Université de Rennes

Master Informatique - Ingénierie Logicielle

Rapport de projet DD : Redis appliqué aux échecs

Novembre 2025

Auteurs :
Leboeuf Alexis
Vu Tuan Minh
Rochas Thibaut
Kesteman Amaël

Université de Rennes
Année universitaire 2025–2026

1 Introduction

Depuis la création du jeu d'échecs, celui-ci a toujours été considéré comme un symbole d'intelligence, où l'on considère les meilleurs joueurs de ce jeu comme parmi les personnes les plus intelligentes de notre monde.

Il fut donc logique pour les informaticiens, au fil du temps, de vouloir développer différents programmes d'échecs afin de montrer la supériorité de la machine et de ceux-ci. Un des premiers exemples de cette volonté est le programme Turochamp, développé par Alan Turing et David Gawen Champernowne.

Turochamp a été conçu avant même la création des premiers ordinateurs, ce qui montre l'intérêt des échecs pour pouvoir comparer l'intelligence de la machine face à celle de l'homme.

L'ordinateur a pu battre pour la première fois l'homme lors du fameux match Deep Blue contre Garry Kasparov lors de la revanche de 1997 [5][2] avec un score de 3,5 à 2,5 (3 égalités, 2 victoires et 1 défaite pour Deep Blue). Lors de ce match, Deep Blue a utilisé pour la première fois "l'ancêtre" des Tablebase qui était inspiré d'une base de données Tablebase à 5 pièces créée en 1980 par Ken Thompson. Depuis, différentes Tablebases ont été développées et qui vont jusqu'à 7 pièces sur l'échiquier (7 pièces au total) ; cependant, nous allons nous concentrer sur une Tablebase 3-5 pièces pour des raisons de taille de la base. La table Tablebase 3-5 contient donc toutes les combinaisons possibles de coups jusqu'à une fin de partie, en partant de 3 à 5 pièces sur le plateau.

Actuellement, les ordinateurs ont largement dépassé les joueurs humains et les Tablebases en sont une raison principale car elles permettent de connaître le résultat des finales, peu importe les coups joués par les deux joueurs.

Pour étudier ces bases de données, nous allons nous concentrer sur Redis[4], la base de données key/value[1] la plus utilisée. Et nous allons voir si cette base de données peut être utilisée dans les échecs. Dans un premier temps, nous présenterons Redis, son fonctionnement et son histoire. Puis, dans un deuxième temps, nous expliquerons le fonctionnement du stockage de la position, comment nous pouvons transformer la position du plateau d'échecs en une simple clé. Enfin, nous allons voir les performances d'accès à l'aide de Redis et voir si le nombre de pièces a une incidence sur la performance d'accès à la position.

2 Redis

Redis est une structure de données NoSQL[3] clés/valeurs open source, elle a été lancée en 2009 par Salvatore Sanfilippo, son nom signifie **Re**mote **Di**ctionary **S**erver. Sa caractéristique principale est sa rapidité, car les données sont stockées et manipulées principalement en mémoire vive (RAM), contrairement aux systèmes classiques reposant sur le disque dur. Cette architecture permet des temps d'accès très faibles, ce qui la rend particulièrement adaptée comme cache applicatif.

Dans le cadre de notre étude, Redis est utilisé pour stocker les positions d'échecs dans les Tablebases. Chaque position est représentée par une clé unique dérivée par la notation FEN (Forsyth-Edwards Notation) et les informations associées comme les résultats WDL (Win/Draw/Loss) ou DTZ (Distance to Zeroing) sont stockées en tant que valeurs.

Cette approche présente plusieurs avantages. Le chargement et l'interrogation des positions pré-calculées sont extrêmement rapides, et ce, même lorsque les requêtes portent sur des millions d'entrées. Cela permet d'exploiter les Tablebases comme un "oracle" quasi instantané. Cependant, ces performances reposent sur un coût important : comme Redis stocke l'entièreté de ses données en mémoire vive, la consommation de celle-ci augmente directement avec le nombre de positions chargées. Pour des Tablebases à 3, 4 ou 5 pièces, la taille reste raisonnable, mais au fur et à mesure que le nombre de pièces augmente, le volume de données devient très important. Dans ce cas, l'utilisation de Redis peut atteindre des dizaines ou centaines de gigaoctets de

mémoire, posant un compromis entre performance et ressources matérielles disponibles.

