République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université 20 Août-1955-SKIKDA
Faculté des Sciences
Département d'Informatique

Mémoire fin d'étude en vue de l'obtention du diplôme

Master en Informatique

Spécialté : Systèmes d'informations avancés et applications (SIAA)

thème

Validation et Vérification formelle d'un système de cache distribué

Réalisé par :
— BOUCHEMAL Yassmine
— MOUMEN Wisal

Encadré par :
BOULAICHE Mehdi
LAYADI Saïd

Session : Juin 2023

Résumé

Ce mémoire de master propose une étude approfondie sur l'application de la coloration de graphe dans le contexte d'un système de cache distribué. Les systèmes de cache distribué sont largement utilisés pour améliorer les performances des systèmes informatiques en stockant les données fréquemment utilisées dans des caches répartis. Cependant, le placement optimal des données dans les caches reste un défi majeur.
Dans ce mémoire, nous explorons l'utilisation de la coloration de graphe comme approche pour résoudre ce problème de placement des données dans un système de cache distribué.
Nous commençons par une revue approfondie des travaux existants sur les systèmes de cache distribué et la coloration de graphe. Ensuite, nous proposons une méthodologie détaillée pour modéliser et résoudre le problème de placement des données à l'aide de la coloration de graphe. Nous étudions également différentes stratégies d'algorithme de coloration pour optimiser les performances du système de cache distribué.
En conclusion, ce mémoire de master présente une approche novatrice en utilisant la coloration de graphe pour résoudre le problème de temps de réponse et la redondance des données dans un système de cache distribué.
Les résultats obtenus indiquent que l'algorithme utilisé est adaptable à des échelles différentes, mais il ne garantit pas systématiquement la solution optimale.
La vérification formelle dans un système de cache distribué assure la cohérence des données,et la validité des opérations de mise en cache et d'éviction, grâce à l'utilisation de méthodes et d'outils formels.

Mots clés : systèmes de cache distribué, cache répartié, stratégies de cache,coloration de graphe, automates temporisés.

Remerciements

Nous voulons, avant tout, remercier Dieu le tout puissant pour la force, la volonté mais surtout la santé qu'il nous a données pour entamer et finir ce mémoire.

Tout d'abort, ce travail ne serait pas riche et n'aurait pas pu voir le jour sans l'aide et l'encadrement de monsieur **BOULAICHE MEHDI** et **LAYADI SAID**, on les remercient pour la qualité de leur encadrement exceptionnel , leur conseils précieux,pour leur temps, pour leur patience, rigueurs pour leur disponibilité durant notre préparation de ce memoire .

Pour le grand honneur qu'ils nous font en acceptant de juger ce travail. Nous vous remercions de l'honneur que vous nous avez fait en acceptant de présider notre jury. Nous vous remercions de votre enseignement et nous vous sommes très reconnaissants de bien vouloir porter intérêt à ce travail.

Nous remercions les membres du jury pour leur présence, pour leur lecture attentive de notre mémoire ainsi que pour les remarques qu'ils m'adresseront lors de cette examination afin d'améliorer notre travail.

Notre remerciement s'adresse également à tous nos professeurs pour leur générosité et la grande patience dont ils ont su faire preuve malgré leurs charge académique et professioelle.

Nos profonds remerciements vont également à toutes les personnes qui nous ont aidé et soutenue de prés ou de loin.

Dédicace

Avec une sensation de détermination ambitieuse et courageuse j'ai conduit mon rêve vers la réalité. Avec une immence fiérté et l'expression de ma reconnaissance du profond de mon coeur.

je dédie ce travail :

A ma chère mère **FELLA GUESSABI** et A mon cher père **RACHID**

Qui n'ont jamais cessé, de formuler des prières à mon égard, de me soutenir et de m'épauler pour que je puisse atteindre mes objectifs.

A ma chère sœur **HANA** et mon frère **FARES** que dieu les protège et leur offre la chance, le bonheur et la réussite.

A ma chère binôme **WISAL** pour sa entente, sa sympathie et sa compréhension tout au long de ce projet.

A ma grande mère, mes tentes que dieu leur donne une longue et joyeuse vie.

A mes chères amies **CHAFIKA** , **RAYENE** et **OUMAIMA** pour leur soutiens, encouragement et amour inconditionnels .

A mes cousines **SOUMIA**,**SELMA** et **HOUDA** qui m'a toujours soutenu avec tous les moyens possibles.

YASSMIN BOUCHEMAL

Dédicace

A mes plus grands soutiens et sources d'inspiration, je dédie ce travail avec tout mon amour et ma reconnaissance infinis.

A ma mère **HATHOUT Souria** qui a toujours été mon port d'attachement et ma boussole, merci pour ton amour inconditionnel, ton dévouement et ton soutien inébranlable.

A mon père **Ahcene** qui m'a appris l'importance du travail acharné, de la persévérance et de l'honnêteté. Tu m'as inspiré à viser plus haut et à poursuivre mes rêves. Je te suis infiniment reconnaissante pour ta confiance en moi et ton amour.

A mes frères **Tamer** et **Noufel Salah Eddine** ainsi qu'à mes sœurs adorées **Rania, Amani Sirine,** et **Ines** qui sont aussi mes meilleures amies, merci pour votre soutien constant, votre humour contagieux et votre présence réconfortante. Vous êtes ma source de joie et de bonheur, et je suis fiere de vous avoir dans ma vie.

A ma belle-amie **BOUZIDI Ilhem** qui est devenue une sœur et une confidente, merci pour ta gentillesse.

Enfin, à mon binôme **BOUCHEMAL Yassmine** qui est devenue une collaboratrice talentueuse, merci pour notre collaboration fructueuse et notre amitié.

Au-delà des noms cités, il existe un cercle précieux de personnes qui ont joué un rôle significatif dans mon parcours **Ma seule tante et oncles**,je vous exprime ma reconnaissante pour votre présence et votre soutien qui ont marqué positivement ma vie.

MOUMEN Wisal

Table des matières

Table des figures

liste d'abréviations

AT **A**utomate **T**emporisé

R-TA **A**utomate **T**emporisé avec temps **R**elatif

AR **A**utomate de **R**égions

CTAR **C**ontraintes **T**emporelles **A**tomique **R**estrientes

CTAG **C**ontraintes **T**emporelles **A**tomique **G**lobale

GR **G**raphe de **R**égion

Chapitre 1

Introduction générale

Sommaire

Un système de cache distribué est un ensemble de nœuds répartis sur plusieurs ordinateurs ou serveurs qui sont utilisés pour améliorer les performances d'un système en mémorisant les données fréquemment utilisées.
Ce système permet de réduire les temps d'accès aux données en les conservant dans une mémoire plus proche des utilisateurs, ce qui permet d'éviter les temps d'attente pour les requêtes de données fréquentes et de fournir une tolérance aux pannes accrue grâce à la répartition de la charge de travail sur plusieurs nœuds.
Ce système fonctionne en créant d'une copie de données fréquemment utilisées dans le cache distribué. Cette copie est stockée dans différents nœuds du réseau pour améliorer la disponibilité et la résilience du système. Cette dernière peut causer la redondance des données.
L'objectif de notre projet visait à optimiser la redondance des données dans les systèmes de cache distribué tout en réduisant le temps de réponse et en garantissant une cohérence optimale des données, afin d'optimiser les performances et la disponibilité des applications distribuées.
Pour résoudre ce problème, nous proposons l'utilisation de l'algorithme de coloration des sommets de graphes pour optimiser la répartition des données entre les différents nœuds du réseau.
Nous voulons aussi de mettre en place une application qui automatise le principe de fonctionnement de cet algorithme sur des exemples de systèmes de caches distribués et qui montre que notre approche donne des bons résultats.
Enfin,La modélisation du système de cache distribué par un automate permet de représenter de manière structurée les différents états, les transitions et les interactions, facilitant ainsi l'analyse formelle, la détection d'erreurs potentielles et l'optimisation des performances du cache distribué.

1.1 Organisation du mémoire

En plus de cette première partie qui comporte le chapitre Introduction générale, le reste est organisé en quatre Chapitre :

Le chapitre 02 consiste en quelques concepts de base pour le cache répartie,le serveur cache,et le système de cache distribué.

Le chapitre 03 dans ce chapitre fournit un aperçu des concepts clés liés aux types et classes de systèmes de cache distribué, aux réseaux orientés au contenu, aux stratégies de cache, à l'étude connexe, à l'environnement commun de simulation et aux métriques d'évaluation. Ces notions posent les bases nécessaires pour la compréhension et l'analyse approfondie des systèmes de cache distribué dans les chapitres suivants.

Le chapitre 04 dans ce chapitre nous allons modéliser le système de cache distribué par un graphe et proposer d'appliquer l'algorithme de coloration de graphe "algorithme de welsh powell " sur cet graphe, pour le colorer avec un nombre minimal de couleurs possible.

Le chapitre 05 est dédié à la mise-en-ouvre et l'application de l'algorithme.Nous décrivons tout d'abord l'environnement de travail et les étapes d'installation des outils logiciels et matériels nécessaires, avant de se focaliser sur l'application de l'algorithme la validation des résultats, durée d'exécution, et la qualité de la solution.

Le chapitre 06 dans ce chapitre vérification formelle d'un système de cache distrubué. Nous présentons le modèle d'un automate temporisé,son propriétés, et les règles de passage d'un diagramme état transition vers un automate temporisé.Et on termine ce chapitre par donner une présentation succincte sur l'outil Uppaal et une pratique de la vérification sur notre exemple illustratif.

Et enfin une conclusion générale pour discuter les résultats et proposer quelques perspectives des travaux présentés dans ce manuscrit.

Chapitre 2

Généralités sur les Systèmes de caches distribués

Sommaire

2.1 Cache répartie

2.1.1 l'historique de cache répartie :

Pour atteindre cet objectif, nous proposons un cadre commun. Cette portée compare toutes les stratégies de mise en cache et décide laquelle est la meilleure dans chaque cas John von Neumann mentionnait déjà l'utilisation des hiérarchies de mémoire en 1947 coût de compensation 1 Les extrêmes de la mémoire rapide (16)
utiliser la pensée en couches Les mémoires exploitant la localité temporelle ont été introduites par Maurice en 1965 Wilkes a ajouté un Espace de stockage rapide, laissez-le "à portée de main" le plus récent consignes qu'il a exécutées. Quelques années plus tard, en 1968, le terme de cache apparaît avec l'ordinateur *IBM System/360-85*(42), l'un des premiers Intégrer la mémoire cache dans son processeur.

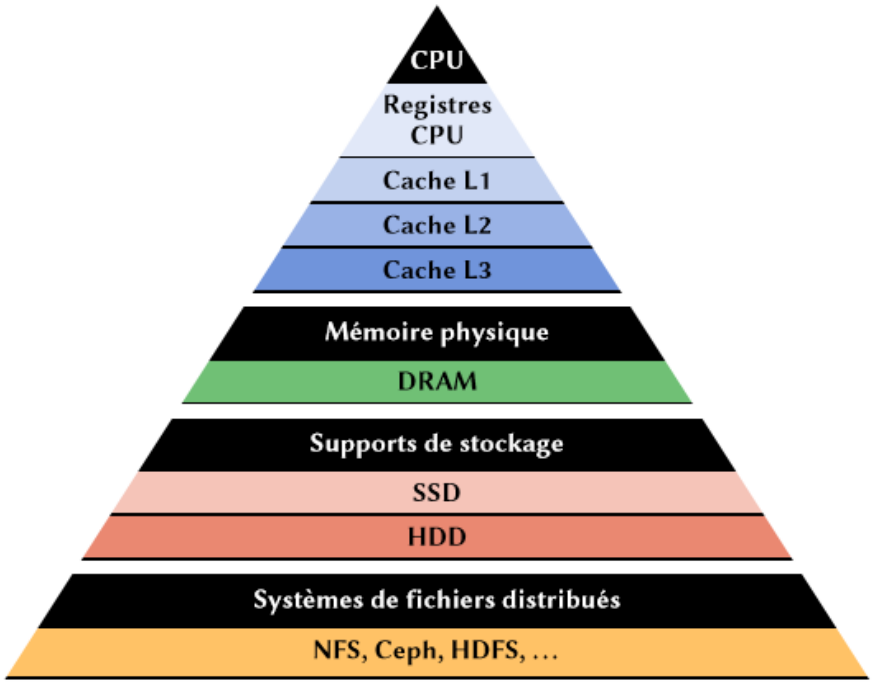


FIGURE 2.1 – Hiérarchie des mémoires

Les processeurs d'aujourd'hui contiennent Plusieurs niveaux de cache pour une capacité accrue. Si l'objectif premier des premiers caches était d'accélérer les accès mémoire, ce principe a rapidement été étendu aux périphériques, notamment les disques.
Dans le même temps, la popularité des systèmes de fichiers distribués a conduit à l'introduction de caches pour réduire la charge sur les serveurs de fichiers et mieux passer À grande échelle : en ajoutant des données (locales) en cache aux machines clientes Système de fichiers distribué, nous réduisons l'accès au serveur. cache pour Les environnements distribués apparaissent notamment dans le système de fichiers distribué Andrew File System (46)(60)(68), nous utilisons le cache local Le SSD (Solid State Drive) accélère l'accès au disque dur traditionnel (HDD) (5)(35). Ces différents niveaux de la hiérarchie mémoire sont représentés par la figure 1.
Ce n'est qu'au début des années 90 que seront introduits les caches répartis entre plusieurs nœuds. Ainsi, partant du constat que la latence d'accès à un autre nœud dans un réseau local (LAN) est plus faible que la latence d'accès à un disque dur, Douglas Comer et James Grioen proposent d'utiliser la mémoire en réseau fournie par des serveurs pour étendre la mémoire locale de nœuds clients (18)

2.1.2 Propriètés de caches :

Les caches sont des systèmes complexes caractérisés par un ensemble de propriétés, nous présentons les principales dans cette section. Nous discutons notamment de la localisation des données dans le cache, des différentes stratégies de caches, de la gestion des écritures ainsi que du choix des données à évincer du cache.

Placement et localisation des données :

La méthode par laquelle un élément de données est placé dans le cache est une caractéristique importante qui a un impact important sur les performances du cache. Une approche simple consiste à appliquer une fonction de hachage à l'identité des données Obtient le numéro de la "boîte" de cache où les données seront stockées.
Par exemple, dans le cadre des caches CPU, on parlera des caches qui correspondent direct, chaque ligne de mémoire ne peut entrer qu'une seule ligne de cache Réserve. Cette approche permet une localisation rapide des données, mais au prix de Taux de données manquantes plus élevé : deux données de point d'accès peuvent être placées dans la même boîte et s'exclure. Quand les données peuvent aller à plusieurs endroits Pour différents caches, on parlera de cache associatif à N emplacements distincts (29) à N emplacements différents. Le choix du nombre de canaux est un compromis entre le coût Taux de recherche et taux de ratés : Plus il y a de voies, plus le taux d'échec est élevé sera faible, mais la localisation des données sera plus coûteuse. cette méthode Il peut également être appliqué aux caches locaux sur des disques de taille xe, où une fonction de hachage peut calculer l'emplacement des données (69) . Avec couverture Distribuée, une table de hachage distribuée peut être utilisée pour déterminer Les nœuds de cache distribué auxquels les données seront envoyées (21) (24) (32) Lorsque l'on veut pouvoir placer des données n'importe où dans le cache, il est Nécessaire pour conserver certaines métadonnées afin de permettre la récupération de ces données.
Ces métadonnées décrivent les identifiants de données et Le numéro de l'espace de cache où il réside. En cas de cache Distribuées, ces métadonnées peuvent être locales, distribuées ou déléguées à un nœud (gestionnaire) responsable de l'accès centralisé aux métadonnées (19) (21) (59)

Algorithmes de remplacement de cache

Besoin de sélectionner les données d'une victime expulsée lorsque le cache est plein La cache prend sa place. La sélection des données est déléguée à l'algorithme remplacement du cache. Idéalement, ces derniers devraient choisir de ne plus être Il est utilisé depuis le plus longtemps (11).
parce que c'est impossible Pour prédire l'avenir, la plupart des algorithmes de remplacement de cache s'appuient sur Principe local :

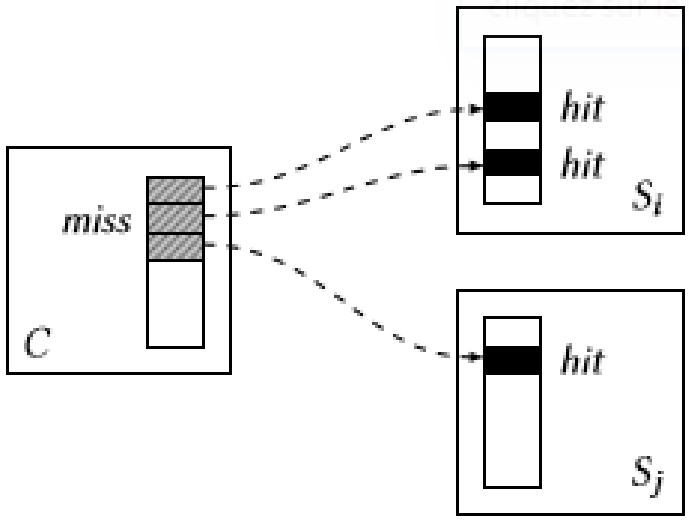
- Localité spatiale : Si une donnée est référencée à un instant donné, il existe Il y a de fortes chances que les données adjacentes soient bientôt accessibles ;
- Localité temporelle : Les données référencées à un moment donné sont susceptibles d'être à nouveau consultées dans un futur proche.

De nombreux algorithmes de remplacement de cache ont été conçus pour résoudre différents problèmes (type d'accès, taille du cache, latence, etc.). Les plus classiques comme LRU (Least Récemment Utilisé) ou FIFO (First-in First-out) ont de nombreuses variantes pour améliorer le taux de réussite (hit) du cache sous certains aspects Coût d'exécution du cas ou de l'algorithme limite. Par exemple, les secondes chances sont Une variante de FIFO où les données les plus anciennes sont conservées Une liste FIFO, si elle a été utilisée pendant la durée de vie de la liste. CLOCK est une modification de cet algorithme qui s'appuie sur des listes circulaires, évitant Les données de la liste doivent être déplacées. Les algorithmes de rempla-

cement de cache peuvent également être combinés comme Former de nouveaux algorithmes. Par exemple, 2Q (two queue) (34) est basé sur Liste FIFO et liste LRU. Lorsque des données sont entrées dans le cache, elles sont placées dans la liste FIFO. S'il est référencé alors qu'il figure déjà dans cette liste, il est Sont considérés comme des hotspots et sont promus dans la LRU. algorithme utilisé Le noyau Linux, que nous détaillons dans la section 3.3, est similaire à l'algorithme 2Q.

2.1.3 Cache distant :

Les caches distants ont été présentés par Comer et Griffonen (28). Ce sont des caches répartis qui comprennent des serveurs de cache qui fournissent de la mémoire comme support de stockage pour le cache, et des clients qui utilisent ce cache sans interagir entre eux.



(a) Exemple de cache distant

FIGURE 2.2 – cache distant

l'architecture de cache distant :

La figure 2.3 présente l'architecture globale de notre cache distant. Il est constitué par 2 modules noyau : un module « client » et un module « serveur » Une couche réseau basée sur TCP est partagée entre ces deux modules pour gérer l'envoi et la réception de messages entre le client et le serveur. l'unité de stockage est donc la page(43)

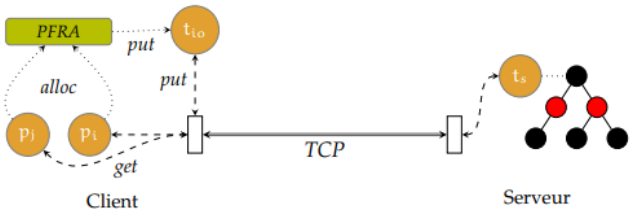


FIGURE 2.3 – architecture globale du cache distant

Protocole de communication :

Le client et le serveur communiquent entre eux par envoi de messages via une socket TCP. Il n'a pas été nécessaire de développer un protocole de communication complexe, en particulier parce que nous réalisons un cache distant qui ne nécessite pas d'intégrer des mécanismes complexes comme la recherche ou de placement de pages. Ainsi, ce protocole repose principalement sur 2 actions simples, put et get :
L'action put :est effectée par le client pour placer une page dans le cache distant. Lorsque le client veut envoyer une page au serveur, il envoie un message put au serveur qui contient la page et des informations permettant de l'identifier de façon unique, puis attend un acquittement du serveur pour l'informer que la page a été correctement reçue.
L'action get : est effectuée par le client pour demander une page au serveur dans le cache distant. Lorsque le client veut récupérer une page du cache distant, il envoie un message get au serveur avec l'identifiant unique de la page. Le serveur répond par un message get response, si la page a été trouvée ce message contient la page, sinon un drapeau indique que la page n'est pas dans le cache distant(43)
Ce protocole repose sur l'existence d'un identifiant unique. Lors des échanges de pages entre le serveur et le client, chaque page est identifiée par un triplet (poolid, inode, index) :

- le poolid identifie le support de stockage sur lequel est stockée la page, cet entier est généré par le client pour chaque support de stockage qui utilise le cache distant ;
- l'inode identifie l'inode qui contient la page ;
- l'index identifie le numéro de la page dans l'espace d'adressage du fichier, en commençant à partir de 0.

2.1.4 Les proprietés de caches réparties :

Dans cette section, nous aborderons les différentes propriétés des caches répartis. Tout d'abord, dans l'Accès concurrents et cohérence des données, nous examinerons la cohérence des données, puis dans Tolérance aux fautes, nous parlerons de la tolérance aux fautes. Nous évoquerons ensuite les problèmes d'intégrité et de confidentialité liés aux caches répartis dans la Sécurité, avant de conclure avec une présentation de la généricité des caches répartis dans la Généricité.

Accès concurrents et cohérence des données :

Lorsqu'un niveau de cache supplémentaire est introduit dans un système distribué, cela complexifie la gestion des accès concurrents aux données afin d'assurer leur cohérence. Bien que les accès en écriture soient de plus en plus courants, notamment dans les environnements cloud(15), les études basées sur l'analyse des traces d'entrée/sortie des systèmes de

fichiers distribués montrent que les accès partagés sont peu fréquents et les accès partagés en écriture le sont encore moins(7)(41) Par conséquent, la gestion de la cohérence des données des systèmes de stockage répartis est généralement basée sur des approches centralisées.

Pour gérer les accès concurrents dans un système distribué, une approche courante est celle qui utilise des baux de Gray et Cheriton(27) Cette méthode consiste à confier la gestion des droits d'accès à un serveur, qui peut octroyer un bail à un client pour effectuer des opérations de lecture, écriture, etc. Lorsque le bail expire, le client doit en obtenir un nouveau auprès du serveur pour continuer à accéder aux données. Si un client demande des droits qui entrent en conflit avec ceux d'autres baux en cours (par exemple, un conflit entre un lecteur et un écrivain), le serveur peut révoquer les baux existants pour donner un bail au nouveau client.

De nombreux systèmes de fichiers distribués, tels que *NFS*(63), *Ceph*(71) et *Sprite*(48), utilisent des mécanismes basés sur des baux pour gérer et optimiser les accès partagés. Par exemple, dans *Ceph*, un serveur de métadonnées accorde des droits aux clients pour mettre en cache les blocs d'un fichier. Si un conflit entre un lecteur et un écrivain survient, le serveur de métadonnées révoque les droits de mise en cache des clients et les oblige à effectuer des entrées/sorties synchrones, ce qui permet au système de fichiers distribué de respecter le modèle de cohérence spécifié par *POSIX*(31)
"Chaque lecture (*read*) consécutive à une écriture (*write*), même avec des processus différents, doit renvoyer les données modifiées par l'écriture."
Ces approches de gestion des accès concurrents sont essentielles pour assurer la cohérence des données dans les systèmes distribués de stockage de fichiers.

