Date : 02/04/2023 Groupe : IE-I92

Rapport du projet Info4B

Calcul distribué de la persistance additive et multiplicative des nombres

Ont participé à la réalisation du projet :

Saman ISMAIL

Delshad KADDO

Tristan HOARAU

Table des matières

Introduction

Nou somme 3 étudiants en 2ième année de licence informatique et électronique à l'université de Bourgogne et suivons la matière Info4B (Principes des systèmes d'exploitation) et dans le cadre de cette matière nous avions le choix entre 2 sujets pour réaliser un projet qui nous permettra de mettre en œuvre nos connaissances théoriques.

Le projet se compose de deux sujets distincts : d'une part, le calcul des persistances multiplicatives, et d'autre part, la réalisation d'une version informatique du jeu Burger Time, un jeu classique des années 80.

Pour notre projet, nous avons opté pour le premier sujet portant sur les persistances multiplicatives et afin d'enrichir notre travail, nous avons intégré d'autres concepts notamment la persistance additive en raison de notre groupe de travail composé de trois personnes.

Présentation et objectifs du projet

Le projet des persistances additives et multiplicatives consiste à exploiter la puissance de plusieurs machines pour répartir le traitement d'un grand nombre de tâches de calcul de persistance multiplicative et additive.

La persistance d'un nombre correspond au nombre de répétitions du processus consistant à multiplier ou additionner tous ses chiffres jusqu'à obtenir un nombre à un seul chiffre. Le système sera composé d'un serveur qui enverra des requêtes et les distribuera à des machines distantes appelées "workers".

Les workers se connecteront au serveur pour indiquer leur disponibilité.

Le serveur utilisera des hashtables pour conserver ces informations. L'objectif est d'optimiser le traitement des tâches en répartissant au mieux la charge entre les différents workers. Ce projet permettra de bénéficier d'une grande capacité de calcul en utilisant plusieurs machines en parallèle.

Un programme client permettra l'affichage des résultats en se connectant au serveur, il peut afficher des statistiques telles que la moyenne et la médiane de la persistance, le nombre d'occurrences par valeur de persistance.

Le client peut également demander les résultats sur un intervalle spécifique, consulter la persistance d'un nombre, la liste des nombres avec la plus grande persistance ainsi que l'état du serveur.

Plusieurs clients peuvent se connecter au serveur en même temps.

Justification du choix du sujet 1 : calcul distribué de la persistance des nombres

Tout d'abord, le calcul de la persistance multiplicative et additive des nombres est un problème mathématique intéressant et non résolu. Bien que l'on sache que la persistance maximale est de 11 pour les nombres inférieurs à 10^333, nous ne connaissons pas encore la valeur exacte de cette persistance pour tous les nombres.

Ensuite, la mise en place d'un système de traitement distribué permet de bénéficier d'une grande capacité de calcul en utilisant plusieurs machines en parallèle. Cela permet de réduire le temps nécessaire pour effectuer les calculs, ce qui est particulièrement important compte tenu de la taille des nombres à traiter.

Enfin, la mise en place d'un système de traitement distribué nécessite la mise en place de techniques de gestion de la charge et de répartition des tâches, ce qui est un défi intéressant en soi.

La mise en place d'un tel système peut également être utile pour d'autres applications nécessitant le traitement de grandes quantités de données.

Analyse fonctionnelle

Principe de la persistance multiplicative et additive des nombres

Le principe de la persistance des nombres est une propriété mathématique intéressante qui consiste à multiplier tous les chiffres d'un nombre donné, puis à répéter ce processus avec le résultat obtenu jusqu'à obtenir un seul chiffre. La persistance d'un nombre est le nombre d'itérations nécessaires pour obtenir ce chiffre unique.

Par exemple, prenons le nombre 77. La première itération consiste à multiplier 7 x 7, ce qui donne 49. Pour la deuxième itération, nous multiplions les chiffres de 49 : 4 x 9, ce qui donne 36. Nous continuons le processus avec les itérations suivantes : 3 x 6 = 18, 1 x 8 = 8. Ainsi, nous avons obtenu un seul chiffre après 4 itérations. Par conséquent, la persistance du nombre 77 est de 4.

Il existe des nombres qui ont une persistance très élevée, c'est-à-dire qu'ils nécessitent un grand nombre d'itérations avant d'obtenir un chiffre unique. Par exemple, le nombre 277777788888899 a une persistance de 11, ce qui signifie qu'il faut effectuer 11 itérations pour obtenir un chiffre unique.