Ainsi, Redis offre une solution particulièrement efficace pour manipuler des positions d'échecs pré-calculées, mais nécessite une maîtrise fine du stockage mémoire et de la structure des données utilisées, surtout lorsque la taille des Tablebases augmente.

3 Stockage

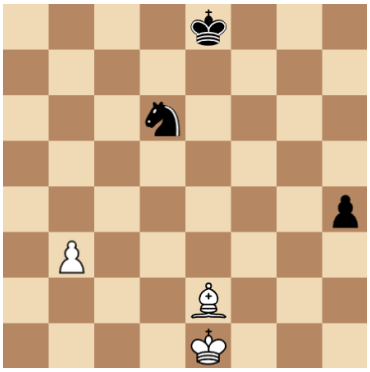
Afin de stocker efficacement les positions d'échecs, qui sont extrêmement nombreuses, nous utilisons la notation FEN (Forsyth-Edwards Notation). Celle-ci compacte encode l'ensemble de l'état d'un échiquier dans une chaîne de caractères unique. Les bases de données de type Syzygy[6][7] utilisent cette FEN comme point de départ : elle est ensuite hachée ou convertie en un index interne afin d'accéder rapidement aux informations de Tablebase.

Dans notre étude, nous utilisons Redis, qui peut stocker cette FEN directement comme clé dans un schéma key/value. Prenons pour exemple la FEN suivante :

```
4k3/8/3n4/8/7p/1P6/4B3/4K3 b - - 0 1
```

Chaque élément de la FEN représente une ligne du plateau. Nous allons pouvoir la décomposer :

- 4k3 pour la 8e ligne indique 4 cases vides, puis le roi noir (k), puis 3 cases vides.
- Les lettres minuscules (k, q, r, b, n, p) représentent les pièces noires, et les lettres majuscules (K, Q, R, B, N, P) les pièces blanches.
- La lettre b indique que c'est au tour des Noirs de jouer.
- Les symboles suivants concernent les règles particulières du jeu, comme le roque, le coup en passant et le compteur de coups depuis le dernier mouvement de pion ou capture.



Voici toutes les lettres correspondant à une pièce :

```
K = roi, N = cavalier, B = fou, R = Tour, Q = Reine, P = pion
```

Par la suite, il y a une lettre b ou w indiquant le jour qui joue le prochain coup. Enfin, il y a 4 caractères par la suite concernant les règles d'échecs particulières qui ne seront pas détaillés ici. Voici un second exemple représentant la position de départ d'une partie d'échecs :

```
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w - - 0 1
```

Utilisation comme clé de stockage

Une fois la position encodée en FEN, on peut utiliser directement comme clé unique dans une base de données.

Dans le cas de Syzygy, la chaîne FEN est transformée en un index interne ou hachée pour pointer vers les données de Tablebase.

Dans Redis, cette même FEN peut servir telle quelle comme clé (string) d'où l'on peut associer des valeurs comme WDL ou DTZ.

Cette approche garantit qu'une position donnée correspond toujours à une clé unique et reproductible, ce qui permet un accès direct à l'information associée, sans ambiguïté.

4 Approche Scientifique et Méthodologie Expérimentale

Cette section décrit la méthodologie utilisée pour évaluer les performances de Redis contre Syzygy dans le stockage et l'accès aux positions d'échecs. L'objectif est d'obtenir des mesures reproductibles, interprétables et valides mais aléatoires, tout en identifiant les biais possibles affectant l'analyse.

4.1 Objectifs expérimentaux

Les expériences menées ont pour but de répondre à ces questions suivantes :

H1 - incidence du nombre de pièces:
Nous supposons que le nombre de pièces n'influe pas sur la performance de Redis.

H2 - Gain de performance:
Nous supposons que Redis, en tant que base de données spécialisée dans les Key-Values, donne de meilleures performances que l'accès direct sur disque aux tables syzygy. Également, puisque Redis stocke ses paires key/value en mémoire vive, nous nous attendons à un grand gain de performances dû au temps d'accès RAM bien inférieur à celui du disque.

