Université de Bourgogne

Rapport Systeme Distribuer

Enseignants :Eric Leclercq et Annabelle Gillet Auteurs : DIARRA Ibrahim

14 septembre 2023

Table des matières

1 Introduction

Le sujet qui a été choisi est le sujet numéro 4. Il parle du jeu de la vie. Ce dernier est un automate cellulaire imaginé par John Horton Conway en 1970. Malgré des règles très simples, il est Turing-complet (c'est à dire qu'il possède un pouvoir expressif au moins équivalent à celui des machines de Turing.). C'est un jeu de simulation au sens mathématique. Il y'a 0 joueur. Le jeu de la vie utilise habituellement un plateau à deux dimensions infinis dans lequel chaque case contient 0 ou 1(pour mort ou vivant). On calcule l'état que va prendre une case à la génération suivante en prenant en compte son état et celui de ses 8 voisins. Les règles sont les suivantes :

1. Si la case possède exactement 3 voisins vivants(cad à 1), elle nait.
2. Si la case possède exactement 2 voisins vivants , elle reste dans son état actuel.
3. Si la case possède plus de 3 voisins vivants ou moins de 2 voisins vivants, elle meurt.

Ces règles sont celles de Conway mais on peut utiliser nos propres règles et notre propre plateau.

2 Annalyse du sujet

Dans notre cas, nous utiliserons un plateau à trois dimensions au lieu du 2 habituelle. Les deux chiffres utiliser par les regles seront modifiable par l'utilisateur ainsie que la taille du plateau. Ce dernier aura une taille fini avec les bords à 0 sur les 3 dimensions. Toutes ces modifications du jeu d'origine augmentent drastiquement les calcules à effectuer pour chaques générations du plateau. Le but est de pouvoir lancer la simulation(le jeu) sur un systeme distribuer. C'est à dire que l'application doit pouvoir utiliser plusieur machines pour accelerer les calcules.
Pour de tels applications, plusieur paradigmes existent. Chacun possede ses caracteristiques, ses point positifs et négatifs.
Il y'a :

1. MPI(Message Passing Interface) pour l'envoie de messages
2. Java RMI pour les objets distribués.
3. Akka avec son modele d'acteur.

Il existe également jms et map-réduce mais ils ne sont pas adaptés à notre application. Jms sera plutot utilisé pour des applications avec un temps de réponse long . Quant à map-reduce, il sera utilisé pour des applications traitant des données massives sur lesquelles les mises à jours sont rares.

Cela nous conduit naturelement à se demander quel est le meilleur paradigme pour réaliser notre application entre MPI,java RMI et akka ?

2.1 Paradigme rétenu

Le modele d'acteur de akka permetrait d'introduire le parallélisme des traitements et d'éviter la gestion de la concurrence. Pour une application avec de nombreux acces conccurent aux réssources et dont la gestion de la concurence est une tache hardu, ce paradigme serait l'ideal. Pour notre application, cela n'est pas le cas. Akka n'est donc pas le paradigme retenu.

Ensuite il y'a MPI qui nous permettrait d'implementer le modele SPMD (Single Program Multiple

Data) . Ce paradigme semble être le meilleur pour notre application. Il apporte de grandes performances car (on peut l'utiliser en c+-+). L'inconvenient réside dans sa logique de programmation. L'implémentation d'algorithmes utilisant MPI est une tache hardu contrairement à javaRMI qui est un plus simple à manipuler. C'est la raison principale qui fait que MPI n'a pas été choisie pour ce projet.

Le paradigme choisi pour ce projet est donc javaRMI.

1. Dans un prémier temps,sa simplicité permet d'implementer facilement les algorithmes. Les objet distribuer se malipulent de la meme façon q'un objet local. La programmation reste rélativement proche du non distribuer.
2. Dans un second temps, le fait que ce soit en java permet d'utiliser tous les mecanismes de ce dernier(heritage,encapsulation,polymorthysme,etc...) et offre également une grande capacité de transparence à l'hétérogénéité.
3. Bien qu' il ne soit pas au niveau de Mpi, il offre cépendant des pèrformances correctes pour notre application.

2.2 algorithmes

Pour chaque case du plateau, il faut appliquer les règles afin de déterminer l'etat de la case à la géneration suivante.
Pour un plateau de 3x3x3 cela revient à effectuer 27 fois ce calcule. Pour un plateaux de 50x50x50 cela donne 12500 et pour un plateau de 100x100x100 cela donne 1000000. Le nombres de calcules à effectuer explose d'autant plus quand le nombre de génération à calculer est grand. En effet, les chiffres précédents répresentent le calcule de seulement une génération. Donc pour le plateau de 100x100x100,1000 génération donnera 1000000000. Mais on peut s'appercevoir assez facilement que ces calcules sont parallelisables.

