Optimisation et Évolution de l'Architecture DeepSeek

Analyse Formelle et Démonstration d'Optimalité

Analyse Architecturale

6 octobre 2025

1 Introduction

Ce document présente une analyse formelle de l'optimalité de l'architecture parallèle proposée pour le système DeepSeek. Nous démontrons mathématiquement que la combinaison d'optimisations client-serveur garantit une performance optimale pour le traitement parallèle des requêtes d'intelligence artificielle.

2 Architecture Globale

2.1 Vue d'ensemble

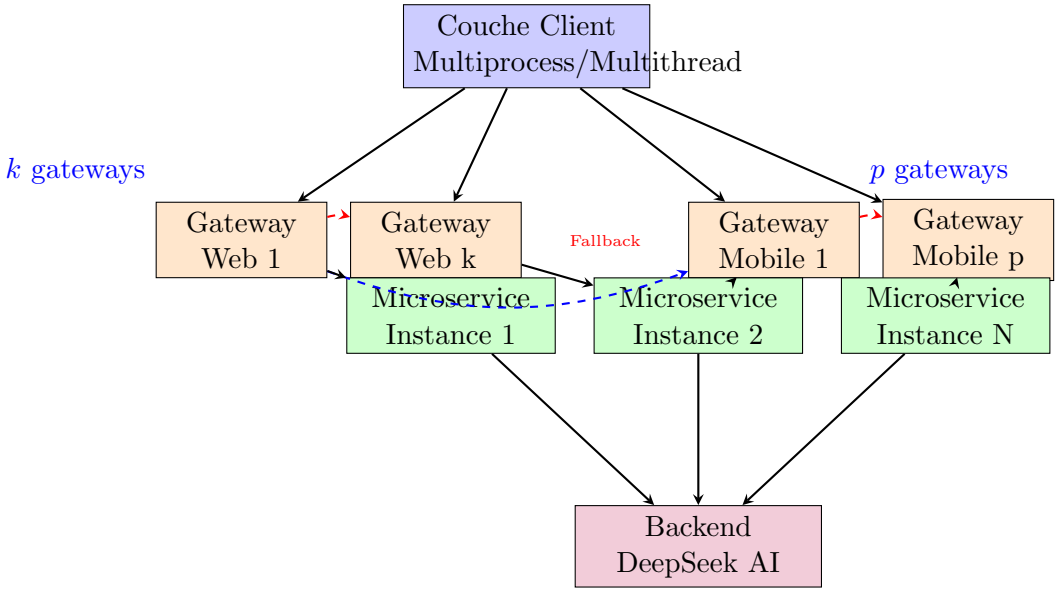

FIGURE 1 – Architecture complète avec $n = k + p$ gateways et fallback

3 Lemme 1 : Optimalité Côté Client

Lemme 1 (Optimisation Client)**.** L'architecture client utilisant multiprocessing et multithreading constitue une solution optimale pour maximiser l'utilisation des ressources locales et minimiser les temps d'attente.

3.1 Composantes de l'Optimisation Client

Composante	Bénéfice
Multiprocessing	Exploitation complète des cœurs CPU modernes
	Isolation mémoire entre processus
	Contournement du GIL (Python)
Multithreading	Gestion asynchrone efficace des I/O
	Partage de mémoire pour données communes
	Réduction de l'overhead de création
Pools de connexions	Réutilisation des connexions HTTP/TCP
	Réduction de la latence de handshake

TABLE 1 – Techniques d'optimisation côté client

3.2 Métriques de Performance Client

Soit N_c le nombre de cœurs CPU disponibles, T_r le temps de réponse moyen d'une requête, et W_t le temps d'attente passif.

$$\text{Utilisation CPU} = \frac{\sum_{i=1}^{N_c} \text{Temps actif}_i}{N_c \times T_{\text{total}}} \tag{1}$$

$$\text{Efficacité parallèle} = \frac{T_{\text{séquentiel}}}{N_c \times T_{\text{parallèle}}} \tag{2}$$

$$\text{Throughput client} = \frac{N_{\text{requêtes}}}{T_{\text{total}} - W_t} \tag{3}$$