H3 - Optimisation des clés via zobrist:
Nous supposons que nous pouvons améliorer le temps d'accès ou l'utilisation mémoire en utilisant le hash de python chess zobrist.

H4 Optimisation des clés via hash classique:
Nous supposons que nous pouvons améliorer le temps d'accès ou l'utilisation mémoire en utilisant le hash classique mais moins efficacement que le hash de python chess.

4.2 Environnement expérimental

4.3 Méthodologie de mesure

Pour vérifier chacune de ces hypothèses, nous allons étudier les différentes métriques sur des jeux de données de test :

1. Temps moyen d'exécution: Mesure du temps pour accéder à une position dans Redis ou Syzygy par les tests stastistiques
2. L'espace mémoire utilisé par Redis: évaluations de consommation en mémoire vive par la libraire de Redis
3. Le nombre de requêtes par seconde
4. Le temps médian et le percentile 95: pour évaluer la dispersion et les valeurs extrêmes

4.4 Génération des positions

Pour chacun de ces tests, nous utiliserons une base de 2250 positions tirées des configurations de tables syzygy que nous avons utilisées. Nous limitons à 2250 afin que chacun puisse tester sur sa machine, quelque soit la taille de sa mémoire vive (RAM).

4.5 Comparatif

Puisque nous aurons le même jeu de données, et que nous aurons plusieurs machines à disposition, nous pourrons comparer, en fonction de la taille de la mémoire RAM, du type de disque et du processeur, le comportement et l'utilisation de Redis, ainsi que le temps de réponse.

Également, si nous en avons la possibilité, nous pourrons alors évoluer vers des jeux de données plus grands ou plus petits afin de récupérer davantage de données, afin d'affiner nos statistiques.

5 Résultats et Interprétations

5.1 Test 1: Incidence du nombre de pièces

Afin de tester si le nombre de pièces présentes dans la configuration a une incidence sur le temps de chargement depuis Redis, nous avons réalisé un script permettant de faire plusieurs requêtes sur le serveur Redis pour 3, 4 et 5 pièces. Nous avons donc un script permettant de faire n requêtes sur Redis pour chaque taille, et nous avons effectué les tests pour plusieurs nombres de requêtes :

- 100
- 1000
- 5000
- 10000
- 50000
- 10000

Pour chaque test, cela nous donne le temps minimal, moyen et maximal pour les requêtes. Après analyse des valeurs obtenues, nous voyons que les valeurs obtenues sont proches, malgré un nombre de tests bien plus importants. De plus, que ce soit pour 3 ou 5 pièces, nous pouvons voir dans la figure 1 que les temps restent environ d'un dixième de seconde. Également, nous pouvons voir dans les graphiques de temps minimum et maximum que les valeurs restent cohérentes. Les quelques valeurs aberrantes sont probablement liées au matériel, plutôt qu'à l'algorithme. En effet, en se documentant, nous avons pu voir que la mémoire DDR4 aux alentours de 3000 MHz pouvait lire environ 20 Go/s [9], donc puisque les tables ne dépassent pas les 20 Mo, alors les résultats obtenus sont cohérents.

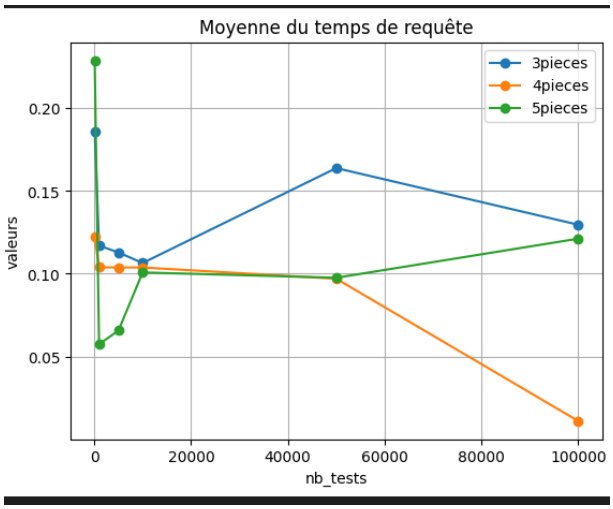


Figure 1: Moyenne du temps de requête en fonction du nombre de pièces et du nombre de requêtes.