2.2.1 Algorithme de la distribution

Pour notre application, nous utiliserons un algorithme simple qui accelère les calcules en parallelisant ces derniers.
Son principe est le suivant : Plusieur machines calculeront chacune une partie du plateau au meme moment. Ainsie le temps de calcule devrait être grandement diminuer. IL exitent des algorithmes plus efficace mais beaucoup plus complexe à mettre en oeuvre(comme hashlife). Notre but n'est pas d'implementé le meilleur des algorithmes mais d'accelerer neanmoins les calcules afin de trouver des configurations stable du jeu de la vie en 3d. Pour l'acceleration, on peut proceder de la façon suivant :

L'une de nos machines sera un serveur de taches et les autres des worker. Le serveur de taches disposera d'un bagOfTask.

1. Le Plateau est découpé en un certain nombre(fixer par l'utilisateur) de tache par le serveur . Chaques taches contient un certain nombre(fixer par l'utilisateur) de cases à calculer.
2. Les workers démandent la tache suivante au serveur, l'execute ,renvoie la réponse et rédemande la tache suivante.

3. Quand le calcule de la génération suivante est terminé et que la génération cible n'est pas encore atteinte,le resulat est rédecouper en taches par le serveur et les workers calculent la génération suivant.
4. Une foi la génération cible atteinte, le resultat est afficher et les worker sont stopés mais le serveur est toujour actif .
5. Quand on relance les workers une générations en plus est calculer.

2.2.2 Algorithme de decoupage du Plateau

Pour découper le plateau en t taches contenant chancune c cases ainsie que le necessaire(pour chaque case il y'a aussi ses voisins) pour le calcule,nous allons utiliser l'algorithme suivant :

1. on parcourt les cases à calculer(les bord de chaque dimensions du plateau ne changent jamains).
2. pour chaque case, on l'enregistre ainsie que ses voisins. cela constitue une sous-tache.
3. pour chaque tache, on enregistre le nombre de sous-tache necessaire défini par l'utilisateur. Puis on enregistre la tache dans le bagoftask.
4. S'il reste des cases dont le nombre ne permet pas de faire une tache de la taille défini, ils formerront la dernière tache.

2.2.3 Algorithme de traitement

Pour traiter une tache , un worker procède de la maniere suivante :

1. il parcour chaque sommets de la tache
2. Et pour chaque sommets, il addittionne la valeur des voisins.
3. Ensuite il verifie les conditions en fonctions des parametttres de vie et de mort entrer par l'utilisateur au moment de lancer les workers.
4. Enfin il retourne le resultat de la tache.

3 Classes principales et fonctions principales

3.1 BagOfTask

Comme sont nom l'indique, C'est le bagoftask. Cette classe est utilisé par le serveur pour découper le plateau en taches et stocker ces dernier. Les taches sont stocké à l'aide d'une table de hachage. Cela permet de retrouver une tache tres rapidement grâce à sa clé.

Les principales methodes de cette classe sont les suivantes :

1. la methode ***diviser*** qui permet de decouper le plateau en taches.
2. la methode ***voisin*** qui est utiliser par la methode ***diviser*** pour trouver tous les voisins d'une case.
3. la methode ***getTackSuivante*** retourne la tache suivante à calculer.

3.2 Tache

Cette classe represente une tache. Elle dispose d'une liste de donnée(tableau avec un sommet et ses voisin) ansie qu'une liste de la position pour chaque case . Cette classe encapsule tous les donné necesaire à une tache.Elle implemente l'inteface serializable pour permettre sa mobiliter sur le reseaux.
Les principales methodes de cette classe sont les suivantes :

1. la methode ***traiter*** qui sera executer par un worker. c'est la methode implementant les calcules à effectuer pour une tache.
2. la methode ***addElement*** qui est utiliser par la methode ***diviser*** du bagoftasks pour ajouter une sous-tache à une tache.

3.3 LifeGame

Cette classe est une interface utiliser pour la mise en place de l'objet distant. Elle etant la classe Remote .Cela permet aux objets qui vont l'implementer d'etre à leur tour remote .Seule les objet remote peuvent être distribuer avec javaRMI.
Les principales methodes de cette interface sont les suivantes :

1. la methode ***getTask*** qui sera executer par un worker. C'est la methode qui permetrat à ce dernier de demander et de recuperer la tache suivante au serveur.
2. la methode ***ReturneRes*** qui est utiliser par les worker pour returner le resultat d'une tache.

On peut constater que la communication entre les workers et le serveur se fait grâce à ces deux méthodes. La communication n'est pas gérer par le programmeur mais par RMI lui même .