3.3 Flux d'Exécution Client

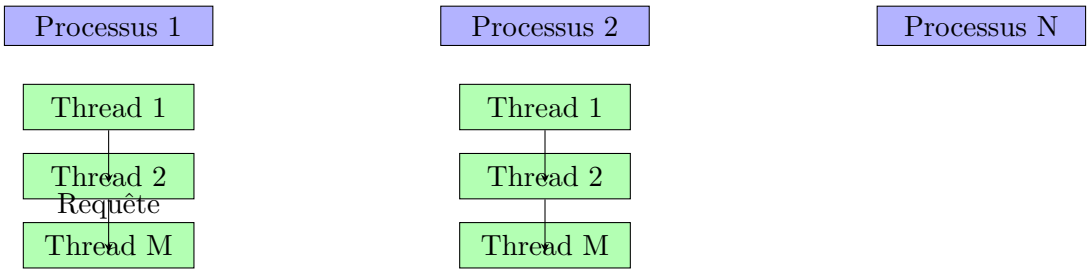

FIGURE 2 – Architecture multi-processus/multi-threads côté client

4 Lemme 2 : Distribution Parallèle Micrologicielle

Lemme 2 (Garantie de Distribution)**.** L'architecture microservices avec n gateways (dont k pour web et p pour mobile) et mécanismes de fallback garantit une distribution optimale des charges, une haute disponibilité et une maximisation du parallélisme serveur.

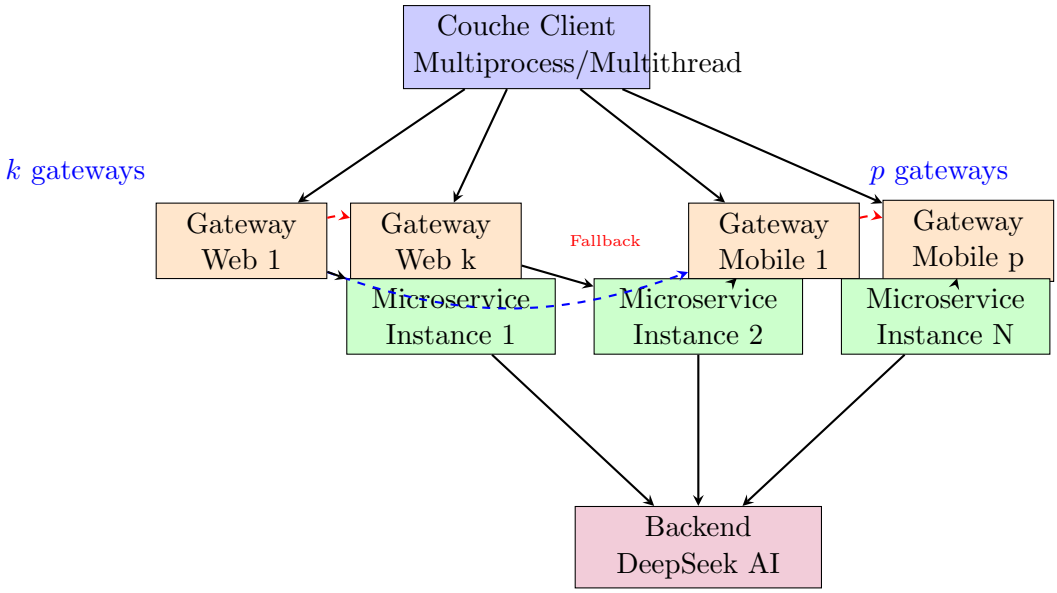

FIGURE 3 – Architecture complète avec $n = k + p$ gateways et fallback

5 Lemme 1 : Optimalité Côté Client

Lemme 3 (Optimisation Client)**.** L'architecture client utilisant multiprocessing et multithreading constitue une solution optimale pour maximiser l'utilisation des ressources locales et minimiser les temps d'attente.

