République Algérienne Démocratique et Populaire

Ministère de l'enseignement supérieur et de la recherche scientifique

Université de 8 Mai 1945 - Guelma -

Faculté des Mathématiques, d'Informatique et des Sciences de la matière

Département d'Informatique

Mémoire de fin d'études Master

Filière : Informatique

Option : Système informatique

Thème :

Réalisation d'un système de recherche d'information parallèle

Encadré Par :

Dr. SOUSSI HAKIM

Présenté par :

BENCHEIKH
SALAH EDDINE

Juin 2018

Remerciements

Mes remerciements vont à mes parents pour tous les sacrifices qu'ils ont consentis pour me permettre de suivre mes études dans les meilleures conditions possibles et n'avoir jamais cessé de nous encourager tout au long de nos années d'études.

Je remercie mon ami DJAGOUT BILAL dont l'aide et le soutien pour accomplir ce travail.

Je remercie aussi mon encadreur monsieur HAKIM SOUSSI dont la disponibilité et le savoir-faire.

Enfin, je remercie tous les professeurs du département de l'informatique de l'université de 8 mai 1945 à Guelma.

Résumé

La recherche d'information est le domaine qui s'intéresse à l'extraction des informations. En d'autres termes, elle fournit des outils qui répondent aux besoins des utilisateurs exprimés sous forme de requêtes. Ces outils sont appelés système de recherche d'information (SRI).

Le fonctionnement d'un SRI est généralement résumé en deux grandes étapes : l'indexation et la comparaison. Cependant la nécessité d'avoir les résultats plus vite exige l'intégration d'outils qui permettent l'accélération du processus de recherche d'information ce qui implique l'utilisation des techniques de parallélisme.

Le but de ce travail est de mettre au point un système de recherche d'information parallèle dont le but est d'accroitre les performances en termes de temps par rapport aux systèmes de recherche d'informations séquentiels classiques.

Mots clés : Recherche d'information, parallélisme, indexation, comparaison

Sommaire

Remerciements i

Résumé ii

Tables des matières iii

Liste des figures vi

Liste des tableaux vii

Introduction générale 1

Chapitre 1 : La recherche d'information 4

1. Introduction 4

2. Concepts de base de recherche d'information 4

3. Le processus d'indexation 8

3.1. L'analyse lexicale 9

3.2. La sélection 10

3.3. La radicalisation 10

3.3.1. Algorithme de LOVINS 11

3.3.2. Algorithme de PORTER 11

3.3.3. Algorithme de CARRY 16

3.4. La pondération 16

4. Le processus de recherche 16

4.1. Le modèle booléen 17

4.2. Le modèle probabiliste 18

4.3. Le modèle vectoriel 19

5. Reformulation de la requête 21

6. Le processus d'ordonnancement 22

6.1. Tri fusion de BATCHER 22

6.2. Tri par transposition pair impair 23

7. Les mesures d'évaluation 25

8. Conclusion 26

Chapitre 2 : Le parallélisme 28

1. Introduction 28
2. Les concepts de base de parallélisme 28
3. Pourquoi le parallélisme ? 31
4. Historique d'évolution du parallélisme 31

4.1. L'évolution du système parallèle 31

4.2. L'évolution au sein du processeur 32

5. Les architectures parallèles 35
6. Les types de parallélisme 37

6.1. Parallélisme des tâches 37

6.2. Parallélisme des données 37

7. Les modèles de programmation parallèles 38

7.1. Le modèle à mémoire distribuée 38

7.2. Le modèle à mémoire partagée 39

8. Conclusion 40

Chapitre 3: Conception et implémentation 42

1. Introduction 42
2. L'architecture du système de recherche 42
3. Principe du fonctionnement du système séquentiel 45
4. Principe du fonctionnement du système parallèle 46
5. Intégration du parallélisme 47
6. Conception du l'application 48
7. Comparaison entre le système parallèle et le système séquentiel 51
8. Conclusion 54

Conclusion générale 55

Perspectives 56

Annexe 57

Bibliographie 57

Liste des figures

Figure 1 Processeurs en U de la recherche d'information [43] 5
Figure 2 Les processus d'un SRI 8
Figure 3 Requêtes booléennes sous forme de diagramme de Venn [34] 18
Figure 4 Tri de deux liste de taille = 1 23
Figure 5 Fusion de BATCHER de deux listes de taille = 2 23
Figure 6 Fusion de BATCHER de deux listes de taille = 4 23
Figure 7 transposition pair impair avec taille= 6 24
Figure 8 Tri transposition pair impair avec taille =5 25
Figure 9 Machine à mémoire partagé 29
Figure 10 Machine à mémoire distribué 29
Figure 11 Machine à mémoire hybride 30
Figure 12 Progression de cinq instructions dans le pipeline à trois étages [38] 33
Figure 13 Architecture de processeur à quatre cœurs [38] 35
Figure 14 Architecture SMP à quatre processeurs [38] 35
Figure 15 Architecture NUMA à deux nœuds [38] 36
Figure 16 Machine SIMD 36
Figure 17 Machine MIMD 37
Figure 18 Mécanisme d'envoie et réception de données entre les processeurs 39
Figure 19 Architecture du système de recherche d'information 44
Figure 20 Diagramme d'activité de système séquentiel 46
Figure 21 L'exécution parallèle de l'indexation 47
Figure 22 L'exécution parallèle de la partie comparaison 48
Figure 23 Diagramme de cas d'utilisation 49
Figure 24 L'interface principale 50
Figure 25 La fenêtre SHOW parallèle 50
Figure 26 La fenêtre SHOW séquentiel 50
Figure 27 Diagramme en barres du temps d'indexation parallèle et séquentiel 51
Figure 28 Le gain de temps durant l'indexation 52
Figure 29 Diagramme en barres du temps de recherche parallèle et séquentiel 53
Figure 30 Le gain de temps dans l'étape de comparaison 53
Figure 31 La courbe du gain total 54

Liste des tableaux

Tableau 1 L'évaluation de la requête en modèle booléen.....17
Tableau 2 Taxinomie de Flynn 36
Tableau 3 Variation du temps d'indexation 51
Tableau 4 Variation du temps de comparaison.....52
Tableau 5 Le gain total en seconde.....53
Tableau 6 Extraire la moyenne de temps pour le temps d'indexation.....56
Tableau 7 Extraire la moyenne de temps pour le temps de recherche.....56

Introduction générale

L'informatique est le traitement automatique de l'information, cette dernière joue actuellement un rôle primordiale dans le quotidien des individus. Cependant l'intégration de l'informatique à tous les domaines à conduit à la production d'un volume important d'informations numériques. Par conséquent, il devient difficile d'obtenir l'information pertinente rapidement.

La solution est de développer des outils informatiques qui permettent le stockage et l'organisation de cette masse d'information ainsi que de pouvoir la localiser rapidement repondant ainsi à un besoin en information d'un utilisateur. Ces outils sont appelés ***Système de Recherche d'Information (SRI).*** Un SRI est créé pour gérer une collection de documents stockés sous forme d'une représentation intermédiaire permettant de refléter aussi fidèlement que possible leur contenu sémantique.
Le rôle du SRI peut être défini comme l'ensemble des procédures et des opérations permettant la gestion, la représentation, l'interrogation, la recherche, le stockage et la sélection des informations.

Généralement, le processus de recherche d'information au niveau des SRI passe par deux étapes :

- **L'indexation :** elle détermine de quelle manière les connaissances contenues dans les documents fournis sont représentées, elle permet aussi de réduire la complexité des documents et les rendre plus faciles à manipuler. Elle a lieu à chaque ajout d'un document dans l'ensemble des documents étudiés.
- **La comparaison :** appelé aussi l'appariement document-requête, l'objectif de cette étape est de comparer le degré de similarité entre un document et une requête utilisateur en s'appuyant sur un modèle de recherche.

Le but de ce travail est de mettre au point un système de recherche d'information parallèle pour l'indexation automatique et la recherche des documents dont le but est d'accroitre les performances des systèmes de recherche d'informations (SRI) en termes de temps par rapport aux systèmes séquentiels classiques.

Ce mémoire est organisé en trois chapitres :

Le premier chapitre sera consacré aux concepts de base de la RI, ainsi qu'aux grandes étapes du processus de la recherche d'information, à savoir l'indexation, les modèles de recherche d'information. On verra aussi quelques algorithmes existants qui traitent la radicalisation tels que les algorithmes de Porter et Carry.

Le deuxième chapitre présentera quelques concepts de base utilisés dans le parallélisme, ainsi que les différentes architectures parallèles, les types de parallélisme et les modèles de programmation parallèles.

Dans le troisième chapitre, une description du modèle issu de cette recherche ainsi que l'implémentation et les résultats obtenus seront présentés et discutés.

Enfin, une conclusion finale et les perspectives viendront résumer le travail effectué ainsi que les éventuelles améliorations qui peuvent être ajoutés.

Chapitre 1 :

La recherche d'information

Chapitre 1 : La recherche d'information

1. Introduction

La recherche d'information n'est pas un domaine récent, il date des années quarante. Il s'intéresse à l'extraction d'un document ou un ensemble de document qui répond à une requête d'utilisateur.

Dans ce chapitre nous allons définir des concepts de base de recherche d'information, puis nous allons détailler les différentes étapes de la phase d'indexation, nous allons détailler aussi les modèles de comparaison. Enfin, nous allons expliquer deux phases secondaires qui sont l'ordonnancement et la reformulation de requête.

2. Concepts de base de recherche d'information

Tout d'abord, il existe plusieurs définitions de recherche d'information nous citons dans ce contexte les quatre définitions suivantes :

1) **Définition 1 :** « la recherche d'information est un domaine qui pour objectif la représentation, l'analyse, le stockage et l'accès à l'information » [1].
2) **Définition 2 :** « la recherche d'information est une activité dont la finalité est de localiser et de délivrer des granules documentaires » [2].
3) **Définition 3 :** « la recherche d'information est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche, la sélection de l'information » [3].
4) **Définition 4 :** « est une discipline de recherche qui intègre des modèles et des techniques dont le but est de faciliter l'accès à l'information pertinente pour un utilisateur ayant un besoin en information » [4].

La plupart des définitions ci-dessus partagent la même idée est que la recherche d'information est une branche qui étudie la sélection d'information pertinente qui répond à un besoin d'utilisateur, généralement cette sélection est faite des SRI.

L'architecture générale d'un SRI illustré par (la figure 1) fait ressortir des éléments tel que : document, requête, ainsi des fonctionnalités : indexation et recherche qui sont

les principales parties qui constituent un SRI. D'autres parties peuvent constituer un SRI tel que la partie de reformulation de requête qui est un mécanisme qui prend la requête initiale formulée par l'utilisateur et renvoie des renvoie des suggestions proche de la requête initiale dont le but est d'extraire d'autres résultats.

Un autre processus d'amélioration est le processus d'ordonnancement, il permet d'ordonner les documents retournés par le SRI selon le degré de pertinence pour accéder rapidement aux documents les plus pertinents.

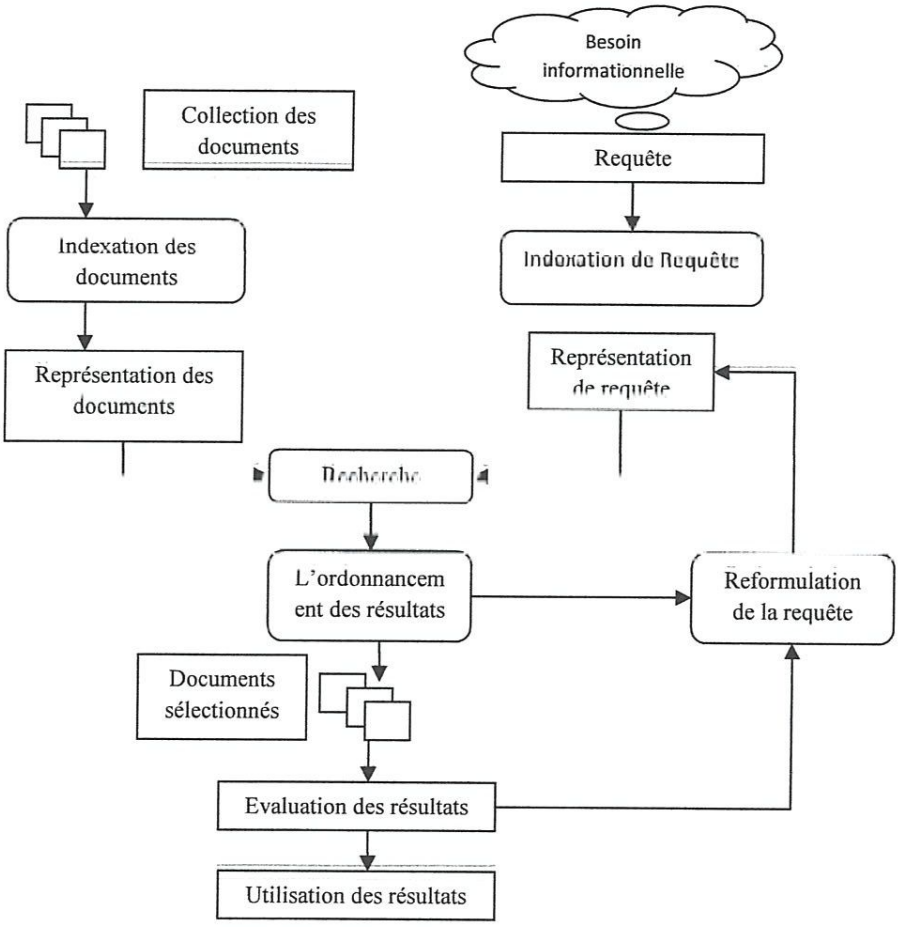

Figure 1 processus en U de la recherche d'information [43]

- ***La collection de document :*** ou le fond documentaire est un ensemble de document stocké sur un support.
- ***Document :*** il est l'information élémentaire de la collection de document qui peut être du texte, une page web, une image, une séquence vidéo. Dans ce travail les documents traités sont les documents textuels numériques.

- ***Besoin en information :*** c'est la partie de réflexion. Cosijn et Ingwersen [5] ont défini trois types de besoin utilisateur :
  - ***Besoin vérificatif :*** un besoin de type vérificatif est dit stable c'est-à-dire qu'il ne change pas au cours de la recherche. Ici l'utilisateur cherche à vérifier le texte avec des données connues qu'il possède déjà. Il cherche donc une donnée particulière et sait même comment y accéder. Par exemple : recherche un article sur internet à partir d'une adresse connue ou encore chercher la date de publication d'un ouvrage dont la référence est connue.
  - ***Besoin thématique connu :*** l'utilisateur cherche à clarifier, à revoir ou à trouver des nouvelles informations dans un sujet et un domaine connu. Un besoin de ce type peut être stable ou variable. Ici l'utilisateur peut exprimer leur besoin de façon incomplète c'est-à-dire que l'utilisateur n'énonce pas nécessairement tout ce qu'il sait dans sa requête mais un sous ensemble.
  - ***Besoin thématique inconnu :*** ici l'utilisateur cherche de nouveaux concepts ou de nouvelles pistes en relation avec les sujets et les domaines qui lui sont familiers. Le besoin est variable et il est toujours exprimé par l'utilisateur de façon incomplète.
- ***Requête :*** elle constitue l'expression de besoin en information de l'utilisateur, elle représente l'interface ou le lien entre l'utilisateur et le système. Elle peut être exprimée de différentes manières :
  - Une liste de mots clés : le cas de système Okapi [6],
  - En langage naturel : cas de système SMART [7] et SPIRTIT [8],
  - En langage booléen : c'est le cas de système DIALOG [9].
  - En langage graphique : cas du système NEURODOC [10].
- ***Modèle de représentation interne :*** il représente le résultat de l'indexation. En d'autre terme, c'est un descripteur de document et de requête.
- ***Modèle de recherche :*** c'est les étapes à suivre pour associer un document ou plusieurs documents à une requête donnée.
- ***Pertinence :*** est une valeur donnée après l'évaluation de degré de similitude entre un document et une requête. Il existe deux types de pertinence : pertinence système et pertinence utilisateur.
  - ***Pertinence système :*** cette pertinence est donnée par le système, elle est représentée par une valeur ou un score qui représente du contenu des documents

vis-à-vis de celui de la requête [11]. Ce type de pertinence est objectif et déterministe.