Plusieurs travaux ont été menés pour améliorer les performances du système de fichiers distribué *NFS*, en prenant en compte la cohérence des données. Deux exemples de tels travaux sont *NFS-cc*(73) et NFS-CD(9) NFS-cc est une version modifiée de NFS qui intègre un cache collaboratif. Afin de maintenir le modèle de cohérence de NFS (close-to-open(63),le serveur retransmet les requêtes de blocs avec les attributs des fichiers au client qui possède le bloc, pour que celui-ci puisse vérifier si sa copie est à jour. *NFS-CD* permet de déléguer le rôle de serveur à un client spécifique pour un fichier donné, ce qui permet de diminuer la charge du serveur. Cette architecture, appelée cluster-delegation, permet au client de gérer les conflits de manière similaire à un serveur *NFS* classique.

Cortes et al(19) ont proposé une approche alternative pour contourner le problème des accès partagés, qui consiste à éviter l'utilisation de cache local. Dans cette approche, les blocs de données sont directement transférés du cache global (coopératif) vers l'espace d'adressage de l'utilisateur, ce qui évite la nécessité d'une recopie dans le cache local. En supprimant la réplication des données dans le cache local, ce système élimine les problèmes de cohérence (au sens *POSIX*) liés à l'accès partagé. Toutefois, il est impératif de ne pas avoir de réplication dans le cache global pour garantir la cohérence des données.

Tolérance aux fautes

Comme tout système distribué, les caches répartis peuvent être confrontés à des pannes qui doivent être gérées pour éviter des pertes de données et garantir des propriétés de vivacité. Nous nous intéressons ici plus spécifiquement aux pannes franches, caractérisées par l'arrêt complet d'un nœud qui ne répond plus, ainsi qu'aux pannes franches avec reprise, lorsque le nœud qui a cessé de fonctionner recommence à participer au cache réparti. Bien que les fautes byzantines ne soient pas directement prises en compte, les mécanismes de contrôle d'intégrité peuvent être utilisés pour les gérer partiellement.

— **Les caches en lecture seul** seule présentent l'avantage de simplifier la gestion de la cohérence puisque les données du cache sont synchronisées avec celles du périphérique de stockage principal. Cependant, il est essentiel de détecter les pannes des nœuds du système de cache réparti pour garantir certaines propriétés, telles que la vivacité. Ainsi, des mécanismes de surveillance peuvent être mis en place pour surveiller la disponibilité des nœuds du système et éviter les attentes interminables pour une donnée d'un nœud du cache réparti.

— **les caches en écriture** : Les caches répartis plus complexes, tels que ceux qui permettent les écritures différées, nécessitent des mécanismes de tolérance aux pannes pour éviter toute perte de données. Gray et Cheriton ont proposé l'utilisation de *leases* (27) pour garantir la cohérence des données et éviter les pertes de mises à jour. Ces leases donnent aux clients des baux d'une durée limitée et une fois le bail expiré, le client doit synchroniser sa donnée sur le support de stockage principal. Les leases permettent de tolérer les pannes non-byzantines, mais il est crucial de contrôler la dérive des horloges du système.

NFS-CD(9)(10) un exemple de système de cache réparti qui utilise la technique de réplication pour améliorer la tolérance aux pannes. Dans ce système, les écritures sont répliquées de manière synchrone dans la mémoire de plusieurs nœuds, permettant de tolérer au plus N-1 fautes. Toutes les écritures sont finalement écrites sur un support de stockage stable et partagé entre les nœuds.
Dans *PAFS*(19)(20), un système de fichiers réparti, les buffers sont répartis sur plusieurs nœuds, et les créateurs ont proposé l'utilisation de techniques de calcul de parité similaires à celles utilisées dans le *RAID 5*(52), ainsi que des serveurs de parité responsables de la maintenance des informations de parité et de la reconstruction des blocs en cas de défaillance.

Sécurité

Le transfert de données d'un cache local vers un cache distant peut être réalisé par un cache réparti. Les systèmes de cache coopératifs supposent généralement un environnement de confiance où les machines peuvent se faire mutuellement confiance (59)(73)(24). Cependant, dans des contextes tels que les clouds publics, où tous les nœuds du cache ne sont pas fiables, il est nécessaire d'inclure des mécanismes de sécurité pour garantir l'intégrité et la confidentialité des données stockées sur des machines tierces dans le système de cache réparti. Assurer l'intégrité des données implique la capacité de déterminer si les données ont été

altérées ou non. Lorsqu'il s'agit de caches distants, une méthode simple consiste à stocker un hash des blocs transmis à d'autres nœuds, ce qui permet de vérifier l'intégrité du bloc lorsqu'il est récupéré. Cependant, cette approche peut devenir coûteuse si certains nœuds sont autorisés à modifier les données, car le hash doit alors être recalculé. Pour résoudre ce problème, Shrira et Yoder (64) ont proposé une solution basée sur le hachage incrémental (12), qui garantit l'intégrité des données exportées vers d'autres nœuds tout en permettant des modifications partielles sans avoir besoin de recalculer le hash de l'ensemble des données.
Assurer la confidentialité revient à empêcher toute entité non autorisée de lire des données. Le chiffrement est une méthode couramment utilisée pour garantir cette confidentialité, que ce soit en chiffrant les communications entre deux nœuds de confiance ou en chiffrant les données stockées dans un cache distribué.
Le système de fichiers distribué Shark est un exemple de système qui assure à la fois l'intégrité et la confidentialité des données. Dans ce système, le serveur de fichiers distribue Shark (26) aux clients autorisés, leur permettant de prouver qu'ils ont le droit de lire un fichier particulier. Les clients utilisent ces jetons pour accéder aux données en chiffrant les communications entre les nœuds à l'aide d'un algorithme de chiffrement symétrique tel que AES.

Généricité

La généricité est une des caractéristiques importantes des caches répartis que nous examinons. Le fait qu'ils puissent être implantés à plusieurs niveaux au sein d'un système d'exploitation (voir la figure 4) signifie que différentes approches ont des contraintes particulières quant aux types d'applications qui peuvent les utiliser.
Certains caches répartis se situent au *niveau applicatif*, c'est-à-dire dans l'espace utilisateur. Par exemple, Memcached (25) un cache réparti principalement utilisé pour stocker en cache les résultats de requêtes à des bases de données. Cependant, l'inconvénient de cette approche est que les applications qui utilisent ce type de cache doivent être conçues spécifiquement pour cela ou doivent être adaptées pour utiliser l'API offerte par le service de cache. Pour illustrer, le serveur de base de données PostgreSQL (65) une extension (pgmemcache) pour fournir une interface à Memcached.

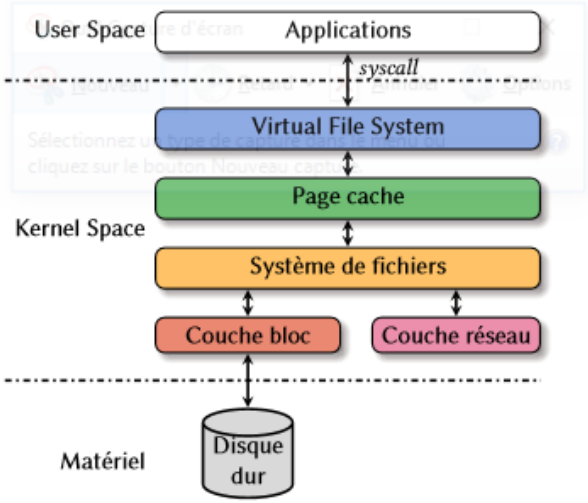


FIGURE 2.4 – Les différentes couches d'un système d'exploitation

Les approches intégrées au système d'exploitation présentent l'avantage d'être plus modulaires et peuvent être utilisées avec les applications sans nécessiter d'adaptation. Par exemple, il est possible de proposer un cache réparti *au niveau du système de fichiers*(4)(26)(19), ce qui permet de bénéficier des fonctionnalités offertes par les couches inférieures du système telles que le buffering et le prefetching. Cependant, il existe de nombreux systèmes de fichiers différents, chacun offrant des fonctionnalités variées telles que le chiffrement, la compression, (buffering, prefetching, ...). Certains sont même spécialisés pour des supports de stockage utilisant des technologies particulières comme la mémoire flash (40) (72) ou le Shingled Magnetic Recording (SMR) (39), (66)

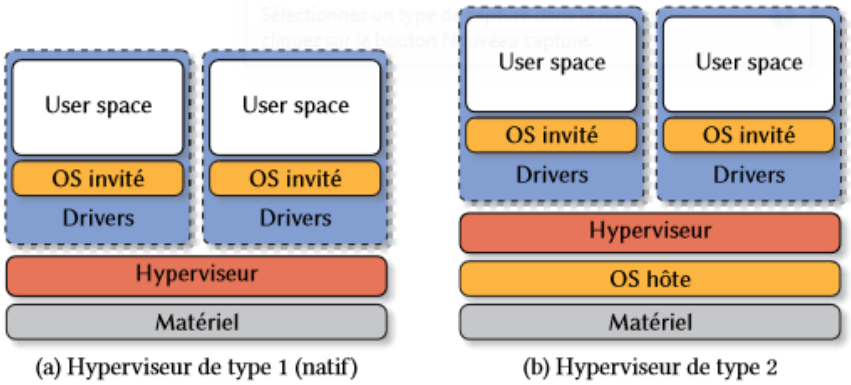


FIGURE 2.5 – Les différents types d'hyperviseurs

Une autre option consiste à se rapprocher du périphérique de stockage en se positionnant *au niveau de la couche bloc* (28) Cependant, cette approche présente un inconvénient supplémentaire : certains systèmes de fichiers, en particulier les systèmes de fichiers distribués tels que NFS, ne sont pas basés sur la présence d'un périphérique bloc, ce qui rend ce type de cache inadapté. une autre solution privilégiée consiste à se situer à un niveau intermédiaire, directement dans la page cache du système d'exploitation (voir section 4). Cette approche permet de rendre le cache réparti compatible avec n'importe quel système de fichiers.

2.2 Le serveur cache

2.2.1 Définition de serveur cache

Un serveur cache est un serveur qui stocke temporairement des données afin d'améliorer les performances du système en réduisant le temps d'accès aux données. Il peut s'agir de données fréquemment utilisées ou demandées par les utilisateurs, telles que des pages Web, des images ou des vidéos. Les serveurs cache peuvent être utilisés pour améliorer les temps de réponse du système et réduire les coûts de bande passante (51).

2.2.2 Les avantages de serveur cache

Les avantages d'un serveur cache sont nombreux :

1. Amélioration des performances : En stockant des données fréquemment utilisées dans un serveur cache, les temps d'accès sont réduits, ce qui améliore les performances du système dans son ensemble ;
2. Réduction des coûts de bande passante : Les serveurs cache peuvent réduire les coûts de bande passante en stockant des données fréquemment utilisées localement plutôt que de les récupérer à partir d'un serveur distant à chaque fois qu'elles sont demandées ;
3. Réduction de la charge du serveur : En stockant des données fréquemment utilisées dans un serveur cache, la charge sur les serveurs de stockage est réduite, ce qui permet aux serveurs de stockage de mieux gérer les demandes d'accès aux données ;
4. Amélioration de la disponibilité : Les serveurs cache peuvent améliorer la disponibilité en stockant une copie des données localement, ce qui permet de fournir une réponse rapide même si le serveur de stockage est indisponible(49).

2.2.3 les inconvénients de serveur cache

Bien que les serveurs cache présentent de nombreux avantages, ils peuvent également présenter des inconvénients :

1. Cohérence des données : Lorsqu'un serveur cache stocke une copie de données, il peut y avoir des problèmes de cohérence des données si la copie stockée n'est pas mise à jour correctement lorsqu'une modification est apportée aux données d'origine ;
2. Coût élevé : Les serveurs cache peuvent être coûteux à mettre en place, en particulier pour les grandes entreprises qui ont besoin de stocker de grandes quantités de données ;
3. Besoin de mise à jour : Les données stockées dans le serveur cache doivent être mises à jour régulièrement pour s'assurer que les données stockées sont à jour et précises ;
4. Sécurité : Les serveurs cache peuvent présenter des risques de sécurité si les données stockées dans le cache sont sensibles et ne sont pas correctement sécurisées.

2.2.4 L'architecture de serveur cache :

L'architecture de serveur cache est composé de deux éléments :

1. un thread t_s qui gère la réception et le traitement des messages provenant du client ;
2. une zone de stockage destinée à contenir les pages envoyées par le client . (43)

2.2.5 le fonctionement de serveur cache

Le fonctionnement d'un serveur cache consiste à stocker temporairement des données pour les rendre plus rapidement accessibles. Lorsqu'un utilisateur demande du contenu à partir d'un site web, le serveur cache récupère ce contenu à partir d'un serveur d'origine, puis enregistre une copie de ce contenu dans le cache. Si un autre utilisateur demande le même contenu, le serveur cache peut fournir la copie stockée plutôt que de récupérer le contenu à partir du serveur d'origine, ce qui accélère le temps de réponse[A].

Le cache est une couche de stockage de données grande vitesse qui stocke un sous-ensemble de données, généralement transitoires, de sorte que les demandes futures pour ces données sont traitées le plus rapidement possible en accédant à l'emplacement de stockage principal des données[B].
Le serveur cache est pratiquement toujours un serveur proxy (ou serveur de délégation) qui intercepte les demandes Internet et les administre[C].

2.3 Le système de caches distribué

2.3.1 Définition de système de caches distribué

Un système de caches distribué est un système informatique qui stocke des données dans un cache partagé par plusieurs serveurs d'applications[D] .
Les données sont stockées localement dans un stockage temporaire pour une utilisation ultérieure[E] . Les systèmes de caches distribués sont conçus pour améliorer les performances et la disponibilité des applications en stockant des données en mémoire vive[F]. Les données sont stockées dans une couche de stockage de données grande vitesse qui stocke un sous-ensemble de données, généralement transitoires[G].

2.3.2 Objectif de système de caches distribué :

Un système de cache distribué a pour objectif d'améliorer les performances des applications en stockant des données en mémoire, ce qui permet d'accéder plus rapidement aux données et de réduire la charge sur les bases de données[H] [I]. Les systèmes distribués, en général, visent à éliminer les points centraux de défaillance et à partager les ressources matérielles, logicielles ou de données[J]. Le but est de concevoir une gestion des tâches efficace et flexible[K]. Les systèmes de traçage distribué sont conçus pour fonctionner sur une infrastructure de services distribués, ce qui leur permet de suivre plusieurs applications et processus simultanément sur de nombreux nœuds et environnements informatiques concurrents[L].

2.3.3 l'architecture d'un système de caches distribué

L'architecture d'un système de cache distribué peut varier en fonction des besoins et des contraintes du système. Cependant, en général, un système de cache distribué est composé de plusieurs nœuds ou serveurs qui stockent des données localement. Ces nœuds sont interconnectés pour former un cluster et utilisent des protocoles de communication pour échanger des informations sur les données stockées dans le cache.
L'architecture d'un système de cache distribué peut être organisée selon différentes approches, telles que la mise en cache en cluster, la mise en cache distribuée, la mise en cache distribuée hiérarchique, etc. Cependant, l'architecture la plus couramment utilisée est la mise en cache distribuée.
Dans un système de cache distribué, chaque nœud dispose d'un cache local pour stocker les données les plus fréquemment utilisées. Lorsqu'un utilisateur demande des données, le système recherche d'abord les données dans le cache local du nœud. Si les données ne sont pas disponibles localement, le système recherche les données dans les caches distants des autres nœuds.

Pour assurer une cohérence des données, le système de cache distribué utilise des protocoles de mise à jour de la cohérence, tels que les protocoles d'invalidation et de mise à jour à la demande. Ces protocoles permettent de maintenir la cohérence des données en synchronisant les caches locaux et distants .

2.3.4 Les avantages d'un système de caches distribué

Les avantages d'un système de cache distribué sont nombreux :

1. Il permet d'améliorer la performance globale des services en répartissant la charge de travail sur plusieurs serveurs cache, ce qui réduit les temps de réponse et améliore la disponibilité du système[M] ;
2. Peuvent être mis à l'échelle verticalement et horizontalement, ce qui permet d'augmenter la capacité de stockage et de répondre à une demande croissante[N] ;
3. Peuvent également améliorer la fiabilité et la tolérance aux pannes en répartissant les données sur plusieurs serveurs, ce qui réduit les risques de perte de données en cas de panne d'un serveur[O] ;
4. Les systèmes de cache distribué peuvent être plus abordables que les superordinateurs extrêmement chers, car ils utilisent des ordinateurs courants peu coûteux avec des microprocesseurs [P].

2.3.5 Les inconvénients des systèmes de cache distribué

Les systèmes de cache distribué peuvent présenter des inconvénients :

1. **Complexité de mise en œuvre :** la mise en place d'un système de cache distribué peut être complexe et nécessiter des compétences techniques avancées pour assurer une configuration optimale.

2. **Cohérence des données :** lorsqu'il y a plusieurs nœuds de cache, la cohérence des données peut devenir un problème. Il est essentiel de mettre en place des mécanismes de synchronisation pour assurer la cohérence des données entre les différents nœuds.

3. **Coût :** les systèmes de cache distribués peuvent être coûteux à mettre en place et à maintenir, en particulier pour les entreprises qui n'ont pas les ressources techniques nécessaires en interne.
4. **Performance :** bien que les systèmes de cache distribués soient conçus pour améliorer les performances, ils peuvent parfois entraîner des ralentissements si la configuration ou la gestion de la mémoire n'est pas optimale.

5. **Sécurité :** la sécurité des données en cache peut être un inconvénient si elle n'est pas correctement gérée. Les données en cache peuvent être sensibles et doivent être protégées contre les attaques de sécurité.

6. **Délai :** en raison de la communication requise entre les différents nœuds de cache, il peut y avoir une latence ajoutée qui affecte les performances globales du système. Cela peut entraîner des temps de réponse plus longs que ceux observés avec un système de stockage centralisé.

7. **Redondance :** en raison de la réplication de données sur plusieurs nœuds, la mise en cache distribuée peut entraîner une certaine redondance de données. Cela peut entraîner une utilisation accrue de l'espace de stockage et compliquer la gestion de la cohérence des données.

8. **Stockage :** la capacité de stockage disponible dans les nœuds de cache distribués peut être limitée par rapport à un système de stockage centralisé, ce qui peut limiter la quantité de données pouvant être mises en cache.

Chapitre 3

Les stratégies de cache

Sommaire

3.1 Les types de système de cache distribué :

Il existe plusieurs types de systèmes de cache distribué, chacun ayant ses propres caractéristiques et utilisations. Voici quelques-uns des types les plus courants :

3.1.1 Le système de caches en mémoire partagé :

est une technique utilisée dans les ordinateurs multi-processeurs pour améliorer les performances en réduisant le temps d'accès à la mémoire. Il consiste à stocker temporairement les données les plus souvent utilisées dans une mémoire cache partagée entre les différents processeurs, ce qui évite des accès répétés à la mémoire centrale qui est plus lente.
Lorsqu'un processeur a besoin d'accéder à une donnée, il vérifie d'abord si elle se trouve dans la cache partagée, et si c'est le cas, il peut l'obtenir plus rapidement. Si elle n'est pas dans

la cache, il faut aller la chercher dans la mémoire centrale, mais cette opération sera moins fréquente grâce au système de cache en mémoire partagé.
Ce type de système stocke le cache en mémoire partagée, ce qui permet une communication rapide et efficace entre les nœuds de cache. Un exemple de système de cache distribué en mémoire partagée est Hazelcast(33)

3.1.2 Le système de caches basé sur des objets :

est une technique de mise en cache qui stocke les résultats de calculs ou les données les plus souvent utilisées dans un cache sous forme d'objets. Chaque objet est identifié par une clé unique, ce qui permet de retrouver facilement les données en cache lorsque nécessaire. Lorsqu'une application a besoin d'une donnée, elle vérifie d'abord si cette donnée est présente dans le cache.
Si c'est le cas, l'application récupère la donnée à partir du cache plutôt que de la recalculer ou d'aller la chercher dans une source de données externe, ce qui peut être plus rapide. Les systèmes de cache basés sur des objets sont largement utilisés dans les applications Web pour améliorer les performances en réduisant le temps d'accès aux données et en limitant les requêtes vers les bases de données ou d'autres sources de données.
Ce type de système stocke les données en tant qu'objets dans le cache, généralement organisés en une hiérarchie de clés. Un exemple de système de cache distribué basé sur les objets est Redis (22)

3.1.3 Le système de caches basé sur le contenu :

Le système de caches basé sur le contenu est un système de mémoire intermédiaire numérique qui conserve les données consultées en vue d'un accès ultérieur[Q]
. Lors des consultations suivantes, c'est le cache qui répond aux requêtes et il n'est alors pas nécessaire de contacter l'application d'origine. La mise en cache peut être appliquée devant tout type de base de données, y compris relationnelle et NoSQL [R]. La mise en cache de contenu est également utilisée pour accélérer le téléchargement des logiciels distribués par Apple et des données que les utilisateurs stockent dans iCloud [S].

3.1.4 Le système de caches basé sur la répartition :

Le système de caches basé sur la répartition est une technique qui permet d'améliorer les performances et la scalabilité d'un système en copiant les données fréquemment utilisées dans un cache[T] [U].
Cette technique permet d'accéder plus rapidement aux données stockées dans le cache, ce qui peut améliorer les performances des applications qui y accèdent. La mémoire cache est un type de mémoire vive RAM à laquelle le microprocesseur peut accéder plus rapidement qu'à la mémoire RAM habituelle[V]. On peut trouver des caches répartis dans le monde entier, notamment dans le cadre de la pratique du géocaching[W].

3.2 les classes de système de caches distribué :

Les caches distribués peuvent être classés en fonction de différentes caractéristiques, mais voici quelques classes courantes :

3.2.1 Cache en mémoire vive distribuée (Distributed Memory Cache) :

ce type de cache stocke les données en mémoire vive (RAM) répartie sur plusieurs nœuds de calcul pour fournir un accès rapide aux données en mémoire.
Voici un exemple de caches en mémoire vive distribuée (Distributed Memory Cache) :
Memcached : Memcached est un autre système de cache en mémoire vive distribuée open source très populaire qui stocke les données sous forme de paires clé-valeur. Il est conçu pour être très rapide et évolutif (25).

3.2.2 Cache en mémoire distribuée (Distributed Caching) :

données dans un cluster distribué de serveurs, généralement en utilisant des clés/valeurs stockées en mémoire.
Voici un exemple de cette technologie :
Oracle Coherence : Oracle Coherence est un système de cache en mémoire distribué commercial qui permet de stocker des données clés-valeur et des objets. Il utilise une architecture en grille pour fournir une haute disponibilité et une scalabilité élevée. Il est souvent utilisé pour la mise en cache de données dans les applications d'entreprise et les applications de traitement en temps réel (62).

