (l_tax +1)),
        avg(l_quantity / (l_tax +1)),
        sum(l_tax)/sum(l_extendedprice),
        sum(l_quantity * 0.5), avg(l_quantity *
        0.2), count(*)
    FROM lineitem
    WHERE l_suppkey <= 1000 and l_suppkey >= 500
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 35
    SELECT count(*)
    FROM lineitem
    WHERE l_suppkey <= 1000 ;

    -- Query 36
    SELECT l_shipmode, l_shipinstruct,
        sum(l_quantity), avg(l_quantity), sum(
        l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        avg(l_discount * (l_tax +1)),
        avg(l_quantity / (l_tax +1)), sum(
        l_tax)/sum(l_extendedprice),
        sum(l_quantity * 0.5), avg(l_quantity *
        0.2), count(*)
    FROM lineitem
    WHERE l_suppkey <= 100000 and l_suppkey >=
        5000
    GROUP BY l_shipmode, l_shipinstruct
    ORDER BY l_shipmode, l_shipinstruct;

    -- Query 37
    SELECT count(*) FROM lineitem WHERE
        l_suppkey <= 10000 and l_suppkey >= 500
        ;
    SELECT sum(l_quantity), avg(l_quantity),
        count(*)
    FROM lineitem
    WHERE l_suppkey <= 1000 and l_suppkey >= 500
        ;

    -- Query 38
    SELECT sum(l_quantity), avg(l_quantity),
        count(*)
    FROM lineitem
    WHERE l_suppkey <= 100000 and l_suppkey >=
        5000 ;

    -- Query 39
    SELECT sum(l_quantity), avg(l_quantity),
        sum(l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        count(*)
    FROM lineitem
    WHERE l_suppkey <= 1000 and l_suppkey >= 500
        ;

    -- Query 40
    SELECT sum(l_quantity), avg(l_quantity),
        sum(l_quantity - l_extendedprice),
        sum(l_quantity * l_extendedprice - 1),
        count(*)
```

```
239     FROM lineitem
240     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 ;
241
242     -- Query 41
243     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)), count(*)
244     FROM lineitem
245     WHERE l_suppkey <= 1000 and l_suppkey >= 500
            ;
246
247     -- Query 42
248     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)), count(*)
249     FROM lineitem
250     WHERE l_suppkey <= 100000 and l_suppkey >=
            5000 ;
251
252     -- Query 43
253     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)),
            sum(l_tax)/sum(l_extendedprice), sum(
            l_quantity * 0.5), avg(l_quantity *
            0.2), count(*)
254     FROM lineitem
255     WHERE l_suppkey <= 1000 and l_suppkey >= 500
            ;
256
257     -- Query 44
258     SELECT sum(l_extendedprice*l_discount)
259     FROM lineitem
260     WHERE l_shipdate> '1993-08-18' and
            l_shipdate<'1996-08-26' and l_discount
            between 0.04 and 0.07 and l_quantity <
            21;
261
262     -- Query 45
263     SELECT sum(l_extendedprice*l_discount)
264     FROM lineitem
265     WHERE l_shipdate> '1995-08-18' and
            l_shipdate<'1996-09-16' and l_discount
            between 0.02 and 0.06 and l_quantity <
            50 ;
266
267     -- Query 46
268     SELECT sum(l_extendedprice*l_discount)
269     FROM lineitem
270     WHERE l_shipdate>'1994-08-18' and
            l_shipdate<'1997-08-18' and l_discount
            between 0.01 and 0.03 and l_quantity <
            25;
271
272     -- Query 47
273     SELECT sum(l_extendedprice*l_discount)
274     FROM lineitem
275     WHERE l_shipdate>'1992-08-18' and
            l_shipdate<'1997-08-18' and l_discount
            between 0.03 and 0.04 and l_quantity <
            35;
276
277     -- Query 48
278     SELECT sum(l_extendedprice*l_discount)
279     FROM lineitem
280     WHERE l_shipdate> '1994-09-18' and
            l_shipdate<'1995-08-19' and l_discount
            between 0.05 and 0.09 and l_quantity <
            19;
281
282     -- Query 49
283     SELECT sum(l_extendedprice*l_discount)
284     FROM lineitem
285     WHERE l_shipdate> '1991-08-18' and
            l_shipdate<'1993-08-18' and l_discount
            between 0.01 and 0.07 and l_quantity <
            29;
286
287     -- Query 50
288     SELECT sum(l_extendedprice*l_discount),
            avg(l_extendedprice*l_discount),
            count(*)
289     FROM lineitem, orders
290     WHERE l_orderkey = o_orderkey and
            l_shipdate> '1991-08-18' and
            l_shipdate<'1993-08-18' and l_discount
            between 0.01 and 0.07 and l_quantity <
            29 and l_linestatus='F';
291
292     -- Query 51
293     SELECT sum(l_extendedprice*l_discount),
            avg(l_extendedprice*l_discount),
            count(*)
294     FROM lineitem, orders
295     WHERE l_orderkey = o_orderkey and
            l_shipdate> '1991-08-18' and
            l_shipdate<'1993-08-18' and l_discount
            between 0.01 and 0.07 and l_quantity <
            29 and l_linestatus='O';
296
297     -- Query 52
298     SELECT
            sum(l_extendedprice*l_discount)/avg(l_extendedprice*l_disc
299     FROM lineitem, orders
300     WHERE l_orderkey = o_orderkey and
            l_shipdate>'1991-08-18' and
            l_shipdate<'1995-08-18' and l_discount
            between 0.05 and 0.07 and l_quantity <
            39 and l_linestatus='F';
301
302     -- Query 53
303     SELECT sum(l_quantity), avg(l_quantity),
            sum(l_quantity - l_extendedprice),
            sum(l_quantity * l_extendedprice - 1)
304     FROM lineitem, orders
305     WHERE l_orderkey = o_orderkey and
            l_shipdate> '1992-08-18' and
            l_shipdate<'1993-08-18' and l_discount
            between 0.01 and 0.07 and l_quantity <
            40 and l_linestatus='F';
306
307     -- Query 54
308     SELECT sum(l_extendedprice*l_discount),
            avg(l_discount * (l_tax +1)),
            avg(l_quantity / (l_tax +1)),
            sum(o_totalprice)
309     FROM lineitem, orders
310     WHERE l_orderkey = o_orderkey and
            l_shipdate> '1991-08-18' and
            l_shipdate<'1993-08-18' and l_discount
            between 0.04 and 0.07 and l_quantity <
            19 and l_linestatus='O';
311
312     -- Query 55
313     SELECT sum(l_extendedprice*l_discount)
314     FROM lineitem, orders
315     WHERE l_orderkey = o_orderkey and
            l_shipdate>'1991-08-18' and
            l_shipdate<'1993-08-18'
```