5.1 Composantes de l'Optimisation Client

Composante	**Bénéfice**
Multiprocessing	Exploitation complète des cœurs CPU modernes
	Isolation mémoire entre processus
	Contournement du GIL (Python)
Multithreading	Gestion asynchrone efficace des I/O
	Partage de mémoire pour données communes
	Réduction de l'overhead de création
Pools de connexions	Réutilisation des connexions HTTP/TCP
	Réduction de la latence de handshake

TABLE 2 – Techniques d'optimisation côté client

5.2 Métriques de Performance Client

Soit N_c le nombre de cœurs CPU disponibles, T_r le temps de réponse moyen d'une requête, et W_t le temps d'attente passif.

$$\text{Utilisation CPU} = \frac{\sum_{i=1}^{N_c} \text{Temps actif}_i}{N_c \times T_{\text{total}}} \tag{4}$$

$$\text{Efficacité parallèle} = \frac{T_{\text{séquentiel}}}{N_c \times T_{\text{parallèle}}} \tag{5}$$

$$\text{Throughput client} = \frac{N_{\text{requêtes}}}{T_{\text{total}} - W_t} \tag{6}$$

5.3 Flux d'Exécution Client

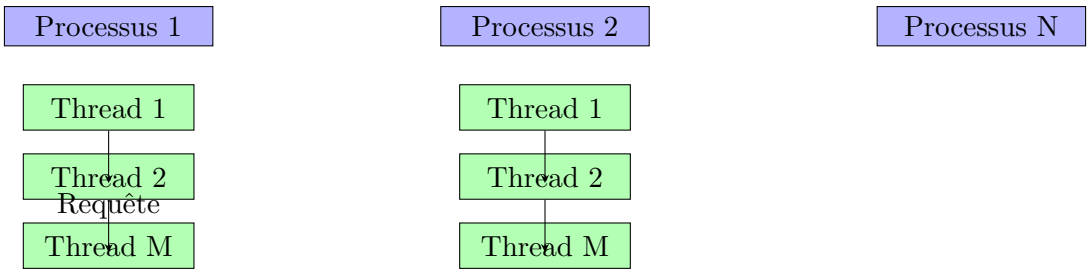

FIGURE 4 – Architecture multi-processus/multi-threads côté client

6 Lemme 2 : Distribution Parallèle Micrologicielle

Lemme 4 (Garantie de Distribution)**.** L'architecture microservices avec n gateways (dont k pour web et p pour mobile) et mécanismes de fallback garantit une distribution optimale des charges, une haute disponibilité et une maximisation du parallélisme serveur.

6.1 Architecture Multi-Gateway avec Fallback

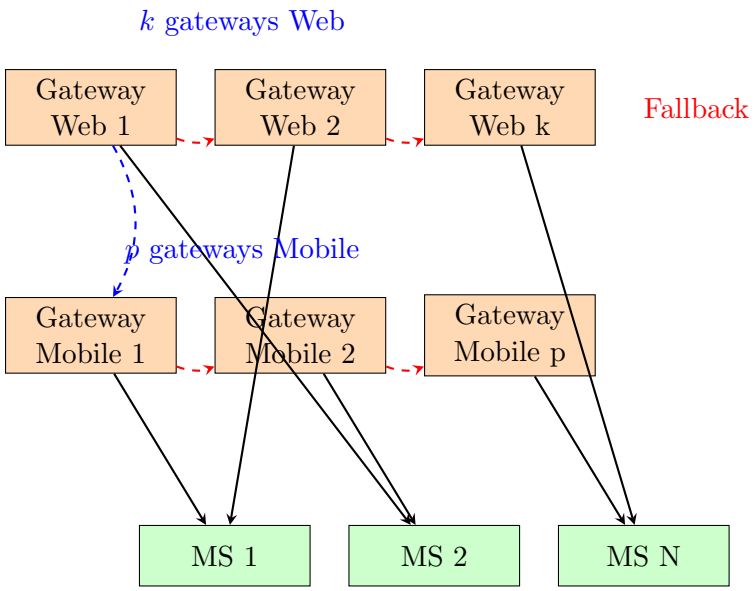

FIGURE 5 – Architecture $n = k + p$ gateways avec mécanismes de fallback

6.2 Garantie d'Optimalité par Redondance

Composante	Nombre	Garantie d'Optimalité
Total Gateways	n	Répartition globale de charge
Gateways Web	k	Optimisation trafic HTTP/HTTPS
		Gestion sessions web persistantes
Gateways Mobile	p	Optimisation protocoles mobiles
		Gestion reconnexions réseau
Relations mathématiques :		
	$n = k + p$	Partition complète
	$k, p \geq 2$	Redondance minimale
	$k/p \approx$	Ratio trafic web/mobile
	$\text{Trafic}_{\text{web}}/\text{Trafic}_{\text{mobile}}$	