3.2.3 Cache de proxy inversé (Reverse Proxy Cache) :

dans laquelle les caches sont déployés près de l'origine du contenu, au lieu de près des clients. Ceci est une solution attrayante pour les serveurs ou les domaines qui attendent un nombre élevé de demandes et veulent assurer un qualité de service. La mise en cache par proxy inverse est également mécanisme utile pour soutenir les fermes d'hébergement Web (domaines virtuels mappés à un seul site physique), un service de plus en plus commun pour de nombreux fournisseurs de services Internet (FSI).
Notez que le déploiement du cache de proxy inverse est totalement indépendamment de la mise en cache des proxy côté client. En fait, ils peuvent coexister et améliorer collectivement les performances Web les perspectives utilisateur, réseau et serveur.(8)
Voici un exemple de cache de proxy inverse :
Nginx : Nginx est un serveur web open-source qui prend en charge la mise en cache de proxy inversé. Il est souvent utilisé comme proxy inversé pour les sites web à haute disponibilité car il peut être configuré pour équilibrer la charge entre plusieurs serveurs web et pour mettre en cache les ressources statiques. Nginx prend également en charge des fonctionnalités telles que la compression de contenu et la réécriture d'URL (54).

3.2.4 Cache de contenu distribué (Distributed Content Cache) :

ce type de cache stocke des fichiers, des images, des vidéos et d'autres contenus statiques sur plusieurs serveurs pour une distribution efficace du contenu à travers le réseau.
Voici quelques exemples de cette technologie :
Akamai : Akamai est un fournisseur de services de contenu en ligne qui utilise un réseau de serveurs de cache distribués pour stocker du contenu Web, tels que des fichiers vidéo, audio, images et pages Web. Akamai utilise une technologie de mise en cache de contenu distribué pour stocker des copies de contenu populaire sur des serveurs de cache proches de l'utilisateur final (50).
Amazon CloudFront : Amazon CloudFront est un service de diffusion de contenu qui utilise un réseau de serveurs de cache distribués pour stocker des copies de contenu Web populaire. Les utilisateurs peuvent configurer des règles de cache pour spécifier les fichiers à mettre en cache, la durée de conservation du cache et les conditions d'invalidation du cache. Amazon CloudFront prend également en charge la compression de contenu, la réécriture d'URL et la gestion des cookies.
Content Delivery Networks (CDN) : Les CDN sont des réseaux de serveurs de cache distribués qui stockent du contenu Web pour réduire les temps de réponse et améliorer les performances. Les CDN fonctionnent en stockant des copies de contenu sur des serveurs de cache proches de l'utilisateur final, de manière à réduire la distance physique entre l'utilisateur et le serveur de contenu. Les CDN sont utilisés par de nombreux sites Web populaires pour stocker des fichiers de contenu statique, tels que des images, des scripts et des fichiers CSS (77).

3.3 Les réseaux orientés au contenu

Les Réseaux Orientés Contenu ou Content Centric Networking (CCN) constituent une architecture novatrice pour l'Internet du futur. Ils visent à reconstruire la pile de protocoles de l'Internet en éliminant la pile TCP/IP.

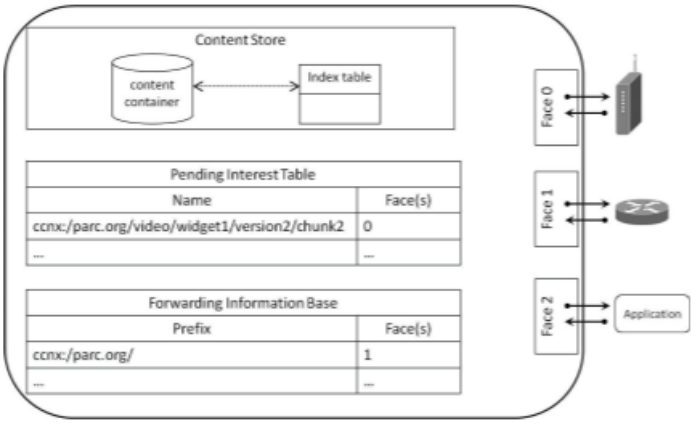


FIGURE 3.1 – La structure d'un nœud CCN

L'architecture CCN repose principalement sur deux fonctions essentielles : "Interest" et "Data". Lorsqu'un utilisateur souhaite accéder à un contenu, il envoie un message "Interest" à travers le réseau. Si un nœud sur le chemin du message détient le contenu demandé, il peut alors y répondre en envoyant un message "Data". Les données sont ensuite transmises au demandeur, et chaque nœud sur le chemin peut les conserver en cache pour les utiliser ultérieurement.
Les messages sont dirigés vers leur destination par des routeurs de contenu (CR) en utilisant un mécanisme d'anycast. Chaque CR dispose de trois structures de données : Forwarding Information Base (FIB), Pending Interest Table (PIT) and Content Store (CS). La FIB contient les noms de contenu associés aux interfaces de sortie, et elle est utilisée pour acheminer les messages d'intérêt vers leur destination. La PIT enregistre les interfaces d'entrée par lesquelles les messages d'intérêt sont reçus, et elle est utilisée pour renvoyer les messages de données contenant le contenu demandé aux nœuds émetteurs. Enfin, le CS stocke le contenu qui passe par le CR, et il sert de cache pour les futures demandes de contenu.
Lorsqu'un message d'intérêt arrive, le CR extrait le nom du contenu demandé et vérifie dans son CS si le contenu est disponible. Si le contenu est trouvé, le CR envoie un message de données contenant le contenu vers l'interface d'où le message d'intérêt a été reçu. Si le contenu n'est pas trouvé, le CR effectue une correspondance de préfixe de nom la plus longue avec le nom dans la table PIT pour déterminer si une réponse est attendue pour une demande similaire. Si une entrée est trouvée, le message d'intérêt est supprimé. Si aucune entrée n'est trouvée, le CR consulte sa FIB pour déterminer l'interface par laquelle le message d'intérêt doit être acheminé.(14) Contrairement à l'architecture actuelle d'Internet, qui dispose de ressources limitées pour le cache et où les serveurs de cache sont situés en des points fixes, CCN est un réseau de caches avec des serveurs de cache répartis partout. De plus, CCN est un réseau (content-aware), c'est-à-dire qu'il est capable de prendre en compte les caractéristiques bernardini2015popularity qui transite sur le réseau(14)
Ces deux caractéristiques de CCN donnent lieu à de nouvelles propriétés :

- Transparence de cache : La transparence de cache permet aux utilisateurs de ne pas être conscients de la présence de caches sur le réseau ;
- Ubiquité de cache : Dans l'architecture actuelle d'Internet, les ressources de cache sont situées à des endroits fixes (70) et la localisation du contenu est déterminée en résolvant un modèle analytique basé sur les demandes antérieures. Dans CCN, les caches sont omniprésents et disponibles partout. Leur localisation n'est plus figée et le contenu doit évoluer en fonction des décisions prises de manière dynamique et adaptée à l'environnement ;
- Granularité fine des contenus en cache. L'unité de paquet dans CCN est le bloc. Un contenu est composé de plusieurs blocs et la même unité est utilisée dans le réseau CCN. Comme tous les paquets sont nommés, toutes les blocs de contenu sont nommés. Le changement d'unité par rapport aux réseaux IP (paquet) fait apparaître le suivant. Dans l'Internet il n'y a pas de schéma commun de nommage des contenus. Si le même contenu est transmis avec différents protocoles les fonctionnalités de cache ne peuvent pas être partagées. Comme il n'y a pas un schéma commun de nommage des contenus, il n'est pas possible de déterminer si deux paquets transmettent le même contenu. CCN est créé avec un schéma commun de nommage des contenus. Le contenu transmis via différents protocoles partage le même nom. Ainsi, différent es logiciels et protocoles

peuvent utiliser des ressources de cache pour fournir le contenu demandé avec un autre protocole que le Protocol d'origine. Cette nouvelle caractéristique permet de réduire le temps pour récupérer un contenu et augmenter la performance des caches.

CCN étant un réseau de cache, l'organisation individuelle des caches devient essentielle. Des stratégies de cache sont necessaires pour organiser et gérer la collection de ressources disponibles dans un réseau CCN. Dans cette thèse, nous elaborons des solutions pour améliorer l'efficacité des caches. Nous sout en ons que meilleure efficacité des caches va entraîner un meilleur comportement de l'ensemble du réseau(14)

3.4 Les stratégies de cache

Les stratégies de mise en cache sont des politiques de gestion. Ils décident quelles informations conserver et où les conserver afin d'atteindre les objectifs généraux du réseau. Ces décisions incluent un large éventail de possibilités, telles que le placement de contenu, la détection de modèles de trafic ou la sélection de types d'informations.

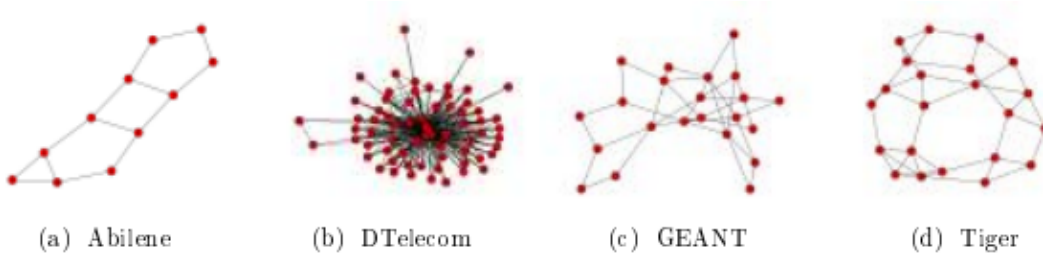


FIGURE 3.2 – Les topologies des fournisseurs d'Interest

Les stratégies de mise en cache fournissent un mécanisme efficace pour gérer le trafic réseau qui est étroitement lié à la distribution globale du contenu. Les stratégies de mise en cache utilisent le stockage en cache pour absorber le trafic en dupliquant le contenu populaire dans des emplacements stratégiques. En réduisant le trafic, on s'attend à une meilleure qualité de service et on peut libérer des ressources réseau qui peuvent être utilisées pour d'autres activités du réseau. Les principales stratégies de cache de la littérature sont résumées dans la liste suivante :

- Leave Copy Everywhere (LCE). Elle consiste à sauvegarder tous les messages de contenu qui passent dans le réseau. Cette stratégie est la plus utilisée et notamment dans CCN (57).
- Leave Copy Down (LCD) (76). Quand un message d'Interest est trouvé dans un cache, la stratégie de cache copie le contenu dans le cache du voisin vers la requête originale.
- Max-Gain In-Network Caching (MAGIC) (55). Cette stratégie de cache cherche à maximiser le bénéfice en mettant un contenu dans un cache, et en même temps à minimiser la perte en replaçant un autre contenu. Les maximisations et minimisation sont calculées avec les informations de popularité.
- ProbCache (53). Probabilist ic In-Network CAching (ProbCache) est une stratégie basée sur la probabilité. Elle distribue le contenu dans un chemin de caches. Avec une

loi de probabilité, ProbCache séêlectionne quel est le meill eur næud pour sauveg ar der un cont enu.

— Cache "Less " For More (17). Cache "Less" For More est une strat égie de cache basée sur une métrique de centralité qui utilise le concept de be tweenness centrality pour améliorer le béaéfice d'introduire un nouvel dément. De plus, elle réduit la redondance par rapport au LCE.

Toutes ces stratégies de cache ont des objectifs différents et elles utilisent différent types d'informations pour réagir. Cache "Less" For More cherche à obtenir des résultats similaires à LCE tandis qu'elle réduit le nombre des opérations de remplacement de contenu dans les caches. MAGIC cherche à maximiser l'efficacité de l'ajout d'un nouveau contenu dans les caches en utilisant la métrique du Cache Hit. Par rapport aux types d'informations, diverses alternatives ont été prises en compte dans la littérature. Cache "Less" For More utilise des informations de topologie lorsque MAGIC et ProbCache utilisent des informations de popularité. Cependant parmi toutes ces stratégies, ce n'est pas évident d'identifier laquelle fonctionne le mieux. Nous affirmons que l'on peut construire des mécanismes alternatifs pour des scénarios qui n'ont encore jamais été considérés.Pour atteindre cet objectif, nous proposons un cadre commun. Dans ce cadre, nous comparons toutes les stratégies de mise en cache et décidons laquelle est la meilleure dans chaque cas.

3.5 étude connexe :

3.5.1 Client Cache (CC) :

Dans la stratégie client-cache (CC), la validité des contenus mis en cache est observée. Le concept de CC est dérivé de la mise en cache centralisée, dans laquelle le contenu est mis en cache sur des routeurs qui sont liés à plus de routeurs (44) L'objectif des CC est d'accroître la validité d'un contenu donné. La validité est mesurée en fonction de la durée de vie du contenu mis en cache sur les routeurs intermédiaires et de l'éditeur. Le contenu est sélectionné comme valide si sa durée de vie chez l'éditeur est supérieure à sa durée de vie qui a été mise en cache sur un routeur intermédiaire.
Dans la figure 6 (scénario Client-Cache), divers intérêts des consommateurs A et B sont envoyés à récupérer le contenu C1. Principalement, la durée de vie du contenu C1, du contenu C2 et du contenu C3 sont indiquées par VC6, VC4 et VC5, respectivement, à la figure 6. Dans CC, la durée de vie du contenu est considérée comme VC, ce qui montre la validité du contenu. Par conséquent, la durée de vie des contenus C1 et C2 est plus élevée chez l'éditeur que chez le routeur R5. Ceci indique que les contenus C1 et C2 doivent être mis en cache sur le routeur R5 ; ainsi, C1 sera mis en cache sur le routeur R5, comme indiqué dans Figure 6.

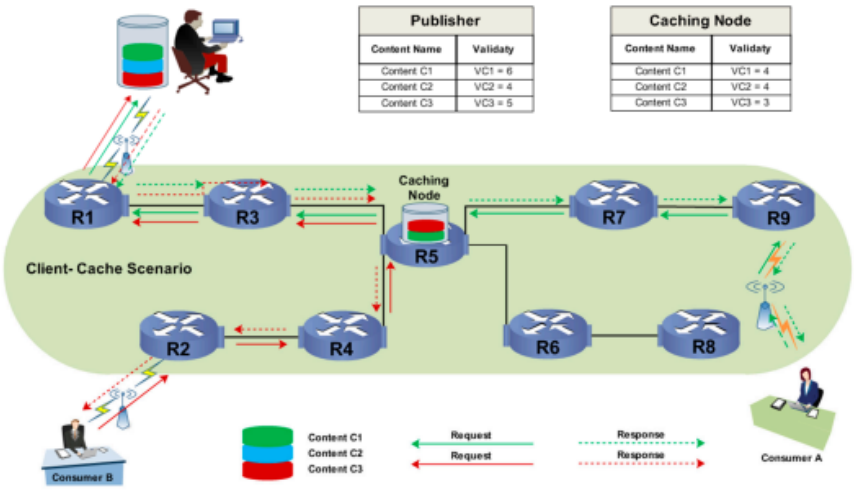


FIGURE 3.3 – Client Cache

3.5.2 Stratégie de mise en cache flexible basée sur la popularité (FlexPop) :

La stratégie de mise en cache FlexPop compile deux mécanismes pour compléter sa mise en cache de contenu procédure (23) Principalement, il effectue une procédure de cache de contenu pour mettre en cache le contenu transmis le long du chemin de routage des données. Il exécute une deuxième procédure d'expulsion de contenu si le contenu diffusé n'identifie pas l'espace de cache libre pour l'hébergement sur les routeurs intermédiaires.
FlexPop nécessite la maintenance d'une table de popularité qui aide à compter le nombre d'intérêts à chaque routeur pour tous les noms de contenu. Sur la base des intérêts reçus, la popularité d'un contenu donné est calculée dans le PT en utilisant le compteur de contenu et la balise de popularité. Initialement, le contenu est stocké dans le PT pour calculer sa popularité. Si le contenu du PT indique que sa popularité est égale ou supérieure au seuil, il le transmet au tableau de comparaison (CT).
Le CT est responsable de la tenue à jour des informations sur le contenu populaire. Il compare la popularité du nouveau contenu avec les popularités du contenu populaire précédent ; si le nouveau contenu démontre une demande plus importante que l'autre contenu, il est étiqueté comme populaire, et le CT est partagé avec les routeurs voisins. Lorsque la popularité de ce contenu atteint un seuil, le contenu est transmis au routeur qui dispose du nombre maximum d'interfaces sortantes à mettre en cache. Si le cache du routeur ayant le maximum d'interfaces sortantes déborde, le contenu est recommandé pour le cache du routeur associé au deuxième plus grand nombre d'interfaces sortantes.
La figure 7 illustre la procédure de mise en cache du contenu dans FlexPop. Initialement, deux contenus, C2 et C3, sont mis en cache sur le routeur R5. Le routeur R5 est associé aux interfaces sortantes maximales, et seulement deux morceaux de contenu peuvent résider dans son cache en raison de sa capacité limitée. Trois intérêts des consommateurs A et B sont envoyés au routeur R2 pour récupérer le contenu C2.
En réponse aux intérêts reçus, le routeur R2 devient le fournisseur et envoie le contenu C1 aux consommateurs A et B. En même temps, la popularité du contenu C1 est mesurée sur la

base des intérêts reçus pour le contenu C1. Selon FlexPop, C1 gagne en popularité, comme le montre le TC dans la figure 7 ; par conséquent, il est étiqueté « populaire » et recommandé pour la mise en cache au niveau du routeur avec le nombre maximum d'interfaces sortantes (par exemple, routeur tR5). Cependant, il n'y a pas d'espace libre au routeur R5 pour mettre en cache le contenu C1 ; par conséquent, il sera mis en cache au routeur ayant le deuxième plus grand nombre d'interfaces sortantes. Ainsi, C1 sera mis en cache sur les routeurs R4 et R6.

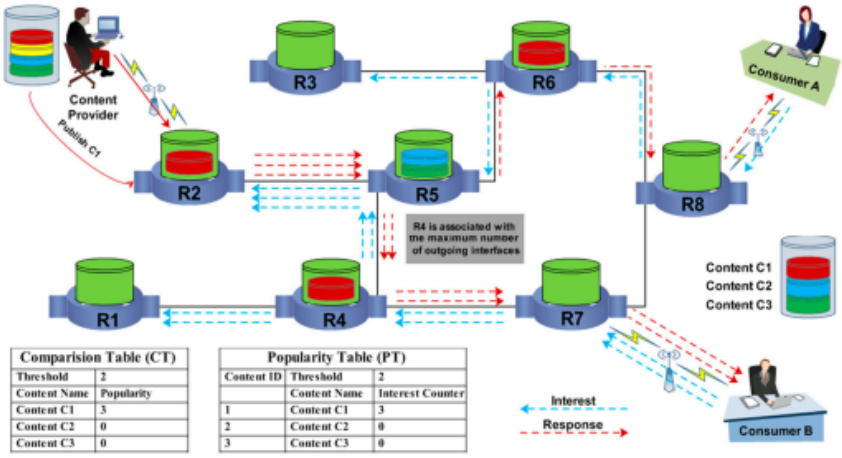


FIGURE 3.4 – Stratégie de mise en cache flexible basée sur la popularité (FlexPop)

3.5.3 Stratégie de mise en cache centralisée (CCS)

Ce mécanisme de mise en cache de contenu nécessite deux approches. Premièrement, il détermine l'intervalle noeud de centralité en calculant les liens associés à chaque noeud. Deuxièmement, il décide comment mettre en cache le contenu transmis le long du chemin de routage des données (74) Dans ce mécanisme de mise en cache, l'intéressant le contenu est transmis au nœud qui a le nombre maximum d'interfaces sortantes ou le maximum nombre de chemins associés. Si un noeud est associé à un nombre élevé de chemins de routage de données, il a plus d'occasions de mettre en cache le contenu diffusé (36) La figure 8 illustre le contenu mécanisme de mise en cache utilisant la mise en cache centralisée dans laquelle les consommateurs A, B et C sont associés avec les routeurs R4, R7 et R9, respectivement. Ces consommateurs ont envoyé trois intérêts pour récupérer le contenu C1, que ce contenu est déjà publié dans le réseau par le fournisseur de contenu (P). Comme les intérêts de contenu C1 atteindre routeur R3, le contenu requis est obtenu. Par conséquent, routeur R3 agit comme un fournisseur et transmet le contenu C1 aux consommateurs intéressés (c.-à-d. A, B et C). Pendant la transmission du contenu, chaque routeur calcule le nombre de chemins de routage de données qui lui sont associés. Selon le nature de mise en cache du CCS, le routeur R6 est sélectionné comme routeur intermédiaire de centralité car il a la le plus grand nombre de chemins associés le long du chemin de livraison des données entre le fournisseur et les consommateurs. Par conséquent, le contenu C1 sera mis en cache à R5.

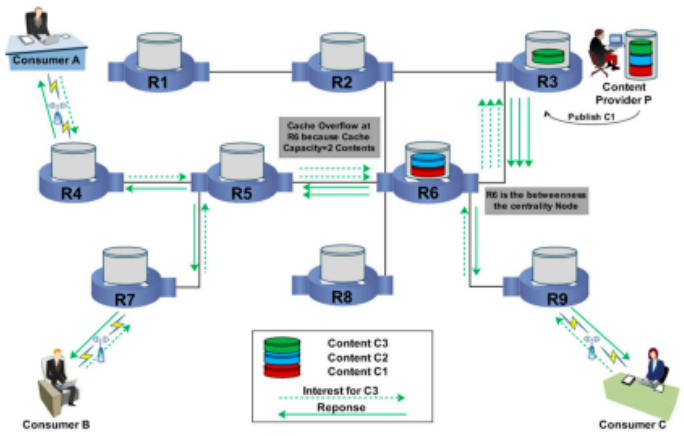


FIGURE 3.5 – Stratégie de mise en cache centralisée (CCS)

3.6 Environnement Commun de Simulation

Un scénario commun d'évaluation a été défini. Il utilise les paramètres les plus utilisés dans la littérature. Par rapport à la topologie, l'architecture de l'Internet du futur est construite sur une architecture CCN. Cette nouvelle architecture de l'Internet sera organisée avec des fournisseurs Internet et son infrastructure sera probablement proche de l'actuelle. Les topologies de fournisseurs d'Internet sont donc les meilleures alternatives pour évaluer les stratégies. De toutes les topologies disponibles, quatre topologies de fournisseurs d'Internet : Abilene, DTelecom, GEANT et Tiger. Elles sont présentées dans le Figure 2 (57).