```
316     GROUP BY l_shipmode, l_shipinstruct;
317
318     -- Query 56
319     SELECT sum(l_extendedprice*l_discount)
320     FROM lineitem, orders WHERE l_orderkey =
            o_orderkey and l_shipdate> '1991-08-18'
            and l_shipdate<'1995-08-18'
321     GROUP BY l_shipmode;
322
323     -- Query 57
324     SELECT sum(l_extendedprice*l_discount),
            avg(o_shippriority), sum(o_totalprice -
            l_extendedprice)
325     FROM lineitem, orders
326     WHERE l_orderkey = o_orderkey and
            l_shipdate> '1991-08-18' and
            l_shipdate<'1995-08-18' and l_quantity
            < 19 and l_linestatus='O'
327     GROUP BY l_shipinstruct;
328
329     -- Query 58
330     SELECT sum(l_extendedprice*l_discount) ,
            avg(o_shippriority * l_extendedprice -
            1), count(*) FROM lineitem, orders
331     WHERE l_orderkey = o_orderkey and
            l_shipdate> '1991-08-18' and
            l_shipdate<'1995-08-18' and l_quantity
            < 19 and l_linestatus='O' and
            l_returnflag='N';
332
333     -- Query 59
334     SELECT sum(l_extendedprice*l_discount) ,
            avg(o_shippriority * l_extendedprice -
            1), count(*) FROM lineitem, orders
335     WHERE l_orderkey = o_orderkey and
            l_shipdate> '1991-08-18' and
            l_shipdate<'1995-08-18' and l_quantity
            < 19 and l_linestatus='O' and
            l_returnflag='N' and l_shipmode='TRUCK';
336
337     -- Query 60
338     SELECT l_shipmode, sum(l_quantity)
339     FROM orders, lineitem
340     WHERE l_quantity BETWEEN 62 AND 62+130 AND
            l_orderkey = o_orderkey AND l_shipdate
            BETWEEN cast('1990-05-25' AS date) AND
            cast('2000-07-24' AS date) AND
            l_shipmode IN ('MAIL', 'SHIP') AND
            o_orderdate < date '1990-01-01' +
            interval '1' year
341     GROUP BY    l_shipmode
342     ORDER BY    l_shipmode LIMIT 100;
343
344     -- Query 61
345     SELECT count(*) as line,
            sum(l_quantity*l_extendedprice),
            sum(l_tax)
346     FROM lineitem
347     GROUP BY    l_shipmode
348     ORDER BY    l_shipmode LIMIT 100;
349
350     -- Query 62
351     SELECT sum(ps_availqty*ps_supplycost)
352     FROM partsupp, supplier
353     WHERE ps_suppkey=s_suppkey
354     GROUP BY s_suppkey LIMIT 100;
355
356     -- Query 63
357     SELECT sum(l_extendedprice*l_quantity) as
            revenue, o_orderdate, o_orderpriority,
            count(*) as line
358     FROM lineitem, orders
359     WHERE l_orderkey = o_orderkey AND l_shipmode
            IN ('MAIL', 'SHIP')
360     GROUP BY o_orderdate, o_orderpriority
361     ORDER BY o_orderdate, o_orderpriority DESC
            LIMIT 300;
362
363     -- Query 64
364     SELECT sum(l_extendedprice*l_quantity) as
            revenue, l_returnflag, l_shipmode,
            count(*) as line FROM lineitem, orders
365     WHERE l_orderkey = o_orderkey AND
            o_orderdate > date '1990-01-01' +
            interval '1' year
366     GROUP BY l_returnflag, l_shipmode
367     ORDER BY l_returnflag, l_shipmode DESC LIMIT
            300;
368
369     -- Query 65
370     SELECT sum(o_totalprice) as Total, count(*)
            as Nombre, c_name
371     FROM orders, customer
372     WHERE c_custkey = o_custkey
373     GROUP BY c_custkey, c_name
374     ORDER BY sum(o_totalprice) DESC LIMIT 300;
375
376     -- Query 66
377     WITH cte AS ( SELECT sum(o_totalprice) as
            Total, count(*) as Nombre, n_name,
            row_number() over (ORDER BY
            sum(o_totalprice) ) as num
378     FROM orders, customer, nation
379     WHERE c_custkey = o_custkey and
            c_nationkey=n_nationkey
380     GROUP BY n_nationkey, n_name
381     ORDER BY sum(o_totalprice) ) SELECT * FROM
            cte WHERE num<=10;
382
383     -- Query 67
384     SELECT sum(l_extendedprice*l_quantity) as
            Total, count(*) as Nombre, p_brand
385     FROM lineitem, part
386     WHERE l_partkey = p_partkey
387     GROUP BY p_brand, l_shipdate
388     ORDER BY p_brand, l_shipdate,
            sum(l_extendedprice*l_quantity) LIMIT
            100;
389
390     -- Query 68
391     SELECT sum(l_extendedprice*l_quantity) as
            Total, count(*) as Nombre, p_brand,
            c_name
392     FROM lineitem, part, orders, customer
393     WHERE l_partkey = p_partkey and
            l_orderkey=o_orderkey and
            o_custkey=c_custkey and o_orderdate >
            date '1990-01-01' + interval '1' year
394     GROUP BY c_custkey,c_name, p_brand
395     ORDER BY c_custkey,c_name, p_brand limit 500;
396
397     -- Query 69
398     WITH c_item AS (SELECT c_custkey, c_name,
            c_nationkey
399     FROM customer
400     WHERE c_nationkey IN   (
401     SELECT n_nationkey
402     FROM nation, region
403     WHERE n_regionkey= r_regionkey  and r_name
            IN ('AFRICA','AMERICA', 'EUROPE')
404     )    )
405     SELECT sum(l_extendedprice*l_quantity),
            sum(l_tax), avg(l_tax), c_name
406     FROM c_item , orders, lineitem
407     WHERE l_receiptdate > date '1990-01-01' +
            interval '1' year and
```

