Programmation concurrentielle :

PROJET : SYSTÈME DE CALCUL DISTRIBUÉ POUR ANALYSE DE PERFORMANCE COMMERCIALE

Python / Docker

Sujet : Programmation concurrentielle (Projet)

Auteur du document : Faouzi Tchenar

Date création :	Première Version	Version actuelle
27/05/2025	1.0	1.0

Sommaire :

Table des matières

1. Présentation de l'entreprise : AutoConnect Solutions

Contexte de l'entreprise

AutoConnect Solutions est un concessionnaire automobile premium fondé en 2018, spécialisé dans la vente et la location de véhicules haut de gamme (BMW, Mercedes, Audi, Tesla). L'entreprise a connu une croissance rapide et opère désormais sur trois marchés stratégiques français : **Lyon**, **Paris** et **Marseille**.

Avec un chiffre d'affaires annuel de 45 millions d'euros et une flotte de plus de 2 500 véhicules en location, AutoConnect Solutions gère quotidiennement des centaines de transactions commerciales. L'entreprise propose deux services principaux :

- **Vente de véhicules neufs et d'occasion** (tickets moyens : 35 000€ à 85 000€)
- **Location longue durée et courte durée** (contrats de 1 mois à 48 mois)

Problématique métier

Face à l'expansion géographique et à l'augmentation du volume transactionnel, la direction d'AutoConnect Solutions fait face à des défis d'analyse de performance :

- **Consolidation complexe** des données commerciales multi-sites
- **Temps de traitement** prohibitifs pour les rapports mensuels (actuellement 4-6 heures)
- **Besoins d'analyses en temps réel** pour optimiser les stratégies commerciales régionales
- **Scalabilité** : préparation à l'ouverture de nouveaux sites (Toulouse, Bordeaux prévus en 2026)

2. Objectifs du projet

Mission technique

Développer un **système de calcul distribué** utilisant des conteneurs Docker pour traiter parallèlement les données de ventes et locations, permettant une analyse rapide et scalable des performances commerciales par ville et par période.

Calculs requis

- **Chiffre d'affaires mensuel par ville** (vente + location)
- **Répartition vente/location par ville**
- **Performance comparative inter-villes**

Calculs supplémentaires proposés

- **Top 5 des modèles les plus vendus/loués par ville**
- **Analyse de saisonnalité** (variations mensuelles)
- **Temps moyen de rotation des véhicules** en location
- **Marge bénéficiaire moyenne** par type de transaction et par ville
- **Prédiction de tendance** (régression linéaire simple sur 3 mois)

3. Spécifications techniques

Architecture distribuée

- **Conteneurs Docker** : 3 à 6 conteneurs (au choix des étudiants)
- **Communication inter-processus** : Message Queue (RabbitMQ/Redis), Pipes nommés, ou Shared Memory
- **Modèles de parallélisme** : Pipeline, Map-Reduce, ou Fork-Join (au choix)

Format des données

Fichier CSV : transactions_autoconnect.csv

```
transaction_id,date,ville,type,modele,prix,duree_location_mois
TX001,2024-01-15,Lyon,vente,BMW_X3,45000,NULL
TX002,2024-01-16,Paris,location,Tesla_Model_3,800,12
TX003,2024-01-17,Marseille,vente,Mercedes_C_Class,52000,NULL
```

4. Organisation des tâches (6 heures)

Répartition binôme :

Étudiant 1	**Étudiant 2**	**Travail Commun**
Architecture système (2h)	**Traitement de données** (2h)	**Gestion projet** (2h)
- Design de l'architecture distribuée	- Parsing et validation CSV	- Planification et coordination
- Configuration Docker Compose	- Algorithmes de calcul	- Tests d'intégration
- Gestion communication inter-processus	- Optimisation performances	- Documentation technique
- Orchestration des conteneurs	- Gestion des erreurs	- Démonstration finale

Planning détaillé (6h)

Phase 1 : Conception et architecture (1h30)
- **Commun** : Analyse du cahier des charges, choix architectural
- **Étudiant 1** : Design de l'architecture, choix des outils de communication
- **Étudiant 2** : Analyse du format de données, conception des algorithmes

Phase 2 : Développement (3h)

- **Étudiant 1** : Développement de l'infrastructure Docker, configuration réseau
- **Étudiant 2** : Implémentation des algorithmes de calcul, parsing CSV
- **Commun** : Intégration continue, résolution des conflits

Phase 3 : Tests et optimisation (1h)
- **Commun** : Tests de performance, validation des résultats
- **Étudiant 1** : Monitoring des conteneurs, ajustements réseau
- **Étudiant 2** : Optimisation des calculs, gestion des cas limites

Phase 4 : Documentation et démonstration (30min)
- **Commun** : Rédaction documentation, préparation démo

5. Livrables attendus

Code et infrastructure

- Code source complet (Python)
- Fichiers Docker et docker-compose.yml
- Scripts de test et jeux de données

Documentation technique

- Diagramme d'architecture distribuée
- Guide d'installation et d'utilisation
- Analyse des performances (temps d'exécution, scalabilité)

Rapport d'analyse

- Comparaison des modèles de parallélisme testés
- Justification des choix techniques
- Perspectives d'amélioration

6. Critères d'évaluation

- **Fonctionnalité** (40%) : Calculs corrects, gestion des erreurs
- **Architecture** (30%) : Design distribué, scalabilité
- **Performance** (20%) : Optimisation, temps d'exécution
- **Documentation** (10%) : Clarté, complétude

7. Références bibliographiques

1. Documentation officielle Python – multiprocessing (Pipe, Queue, Manager)

- multiprocessing — Process-based parallelism — Python 3.13.3 documentation (en anglais)
  Explications et exemples sur les communications entre processus avec Queue, Pipe, Manager, ainsi que sur la synchronisation et le partage d'état.
- multiprocessing — Parallélisme par processus — Documentation Python (français)
  Version française de la documentation officielle, avec exemples d'utilisation de queues, pipes et managers.

2. Tutoriels et explications sur Pipe, Queue, Manager

- Using Multiprocessing Queues and Pipes – Datanovia.com (en anglais)
  Tutoriel clair sur l'utilisation de Queue, Pipe et objets partagés, avec exemples pratiques et conseils pour éviter les pièges courants.
- Pipes, queues, and lock in multiprocessing in Python – Educative.io (en anglais)
  Présentation synthétique des différences entre Pipe et Queue, exemples de code, et explication de la synchronisation avec Lock.
- python - Multiprocessing - Pipe vs Queue - Stack Overflow
  Discussion détaillée sur les différences d'usage, de performance et de cas d'utilisation entre Pipe et Queue dans le module multiprocessing.
- Communication Between Processes - Python Module of the Week (en anglais)
  Exemples pratiques sur la communication entre processus avec Queue et Pipe, explications sur la sérialisation et le passage de messages.
- Reading and Hacking Python's multiprocessing.managers: Part 1 (en anglais)
  Article d'introduction sur l'utilisation avancée de multiprocessing.Manager pour le partage d'objets complexes entre processus.

3. Modèles de parallélisme (pipeline, map-reduce, fork-join) en Python

- Parallel Processing in Python – A Practical Guide with Examples (en anglais)
  Présente les différents modèles de parallélisme en Python, y compris pipeline, map-reduce, et l'utilisation de multiprocessing.
- MapReduce Programming Model (Wikipedia, en anglais)
  Présentation du modèle Map-Reduce, principes, étapes et exemples.