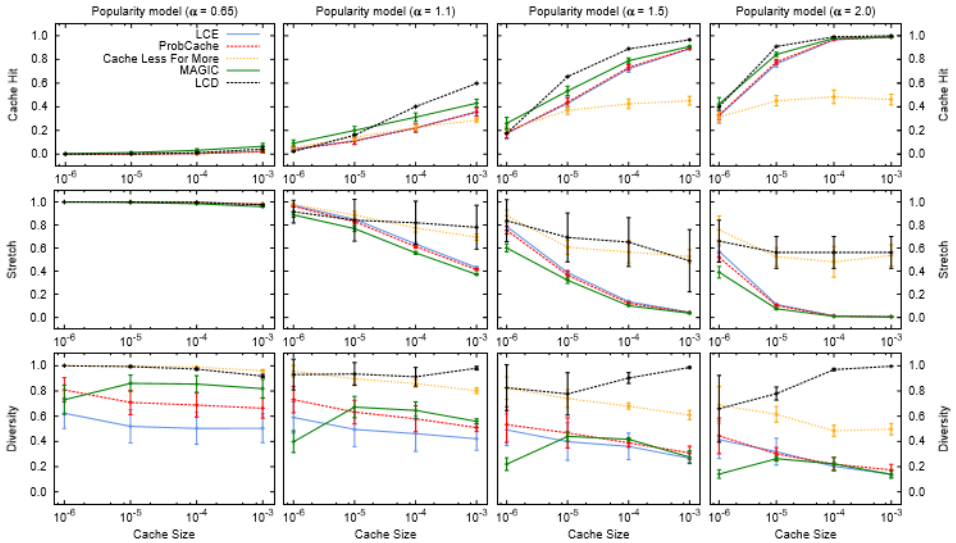


FIGURE 3.6 – Comparaison de stratégies de Cache

Par rapport au modèle de popularité de contenu, des modèles de popularité basés sur la loi de probabilité MZipf ont été utilisés. Le paramètre a de MZipf varie dans la littérature

parmi les valeurs 0.65, 1.1, 1.5 et 2.0. La valeur 0.65 se réfère à un scénario avec très peu de contenus populaires et ce scénario ressemble à une loi de probabilité uniforme.. La probabilité de choisir un contenu est similaire à une loi de probabilité uniforme. La valeur 1.1 se réfère à un scénario avec de la popularité normale. Quand nous parlons de popularité normale, nous nous référons à une loi MZipf sur laquelle 90% de requêtes demandent le 60% des contenus. Finalement, deux autres scénarios avec une forte popularité égale à 1.5 et 2.0 sont considérés. Dans la reste de cette thèse, ces scénarios sont référencés comme scénarios de base, normal et à haute popularité.
Par rapport à la taille du catalogue de contenu, un catalogue de 10^4 contenus a été le modèle le plus utilisé dans la littérature. Cependant, ce volume est très petit et il représente un catalogue de contenus qui ne ressemble pas à la taille du catalogue de contenus de l'Internet. Un catalogue de 10^6 contenus a donc été considéré.
La taille du cache va changer de 1 jusqu'à 1000 contenus ; ce qui correspond à une facteur de réduction de 10^{-6} jusqu'à 10^{-3} par rapport au catalogue. Les tailles de caches sont considérées comme uniformes (tous les noeuds ont la même taille de cache). La politique de remplacement (CRP) utilisée dans la comparaison est Last Recently Used (LRU). Rosenweig et al (56) montrent que les CRPs peuvent être groupées dans des groupes d'équivalence,i.e., différentes CRPs vont exhibir la même performance.

3.7 Les métriques d'évaluation

Les stratégies de cache vont être comparées entre elles suivant les métriques suivantes : Cache Hit, Stretch ,Diversity,Cache Evictions,La redondance de contenu, le délai (le temps de réponse),Consommation de mémoire, et Eviction Ratio.
Cache Hit :
Un Cache Hit est la probabilité que le contenu apparaisse dans le cache le long du chemin. Pour chaque contenu trouvé dans le cache, nous considérons soit un succès, soit un échec. Les stratégies visent à augmenter le nombre d'opérations réussies. Cache Hit Voir l'équation 1 pour plus de détails :$hits_i$ and $miss_i$ se fait référence au nombre de contenu demandé par le message d'interest trouvé ou non présent dans le cache.

$$CacheHit = \frac{\sum_{i=1}^{N} hits_i}{\sum_{i=1}^{|N|} hits_i + \sum_{i=1}^{N} miss_i} \tag{3.1}$$

La figure 7 et la figure 8 montrent les effets du rapport de frappe du cache sur les topologies Abilene et GEANT en utilisant différents modèles de popularité du contenu. Parmi les chiffres donnés, la stratégie de mise en cache DCS a mieux fonctionné en termes de rapport de coup de cache avec les deux topologies de contenu, parce que DCS tente d'améliorer l'allocation de cache de contenus populaires. En outre, DCS cache le contenu le plus populaire sur le routeur périphérique et les routeurs de proximité centrality. Par conséquent, les intérêts ultérieurs sont satisfaits des routeurs périphériques, plutôt que du routeur distant.

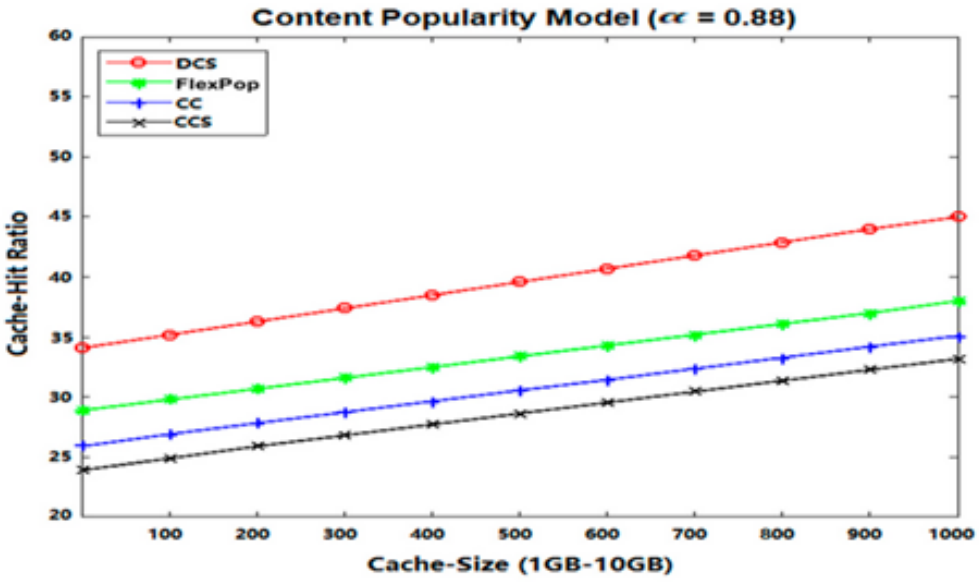


FIGURE 3.7 – Cache Hit Ratio on Abilene Topology with UGC

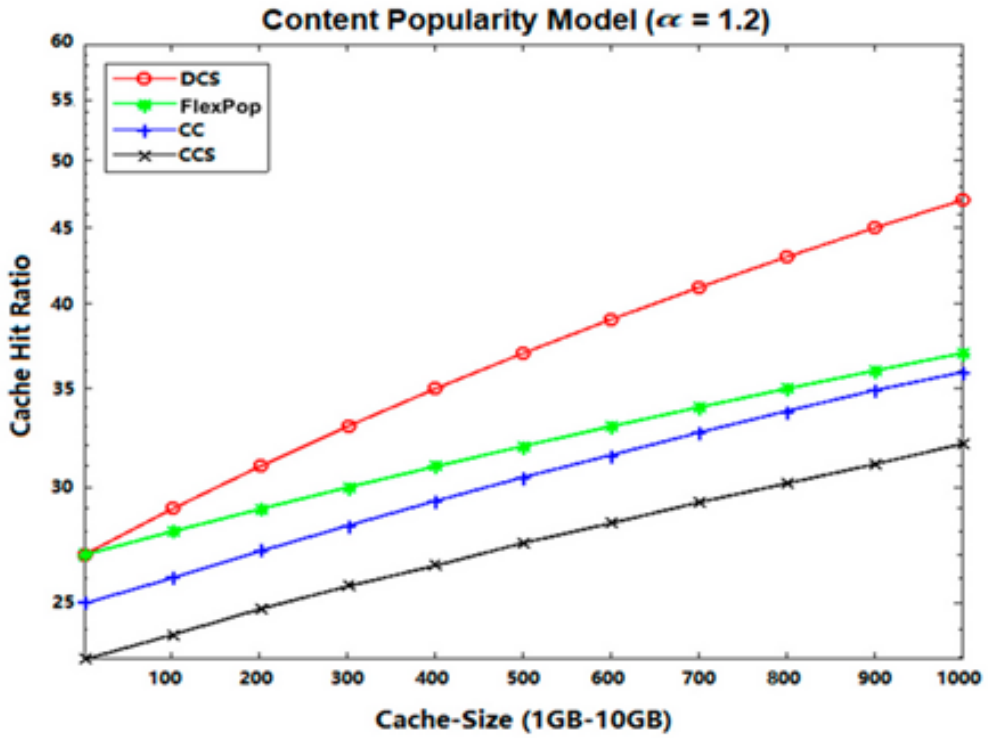


FIGURE 3.8 – Cache Hit Ratio on GEANT Topology with VoD

Si un intérêt ne peut pas être servi par le routeur périphérique, il est satisfait de la proximité routeur centrality. En attendant, l'approche CCS ne définit aucun critère pour gérer le contenu populaire lorsque le cache du routeur de centralité est plein. Par conséquent, tous les

intérêts devaient être transmis à la source de données principale (ou routeur distant), ce qui augmente la longueur du chemin et diminue le taux de succès du cache. Comparativement à l'approche CSC, les approches CC et FlexPop se sont mieux comportées. Cependant, les deux stratégies produisent un faible taux de succès, parce que moins de contenu est hébergé entre les routeurs de centralité. D'autre part, DCS cache le contenu en morceaux pour augmenter la disponibilité de l'espace de stockage au routeur central. Par conséquent, nous concluons que la stratégie proposée des SIR a donné de bien meilleurs résultats en cachant le contenu près des consommateurs à la périphérie du réseau(45)

Stretch :

Le Stretch est également une métrique très populaire pour évaluer les stratégies de mise en cache (57) (55) (53) (17) (58) (30). Le Stretch fait référence au pourcentage du chemin parcouru pour récupérer le contenu. Le $hops - walked_i$ décrit le nombre de sauts entre le nœud émettant la requête i et le nœud répondant à la requête avec son contenu mis en cache. $total_h$ ops_idécrit le nombre de sauts depuis le nœud envoyant la requête i jusqu'au nœud fournissant le contenu. Si la valeur tend vers 0, les demandes sont satisfaites et les performances du réseau sont meilleures en raison du contenu à proximité de l'utilisateur.

$$Stretch = \frac{\sum_{i=1}^{|R|} hops - walked_i}{\sum_{i=1}^{|R|} total - hops_i} \tag{3.2}$$

Comme la capacité de cache est faible par rapport au contenu diffusant, moins de contenu peut être hébergé dans les routeurs de centralité. En outre, CCS cache tout le contenu sans tenir compte de leur popularité ; ainsi, les contenus les plus populaires ont moins de chances d'être mis en cache à la position de centralité entre les deux en raison de l'indisponibilité d'un module de popularité. Par conséquent, la performance globale est réduite en termes d'étirement, parce que tous les intérêts pour les contenus les plus populaires doivent être transmis au fournisseur à distance, augmentant ainsi la longueur du chemin entre le consommateur et le fournisseur.(45) La longueur du chemin est augmentée pour chaque intérêt et réponse. En même temps, le cache CC et FlexPop offre la possibilité d'accueillir des contenus populaires à des endroits intermédiaires pendant une période spécifique, ce qui peut réduire la longueur du chemin entre les consommateurs et les fournisseurs.

La raison en est que la plupart des intérêts sont satisfaits des positions centrales. Cependant, ces stratégies offrent la possibilité de stocker des contenus populaires, mais en raison de la capacité limitée du cache au routeur central entre les deux, CC et FlexPop ne peuvent pas obtenir de meilleurs résultats en termes d'étirement, parce que les deux stratégies sont utilisées pour mettre en cache des contenus moins populaires en raison de leurs petits seuils. D'autre part, le SCD cache le contenu dans un format fragmenté, augmentant ainsi la possibilité d'accueillir plus de contenu.(45) Par conséquent, la plupart des intérêts entrants sont satisfaits de l'emplacement central. De plus, le SCD obtient de meilleurs résultats en ce qui concerne la réduction de l'étirement du parcours, car il permet de stocker du contenu près des consommateurs. En outre, il étend la capacité de cache pour stocker le cache de niveau de morceaux de contenu populaire qui est utilisé pour augmenter l'espace disponible pour le nouveau contenu populaire. De plus, le DCS cache le contenu populaire sur les routeurs de bordure, réduisant ainsi l'écart entre les consommateurs et les fournisseurs ; par conséquent, la stratégie de mise en cache proposée offre de bien meilleurs résultats en termes de réduction

du ratio d'étirement global. À la figure 9 et à la figure 10, les résultats indiquant que l'ECL fonctionne mieux que l'ESCC, le CC et le FlexPop sont clairement indiqués.(45)

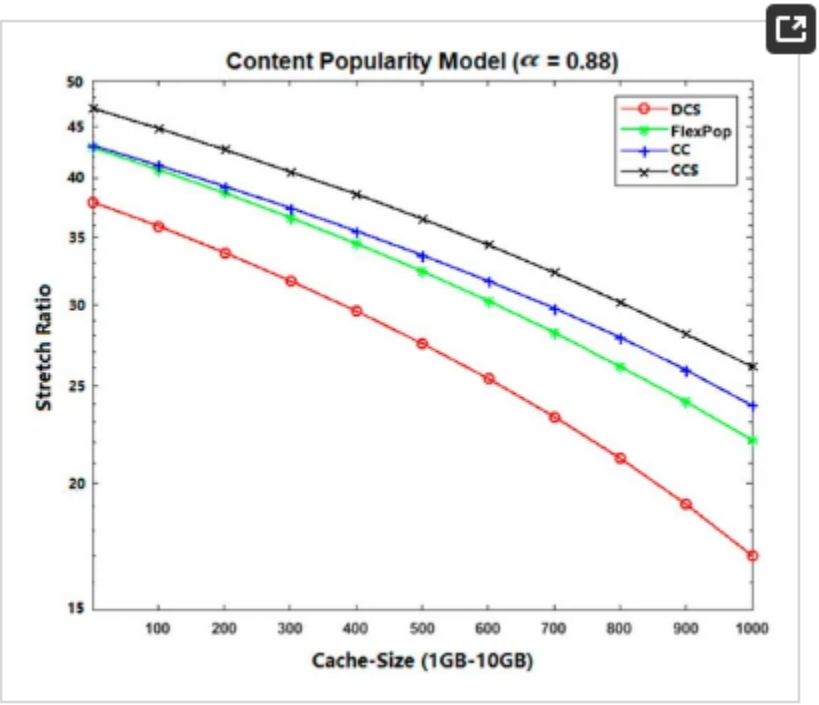


FIGURE 3.9 – Stretch sur la topologie d'Abilene avec UGC

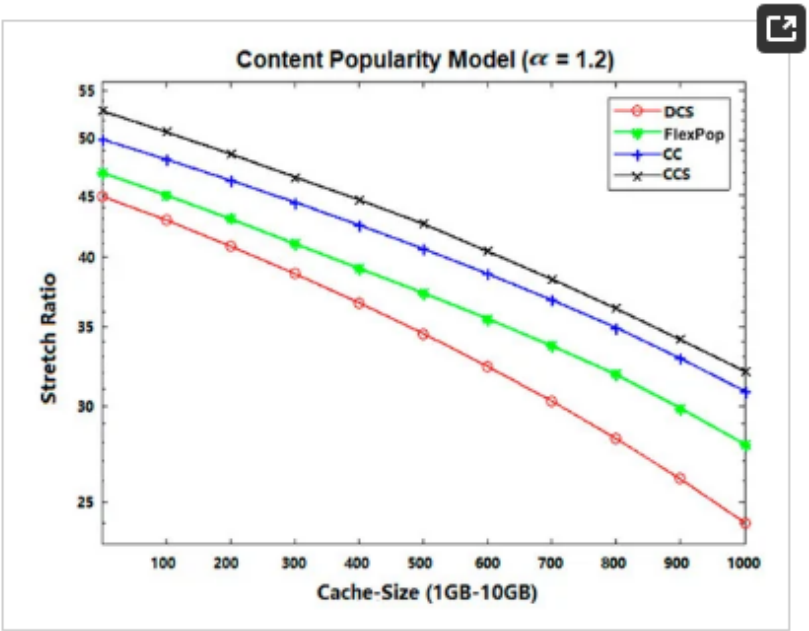


FIGURE 3.10 – Stretch sur la topologie d'GEANT avec VoD

Diversité :

La diversité représente la proportion de contenus différents stockés dans le cache. La diversité du contenu du cache est illustrée dans l'équation 3. La valeur de diversité va de $1/(|N|)$ à 1 : lorsque la valeur tend vers$1/(|N|)$, cela signifie que le réseau de cache contient plusieurs copies du même contenu ; sinon (c'est-à-dire : la valeur tend vers 1), cela signifie que les caches du réseau contiennent un contenu différent..

$$Diversit = \frac{len(U_{n=1}^{|N|} C_n)}{\sum_{n=1}^{|N|} len(C_n)} \tag{3.3}$$

Cache Evictions :
Cache Evictions compte le nombre de fois que la politique de remplacement a été invoquée (équation 3). Dans notre article, nous utilisons le rapport des éléments mis en cache (équation 3). Il est défini comme le rapport relatif des taux d'éviction du cache entre deux stratégies de mise en cache.

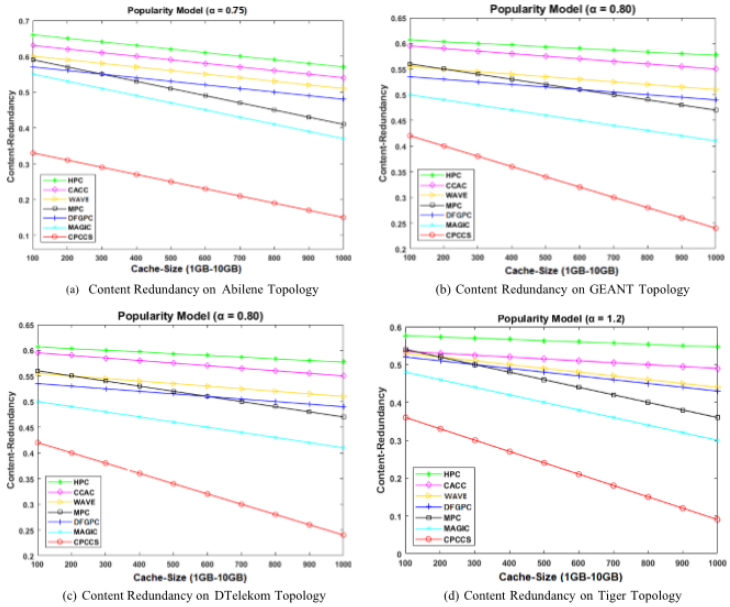
$$CacheEvictions = \sum_{i=1}^{|N|} o_i \tag{3.4}$$

$$RadioofCachedElements = \frac{CacheEvictionsforStrategy1}{cacheEvictionsforStrategy2} \tag{3.5}$$

La redondance de contenu :
La redondance de contenu est la fréquence du contenu indique la quantité de contenu excédentaire dans plusieurs emplacements dans le même réseau (47) Peut En utilisant la formule suivante :

$$Redondance = \sum_{i=1}^{n} Rc_i \tag{3.6}$$

où Rc_i montre la redondance donnée par le ie article du contenu en cache. La figure 11 (a, b, c, d) illustre le contenu-redondances sur Abilene, GEANT, DTelekon et Tiger topologies.



(a) Content Redundancy on Abilene Topology (b) Content Redundancy on GEANT Topology

(c) Content Redundancy on DTelekom Topology (d) Content Redundancy on Tiger Topology

FIGURE 3.11 – Résultats de simulation de la redondance de contenu

Selon les résultats de simulation donnés :

HPC et CACC : montrent des redondances de contenu plus élevées parce que les deux stratégies cachent tout le contenu à tous les routeurs sur le même chemin qui augmente la quantité élevée de similaire duplications de contenu.
MPC, WAVE et DFGPC : montrent moins répétitions de contenus similaires à HPC et CACC car MPC cache le contenu sur les routeurs voisins uniquement.
WAVE :

- effectue des opérations de mise en cache après avoir reçu un grand nombre de Intérêts ;
- consomme plus de temps pour les caches contenu à tous les routeurs parce qu'il fournit le contenu en forme de morceaux distribués qui pousse progressivement le contenu vers les consommateurs.

MAGIC montre de faibles réplications d'homo-contenu geneous parce qu'il permet au contenu d'être mis en cache à routeurs limités. Par conséquent, il montre de meilleures performances en termes de redondance.
DFGPC tente à plusieurs routeurs pour différents intervalles de temps. Par conséquent, les intérêts ultérieurs sont satisfaits de ces routeurs à une certaine mesure qui augmente les opérations de cache redondantes par des contenus similaires.
La valeur optimale de HPC, CACC, MPC, WAVE, DFGPC, MAGIC, et DFGPC est lorsque la redondance de contenu entre 0.6 et 0.7 dans le modéle de popularité ($\alpha = 0.75$) **dans la topologie Abilene .**
CPCCS est plus performant en termes de redondance opérations de cache que les autres stratégies en raison de son nature de la mise en cache à des emplacements partiels le long du le chemin de routage des données.
La valeur optimale de CPCCS est lorsque la redondance de contenu entre 0.4 et

0.45 dans le modéle de popularité ($\alpha = 0.80$) **dans la topologie GEANT et DTelekom.**
le délai (le temps de réponse) :
Le temps de réponse dans la mise en cache peut être significativement réduit par rapport à un accès direct aux données d'origine, car les données les plus souvent demandées sont stockées dans la mémoire cache qui est plus rapide à accéder que le stockage principal.

Si une donnée est déjà en cache, le temps de réponse sera très rapide, car il n'y aura pas besoin d'accéder au stockage principal. Si la donnée n'est pas en cache, alors le temps de réponse sera plus long, car il faudra accéder au stockage principal pour récupérer la donnée, puis la stocker en cache pour les accès futurs.

En général, une mémoire cache plus grande et plus rapide peut améliorer le temps de réponse, car elle peut stocker plus de données fréquemment utilisées et les accéder plus rapidement. Un algorithme de mise en cache efficace peut également améliorer le temps de réponse en stockant les données les plus fréquemment utilisées dans la mémoire cache et en remplaçant les données moins utilisées.
Le temps de réponse, ou délai, dans la mise en cache dépend de plusieurs facteurs, tels que :

- la taille de la mémoire cache ;
- la fréquence d'accès aux données mises en cache ;
- la vitesse de la mémoire cache ;
- la complexité de l'algorithme de mise en cache utilisé.