```
        l_orderkey=o_orderkey and
        o_custkey=c_custkey
    GROUP BY l_shipdate, l_shipmode, c_custkey,
        c_name
    ORDER BY l_shipdate, l_shipmode, c_custkey,
        c_name, sum(l_extendedprice*l_quantity)
        desc limit 500;

    -- Query 70
    SELECT sum(l_extendedprice*l_quantity) as
        revenue, l_returnflag, l_shipmode,
        count(*) as line FROM lineitem, orders
    WHERE l_orderkey = o_orderkey AND
        o_orderdate > date '1990-01-01' +
        interval '4' year
    GROUP BY l_returnflag, l_shipmode
    ORDER BY l_returnflag, l_shipmode DESC LIMIT
        200;
```

B.3 Requêtes TPC-DS utilisées

```
    -- 1 Query 3
    SELECT dt.d_year,
    item.i_brand_id        brand_id,
    item.i_brand           brand,
    Sum(ss_ext_discount_amt) sum_agg
    FROM   date_dim dt,
    store_sales,
    item
    WHERE dt.d_date_sk =
        store_sales.ss_sold_date_sk
    AND store_sales.ss_item_sk = item.i_item_sk
    AND item.i_manufact_id = 427
    AND dt.d_moy = 11
    GROUP BY dt.d_year,
    item.i_brand,
    item.i_brand_id
    ORDER BY dt.d_year,
    sum_agg DESC,
    brand_id
    LIMIT 100;

    -- 2 Query 6
    SELECT a.ca_state state,
    Count(*)  cnt
    FROM   customer_address a,
    customer c,
    store_sales s,
    date_dim d,
    item i
    WHERE a.ca_address_sk = c.c_current_addr_sk
    AND c.c_customer_sk = s.ss_customer_sk
    AND s.ss_sold_date_sk = d.d_date_sk
    AND s.ss_item_sk = i.i_item_sk
    AND d.d_month_seq = (SELECT DISTINCT (
        d_month_seq )
    FROM   date_dim
    WHERE d_year = 1998
    AND d_moy = 7)
    AND i.i_current_price > 1.2 * (SELECT
        Avg(j.i_current_price)
    FROM   item j
    WHERE j.i_category = i.i_category)
    GROUP BY a.ca_state
    HAVING Count(*) >= 10
    ORDER BY cnt
    LIMIT 100;

    -- 3 Query 7
    SELECT i_item_id,
    Avg(ss_quantity)  agg1,
    Avg(ss_list_price) agg2,
    Avg(ss_coupon_amt) agg3,
    Avg(ss_sales_price) agg4
    FROM   store_sales,
    customer_demographics,
    date_dim,
    item,
    promotion
    WHERE ss_sold_date_sk = d_date_sk
    AND ss_item_sk = i_item_sk
    AND ss_cdemo_sk = cd_demo_sk
    AND ss_promo_sk = p_promo_sk
    AND cd_gender = 'F'
    AND cd_marital_status = 'W'
    AND cd_education_status = '2 yr Degree'
    AND ( p_channel_email = 'N'
    OR p_channel_event = 'N' )
    AND d_year = 1998
    GROUP BY i_item_id
    ORDER BY i_item_id
    LIMIT 100;

    -- 4 Query 10
    SELECT cd_gender,
    cd_marital_status,
    cd_education_status,
    Count(*) cnt1,
    cd_purchase_estimate,
    Count(*) cnt2,
    cd_credit_rating,
    Count(*) cnt3,
    cd_dep_count,
    Count(*) cnt4,
    cd_dep_employed_count,
    Count(*) cnt5,
    cd_dep_college_count,
    Count(*) cnt6
    FROM   customer c,
    customer_address ca,
    customer_demographics
    WHERE c.c_current_addr_sk = ca.ca_address_sk
    AND ca_county IN ( 'Lycoming County',
        'Sheridan County',
    'Kandiyohi County',
    'Pike County',
    'Greene County' )
    AND cd_demo_sk = c.c_current_cdemo_sk
    AND EXISTS (SELECT *
    FROM   store_sales,
    date_dim
    WHERE c.c_customer_sk = ss_customer_sk
    AND ss_sold_date_sk = d_date_sk
    AND d_year = 2002
    AND d_moy BETWEEN 4 AND 4 + 3)
    AND ( EXISTS (SELECT *
    FROM   web_sales,
    date_dim
    WHERE c.c_customer_sk = ws_bill_customer_sk
    AND ws_sold_date_sk = d_date_sk
    AND d_year = 2002
    AND d_moy BETWEEN 4 AND 4 + 3)
    OR EXISTS (SELECT *
```