En plus de la persistance des nombres, il existe également la persistance additive, qui consiste à ajouter les chiffres de la persistance d'un nombre donné jusqu'à obtenir un seul chiffre.

Il est intéressant de noter que pour certains nombres, les persistances ont des propriétés intéressantes. Par exemple, tous les nombres ayant une persistance de 1 ont une addition de persistance de 1 également. De plus, il a été conjecturé que tous les nombres ayant une persistance maximale de 11 ont une persistance additive de 2. Ces propriétés et conjectures rendent le sujet de la persistance des nombres encore plus fascinant en mathématiques.

La persistance des nombres est un sujet fascinant en mathématiques, et de nombreuses conjectures ont été proposées à ce sujet. Par exemple, on ne sait pas s'il existe un nombre avec une persistance de 5 ou plus qui n'est pas divisible par 10.

Architecture logicielle globale

Serveur : cette couche reçoit les requêtes des clients, envoi des requêtes aux workers et stocke les résultats retournés par ces derniers sur le disque dans le dossier Additive ou Multiplicative. Le serveur maintient également des informations sur l'état des workers et les résultats enregistrés. Il utilise des hashtables pour conserver ces informations.

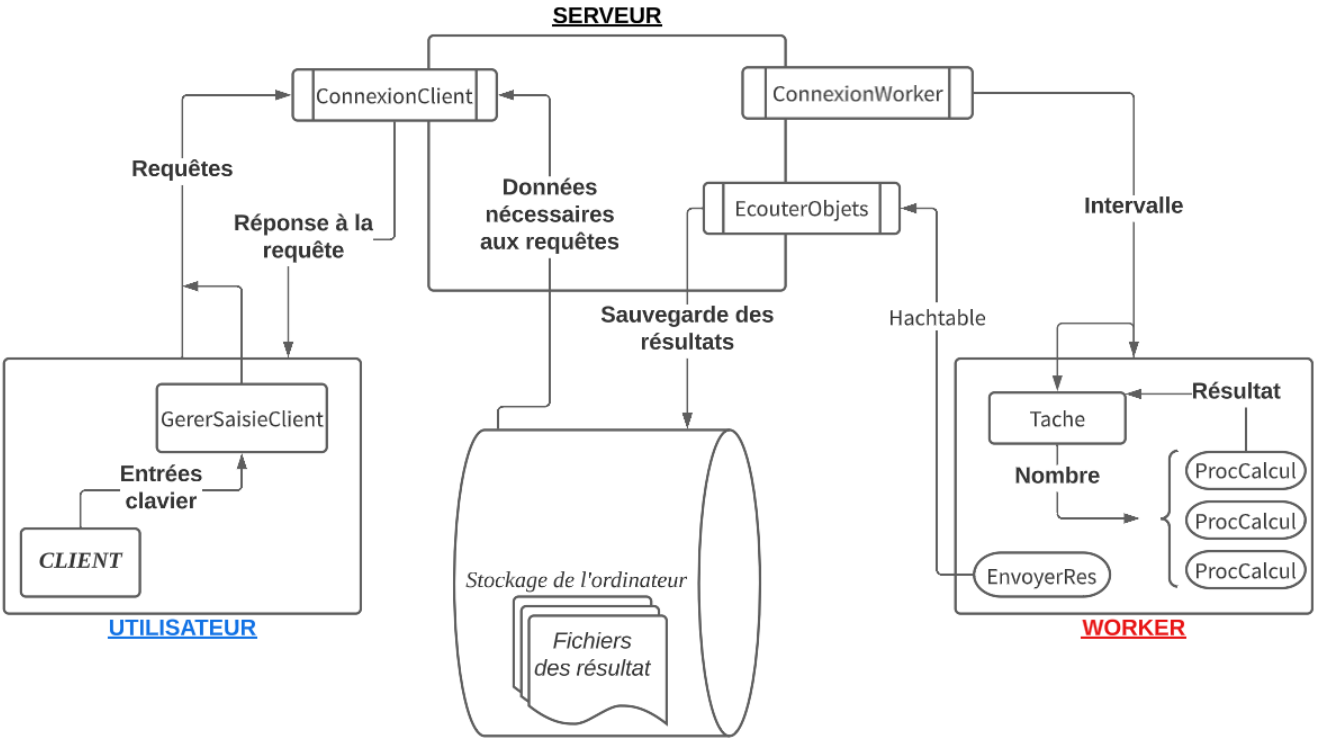
Workers : cette couche est constituée de machines distantes qui exécutent les requêtes envoyées par le serveur sur un intervalle initialisé dans le serveur. Chaque worker peut exécuter plusieurs tâches en parallèle, selon le nombre de cœurs disponibles.