- ***Pertinence utilisateur :*** ici l'utilisateur lui-même évalue la pertinence des documents restitués par le SRI. La pertinence utilisateur est subjective car pour un même document retourné en réponse à une même requête, il peut être évalue différemment par deux utilisateurs distincts (qui ont des centres d'intérêt différents). De plus, cette pertinence est évolutive c'est-à-dire un document jugé non pertinent à l'instant « ***t*** » pour une requête peut être jugé pertinent à l'instant « ***t+1*** », car la connaissance de l'utilisateur sur le sujet peut évoluer [11].

- ***Appariement document-requête :*** la fonction d'appariement document-requête permet de mesurer la valeur de pertinence d'un document vis-à-vis d'une requête. Afin de realiser cette fonction, le SRI représente le document et la requête avec un même formalisme, puis le SRI compare les deux représentations. Le résultat de cette comparaison se traduit par un score qui détermine le degré de similitude ou de ressemblance du document vis-à-vis de la requête. Cette fonction d'appariement est notée RSV (q, d) (Retrieval Statut Value), où ***d*** représente un document de la collection et ***q*** la requête. Il existe deux types d'appariement :
  - ***Appariement approché :*** les documents retournés sont triés selon le degré de pertinence.
  - ***Appariement exacte :*** les documents retournés ne sont pas triés et respectent exactement la requête.

- ***Les SRI :*** un système de recherche d'information est défini par un modèle de représentation interne des documents et de requête, ainsi qu'une fonction de mise en correspondance des deux représentations internes de document et de requête en vue de fournir comme résultats des documents pertinents pour l'utilisateur c'est-à-dire répondant à son besoin en information [12].

En générale, l'opérationnalisation de la RI est réaliser par ces systèmes informatiques qui permettent de retourner à partir d'un ensemble de documents, ceux dont le contenu correspond le mieux à un besoin en information d'un utilisateur, exprimé à l'aide d'une requête.

Les processus d'un SRI :

Un système de recherche d'information est composé de processus de base et de processus d'amélioration, la figure suivante résume les processus d'un SRI.

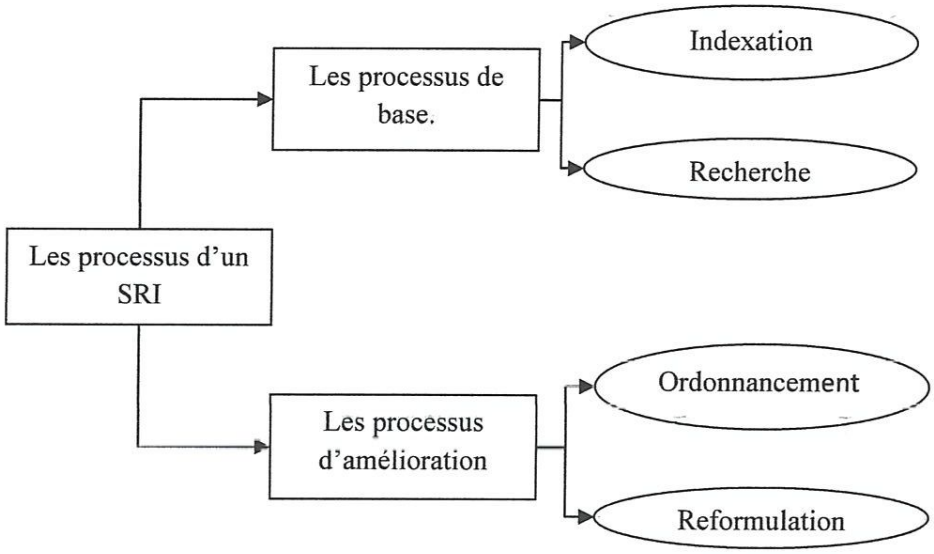

Figure 2 Les processus d'un SRI

Les processus de base :

Les processus de base sont les principaux processus dans la construction d'un SRI. Il existe deux processus :

- **L'indexation :** elle a pour objectif la représentation interne de document et requête.
- **La recherche :** elle a pour objectif de faire la correspondance entre la représentation interne de document et la représentation interne de requête.

3. Le processus d'indexation

Pour que la recherche d'information se réalise avec des couts acceptables, il convient d'effectuer une opération fondamentale sur les documents de la collection. Cette opération est nommée « indexation » [13] [14]. Elle consiste à associer à chaque document une liste de mots clés appelée aussi descripteur, susceptible de représenter au mieux le contenu sémantique des documents [15]. L'indexation peut être manuelle, automatique ou semi-automatique.

- **L'indexation manuelle :**

En ce mode d'indexation, un opérateur humain (généralement un expert de domaine) qui se charge de caractériser selon ses connaissances propres le contenu sémantique d'un document [16] [17] [18]. Cette approche permet de retourner un indexe plus proche de document originale car il est faite par un être humain qui peut extraire exactement le sémantique de document puis le extraire les descripteurs de ce document, néanmoins elle présente deux inconvénients :

- Elle est subjective puisque le choix des termes d'indexation dépend de l'indexeur et de ses connaissances,
- Elle est pratiquement inapplicable aux corpus de textes volumineux.

- **Indexation automatique :**

C'est un processus complètement automatisé qui se charge d'extraire les termes qui décrit le document [19] [20]. L'intérêt d'une telle approche réside dans sa capacité à traiter les documents plus rapidement que l'approche manuelle, et de ce fait elle est particulièrement adaptée aux corpus volumineux.

- **Indexation semi-automatique :**

Cette approche est appelée aussi « indexation supervisée », elle est une combinaison des deux approches d'indexation précédentes. Dans ce cas les indexeurs utilisent un vocabulaire contrôlé sous de thésaurus ou de base terminologique [21] [22] [23]. Le choix final des termes d'indexation à partir du vocabulaire fourni est laissé ainsi à l'indexeur humain.

Le processus d'indexation automatique se fait en quatre sous processus suivants : l'analyse lexicale, la sélection ou la suppression des mots vides de sens, la racinisation ou la radicalisation, et la pondération.

1.1. L'analyse lexicale

L'analyse lexicale est la première étape de processus d'indexation automatique qui permet de transformer le document en un ensemble de termes seulement c'est-à-dire la ponctuation, la casse (majuscule et minuscule), et la mise en page sont supprimées.

1.2. La sélection

Afin de ne garder que les termes significatifs, plusieurs techniques peuvent être mise en œuvre [24]. Parmi celles-ci, la technique de ***stoplist*** ou l'anti dictionnaire, cette technique permet de supprimer les mots existants dans un anti dictionnaire qui ne reflète pas le contenu informationnel de document et garder les mots qui ne sont pas existants dans l'anti dictionnaire qui reflète le contenu informationnel de documents, ces mots sont appelés les descripteurs de documents.

Généralement, la liste de l'anti dictionnaire contient les articles définis et les articles prédéfinis, les pronoms, les prépositions ainsi que les mots mathématiques. Tous ces mots n'ayant pas de réel rapport avec le sujet traité.

La suppression de mots vides de sens doit être contrôlée car elle influence sur la qualité de la recherche. Il existe des anti-dictionnaires définis comme l'anti dictionnaire du système SMART. Il est possible d'enrichi ces anti-dictionnaires avec d'autres mots vides de sens.

1.3. La radicalisation

La radicalisation est une procédure qui prend les descripteurs d'un document puis appliquer à chaque descripteur une succession des traitements afin d'extraire leur radical.

Le radical d'un mot correspond à la partie du mot restante après la suppression de son affixe (suffixe et préfixe). Elle est parfois connue sous le nom de *stem* d'un mot contrairement au lemme qui correspond à un mot réel de la langue [1].

Le *stem* ne correspond généralement pas à un mot réel de langue. Par exemple le mot « chercher » après la radicalisation sera « cherch » ce radical ne correspond pas à un mot réel de langue française.

Le sous processus « radicalisation » est important pour le processus d'indexation car leur résultat est le regroupement des différentes formes d'un mot autour de leur *stem* qui limite l'espace de mémoire exploité et sauvegarde une grande quantité d'information et cette avantage est forte contrainte de l'indexation.

Les techniques utilisées qui valide ce processus reposent généralement sur une liste d'affixes de la langue considérée et sur un ensemble de règles de dé-suffixations

constitués à priori. Il existe deux types de racinisation : racinisation à base de dictionnaire et racinisation algorithmique.

Pour le racinisation à base de dictionnaire, il existe une liste de racines c'est-à-dire chaque mot a son racine dans cette liste. Cette racinisation est limité pour les mots connus.

Par contre la racinisation algorithmique, il existe un algorithme qui prend un mot quelconque de langue considérée puis le passe par une succession des règles pour obtenir leur radical ou leur racine. Cette approche est souvent plus rapide mais le taux d'erreur est plus élevé car parfois il existe des mots qui n'acceptent pas la racinisation et l'algorithme ne traite pas cette exception, cette situation est nommée « sur racinisation » ou « over stemming ».

Plusieurs algorithmes ont été développés pour raffiner la tâche de racinisation, ces algorithmes procèdent en deux étapes : la première consiste à éliminer les suffixes selon des règles définis, la deuxième consiste à recoder le radical en ajoutant des terminaisons prédéfinis.

L'algorithme de LOVINS procède les deux étapes l'une après l'autre par contre l'algorithme de PORTER procède les deux étapes en simultané. Dans ce qui suit nous allons expliquer trois algorithmes de racinisation (radicalisation) suivants : l'algorithme de LOVINS, PORTER et CARRY.

1.3.1. Algorithme de LOVINS

Le premier stemmer a été développé par Julie Beth Lovins en 1968 [2]. C'est un racinisateur algorithmique qui travaille sur les mots de la langue anglaise comportant deux étapes : la première étape a pour objectif de déterminer l'emplacement du mot dans la liste des terminaisons puis chercher une correspondance entre le suffixe du mot l'une des terminaisons de la liste. S'il existe une correspondance alors l'algorithme applique une règle parmi les 35 règles existantes.

La deuxième étape « *recodage* »consiste à recoder le *stem* c'est-à-dire ajouter des terminaisons au *stem* cette étape est faite après la première étape.

1.3.2. Algorithme de PORTER

L'algorithme de Porter est l'un des plus célèbres algorithmes de normalisation des mots qui a été développé par Martin PORTER en 1979 au laboratoire informatique au sein de l'université du Cambridge (Angleterre). Cet algorithme a fait ses preuves pour la langue anglaise [3] [4] et il a été appliqué à d'autres langues, notamment au français et à l'italien [5] et à l'allemand [6].

Cet algorithme permet d'éliminer les affixes de mot afin de regrouper plusieurs termes autour de leur racine, ce regroupement réduit le nombre totale de terme dans le système ce qui implique la réduction de temps consacré à la recherche d'un mot existe ou pas et éventuellement la réduction de temps de réponse et l'amélioration des performances de système de recherche.

- ***Description de l'algorithme de PORTER :***

Cet algorithme se compose d'une cinquante de règles classées en sept phases successives [7]. Les mots à analyser passent par tous les stades, dans le cas où plusieurs règles pourraient leur être appliquées, c'est toujours celle comprenant le suffixe le plus long qui est choisie. Le recodage est accompagnée c'est-à-dire dans la même étape de l'élimination du suffixe il existe l'étape de recodage.

L'algorithme de PORTER comprend aussi des règles de contexte, qui indiquent les conditions dans lesquelles un suffixe devra être supprimé. Par exemple la terminaison en « ing » ne sera enlevée que si le radical comporte au moins une voyelle. De cette manière le terme « writing » deviendra « writ » alors que le mot « sing » restera « sing ».

- ***Des notations simplificatrices :***

Une consonne dans un mot est une lettre autre que A, E, I, O ou U, et autre que Y précédé d'une consonne. Par exemple dans TOY les consonnes sont T et Y, et dans SYZYGY elles sont S, Z, G. si une lettre n'est pas une consonne, elle est une voyelle.

Une consonne sera notée par « ***c*** », une voyelle par « ***v*** », une liste de consonne de longueur supérieure à 0 par « ***C*** » et la liste de voyelle de longueur supérieure à 0 par « ***V*** ».

Tout mot ou partie de mot a l'une des quatre formes :

CVCV…C

CVCV…V

VCVC…C

VCVC…V

En utilisant (***VC***) ***m*** pour noter : « ***VC*** répété ***m*** fois » ceci peut être écrit comme : [***C***] (***VC***) {***m***} [***V***] ***m*** sera appelée la mesure de tout mot ou partie de mot si elle est représenté sous cette forme. Voici quelques exemples :

m = 0 TR, EE, ARBRE, Y, BY.

m = 1 TROUBLE, OATS, TREES, IVY.

m = 2 TROUBLES, PRIVATE, OATEN, ORRERY.

Les règles pour supprimer un suffixe seront données sous la forme :

(Condition) S1 - >S1cela signifie que si un mot se termine par le suffixe S1 et que la partie avant S1 satisfait la condition donnée, S1 est remplacé par S2. La condition est généralement donnée en terme de ***m*** par exemple : (***m*** >1) ement -> , ici S1 est « ement » et S2 est nul, cela permette de remplacer « replacement » à « replac ».

La partie « condition » peut également contenir les éléments suivants :

* S : le radical se termine par S (et de même pour les autres lettres).

* v * : le radical contient une voyelle.

* d : le radical se termine par une double consonne.

* o : le radical se termine par « ***cvc***» où le seconde ***c*** n'est pas W, X ou Y par exemple …WIL, …HOP.

Aussi la partie condition peut contenir des expressions avec AND, OR ou NOT par exemple (***m***>1 AND (*S OR *T)).

Si possible d'appliquer plusieurs règle sur un mot donné au même temps la règle qui contient le suffixe le plus grand sera applicable en premier par exemple : sur le mot « CARESSES » nous pouvons appliquer les deux règles suivantes (SSES ->SS) et aussi (IES -> I) dans ce cas la première règle sera applicable en premier.

- ***Les étapes de l'algorithme de PORTER :***

L'algorithme suit les étapes suivantes :

Étape 1a	
La règle	Exemple
SSES -> SS IES -> I SS -> SS S ->	Caresses -> caresse Ponies -> poni Caress -> caress cats -> cat

Étape 1b	
La règle	Exemple
(m> 0) EED -> EE (* v *) ED -> (* v *) ING ->	Feed -> Feed Agreed -> agree Palastered -> palaster Motoring -> motoring

Si la deuxième ou la troisième des règles de l'étape 1b réussit, les opérations suivantes sont effectuées:

La règle	Exemple
AT -> ATE BL -> BLE IZ -> IZE (* d et non (* L ou * S ou * Z)) -> lettre unique (m = 1 et * o) -> E	Conflat -> conflate. Troubl -> trouble Siz -> size Hopp -> hop Fil -> file
Étape 1c	
La règle	Exemple
(* v *) Y -> I	Happy -> happi
Étape 2	
La règle	Exemple
(m>0)ational -> Ate (m >0)tional -> tion (m >0)enci -> ence (m >0)anci -> ance (m >0)izer -> ize (m >0)abli -> able (m >0)alli -> al (m >0)entli -> ent (m >0)eli -> e (m >0)ousli -> ous (m >0)ization -> ize (m >0)ation -> ate (m >0)ator -> ate (m >0)alism -> al (m >0)iveness -> ive (m >0)fulness -> ful (m >0)ousness -> ous (m >0)aliti -> al (m >0)iviti -> ive (m >0)biliti -> ble	relational -> relate Conditional -> Condition valenci -> valence hesitanci -> hesitance digitizer -> digitize conformabli -> conformable radicalli -> radical differentli -> different vileli -> vile analogousli -> analogous vietnamization -> vietnamize predication -> predicate operator -> operate feudalism -> feudal decisiveness -> decisive hopefulness -> hopeful callousness -> callous formaliti -> formal sensitiviti -> sensitive sensibiliti -> sensible

Étape 3	
La règle	Exemple
(m >0)icate -> ic	triplicate -> triplic
(m >0)ative ->	formative -> form
(m >0)alize -> al	formalize -> formal
(m >0)iciti -> ic	electriciti -> electric
(m >0)ical -> ic	electrical -> electric
(m >0)ful ->	hopeful -> hope
(m >0)ness ->	goodness -> good
Étape 4	
La règle	Exemple
(m >1)al ->	revival -> reviv
(m >1)ance ->	allowance->allow
(m >1)ence ->	inference ->infer
(m >1)er ->	airliner -> airlin
(m >1)ic ->	gyroscopic -> gyroscop
(m >1)able ->	adjustable ->adjust
(m >1)ible ->	defensible -> defens
(m >1)ant ->	irritant -> irrit
(m >1)ement ->	replacement ->replac
(m >1)ment ->	adjustment -> adjust
(m >1)ent ->	dependent -> depend
(m >1 et (* s ou * t))ion ->	Adoption ->adopt
(m >1)ou ->	homologou ->homolog
(m >1)ism ->	communism -> commun
(m >1)ate ->	activate -> activ
(m >1)iti ->	angulariti -> angular
(m >1)ous ->	homologous -> homolog
(m >1)ive ->	effective -> effect
(m >1)ize ->	bowdlerize -> bowdler
Étape 5	
La règle	Exemple
(m >1)e ->	Probate -> probate
(m >1 et non *o)e ->	Cease -> ceas
(m >1 et *d et *l) -> Lettre non doublée	Control l ->control

1.3.3. Algorithme de CARRY

Cet algorithme est une version améliorée de l'algorithme de PORTER avec en plus les suffixe du français.