```
109       FROM   catalog_sales,
110       date_dim
111       WHERE c.c_customer_sk = cs_ship_customer_sk
112       AND cs_sold_date_sk = d_date_sk
113       AND d_year = 2002
114       AND d_moy BETWEEN 4 AND 4 + 3) )
115       GROUP BY cd_gender,
116       cd_marital_status,
117       cd_education_status,
118       cd_purchase_estimate,
119       cd_credit_rating,
120       cd_dep_count,
121       cd_dep_employed_count,
122       cd_dep_college_count
123       ORDER BY cd_gender,
124       cd_marital_status,
125       cd_education_status,
126       cd_purchase_estimate,
127       cd_credit_rating,
128       cd_dep_count,
129       cd_dep_employed_count,
130       cd_dep_college_count
131       LIMIT 100;
132
133       -- 5 Query 12
134       SELECT
135       i_item_id ,
136       i_item_desc ,
137       i_category ,
138       i_class ,
139       i_current_price ,
140       Sum(ws_ext_sales_price)
              AS itemrevenue ,
141       Sum(ws_ext_sales_price)*100/Sum(Sum(ws_ext_sales_p
              OVER (partition BY i_class) AS
              revenueratio
142       FROM     web_sales ,
143       item ,
144       date_dim
145       WHERE    ws_item_sk = i_item_sk
146       AND      i_category IN ('Home',
147       'Men',
148       'Women')
149       AND      ws_sold_date_sk = d_date_sk
150       AND      d_date BETWEEN Cast('2000-05-11' AS
              DATE) AND (
151       Cast('2000-05-11' AS DATE) + INTERVAL '30'
              day)
152       GROUP BY i_item_id ,
153       i_item_desc ,
154       i_category ,
155       i_class ,
156       i_current_price
157       ORDER BY i_category ,
158       i_class ,
159       i_item_id ,
160       i_item_desc ,
161       revenueratio
162       LIMIT 100;
163
164       -- 6 Query 15
165       SELECT ca_zip,
166       Sum(cs_sales_price)
167       FROM   catalog_sales,
168       customer,
169       customer_address,
170       date_dim
171       WHERE cs_bill_customer_sk = c_customer_sk
172       AND c_current_addr_sk = ca_address_sk
173       AND ( Substr(ca_zip, 1, 5) IN ( '85669',
              '86197', '88274', '83405',
174       '86475', '85392', '85460', '80348',
175       '81792' )
176       OR ca_state IN ( 'CA', 'WA', 'GA' )
177       OR cs_sales_price > 500 )
178       AND cs_sold_date_sk = d_date_sk
179       AND d_qoy = 1
180       AND d_year = 1998
181       GROUP BY ca_zip
182       ORDER BY ca_zip
183       LIMIT 100;
184
185       -- 7 Query 16
186       SELECT
187       Count(DISTINCT cs_order_number) AS `order
              count` ,
188       Sum(cs_ext_ship_cost)        AS `total
              shipping cost` ,
189       Sum(cs_net_profit)           AS `total net
              profit`
190       FROM     catalog_sales cs1 ,
191       date_dim ,
192       customer_address ,
193       call_center
194       WHERE    d_date BETWEEN '2002-3-01' AND   (
195       Cast('2002-3-01' AS DATE) + INTERVAL '60'
              day)
196       AND      cs1.cs_ship_date_sk = d_date_sk
197       AND      cs1.cs_ship_addr_sk = ca_address_sk
198       AND      ca_state = 'IA'
199       AND      cs1.cs_call_center_sk =
              cc_call_center_sk
200       AND      cc_county IN ('Williamson County',
201       'Williamson County',
202       'Williamson County',
203       'Williamson County',
204       'Williamson County' )
205       AND      EXISTS
206       (
207       SELECT *
208       FROM   catalog_sales cs2
209       WHERE cs1.cs_order_number =
              cs2.cs_order_number
210       AND    cs1.cs_warehouse_sk <>
              cs2.cs_warehouse_sk)
211       AND      NOT EXISTS
212       (
213       SELECT *
214       FROM   catalog_returns cr1
215       WHERE cs1.cs_order_number =
              cr1.cr_order_number)
216       ORDER BY count(DISTINCT cs_order_number)
217       LIMIT 100;
218
219       -- 8 Query 19
220       SELECT i_brand_id           brand_id,
221       i_brand                brand,
222       i_manufact_id,
223       i_manufact,
224       Sum(ss_ext_sales_price) ext_price
225       FROM   date_dim,
226       store_sales,
227       item,
228       customer,
229       customer_address,
230       store
231       WHERE d_date_sk = ss_sold_date_sk
232       AND ss_item_sk = i_item_sk
233       AND i_manager_id = 38
234       AND d_moy = 12
235       AND d_year = 1998
236       AND ss_customer_sk = c_customer_sk
237       AND c_current_addr_sk = ca_address_sk
238       AND Substr(ca_zip, 1, 5) <> Substr(s_zip, 1,
              5)
```

```
AND ss_store_sk = s_store_sk
GROUP BY i_brand,
i_brand_id,
i_manufact_id,
i_manufact
ORDER BY ext_price DESC,
i_brand,
i_brand_id,
i_manufact_id,
i_manufact
LIMIT 100;

-- 9 Query 20
SELECT
i_item_id ,
i_item_desc ,
i_category ,
i_class ,
i_current_price ,
Sum(cs_ext_sales_price)

    AS itemrevenue ,
Sum(cs_ext_sales_price)*100/Sum(Sum(cs_ext_sales_p
    OVER (partition BY i_class) AS
    revenueratio
FROM    catalog_sales ,
item ,
date_dim
WHERE   cs_item_sk = i_item_sk
AND     i_category IN ('Children',
'Women',
'Electronics')
AND     cs_sold_date_sk = d_date_sk
AND     d_date BETWEEN Cast('2001-02-03' AS
    DATE) AND (
Cast('2001-02-03' AS DATE) + INTERVAL '30'
    day)
GROUP BY i_item_id ,
i_item_desc ,
i_category ,
i_class ,
i_current_price
ORDER BY i_category ,
i_class ,
i_item_id ,
i_item_desc ,
revenueratio
LIMIT 100;

-- 10 Query 21
SELECT
*
FROM   (
SELECT  w_warehouse_name ,
i_item_id ,
Sum(
CASE
WHEN (
Cast(d_date AS DATE) < Cast ('2000-05-13' AS
    DATE)) THEN inv_quantity_on_hand
ELSE 0
END) AS inv_before ,
Sum(
CASE
WHEN (
Cast(d_date AS DATE) >= Cast ('2000-05-13'
    AS DATE)) THEN inv_quantity_on_hand
ELSE 0
END) AS inv_after
FROM    inventory ,
warehouse ,
item ,
date_dim
WHERE   i_current_price BETWEEN 0.99 AND
    1.49
AND     i_item_sk = inv_item_sk
AND     inv_warehouse_sk = w_warehouse_sk
AND     inv_date_sk = d_date_sk
AND     d_date BETWEEN (Cast ('2000-05-13'
    AS DATE) - INTERVAL '30' day) AND (
cast ('2000-05-13' AS     date) + INTERVAL
    '30' day)
GROUP BY w_warehouse_name,
i_item_id) x
WHERE   (
CASE
WHEN inv_before > 0 THEN inv_after /
    inv_before
ELSE NULL
END) BETWEEN 2.0/3.0 AND   3.0/2.0
ORDER BY w_warehouse_name ,
i_item_id
LIMIT 100;

