Modèle client / serveur

M1 RÉSEAUX ET TELECOMS - RT0704

OLIVIER FLAUZAC

Introduction

Définitions

Serveur

- programme offrant un service, ou un ensemble de services sur un réseau

Client

- programme émettant des demandes de services (des requêtes)

Application client / serveur

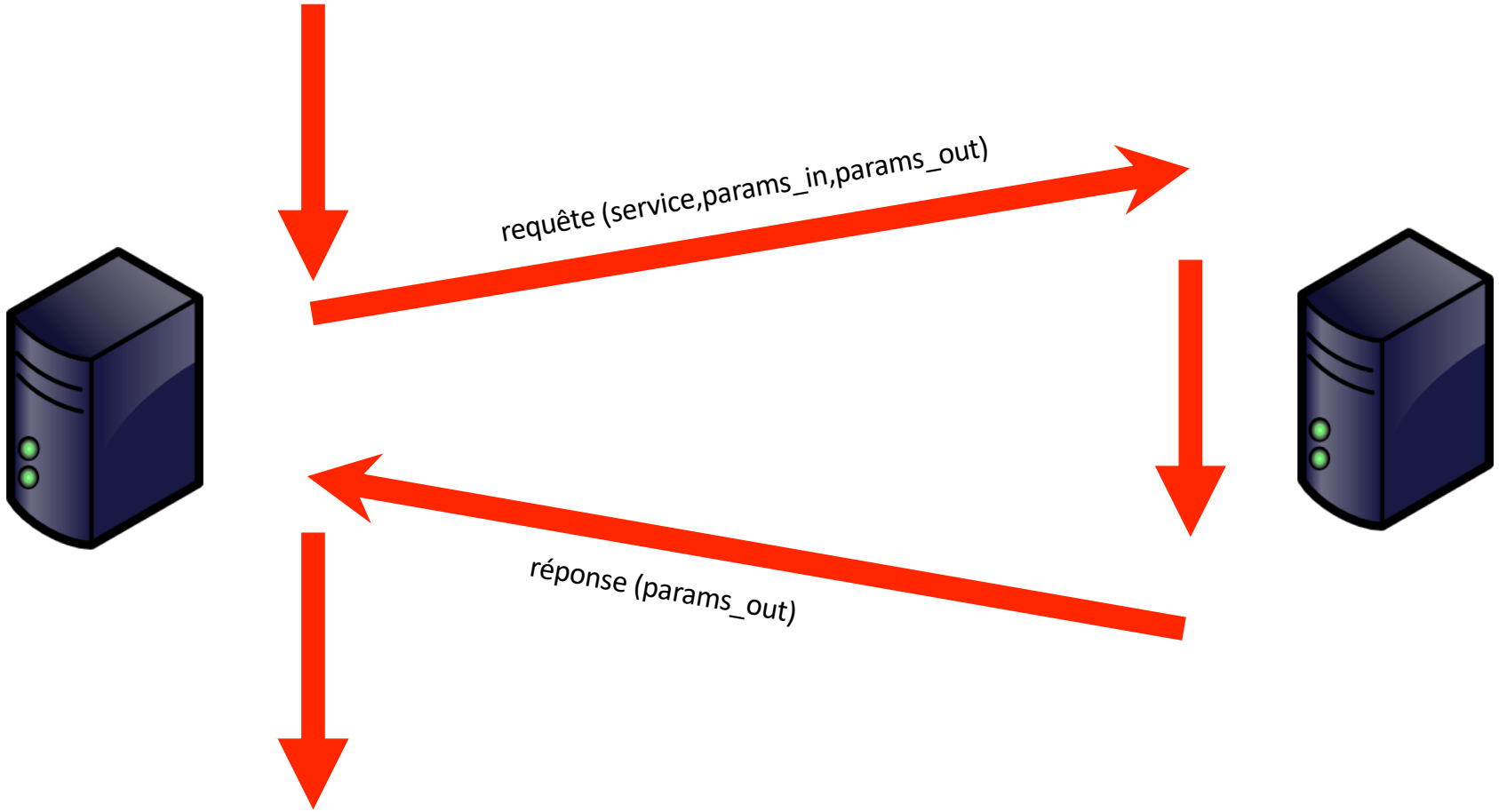
- application exploitant des services distants au travers d'échanges de messages (requêtes) plutôt que par un partage de données (mémoire ou fichier)

Principe

Echange de messages

- deux messages
- la requête
  - nom de la procédure appelée (service)
  - Paramètres
- la réponse
  - Résultat
  - erreur

Principe (suite)



Côté client

Objectifs

- masquage si possible de l'appel distant
- transparence du code

Problématique

- passage de paramètres
- modèle de synchronisation

Côté serveur

Les requêtes

- Réception des requêtes
- Gestion des requêtes
  - stratégie de traitement
  - Priorités