Tout celui de PORTER, l'algorithme de CARRY se déroule en diverse étapes par les quelles les mots à traiter passent successivement. Selon les règles, quand l'analyseur reconnaît un suffixe de la liste, soit il le supprime soit il le transforme. Ici aussi le suffixe le plus long qui détermine la règle à appliquer.

1.4. La pondération

Dans cette étape on a associé à chaque terme un poids wij qui signifie l'importance de ce terme dans le document de la collection. Cette pondération est calculée par la formule suivante :

$$w_{ij} = tf_{ij} \ / df_j \quad [42]$$

Où :

- tf_{ij} est la fréquence d'occurrences du terme tj dans le document d_i.
- df_j est la fréquence documentaire du terme tj (i.e. la proportion de documents de la collection qui contiennent tj) et $idf_j = 1/ df_{ij}$ est sa fréquence documentaire inverse.

La mesure *tf * idf* est une bonne approximation de l'importance d'un terme dans un document particulièrement dans des corpus de documents de tailles intermédiaires [3].

4. Le processus de recherche

Tout système de recherche d'information s'appuie sur un modèle de recherche d'information. Ce modèle se base sur une fonction de correspondance qui met en relation les descripteurs d'un document avec ceux d'une requête en établissant une relation d'égalité entre ces termes [32].

Il existe un certain nombre de modèles théoriques dans la littérature les plus connus étant le « *le modèle booléen* », le « *modèle vectoriel* » et le « *modèle probabiliste* ».

Dans le modèle booléen, l'appariement document-requête est exacte c'est-à-dire le système qui s'appuie sur cette modèle évalue un document par apport à une requête par : pertinent ou pas. Par contre Le modèle probabiliste tente d'estimer la probabilité qu'un document soit pertinent pour une requête donnée [33], aussi que le modèle

vectoriel considère les documents et les requêtes comme des vecteurs pondérés et l'appariement document-requête est approchée.

4.1. Le modèle booléen

Ce modèle de recherche [44] considère la requête comme une équation logique où les opérandes sont les termes et les opérateurs sont soit *AND, OR, NOT*. Il est définit par un quadruplet (*T, Q, D, F*) où :

- *T* : est l'ensemble des termes d'indexation.
- *Q* : est la requête logique
- *D* : la collection de document.

F : la fonction de présence défini par : $D*Q \quad \{0, 1\}$

$F(D, T) = 1$ Si *T*est présent dans *D*

0 Sinon.

L'évaluation de la requête au niveau du système dépend de la forme de cette dernière :

Fonction de présence	Niveau théorique	Niveau système
F (*D*, T_1 **et** T_2)	**Min** (F (*D*, T_1), F (D, T_2))	F (*D*, T_1) * F (*D*, T_2)
F (*D*, T_1 **ou** T_2)	**Max** (F (*D*, T_1), F (D, T_2))	F (*D*, T_1) + F (*D*, T_2) - F (*D*, T_1) * F (*D*, T_2)
F (*D*, **Non** *T*)	1- F (*D*, *T*)	1- F (*D*, *T*)

Tableau 1 L'évaluation de la requête en modèle booléen

Exemple 1 :

La requête *information ET renseignement* donne comme résultat les documents qui ont été indexés avec les deux termes, c'est-à-dire l'intersection des deux ensembles.

Exemple 2 :

La requête *information OR renseignement* retrouvera les documents qui ont été indexés avec *information* ou *renseignement* (ou les deux), le résultat est donc l'union des deux ensembles.

La figure suivante représente les documents restitués (parties grisés) pour les deux exemples :

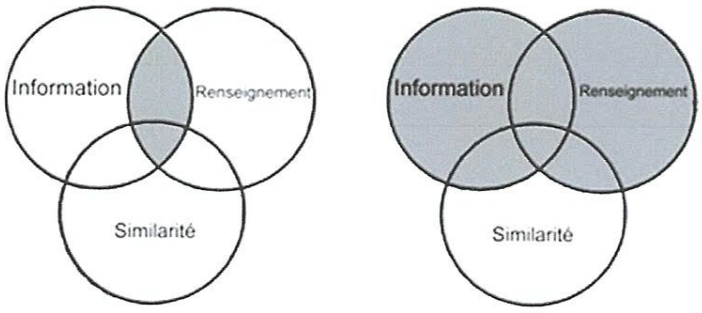

Figure 3 requêtes booléennes sous forme de diagramme de Venn [34]

Le principal avantage du modèle booléen est sa simplicité de mise en œuvre, toutefois, il présente deux principaux inconvénients [9]:

- Etablir une requête logique nécessite un savoir-faire.
- La fonction d'appariement n'est pas une fonction d'ordre.

4.2. Le modèle probabiliste

Le premier modèle probabiliste a été proposé par Maron et Kuhns [35] au début des années 60, il utilise un modèle mathématique fondé sur la théorie de la probabilité.

Le principe de base du modèle probabiliste consiste à présenter les résultats d'un SRI dans un ordre basé sur la probabilité de pertinence d'un document vis-à-vis d'un requête, c'est-à-dire le modèle prend une requête notée Q et un document noté D puis estime la probabilité que le document D appartient à la classe des documents pertinents $P(R/D)$ (ou non pertinent $P(NR/D)$). Le score d'appariement entre un document et une requête $RSV(D, Q)$ est donné par :

$$RSV(D,Q) = \frac{p\left(\frac{R}{D}\right)}{p\left(\frac{NR}{D}\right)} \quad [10].$$

Parmi les modèles qui sont utilisées pour estimer ces différentes probabilités le modèle d'interdépendance binaire, connu sous le modèle BIR (Binary Independance Retrieval).

4.3. Le modèle vectoriel

Dans ce modèle [44], un document est représenté sous forme d'un vecteur de descripteurs et leurs poids. Formellement, un document $\boldsymbol{d_i}$ est représenté par un vecteur de dimension $\boldsymbol{n}$.

$\boldsymbol{d_i} = ((t_{i1}, w_{i1}), (t_{i2}, w_{i2}), \ldots, (t_{in}, w_{in}))$ *pour* $\boldsymbol{i} = 1, 2, \ldots, m$ où :

- t_{ij} est le terme j dans le document i.
- w_{ij} est le poids du terme t_j dans le document d_i .
- m est le nombre de documents dans la collection.
- n est le nombre de terme d'indexation.

Une requête $\boldsymbol{Q}$ est aussi représentée par un vecteur de mots clés :

$\boldsymbol{Q} = ((t_{Q1}, w_{Q1}), (t_{Q2}, w_{Q2}), \ldots, (t_{Qn}, w_{Qn}))$ où :

- w_{Qj} est le poids du terme t_j dans la requête $\boldsymbol{Q}$.

Le poids d'un terme de requête peut être soit une forme de *tf *idf* soit un poids attribué manuellement par l'utilisateur.

Le modèle vectoriel exprime le degré de similitude entre un document et la requête par l'un des mesures suivantes :

Le produit scalaire :

$$Sim(d_i, Q) = \sum_{j=1}^{n} w_{qj} * \ w_{qj}$$

Cette formule permet de multiplier le poids d'un terme d'indexation dans la requête par le poids du même terme dans un document (les autres termes ne sont pas considérés).

Les documents ayant les plus hauts degrés de correspondance sont retournés en réponse pour cette requête.

Exemple explicatif :

Soit une requête représentée par le vecteur suivant :

Q ((système, 1), (informatique, 1), (master, 1)).

Un document D1 est représenté aussi par un vecteur :

D1 ((système, 35), (informatique, 50), (master, 0)).

Un document D2 est représenté par :

D2 ((système, 0), (informatique, 10), (master, 20)).

Quel est le document le plus pertinent ? la réponse est : le document D1 est le plus pertinent car selon la formule de produit scalaire les degrés de similitude sont calculés comme suit :

Sim (D1, Q) = 35 * 1 + 50 * 1 +0 * 1 =85.

Sim (D2, Q) = 0*1 + 10 * 1 +20 * 1 =30.

La mesure de cosinus :

$$Sim(d_i, Q) = \frac{\sum_{j=1}^{n} w_{qj} * w_{ij}}{\left(\sum_{j=1}^{n} w_{qj}\right)^{1/2} * \left(\sum_{j=1}^{n} w_{ij}\right)^{1/2}}$$

La mesure de Dice :

$$Sim(d_i, Q) = 2 * \frac{\sum_{j=1}^{n} w_{qj} * w_{ij}}{\left(\sum_{j=1}^{n} w_{qj}^{2}\right) + \left(\sum_{j=1}^{n} w_{ij}^{2}\right)}$$

La mesure de Jacard :

$$Sim(d_i, Q) = \frac{\sum_{j=1}^{n} w_{qj} * w_{ij}}{\left(\sum_{j=1}^{n} w_{qj}^{2}\right) + \left(\sum_{j=1}^{n} w_{ij}^{2}\right) - \sum_{j=1}^{n} w_{qj} * w_{ij}}$$

La mesure de Superposition :

$$Sim(d_i, Q) = \frac{\sum_{j=1}^{n} w_{qj} * w_{ij}}{\min\left(\left(\sum_{j=1}^{n} w_{qj}^{2}\right), \left(\sum_{j=1}^{n} w_{ij}^{2}\right)\right)}$$

Avantages et les désavantages du modèle vectoriel :

Le modèle vectoriel est l'un des modèles de recherche les plus utilisés à cause de sa simplicité conceptuelle et de mise en œuvre, il permet aussi de retourner les documents selon leurs degrés de pertinence ce qui offre la possibilité d'ordonner les documents du plus importants au moins pertinents.

Cependant, il a comme inconvénient de ne permet pas de modéliser les associations entre les termes d'indexation.

Les processus d'amélioration :

Les processus d'amélioration sont des processus optionnel c'est-à-dire que certains SRI les disposent et d'autres non. Ils ont pour objectif d'augmenter les performances des SRI par exemple le SRI qui retourne les documents en ordre de leurs degrés de similitude est le plus performant que le SRI qui n'ordre pas les documents de résultat.

Parmi ces processus : le processus d'ordonnancement et le processus de reformulation de requête. Le premier processus permet d'ordonner les documents en résultat selon leurs degrés de pertinence, le deuxième processus est un mécanisme qui permet de restituer plus de documents en résultat.

5. Reformulation de requête

Dans un SRI la requête initiale de l'utilisateur est souvent insuffisante pour l'extraction des documents pertinents répondants au leur besoin, la reformulation de cette dernière par le SRI permet à l'utilisateur d'avoir des nouvelles requêtes qui peuvent être utiles pour avoir d'autres résultats.

De ce fait plusieurs techniques de reformulation ont été développées pour améliore la performance des SRI :

- La technique de réinjection de pertinence (*relevance feedback*).
- L'expansion automatique de requête en s'appuyant sur des ressources linguistiques telles que le thésaurus[1].
- Combinaison des termes de la requête.

6. Le processus d'ordonnancement

Le processus d'ordonnancement consiste à ordonner la liste de documents retournés par le SRI de manière décroissant selon le degré de pertinence. Ce processus d'amélioration a une grande importance dans le processus de recherche d'information car il offre à l'utilisateur les documents le plus pertinents en premier ce qui implique la rapidité d'accéder à l'information pertinent.

Ce processus est réalisable seulement pour les SRI qui implémentent une fonction d'appariement approché au niveau de modèle de recherche, la fonction d'appariement dans le modèle de recherche vectoriel est approchée.

[1] Un thesaurus est un outil permettant de représenter la proximité ou voisinage sémantique entre termes de la collection.

Ce processus est exécutable après le processus de recherche à l'aide d'un algorithme de tri. Dans ce travail nous allons utiliser l'un des réseaux de tri existants qui le réseau de tri « transposition pair impair ». Dans ce qui suit un panorama de ces réseaux en focalisant le réseau de « transposition pair impair ».

Définition d'un réseau de tri :

Le réseau de tri est un ensemble des comparateurs qui travaillent ensemble pour trier une liste de nombre. Chaque comparateur a deux portes en entrées et deux portes en sorties, il prend deux éléments à la fois puis les compare.

6.1. Tri fusion de BATCHER

Description de l'algorithme :

Cet algorithme utilise deux suites de nombre qui satisfait les deux conditions suivantes :

- Les deux suites doivent de taille de puissance de 2,
- Les deux suites doivent triées.

Soit une liste des nombres nommée « liste », on note TRI (liste) la liste à triée, mais c'est la suite est déjà triée on le note TREE (liste). Aussi on note par
FUSION (liste1, liste2) est un opérateur qui fusionne deux suites de nombres qui sont déjà triées.

Supposons que les deux suites sont de taille de deux à la puissance de ***m***.
Pour $m = 0$ il suffit un seul comparateur.
Pour $m = 1$ on peut utiliser trois comparateurs.
Pour $m = n$ on utilise deux copies de réseau FUSION (m-1). La première copie fusionne les éléments d'indice pair et la seconde fusionne les éléments d'indice impair avec les hypothèses suivantes :

- Copie 1= FUSION (les impaire de liste1), (les impaires de liste2).
- Copie 2= FUSION (les paire de liste 1), (les paires de liste2).
- TREE (I^{er} élément de copie1, Min ($II^{ème}$ de copie1, I^{er} copie2), Max ($II^{ème}$ de copie1, I^{er} de copie 2),…, $n^{ième}$ de copie 2).

Exemple1 :

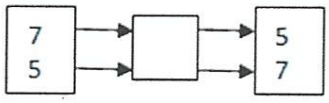

Figure 4 Tri de deux liste de taille 1

Exemple 2 :

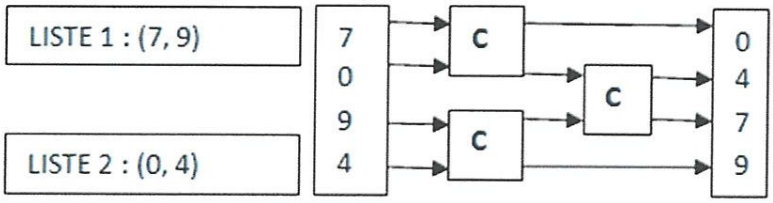

Figure 5 Fusion de BATCHER de deux listes de taille = 2

Exemple 3 :

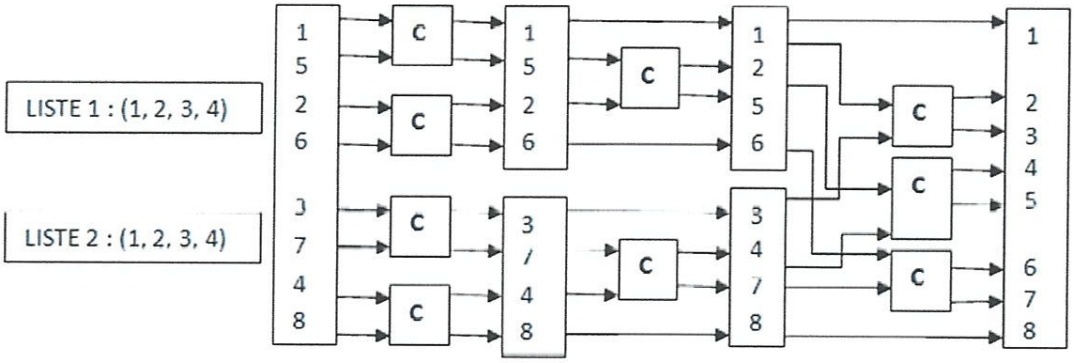

Figure 6 Fusion de BATCHER de deux listes de taille = 4

6.2. Tri par transposition pair impair

Cet algorithme prend en entrée une suite non ordonnée et n'oblige aucune contrainte.