3.4 Client(Worker)

Cette classe represente un worker. Elle est utiliser pour faire travailler une machine pour le serveur. Pour ajouter un worker , il suffi de lancer le client sur une nouvelle machine.
Cette classe ne dispose que d'une fonction main. Cette derniere utilise l'api Jndi(java Naming and Directory Interface) pour se connecter au rmiregistry du serveur et recuperer le stub necesaire à l'appel de methodes distante.L'interface LifeGame sera utiliser pour acceder au methodes du serveur .
Apres cela , 3 chose sont executer en boucle justqu'a la génération cible.
la methode getTask pour recuperer une tache , la methode traiter de la tache recuperer pour faire les calcule,et la methode ReturneRes pour renvoyer le resultat.

3.5 Serveur

C'est la classe qui centralise la plupart des objet. C'est elle qui implemente l'interface LifeGame. Elle dispose de 2 bagoftask. A la fin du calcule d'une generation, un "swap" de bagoftask est effectué. L'ancienne est reenitialisé et la generartion suivante utilise l'autre bagoftask. Cela permet de rendre possible les simulations "infini".Autrement, la simulation s'arrete apres 4 294 967 296(limite d"un int) tache car la cle de la table de hachage utiliser par le bagoftask est un entier(interger).

L'attribut courant permet de savoir à tous moments quel bagoftask est utiliser.
L'attribut grain represente le nombre de sommet par tache. Il est specifier par l'utilisateur.
L'attribut generation répresente le nombre de generation à calculer. Il est egalement specifier par l'utilisateur.
l'attribut start sera utiliser pour les satistiques et aidera à determiner le temps de traitement.

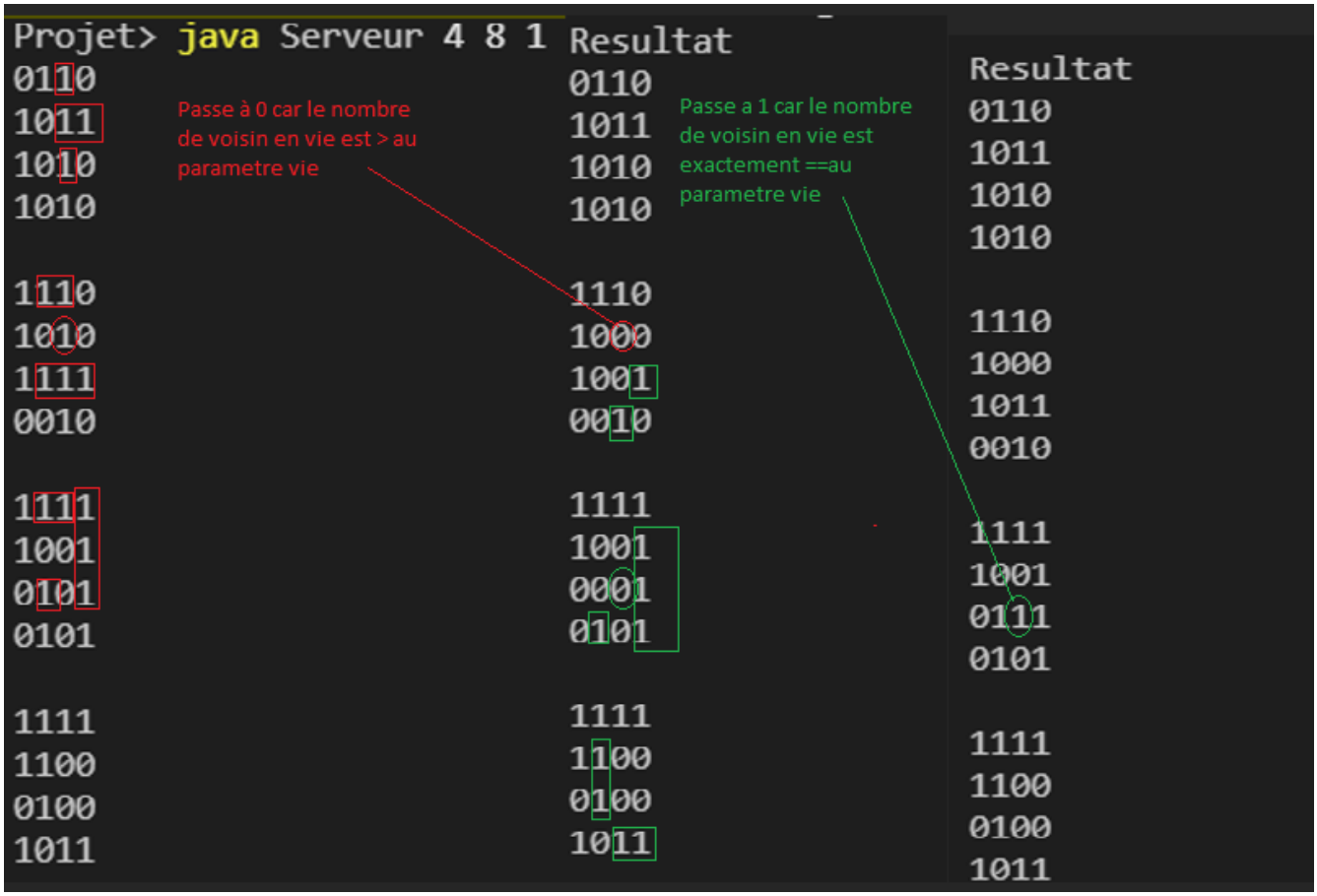
Les principales methodes de cette classe sont les suivantes :

