Nicotte Loann / Bayle Sei

Rendu partie 4

Quels sont les mécanismes proposés par les RPC pour essayer de relancer une requête du client sans réponse ?

Les mécanismes proposés par les RPC pour relancer une requête du client sans réponse sont les timeouts et les retransmissions automatiques. Lorsqu'un client envoie une requête à un serveur, il peut définir un délai d'attente (timeout). Si le client ne reçoit pas de réponse dans ce délai, il peut automatiquement renvoyer la requête. Ce processus peut être répété un certain nombre de fois avant que le client n'abandonne la tentative de communication. Ces mécanismes aident à gérer les situations où les messages peuvent être perdus ou retardés dans le réseau.

Quels sont les mécanismes dont nous disposons pour détecter une panne d’une application serveur ?

Pour détecter une panne d'une application serveur, plusieurs mécanismes peuvent être utilisés :

- Timeouts : Si le client ne reçoit pas de réponse du serveur dans un délai spécifié, il peut considérer que le serveur est en panne.
- Heartbeat messages : Le serveur peut envoyer périodiquement des messages de "heartbeat" au client pour indiquer qu'il est toujours en vie. Si le client ne reçoit pas ces messages, il peut conclure que le serveur est en panne.
- Monitoring tools : Des outils de surveillance peuvent être utilisés pour surveiller l'état du serveur et alerter les administrateurs en cas de panne.

Analyse des Tests

Fonctionnement Normal

Test	Commande	Temps	Résultat
INIT	`./calc_client localhost init 100`	0.24 ms	Mémoire = 100
ADD	`./calc_client localhost add 10 20`	0.17 ms	Mémoire = 30
MUL	`./calc_client localhost mul 5 mem`	0.18 ms	Mémoire = 150

Performance : < 1 ms → Excellent pour du local

Serveur Arrêté

```
[WARNING] Timeout/erreur réseau détectée
[RETRY] Tentative 2/3 pour ADD...
[RETRY] Tentative 3/3 pour ADD...
```

```
[ÉCHEC] Opération ADD échouée après 3 tentatives
Code RPC: 4 (RPC_CANTRECV)
```

Durée totale : 19 secondes (3 timeouts × 5s + 2 délais × 2s)

Mécanismes Implémentés

Mécanisme	Description
Timeout	5 secondes par tentative
Retry automatique	3 tentatives avec 2s d'attente
Détection d'erreur	Retry uniquement si erreur réseau
Mesure perf	gettimeofday() précision ms

Comparaison

Critère	Serveur OK	Serveur KO
Temps	0.17-0.24 ms	19 secondes
Tentatives	1	3
Résultat	Succès	Échec contrôlé

Améliorations Possibles

1. **Timeout adaptatif** : Réduire après 1er échec (5s → 2s)
2. **Circuit breaker** : Arrêter après 10 échecs consécutifs
3. **Health check** : Vérifier disponibilité avant appel
4. **Export métriques** : Log CSV pour monitoring

Conclusion

Note : 8/10

Points forts :

- Détection rapide (5s)
- Retry intelligent
- Performance excellente (<1ms)
- Messages clairs
- Robuste (pas de plantage)

Limites :

- 19s c'est long

- Pas de circuit breaker
- Pas de health check proactif

Quelles sont les différences entre gRPC (de Google) et les RPC (initiaux de Sun)

ONC RPC repose sur l’ancienne sérialisation XDR, tandis que gRPC utilise Protocol Buffers (Protobuf), beaucoup plus compact et efficace. gRPC fonctionne au-dessus de HTTP/2, ce qui lui permet de gérer du streaming bidirectionnel et du multiplexage, contrairement à ONC RPC limité à UDP/TCP et aux communications simples requête/réponse. De plus, gRPC est multiplateforme et multilingue (C++, Python, Java, Go, etc.), alors que ONC RPC s'utilise principalement en C. En termes de fiabilité, gRPC intègre des mécanismes modernes comme les deadlines, timeouts configurables, politiques de retry et le load balancing, tandis que ONC RPC se contente de timeouts simples. Enfin, gRPC est conçu pour les microservices et le cloud, avec un support intégré pour TLS, Auth et Kubernetes, contrairement à ONC RPC qui reste une technologie réseau plus ancienne et bas niveau.