En ce qui concerne les temps minimum (Figure 2) et maximum (Figure 3), puisque les accès sont très dépendant du fonctionnement du matériel, alors nous ne pouvons pas réellement tirer d'informations de celles-ci, car ce sont les cas extrêmes, qui ne sont pas réellement représentatifs. Seulement, nous pouvons, en regardant le minimum, voir que ces valeurs restent proche, démontrant une fois de plus l'abscence d'impact du nombre de pièces.

5.2 Test 2: Gain des performances

Après avoir testé Syzygy et Redis sur la même base de données (voir Notebook Python), nous sommes arrivés aux conclusions suivantes: Redis est presque 11 fois plus rapide que Syzygy. (Redis a une latence moyenne de 0.35 ms tandis que Syzygy en a une de 3.82 ms.) Cela se ressent également avec la médiane où Redis a une médiane de 0.33 ms contre 1.06 pour Syzygy. Le gain de performances notable est aussi présent car 1969 rps sont effectués par Redis pour seulement 280 pour Syzygy. Ces résultats sont statistiquement significatifs selon le test t de Student apparié (t = 92.45, p ¡ 0.001) et le test de Kolmogorov-Smirnov (KS = 0.708, p ¡ 0.001), validant ainsi notre hypothèse H2.

5.3 Test 3: Optimisation des clées avec fonction de hashage pour les échecs.

Pour vérifier si une autre structure pouvait possiblement améliorer l'utilisation mémoire, nous avons décidé de tester, en plus de la FEN classique, le stockage via le hash de Python chess. Nous avons pu comparer les valeurs entre la base avec les FEN et la base avec les clés hashées Zobrist à l'aide des benchmarks, et voici ce que nous avons pu observer:

On constate que Zobrist est presque 20 % plus rapide que le stockage en FEN avec une moyenne de 252.5 s contre 308.1 avec les FEN. Celui-ci a également un débit nettement supérieur, il permet d'effectuer en moyenne 55 % de requêtes en plus(3405 contre 2197). Concernant la mémoire, on constate également une amélioration de 8 % avec une clé FEN prenant 117.26 bytes/clé contre 107.64 bytes/clé. Ces résultats sont statistiquement significatifs selon le test t de Student apparié (t = 111.79, p ¡ 0.001) et le test de Kolmogorov-Smirnov (KS = 0.45, p ¡ 0.001), validant ainsi notre hypothèse H3.

5.4 Test 4: Optimisation des clées avec fonction de hashage classique

Pour vérifier l'hypothèse sur l'optimisation des clés, nous avons créé une base de données qui utilise cette fois-ci les clés hashées. Nous avons pu comparer les valeurs entre la base avec les FEN et la base avec les clés hashées et voici ce que nous avons pu observer: La latence moyenne est de 0.709 ms et on peut faire 1068 requêtes par seconde avec la base FEN. Les performances avec le hash sont nettement supérieures, on observe une amélioration de plus de 35 % avec 0.448 ms de latence moyenne et 1496 requêtes par seconde en moyenne. Enfin ce gain n'est pas seulement visible sur les performances mais aussi sur la mémoire utilisée: On estime qu'une clé FEN fait en moyenne 136.46 bytes contre 97.94 en clé hash. Cela nous donne un gain de 28.3 %. Ces résultats sont statistiquement significatifs selon le test t de Student apparié (t = -44.042, p ¡ 0.001) et le test de Kolmogorov-Smirnov (KS = 0.69, p ¡ 0.001), validant ainsi notre hypothèse H4.