1. Il y'a La methode getTask() qui va cherche la tache suivante dans le bagoftask courant et la returner au worker qui à appeler la méthode.
2. Il y'a La methode returnRes() qui va permetre aux workers de retourner leur resulats. Elle permet egalement au serveur d'effectuer l'operation de swap si on est à la fin d'une génération,d'arreter les workers si la génération cible est atteinte.

Ces deux méthodes sont synchroniser car ils répresentent des sections critique.

4 jeu de test

Sur ce test on peut voir la generation 0,1,2 d'un tableau calculer avec 1 worker ayant pour parametre vie 10 et mort 6.



5 evaluation/comparaison des performances

Ces testes ont été realisé sur un seule ordinateur en ouvrant plusieur termineaux.

Nombre de point	Nombre de jvm	temp
12.5 million	1	11s
12.5	2	6s
12.5	4	6s

6 Documentation pour compiler/executer

6.1 le Serveur

Les classes necessaires pour le serveur sont :

1. Serveur.java
2. LiveGame.java
3. BagofTask.java
4. Tache.java

Pour compiler le serveur et l'executer

1. javac Serveur.java
2. java Serveur argument1 argument2 argument3

argument1 : C'est la taille du tableau de la simulation. Donner la taille que vous souhaiter +2. Pour un tableaux de 20x20x20 donner 22et pour un tableau de 50x50x50 donner 52. Le tableau sera génerer de facon alleatoir.

argument2 : C'est le grain. Le nombre de point par tache. Pour des performances optimale, veuillez donner comme grain la (taille x taille x taille)/(nombre de worker prevu). Par exemple pour une taille de 50= (argument1-2) et 2 worker donner (50x50x50)/2 ce qui fait 62500 et pour 4 worker cela fait 31250.
Si c'est plus petit , le nombre de communication réseaux augmente et si c'est plus grand certain worker auronts plus de travaille que d'autres.

argumen3 : C'est le nombre de génération. Il doit etre strictement superieur à 0.

6.2 le Client

Les classes necessaires pour le client sont :

1. Client.java
2. LiveGame.java
3. Tache.java

Pour compiler le Client et l'executer coorectement , modifier :

1. Modifier Client.java .Mettre l'adresse ip de la machine sur lequel à été lancer le serveur a la place de "localhost" à la ligne 5.
2. javac Client.java
3. java Client argument1 argument2

argument1 : C'est le parametre de vie pour le calcule de l'etat d'une case .

argument2 : C'est le parametre de mort pour le calcule de l'etat d'une case.

Pour le jeux de la vie de Conway vie =3 et mort =2 ;. Pour ce projet tous les testes ont été effectuer avec vie =10 et mort =6 . Tous les worker doivent avoir les mêmes parametres pour rester coherent avec le jeux de la vie classic. Si c'est n'est pas le cas ,certaines partie du plateau seront favorisées(auront plus de chance de vivre) par rapport à d'autres.
Dans tous les cas, Vie doit etre strictement superieur à (mort +2) et mort doit etre strictement superieur à 0.

Quand la simulation se termine, le resulat est afficher sur le serveur. il est possible d'afficher les résulats intermediaire en décommentant les lignes 66 et 80 de Serveur.java mais cela reduit considerablement le temps de calcule.
La ligne 85 permet d'avoir une idée du temps de calcule. il ne prend pas en compte le temps de lancement des worker et commence a compter des que le seurveur est lancer.
La ligne 86 est l'affichage finale. Pour une meilleur visibiliter tu temps commenter cette ligne le temps sera le premier chiffre qui saffiche .
La ligne 112 permet d'afficher le tableux de la generation 0.

7 Conclusion

Au terme de cette étude , on peut affirmer que ce projet fut des plus instructifs. Il nous à permit d'approfondire nos connaisance sur les systems distribuer notament sur l'algorithmique distribuer. Il nous a permit de ce confronter aux nombreux defits que ce dernier apporte.