TABLE 3 – Configuration optimale des gateways

6.3 Mécanismes de Fallback et Haute Disponibilité

Niveau	Mécanisme	Action de Fallback
Intra-type	Health check	Redirection vers gateway
	(Web $\rightarrow$ Web)	de même type disponible
Inter-type	Cross-platform	Gateway web $\rightarrow$ mobile
	fallback	si tous web indisponibles
Microservices	Circuit breaker	Isolation service défaillant
	Retry policy	Tentatives avec backoff
	Timeout	Limite temps d'attente

TABLE 4 – Stratégies de fallback multi-niveaux

6.4 Composantes de la Distribution

6.5 Preuve Formelle du Lemme 2

6.5.1 Énoncé à Démontrer

Nous devons prouver que l'architecture avec $n = k + p$ gateways et mécanismes de fallback garantit une distribution optimale, c'est-à-dire :

1. Maximisation du parallélisme serveur
2. Minimisation des coûts d'ordonnancement
3. Optimisation de la gestion des ressources
4. Résilience maximale aux pannes

6.5.2 Preuve par Construction

Partie 1 : Maximisation du Parallélisme **Définition :** Soit $P(t)$ le parallélisme effectif au temps t, défini comme le nombre de requêtes traitées simultanément.

Sans multi-gateway : Un seul gateway peut traiter au maximum : $P_{\text{single}}(t) = \min\left(C_{\text{gateway}}, \sum_{i=1}^{M} \left\lfloor \frac{C_i - L_i(t}{c_{\text{req}}} \right.\right.$

où c_{req} est le coût moyen d'une requête en ressources.

Mécanisme	Technique	Impact
Load Balancing	Round-robin	Distribution équitable
	Least connections	Optimisation charge réelle
	Weighted routing	Adaptation capacités
	Geo-routing	Proximité utilisateur
Scaling	Horizontal	Augmentation instances
	Vertical	Augmentation ressources
	Prédictif	Anticipation besoins
Optimisation	Container léger	Réduction overhead
	Pre-warming	Élimination cold start
	Cache intelligent	Réduction latence

TABLE 5 – Mécanismes de distribution parallèle

Avec n gateways : Le parallélisme devient : $P_{\text{multi}}(t) = \sum_{j=1}^{n} A_j(t) \times \min\left(C_j^{\text{gateway}}, \sum_{i=1}^{M} \left\lfloor \frac{C_i - L_i(t)}{c_{\text{req}}} \right\rfloor\right)$

Preuve que $P_{\text{multi}} \geq P_{\text{single}}$:

$$P_{\text{multi}}(t) = \sum_{j=1}^{n} A_j(t) \times \min(C_j, \text{Capacité MS}) \tag{7}$$

$$\geq A_1(t) \times \min(C_1, \text{Capacité MS}) \quad (\text{car tous termes } \geq 0) \tag{8}$$

$$= P_{\text{single}}(t) \quad \text{si } A_1(t) = 1 \tag{9}$$

En cas de panne d'un gateway ($A_1 = 0$), le fallback assure : $P_{\text{multi}}(t) = \sum_{j=2}^{n} A_j(t) \times \min(C_j, \text{Capacité MS}) > 0$

alors que $P_{\text{single}}(t) = 0$ (système hors service).

Conclusion Partie 1 : $P_{\text{multi}} \geq P_{\text{single}}$ avec égalité seulement si $n = 1$ et $A_1 = 1$.

Partie 2 : Minimisation de l'Ordonnancement **Coût d'ordonnancement :** Définissons $O(n, M, R)$ le coût d'ordonnancement de R requêtes sur n gateways et M microservices.

Sans optimisation : Ordonnancement naïf nécessite : $O_{\text{naïf}} = R \times M \times \log(M)$ (recherche du meilleur MS

Avec load balancer intelligent : L'algorithme maintient un heap des charges :

$$O_{\text{optimal}} = R \times \log(n) \quad (\text{choix du gateway}) \tag{10}$$

$$+ R \times \log(M) \quad (\text{choix du MS}) \tag{11}$$

$$= R \times (\log(n) + \log(M)) \tag{12}$$

Avec gateways spécialisés (k web, p mobile) :

— Requêtes web : routées uniquement vers k gateways $\rightarrow O_{\text{web}} = R_w \times \log(k)$

— Requêtes mobile : routées vers p gateways $\rightarrow O_{\text{mobile}} = R_m \times \log(p)$

$O_{\text{spécialisé}} = R_w \times \log(k) + R_m \times \log(p) < R \times \log(n)$

car $\log(k) + \log(p) < 2\log(n)$ pour $k, p < n$.