En général, plus la taille de la mémoire cache est grande, plus le temps de réponse sera rapide, car il y aura plus de données disponibles en cache. De même, plus la fréquence d'accès aux données mises en cache est élevée, plus le temps de réponse sera rapide, car les données seront plus souvent disponibles en cache.
La vitesse de la mémoire cache joue également un rôle important dans le temps de réponse, car une mémoire cache plus rapide permettra d'accéder aux données plus rapidement.
Enfin, l'algorithme de mise en cache utilisé peut affecter le temps de réponse, car certains algorithmes sont plus efficaces que d'autres pour déterminer quelles données mettre en cache. Un algorithme de mise en cache bien conçu peut réduire les temps de réponse en plaçant les données les plus souvent demandées dans la mémoire cache.
le temps de réponse dans la mise en cache dépend de plusieurs facteurs et peut varier en fonction de la taille de la mémoire cache, de la fréquence d'accès aux données, de la vitesse de la mémoire cache et de l'algorithme de mise en cache utilisé.
Consommation de mémoire :
La consommation de mémoire indique la quantité de contenu transmis qui peut être mise en cache lors du téléchargement du chemin de données pour un intervalle de temps particulier (6) Les consommateurs peuvent télécharger le contenu de plusieurs routeurs. Dans le ICN, la consommation de mémoire peut être précisée comme terme de capacité, qui indique le volume utilisé par l'intérêt et le contenu des données. Il peut être calculé à l'aide de l'équation suivante :

$$Consommationdemémoire := \frac{U_m}{T_m} * 100 \tag{3.7}$$

où U_m affiche la mémoire utilisée par le contenu en cache et T_m présente la mémoire totale (stockage en cache) du routeur le long du chemin de livraison des données.
Le DCS est plus performant que le CCS, le CC et le FlexPop en termes de consommation de mémoire car il offre la possibilité de mettre en cache le contenu au niveau des morceaux, réduisant ainsi l'utilisation de la mémoire et la congestion dans les liens de chemin. De plus, il offre le contenu le plus populaire auprès des consommateurs, réduisant le trafic de données et permettant aux contenus de circuler librement sur le réseau. FlexPop et CC offrent de mauvaises performances en termes de consommation de mémoire en raison de leur mise en cache de contenu populaire uniquement sur un routeur central, un processus qui augmente la congestion du trafic dans la capacité de cache limitée.(45) Le CCS cache tout le contenu à la position centrale intermédiaire sans tenir compte de la popularité du contenu, maximisant ainsi la consommation de mémoire. La figure 12 et la figure 13 montrent les résultats de la simulation sur la consommation de mémoire en utilisant deux topologies différentes (Abilene et GEANT). À partir de ces chiffres, on peut voir que la stratégie de mise en cache du DCS proposée donne de bien meilleurs résultats que FlexPop, CC et CCS. Ainsi, nous pouvons conclure que le DCS est meilleur pour améliorer la performance globale du cache ICN en termes de consommation de mémoire efficace.(45)

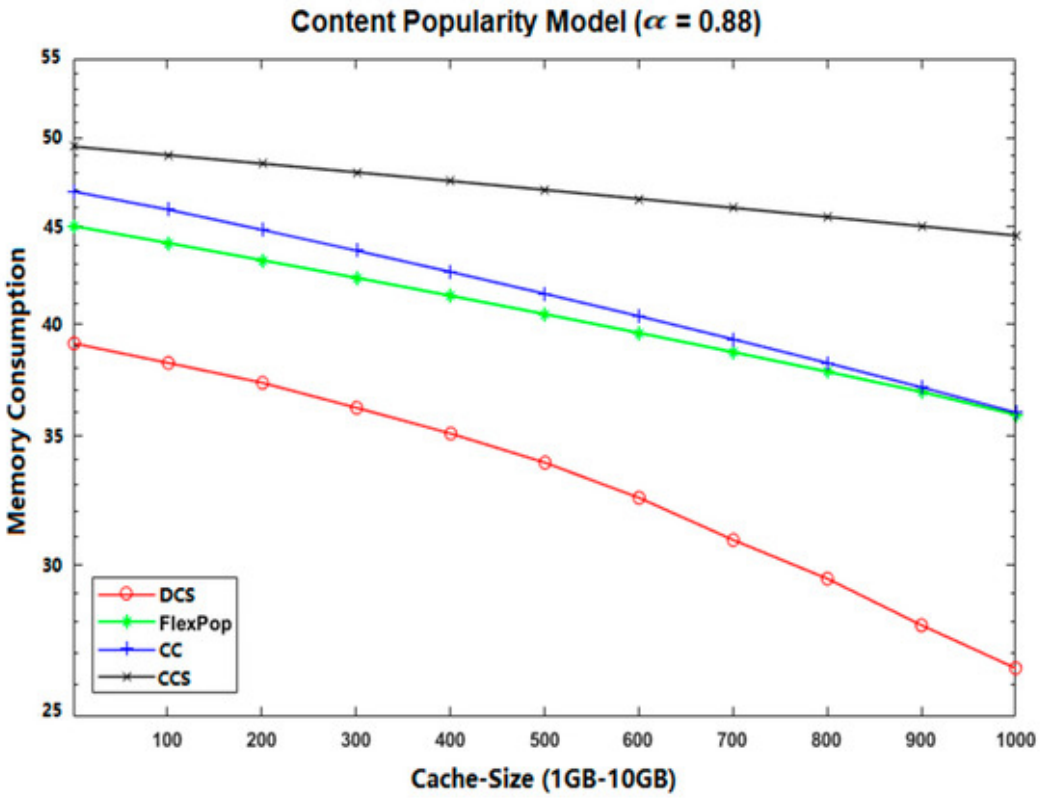


FIGURE 3.12 – Consommation de mémoire sur la topologie Abilene avec UGC

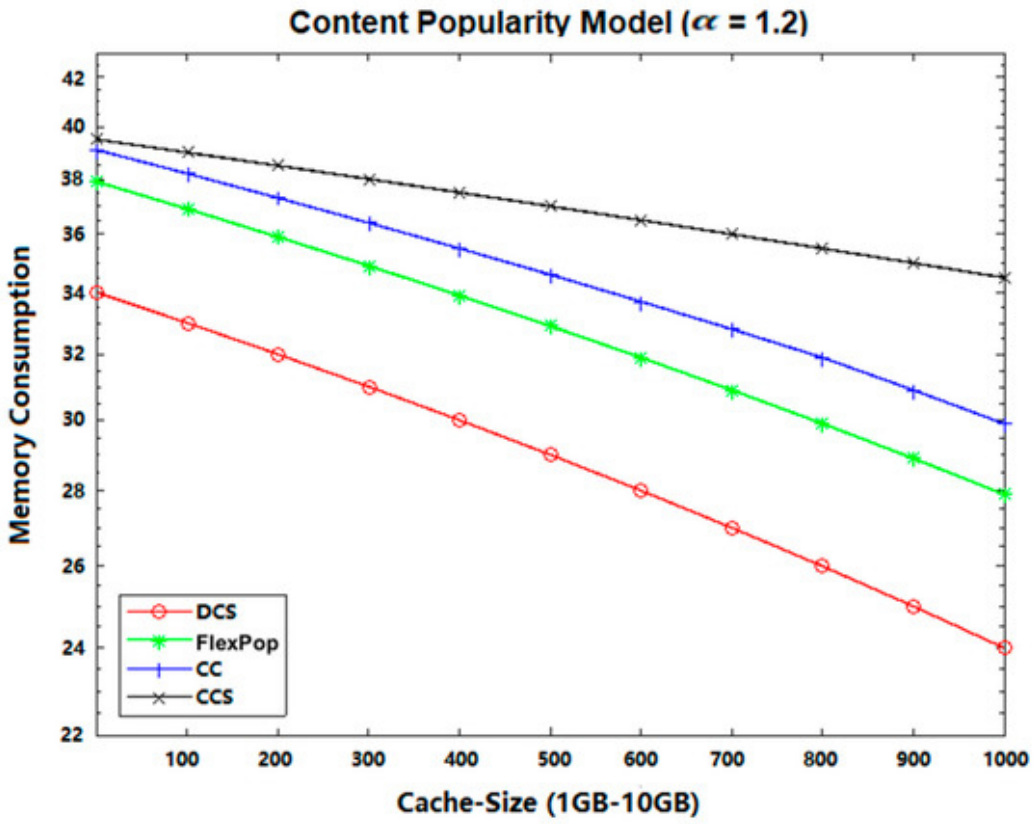


FIGURE 3.13 – Consommation de mémoire sur GEANT Topologie avec VoD.

Eviction Ratio Eviction de contenu est également l'un des paramètres importants pour mesurer le rendement de l'architecture du RCI fondée sur la mise en cache. Il peut être défini comme lorsque le cache d'un nœud réseau devient saturé et qu'il est nécessaire de supprimer du contenu pour accommoder le nouveau contenu. Il peut être calculé à l'aide de l'équation 8 suivante :

$$Evictionratio = \frac{evictedcontent}{totalcontent} \tag{3.8}$$

Le dernier nombre d'expulsions de contenu perturbe le débit du réseau et réduit les rapports de frappe et d'étirement du cache. La raison en est que tous les intérêts entrants doivent être transmis à la source distante pour télécharger le contenu approprié en raison d'un nombre excessif d'expulsions de contenu populaire. La figure 14 et la figure 15 illustrent les résultats générés par les comparaisons des stratégies de mise en cache centralisée. Dans la figure donnée, nous pouvons voir que le CCS montre un ratio d'expulsion de contenu élevé, parce que le CCS cache généralement tous les contenus sans tenir compte de leur popularité, et donc, tous les intérêts arrivant doivent être transmis au fournisseur à distance(45)

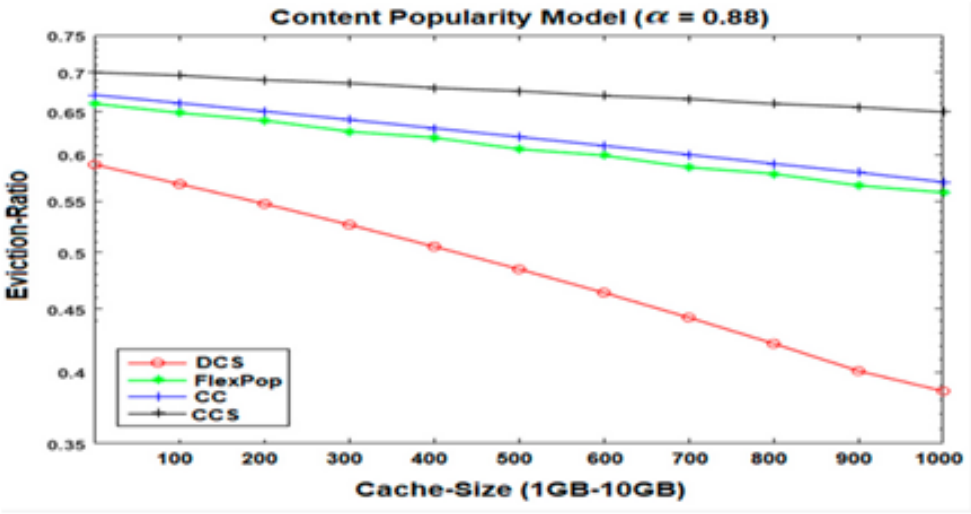


FIGURE 3.14 – Content Eviction Ratio on Abilene Topology with UGC

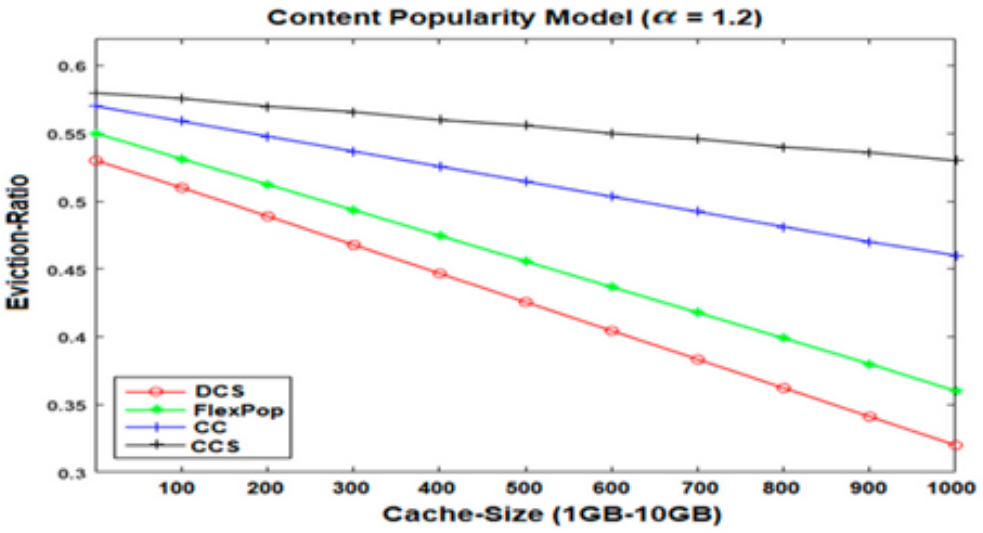


FIGURE 3.15 – Content Eviction Ratio on GEANT Topology with VoD

CC et FlexPop semblent afficher de meilleures performances en termes de ratio d'expulsion de contenu, car les deux stratégies sont utilisées pour mettre en cache le contenu populaire sur les routeurs de centralité. Cependant, en raison de seuils faibles et statiques, ces stratégies de mise en cache cachent également les contenus les moins populaires, provoquant un nombre élevé d'expulsions de contenu. D'autre part, la stratégie de mise en cache proposée par le SCD a donné de meilleurs résultats en termes de réduction du ratio d'expulsion de contenu par rapport à la stratégie de mise en cache CC, CCS et FlexPop. La raison en est que le SCD distribue et cache le contenu dans un format de blocs qui augmente le stockage global du cache pour tenir compte du nouveau contenu. En outre, il utilise pour mettre en cache sur le contenu le plus populaire sur les routeurs de centralité qui augmentent la disponibilité de cache gratuit pour fournir du contenu populaire. En outre, DCS cache le contenu le moins

populaire aux routeurs de bord, et donc, les intérêts ultérieurs sont accomplis à partir des routeurs les plus proches. Ainsi, DCS minimise le ratio d'éviction de contenu en cachant le contenu le moins populaire aux routeurs de bord et le contenu le plus populaire aux routeurs de centralité(45)

3.8 Evaluation

Dans la première ligne de la Figure 2, nous comparons la Cache Hit des stratégies de cache dans les différents scénarios de popularité ($\alpha = 0.65, \alpha = 1.1, \alpha = 1.5 and \alpha = 2.0$). Une attention particulière doit être portée aux petits caches. Le plus grand est le cache, le plus facile d'avoir des meilleurs résultats. Cependant, le but est en ayant le minimum de ressources (taille de cache) à améliorer la performance des caches. Par exemple, lorsque l'on considère un catalogue de contenus, un cache de contenus se réfère à un cache de dix millions de contenus. Lorsqu'une taille de cache de contenus se réfère à dix milliards de contenus. Par rapport à la Cache Hit, MAGIC et LCD montrent les meilleurs résultats. .
Dans les scénarios avec une basse diversité, MAGIC offre des meilleurs résultats avec des petites caches. Lorsque la taille du tampon augmente, MAGIC est dépassé par LCD. LCE et ProbCache présentent quasiment le même niveau de performances. Cependant, ProbCache est meilleur que LCE car il appelle moins des opérations que la stratégie de remplacement.
La stratégie Cache « Less » For More a montré des résultats différents par rapport aux autres stratégies. Dans les scénarios avec une basse popularité ($\alpha = 0.65 and \alpha = 1.1$) notamment pour les petits caches (10^{-6}et 10^{-5}) Cache « Less » For More obtient le même niveau de performance que les autres stratégies. Cependant, avec une haute popularité, ces résultats ne sont pas bons du tout et toutes les autres stratégies de cache montrent des meilleurs résultats ($\alpha = 1.5 and \alpha = 2.0$).
MAGIC et LCD ont montré les meilleurs résultats de performances de Cache Hit. Nos résultats ont montré que LCD devrait être utilisé comme remplaçant de LCE, la stratégie de cache par défaut en CCN. Si on devait choisir entre LCE et ProbCache, ProbCache est une meilleure option dû au nombre moins élevé des opérations de cache faites dans le réseau. Finalement, on considère que Cache « Less » For More est utile dans les scénarios avec popularité basse et normal et avec un cache plus petit que 10^{-5} contenus(14)

Chapitre 4

Modélisation et conception de systéme

Sommaire

4.1 Introduction

Un système de cache distribué est une infrastructure informatique qui permet de stocker temporairement des données fréquemment utilisées en mémoire cache sur plusieurs nœuds ou serveurs différents, plutôt que sur une seule machine.
Les applications informatiques modernes ont de plus en plus recours aux systèmes de cache distribué pour améliorer leurs performances et réduire le temps d'accès aux données. Cependant, le stockage de données dans ces systèmes peut engendrer des conflits d'accès et une utilisation inefficace de la mémoire cache, ce qui peut avoir un impact négatif sur les performances globales du système.
Pour remédier à ce problème, dans ce chapitre, nous proposons de modéliser le système de cache distribué par un graphe pour ensuite appliquer le fameux algorithme de coloration afin d'optimiser l'exploitation de l'espace de stockage d'un coté, et améliorer le temps d'accès aux données d'un autre coté. La solution proposée permettrait l'attribution optimale des données aux nœuds du système de cache distribué.
Le problème de coloration de graphe consiste à assigner à chaque sommet une couleur de sorte que deux sommets adjacents n'aient pas la même couleur, tout en utilisant un nombre minimal de couleurs.
La coloration de graphe est très utile dans une longue liste des applications pratiques. Par exemple, lorsque l'on veut optimiser l'affectation de ressources à plusieurs utilisateurs avec la contrainte qu'une ressource ne peut être partagée entre plusieurs utilisateurs, il suffit de résoudre le problème de coloration pour le graphe dont les sommets représentent les utilisateurs et où deux sommets sont adjacents s'ils demandent la même ressource simultanément.

Aussi, l'allocation des bandes de fréquences dans un réseau cellulaire où des cellules voisines ne doivent pas avoir la même bande de fréquences afin d'éviter les interférences. Il suffit de résoudre le problème de coloration pour le graphe dont les sommets représentent les cellules et les bandes de fréquences sont les couleurs. Un autre exemple de problème est la planification des examens dans une université de sorte qu'un étudiant n'ait pas deux examens au même moment. La aussi, il suffit de résoudre le problème de coloration pour le graphe dont les sommets représentent les examens et les couleurs représentent les horaires.
L'objectif principale d'utiliser l'algorithme de coloration de graphe dans notre système est chaque copie de données à un nœud (ou site) de manière à minimiser les délais d'accès et à maximiser l'utilisation de la mémoire cache, ce qui améliore les performances globales du système. En conséquence, l'algorithme de coloration de graphe est devenu une méthode courante pour résoudre les problèmes dans de nombreuses applications informatiques modernes.

4.2 L'algorithme de welsh-powel

L'algorithme de coloration de Welsh-Powell est largement utilisé car il est relativement simple et rapide tout en fournissant une solution efficace pour le problème de coloration de graphe. En trouvant une coloration avec le moins de couleurs possible, cet algorithme peut aider à optimiser l'utilisation des ressources dans de nombreuses applications, telles que la planification de cours, la conception de circuits, l'ordonnancement de production, l'attribution de fréquences dans les réseaux sans fil, et bien d'autres encore. De plus, l'algorithme de Welsh-Powell est facilement implémenté et s'adapte bien à une grande variété de situations, ce qui le rend particulièrement utile dans les applications pratiques où la rapidité et la simplicité de l'algorithme sont des priorités.
En résumé, l'algorithme de coloration de Welsh-Powell est utilisé car il offre une solution efficace et pratique à un problème important et courant dans de nombreux domaines de l'informatique et des sciences appliquées.

4.3 Les avantages de l'algorithme de welsh-powell

— Évite les retours en arrière qui se produisent en raison de conflits lors de la coloration ;
— Minimise le temps ;
— Utilise moin s de couleurs que les algorithmes de coloration classiques.

4.4 Description de notre système

On considère un système (ou serveurs) de cache distribué composé de plusieurs sites de stockage temporaires de sites web pour un accès ultérieur plus rapide. L'espace de stockage au niveau de chaque site est limité et ne peux contenir qu'un nombre limité de sites. On suppose qu'un utilisateur peut solliciter une page web à partir d'un serveur de cache directement accessible. Et si la page n'existe pas (défaut de page), il récupère la page à partir d'un autre serveur qui n'est pas directement accessible.
Notre système fonctionne comme suit : une copie de données fréquemment utilisées est stockée

dans le cache distribué. Cette copie est stockée dans différents nœuds du réseau pour améliorer la disponibilité et la résilience du système. Lorsqu'une demande est faite pour une donnée particulière, le système vérifie d'abord si cette donnée est déjà présente dans le serveur directement accessible. Si c'est le cas, le système la récupère directement à partir de là, ce qui réduit considérablement le temps de réponse. Si la donnée n'est pas présente dans le serveur local, le système récupère la donnée à partir du serveur le plus proche dans le système afin de minimiser le temps de réponse.
Pour symboliser le temps entre les serveurs dans un système de cache distribué, on peut utiliser la notation suivante :
$(t - server1 \rightarrow t - server2 \rightarrow t - server3 \rightarrow ... \rightarrow t - serverN.)$
Le temps entre serveur1 et serveur2 représenté par t.
Le temps entre serveur1 et serveur3 égale : 2t.
Cette notation signifie que le temps nécessaire pour récupérer des données à partire de deux serveurs directement connectés est égal à t. Alors que le temps nécessaire pour récupérer des données à partir d'un serveur à deux sauts est égal à 2t.
Bien que la redondance de données puisse offrir une disponibilité accrue et des temps de réponse plus rapides dans un système de cache distribué, elle peut également poser plusieurs problèmes. Tout d'abord, la duplication de données peut entraîner une utilisation inefficace de l'espace de stockage, ce qui peut devenir un problème lorsque le nombre de copies augmente.
De plus, la mise à jour de toutes les copies lorsqu'une modification est effectuée sur une copie peut entraîner une surcharge réseau importante, ce qui peut affecter les performances du système.

4.5 Modélisation du système

Nous modélisons le système de cache distribué par considère un graphe G=[X ,E,C], ou :
X correspond aux serveurs de cache et l'ensemble des arcs
E correspond au différentes liaisons (connexions réseaux) entre ces serveurs.
L'ensemble des couleurs C (rouge , vert , bleu, ...) correspond aux différentes pages web fréquemment sollicitées et qui doivent être stockée dans plusieurs serveurs.
La figure suivante présente un exemple illustratif de modélisation de notre système par un graphe où
Pc1 et Pc2 : sont deux utilisateurs qui demandent des services différents . Pour modéliser l'ensemble de sommets et ensemble des arcs dans un système informatique distribué, nous pouvons utiliser la théorie des graphes et créer un graphe ou on applique les règles suivantes :

- chaque sommet représente un serveur(Par exemple le serveur x1 est représenté par un sommet dans le graphe) ;
- chaque arête représente une connexion entre deux serveurs qui sont directement liés ;
- chaque couleur représente une copie d'un service .

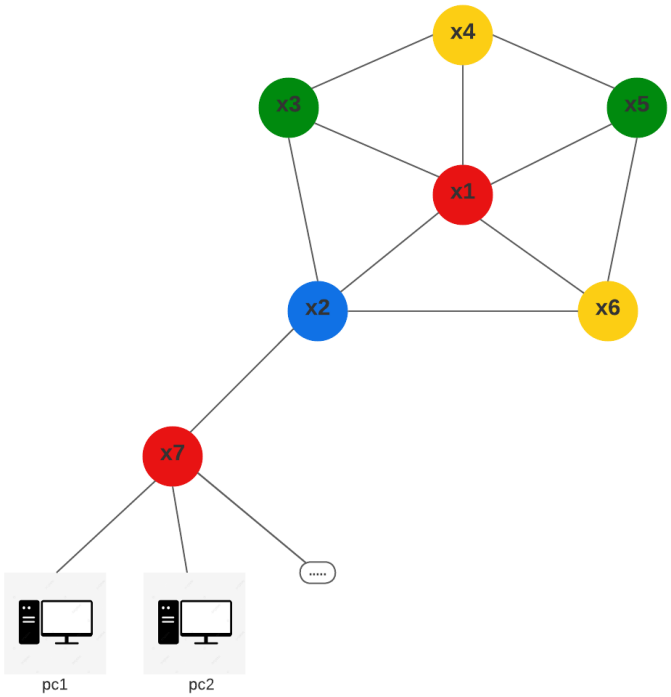


FIGURE 4.1 – Exemple de modélisation

4.6 La solution proposé

L'idée de base de la solution proposé repose sur le fait que si deux sommets sont adjacents, cela signifie qu'ils représentent deux serveurs qui sont directement connecté et qu'ils ne peuvent pas stocker la même copie d'une page web. Sur le graphe ces deux sommets ne peuvent pas être colorés parà la même couleur. En assignant des couleurs différentes aux sommets adjacents, on garantit que chaque serveur stocke une copie d'une page a une couleur unique parmi ses voisins, assurant ainsi une meilleure distribution de l'espace de stockage sur plusieurs serveurs tout en réduisant les délais d'accès.
Dans cette section nous proposons d'appliquer l'algorithme de coloration de graphe : « algorithme de Welsh-Powell » sur le graphe qui représente le système de cache distribué .
Cet algorithme permet d'assigner une couleur à chaque sommet d'un graphe de telle sorte que deux sommets adjacents n'aient pas la même couleur .
Les étapes de cet algorithme sont la suivante :

1. Trouvez le degré de chaque sommet ;
2. Répertoriez les sommets par ordre décroissant de degrés ;
3. Colorez le premier sommet avec la couleur 1 ;
4. Descendez dans la liste et colorez tous les sommets non connectés au sommet coloré avec la même couleur ;
5. Répétez l'étape 4 sur tous les sommets non colorés avec une nouvelle couleur, par ordre décroissant de degrés jusqu'a ce que tous les sommets soient colorés.