Description de l'algorithme :

Le réseau de cet algorithme est formé d'une succession de lignes de comparateurs. Pour trier une suite de taille *n,* il faut tout d'abord vérifier ce *n* est ce qu'il est pair ou impaire :

S'il est pair alors il faut *n* /2 copies où une copie est formé de deux lignes, la première ligne *n* /2 comparateurs dont chaque comparateur compare entre deux éléments de la suite en commençant par le premier élément. Le seconde contientss (*n* /2 -1) comparateurs dont chaque comparateur compare entre deux éléments de la suite en commençant par le deuxième élément c'est-à-dire il reste le premier et le dernier élément sans comparaison.

S'il est impair alors il faut le réseau est le même que si la taille paire sauf que la deuxième ligne contient $n / 2$ comparateurs.

La figure 7 présente un exemple d'exécution de tri transposition pair impair où la taille est égale à six nombres entiers. Dans ce cas la taille est paire donc il faut trois copies de réseau. « ***C*** » signifie un comparateur qui prend en entrée deux éléments successifs puis faire une permutation si l'élément à gauche est inférieur à l'élément droit.

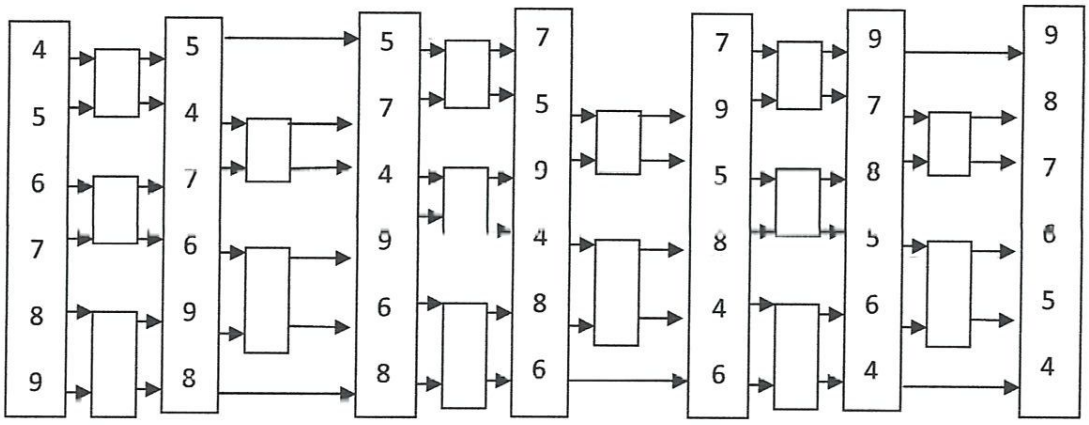

Figure 7 transposition pair impair avec taille= 6

La figure 8 présente un exemple d'exécution de tri transposition pair impair où la taille est égale à cinq nombres. Dans ce cas la taille est impaire donc il faut trois copies et demi copie de réseau.

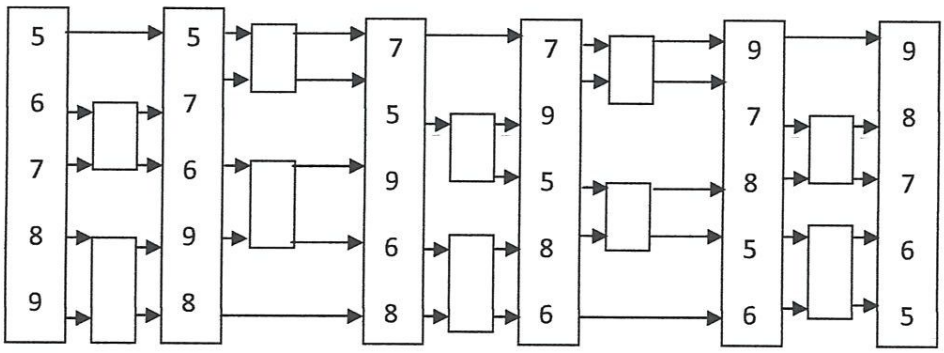

Figure 8 tri transposition pair impair avec taille =5

Le mécanisme de ce réseau est simple et très efficace car il s'appuie à la fois sur la séparation c'est-à-dire il décompose la liste en sous suite de base qui contient deux

éléments qui conduit à la possibilité d'intégrer le parallélisme pour accélérer le traitement où chaque thread sera considérer comme un comparateur qui compare deux éléments de la liste indépendamment d'autres comparateurs comme elle est montré dans la figure . Autre part la fonction de traitement est « la permutation » qui est une fonction simple. Ainsi que l'application de ce réseau est facile.

7. Les mesures d'évaluation

Le principal objectif d'un système de recherche d'information est de restituer à l'utilisateur tous les documents pertinents et de rejeter tous les documents non pertinents (11). Cet objectif est évalué à l'aide de différentes mesures d'évaluations, parmi ces mesures :

- **La précision :** C'est le rapport du nombre de documents pertinents restitués par le système (*DP*) sur le nombre total de documents restitués (*D*), exprimée ainsi :

$$précision = \frac{\mathrm{DP}}{\mathrm{D}}$$

- **Le rappel :** C'est le rapport du nombre de documents pertinents restitués (*DP*) sur le nombre total de documents pertinents (*p*) exprimée ainsi :

$$rappel = \frac{\mathrm{DP}}{\mathrm{P}}$$

- **Le bruit :** C'est le nombre de documents non pertinents restitués, elle définit par :

$$\mathrm{B} = 1 - p.$$

- **Le silence :** C'est le nombre de documents pertinents non restitués, elle définit par :

$$S = 1 - \mathrm{R}$$

Un système idéal devrait retourner tous les documents pertinents et rien que les documents pertinents, c'est-à-dire un taux de précision et de rappel égale à 100%.

8. Conclusion

Dans ce chapitre nous avons passé en revue les principaux concepts de la recherche d'information, nous avons aussi cité quelques réseaux de tri nécessaire à notre approche. Dans le chapitre suivant nous aborderons les concepts de parallélisme sur lesquels se base notre système.

Chapitre 2 :

le parallélisme

Chapitre 2 : le parallélisme

1. Introduction

Le parallélisme en informatique est une technique qui consiste à mettre en commun plusieurs ressources de calculs pour réaliser de multiples traitements de façon simultanée afin de traiter plus rapidement les problèmes plus grands.

Dans ce chapitre nous allons définir plusieurs concepts de base utilisés da le parallélisme. Nous allons essayer de répondre à la question : « pourquoi le parallélisme ? », nous allons voir aussi les différentes architectures parallèles, les types de parallélisme et les modèles de programmation parallèles.

2. Les concepts de base de parallélisme

- ***Tâche :*** c'est une portion de travail à exécuter sur un ordinateur, du type un ensemble d'instructions d'un programme qui est exécuter sur un processeur.
- ***Tâche parallèle :*** c'est une tâche qui peut s'exécuter sur plusieurs processeurs, sans risque sur la validité des résultats.
- ***Exécution séquentiel :*** c'est l'exécution d'un programme séquentiel, une instruction à la fois.
- ***Exécution parallèle :*** c'est exécution d'un programme par plusieurs tâches, chaque tâche pouvant exécuter la même instruction ou une instruction différente.
- ***Ordinateur parallèle :*** un ordinateur parallèle est une machine composée de plusieurs processeurs qui coopèrent entres eux afin de résoudre un problème.
- ***Mémoire partagée :*** d'un point de vue générale, on parle d'une machine dont tous les processeurs ont un accès direct à une mémoire commune. Par contre d'un point de vue modèle de programmation : toutes les tâches ont la même image mémoire et peuvent directement adresser et accéder au même emplacement mémoire logique (figure 9).

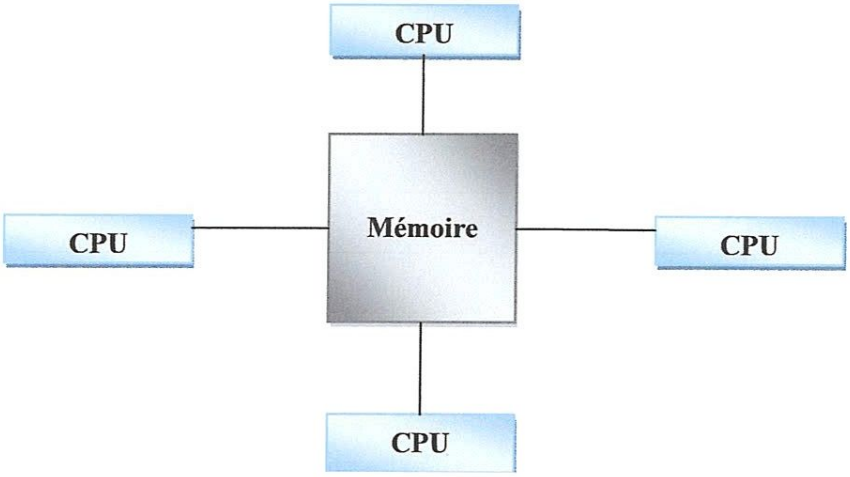

Figure 9 Machine à mémoire partagé

- ***Mémoire distribuée :*** d'un point de vue physique, elle est basée sur un accès mémoire réseau pour une mémoire physique non commine. D'un point de vue modèle de programmation, les tâches ne peuvent voir que la mémoire de la machine locale et doivent effectuer des communications pour accéder à la mémoire d'une machine distante, sur laquelle d'autres tâches s'exécutent (la figure 10).

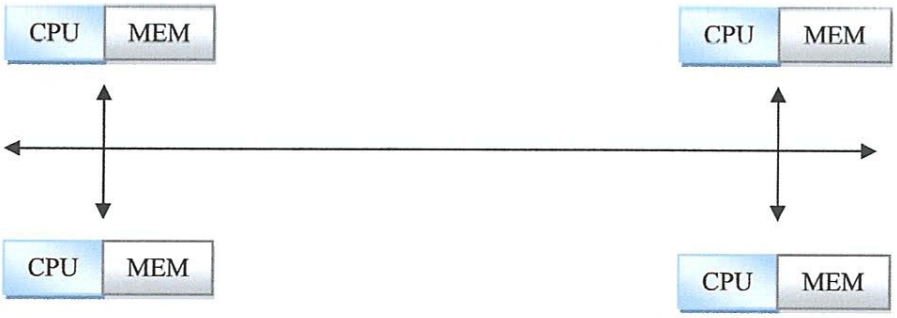

Figure 10 Machine à mémoire distribué

- ***Mémoire hybride :*** ce type de modèle mixe entre les deux types précédents mémoire partagé et mémoire distribué (la figure 11).

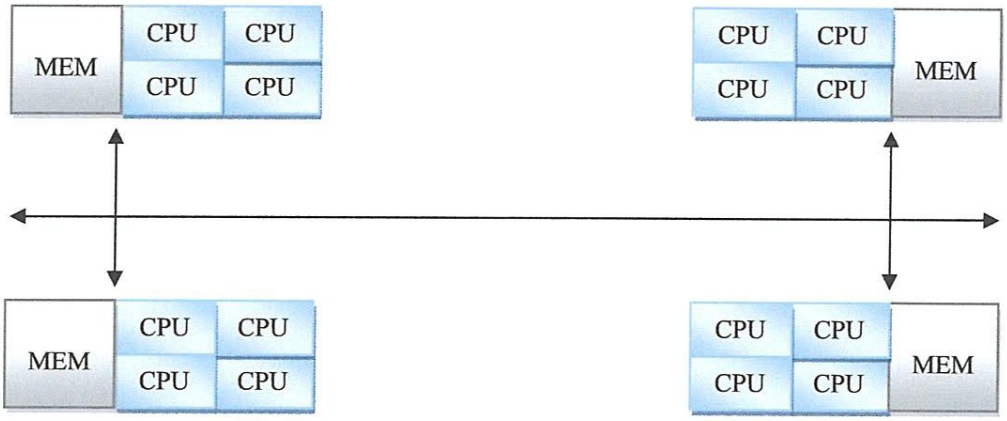

Figure 11 Machine à mémoire hybride

- ***Système distribué :*** un système distribué (ou répartie) est un système composé de plusieurs processeurs impliqués dans la résolution d'un ou plusieurs problèmes.
- ***Communication :*** les tâches parallèles échangent des données entres eux par différents moyens physiques : via un bus à mémoire partagée ou via un réseau. quel que soit la méthode employée, on parle de « communication ».
- ***Synchronisation :*** c'est la coordination des tâches en temps réel est souvent associée aux communications, elle est implémentée en introduisant un point de synchronisation au-delà duquel la tâche ne peut continuer tant qu'une ou plusieurs autres tâches ne l'ont pas atteint.
- ***Scalabilité :*** elle réfère à la capacité d'un système parallèle à fournir une augmentation de l'accélération proportionnelle à l'augmentation du nombre de processeurs.
- ***L'accélération :*** considérons un algorithme qui s'exécute sur un ordinateur parallèle comportant P processeurs (identiques) en un temps T_P, et soit T_1 son temps d'exécution séquentiel. L'accélération S_p est définie comme suit :

$$S_P = \frac{T_1}{T_P}$$

Généralement on a : $1 < S_p < p$.

- ***l'efficacité :*** l'efficacité *Ep* d'un algorithme parallèle est le rapport :

$$E_P = \frac{S_1}{P}$$

3. Pourquoi le parallélisme ?

Le parallélisme est considéré comme un élément de base pour traiter les problèmes dans plusieurs domaines scientifiques constituant de grands challenges (problèmes plus grand et/ ou complexes), tels que : la météo, la biologie, la géophysique (activité sismique). Cependant, aujourd'hui on trouve le parallélisme dans des applications commerciales : bases de données parallèles, réalité virtuelle et touche aussi tous les domaines afin d'exploiter au maximum les architectures actuelles.

4. Historique d'évolution du parallélisme

Le travail de Clet-Ortega Jérôme [38] nous permet de résumer l'histoire de l'évolution des systèmes parallèles et aussi l'évolution technologique au sein d'unité de calcul (processeur).

4.1. L'évolution des systèmes parallèles

La chronologie de l'évolution du matériel de calcul hautes-performances, commence par la domination des super calculateurs depuis le milieu des années 60 jusqu'aux années 90. Des premiers super-calculateurs étaient dotés d'un processeur central assisté par des processeurs périphériques destinés à réaliser les opérations d'entrées/ sorties en parallèle.

Au cours des années 70, les constructeurs ont intégré des processeurs vectoriels capables d'appliquer une même instruction à un ensemble de données, favorisant ainsi les calculs s'appliquant à des tableaux de nombres. Afin d'accroitre les performances des super-calculateurs, plusieurs de ces processeurs vectoriel sont associés pour fonctionner en parallèle.

A l'approche des années 90, ces unités de calcul vectorielles ont laissé la place à des processeurs scalaires plus simples. Ce type d'architecture est souvent désigné sous le terme de système massivement parallèle, le nombre de processeurs peuvent atteindre

plusieurs centaines dans un premier temps, puis plus récemment plusieurs centaines de milliers. Ce type est moins coûteux.

En marge des progrès réalisés dans les super-calculateurs, les micro-ordinateurs ont eux aussi connu de profondes mutations. L'ensemble de ces stations de travail s'est avéré être une alternative intéressante pour les centres ne disposant pas de moyens financiers suffisants. En effet, en réunissant la puissance de calcul de chacune des machines grâce à un réseau d'interconnexion hautes-performances, cet ensemble est nommée les grappes de calcul.

L'interconnexion de plusieurs grappes de calcul est apparue comme une conséquence naturelle du succès des grappes de calcul. L'idée s'inspire directement de cet assemblage d'ordinateurs standard pour créer une sorte de *méta-grappe* où les nœuds sont des grappes, reliées entre elles par un réseau à haut débit. Ces grappes peuvent faire partie d'un même laboratoire de recherche ou bien de différents laboratoires à l'échelle nationale ou internationale.

Suite à la propagation des grappes de calcul il apparut les grilles. Cette approche consiste à relier plusieurs ressources de calcul, potentiellement distante de plusieurs centaines de kilomètres afin de disposer d'un grand système de calcul réparti. Les éléments reliés sont relativement variés : machine séquentiel, super-calculateurs, grappes de calcul ou baies de stockage.

4.2. Evolution au sein de processeur

Depuis le milieu des années 80, les constructeurs font appel à une technique très simple et directement inspirée des chaînes de montage : *le pipeline*. Le principe de cette technique est de décomposer le chemin du traitement d'une instruction en plusieurs étapes : chargement, décodage, exécution, …etc. chacune de ces étapes est réalisée par un circuit spécifique et indépendant des autres. Ainsi il n'est pas nécessaire d'attendre l'accomplissement d'une instruction pour commencer la suivante alors un seul flot d'exécution se divise en plusieurs instructions réparties sur différentes unités fonctionnelles. La figure12 donne une représentation schématique d'un pipeline à trois étages où cinq instructions peuvent s'exécuter de façon concurrente.