6 Conclusion

6.1 Synthèse des résultats

Avec l'aide de nos expériences, nous avons pu montrer que d'autres solutions plus rapides que les solutions actuellement mises en place par les sites d'échecs existent. En effet, nous avons vu que Redis est nettement plus rapide que Syzygy et qu'il offre également un débit presque 7 fois supérieur. Il en va de même pour la façon de stocker les positions.Si l'on garde le système de stockage classique des échecs, on ralentit encore plus le système, car les fonctions de hashage nous permettent également de gagner en performance et en mémoire, avec pour chacune des fonctions, une augmentation de minimum 18% en termes de débit mais aussi d'un gain en mémoire conséquent avec plus de 25% pour la fonction de hashage classique. La question se pose alors de pourquoi l'on n'utilise pas Redis et le hashage. Utiliser le hashage reviendrait à perdre de l'information, ce qui est dans ce cas contradictoire comme c'est justement ce que l'on cherche à stocker. Enfin, le gain entre Redis et Syzygy est minime et n'est pas pertinent pour plusieurs raisons : Redis en tant que système in-memory est inadapté pour les Tablebases qui font des centaines de Go, il faut donc un système sur disque où Syzygy est bien meilleur grâce à la compression. L'accès aux Tablebases est aussi très rare et se fait de manière ciblée, sur une position et s'il y a un changement, cela se tourne vers une position très similaire, le gain de performances de Redis est donc moins pertinent dans le véritable usage. Enfin, il ne serait pas possible économiquement de maintenir un tel système avec Redis.

Cette étude nous a néanmoins permis de comparer les différences de performances entre les approches disques (Syzygy) et les approches mémoires (Redis), de démontrer le poids du choix de la structure des clés et de mieux comprendre quel système doit être utilisé selon le contexte.

Pour conclure, bien que Redis et le hashage offrent des gains non négligeables, le système actuel utilisant la FEN et Syzygy reste le meilleur en s'appuyant sur l'efficacité du stockage et la maintenabilité que la performance brute dans le cadre du stockage complet des Tablebases.

6.2 Limites de l'étude

Même si les résultats obtenus sont informatifs, plusieurs aspects doivent être pris en compte. Tout d'abord, notre étude ne s'appuie que sur un sous-ensemble réduit des véritables tablebases Syzygy : seules les configurations jusqu'à 5 pièces ont été testées, ce qui exclut les tablebases plus volumineuses à 6 et 7 pièces. L'échantillon utilisé reste donc limité et ne permet pas de tirer des conclusions définitives sur la performance à grande échelle.

Les expériences ont été menées sur nos machines personnelles plutôt que sur des serveurs dédiés ou des configurations hautes performances utilisées par certaines organisations spécialisées.[8] Cela peut influencer à la fois la consommation mémoire observée et les temps d'accès, et limite la généralisation des résultats.

Enfin, notre analyse s'est concentrée uniquement sur les aspects performance (latence, RPS, mémoire vive). D'autres dimensions importantes des tablebases — telles que la fiabilité, la compression, l'intégrité des données, ou encore les implications pratiques pour les moteurs d'échecs — n'ont pas été explorées dans cette étude.

À ces limitations s'ajoute une contrainte méthodologique liée à la mesure de la mémoire : Redis stocke et traite ses données exclusivement en mémoire vive, ce qui rend impossible une séparation nette entre mémoire de stockage et mémoire d'exécution. Pour Syzygy, nous avons dû mesurer la mémoire du processus via la bibliothèque psutil, tandis que pour Redis nous n'avons accès qu'à la consommation totale rapportée par le serveur. Nous avons observé que Syzygy a utilisé environ 205Mo de RAM pour exécuter et lire des tablebases qui est 21,6 Mo, tandis que Redis a consommé seulement environ 169 Mo pour le même jeu de données. Toutefois, ces valeurs sont fortement dépendantes de la taille réduite du dataset utilisé. Sur des bases plus importantes, la consommation de Redis pourrait croître beaucoup plus rapidement, rendant les

comparaisons moins triviales.

6.3 Ouverture (Tablebases à 8 pièces)

Depuis 2012, Les Tablebases à 7 pièces existent, mais il a fallu attendre août 2018 avant qu'elles soient complètes. Ceci illustre la difficulté du jeu d'échecs et montre pourquoi, au contraire d'autres jeux comme le jeu de dames, ne sera pas résolu. Actuellement, les Tablebases à 8 pièces ont 15 % des finales sans pions résolues. [10] Ceci montre la difficulté de la progression pour finir les Tablebases. De plus, on estime la taille totale à plusieurs Pétaoctects, ce qui ne serait même pas utilisable pour la majorité des ordinateurs, même les plus puissants. Ceci vient du fait que la base serait 90 fois plus grande que la base à 7 pièces. Enfin ceci aurait seulement un intérêt théorique et changerait juste la vision de certaines finales (certaines fins de partie autrefois considérées comme perdantes ont été classifiées comme nulle avec l'apparition des tablebases).