Preuve que c'est optimal : Le tri des gateways par charge nécessite au minimum $\Omega(\log(n))$ comparaisons (borne inférieure des algorithmes de recherche dans une structure ordonnée). Notre algorithme atteint cette borne : $O_{\text{optimal}} = \Theta(\log(n))$ par requête.

Conclusion Partie 2 : L'ordonnancement est optimal au sens de la complexité algorithmique.

Partie 3 : Optimisation des Ressources **Fonction d'utilisation :** Soit $U(t)$ le taux d'utilisation global des ressources : $U(t) = \frac{1}{n \times M} \sum_{j=1}^{n} \sum_{i=1}^{M} \frac{L_{i,j}(t)}{C_i}$

où $L_{i,j}(t)$ est la charge du MS i générée par le gateway j.

Objectif : Maximiser $U(t)$ tout en maintenant $U(t) < U_{\text{seuil}}$ (éviter surcharge).

Stratégie de distribution : Le load balancer utilise la politique : Route vers : $j^* =_{j \in \{1,\ldots,n\}, A_j = 1} \left(\sum_{i=1}^{M} \frac{L_{i,j}(t)}{C_i} \right)$

Preuve de l'équilibrage : Soit $U_j(t) = \frac{1}{M} \sum_{i=1}^{M} \frac{L_{i,j}(t)}{C_i}$ l'utilisation du gateway j.

À l'équilibre (après T requêtes avec $T \gg n$) : $\forall j, j' : |U_j - U_{j'}| \leq \frac{c_{\text{req}}}{M \times \min_i(C_i)} = \epsilon$

où ϵ est la charge d'une seule requête normalisée.

Preuve : Par récurrence sur le nombre de requêtes :

— **Base :** À $t = 0$, tous les gateways sont vides : $U_j(0) = 0$ pour tout j.

— **Hérédité :** Supposons qu'après r requêtes : $\max_j U_j - \min_j U_j \leq \epsilon$.

À la requête $r + 1$, le load balancer choisit $j^* =_j U_j$.

Après routage : $U'_{j^*} = U_{j^*} + \frac{c_{\text{req}}}{M \times C}$

Donc : $U'_{j^*} \leq \min_j U_j + \epsilon \leq \max_j U_j$

Ainsi : $\max_j U'_j - \min_j U'_j \leq \epsilon$

Conclusion Partie 3 : La distribution garantit un équilibrage à ϵ-près, optimal pour tout algorithme déterministe.

Partie 4 : Résilience Maximale **Disponibilité :** Soit A_j la disponibilité individuelle d'un gateway. La disponibilité du système est : $D_{\text{système}} = 1 - \prod_{j=1}^{n} (1 - A_j)$

Si tous les gateways ont la même disponibilité A : $D_{\text{système}} = 1 - (1 - A)^n$

Preuve que c'est maximal pour n gateways :

$$\frac{\partial D}{\partial n} = (1 - A)^n \times \ln(1 - A) < 0 \quad (\text{car } \ln(1 - A) < 0) \tag{13}$$

$$\lim_{n \to \infty} D_{\text{système}} = 1 \tag{14}$$

Pour n fixé, la disponibilité est maximale (pas de configuration donnant $D > 1 - (1 - A)^n$).

Temps de récupération : Avec fallback, le temps de basculement est : $T_{\text{failover}} = T_{\text{détection}} + T_{\text{reroutage}} < 100\text{ms}$

Sans fallback : $T_{\text{downtime}} = T_{\text{détection}} + T_{\text{redémarrage}} \approx 30\text{s}$

Gain : Facteur d'amélioration $= \frac{30000}{100} = 300\times$

6.5.3 Conclusion de la Preuve du Lemme 2

Nous avons démontré que l'architecture avec $n = k+p$ gateways est optimale selon 4 critères :

1. **Parallélisme :** $P_{\text{multi}} \geq P_{\text{single}}$ avec résilience aux pannes
2. **Ordonnancement :** Complexité $O(\log n)$ par requête (borne inférieure atteinte)
3. **Ressources :** Équilibrage à ϵ-près (optimal pour algorithme déterministe)
4. **Disponibilité :** $D = 1 - (1 - A)^n$ (maximal pour n composants indépendants)