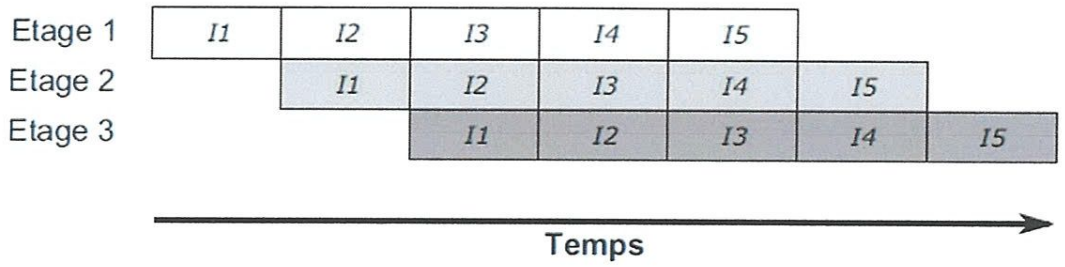

Figure 12 Progression de cinq instructions dans le pipeline à trois étages [38].

Les constructeurs réduisent au maximum la taille des composants afin d'en rajouter autant que possible sur l'espace disponible. Ils rendent les pipelines plus complexes en intégrant des étages supplémentaires pour maximiser le parallélisme. Cependant un obstacle inévitable a stoppé la progression vers des fréquences plus élevées : la barrière thermique. Les composants ne pouvaient pas supporter de fonctionner à des températures trop élevées sans dommages physiques. Une solution envisagée a été de permettre à plusieurs threads matériels [1] d'accéder aux unités fonctionnelles de façon entrelacée, ce qui implique de gérer efficacement plusieurs contextes de threads en parallèle. Ainsi, il existe deux types de multithreading au sein de processeurs : le *Fine-grained multithreading* (FMT) et le *Coarse-grained Multithreading* (CMT). Dans le premier cas, à chaque cycle l'état du thread courant est sauvegardé puis l'on change de thread, pourvu qu'il y en ait un de prêt. Le CMT réalise des changements de contexte uniquement en réponse à certains événements comme les accès à une mémoire distante dans un système à mémoire partagé ou suite à un défaut de cache.

En 1995, l'équipe de H. Levy propose la technique de *Simultaneous multithreading* [39] (SMT). L'idée est de permettre l'exécution simultanée de plusieurs flots d'instructions indépendants sur un même pipeline la plupart des ressources du processeur (mémoire cache, unité de traitement,...etc.) sont partagées, seul un ensemble de registres est dupliqué et le processeur est capable de récupérer des instructions provenant de plusieurs threads en un cycle.

D'autre part, certains constructeurs ont pensé à dupliquer les unités de calcul. Ainsi disposées sur une même puce, plusieurs unités de calcul surnommées cœurs se partagent l'accès à une ou plusieurs mémoires caches et au bus de données. Le

[1] Un thread matériel correspond à un ensemble de circuits constitué principalement par un jeu de registre de données et de contrôle ainsi que d'un contrôleur d'interruptions permettant de gérer le contexte d'un flot d'exécution

principe de construction de ces processeurs dits *multicoeurs.* La figure 13 présente la structure de processeur à quatre cœurs où chaque cœur du processeur dispose de son propre cache de niveau 1 et le cache et le cache de niveau 2 est commune aux quatre cœurs.

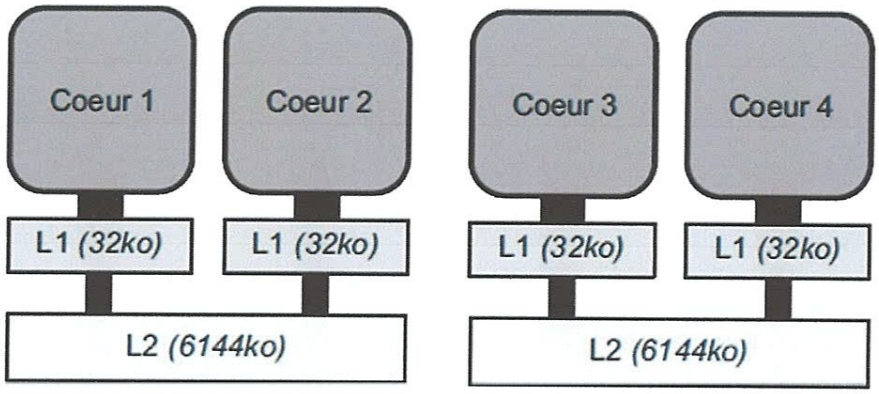

Figure 13 Architecture de processeur à quatre cœurs [38].

En marge des progrès à l'intérieur de même processeur, les constructeurs ont également fait évoluer l'architecture des machines de calcul. Ils ont eu l'idée de conjuguer la puissance de plusieurs processeurs en les interconnectant au sein d'une même machine. Ces architectures baptisees *multiprocesseurs* se scindent en deux grandes familles : *les processeurs symétriques* (SMP) *et les multiprocesseurs à accès mémoire non-uniformes* (NUMA).

Un système multiprocesseur symétrique se construit d'un ou plusieurs monoprocesseurs qui sont connectés au bus. Tous ces processeurs accèdent à la mémoire commune par ce bus et de façon uniforme c'est-à-dire le temps d'accès est le même pour tous. Un exemple d'architecture SMP est visible en figure 14 où 4 processeurs sont reliés au bus.

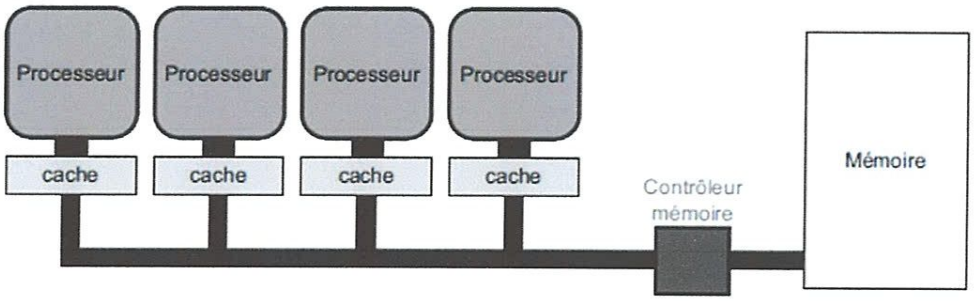

Figure 14 Architecture SMP à quatre processeurs [38]

L'architecture NUMA est un ensemble de nœuds où chaque nœud NUMA peut être assimilé à une architecture SMP dans laquelle chaque processeur accède non uniformément au banc mémoire. Tous les NUMA sont reliés entre eux par un réseau d'interconnexion comme il est montré dans la figure 15

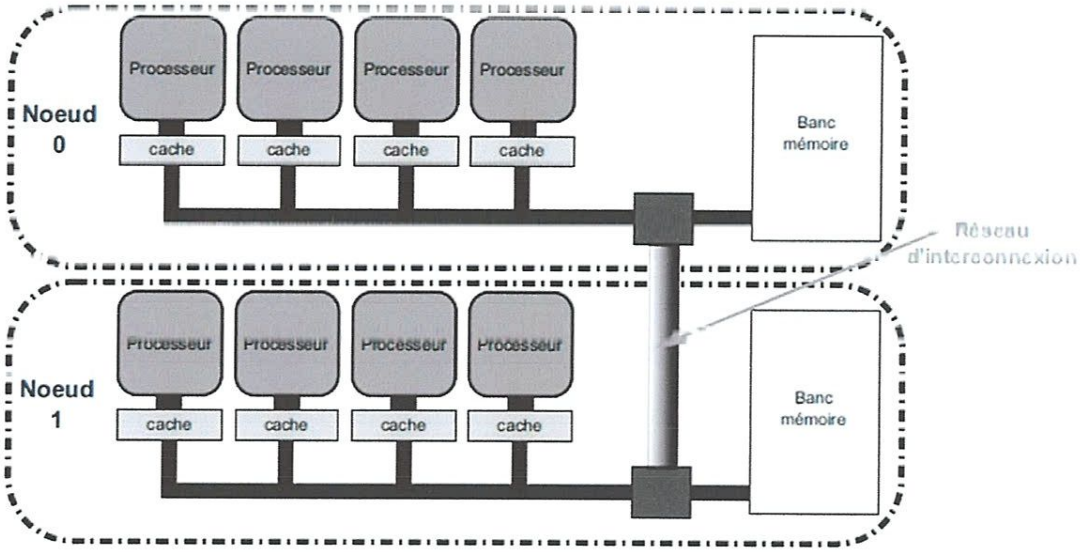

Figure 15 Architecture NUMA à deux nœuds [38]

5. Les architectures parallèles

Pour rendre possible l'exécution des programmes parallèles, des architecture de machines ont été développées. En 1972, MICHAEL J.FLYNN définit une classification des architectures des ordinateurs [14], basée sur :

- La nature de flux d'instructions exécutés par le processeur.
- La nature de flux de données sur lesquels opèrent les instructions.

	Flux d'instruction unique	Flux d'instruction multiple
Flux de donnée unique	**SISD (séquentiel)**	**MISD**
Flux de donnée multiple	**SIMD**	**MIMD**

Tableau 1 Taxinomie de Flynn.

- ***Machine SISD* (Single Instruction Single Data) :**

Dans cette machine, une seule instruction est exécutée et une seule donnée est traitée à tout instant. Les traitements dans ce type de machine sont donc séquentiels (sans parallélisme). Ce modèle correspond à une machine monoprocesseur.

- ***Machine SIMD* (Single Instruction Multiple Data):**

C'est une machine dans laquelle tous les processeurs sont synchrones. Un seul processeur exécute le programme et envoie la même instruction à tous les processeurs de calcul. Le SIMD sont développés au cours des années 80-90. Leur principal avantage a été leur « simplicité ».

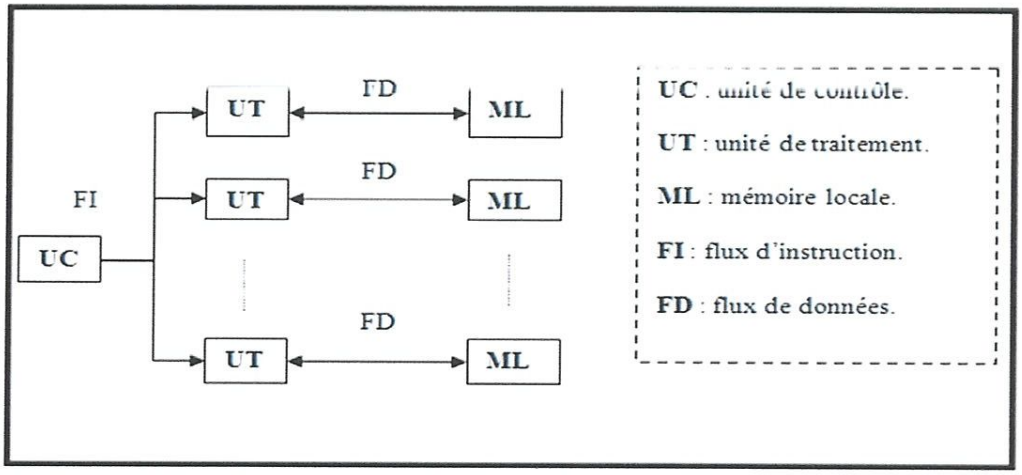

Figure 16 Machine SIMD

- ***Machine MISD* (Multiple Instruction Single Data) :**

C'est une machine dans laquelle chaque processeur exécute une séquence différente d'instructions sur les mêmes données.

- ***Machine MIMD* (Multiple Instruction Single Data) :**

C'est une machine dans laquelle tous les processeurs sont autonomes (chaque processeur traite une donnée distincte et lui applique son propre flux d'instructions). Les MIMD sont développés au début des années 90.

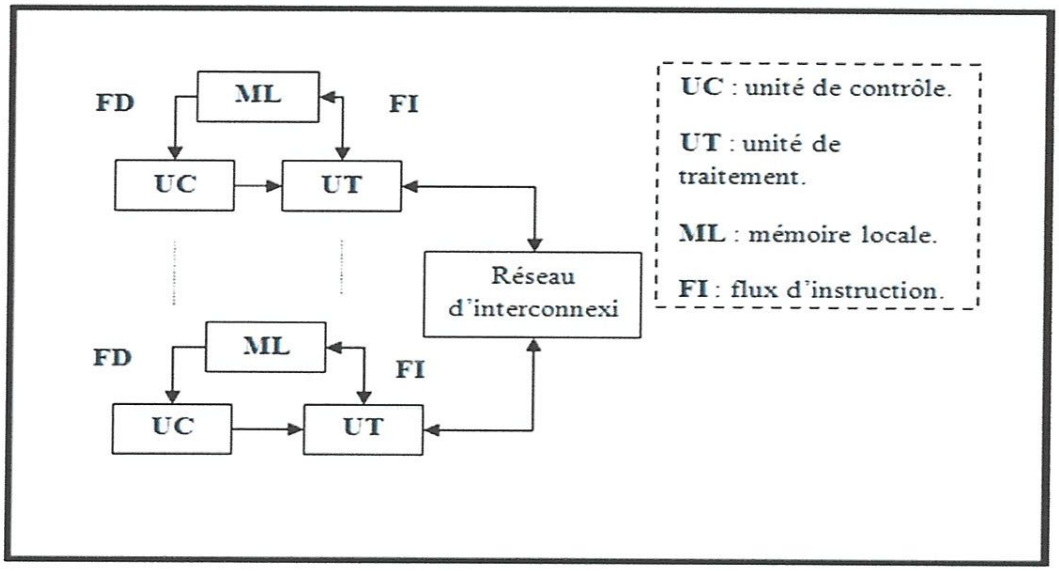

Figure 17 Machine MIMD

6. Les types de parallélisme

6.1. Parallélisme des tâches

Ce type de parallélisme cherche à diviser un programme en un ensemble de tâches qui peuvent être dépendantes mai aussi indépendantes, dans ce cas c'est l'exécution de programme qui cherche à être parallélisée [41].

Les architectures MISD et MIMD peuvent être associés au parallélisme de tâches. Dans le premier cas, des opérations successives sont appliqués sur un jeu de données d'entrée. Dans l'architecture MIMD quant à lui est rencontré plus fréquemment et offre plus de possibilité de parallélisme, dans ce cas on cherchera à identifier des tâches travaillant sur des données différentes ce qui rend les tâches indépendantes les unes des autres. Toutefois, le développeur devra se charger de synchroniser les différentes tâches ensemble afin de garantir la cohérence des résultats.

6.2. Parallélisme des données

Ce type de parallélisme se focalise sur la façon dont les données sont distribuées sur les différents processeurs [41]. L'ensemble des processeurs effectuent alors le même jeu d'instructions sur des données d'entrée qui leurs sont propres. Dans ce cas les

tâches effectuées par le programme sont peu modifiées. Il faut toutefois réfléchir et concevoir les communications, les échanges ou les synchronisations nécessaires entre les processeurs pour que le programme parallèle soit correcte et donne même résultat qu'en séquentiel. L'architecture SIMD est associée au parallélisme de données.

7. Les modèles de programmation parallèles

Un modèle de programmation parallèle est un modèle qui permet d'exprimer la façon dont une application parallèle sera programmée. Parmi ces modèles : le modèle de programmation parallèle à mémoire partagée et le modèle de programmation parallèle à mémoire distribuée.

7.1. Le modèle à mémoire distribuée

Dans ce modèle, chaque processeur a sa propre mémoire locale, il n'y a pas de notion d'espace d'adressage globale entre tous le processeurs. La communication entre les mémoires des différents processeurs se fait à l'aide d'un réseau d'interconnexion.

Les processeurs opèrent indépendamment les uns des autres c'est-à-dire qu'une modification en mémoire locale n'a pas d'effet sur la mémoire des autres processeurs. Si un processeur a besoin d'une donnée dans la mémoire d'un autre processeur, le programmeur doit définir explicitement la communication.

Les réseaux d'interconnexion sont divers, avec des niveaux de performances très variables. La figure 18 présente le mécanisme d'envoie et réception de données entre deux processeurs dans modèle à mémoire distribuée, cette communication passe par : l'empaquète des données, déplacement de paquet, puis la réception de paquet, le dépaquète des données et enfin l'utilisation des données.