Client : cette couche est responsable de l'interaction avec le client et de l'affichage des résultats. Elle permet aux clients de se connecter au serveur, il n'y a pas d'interface pour le client, tout est fait dans la console et plusieurs clients peuvent se connecter simultanément.

Disque : cette couche est utilisée pour stocker le résultat des calculs effectués par les Workers. Le Worker ne peut pas écrire directement sur le disque, tout passe par le Serveur qui se charge de créer des fichiers hashtables.

Tâche : cette couche est utilisée par les Workers pour calculer les deux persistances des nombres et envoi le résultat au Serveur. Les Workers créent des tâches à exécuter eux-mêmes.

Cette architecture permet de répartir efficacement la charge de calcul sur plusieurs machines, d'assurer la disponibilité et la résilience du système, ainsi que de stocker les résultats produits pour une utilisation ultérieure.



Structures de Données

Structures de données envisagées

Pour ce projet, plusieurs structures de données sont envisagées afin de stocker et manipuler les informations nécessaires au calcul des deux persistances.

Tout d'abord, il est prévu d'utiliser des **Tableaux** de différents types qui seront utilisés pour conserver des informations sur les workers et les clients par exemple leurs adresses IP, les ports d'écoute des Workers, les PrintWriter des différents clients et workers et la liste des Workers disponibles.

Pour représenter les nombres de grandes tailles impliqués dans les calculs, la structure de données **BigInteger** de Java sera utilisée. Elle permet de représenter des entiers de taille arbitraire et de réaliser des opérations arithmétiques avec une grande précision.

De plus, les **Hashtables** seront utilisées par les Workers pour stocker les nombres et leurs résultats puis elles seront utilisées par le Serveur qui va se charger de créer des fichiers sur le disque et y stocker les Hashtables reçues par les Workers.
Cette structure de données est particulièrement adaptée pour la recherche et la récupération efficaces des résultats en utilisant les clés correspondantes.

Enfin, nous créons un objet **Hachtable** qui implémente la classe Serializable pour envoyer les résultats depuis le Worker vers le Serveur. Bien qu'un objet créé par l'utilisateur ne soit pas considéré comme une structure de données, la classe Hachtable peut être perçue comme telle car elle ne contient que des structures de données telles que les objets BigInteger et d'autres objets Hashtable (persistance additive et persistance multiplicative) qui stockent les résultats des calculs de persistances.
Il s'agit en réalité d'une implémentation d'une structure de données particulière.

Justification des choix

En ce qui concerne le choix de la structure de données pour représenter les nombres de grandes tailles impliqués dans les calculs, nous avons opté pour la classe BigInteger de Java plutôt que pour le type primitif int.
En effet, la classe BigInteger permet de représenter des entiers de taille arbitraire, tandis que le type int est limité à des valeurs entre -2 147 483 648 et 2 147 483 647.
Dans le cadre de notre projet et la démonstration, nous n'arriverons pas à montrer ces valeurs mais nous avons pensé que cela serait possible si nous mettons ce projet en ligne et que plusieurs Workers distants se connectent, nous serons susceptibles de manipuler des nombres beaucoup plus grands que cette plage de valeurs, d'où le choix de la classe BigInteger.
De plus, la classe BigInteger offre des méthodes pour effectuer des opérations arithmétiques avec une grande précision, ce qui est essentiel pour le calcul des persistances.

Les Hashtables ont été choisies pour stocker les résultats des calculs de persistances pour plusieurs raisons. Tout d'abord, elles offrent un accès rapide aux résultats à partir de leurs

clés, ce qui est important dans le cadre de ce projet où il peut y avoir de grandes quantités de données à stocker et à récupérer. De plus, elles permettent une gestion efficace des collisions, grâce à leur mécanisme de hachage. Les Hashtables offrent également une grande souplesse dans la définition des clés et des valeurs et c'est ce qui est demandé dans le cahier des charges de ce projet

Spécification des packages, des classes et explication des statics

Les packages

Dans ce projet, nous avons choisi de diviser le code en quatre packages distincts : ClientPackage, ServeurPackage, WorkerPackage et Commun.