Exécution du service demandé

- Séquentiel
- Concurrent

Gestion de l'état du client

Exemples

Serveur WEB

Serveur d’impression

Serveur de calcul

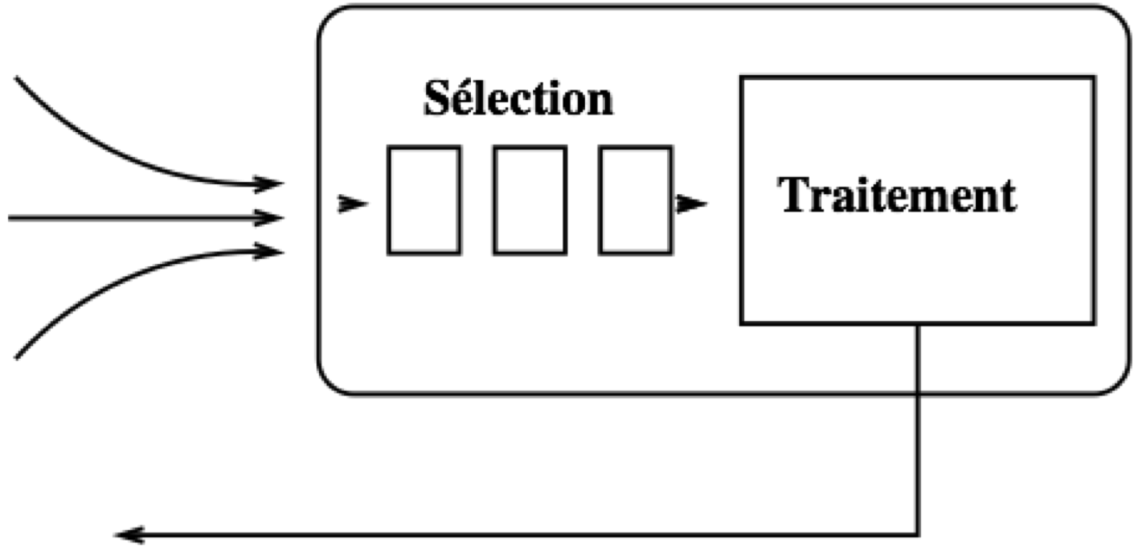
Serveur de nom

…

Mises en oeuvre

Serveur unique

Traitement successif des requêtes en fonction des services



Serveur unique : analyse

Avantages

- solution la plus simple
- pas ou peu de contraintes

Inconvénients

- solution séquentielle
- sous exploitation de la machine
- pas de souplesse

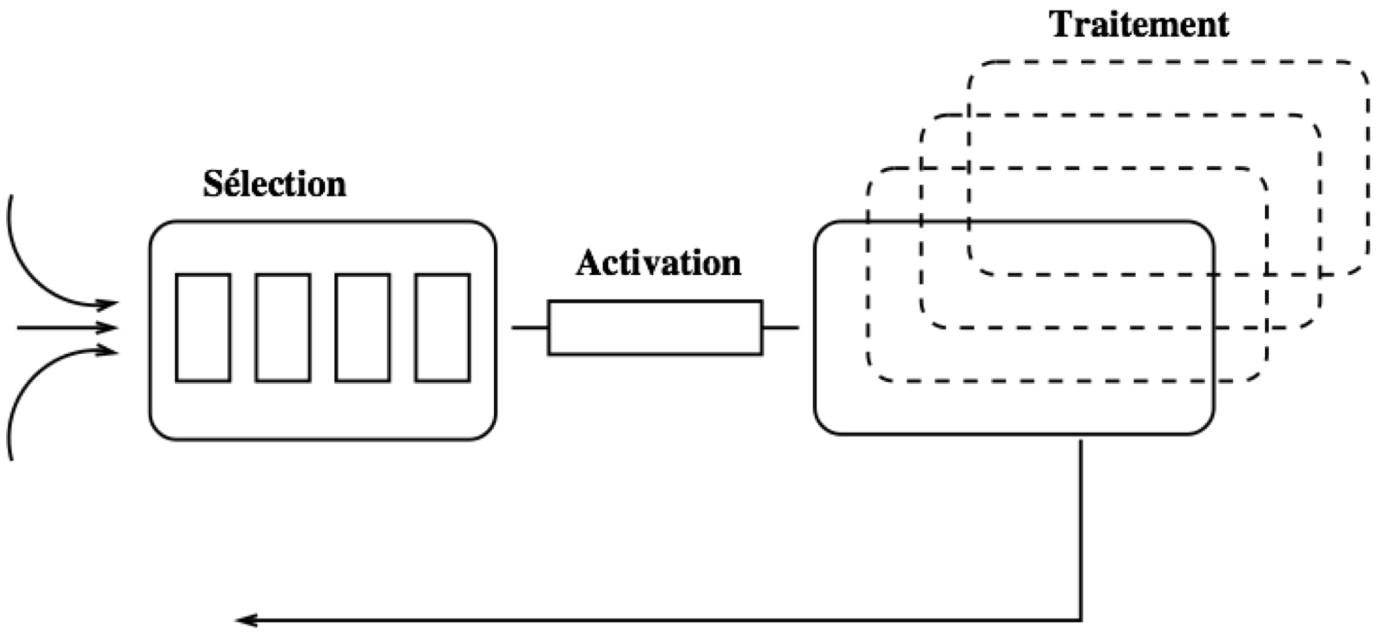
Serveur unique : procédure

```
while(true){
      receive_request(client_id,msg)
      extract(msg,service_id,params)
      exec_service[service_id](params,result)
      send(client_id,result)
}
```