Chapitre 3 :

Conception et implémentation

Chapitre 3 : Conception et implémentation

1. Introduction

Dans ce chapitre nous allons construire deux systèmes de recherche d'information; l'un parallèle qui est l'objectif de cette étude, et l'autre séquentiel qui va servir d'outil de comparaison afin de mieux voir les bénéfiques d'utiliser un système de recherche d'information parallèle. Pour ce faire, nous allons faire une comparaison entre les deux systèmes en termes de temps de réponse dans de multiples conditions pour prouver l'efficacité de parallélisme.

2. L'architecture du système de recherche

Le système de recherche d'information prend au début une requête utilisateur ainsi qu'une collection de documents textuels numériques rédigés dans la langue anglaise. Le système supporte plusieurs types de documents textuels : txt, docx, pdf.

Ensuite on trouve l'étape d'indexation, cette étape contient quatre sous étapes suivantes :

- **L'analyse lexicale :** Dans cette étape on va transformer un document textuel en un ensemble de termes « *lexème* ». en d'autres termes, on va éliminer : la ponctuation, la casse et la mise en page.
- **La normalisation (la sélection) :** La deuxième sous étape est la normalisation qui a pour objectif d'éliminer les mots vides de sens tel que : before, after, however,...etc. pour ce faire, on utilise un anti-dictionnaire qui contient une liste de mots vides de sens tel que les articles, prénoms, prépositions, ainsi que les mots athématiques. Après cette étape le document contiendra un nombre réduit de termes susceptibles d'être significatifs pour la requête utilisateur.
- **L'extraction des radicaux :** L'étape de radicalisation permet de transformer les termes en leurs radicale. Pour satisfaire cette étape il existe plusieurs algorithmes de radicalisation, nous avons choisir l'algorithme de PORTER car il est destiné principalement à la langue anglaise et il peut évoluer pour qu'il puisse s'adapter à d'autres langues comme la langue française. Cet algorithme prend en entrée un terme, puis le passe séquentiellement sur cinq étapes,

chaque étape contient un ensemble de règles de type « *si condition alors action* » ; si un terme vérifie une condition bien définit, il existe une modification qui sera appliqué à ce terme (cet algorithme est détaillé dans le chapitre 1). Le rôle de cette étape est d'accroitre l'efficacité du système, il s'agit de réunir plusieurs variantes d'un même terme sous une seule et unique forme. Par exemple si une requête contient le terme « connexion », et un document de la collection ne contient pas ce terme mais il contient ces variantes tel que « connecte, connected ou encore connecting,… » sans l'étape de radicalisation le système ne considèrera pas ce document comme pertinent car le terme exacte n'existe pas. Cependant avec la radicalisation le système vérifie directement si le radicale du terme « connexion » existe ou pas dans les documents de la collection.

- **La pondération :** Dans cette étape on a associé à chaque terme un poids w_{ij} qui signifie l'importance de ce terme dans le document de la collection. Cette pondération est calculée par la formule suivante :

$$w_{ij} = tf_{ij} \ / df_j = tf_{ij} \ * 1/ df_{ij} = tf_{ij} * idf_j$$

Où :

- tf_{ij} est la fréquence d'occurrences du terme tj dans le document d_i.
- df_j est la fréquence documentaire du terme tj (i.e. la proportion de documents de la collection qui contiennent tj) et idf_j sa fréquence documentaire inverse.

Après l'étape d'indexation (requête et documents) vient l'étape de comparaison (la figure 19) où le système calcule le degré de similitude entre la requête et chaque document de la collection. Dans ce travail nous avons opté pour le modèle vectoriel car c'est une structure régulière bien adapté à l'informatique. Ce modèle propose plusieurs formules pour calculer le degré de similitude entre la requête et un document. Dans ce cadre, nous avons choisir la formule du produit scalaire car cette dernière est la base des autres formules. Elle est définit comme suit :

$$Sim(d_i, q) = \sum_{j=1}^{n} (w_{qj} * w_{ij})$$

Où :

- d_i : est un document.
- Q : est une requête.
- w_{ij} : est le poids du terme t_j dans le document d_i.
- w_{qj} : est le poids du terme t_j dans la requête q.

L'indexation et la recherche sont considérées comme les principales étapes d'un système de recherche d'information. Dans ce travail nous avons ajouté le processus d'ordonnancement qui est un processus très important pour un système de recherche et très bénéfique à l'utilisateur car il accélère l'accès à l'information pertinente. Pour ce faire, il existe les réseaux de tri qui sont très simple, très efficace et plus adaptable au parallélisme. Dans ce travail, nous avons choisi le réseau de tri « *transposition pair impaire* » car ce réseau travaille sur n'importe quelle liste de nombre sans contrainte.

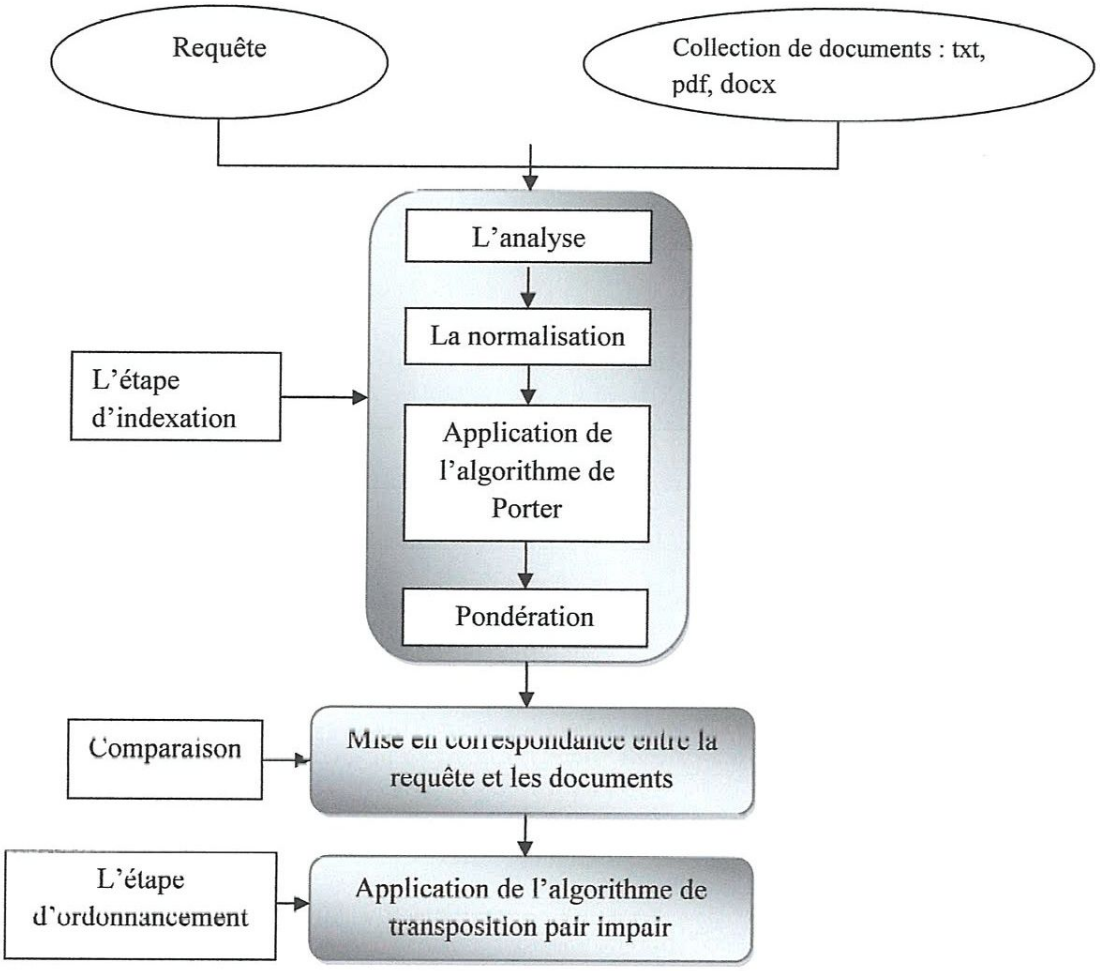

Figure 19 Architecture du système de recherche d'information

3. Principe de fonctionnement du système séquentiel

Dans l'exécution séquentiel une seule tâche sera exécuté à la fois c'est-à-dire le système charge un seul document à la fois afin de l'indexer. Comme on la vue précédemment, l'indexation d'un document ou d'une requête passe par plusieurs étapes ; la première est l'analyse lexicale qui va éliminer toute la ponctuation, la mise en forme et la casse. Ensuite la normalisation où le système prend chaque mot du document ou de la requête et il le compare avec tous les mots de *stoplist (anti-dictionnaire),* si le mot existe dans le *stoplist* alors le système le supprime sinon le garde.

Après l'étape de la normalisation le document contient seulement les mots qui ne sont pas vides de sens, le système prend alors un mot à la fois et applique sur lui l'algorithme de PORTER qui a pour objectif l'extraction du radical, puis le système remplace le mot d'origine par son radical.

Enfin, l'étape de la pondération où le système calcule deux paramètres essentiels pour le calcul du poids du terme : le nombre d'occurrence locale du terme c'est à dire combien de fois ce terme est apparu dans le document en question, ainsi que le nombre le nombre d'occurrence globale c'est à dire combien de fois ce terme est apparu dans toute la collection. Dans cette phase le système indexe un seul document à la fois et il ne passe pas à l'étape de comparaison sauf s'il indexe toute la collection.

Dans la phase de comparaison le système applique la formule de produit scalaire. Supposons que la requête est composée de deux termes t_1 et t_2 avec $w_{q1} = 1$, $w_{q2} = 1$ et un document contient t_1 et d'autres termes et ne contient pas t_2 avec $w_{q1} = 20$ le système fait ce calcule 20*1+1*0=20 et 20 sera le degré de pertinence de ce document à cette requête. Dans cette phase le système faire cette comparaison pour un seul document à la fois et il ne passe pas à l'étape d'ordonnancement sauf s'il compare tous les documents avec la requête d'utilisateur.

A la fin le système retourne une liste de documents avec leurs degrés de similitude non ordonné il applique sur cette liste l'algorithme de transposition pair impaire. Le fonctionnement de ce système est résumé dans la figure suivante.

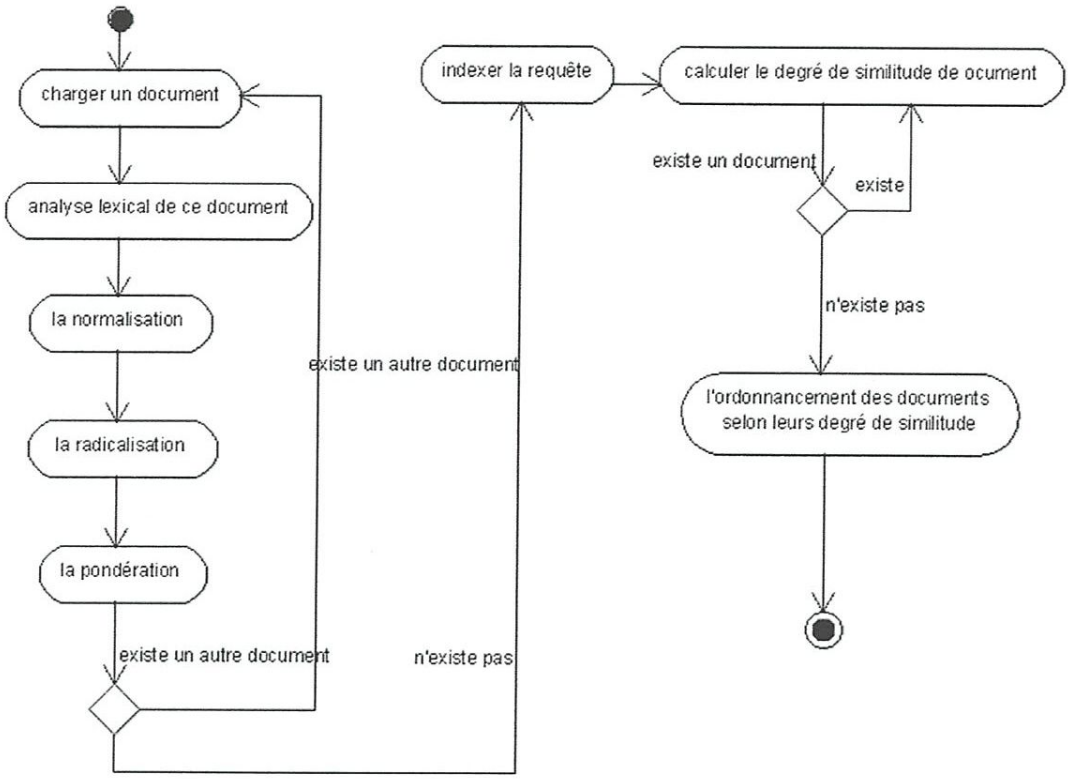

Figure 20 Diagramme d'activité de système séquentiel

Nous pouvons conclure que le système exécute les étapes : indexation, recherche, ordonnancement de manière séquentiel, il indexe un seul documcnt dc la collection à la fois, il compare à la fois un document avec la requête utilisateur. Dans la section suivante nous avons expliqué comment nous avons intégré le parallélisme dans les deux étapes : l'indexation et la comparaison.

4. Principe du fonctionnement du système parallèle

Le système parallèle a pour objectif de retourner les documents pertinents par apport à une requête donnée dans un temps inférieure que celui du système séquentiel.

Comme nous avons défini le parallélisme dans le deuxième chapitre par : « le parallélisme en informatique est une technique qui consiste à mettre en commun plusieurs ressources de calculs pour réaliser de multiples traitement de façon simultanée afin de traiter plus rapidement des problèmes plus grands », nous avons construit un deuxième système en s'appuyant sur le modèle de programmation parallèle à mémoire partagé (vu dans le deuxième chapitre), une architecture de machine SIMD car nous avons les mêmes instructions (étapes) sur des données différentes (documents) et une machine parallèle quadruple cœurs pour une exécution

parallèle. Nous avons introduit le parallélisme dans les parties principales qui composent le système de recherche d'information c'est-à-dire l'indexation et la comparaison.

5. Intégration du parallélisme

A. L'indexation

L'indexation comporte un nombre fini d'étapes (l'analyse lexicale,…) sur la requête utilisateur et chaque document de la collection, donc à la place de lancer un seul thread à la fois (séquentiel) nous avons construit un système d'indexation parallèle de tel manière qu'on puisse lancer un nombre maximum de threads où chaque thread traite un document et tous les threads travaillent en même temps. La figure suivante illustre l'exécution parallèle de l'indexation dans le cas de deux threads.