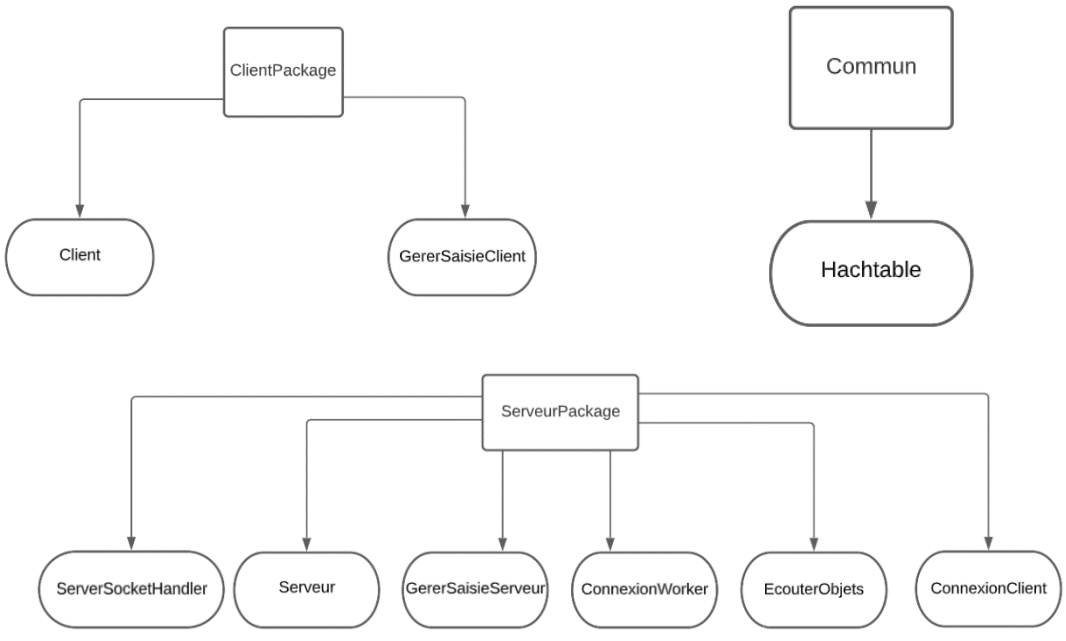
Le package ClientPackage contient toutes les classes nécessaires à la mise en place d'un client permettant de se connecter au serveur et d'envoyer des requêtes de calcul de persistances. Il contient notamment les classes pour la création de sockets de connexion, la gestion des flux de données entrants et sortants, et la gestion des erreurs de connexion.

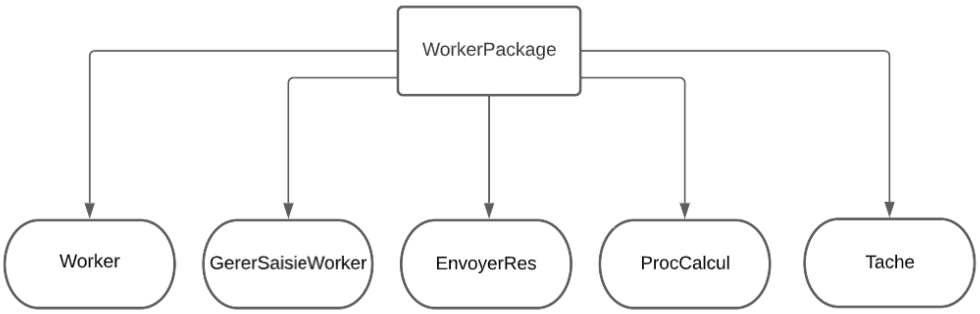
Le package ServeurPackage contient toutes les classes nécessaires à la mise en place du serveur qui sera chargé de recevoir les requêtes et d'effectuer des recherches en fonction des requêtes des clients et d'envoyer des requêtes aux Workers pour le calcul des persistances. Il contient notamment les classes pour la création des sockets de connexion, la gestion des requêtes des clients, la gestion des workers, la création des répertoires ou fichiers nécessaires et le stockage des résultats des calculs.

Le package WorkerPackage contient toutes les classes nécessaires à la mise en place des workers qui seront chargés d'effectuer les calculs de persistances pour le serveur. Il contient notamment les classes pour la création des sockets de connexion avec le serveur, la gestion des requêtes de calcul de persistances et l'envoi des résultats au serveur.

Enfin, le package Commun contient la classe Hachtable qui sera accessible par le Serveur et par le Worker pour pouvoir faire la sérialisation et la désérialisation correctement.