7 Annexe

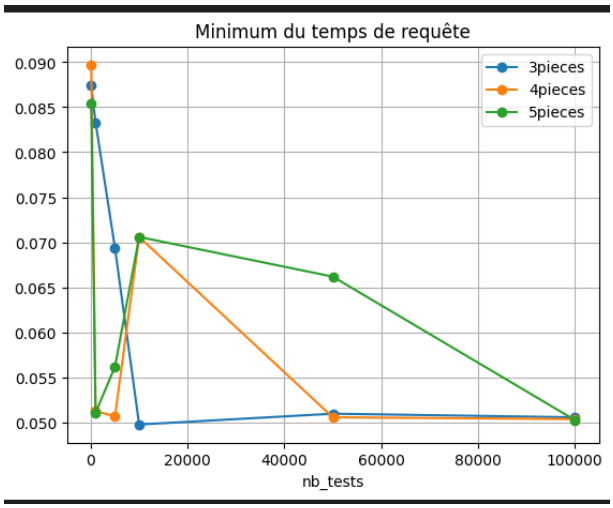


Figure 2: Minimum du temps de requête en fonction du nombre de pièces et du nombre de requêtes.

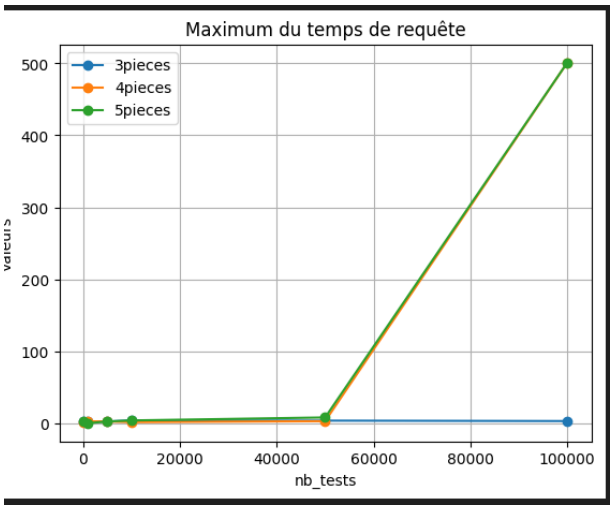


Figure 3: Maximum du temps de requête en fonction du nombre de pièces et du nombre de requêtes.

8 Glossaire

FEN Notation standard utiliser dans le monde des échecs pour exprimer une position en chaîne de caractères.. 1

Tablebase Base de données exhaustive, contenant toutes les possibilités dans un monde limité ainsi donc que leur résultat optimal.. 1

9 Références

[1] Key/value. https://en.wikipedia.org/wiki/Key%E2%80%93value_database.

[2] Matchs deep blue contre kasparov. https://fr.wikipedia.org/wiki/Matchs_Deep_Blue_contre_Kasparov.

[3] Nosql. https://fr.wikipedia.org/wiki/NoSQL.

[4] Redis wikipedia. https://en.wikipedia.org/wiki/Redis.

[5] 1997 : l'ordinateur bat garry kasparov, un tournant dans l'histoire des échecs, May 11, 2022. https://www.ina.fr/ina-eclaire-actu/1997-l-ordinateur-deep-blue-bat-garry-kasparov-un-tournant-dans-l-histoire-des-echecs.

[6] Ronald de Man. Syzygy engame tablebases, April 01, 2013. https://syzygy-tables.info/.

[7] Ronald de Man. Syzygy engame tablebases github repository, April 01, 2013. https://github.com/syzygy1/tb.

[8] Fédération Française des échecs. Microsoft azure, partenaire de la fédération française des Échecs au service de la performance, December 12, 2022. https://www.echecs.asso.fr/Actu.aspx?Ref=14252.

[9] Jean-Pierre Novak. Taux de transfert, bande passante, latences et voltage des mémoires ram ddr4, October 2, 2023. https://www.eatyourbytes.com/fr/debit-vitesse-latences-et-voltage-des-memoires-ram-ddr4.

[10] Peter Wong. Eight-piece tablebases – a progress update and some results, August 29, 2025. https://www.chess.com/blog/Rocky64/eight-piece-tablebases-a-progress-update-and-some-results.