Voici le pseudo code de l'algorithme de Welsh-Powell :

```
Fonction WelshPowell(graph) :
Trier les sommets du graph par ordre décroissant de leur degré
Initialiser un tableau de couleurs avec la valeur -1 pour tous les sommets

Pour chaque sommet v dans le graph :
Si la couleur de v est -1 :
Attribuer une nouvelle couleur à v
Pour chaque sommet voisin u de v :
Si la couleur de u est -1 et n'est pas déjà utilisée par un voisin de v :
Attribuer la couleur à u

Retourner le tableau de couleurs
Fin de la fonction WelshPowell
```

Dans la figure 4.2, nous commençons par colorer le sommet x1 avec la couleur rouge. Le sommet x1 a une arête avec x2, x3, x4, x5 et x6, donc nous ne pouvons pas utiliser la couleur rouge. Nous colorons donc le sommet x3et x5 (il ya pas de liaison entre x3 et x5) avec la couleur vert. Et le sommet x4 avec la couleur jaune. Le sommet x6 on peut le coloré en jaune parce que il y'a pas une arête entre x6 et x4. le sommet x2 on le coloré par une nouvelle couleur : bleu parce qu'il a une arête avec x3(coloré par le vert) , x1(coloré par le rouge) et x6(coloré par le jaune) . Le sommet x7 a juste une arête avec x2 (coloré déjà en bleu) donc nous pouvons le coloré par le rouge. A la fin de l'exécution de l'algorithme de coloration, on obtient l'ensemble de sites sur lesquels un même service doit être stocké. Par exemple, le service représenté par la couleur jaune doit être stocké uniquement sur les deux sites x4 et x6 (figure 4.1).

Exemple d'exécution de l'algorithme de welsh-powell :

Soit le graphe ci-dessous qui représente un système de cache distribué. Ce qui suit montre les étapes d'application et on applique de l'algorithme de welsh-powell :

1. Énumérer les sommets en fonction de degré ;

2. x1 est en tête de la liste . la couleur rouge ;

3. On choisit nouvelle couleur jaune pour les sommets non coloré x4 et x6 ;

4. Choisissez une nouvelle couleur vert pour les sommets non coloré x3 et x5 ;

5. En descendant dans la liste , nous colorons x7 avec le rouge.

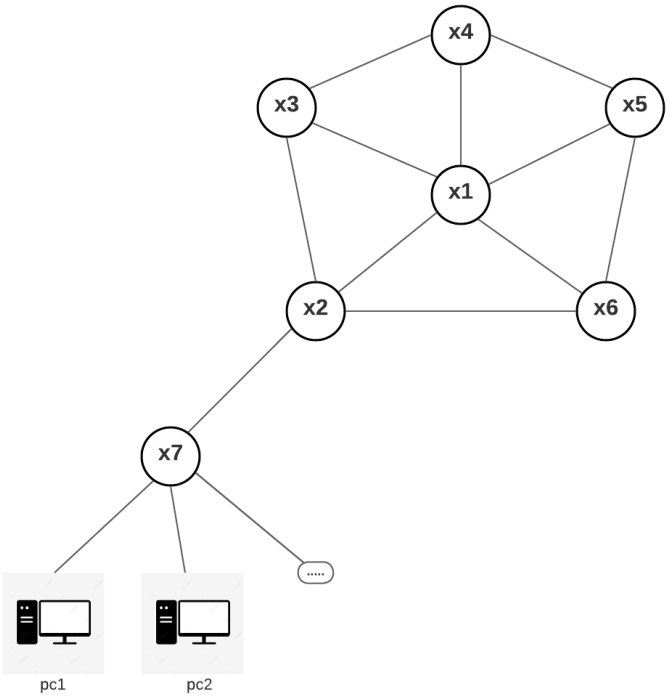


FIGURE 4.2 – Exemple de graphe qui trompe l'algorithme Welsh-Powell

Sommet	Degré	Successeurs
1	5	x2,x3,x4,x5,x6
2	4	x1,x3,x6,x7
3	3	x1,x2,x4
4	3	x1,x3,x5
5	3	x1,x4,x6
6	3	x1,x2,x5
7	1	x2

Sommet/couleur	x1	x2	x3	x4	x5	x6	x7
Bleu		oui					
Vert		non	oui		oui		
Rouge	oui	non	non		non		oui
Jaune	non	non	non	oui	non	oui	non

Les Couleurs :
jaune : x4 , x6 ;
vert : x3, x5 ;
bleu : x2 ;
rouge : x1 , x7 ;
Ordre λ : 1 , 2 , 3 , 5 , 4 , 6 , 7.
On obtient alors la coloration de graphe représentée dans la - figure 4.1- ci-dessus.

4.7 Diagramme d'états transition de système de cache distribué

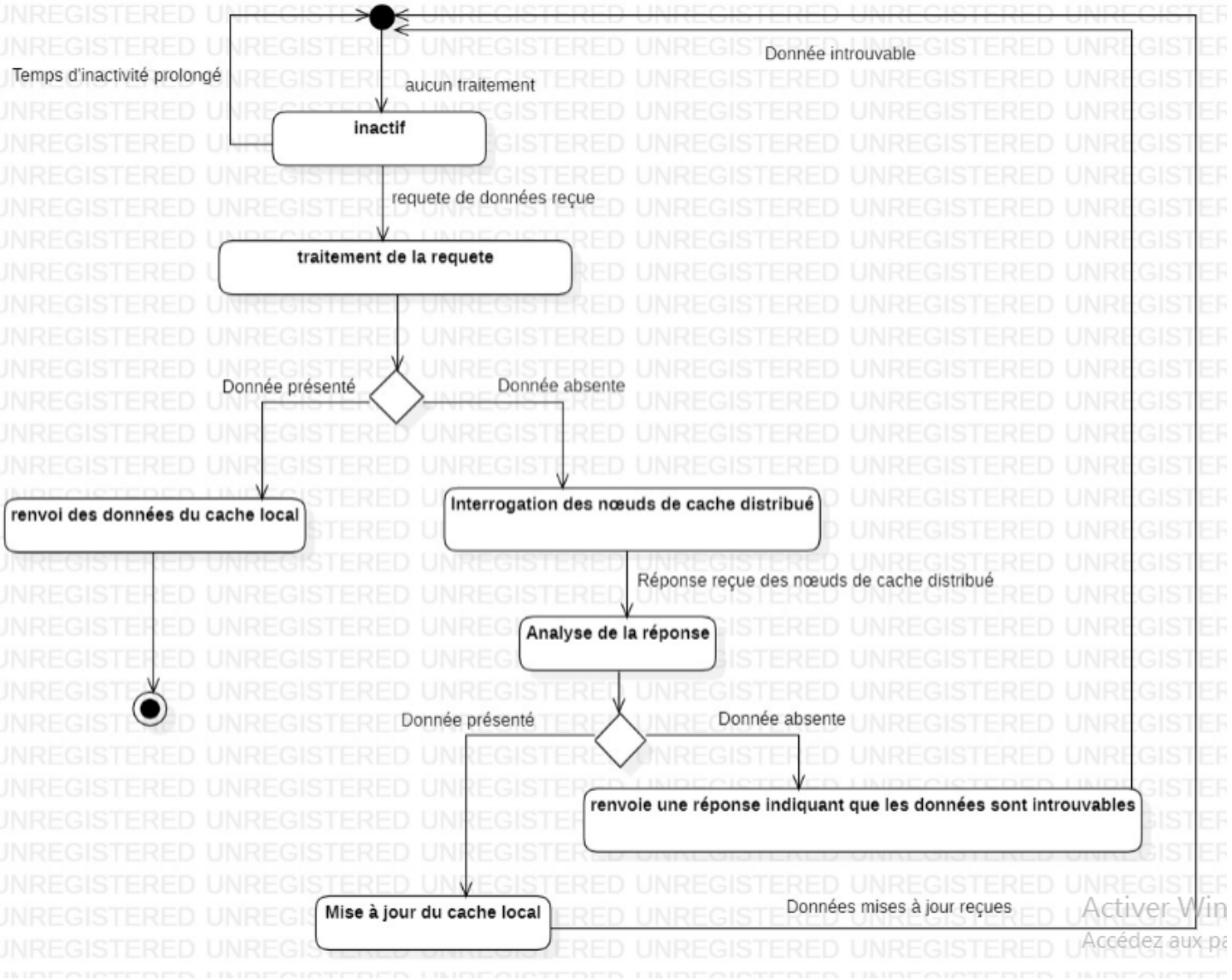


FIGURE 4.3 – Diagramme d'états transition de système de cache distribué

— État initial : Inactif
  — Lorsque le système est inactif, aucun traitement de requête n'est effectué.
— Événement : Requête de données reçue
  — Lorsqu'une requête de données est reçue par le système, il passe à l'état de traitement de la requête.
— État : Traitement de la requête
  — Le système vérifie si les données demandées sont présentes dans le cache local.
  — Condition : Données présentes.
    • Si les données sont présentes, le système renvoie les données du cache local avec

un temps de réponse rapide et revient à l'état initial.
  - Condition : Données absentes.
    • Si les données sont absentes, le système passe à l'état d'interrogation des nœuds de cache distribué.
- État : Interrogation des nœuds de cache distribué.
  - Le système envoie une requête aux nœuds de cache distribué pour vérifier si les données sont disponibles.
  - Pendant l'attente de réponse, le système reste dans cet état.
- État : Analyse de la réponse
  - Le système vérifie si les données demandées sont présentes dans la réponse reçue.
  - Condition : Données présentes
    • Si les données sont présentes, le système met à jour le cache local avec les nouvelles données et renvoie les données avec un temps de réponse rapide, puis revient à l'état initial.
  - Condition : Données absentes.
    • Si les données sont absentes, le système renvoie une réponse indiquant que les données sont introuvables, puis revient à l'état initial
- Événement : Données mises à jour reçues
  - Lorsque le système reçoit des données mises à jour, il passe à l'état de mise à jour du cache local.
- État : Mise à jour du cache local

  - Le système met à jour le cache local avec les nouvelles données reçues.
  - Une fois la mise à jour terminée, le système revient à l'état initial.
- Événement : Temps d'inactivité prolongé
  - Si le système reste inactif pendant une période prolongée, il revient à l'état initial.

4.8 Conclusion

En conclusion, l'application de la coloration de graphe dans la conception d'un système de cache distribué constitue une approche prometteuse pour optimiser la répartition des données entre les différents nœuds du réseau. Cette technique permet d'optimiser l'utilisation du cache tout en minimisant les délais d'accès aux données, et d'optimiser l'utilisation du cache, ce qui contribue à améliorer les performances globales. En tirant parti des avantages de la coloration de graphe, la conception d'un système de cache distribué peut être plus efficace et offrir une meilleure expérience utilisateur dans un environnement distribué.

Chapitre 5

Conception et implémentation

Sommaire

5.1 Introduction

Dans le chapitre précédent nous avons proposé l'algorithme de Welsh-Powell pour la coloration de graphes. Cet algorithme est de colorer un graphe de telle manière que deux sommets adjacents aient des couleurs différentes.
Dans ce chapitre, nous allons implémenter deux codes : le premier calcule le délai en fonction du nombre de nœuds et le 2 éme calcule l'espace occupé en fonction du nombre de nœuds .
Nous allons commencer par la préparation d'un environnement d'exécution équipé d'un système d'exploitation Windows 10, JDK, Eclipse , staruml , lucidchart ,canva nous allons réaliser expérimentations afin de optimiser l'espace de stockage et minimiser le temps de réponse.

5.2 Les outils Hardware et Software utilisés dans notre mémoire

5.2.1 Les outils Software

Langage JAVA

Java est un langage orienté objet, utilisé pour le développement de divers types d'application. Il est caractérisé par sa probabilité due à l'utilisation d'une machine virtuelle la JVM(Java Virtual Machine) qui fait interface entre le programme et le système d'exploitation.

FIGURE 5.1 – Logo JAVA

Overleaf

Dans notre mémoire, on a utilisé l'outil Overleaf pour la rédaction du manuscrit en format **.tex**. Overleaf est un éditeur LaTeX en ligne, collaboratif en temps réel.

FIGURE 5.2 – Logo Overleaf

Lucidchart

Lucidchart est une plateforme d'édition de diagrammes intelligents qui favorise la collaboration des équipes et les aide à prendre des décisions éclairées tout en construisant l'avenir.

FIGURE 5.3 – Logo lucidchart

Canva

Canva est une plateforme en ligne dédiée au design et à la communication visuelle. Son objectif principal est de fournir à chacun les moyens de créer et de publier du contenu selon ses propres aspirations et besoins.

FIGURE 5.4 – Logo Canva

StarUML

StarUML est un outil hautement spécialisé dans la modélisation UML, qui s'avère être extrêmement pratique pour le développement d'applications. C'est un logiciel complet conçu spécialement pour les utilisateurs expérimentés dans ce domaine.

FIGURE 5.5 – Logo StarUML

5.2.2 Les outils Hardware

Nous avons uilisé dans notre mémoire deux micro-portables HP ELITEBOOK et DELL :

HP ELITEBOOK

- Processeur Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz 2.40 GHz
- Mémoire installées(RAM) : 8 Go
- Système d'exploitation 64 bits

DELL

- Processeur : Intel(R) Celeron (R) N4000CPU @1,10GHz 1,10 GHz
- Mémoire installées(RAM) : 4 Go
- Système d'exploitation 64 bits

5.3 Exemple d'exécution de l'algorithme

Dans cette section, nous présentons un exemple d'exécution du l'algorithme de welsh-powell utilisant le graphe de la Figure 4.1 (a).

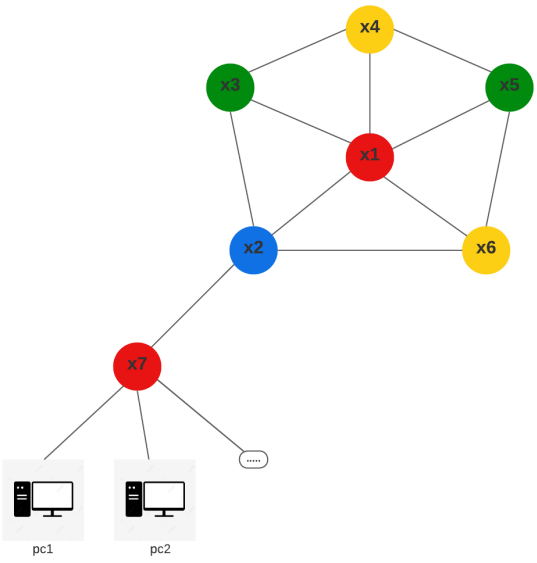


FIGURE 5.6 – (a) Exemple de graphe

```
4     Transparent,
5     Rouge,
6     jaune,
7     vert,
8     bleu }
```

FIGURE 5.7 – (b) Liste des couleurs

```
16          G.lier('1', '2');
17          G.lier('1', '3');
18          G.lier('1', '4');
19          G.lier('1', '5');
20          G.lier('1', '6');
21          G.lier('2', '3');
22          G.lier('2', '6');
23          G.lier('2', '7');
24          G.lier('3', '2');
25          G.lier('3', '4');
26          G.lier('3', '6');
27          G.lier('4', '1');
28          G.lier('4', '3');
29          G.lier('4', '5');
30          G.lier('5', '1');
31          G.lier('5', '4');
32          G.lier('5', '6');
33          G.lier('6', '1');
34          G.lier('6', '2');
```

FIGURE 5.8 – (c) Liste d'adjacence du graphe du Figure

```
Problems  Javadoc  Declaration  Console ×
<terminated> App [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe
Rouge ( 1 7 )
jaune ( 6 4 )
vert ( 3 5 )
bleu ( 2 )
```

FIGURE 5.9 – (d) Résultat de l'exécution de l'algorithme sur le graphe de le Figure

5.4 Expérimentation et analyse des résultats

Dans cette section nous avons calculé le délai et l'espace occupé en fonction du nombre de nœuds , nombre de couleurs et nombre de arcs. On trouve que à chaque fois on augmente le nombre des noeuds et le nombre des arcs puis on examinant le temps total et le nombre des couleurs.

Les résultats obtenus sont montrés par le tableau et illustres dans la figure :

Nœuds - couleur	4	6	8	10
7	1.75	2	2	2.2
9	2	2.16	2.12	2.4
11	2.25	2.33	2.25	2.5
13	2.5	2.5	2.37	2.6

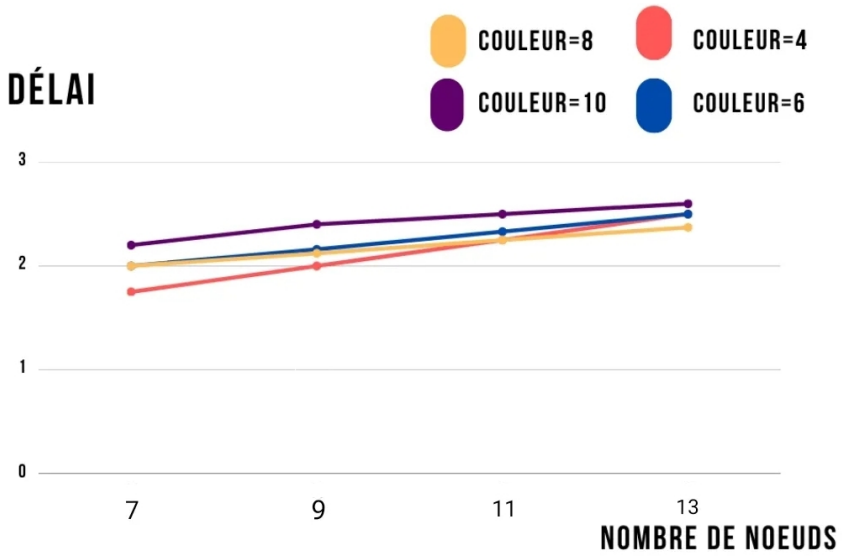


FIGURE 5.10 – (a) Le délai en fonction du nombre de nœuds

Nombre des sommets	Nombre des arcs	Nombre des couleurs
7	6	2
7	8 (x1 et x2-x1 et x3)	3
7	10(x1 et x5-x2 et x6)	4
7	12(x1 et x4-x5 et x7)	5
9	8	2
9	10(x1 et x5-x2 et x6)	3
9	12(x3 et x8-x4 et x2)	4
9	14(x1 et x2-x5 et x3)	5
11	10	2
11	12(x2 et x4-x8 et x9)	3
11	14(x3 et x1-x6 et x7)	4
11	16(x5 et x10-x4 et x6)	5
13	12	2
13	14(x1 et x5-x8 et x9)	3
13	16(x12 et x7-x5 et x3)	4
13	18(x4 et x2-x4 et x2)	5

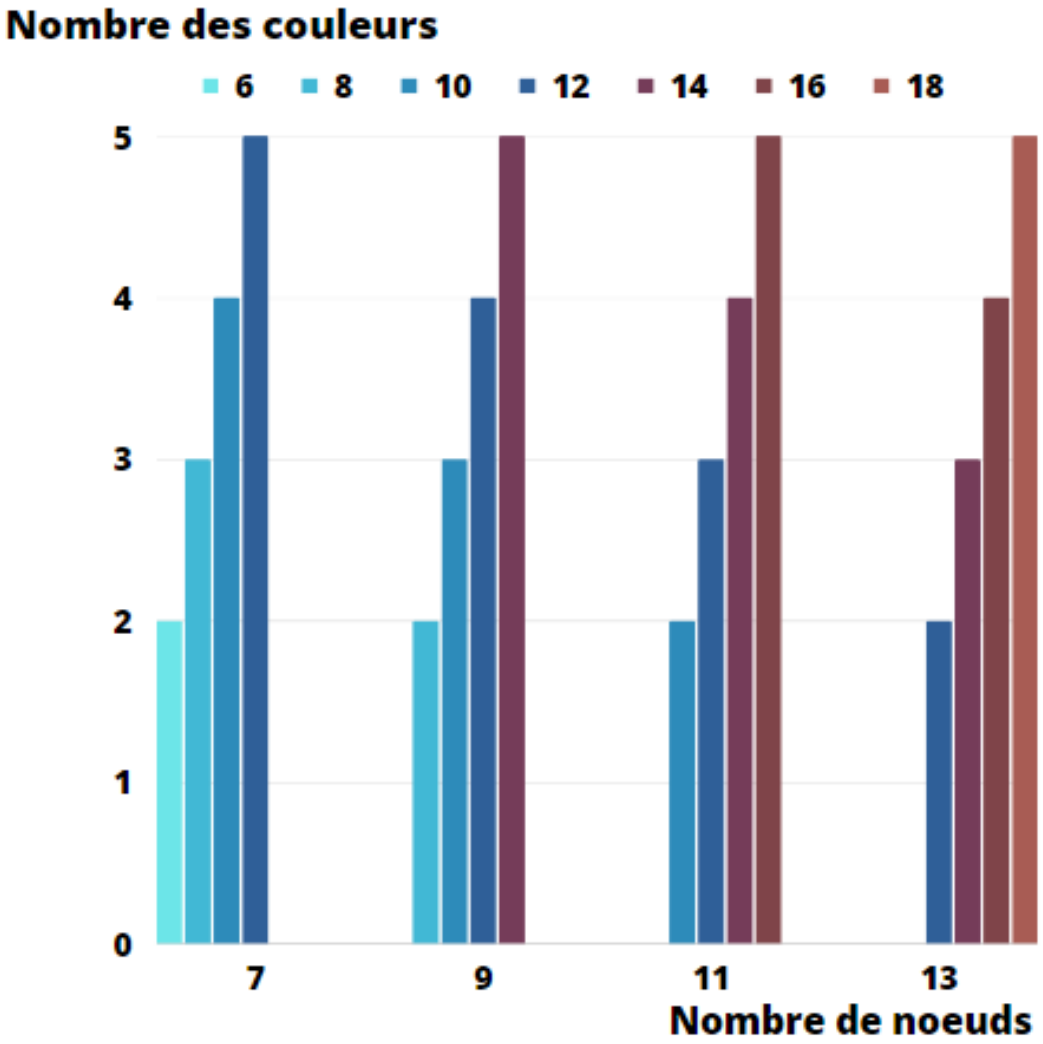


FIGURE 5.11 – (b) Nombre de couleur utilisé par chaque graphe

5.5 Description des schéma

5.5.1 Figure (a)

A travers la courbe graphique du figure ..qui représente le délai en fonction du nombre de nœuds , nous remarquons que :
A chaque fois on augmente le nombre de nœuds et le nombre de couleur , le délai augmente
Par exemple :

— Lorsque le nombre de couleur égale 4 et le nombre de nœuds égale 7 le délai égale 1.75.

On fixe le nombre de couleur a 4 a chaque fois on augmente le nombre de nœuds(9-11-13) le délai augmente (2 - 2.25 - 2.5) par ordre .

— Lorsque le nombre de couleur égale 6 et le nombre de nœuds égale 7 le délai égale 2.

On fixe le nombre de couleur a 6 a chaque fois on augmente le nombre de nœuds(9-11-13) , le délai augmente (2.16 - 2.33 - 2.5) par ordre .

— Lorsque le nombre de couleur égale 8 et le nombre de nœuds égale 7 le délai égale 2.
On fixe le nombre de couleur a 8 a chaque fois on augmente le nombre de nœuds(9-11-13) , le délai augmente (2.12 - 2.25 -2.37) par ordre .