Un processus par service

Classement des requêtes en fonction des services demandés

Exécution concurrente par les différents services



Un processus par service : analyse

Avantages

- gestion évoluée des services
- Traitement concurrent
- gestion simple des ressources non partagées

Inconvénients

- sous exploitation de la machine
- gestion « complexe » des ressources partagées

Un processus par service : procédure

Processus veilleur

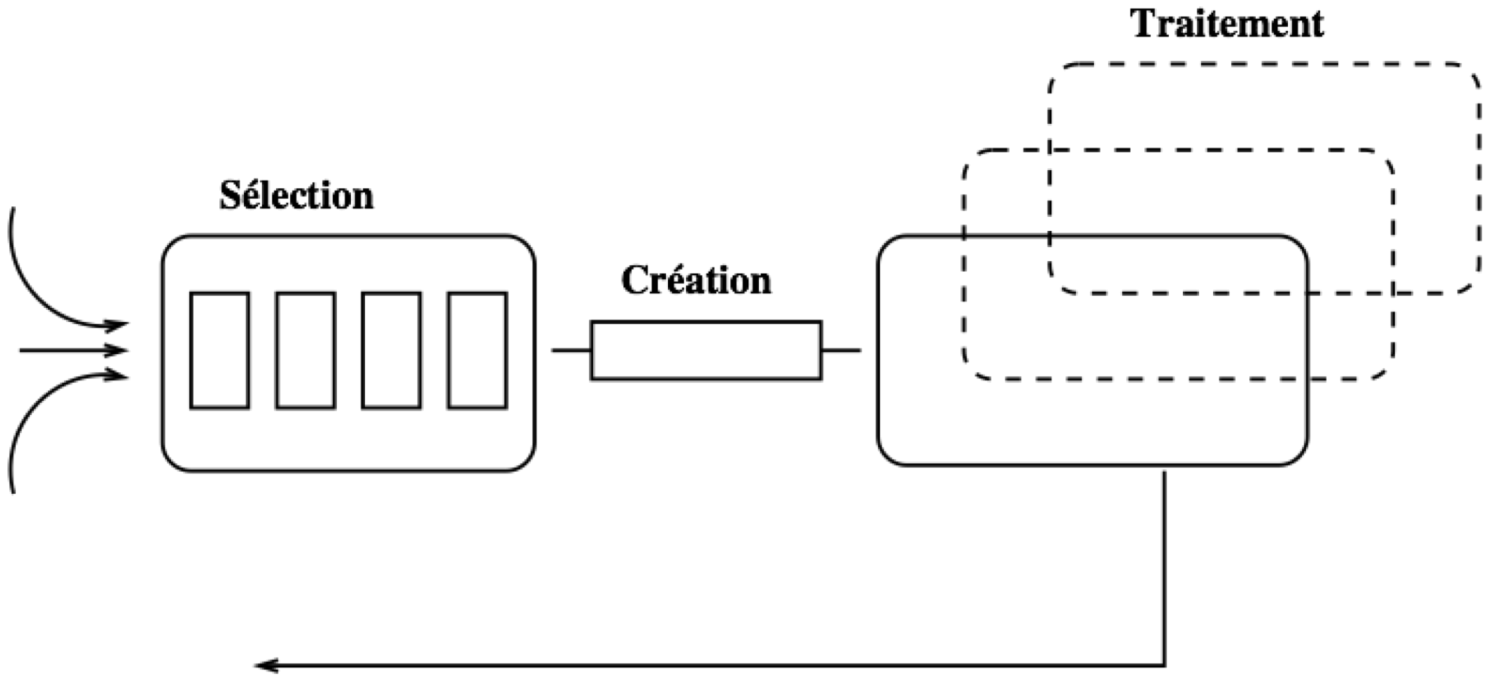
```
while(true){
    receive(client_id, msg)
    extract(message_id, service_id, params)
    work_to_do.put(client_id,service_id,params)
}
```