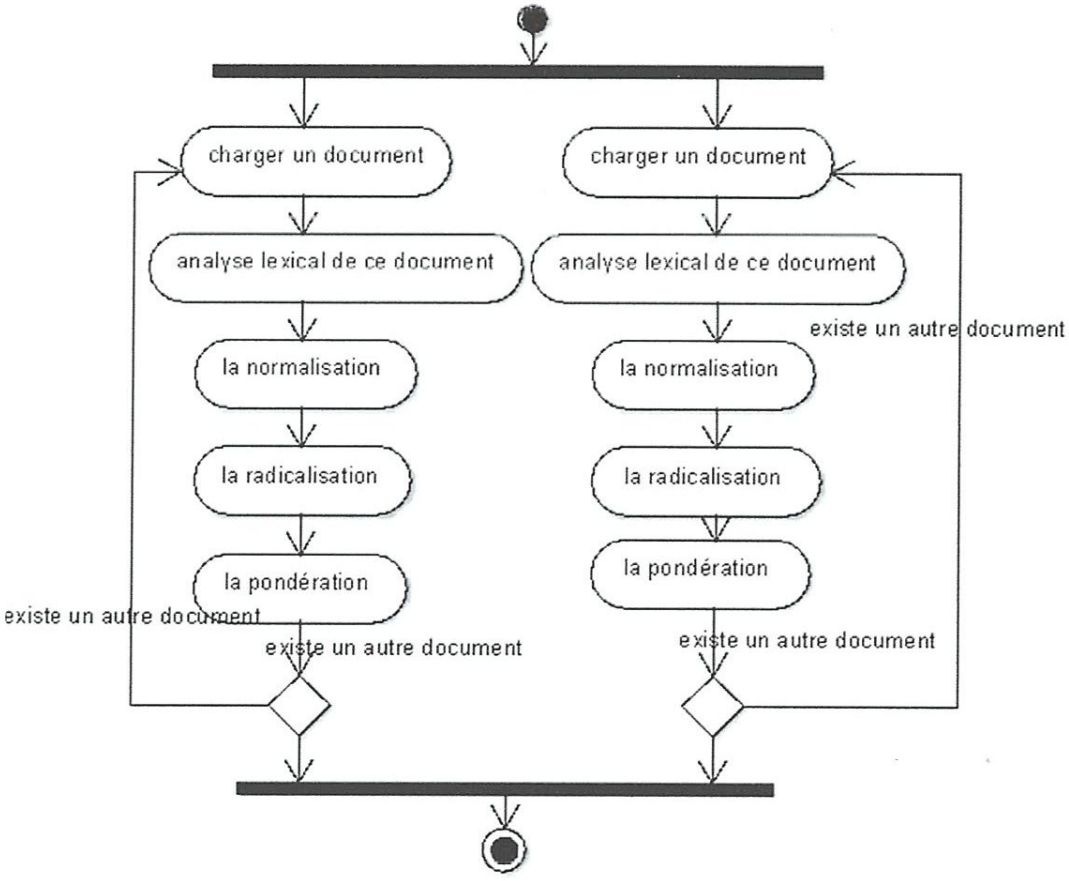

Figure 21 L'exécution parallèle de l'indexation

B. La comparaison :

Etant donné quand nous avons utilisé le modèle vectoriel, nous avons utilisé la formule de produit scalaire afin de calculer de degré de similitude entre un document et la requête, ce calcul se fait pour chaque document de la collection. Dans le cas d'une grande collection de documents les calculs pourrait coûter très cher en termes de temps, d'où l'idée de parallélisme qui est de lancer plusieurs threads simultanément où chaque thread calcul le degré de similitude d'un document et tous les threads travaille en même temps. La figure suivante illustre l'exécution parallèle

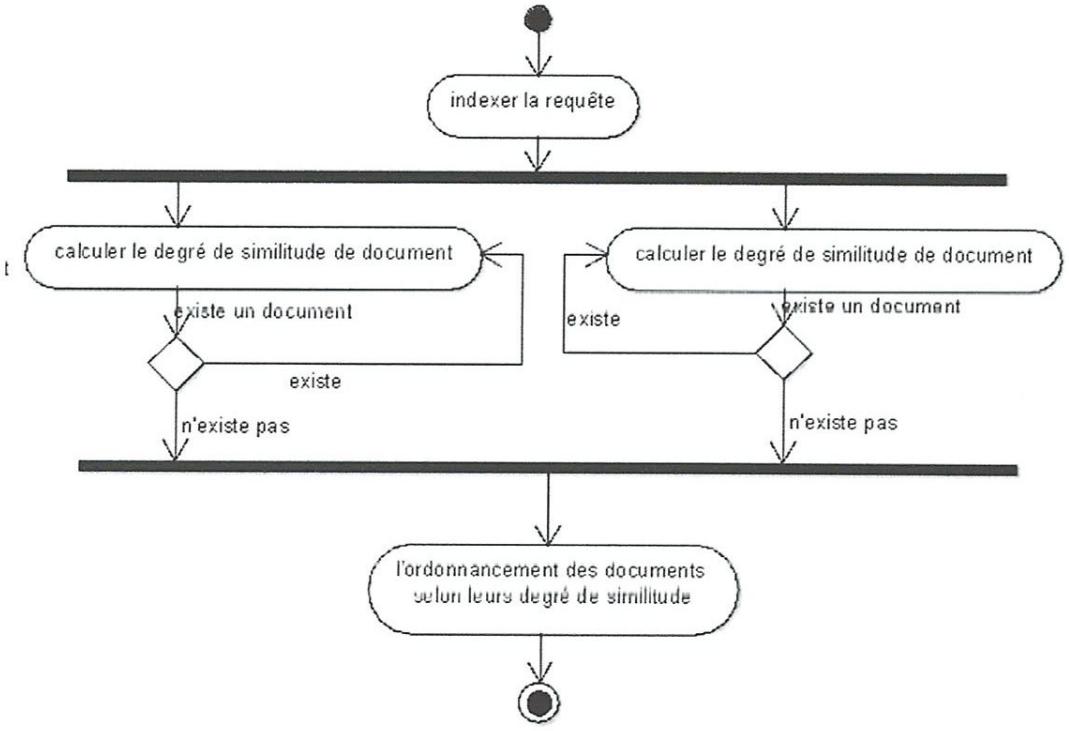

Figure 22 L'exécution parallèle de la partie comparaison.

6. Conception d'application

Nous avons construit deux systèmes de recherche d'information ; un séquentiel et l'autre parallèle. Les deux systèmes sont gérés par une seule et même application (la figure 23). Cette application va nous permettre d'effectuer plusieurs tests sur les deux systèmes, ce qui va nous permettre d'effectuer une comparaison entre les deux systèmes en termes de temps de réponse. La figure 5 montre les différentes fonctionnalités du système résumé dans un diagramme.

Cette application est réalisée avec le langage de programmation *JAVA* version 1.8.0 sous l'environnement *eclipse*. Le nombre de cœurs de processeur égale à 4, avec 8 GO de RAM.

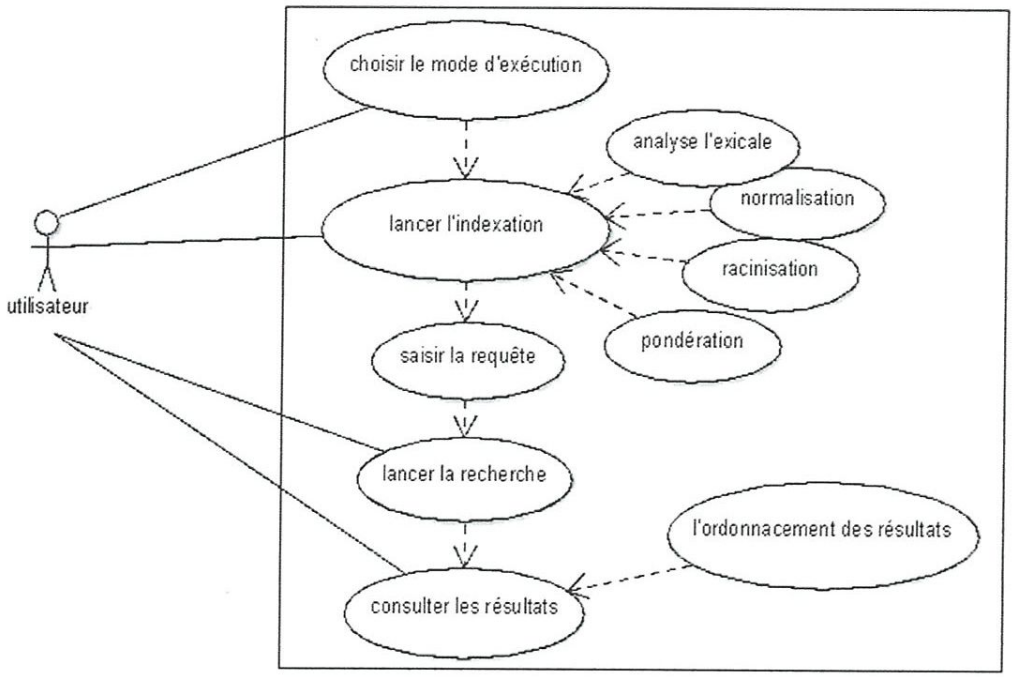

Figure 23 Diagramme de cas d'utilisation

Figure 24 L'interface principale

La figure 25 et la figure 26 montrent les résultats de la recherche pour les deux systèmes parallèle et séquentiel après les étapes ; d'indexation, de comparaison et d'ordonnancement. Pour chaque système on a le temps d'indexation, le temps de recherche, le temps d'ordonnancement ainsi que le temps total et des documents ordonnés selon leur degré de pertinence.

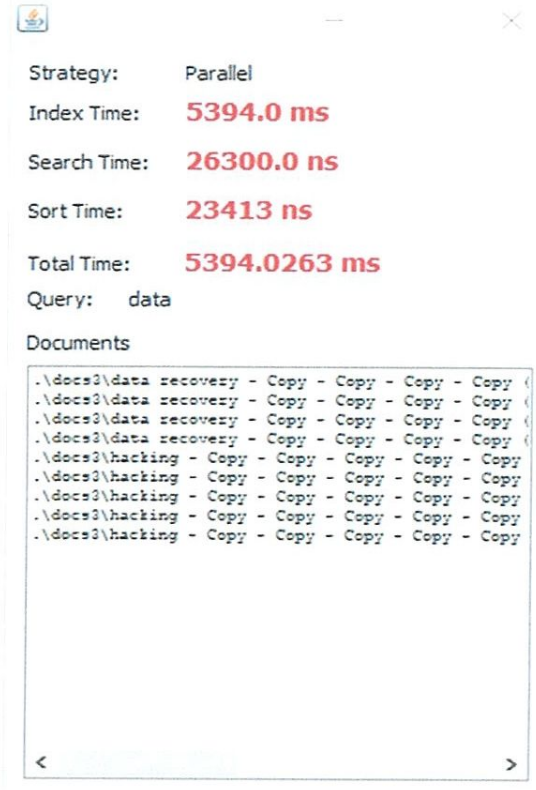

Figure 25 La fenêtre SHOW parallèle

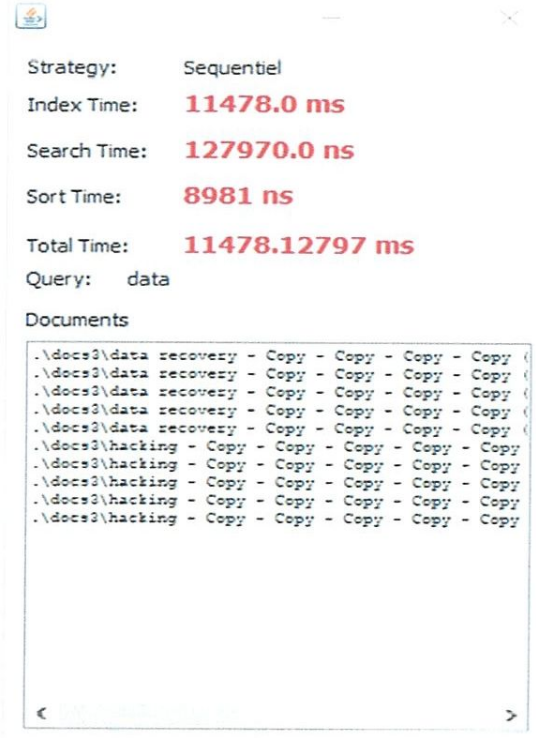

Figure 26 La fenêtre SHOW séquentiel

7. Comparaison entre le système parallèle et le système séquentiel

Pour faire la comparaison entre les deux systèmes nous avons fait un ensemble de tests sur un nombre de documents qui varie de 10 à 500 documents. Le temps d'exécution que se soit pour le système parallèle ou séquentiel dépend en partie de l'état actuel de la machine. Pour ce faire, nous avons effectué pour chaque cas trois tests successifs, ensuite nous avons prend la moyenne de ces tests pour chaque cas (tableau 5 et 6 dans l'annexe).

Dans un premier temps on trouve les résultats concernant la partie indexation (tableau1). Ces résultats sont présentés par un diagramme en barres et un graphique (figure10) en courbe qui présente le gain de temps marqué durant cette étape.

Nombre de fichiers	10	20	30	40	50	100	500
Temps d'indexation parallèle (ms)	4429	8126,333	12130,67	15593,67	19316,33	37434,67	186326,7
Temps d'indexation séquentiel (ms)	11298,33	22224	33399,67	44535,67	55866,33	111084,3	581850
R1=TS/TP	2,55	2,73	2,75	2,86	2,89	2,97	3,12

Tableau 3 Variation du temps d'indexation

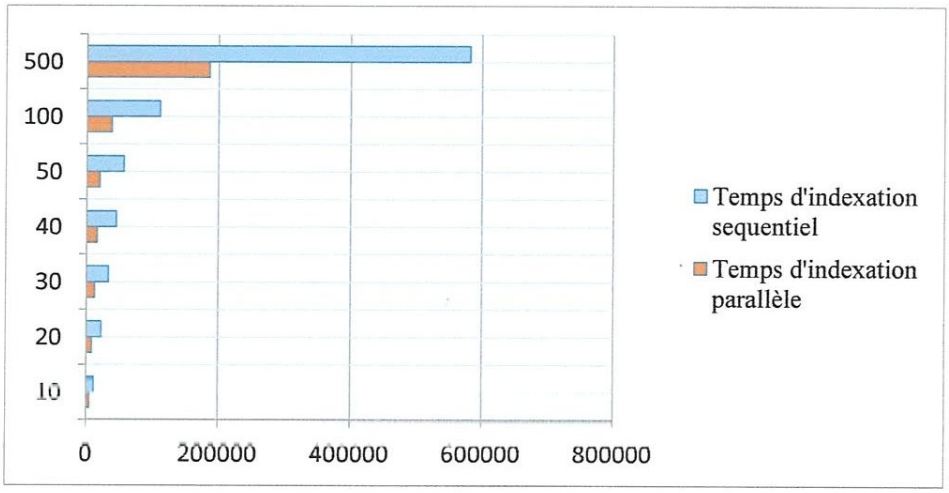

Figure 27 Diagramme en barres du temps d'indexation parallèle et séquentiel

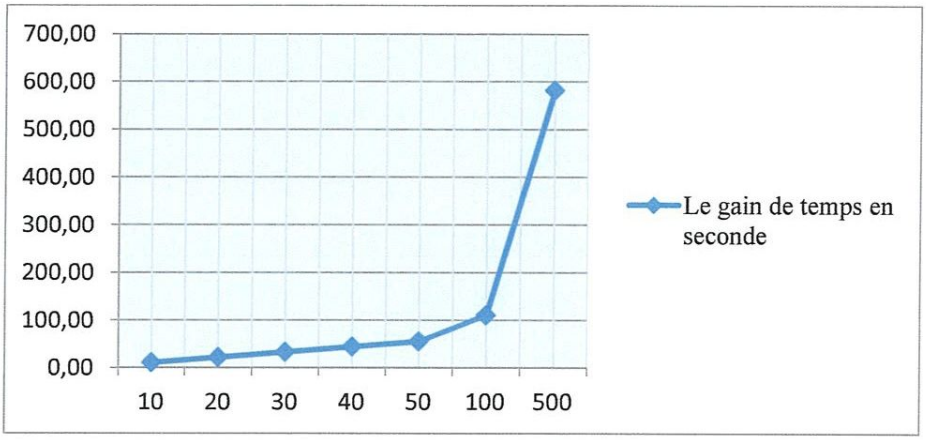

Figure 28 Le gain de temps durant l'indexation

A partir des résultats, nous constatons que le temps d'exécution parallèle est inférieur au temps d'exécution séquentiel. Nous constatons aussi que si le nombre de documents augmente alors la différence entre les deux temps augmente aussi. Par exemple pour un nombre de documents égale à 50 T_P=*19316.33 ms* par contre T_S=*55866.33 ms* c'est-à-dire ; T_S=*2.89* *T_p. Cependant, pour un nombre de documents égale à 500 T_P=*186326.7 ms* et T_S=*581850 ms* c'est-à-dire ; T_S=*3.12* *T_p.