— Lorsque le nombre de couleur égale 10 et le nombre de nœuds égale 7 le délai égale 2.2.
On fixe le nombre de couleur a 10 a chaque fois on augmente le nombre de nœuds(9-11-13) , le délai augmente (2.4 - 2.5 - 2.6) par ordre .

Justification

Dans le système cache distribué, l'augmentation du nombre de sommets et du nombre de couleurs peut entraîner une augmentation du délai pour plusieurs raisons :

1. **Communication inter-sommets :** Lorsque le nombre de sommets augmente dans un système de cache distribué, la communication entre les sommets devient plus complexe. Lorsqu'un sommet reçoit une demande de données qui ne sont pas présentes dans son cache, il doit contacter d'autres sommets pour obtenir ces données. Avec un nombre croissant de sommets, il y a plus de sommets potentiels à contacter, ce qui peut entraîner une augmentation de la latence de communication entre les sommets. Par conséquent, le délai pour récupérer les données demandées peut augmenter ;
2. **Coordonnée et synchronisation :**la coordination et la synchronisation entre ces nœuds deviennent plus complexes. Lorsqu'une donnée est mise à jour sur un nœud, il est nécessaire de propager cette mise à jour à tous les autres nœuds qui ont également une copie de cette donnée. Cependant, avec un grand nombre de nœuds, la gestion de cette coordination et de cette synchronisation devient plus difficile ;
3. **Charge et congestion du réseau :** Lorsque le nombre de nœuds augmente dans un système de cache distribué, la charge sur le réseau interne reliant ces nœuds peut également augmenter. Cette augmentation de la charge peut entraîner des problèmes de congestion du réseau, où la capacité du réseau est dépassée par le volume croissant de données échangées entre les nœuds.

5.5.2 Figure (b)

A travers le colonne graphique qui représente le nombre de couleurs utilisée par chaque graphe de nœuds différents on trouve que :
A chaque fois on augmente le nombre de nœuds et le nombre des arcs, le nombre de couleur augmente .
Par exemple :

— Lorsque le nombre de noeuds égale 7 et le nombre des arcs augmente (6,8,10,12), le nombre de couleurs augmente (2,3,4,5) par ordre.

— Lorsque le nombre de noeuds égale 9 et le nombre des arcs augmente (8,10,12,14), le nombre de couleurs augmente (2,3,4,5) par ordre.
— Lorsque le nombre de noeuds égale 11 et le nombre des arcs augmente (10,12,14,16), le nombre de couleurs augmente (2,3,4,5) par ordre.
— Lorsque le nombre de noeuds égale 13 et le nombre des arcs augmente (12,14,16,18), le nombre de couleurs augmente (2,3,4,5) par ordre.

Justification

Dans un système de cache distribué, lorsque le nombre de nœuds (ou sommets) et le nombre d'arcs augmentent, cela entraîne une complexité et une taille accrues du système. En conséquence, il peut être nécessaire de diviser les données en plusieurs catégories ou partitions, appelées "couleurs", pour une répartition équilibrée et une utilisation efficace des ressources.

— L'augmentation du nombre de nœuds implique souvent l'ajout de nouvelles entités au système, telles que des serveurs ou des dispositifs, qui participent à la gestion des données en cache. Chaque nœud est responsable du stockage et de la gestion d'une partie des données du cache ;
— De même, l'augmentation du nombre d'arcs indique une augmentation des relations et des interactions entre les nœuds. Les arcs représentent les canaux de communication ou de transfert de données entre les nœuds du système de cache ;
— Pour assurer une répartition équilibrée des données et éviter la surcharge de certains nœuds, il est nécessaire de répartir les données en cache de manière fine. Cela signifie diviser les données en différentes partitions ou couleurs. Chaque couleur représente une catégorie distincte de données qui sont réparties entre les nœuds ;
— L'augmentation du nombre de couleurs permet une meilleure répartition des données entre les nœuds, ce qui contribue à éviter la surcharge et à améliorer les performances globales du système de cache distribué.

Il est important de souligner que le choix du nombre de couleurs et la répartition des données en cache dépendent du contexte spécifique du système et des objectifs visés, tels que l'équilibrage de la charge, la réduction des collisions de données ou l'optimisation des performances globales. Les décisions de conception relatives à ces paramètres doivent être prises en tenant compte des exigences spécifiques du système et des contraintes associées.

5.6 Conclusion

Dans ce chapitre nous avons décrit l'environnement du travail et fait l'exécution de l'algorithme a notre système . Par la suite, nous avons testé la scalabilité du système en terme du délai et l'espace occupée (nombre de couleurs).Les résultats obtenus montrent que l'algorithme appliqué est scalable mais il ne donne pas toujours la solution optimale.

Chapitre 6

Vérification formelle d'un système de cache distribué

Sommaire

6.1 Introduction

Depuis la fin des années 1970, la vérification formelle des systèmes critiques a rencontré un succès indéniable. Au cours des quinze dernières années, ces techniques de vérification ont été étendues pour prendre en compte des aspects quantitatifs, ce qui permet notamment de spécifier les délais entre différentes actions du système (2).
Dans ce cadre,en 1994, Alur et Dill ont introduit le modèle des automates temporisés (3) pour représenter le comportement des systèmes qui intègrent des contraintes quantitatives liées au temps. Ces automates incluent des horloges qui progressent de manière continue avec le temps, mesurant ainsi les intervalles de temps entre différentes actions du système modélisé. Les algorithmes de vérification ont été étendus à ces modèles (61)(Laroussinie)(38), et des outils de vérification de modèles(75)(38)(model checking) ont été développés avec succès, appliqués à des exemples provenant du domaine industriel(13)(67).

6.2 le modèle des automates temporisés

Dans les années 1990, les automates temporisés ont été introduits par R. Alur et D. Dill (3). Ils ont étendu les automates classiques par des horloges qui s'incrémentent de façon continue et synchrone avec le temps. Deux nouvelles notions ont été ajoutées aux transitions :

— Une garde définie sur la valeur des horloges décrivant l'instant où la transition peut être tirée et une réinitialisation à zéro pour un ensemble d'horloges lors du tirage de la transition.
— Une autre nouvelle notion appelée invariant a été ajoutée aux localités et exprimée sous la forme d'une contrainte sur les horloges, cet invariant peut forcer le système à quitter une localité pour lancer l'exécution d'une action en limitant le temps d'attente dans cette localité.

Le domaine de temps peut être soit l'ensemble des entiers naturels $\mathbb{N}$, soit l'ensemble des rationnels positifs $\mathbb{Q}_{\geq 0}$ ou bien l'ensemble des réels positifs $\mathbb{R}_{\geq 0}$. Dans cette section, nous considérons les réels positifs, cependant, il faut noter qu'il n'y a pas une grande différence dans les résultats si on considère $\mathbb{Q}_{\geq 0}$ ou $\mathbb{N}$. En effet, par la considération de $\mathbb{N}$, la granularité des périodes de progression du temps peut être adoptée pour l'analyse des propriétés du système.

6.2.1 Préliminaires

Dans ce qui suit, nous considérons $\mathbb{R}_{\geq 0}$ l'ensemble des nombres réels non négatifs. Pour $t \in \mathbb{R}_{\geq 0}$, $\lfloor t \rfloor$ et $fract(t)$ se réfèrent respectivement aux parties intégrale et fractionnaire de t, c'est-à-dire $t = \lfloor t \rfloor + fract(t)$.
Soit H un ensemble d'horloges à valeur dans $\mathbb{R}_{\geq 0}$. Une valuation v pour H est une fonction $(v : H \to \mathbb{R}_{\geq 0})$ qui associe à chaque horloge x sa valeur $v(x)$. On note $\mathbb{R}^H_{\geq 0}$ l'ensemble des valuations pour H. Étant donné un réel $d \in \mathbb{R}_{\geq 0}$, on note $v + d$ la valuation qui associe à l'horloge x la valeur $v(x) + d$. Si R est un sous-ensemble de H, $v[R]$ représente la valuation v' définie par : $v'(x) = 0$ pour tout $x \in R$ et $v'(x) = v(x)$ pour $x \in H \backslash R$.
On note $C(X)$ l'ensemble des contraintes d'horloges sur X, c'est-à-dire l'ensemble des combinaisons booléennes de contraintes atomiques de la forme $x \bowtie c$ avec $x \in X$, $\bowtie \in \{=, <, \leq, >, \geq\}$ et $c \in \mathbb{R}_{\geq 0}$. On désigne par $C \lessdot (X)$ la restriction de $C(X)$ aux combinaisons positives ne contenant que des contraintes de la forme $x \leq c$ ou $x < c$. Les contraintes d'horloges s'interprètent de manière naturelle sur les valuations d'horloges : une valuation v satisfait une contrainte atomique $x \bowtie c$ lorsque $v(x) \bowtie c$, l'extension aux contraintes quelconques est alors immédiate. Lorsqu'une valuation v satisfait une contrainte C, on écrit $v \models C$.

6.2.2 Modèle des automates temporisés et sa sémentique

On peut définir un automate temporisé d'une manière formelle comme suit :

Définition 1

Un automate temporisé A est un 6-uplet (S, s_0, H, I, T, Act) où :
— S est un ensemble fini de localités,
— $s_0 \in S$ est la localité initiale,
— H est un ensemble fini d'horloges,

— $T \subseteq S \times C(H) \times Act \times 2^H \times S$ est un ensemble fini de transitions, $e = (s, G, a, R, s') \in T$ (notée par $s \overset{G,a,R}{\rightarrow} s'$) représente une transition de s vers s' avec une garde G, un ensemble d'horloges R à réinitialiser et une action a qui représente l'étiquette de la transition,
— $I : S \rightarrow C_{\leqslant}(H)$ associe un invariant à chaque localité,

— Act est un alphabet d'actions.
Figure ci dessus représente un exemple d'automate temporisé (61).

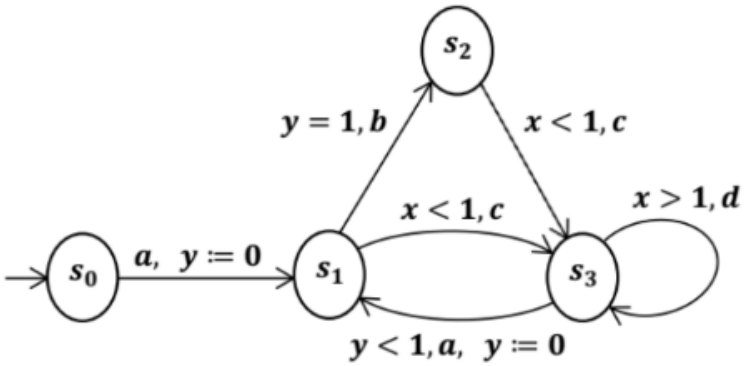


FIGURE 6.1 – Automate temporisé.

Une exécution dans un automate temporisé peut aussi être représentée par un mot temporisé, c'est-à-dire une séquence de paires (action,date). Donc, une exécution peut être vue comme suit : $(s_0, v_0, t_0) \overset{a_1}{\rightarrow} (s_1, v_1, t_1) \overset{a_2}{\rightarrow} \cdots \overset{a_n}{\rightarrow} (s_n, v_n, t_n)$ avec $t_i \in \mathbb{R}_{\geq 0}$, $t_0 = 0$ et $t_{i+1} \geq t_i$ pour tout i. La date t_i coïncide avec la date de l'exécution de l'action a_i.
L'étape $(s_i, v_i, t_i) \overset{a_{i+1}}{\rightarrow} (s_{i+1}, v_{i+1}, t_{i+1})$ s'exprime avec une attente de durée égale à $t_{i+1} - t_i$ puis le lancement de l'action a_{i+1}, la valuation v_{i+1} est donc calculée à partir de $v_i + (t_{i+1} - t_i)$ sans oublier de réinitialiser certaines horloges (selon la transition choisie). Alors, le mot temporisé associé est $(a_1, t_1)(a_2, t_2) \cdots$ Par exemple, le mot temporisé de l'exécution présentée ci-dessus est $(a, 7.35)(b, 8.35) \cdots$

Définition 2

Un système de transitions temporisé (STT) est un quadruplet $\mathcal{S} = (Q, q_0, \rightarrow, Act)$ où Q est un ensemble d'états (éventuellement infini), $q_0 \in Q$ est l'état initial et $\rightarrow \subseteq Q \times (Act \cup \mathbb{R}_{\geq 0}) \times Q$ est la relation de transition. La relation $\rightarrow$ doit satisfaire les trois propriétés suivantes :
— si $q0q'$, alors $q = q'$,
— si qdq' et $q'd'q''$ avec $d, d' \in \mathbb{R}_{\geq 0}$, alors $qd''q''$,
— si qdq' avec $d \in \mathbb{R}_{\geq 0}$, alors pour tout $0 \leq d' \leq d$, il existe $q'' \in Q$ tel que $qd'q''$.
Les trois conditions citées plus haut expriment que le temps est continu et déterministe. Elles sont traditionnelles dans les systèmes temporisés, voir par exemple.

De manière standard, une exécution dans un STT est représentée par une suite de transitions successives. Dans $\mathcal{S}$, un état $q \in Q$ est dit atteignable s'il existe une exécution menant de l'état initial q_0 jusqu'à l'état q.

6.3 Les propriétés des automates temporisés

- **Le problème du vide** : dans le domaine des automates temporisés, consiste à résoudre l'interrogation de la véracité de l'affirmation selon laquelle l'ensemble des états accessibles de l'automate, L(A), est vide (∅). En d'autres termes, il s'agit de déterminer si l'automate peut atteindre un état final à partir de son état initial. Selon la référence (alu), ce problème a été démontré comme étant Pspace-complet pour la classe des automates temporisés classiques :
- **Le problème de l'universalité** : dans le domaine des automates temporisés, est le dual du problème du vide. Pour tout automate temporisé A, le problème de l'universalité consiste à résoudre l'interrogation de la véracité de l'affirmation selon laquelle l'ensemble des états accessibles de l'automate, L(A), est égal à l'ensemble de toutes les séquences infinies d'actions possibles $(\sum \times \mathrm{T})\infty$
  Selon la référence (alu), ce problème a été démontré comme étant indécidable pour la classe des automates temporisés classiques ;
- **Le problème de l'inclusion de langage** :dans le domaine des automates temporisés, vise à déterminer la véracité de l'affirmation selon laquelle, pour tout couple d'automates temporisés A et B, l'ensemble des mots acceptés par A, L(A), est inclus dans l'ensemble des mots acceptés par B, L(B). Selon la référence (alu), ce problème a été démontré comme étant indécidable pour la classe des automates temporisés classiques. Cette indécidabilité est établie en réduisant le problème de l'inclusion de langage au problème de l'universalité ;
- **Le problème de l'équivalence de langage** : dans le domaine des automates temporisés, vise à déterminer la véracité de l'affirmation selon laquelle, pour tout couple d'automates temporisés A et B, les ensembles de mots acceptés par A et B, respectivement L(A) et L(B), sont équivalents, c'est-à-dire si L(A) = L(B). Selon la référence (alu), ce problème a été démontré comme étant indécidable pour la classe des automates temporisés classiques. Cette indécidabilité est établie en réduisant le problème de l'équivalence de langage à un problème de double inclusion des langages.

6.4 les règles de passage d'un diagramme états transition vers un automate temporisé

Lors de la conversion d'un diagramme d'états transition vers un automate, il existe certaines règles à suivre. Voici les étapes générales à suivre :

- **Identifier les états :** Examinez le diagramme d'états transition et identifiez tous les états présents. Chaque état sera représenté par un état dans l'automate.
- **Déterminer l'état initial :** Identifiez l'état initial dans le diagramme d'états transition. Cet état deviendra l'état initial de l'automate.

— **Déterminer les états finaux :** Identifiez les états finaux dans le diagramme d'états transition. Ces états deviendront les états finaux de l'automate.
— **Traduire les transitions :** Pour chaque transition dans le diagramme d'états transition, identifiez l'événement déclencheur et les actions associées à la transition. Chaque transition deviendra une transition dans l'automate.
— **Événements déclencheurs :** Les événements déclencheurs sont les conditions qui provoquent une transition d'un état à un autre. Ils peuvent être représentés par des symboles, des conditions ou des actions spécifiques.
— **Actions associées :** Les actions associées à une transition sont les actions ou les effets qui se produisent lorsque la transition est effectuée. Ils peuvent être représentés par des actions spécifiques à effectuer ou des modifications d'état.
— **Gérer les transitions sans événements déclencheurs explicites :** Dans certains cas, il peut y avoir des transitions sans événements déclencheurs explicites dans le diagramme d'états transition. Il est important de les prendre en compte et de les gérer correctement lors de la conversion. Par exemple, une transition qui se produit automatiquement après un certain laps de temps peut être représentée par une transition avec un événement déclencheur spécial tel que "timeout".
— **Ajouter les transitions manquantes :** Si le diagramme d'états transition ne contient pas toutes les transitions nécessaires pour spécifier le comportement complet de l'automate, vous devrez ajouter les transitions manquantes. Assurez-vous que toutes les conditions et actions requises sont correctement prises en compte.

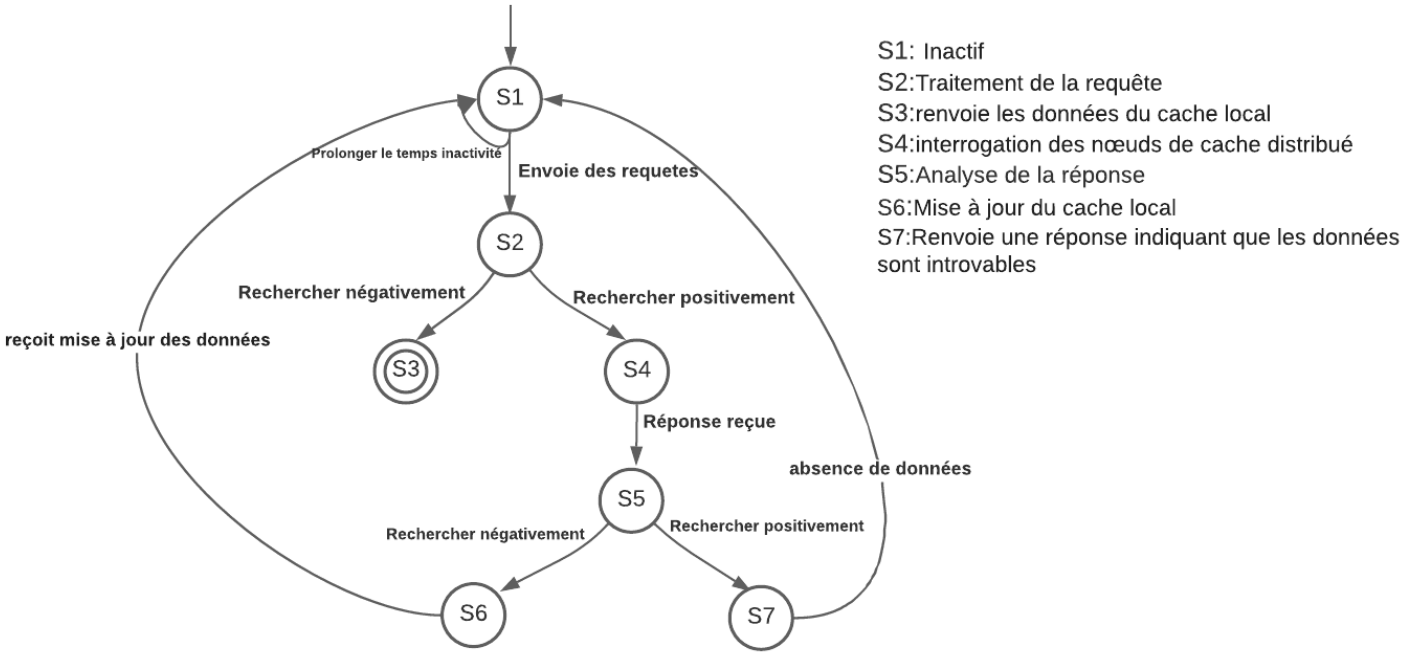


FIGURE 6.2 – Automate temporisé du notre exemple illustratif

6.5 La vérification en pratique : l'outil UPPAAL

UPPAAL est un outil puissant utilisé pour modéliser, valider et vérifier des systèmes temps réel. Il convient particulièrement aux systèmes qui peuvent être représentés sous forme d'automates temporisés ou d'automates hybrides linéaires de classe LHS. De plus, UPPAAL est approprié pour les systèmes qui peuvent être modélisés comme des processus non déterministes avec une structure de contrôle finie, des horloges à valeurs réelles, et qui communiquent via des canaux ou des structures de données partagées.

UPPAAL se compose d'un ensemble d'outils pour la vérification automatique de propriétés de sûreté et de vivacité bornée des systèmes temps réel. La machine UPPAAL, développée en C++, agit comme le serveur, tandis que l'interface graphique utilisateur (GUI), développée en Java, est le client. Ces deux entités communiquent via des protocoles internes, offrant ainsi la flexibilité d'exécuter le serveur et l'interface GUI sur des machines différentes.

La machine UPPAAL comprend plusieurs outils, tels que checkta (vérification de syntaxe) et verifyta (vérification de modèle), qui permettent d'effectuer différentes analyses sur le modèle spécifié. De son côté, l'interface graphique utilise des outils tels que atg2ta et hs2ta pour faciliter la modélisation et la transformation des spécifications.

La collaboration entre ces outils est illustrée dans la figure 6.3 (Vu d'ensemble d'UPPAAL). Cette architecture globale d'UPPAAL permet une approche modulaire et facilite l'interaction entre les différentes fonctionnalités offertes par l'outil.

En résumé, UPPAAL est un outil complet et flexible pour la modélisation, la validation et la vérification des systèmes temps réel. Il offre une gamme d'outils et de fonctionnalités pour faciliter l'analyse des propriétés des systèmes et la détection d'erreurs potentielles .

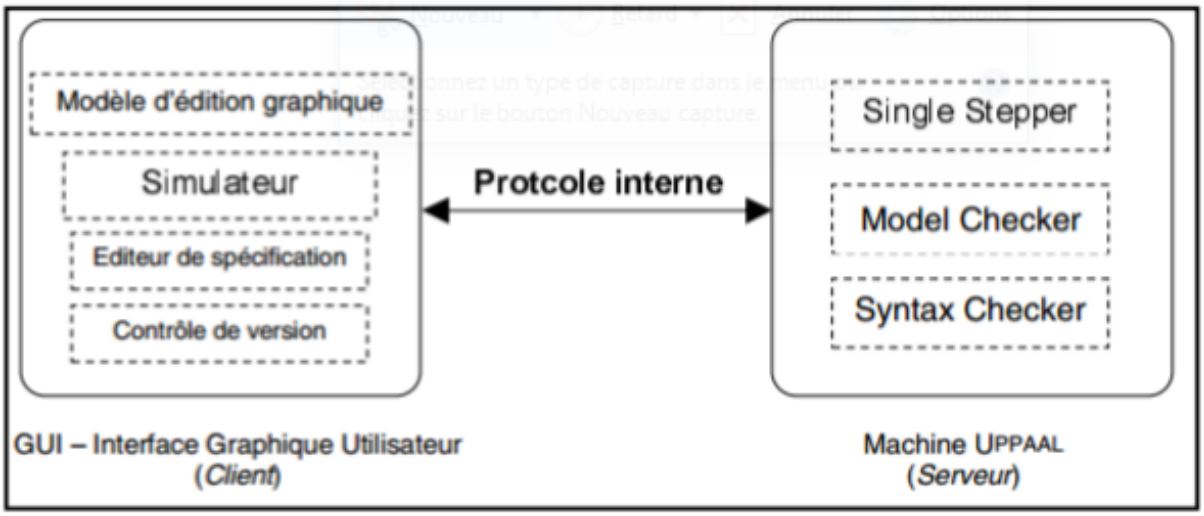


FIGURE 6.3 – Vu d'ensemble d'UPPAAL

Depuis son développement UPPAAL a connu des améliorations en vue de supporter des fonctionnalités désirées qui sont de plus en plus complexes, et d'offrir la rapidité et la flexibilité. La nouvelle architecture est constituée par des couches qui peuvent se communiquer.