L'utilisation des packages nous permet de voir le code plus clairement et offre la possibilité de réutiliser certaines classes dans d'autres projets.





Classes principales

Worker

La classe Worker est chargée d'exécuter les calculs de persistances additives et multiplicatives sur des intervalles de nombres pour le projet. Elle est destinée à être exécutée sur les ordinateurs ayant le rôle de workers dans le projet.

Pour optimiser les performances, nous avons choisi de créer un nombre de threads dans le Worker correspondant au nombre de cœurs de la machine sur laquelle le programme est lancé. Ces threads de type ProcCalcul sont créés une seule fois lors de l'initialisation du Worker et mis en attente lorsqu'ils ne sont pas utilisés pour économiser des ressources. Chaque thread traite un nombre différent et effectue les calculs de ses persistances additives et multiplicatives en utilisant des méthodes spécifiques.

La classe Tache a été mise en place pour assurer la synchronisation entre les threads et les ressources. Elle contient et supervise l'utilisation des ressources utiles aux threads : le nombre courant dans l'intervalle à calculer et les hashtables dans lesquels les résultats sont stockés. Lorsque l'objet Tache reçoit un nouvel intervalle, la méthode LancerTache est appelée pour mettre à jour l'intervalle et permettre aux threads de commencer à calculer. La méthode getNumber est utilisée pour récupérer le nombre courant en veillant à ce que chaque thread traite un nombre différent.

Les threads ajoutent les résultats de leurs calculs dans les hashtables correspondantes en utilisant les méthodes ajouteAdd et ajouteMult qui sont synchronisées pour éviter les conflits d'accès. Un thread de type EnvoyerRes est également créé pour gérer l'envoi des résultats au serveur. Tant que les hashtables des persistances multiplicatives et additives ne sont pas complètes, ce thread est mis en attente.

Pour garantir que tous les threads ont terminé leurs calculs avant que les hashtables soient envoyées au serveur, nous avons un tableau de booléen qui contient l'ensemble des statuts des threads de calcul. Ces statuts sont mis à jour par les threads eux-mêmes lorsqu'ils lancent le calcul d'un nombre et terminent d'écrire leur résultat. Une fois que tous les threads ont terminé leurs calculs et que les hashtables sont remplies, le thread EnvoyerRes ouvre un socket vers le port du serveur gérant la réception des résultats et transmet les hachtables.

Enfin, la classe GererSaisieWorker permet à l'utilisateur de déconnecter le Worker du serveur.

Représentation en couches :

Projet	Système d'exploitation
Serveur	Utilisateur
Worker main et GererSaisieWorker	Interface Utilisateur
Intervalle/hashtable	Mémoire/Fichiers (ressources)
Tache	Couche entre mémoire et processeur
ProcCalcul et EnvoyerRes	Processeur (threads)

Client

La classe Client joue le rôle d'interface entre les utilisateurs et le serveur, leur permettant d'envoyer des requêtes et de recevoir des réponses. Cette architecture est conçue pour que le serveur puisse fournir des services à plusieurs clients simultanément. Le programme commence par établir une connexion avec le serveur en utilisant l'adresse IP et le port spécifiés en ligne de commande ou les valeurs par défaut. Les flux de données entrants et sortants sont ensuite initialisés pour permettre la communication avec le serveur.

Ensuite, une instance de la classe GererSaisieClient est créée et lancée pour gérer la saisie de l'utilisateur. La classe GererSaisieClient hérite de la classe Thread et se charge d'afficher un menu principal qui permet à l'utilisateur de choisir une option de menu. Les options de menu incluent des fonctionnalités telles que le calcul de la persistance multiplicative ou additive, la comparaison entre les deux, ou encore l'affichage de l'état du serveur.

Une fois qu'une option de menu est choisie, la classe GererSaisieClient affiche un sous-menu pour permettre à l'utilisateur de choisir une requête spécifique. La méthode run() envoie alors la requête au serveur et lit la réponse du serveur. Si l'utilisateur choisit l'option "END", la méthode envoie une requête "END" au serveur pour terminer la communication et le déconnecter.

La classe GererSaisieClient contient également une méthode MenuPrincipal() pour afficher le menu principal, ainsi qu'une méthode MenuPers() pour afficher le sous-menu de chaque option de menu. Ces méthodes gèrent la saisie de l'utilisateur et renvoient la requête choisie par l'utilisateur au Serveur.