Pool d'exécutant

```
while(true){
    work_to_do.get(client_id,service_id,params)
    do_service[service_id](params,result)
    send(client_id,result)
}
```

Allocation dynamique par service

Traitement des requêtes lors de l'arrivée, par des processus alloués dynamiquement



Allocation dynamique par service : analyse

Avantages

- exploitation rationnelle des ressources
- gestion dynamique du système

Inconvénients

- gestion de complète de toutes les ressources

Nécessité de mettre en place un ordonnanceur

Allocation dynamique par service : Procédure

Processus veilleur

```
while(true){
   receive(client_id, msg)
   extract(message_id, service_id, params)
   p=create_thread(client_id,service_id,params)
}
```

Exécution dynamique

```
  do_service[service_id](params,result)
  send(client_id,result)
  exit
```

Services / données / exécutions

Serveur et données

Gestion des données sur le serveur

Persistance des données sur le serveur

Exploitation des données d’un appel sur l’autre

Données non rémanentes

Pas de mémorisation de l'état des données sur le serveur

Exécution uniquement en fonction des paramètres

Situation idéale :

- pas de persistance des services
- pas de concurrence
- pas de gestion de la tolérance aux fautes

Calcul d'une fonction scientifique

Données rémanentes

Modification de l'état de données sur le serveur

Exécutions successives : modification du contexte du serveur

Mise en place de solution de gestion :

- persistance des services
- Concurrence
- tolérance aux fautes

Serveur de fichiers réparti, accès concurrent : système de lock

Visions de la rémanence

Session

- mise en place de la rémanence d'un utilisateur

Rémanence globale

- partage de ressources entre des connexion

Nouvelle définition de la portée des objets

Notion d’état

Gestion des états des ressources

Pas lié uniquement aux données

Définition du cycle de vie des services

Définition du mode d’exécution des services

Exécution sans état

Invocations indépendantes et uniques

Pas de lien entre les invocations

Invocations en aveugle

Invocation d’un service d’accès aléatoire à un fichier

Exécution avec états

Invocation dépendante des services

Exécution en fonction de l'état laissé par les appels antérieurs

- gestion des utilisateurs

Accès séquentiel à un fichier

Etat et données

Solutions de gestion des états et des données

- sur le serveur
- sur le client

Gestion serveur

Mémorisation des états des clients

Sécurisation des données

Problème pour la tolérance aux fautes

Coût en ressources serveur ?

- Montée en charge

Gestion de l'association client / données

Gestion client

Libération des ressources serveur

Données

- paramètre des appels
- transférées sur le réseau
- accessibles aux clients

Pas de gestion de fautes sur le serveur

Gestion des fautes

Gestion des fautes

Côté serveur

Côté client

Types de fautes

- Définitives
- Transitoires
  - congestion du réseau
  - surcharge du serveur
  - Corruptions

La gestion d’une faute dépend de son impact sur le système

Panne du serveur

Conséquences

- attente indéfinie du client
- perte des données
- perte des états
- engorgement du réseau (*retry*)

Détection de la panne ?

- vue du client : pas de réponse à une requête
  - panne du serveur ?
  - congestion du serveur ?
  - congestion du réseau ?

Panne du serveur : gestion côté client

Utilisation d'un *timeout*

*S*tratégie de reprise :

- Abandon
- ré-essaie
- choix d'un autre serveur

Risque de d'exécution multiple d'une même requête

- identification des requêtes

Panne du serveur : gestion côté serveur

Journalisation

- Enregistrement
  - des opérations
  - des états
- stratégies de reprise
  - remise dans un état sauvegardé
  - état des clients ?

Panne du client

Conséquences

- risque d'exécution de travaux inutiles
- confusion après reprise entre les anciennes et les nouvelles réponses

Détection de la panne

- pas d'*ACK* à la réponse
  - panne du client ?
  - congestion du client ?
  - congestion du client ?

Panne du client : gestion côté serveur

Détecter les requêtes avant / après la panne

- identification des requêtes

Destruction des requêtes qui précèdent la faute

- vidage du buffer serveur
  - réponses non acquittées
  - nettoyage des messages reliquats sur le client

Risque d'inconsistance entre les états du client et du serveur

Conclusion

Limites du modèle

Modèle de structuration

- description des interactions entre 2 acteurs
- absence de vision globale de l'application
- schéma d'exécution élémentaire (appels synchrones)

Absence de propriétés

- Synchronisation
- tolérance aux fautes
- protection des données
  - Locale
  - Transférées
- …

Ajout au modèle

Mise en place de services de structuration

- Sécurité
- Désignation
- Fichier
- …

Spécification d’un ensemble de services satellites

- création d’un middleware