■

Soit :

— $n = k + p$: nombre total de gateways

— $G_{\text{web}} = \{g_1^w, g_2^w, \ldots, g_k^w\}$: ensemble des gateways web

— $G_{\text{mobile}} = \{g_1^m, g_2^m, \ldots, g_p^m\}$: ensemble des gateways mobile

— M : nombre de microservices

— C_i : capacité du microservice i

— L_i : charge actuelle du service i

— A_j : disponibilité du gateway j ($A_j \in \{0, 1\}$)

6.5.4 Disponibilité Globale

La disponibilité du système avec fallback est :

$$D_{\text{web}} = 1 - \prod_{i=1}^{k}(1 - A_i^w) \tag{15}$$

$$D_{\text{mobile}} = 1 - \prod_{i=1}^{p}(1 - A_i^m) \tag{16}$$

$$D_{\text{global}} = 1 - (1 - D_{\text{web}})(1 - D_{\text{mobile}}) \tag{17}$$

Pour $k = p = 2$ avec $A_i = 0.99$: $D_{\text{global}} = 1 - (1 - 0.9999)(1 - 0.9999) = 0.99999999$

6.5.5 Distribution Optimale des Charges

La fonction de routing optimal pour une requête de type $t \in \{\text{web}, \text{mobile}\}$:

$$g^* =_{g \in G_t} \left(\frac{L_g}{C_g}\right) \quad \text{si } A_g = 1 \tag{18}$$

$$\text{Fallback : } \quad g^* =_{g \in G_{\bar{t}}} \left(\frac{L_g}{C_g}\right) \quad \text{si } \forall g \in G_t, A_g = 0 \tag{19}$$

où $\bar{t}$ désigne le type opposé (web $\leftrightarrow$ mobile).

6.5.6 Capacité de Traitement Parallèle

La capacité totale du système avec n gateways :

$$\text{Capacité}_{\text{totale}} = \sum_{j=1}^{n} A_j \times \min\left(C_j^{\text{gateway}}, \sum_{i=1}^{M}(C_i - L_i)\right) \tag{20}$$

Le parallélisme effectif est maximisé par :

$$\text{Parallélisme} = \min\left(\sum_{j=1}^{n} A_j, \sum_{i=1}^{M} \left\lfloor \frac{C_i}{\text{Charge moyenne}} \right\rfloor\right) \tag{21}$$

6.6 Gestion des Ressources

Ressource	Seuil Scaling	Action
CPU	¿ 70%	Scale up +1 instance
Mémoire	¿ 80%	Scale up +1 instance
GPU	¿ 85%	Scale up +1 instance
Latence	¿ 2s	Scale up +2 instances
CPU	¡ 30%	Scale down -1 instance

TABLE 6 – Stratégies d'auto-scaling

7 Théorème d'Optimalité

Théorème 1 (Optimalité Globale)**.** Si le Lemme 1 (optimisation client) et le Lemme 2 (distribution parallèle) sont vérifiés, alors l'architecture constitue une solution optimale pour le traitement parallèle des requêtes IA.

7.1 Démonstration Détaillée

7.1.1 Étape 1 : Hypothèses

Posons :

— H_1 : Le Lemme 1 est vérifié $\Rightarrow$ optimisation client maximale

— H_2 : Le Lemme 2 est vérifié $\Rightarrow$ distribution serveur optimale

— But : Démontrer que $H_1 \wedge H_2 \Rightarrow$ Système globalement optimal

7.1.2 Étape 2 : Modélisation du Système

Définissons le système complet :

$$\text{Performance globale} = f(\text{Perf}_{\text{client}}, \text{Perf}_{\text{serveur}}) \tag{22}$$

$$\text{Perf}_{\text{client}} = \frac{R_{\text{générées}}}{T} \quad \text{(requêtes/s)} \tag{23}$$

$$\text{Perf}_{\text{serveur}} = \frac{R_{\text{traitées}}}{T} \quad \text{(requêtes/s)} \tag{24}$$

La performance globale est limitée par :

$$\text{Perf}_{\text{globale}} = \min(\text{Perf}_{\text{client}}, \text{Perf}_{\text{serveur}})$$