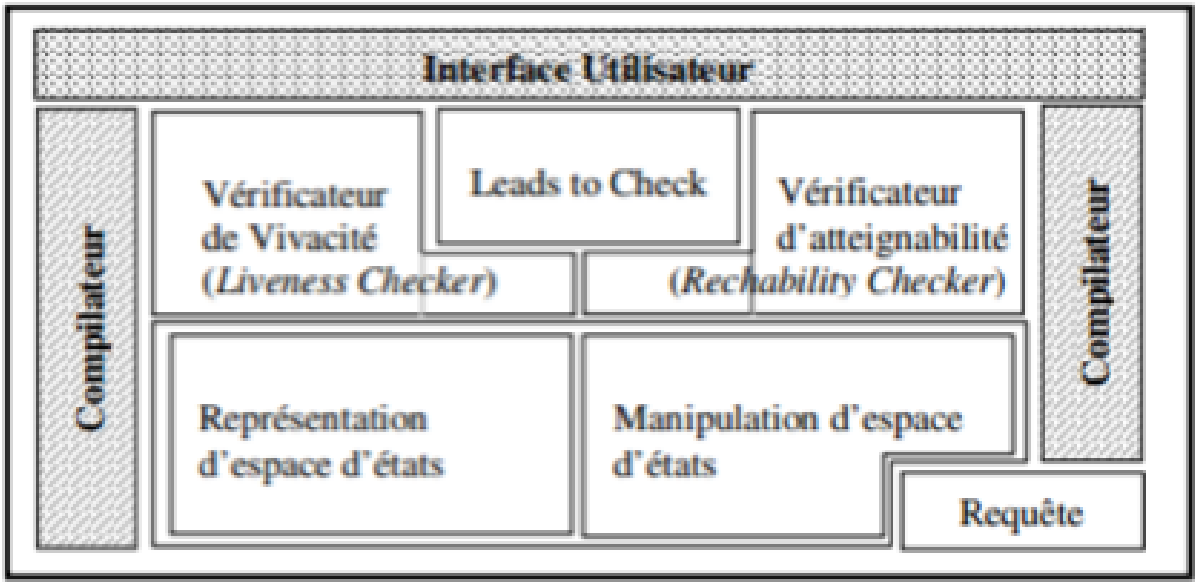


FIGURE 6.4 – Architecture d'UPPAAL

6.6 Vérification du notre exemple illustratif

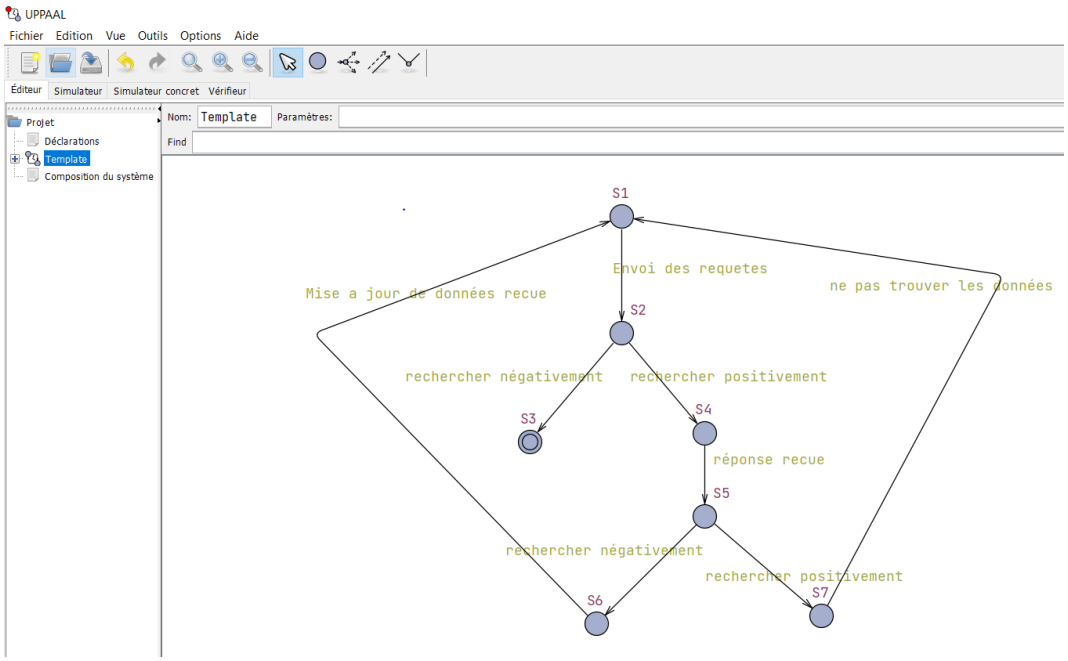


FIGURE 6.5 – automate de notre système

Chapitre 7

Conclusion générale et perspectives

Sommaire

7.1 Bilan

Il a été reconnu que les graphes gagnent en popularité constituant une partie intégrante de nombreuses applications du monde réel. En raison de l'importance des graphes, de nombreuses études ont beaucoup investi dans les technologies de l'exploitation des graphes de données. Ces efforts ont abouti à un nombre croissant d'approches d'interrogation de graphes. Notre projet visait à optimiser la redondance des données dans les systèmes de cache distribué tout en réduisant le temps de réponse et en assurant une cohérence optimale des données.
Les résultats obtenus démontrent une tendance claire : à mesure que le nombre de nœuds augmente tout en maintenant le nombre de couleurs fixe, le délai également augmente de manière progressive. Cette observation est cohérente avec les principes de la coloration de graphe appliquée aux systèmes de cache distribué.
Les résultats obtenus révèlent que l'augmentation du nombre de nœuds et d'arcs dans un système de cache distribué nécessite une augmentation correspondante du nombre de couleurs pour assurer une répartition équilibrée et optimale des données. Cela permet de gérer efficacement la complexité et la croissance du système, en garantissant des performances optimales.
En fin la vérification formelle dans un système de cache distribué assure la cohérence des données,et la validité des opérations de mise en cache et d'éviction, grâce à l'utilisation de méthodes et d'outils formels.

7.2 Perspectives

Comme perspectives on peut proposer :

— Réaliser une étude expérimentale de l'algorithme proposé sur de vastes clusters en utilisant des graphes de grande taille ;

— Valider l'implémentation de l'algorithme sur des plates-formes de Cloud public telles qu'Amazon ;
— Adapter d'autres algorithmes de coloration des sommets d'un graphe pour les intégrer dans notre approche ;
— Effectuer des expérimentations afin de comparer l'algorithme proposé avec d'autres algorithmes de graphes parallèles ou séquentiels existants dans la littérature, en utilisant des graphes de données réels ;
— Réaliser une étude pour évaluer la scalabilité des améliorations mises en œuvre, en déterminant comment elles fonctionnent sur des jeux de données plus volumineux ou sur différents types de graphes.

Webographie

[A] : https ://www.cloudflare.com/fr-fr/learning/cdn/what-is-caching/
[B] : https ://aws.amazon.com/fr/caching/
[C] : https ://www.lemagit.fr/definition/serveur-cache
[D] : https ://learn.microsoft.com/fr fr/aspnet/core/performance/caching/distributed ?view=aspnetcore-7.0
[E] : https ://fr.theastrologypage.com/distributed-cache
[F] : https ://hazelcast.com/glossary/distributed-cache/
[G] : https ://aws.amazon.com/fr/caching/
[H] : https ://learn.microsoft.com/fr-fr/sharepoint/administration/plan-for-feeds-and-the-distributed-cache-service
[I] : https ://aws.amazon.com/fr/caching/
[J] : https ://www.atlassian.com/fr/microservices/microservices-architecture/distributed-architecture
[K] : https ://www.ionos.fr/digitalguide/serveur/know-how/quest-ce-que-le-distributed-computing/
[L] : https ://www.splunk.com/fr$_fr/data-insider/what-are-distributed-systems.html$
[M] : $https : //www.organisation - performante.com/systemes - distribues - quels - avantages - pour - votre - si/$
[N] : $https : //www.organisation - performante.com/systemes - distribues - quels - avantages - pour - votre - si/$
[O] : $https : //www.splunk.com/fr_fr/data-insider/what-are-distributed-systems.html$
[P] : $https : //www.ionos.fr/digitalguide/serveur/know - how/quest - ce - que - le - distributed - computing/$
[Q] : $https : //www.ionos.fr/digitalguide/hebergement/aspects - techniques/quest - ce - quun - cache/$
[R] : $https : //aws.amazon.com/fr/caching/$
[S] : $https : //support.apple.com/fr - fr/guide/mac - help/mchl3b6c3720/mac$
[T] : $https : //www.fortinet.com/fr/resources/cyberglossary/what - is - caching$
[U] : $https : //learn.microsoft.com/fr - fr/azure/architecture/best - practices/caching$
[V] : $https : //www.lemagit.fr/definition/Memoire - cache$
[W] : $https : //context.reverso.net/traduction/francais - anglais/cache + de + r$

Bibliographie

[alu]

[2] Altisen, K., Markey, N., Reynier, P.-A., and Tripakis, S. (2005). Implémentabilité des automates temporisés. In *Actes du 5ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'05)*, pages 395–406. Hermès.

[3] Alur, R., Courcoubetis, C., and Dill, D. (1993). Model-checking in dense real-time. *Information and computation*, 104(1) :2–34.

[4] Anderson, T. E., Dahlin, M. D., Neefe, J. M., Patterson, D. A., Roselli, D. S., and Wang, R. Y. (1995). Serverless network file systems. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 109–126.

[5] Arteaga, D., Otstott, D., and Zhao, M. (2012). Dynamic block-level cache management for cloud computing systems. In *Conference on File and Storage Technologies*.

[6] Badenhop, C. W., Graham, S. R., Ramsey, B. W., Mullins, B. E., and Mailloux, L. O. (2017). The z-wave routing protocol and its security implications. *Computers & Security*, 68 :112–129.

[7] Baker, M. G., Hartman, J. H., Kupfer, M. D., Shirriff, K. W., and Ousterhout, J. K. (1991). Measurements of a distributed file system. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, pages 198–212.

[8] Barish, G. and Obraczka, K. (2000). World wide web caching : Trends and techniques. *IEEE Communications magazine*, 38(5) :178–184.

[9] Batsakis, A. and Burns, R. (2005). Cluster delegation : High-performance, fault-tolerant data sharing in nfs. In *HPDC-14. Proceedings. 14th IEEE International Symposium on High Performance Distributed Computing, 2005.*, pages 100–109. IEEE.

[10] Batsakis, A. and Burns, R. (2008). Nfs-cd : write-enabled cooperative caching in nfs. *IEEE Transactions on Parallel and Distributed Systems*, 19(3) :323–333.

[11] Belady, L. A. (1966). A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal*, 5(2) :78–101.

[12] Bellare, M. and Micciancio, D. (1997). A new paradigm for collision-free hashing : Incrementality at reduced cost. In *Advances in Cryptology—EUROCRYPT'97 : International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings 16*, pages 163–192. Springer.

[13] Bengtsson, J., Griffioen, W. D., Kristoffersen, K. J., Larsen, K. G., Larsson, F., Pettersson, P., and Yi, W. (1996). Verification of an audio protocol with bus collision using uppaal. In *Computer Aided Verification : 8th International Conference, CAV'96 New Brunswick, NJ, USA, July 31–August 3, 1996 Proceedings 8*, pages 244–256. Springer Berlin Heidelberg.

[14] Bernardini, C. (2015). *Popularity-Based Caching Strategies for Content Centric Networking*. PhD thesis, Université de Lorraine.

[15] Birke, R., Bjoerkqvist, M., Chen, L. Y., Smirni, E., and Engbersen, T. (2014). (big) data in a virtualized world : volume, velocity, and variety in cloud datacenters. In *12th {USENIX} Conference on File and Storage Technologies ({FAST} 14)*, pages 177–189.

[16] Burks, A. W., Goldstine, H. H., and Von Neumann, J. (1946). Preliminary discussion of the logical design of an electronic computer instrument. Technical report.

[17] Chai, W. K., He, D., Psaras, I., and Pavlou, G. (2012). Cache "less for more" in information-centric networks. In *11th International Networking Conference (NETWORKING)*, number Part I, pages 27–40. Springer.

[18] Comer, D. E. and Griffioen, J. (1990). A new design for distributed systems : The remote memory model.

[19] Cortes, T., Girona, S., and Labarta, J. (1997a). Avoiding the cache-coherence problem in a parallel/distributed file system. In *High-Performance Computing and Networking : International Conference and Exhibition Vienna, Austria, April 28–30, 1997 Proceedings 5*, pages 860–869. Springer.

[20] Cortes, T., Girona, S., and Labarta, J. (1997b). Design issues of a cooperative cache with no coherence problems. In *Proceedings of the fifth workshop on I/O in parallel and distributed systems*, pages 37–46.

[21] Dahlin, M. D., Wang, R. Y., Anderson, T. E., and Patterson, D. (1994). Cooperative caching : Using remote client memory.

[22] Das, V. (2015). *Learning Redis*. Packt Publishing Ltd.

[23] Din, I. U. (2016). Flexpop : A popularity-based caching strategy for multimedia applications in information-centric networking. *Universiti Utara Malaysia*.

[24] Feeley, M. J., Morgan, W. E., Pighin, E., Karlin, A. R., Levy, H. M., and Thekkath, C. A. (1995). Implementing global memory management in a workstation cluster. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 201–212.

[25] Fitzpatrick, B. (2004). Distributed caching with memcached. *Linux journal*, 2004(124) :5.

[26] Freedman, M. J., Annapureddy, S., and Mazieres, D. (2005). Shark : Scaling file servers via cooperative caching. In *Proc. Second Usenix/ACM Symp. Networked Systems Design and Implementation (NSDI)*.

[27] Gray, C. and Cheriton, D. (1989). Leases : An efficient fault-tolerant mechanism for distributed file cache consistency. *ACM SIGOPS Operating Systems Review*, 23(5) :202–210.

[28] Han, H., Lee, Y. C., Shin, W., Jung, H., Yeom, H. Y., and Zomaya, A. Y. (2011). Cashing in on the cache in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 23(8) :1387–1399.

[29] Hennessy, J. L. and Patterson, D. A. (2011). *Computer architecture : a quantitative approach*. Elsevier.

[30] Hoque, A. M., Amin, S. O., Alyyan, A., Zhang, B., Zhang, L., and Wang, L. (2013). Nlsr : Named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 15–20.

[31] IEEE, I. o. E. and Electronics Engineers, Inc. Staff, C. (1994). Ieee standard for information technology-portable operating system interface (posix) system application program interface (api), amendment 1 : Realtime extension (c language), ieee std 1003.1 b-1993.

[32] Isaila, F., Malpohl, G., Olaru, V., Szeder, G., and Tichy, W. (2004). Integrating collective i/o and cooperative caching into the" clusterfile" parallel file system. In *Proceedings of the 18th annual international conference on Supercomputing*, pages 58–67.

[33] Johns, M. (2013). *Getting Started with Hazelcast*. Packt Publishing Ltd.

[34] Johnson, T., Shasha, D., et al. (1994). 2q : a low overhead high performance bu er management replacement algorithm. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 439–450. Citeseer.

[35] Kgil, T. and Mudge, T. (2006). Flashcache : a nand flash memory file cache for low power web servers. In *Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems*, pages 103–112.

[36] Lal, K. N. and Kumar, A. (2018). A centrality-measures based caching scheme for content-centric networking (ccn). *Multimedia Tools and Applications*, 77 :17625–17642.

[Laroussinie] Laroussinie, F. Timed modal logics for specifying and verifying real-time systems.

[38] Larsen, K. G., Weise, C., Yi, W., and Pearson, J. (1999). Clock difference diagrams (extended version). *Nordic Journal of Computing*, 6 :271–298.

[39] Le Moal, D., Bandic, Z., and Guyot, C. (2012). Shingled file system host-side management of shingled magnetic recording disks. In *2012 IEEE International Conference on Consumer Electronics (ICCE)*, pages 425–426. IEEE.

[40] Lee, C., Sim, D., Hwang, J. Y., and Cho, S. (2015). F2fs : A new file system for flash storage. In *FAST*, volume 15, pages 273–286.

[41] Leung, A. W., Pasupathy, S., Goodson, G. R., and Miller, E. L. (2008). Measurement and analysis of large-scale network file system workloads. In *USENIX annual technical conference*, volume 1, pages 5–2.

[42] Liptay, J. S. (1968). Structural aspects of the system/360 model 85, ii : The cache. *IBM Systems Journal*, 7(1) :15–21.

[43] Lorrillere, M., Sopena, J., Monnet, S., and Sens, P. (2013). Vers un cache réparti adapté au cloud computing. In *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS'2013)-9ème Conférence Française sur les Systèmes d'Exploitation (CFSE'13)*, pages 1–12.

[44] Meddeb, M., Dhraief, A., Belghith, A., Monteil, T., and Drira, K. (2017). How to cache in icn-based iot environments ? In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 1117–1124. IEEE.

[45] Meng, Y., Naeem, M. A., Ali, R., Zikria, Y. B., and Kim, S. W. (2019). Dcs : Distributed caching strategy at the edge of vehicular sensor networks in information-centric networking. *Sensors*, 19(20) :4407.

[46] Morris, J. H., Satyanarayanan, M., Conner, M. H., Howard, J. H., Rosenthal, D. S., and Smith, F. D. (1986). Andrew : A distributed personal computing environment. *Communications of the ACM*, 29(3) :184–201.

[47] Naeem, M. A., Rehman, M. A. U., Ullah, R., and Kim, B.-S. (2020). A comparative performance analysis of popularity-based caching strategies in named data networking. *IEEE Access*, 8 :50057–50077.

[48] Nelson, M., Welch, B., and Ousterhout, J. (1987). Caching in the sprite network file system. *ACM SIGOPS Operating Systems Review*, 21(5) :3–4.

[49] Nxele, N. V. (2017). *Norovirus in children 5 years and below presenting with diarrhoea in KwaZulu-Natal.* PhD thesis.

[50] Nygren, E., Sitaraman, R. K., and Sun, J. (2010). The akamai network : a platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3) :2–19.

[51] Ousterhout, J., Agrawal, P., Erickson, D., Kozyrakis, C., Leverich, J., Mazières, D., Mitra, S., Narayanan, A., Parulkar, G., Rosenblum, M., et al. (2010). The case for ramclouds : scalable high-performance storage entirely in dram. *ACM SIGOPS Operating Systems Review*, 43(4) :92–105.

[52] Patterson, D. A., Gibson, G., and Katz, R. H. (1988). A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116.

[53] Psaras, I., Chai, W. K., and Pavlou, G. (2012). Probabilistic in-network caching for information-centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 55–60.

[54] Reese, W. (2008). Nginx : the high-performance web server and reverse proxy. *Linux Journal*, 2008(173) :2.

[55] Ren, J., Qi, W., Westphal, C., Wang, J., Lu, K., Liu, S., and Wang, S. (2014). Magic : A distributed max-gain in-network caching strategy in information-centric networks. In *2014 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 470–475. IEEE.

[56] Rosensweig, E. J., Menasche, D. S., and Kurose, J. (2013). On the steady-state of cache networks. In *2013 Proceedings IEEE INFOCOM*, pages 863–871. IEEE.

[57] Rossi, D. and Rossini, G. (2011). Caching performance of content centric networks under multi-path routing (and more). *Relatório técnico, Telecom ParisTech*, 2011 :1–6.

[58] Rossi, D. and Rossini, G. (2012). On sizing ccn content stores by exploiting topological information. In *2012 Proceedings IEEE INFOCOM Workshops*, pages 280–285. IEEE.

[59] Sarkar, P. and Hartman, J. (1996). Efficient cooperative caching using hints. In *OSDI*, pages 35–46.

[60] Satyanarayanan, M., Howard, J. H., Nichols, D. A., Sidebotham, R. N., Spector, A. Z., and West, M. J. (1985). The itc distributed file system : Principles and design. *ACM SIGOPS Operating Systems Review*, 19(5) :35–50.

[61] Schapachnik, F., Braberman, V., and Olivero, A. (2002). An architecture-centric approach to the development of a distributed model-checker for timed automata. In *Proceedings of the 24th International Conference on Software Engineering*, pages 710–710.

[62] Seovic, A., Falco, M., and Peralta, P. (2010). *Oracle Coherence 3.5*. Packt Publishing Ltd.

[63] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and Noveck, D. (2003). Network file system (nfs) version 4 protocol. Technical report.

[64] Shrira, L. and Yoder, B. (1999). Trust but check : Mutable objects in untrusted cooperative caches. In *Advances in Persistent Object Systems, Proceedings of the 8th International Workshop on Persistent Object Systems (POS8) and Proceedings of the 3rd International Workshop on Persistence and Java (PJW3)*, pages 29–36.

[65] Stonebraker, M. and Rowe, L. A. (1986). The design of postgres. *ACM Sigmod Record*, 15(2) :340–355.

[66] Suresh, A., Gibson, G., and Ganger, G. (2012). Shingled magnetic recording for big data applications. *Carnegie Mellon University Parallel Data Lab Technical Report CMU-PD L-12–105*.

[67] Tripakis, S. and Yovine, S. (1998). Verification of the fast reservation protocol with delayed transmission using the tool kronos. In *Proceedings. Fourth IEEE Real-Time Technology and Applications Symposium (Cat. No. 98TB100245)*, pages 165–170. IEEE.

[68] van den Haak, E. (2014). Remote data acquisition on block devices in large environments.

[69] Van Hensbergen, E. and Zhao, M. (2006). Dynamic policy disk caching for storage networking. *URL : http ://visa. cs. fiu. edu/ming/dmcache*.

[70] Wang, J. (1999). A survey of web caching schemes for the internet. *ACM SIGCOMM Computer Communication Review*, 29(5) :36–46.

[71] Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D., and Maltzahn, C. (2006). Ceph : A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320.

[72] Woodhouse, D. (2001). Jffs : The journalling flash file system. In *Ottawa linux symposium*, volume 2001. Citeseer.

[73] Xu, Y. and Fleisch, B. D. (2004). Nfs-cc : tuning nfs for concurrent read sharing. *International Journal of High Performance Computing and Networking*, 1(4) :203–213.

[74] Yan, H., Gao, D., Su, W., Foh, C. H., Zhang, H., and Vasilakos, A. V. (2017). Caching strategy based on hierarchical cluster for named data networking. *IEEE Access*, 5 :8433–8443.

[75] Yovine, S. (1997). Kronos : A verification tool for real-time systems. *Int. J. Softw. Tools Technol. Transf.*, 1(1-2) :123–133.

[76] Zhang, G., Li, Y., and Lin, T. (2013). Caching in information centric networking : A survey. *Computer networks*, 57(16) :3128–3141.

[77] Zolfaghari, B., Srivastava, G., Roy, S., Nemati, H. R., Afghah, F., Koshiba, T., Razi, A., Bibak, K., Mitra, P., and Rai, B. K. (2020). Content delivery networks : state of the art, trends, and future roadmap. *ACM Computing Surveys (CSUR)*, 53(2) :1–34.