Dans l'étape de comparaison le temps d'exécution parallèle est très inférieur par rapport au temps d'exécution séquentiel comme il est montré dans le diagramme en barres de la figure 29, où le T_P=*8873.667 nano seconde.* Par contre T_s=*296672.3 nano seconde* avec une ration qui égale à R_2=*33.43.* Ces résultats sont montrés dans le tableau 4:

Nombre de fichiers	**10**	**20**	**30**	**40**	**50**	**100**	**500**
Temps de recherche parallèle (nano s)	**14325,67**	**15929,33**	**12829**	**7911**	**6521,333**	**7377**	**8873,667**
Temps de recherche séquentiel (nano s)	**139195**	**138981,7**	**152773,3**	**99746,33**	**107336,7**	**126580,3**	**296672,3**
R2=TP/TS	**9,72**	**8,72**	**11,91**	**12,61**	**16,46**	**17,16**	**33,43**

Tableau 4 Variation du temps de comparaison

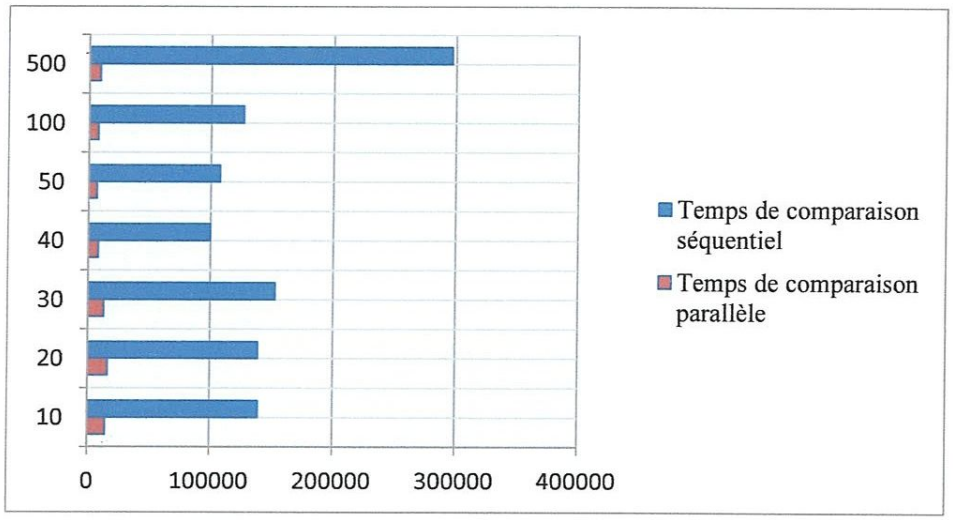

Figure 29 Diagramme en barres du temps de recherche parallèle et séquentiel

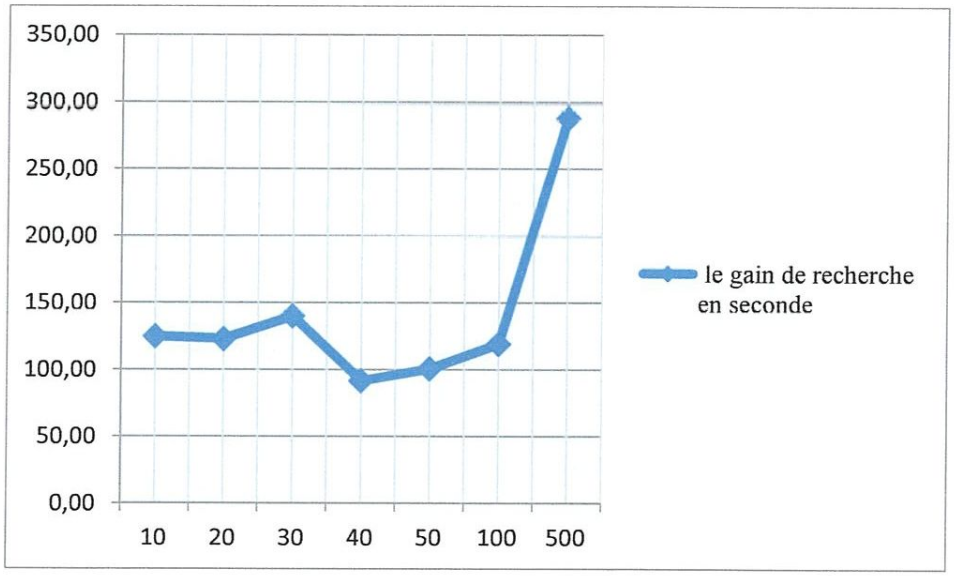

Figure 30 Le gain de temps dans l'étape de comparaison

Nombre de fichiers	10	20	30	40	50	100	500
Gain totale	136.17	145.28	173.34	136.37	156.58	230.29	869.65

Tableau 5 Le gain total en seconde

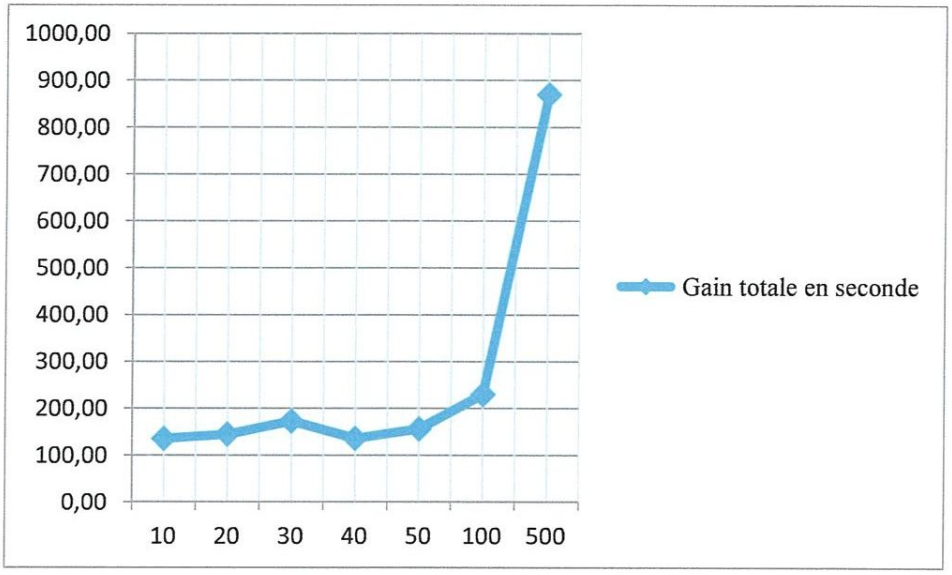

Figure 31 La courbe du gain total

8. Conclusion

A partir de tous ces résultats nous pouvons conclure que le système parallèle est plus efficace que le système séquentiel et son efficacité est remarquable pour une taille de données plus grande. Le gain total du temps pour cette expérimentation varie de *136.17seconde* à *869.65seconde* comme il est montré dans le tableau (gain totale) et la figure (courbe de gain totale).

Conclusion générale

Nous avons essentiellement présenté d'une manière générale les notions de base de la recherche d'information. Nous avons vu aussi les différents modèles de la RI en essayant de donner brièvement les avantages et les limites de chacun.

Etant donné que ce travail est basé sur le processus d'indexation, de comparaison et d'ordonnancement ; on a présenté les différentes étapes qui composent l'indexation et vu les différents modèles qui permettent la comparaison, ainsi que quelques techniques qui permettent le tri.

Dans ce cadre, on a choisi l'algorithme de Porter (pour l'extraction du radicale), ainsi que le modèle vectoriel pour la comparaison, et le réseau de tri par transposition pair impair pour l'ordonnancement afin de développer un nouveau système de recherche d'information utilisant le parallélisme dont le but est d'améliorer les performances en termes de temps.

Le système réalisé dans ce travail permet l'exécution de plusieurs tâches en parallèle ce qui nous a fournis un gain de temps assez conséquent en comparant avec la version séquentielle du même système et dans les mêmes conditions.

Perspectives

L'une des éventuelles améliorations qu'on peut apporter au système de recherche d'information parallèle est l'introduction de la sémantique des termes, c'est-à-dire, qu'il peut arriver dans certain cas que des mots qui se considèrent comme vide sont utilisé pour éviter la répétition. Par exemple « il est le roi de les forets » ; le « il » ici fait référence au « lion » alors on ne peut pas le supprimer.

Dans ce travail, on a pris en considération seulement les documents en anglais ; une éventuelle amélioration serait qu'on améliore le système afin qu'il puisse gérer des documents dans d'autres langues.

Le système peut traiter non seulement les documents numériques textuels mais aussi les images et les vidéos.

Annexe

Nombre de fichiers (parallèle)	10	20	30	40	50	100	500
Test 1	4203	8310	12274	16085	19573	38066	194370
Test 2	4477	7877	12147	15502	19237	37255	181271
Test 3	4607	8192	11971	15194	19139	36983	183339
Moyenne	4429	8126,333	12130,67	15593,67	19316,33	37434,67	186326,7
Nombre de fichiers (séquentiel)	10	20	30	40	50	100	500
Test 1	11332	22506	33365	44672	55905	111268	579989
Test 2	11293	22056	33290	44042	55995	111400	583353
Test 3	11270	22110	33544	44893	55699	110585	582208
Moyenne	11298,33	22224	33399,67	44535,67	55866,33	111084,3	581850

Tableau 1 extraire la moyenne de temps pour le temps d'indexation

Nombre de fichiers (parallèle)	10	20	30	40	50	100	500
Test 1	15394	18923	14753	9301	8018	6094	8981
Test 2	13791	12508	12188	8659	5452	6736	8339
Test 3	13792	16357	11546	5773	6094	9301	9301
Moyenne	14325,67	15929,33	12829	7911	6521,333	7377	8873,667
Nombre de fichiers (séquentiel)	10	20	30	40	50	100	500
Test 1	136629	152666	148176	106482	101029	124442	373006
Test 2	134705	127970	159402	99746	115461	127008	273259
Test 3	146251	136309	150742	93011	105520	128291	243752
Moyenne	139195	138981,7	152773,3	99746,33	107336,7	126580,3	296672,3

Tableau 2 extraire la moyenne de temps pour le temps de recherche

Bibliographie

[1]. **Salton, G, McGill.** *introduction to modern information retrieval.* 1984, McGraw-HillInt. BookCo.

[2]. **N, Hernandez.** *Ontologie de domaine pour la modélisation du contexte en recherche dinformation.* université Paul Sabatier. 2006.

[3]. **F, Boubekeur.** *Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets.* Université Paul Sabatier. 2008. thèse de doctorat en informatique.

[4]. **M, Daoud.** *Accès personnalisé à l'information : approche basée sur l'utilisation d'un profil utilisateur sémantique dérivé d'une ontologie de domaines à travers l'historique des sessions de recherche.* Université Paul Sabatier. 2009. thèse de doctorat en informatique.

[5]. **Cosijn, E. et Ingwersen, P.** *Dimensions of relevance. Information Processing and Management.* 36(4), 2000, pp. 533–550.

[6]. **Robertson, S. Walker, M. Beaulieu.** *OKAPI at TREC 7: Automatic Adhoc Filtering, VLC and Interactive Track.* **S.E.** JULY 1999. 7th text retrieval conference TREC7.

[7]. **Salton, G.** *The Smart Retrieval System : Experiments in AutomaticDocument Processing.*

[8]. **Debili, C. Flurh & F.** Interrogation en Langue Naturelle de Données Textuelles et Factuelles. Grenoble (France) : Intelligent Multimadia Information Systems and Management (RIAO), 1985, pp. 548-556.

[9]. **C. Bourne, B.Anderson.** DIALOG LabWorkbook. Californie (USA) : Lockheed Information Systems, 1979.

[10]. **François, A. Lelu & C.** Information Retrieval Based on Neural Unsupervised Extraction of Thematic Fuzzy Clusters. 1992, pp. pp 93-104.

[11]. **don, Clever.** *Evaluation of information retrieval systems.* 26, s.l. : Journal of documentation,, 1970, Journal of documentation, pp. 55-67.

[12]. **Tambellini, C.** *"Un système de recherche d'information adapté aux données incertaines: adaptation du modèle de langue".* Université de Nice-Sophia Antipolis-UFR sciences,. 2007. Thèse de doctorat en informatique.

[13]. **Baeza_Yates, R, Ribeirio_neto.** *Modern Information Retrieval.* 2011.

[14]. **Flynn, M. J.** *Some Computer Organization and Their Effectivness.* sept 1972.

[15]. **arzaki., M.hammache.** *recherche d'information : un modèle de langue combinant les mots simples et mots composés* . thèse de doctorat.

[16].. **Ren, F, Fan, L, Nie, j-y. SAAK.** *Approach : How to Acquire Knowledge in an actual Application System.* 1999. International Conference On Artificial Intelligence and Soft Computing. pp. pp. 136-140.

[17]. **Salton, G., and Buckley, C.** *Term-weighting approaches in automatic text retrieval. Information Processing & Management.* 1988. pp. 513-523.

[18]. **Salton, G.** Syntactic approaches to automatic book indexing. *the annual meeting on Association for Computational Linguistics (ACL).* 1988, pp. pp. 204–210.

[19]. **Luhn, H.** *A statistical approach to mechanized encoding and searching of literary information.* 1957, IBM Journal of Research and Development 4, pp. 309–317.

[20]. **Salton, G.** *Automatic Information Organization and Retrieval.* New York : s.n., 1968.

[21]. **Maniez, J., and de Grolier, E.** *A decade of research in classification.* 1991, pp. 73-77.

[22]. **Balpe, J., Lelu, A., and Saleh, I.** Hypertextes et hypermédias : réalisations, outils et méthodes. Paris : s.n., 1995.

[23]. **Jacquemin, C., Daille, B., Royanté, J., and Polanco, X.** evaluation of a program for machine-aided indexing. 2002. pp. 765-792.

[24]. **Champcla, Yaël.** Un modèle de recherche d'information basé sur les graphes et les similarités structurelles pour l'amélioration du processus de recherche d'information. Université Paul Sabatier - Toulouse III : s.n., 2009.

[25]. **boukhari, kabil.** *Un Nouvel Algorithme de Stemmatisation pour l'Indexation Automatique de Documents non-structurés : Stemmer SAID.* 2013, master de recherche.

[26]. **Lovins, J. B.** *Development of a stemming algorithm.* Journal of Mechanical Translation.

[27]. **Andrews, K.** *The Development of a Fast Conflation Algorithm for English.* Computer Laboratory, University of Cambridge. cambridge : s.n., 1971. Dissertation for the Diploma in Computer Science.

[28]. **Dawson, J.L.** Suffix Removal and Word Conflation. [éd.] Michaelmas. 1974, pp. p. 33-46.

[29]. **Wechsler, M., Sheridan, P.** *Schäuble, P. Multi-language text indexing for internet retrieval*1997. In Proceedings of the RIAO Conference on Computer-Assisted Information.

[30]. **Kraaij, W., Pohlmann, R.** *Viewing stemming as recall enhancement.* zurich : s.n., 1996. Proceedin proceedings 19th Annual International ACM SIGIR Conference on Research and Development . pp. 40-48.

[31]. **Porter, M.** *An algorithm for suffix stripping.* July,1980, pp. 130-137. Program, 14(3).

[32]. **Salton, G., and Yang, C.***On the specification of term values in automatic indexing* . 1973, In Journal of Documentation, pp. 351–372.

[33]. **Kompaoré, N.D.Y.** *Fusion de systèmes et analyse des caractéristiques linguistiques des requêtes: vers un processus de RI adaptatif».* Université Paul Sabatier de Toulouse, 2008. 2008. Thèse de doctorat en informatique.

[34]. **Tamine, Lynda.** Optimisation de requête dans un système de recherche d'information approche basée sur l'exploitation de techniques. Université PAUL SABATIER. 2000. thèse.

[35]. **Maron, M., and Kuhns, J.** *On relevance, probabilistic indexing and information retrieval.* 1960, Journal of the Association for Computing Machinery 7, pp. pages 216–244.

[36]. **Robert ROBERTSON S. E., WALKER S.** *Some Simple Effective Approximations to the2-Poisson Model for Probabilistic Weighted Retrieval.* 1994, Proceedings of SIGIR 1994, pp. p. 232-241.

[37]. **Sanderson, M.** *Test Collection Based Evaluation of Information retrieval System.* 2010. pp. pp. 247-375.

[38]. **Clet-Ortega., Jérôme.** *Exploitation efficace des architectures parallèles de type grappes de NUMA l'aide de modèles hybrides de programmation. Calcul parallèle, distribué et partagé.* université Sciences et Technologies - Bordeaux I. Bordeaux I : s.n., 2012.

[39]. **D.M. Tullsen, S.J. Eggers, and H.M. Levy.** Simultaneous multithreading :Maximizing on-chip parallelism. *22nd Annual International Symposium on.* JUNE 1995, Vol. pages 392 –403.

[40]. **Coullon, Hélène.** *Modélisation et implémentation de parallélisme implicite pour les simulations scientifiques basées sur des maillages.* université d'orléans. 2014. thèse de doctorat.

[41]. **Louvet, Violaine.** *Architecture des ordinateurs.* Ecole Doctorale MathIf. 2010.

[42] **Salton, G., and Yang, C.***On the specification of term values in automaticindexing.* s.l. : In Journal of Documentation, 29 (1973), pp. 351–372.

[43] **Nicholas J. Belkin, P.** *Proceedings of the 15th Annual International ACM*

Conference on Research and Development in Information Retrieval. June 21-24, 1992.Copenhagen, Denmark: ACM 1992.

[44] **Salton., G.** *A comparison between manual and automatic indexing methods.* 1971, Journal of American Documentation, pp. 20(1) :61–71.