Donc la classe Client et la classe GererSaisieClient sont des composantes importantes d'un système client-serveur. Elles offrent une interface utilisateur efficace et intuitive pour accéder aux services fournis par le serveur. La classe Client gère la connexion au serveur, tandis que la classe GererSaisieClient gère la saisie de l'utilisateur et la communication avec le serveur. En travaillant ensemble, ces deux classes assurent un échange fluide et efficace entre l'utilisateur et le serveur.

Premier menu proposé au client :

Option	Fonctionnalité
mul	Calculer la persistance multiplicative
add	Calculer la persistance additive
comp	Comparaison entre les deux …
stat	Affichage de l'état du serveur

Second menu proposé au client : :

Option	Fonctionnalité
pmax	Calcule la persistance maximale
moy	Calcule la moyenne de persistance
med	Calcule médiane de persistance
pn	Calcule la persistance d'un nombre
pi	Affiche les persistances des nombres sur un intervalle donné (Cette option n'est pas disponible pour comp)
op	Calcule le nombre d'occurrences d'une persistance
nbpmax	Affiche la liste des nombres qui ont la persistance maximale

Troisième menu : Ce menu ne s'affiche pas pour pi ou stat

Option	Fonctionnalité
int	Faire le calcul sur un intervalle
all	Faire le calcul sur toutes les données du serveur

Serveur

La classe Serveur supporte plusieurs connexions entrantes simultanément. Elle écoute les connexions entrantes sur le port 8000 pour les Workers, 9000 pour les clients et 10000 pour les objets reçus depuis le Worker.

Dés qu'un client se connecte au serveur, un thread ConnexionClient est lancé et il en est de même pour le Worker mais quand un Worker se connecte 2 threads sont lancés : ConnexionWorker et EcouterObjets qui vont permettre la bonne communication entre le Serveur et le Worker.

Chaque thread ConnexionClient écoute les requêtes des clients et en fonction de la requête il appelle les méthodes correspondantes.
Chaque thread ConnexionWorker envoi des requêtes aux workers disponibles pour qu'ils calculent les persistances que nous n'avons pas encore calculé.

Le serveur implémente une architecture basée sur les sockets, qui permet une communication bidirectionnelle entre le Serveur, les Workers les Clients.

Les requêtes des clients sont envoyées sous forme de chaînes de caractères,

Les réponses du serveur sont également envoyées sous forme de chaînes de caractères aux Clients.

Le serveur fournit une série de services, chacun étant associé à une méthode spécifique. Les services incluent le calcul de la persistance additive et multiplicative, la comparaison entre ces deux méthodes, ainsi que l'affichage des informations système et de l'utilisation du processeur.

Explication des attributs statics :

maxClients et **maxWorkers** : le nombre maximal de client et workers doit être en static parce qu'il y a plusieurs classes dans le package qui vont utiliser ces variables, on aurait pu les faire passer dans les paramètres des méthodes mais on a jugé que c'était plus claire ainsi.

numClient et **numWorker** : idem pour le nombre de Client et le nombre de Workers

pwClient et **pwWorker**: à chaque connexion du client ou worker on met dans ces tablaux leur PrintWriter pour que quand on ferme le serveur on envoi « END » à tout les clients et workers

ipClient et **ipWorker** : les IP des clients et Workers doit être static car la boucle qui écoute les requêtes des clients est dans le main et main soit être static

nombre : le nombre est mis à jour dans le main et main une méthode static

maxcalcule : idem pour le nombre maximum calculé

WorkersDisponibles : les deux classes EcouterObjets et ConnexionWorker ont besoin d'accéder à la liste des workers disponibles

arreter : la boucle while dans le main vérifie que cette condition n'est pas à true et vu que la méthode main est static nous devons déclarer cette variable en static.

serverSocketEcouteur : on veut créer un seul serverSocket pour écouter les objets sur le port 10000

intervalle : l'intervalle doit être initialisée une seule fois dans le main et toutes les classes travaillent en fonction de cet intervalle donc elle doit être static

tps : c'est l'heure du démarrage du serveur et la classe ConnexionClient a besoin de cette donnée.

ecrireMaxCalculer() : cette méthode doit être accessible par EcouterObjets car à chaque fois qu'un client envoi un objet, les calculs sont bien reçus et bien stockés sur le disque, on appelle cette méthode.

LancerWorkerCalculPersistance() : cette méthode est appelée dans la boucle while du main et donc elle doit être static.