-- 11 Query 25
SELECT i_item_id,
i_item_desc,
s_store_id,
s_store_name,
Max(ss_net_profit) AS store_sales_profit,
Max(sr_net_loss) AS store_returns_loss,
Max(cs_net_profit) AS catalog_sales_profit
FROM  store_sales,
store_returns,
catalog_sales,
date_dim d1,
date_dim d2,
date_dim d3,
store,
item
WHERE d1.d_moy = 4
AND d1.d_year = 2001
AND d1.d_date_sk = ss_sold_date_sk
AND i_item_sk = ss_item_sk
AND s_store_sk = ss_store_sk
AND ss_customer_sk = sr_customer_sk
AND ss_item_sk = sr_item_sk
AND ss_ticket_number = sr_ticket_number
AND sr_returned_date_sk = d2.d_date_sk
AND d2.d_moy BETWEEN 4 AND 10
AND d2.d_year = 2001
AND sr_customer_sk = cs_bill_customer_sk
AND sr_item_sk = cs_item_sk
AND cs_sold_date_sk = d3.d_date_sk
AND d3.d_moy BETWEEN 4 AND 10
AND d3.d_year = 2001
GROUP BY i_item_id,
i_item_desc,
s_store_id,
s_store_name
ORDER BY i_item_id,
i_item_desc,
s_store_id,
s_store_name
LIMIT 100;

-- 12 Query 26
SELECT i_item_id,
Avg(cs_quantity)  agg1,
Avg(cs_list_price) agg2,
Avg(cs_coupon_amt) agg3,
Avg(cs_sales_price) agg4
FROM  catalog_sales,
customer_demographics,
date_dim,
item,
```

```
      promotion
      WHERE cs_sold_date_sk = d_date_sk
      AND cs_item_sk = i_item_sk
      AND cs_bill_cdemo_sk = cd_demo_sk
      AND cs_promo_sk = p_promo_sk
      AND cd_gender = 'F'
      AND cd_marital_status = 'W'
      AND cd_education_status = 'Secondary'
      AND ( p_channel_email = 'N'
      OR p_channel_event = 'N' )
      AND d_year = 2000
      GROUP BY i_item_id
      ORDER BY i_item_id
      LIMIT 100;

      -- 13 Query 35
      SELECT ca_state,
      cd_gender,
      cd_marital_status,
      cd_dep_count,
      Count(*) cnt1,
      Stddev_samp(cd_dep_count),
      Avg(cd_dep_count),
      Max(cd_dep_count),
      cd_dep_employed_count,
      Count(*) cnt2,
      Stddev_samp(cd_dep_employed_count),
      Avg(cd_dep_employed_count),
      Max(cd_dep_employed_count),
      cd_dep_college_count,
      Count(*) cnt3,
      Stddev_samp(cd_dep_college_count),
      Avg(cd_dep_college_count),
      Max(cd_dep_college_count)
      FROM  customer c,
      customer_address ca,
      customer_demographics
      WHERE c.c_current_addr_sk = ca.ca_address_sk
      AND cd_demo_sk = c.c_current_cdemo_sk
      AND EXISTS (SELECT *
      FROM  store_sales,
      date_dim
      WHERE c.c_customer_sk = ss_customer_sk
      AND ss_sold_date_sk = d_date_sk
      AND d_year = 2001
      AND d_qoy < 4)
      AND ( EXISTS (SELECT *
      FROM  web_sales,
      date_dim
      WHERE c.c_customer_sk = ws_bill_customer_sk
      AND ws_sold_date_sk = d_date_sk
      AND d_year = 2001
      AND d_qoy < 4)
      OR EXISTS (SELECT *
      FROM  catalog_sales,
      date_dim
      WHERE c.c_customer_sk = cs_ship_customer_sk
      AND cs_sold_date_sk = d_date_sk
      AND d_year = 2001
      AND d_qoy < 4) )
      GROUP BY ca_state,
      cd_gender,
      cd_marital_status,
      cd_dep_count,
      cd_dep_employed_count,
      cd_dep_college_count
      ORDER BY ca_state,
      cd_gender,
      cd_marital_status,
      cd_dep_count,
      cd_dep_employed_count,
      cd_dep_college_count
      LIMIT 100;

      -- 14 Query 37
      SELECT
      i_item_id ,
      i_item_desc ,
      i_current_price
      FROM    item,
      inventory,
      date_dim,
      catalog_sales
      WHERE   i_current_price BETWEEN 20 AND  20
          + 30
      AND     inv_item_sk = i_item_sk
      AND     d_date_sk=inv_date_sk
      AND     d_date BETWEEN Cast('1999-03-06' AS
          DATE) AND (
      Cast('1999-03-06' AS DATE) + INTERVAL '60'
          day)
      AND     i_manufact_id IN (843,815,850,840)
      AND     inv_quantity_on_hand BETWEEN 100 AND
           500
      AND     cs_item_sk = i_item_sk
      GROUP BY i_item_id,
      i_item_desc,
      i_current_price
      ORDER BY i_item_id
      LIMIT 100;

      -- 15 Query 40
      SELECT
      w_state ,
      i_item_id ,
      Sum(
      CASE
      WHEN (
      Cast(d_date AS DATE) < Cast ('2002-06-01' AS
          DATE)) THEN cs_sales_price -
          COALESCE(cr_refunded_cash,0)
      ELSE 0
      END) AS sales_before ,
      Sum(
      CASE
      WHEN (
      Cast(d_date AS DATE) >= Cast ('2002-06-01'
          AS DATE)) THEN cs_sales_price -
          COALESCE(cr_refunded_cash,0)
      ELSE 0
      END) AS sales_after
      FROM           catalog_sales
      LEFT OUTER JOIN catalog_returns
      ON             (
      cs_order_number = cr_order_number
      AND            cs_item_sk = cr_item_sk) ,
      warehouse ,
      item ,
      date_dim
      WHERE          i_current_price BETWEEN 0.99
          AND         1.49
      AND            i_item_sk = cs_item_sk
      AND            cs_warehouse_sk =
          w_warehouse_sk
      AND            cs_sold_date_sk = d_date_sk
      AND            d_date BETWEEN (Cast
          ('2002-06-01' AS DATE) - INTERVAL '30'
          day) AND     (
      cast ('2002-06-01' AS date) + INTERVAL '30'
          day)
      GROUP BY       w_state,
      i_item_id
      ORDER BY       w_state,
      i_item_id
      LIMIT 100;
```