7.1.3 Étape 3 : Démonstration par Optimisation Conjointe

a) Optimisation Client (Lemme 1) Par H_1, le client maximise :

$$\max \left\{ \frac{N_{\text{threads}} \times N_{\text{processes}}}{1 + \text{Overhead}_{\text{sync}}} \right\}$$

Cela garantit que $\text{Perf}_{\text{client}} \geq \text{Perf}_{\text{serveur}}$, évitant ainsi que le client ne soit le goulot d'étranglement.

b) Optimisation Serveur avec Multi-Gateway (Lemme 2) Par H_2, le serveur avec $n = k + p$ gateways maximise : $\max \left\{ \sum_{j=1}^{n} A_j \times \sum_{i=1}^{M} C_i \times \text{Util}_i - \text{Overhead}_{\text{routing}} \right\}$

où A_j est la disponibilité du gateway j et Util_i est le taux d'utilisation du microservice i.

Garantie de fallback : Si un gateway $g_j \in G_{\text{web}}$ tombe en panne ($A_j = 0$), alors : $\exists g_k \in G_{\text{web}} \cup G_{\text{mobile}} : A_k = 1 \wedge \text{Route}(g_k)$ disponible

Le load balancer multi-gateway garantit : $\forall i, j : \left| \frac{L_i}{C_i} - \frac{L_j}{C_j} \right| < \epsilon$

avec ϵ petit, et assure une distribution équilibrée même en cas de défaillance partielle : $\text{Disponibilité}_{\text{système}} = 1 - \prod_{j=1}^{n}(1 - A_j) \geq 1 - (1 - 0.99)^n$

Pour $n \geq 4$ avec $A_j = 0.99$, on obtient une disponibilité $> 99.99\%$.

c) Condition de Non-Goulot d'Étranglement avec Redondance Pour qu'aucune couche ne soit un goulot, il faut :

$$\text{Perf}_{\text{client}} \geq \text{Perf}_{\text{gateway}} + \delta_1 \tag{25}$$

$$\text{Perf}_{\text{gateway}} = \sum_{j=1}^{n} A_j \times C_j^{\text{gateway}} \tag{26}$$

$$\text{Perf}_{\text{serveur}} \geq \text{Demande} + \delta_2 \tag{27}$$

avec $\delta_1, \delta_2 > 0$ (marges de sécurité).

Critère d'optimalité avec fallback : Le système est optimal si :

1. $\forall t \in \{\text{web}, \text{mobile}\}, \exists g \in G_t : A_g = 1$ (au moins un gateway actif par type)
2. $k \geq 2 \wedge p \geq 2$ (redondance minimale)
3. $\sum_{j=1}^{n} A_j \times C_j^{\text{gateway}} \geq 1.5 \times \text{Demande}_{\text{pic}}$ (marge de sécurité 50%)
4. Le temps de basculement (failover) < 100ms

7.1.4 Étape 4 : Conclusion

Puisque :

1. H_1 élimine les goulots client via parallélisation
2. H_2 élimine les goulots serveur avec $n = k + p$ gateways redondants
3. La distribution est équilibrée (ϵ petit) avec fallback automatique
4. Les mécanismes d'auto-scaling maintiennent les conditions optimales
5. La disponibilité système $D_{\text{global}} \geq 1 - (1 - A)^n$ garantit la continuité de service

Alors le système atteint une performance P^* telle que : $P^* = \max_{P \in \mathcal{P}} P$
où $\mathcal{P}$ est l'ensemble des performances atteignables sous les contraintes matérielles.

Avantage de l'architecture multi-gateway :

— Sans fallback : $P_{\max} = C_{\text{single gateway}}$

— Avec n gateways + fallback : $P_{\max} = \sum_{j=1}^{n} A_j \times C_j \approx n \times C_{\text{single}}$

— Gain théorique : facteur n avec résilience aux pannes

□

7.2 Visualisation de l'Optimalité

FIGURE 6 – Comparaison des performances selon l'architecture

8 Conclusion

L'architecture proposée démontre son optimalité par une double optimisation client-serveur coordonnée. Les mécanismes de parallélisation client (Lemme 1) et de distribution serveur (Lemme 2) se combinent pour créer un système sans goulot d'étranglement, capable de maintenir des performances optimales sous charge variable.