MAJNombre() et **getNombre()** sont des méthodes appelées dans LancerWorkerCalculPersistance() qui es elle-même dans le méthode main donc les 3 méthodes doivent être déclarées en static.

Description de l'algorithme des deux persistances

Pour le calcul des persistances, nous utilisons des BigInteger car ce type permet la manipulation d'entier de taille quasi infinie.

Nous procédons de la façon suivante : nous divisons le nombre par 10 pour obtenir le quotient et le reste. Nous ajoutons/multiplions le reste dans une variable puis nous affectons le résultat de la division au nombre. Nous répétons ces opérations jusqu'à ce que le nombre soit égal à 0.

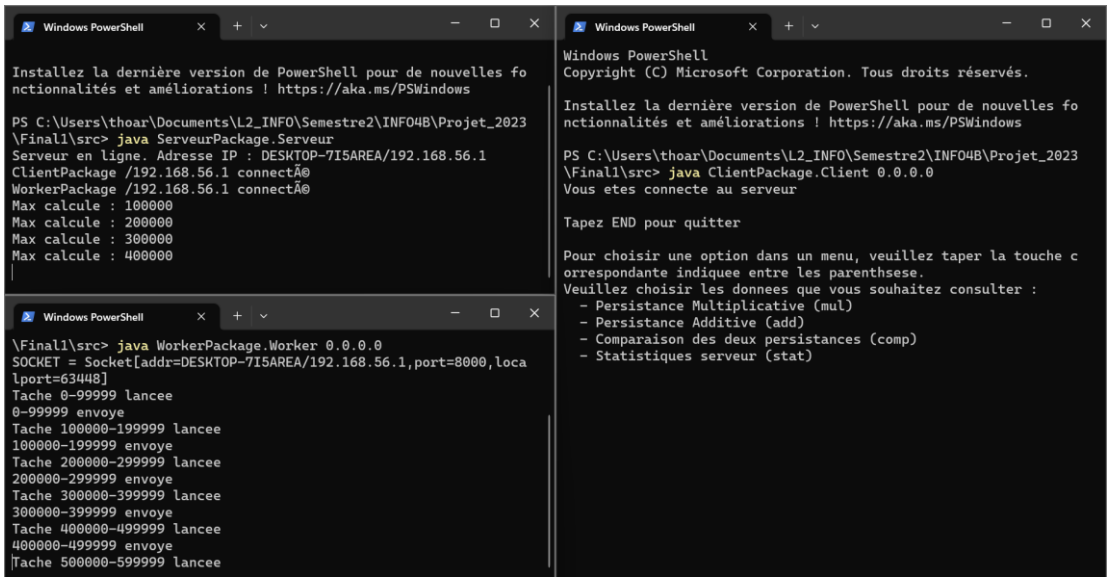
Nous répétons cela tant que le nombre n'est pas égal à un chiffre entre [0,9], en incrémentant un compteur, que nous retournons comme résultat.