```
-- 16 Query 42
SELECT dt.d_year,
item.i_category_id,
item.i_category,
Sum(ss_ext_sales_price)
FROM  date_dim dt,
store_sales,
item
WHERE dt.d_date_sk =
    store_sales.ss_sold_date_sk
AND store_sales.ss_item_sk = item.i_item_sk
AND item.i_manager_id = 1
AND dt.d_moy = 12
AND dt.d_year = 2000
GROUP BY dt.d_year,
item.i_category_id,
item.i_category
ORDER BY Sum(ss_ext_sales_price) DESC,
dt.d_year,
item.i_category_id,
item.i_category
LIMIT 100;

-- 17 Query 52
SELECT dt.d_year,
item.i_brand_id      brand_id,
item.i_brand         brand,
Sum(ss_ext_sales_price) ext_price
FROM  date_dim dt,
store_sales,
item
WHERE dt.d_date_sk =
    store_sales.ss_sold_date_sk
AND store_sales.ss_item_sk = item.i_item_sk
AND item.i_manager_id = 1
AND dt.d_moy = 11
AND dt.d_year = 1999
GROUP BY dt.d_year,
item.i_brand,
item.i_brand_id
ORDER BY dt.d_year,
ext_price DESC,
brand_id
LIMIT 100;

-- 18 Query 55
SELECT i_brand_id           brand_id,
i_brand              brand,
Sum(ss_ext_sales_price) ext_price
FROM  date_dim,
store_sales,
item
WHERE d_date_sk = ss_sold_date_sk
AND ss_item_sk = i_item_sk
AND i_manager_id = 33
AND d_moy = 12
AND d_year = 1998
GROUP BY i_brand,
i_brand_id
ORDER BY ext_price DESC,
i_brand_id
LIMIT 100;

-- 19 Query 93
SELECT ss_customer_sk,
Sum(act_sales) sumsales
FROM  (SELECT ss_item_sk,
ss_ticket_number,
ss_customer_sk,
CASE
WHEN sr_return_quantity IS NOT NULL THEN
( ss_quantity - sr_return_quantity ) *
    ss_sales_price
ELSE ( ss_quantity * ss_sales_price )
END act_sales
FROM  store_sales
LEFT OUTER JOIN store_returns
ON ( sr_item_sk = ss_item_sk
AND sr_ticket_number = ss_ticket_number ),
reason
WHERE sr_reason_sk = r_reason_sk
AND r_reason_desc = 'reason 38') t
GROUP BY ss_customer_sk
ORDER BY sumsales,
ss_customer_sk
LIMIT 100;

-- 20 Query 84
SELECT c_customer_id AS customer_id,
c_last_name
|| ', '
|| c_first_name AS customername
FROM  customer,
customer_address,
customer_demographics,
household_demographics,
income_band,
store_returns
WHERE ca_city = 'Green Acres'
AND c_current_addr_sk = ca_address_sk
AND ib_lower_bound >= 54986
AND ib_upper_bound <= 54986 + 50000
AND ib_income_band_sk = hd_income_band_sk
AND cd_demo_sk = c_current_cdemo_sk
AND hd_demo_sk = c_current_hdemo_sk
AND sr_cdemo_sk = cd_demo_sk
ORDER BY c_customer_id
LIMIT 100;

-- 21 Query 82
SELECT
i_item_id ,
i_item_desc ,
i_current_price
FROM    item,
inventory,
date_dim,
store_sales
WHERE   i_current_price BETWEEN 63 AND
    63+30
AND     inv_item_sk = i_item_sk
AND     d_date_sk=inv_date_sk
AND     d_date BETWEEN Cast('1998-04-27' AS
    DATE) AND (
Cast('1998-04-27' AS DATE) + INTERVAL '60'
    day)
AND     i_manufact_id IN (57,293,427,320)
AND     inv_quantity_on_hand BETWEEN 100 AND
     500
AND     ss_item_sk = i_item_sk
GROUP BY i_item_id,
i_item_desc,
i_current_price
ORDER BY i_item_id
LIMIT 100;
```

Liste des figures

1.1 répartition de la consommation énergétique dans un data center [157]. 16
1.2 Dioxyde d'azote(NO_2) mesurée par la NASA - [SOURCE=[137]] 17
1.3 Architecture d'un optimiseur de requête. 18
1.4 Architecture d'un optimiseur de requête (avec la dimension énergétique). 20
1.5 La répartition des chapitres de la thèse. 22

2.1 Architecture générique des STRs. 30
2.2 Principales étapes d'un STR. 31
2.3 STR d'une BD centralisée. 32
2.4 schéma de modélisation logique déduit du schéma conceptuel E/A 32
2.5 Graphe de l'analyse syntaxique de la requête Q_i. 33
2.6 Graphe des expressions algébriques de la requête Q_i 34
2.7 Exemples d'histogrammes de distribution. 37
2.8 Moteur d'optimisation des requêtes. 38
2.9 Différentes formes de parallélisme. 39
2.10 Exemple de parallélisme 39
2.11 Processus de traitement d'une requête dans un système distribué. 40
2.12 Processus de traitement d'une requête dans un système multi-bases. 41
2.13 Processus de traitement d'une requête dans un système multi-store. 42
2.14 Processus de traitement d'une requête dans un système HadoopDB. 43
2.15 Exemple d'architecture Multi-coeur. 44
2.16 Facteurs impactant la consommation énergétique dans les SSDs. 45

3.1 Estimation annuelle de la population mondiale et de la demande énergétique [128] 51
3.2 Approche générale pour construire et utiliser un modèle d'énergie 54
3.3 Aperçu de quelques benchmarks énergétiques disponible dans la littérature . . . 54
3.4 Interdépendance des différents niveaux ayant un impact sur la consommation énergétique des systèmes informatiques 55
3.5 Vue abstraite de l'efficacité énergétique des logiciels.PM *pour Power Management en français gestion de l'énergie* 58
3.6 Taxonomies des approches d'EE dans les systèmes informatiques. 61
3.7 Classification des approches d'EE orientées matérielles. 62
3.8 Vue d'ensemble de l'intégration de l'efficacité énergétique dans les BDs 64
3.9 Classification des approches d'EE orientées Logicielles. 68

4.1 Différents niveaux de modélisation. 74
4.2 Mode d'exécution des requêtes. 74
4.3 Les différents aspects lors de la phase de conception du modèle. 75
4.4 Les différentes étapes du processus de développement d'un modèle. 75
4.5 Architecture du système PostgreSQL. 78

4.6 Plan parallèle de la requête. 78
4.7 Algèbre relationnelle et instructions MAL générées. 80
4.8 Architecture de fonctionnement interne des systèmes. 81
4.9 Temps et énergie consommée lors du chargement des données. 83
4.10 Comparaison du temps d'exécution des requêtes $Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}$ du benchmark TPC-H en mode parallèle. 83
4.11 Consommation d'énergie moyenne des requêtes $Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_7, Q_8, Q_9, Q_{10}$ du benchmark TPC-H. 84
4.12 L3 et L2 défauts de cache comptés dans Q_1, Q_2, Q_3 du TPC-H. 84
4.13 Représentation du plan d'exécution en mode parallélisée 85
4.14 La segmentation du plan d'exécution en pipelines 85
4.15 Séquençage des pipelines et répartition de la consommation d'énergie au fur de l'exécution de la requête Q5 du TPC-H benchmark 86
4.16 Requête Q_5 du bechmark TPC-H 86
4.17 la variation de la consommation énergétique du système suivant les deux modes d'exécution sur 10 requêtes (PostgreSQL) 87
4.18 la variation de la consommation énergétique du système suivant les deux modes d'exécution sur 10 requêtes (MonetDB) 87
4.19 Énergie totale consommée lors de l'exécution de chaque requête dans les deux modes de traitement 87
4.20 Consommation totale du système lors de l'exécution des requêtes 88
4.21 Comparaison du temps d'exécution et consommation énergétique des requêtes avec/et sans données dans la mémoire cache. 91
4.22 Exemple graphique de la corrélation entre deux variables 93
4.23 Structure classique d'un réseau de neurone artificiel 95
4.24 Réseau de neurones récurrent 95
4.25 Anatomie d'un neurone artificiel 95
4.26 Une illustration de la structure des Random Forest 96
4.27 Architecture de l'environnement de l'expérimentation. 98
4.28 Processus d'apprentissage automatique. 98
4.29 Schéma du benchmark TPC-H. 99
4.30 Paramètres d'apprentissage de la structure neuronale (PostgreSQL) 102
4.31 Structure neuronale du modèle comprenant les poids synaptiques (PostgreSQL) . 102
4.32 Paramètres d'apprentissage de la structure neuronale (MonetDB) 103
4.33 Structure neuronale du modèle comprenant les poids synaptiques (MonetDB) . . 104
4.34 Paramètres d'apprentissage de la structure neuronale (Hyrise) 104
4.35 Structure neuronale du modèle comprenant les poids synaptiques (Hyrise) 105
4.36 Paramètre *#mtry#* lors du processus d'apprentissage du modèle. 105
4.37 La moyenne d'erreur d'estimation des trois techniques en utilisant le benchmark TPC-H (PostgreSQL). 110
4.38 La moyenne d'erreur d'estimation des trois systèmes. 115

5.1 Différents Plans d'évaluation d'une requête conduisant au même ensemble de données. 120
5.2 Coût des différents plans d'exécution de la requête Q_3 en variant le degré de Parallélisme de 0 à 4. 121
5.3 Workflow du Framework GreenPipeline. 124
5.4 Vue des instructions de la fonction *count_power_cost_NLR()*. 124
5.5 Interfaces du framework (configuration des paramètres, de visualisation du plan, Prédiction de l'énergie du plan). 126
5.6 Variations de la consommation énergétique capturée en temps réel. 127
5.7 L'architecture de MonetDB avec vue sur les optimiseurs. 129

5.8 Workflow du Framework EcoMPro. 130
5.9 Interface IHM du framework EcoMPro. 131
5.10 Architecture du framework SimQEcoIM. 133
5.11 Interface IHM du framework SimQEcoIM. 134

Liste des tableaux

1.1 Aperçu du plan d'exécution généré par par l'outil EXPLAIN (PostgreSQL) . . . 19

3.1 Fonction énergétique pour les opérations de base 65
3.2 Coût énergétique pour les deux opérateurs 66

4.1 Énergie statique et dynamique de certains composants du système. 81
4.2 Plan d'exécution généré par EXPLAIN (PostgreSQL) 85
4.3 Analyse statistique de l'erreur de prédiction obtenue avec les différentes approches. 101
4.4 Paramétrage des arguments du réseau de neurone artificiel lors des processus d'apprentissage. 103
4.5 Paramétrage des arguments de l'algorithme des forêts aléatoires lors des processus d'apprentissage. 106
4.6 Statistiques sur les caractéristiques du schéma TPC-DS. 107
4.7 Taux d'erreurs d'estimation d'énergie des requêtes en mode parallèle (PostgreSQL - TPC-H) 108
4.8 Taux d'erreurs d'estimation d'énergie des requêtes en mode parallèle (PostgreSQL - TPC-H) 109
4.9 Taux d'erreurs d'estimation d'énergie des requêtes (PostgreSQL - SSB/TPC-DS) 111