Conclusion

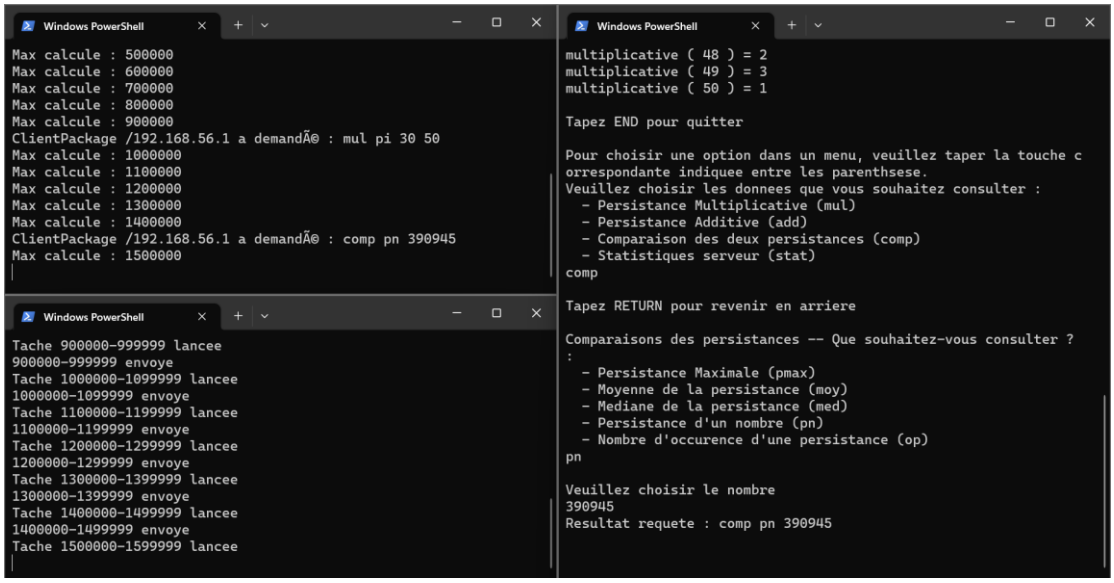
Jeu de test

Au lancement du serveur le Serveur crée 3 dossiers Additive, Multiplicative et Infos et 1 fichier maxCalcule.txt dans Infos si ces fichiers n'existent pas.

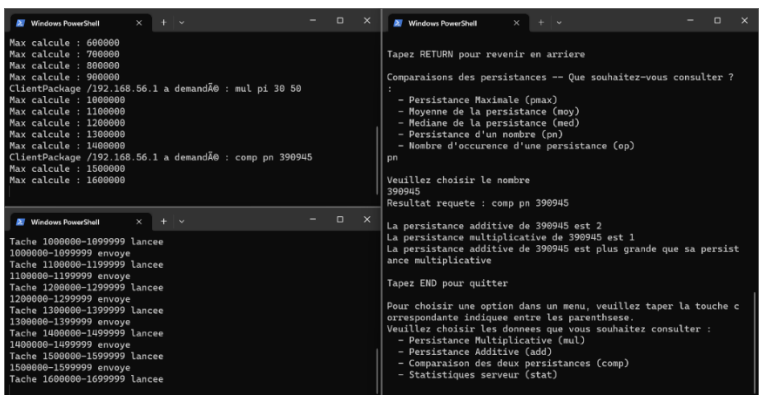
Lancement du Serveur(en haut à gauche) 1 Worker(en bas à gauche) et un Client(à droite)



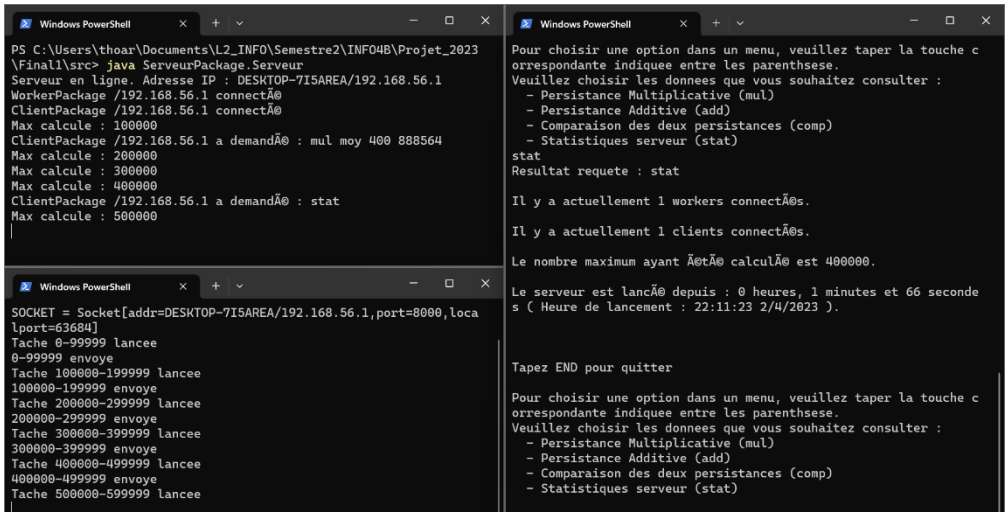
Requête Client : comp pn 390945



Reponse du Serveur à la requête Client :



Demander l’état du Serveur par le Client :



Bilan du projet

Dans l'ensemble, nous avons atteint notre objectif de distribuer efficacement le traitement des tâches de calcul de persistance des nombres à l'aide de plusieurs machines. Le serveur a été en mesure de distribuer les tâches aux workers disponibles et de stocker les résultats sur le disque. Le programme client a fourni une interface ergonomique pour visualiser les résultats et effectuer des requêtes sur les données stockées.

Perspectives d'amélioration

Il serait possible d'ajouter plus de fonctionnalités au programme client, comme la capacité de créer des graphiques pour visualiser les données. De plus, il serait possible d'améliorer l'efficacité du serveur en optimisant la gestion des workers.

FIN