4.10 Taux d'erreurs d'estimation d'énergie des requêtes (Monet - TPC-H) 112
4.11 Taux d'erreurs d'estimation d'énergie des requêtes (Monet - SSB) 113
4.12 Taux d'erreurs d'estimation d'énergie des requêtes (Hyrise - TPC-H) 114
4.13 Taux d'erreurs d'estimation d'énergie des requêtes (Hyrise - SSB/TPC-DS) . . . 114

5.1 Valeurs des paramètres du modèle de coût de PostgreSQL. 120
5.2 Description des paramètres du modèle de coût. 122
5.3 Description des paramètres du modèle de coût. 123
5.4 Analyse de l'évaluation des plans d'exécution dans les différentes configurations. 128
5.5 Analyse de la puissance optimisée au niveau des requêtes du benchmark TPC-H. 128
5.6 Taux d'erreurs d'estimation de la puissance énergétique des requêtes (MonetDB - TPC-H/SSB) 132
5.7 Analyse de l'évaluation des plans d'exécution dans les différentes configurations (MonetDB). 132
5.8 Taux d'erreurs d'estimation de la puissance énergétique des requêtes (HSQL/H2 - SSB) 134

A.1 Description des paramètres pour l'estimation de cardinalité des opérateurs. . . . 162
A.2 Formule de calcul de sélectivité des prédicats. 162
A.3 Description des paramètres pour le calcul du coût des opérateurs. 164

Résumé

Dans le monde d'aujourd'hui, nous dépendons énormément des équipements numériques pour le travail, le divertissement et le social. Par conséquent, nous sommes obligés de chercher tous les moyens pour économiser l'énergie consommée par leurs composants matériels, logiciels, ainsi que les applications qu'ils utilisent. Les systèmes de gestion de bases de données (SGBDs) deviennent des gouffres énergétiques à cause de l'explosion massive des données qu'ils doivent collecter, traiter et stocker. Le processeur des requêtes constitue l'un des composants le plus énergivore des SGBDs. Il a pour rôle de traiter d'une manière efficace les requêtes. Vu le volume des données et la complexité des requêtes d'analyse, l'étude de l'efficacité énergétique de ce composant devient sans aucun doute une question cruciale et urgente. Cette importance a été largement soulignée dans le rapport de Claremont et cosignée par environ une trentaine de chercheurs, experts, architectes du monde de base de données. Ce point est ensuite repris dans le rapport de Beckman qui place l'efficacité énergétique comme un défi à relever dans le domaine des données massives. La majorité des optimiseurs des requêtes actuels sont conçus pour minimiser les opérations d'entrées-sorties et essayent d'exploiter la RAM autant que possible. En conséquence, ils ignorent généralement les aspects énergétiques. Dans cette thèse, pour optimiser l'énergie consommée par un SGBD, nous proposons une approche orientée logicielle que nous baptisons Auditer l'Énergie – Avant de Déployer, permettant de concevoir des processeurs de requêtes moins énergivores. Cette approche se décline en cinq étapes principales: (1) Audit des composants des processeurs de requêtes afin de comprendre son fonctionnement et déterminer ses paramètres sensibles à l'énergie. (2) Élaboration d'un état de l'art sur les solutions existantes dont l'objectif est double: (i) fournir une feuille de route pour les concepteurs et les étudiants qui souhaitent comprendre les questions liées à l'efficacité énergique dans le monde des bases de données et (ii) aider au développement des modèles énergétiques. (3) Modélisation de l'énergie des processeurs de requêtes. Cette modélisation est réalisée par la définition des modèles de coût mathématique dédiés à estimer la consommation énergétique du système lors de l'exécution des requêtes. (4) Utilisation des techniques d'apprentissage automatique profond pour identifier les valeurs des paramètres sensibles à l'énergie appartenant à la fois aux composants matériels et logiciels. Pour appliquer notre approche, nous avons choisi trois systèmes de traitement des données libres dont deux sont orientés disque: PostgreSQL, MonetDB et Hyrise. (5) Déploiement des modèles validés dans les trois SGBDs pour des études d'estimation et d'optimisation énergétique.

Mots-clés : Apprentissage automatique, Bases de données–Gestion, Bases de données–Interrogation, Consommation d'énergie, Coût, Économies d'énergie–Appareils et matériel, Évaluation énergétique.

Abstract

In today's world, we rely heavily on digital equipment for work, entertainment and social purposes. Therefore, we are obliged to seek all means to save the energy consumed by their hardware components, software, as well as the applications they use. Database management systems (DBMS) are becoming energy sinkholes due to the massive explosion of data they must collect, process and store. The query processor is one of the most energy intensive components of the DBMS. Its role is to efficiently process queries. Given the volume of data and the complexity of the analytical queries, the study of the energy efficiency of this component is undoubtedly becoming a crucial and urgent issue. This importance was widely underlined in the Claremont report and co-signed by around thirty researchers, experts and architects of the database world. This point is then echoed in Beckman's report which places energy efficiency as a challenge to be addressed in the field of massive data. The majority of actual query optimizers are designed to minimize input-output (IO) operations and try to exploit main memory (RAM) as much as possible. As a result, they generally ignore the energy aspects. In this thesis, to optimize the energy consumed by a DBMS, we propose a software-oriented approach that we call Energy Audit - Before Deployment, allowing the design of less energy-consuming query processors. This approach consists of five main steps: (1) audit of query processor components in order to understand its operation and determine its energy-sensitive parameters. (2) Development of a state of the art on existing solutions with the dual purpose of (i) providing a roadmap for designers and students to understand energy efficiency issues in the database world and (ii) assisting in the development of energy models. (3) Query processors energy modeling, this modeling is achieved by defining mathematical cost models dedicated to estimate the energy consumption of the system during queries execution. (4) Using deep machine learning techniques to identify values of energy-sensitive parameters belonging to both hardware and software components. To apply our approach, we have chosen three open-source data processing systems, two of which are disk oriented: PostgreSQL (a row storage system) and MonetDB (a column storage system) and a main memory oriented Hyrise (a hybrid storage system). We then validate our approach using a series of experiments using data from several benchmarks (TPC-H, TPC-DS and SSB) and an apparatus to capture energy (Watts-Up-Pro). (5) The deployment of validated models in SGBDs for energy estimation and optimization studies.

Keywords: Machine learning, Database management, Querying (Computer science), Energy consumption, Costs, Energy conservation–Equipment and supplies, Energy auditing.
