14 PRATIQUE PROCÉDURALE

Déroulement de l'activité

Description	Début	Fin	Durée
Retour sur exercices preparatoires	09h00	09h20	00h20
Transparence dans le paradigme client-serveur	09h20	09h40	00h20
Couplage des composants.	09h40	10h00	00h20
Identifier l'hétérogénéité.	10h00	10h20	00h20
PAUSE	10h20	10h35	00h15
Transparence de mise en oeuvre	10h35	10h50	00h15
Transparence de concurrence	10h50	11h10	00h20
Transparence de localisation	11h10	11h25	00h15
Transparence de réplication	11h25	11h40	00h15
Microkernel et Microservices	11h40	11h50	00h10
Technologies sous-jacentes aux conteneurs	11h50	12h00	00h10

Informations pour la personne tutrice et communication aux étudiantes et aux étudiants

Il faut aider les étudiants à se questionner au sujet des différentes manières de concevoir le logiciel et les implications que leurs choix vont avoir sur la sécurité, la fiabilité et la performance du système.

Le contenu du guide de l'étudiante et de l'étudiant pour cette activité suit. Les solutions et les directives y sont ajoutées.

But de l'activité

— Cette activité a pour but de vous familiariser avec les concepts et les architectures de systèmes répartis, de même qu'avec les technologies de conteneurisation.

14.1 EXERCICES PRÉPARATOIRES

P.P1 Questions.

Répondez aux questions suivantes avant de vous présenter au Procédural.

1. Qu'est-ce qu'une application centralisée ?

   Une application centralisée rassemble les ressources nécessaires à un traitement sur un hôte central, généralement à l'aide de terminaux qui sont connectés à un ordinateur

central. Cet ordinateur peut lui-même contrôler directement tous les périphériques (s'ils lui sont physiquement connectés) ou ils peuvent être gérés par un serveur de terminal. Si les terminaux en ont la capacité, ils peuvent être en mesure de se connecter à l'ordinateur central directement depuis un réseau informatique. L'ordinateur central peut être un serveur central dans le cas d'une architecture client-serveur. Les terminaux peuvent être de type passif ou actif, ils peuvent afficher des environnements CLI, TUI ou GUI, il peut s'agir d'une console système, d'une imprimante d'un client léger ou d'un client plus important par exemple.

2. Qu'est-ce qu'une application distribuée ?

   L'architecture distribuée désigne un système d'information pour lequel l'ensemble des ressources utilisés ne se trouvent pas au même endroit ou sur la même machine. Ce concept, dont une version peut être une combinaison de transmissions du type client-serveur, s'oppose à celui d'application centralisée .

   Internet est un exemple de réseau distribué puisqu'il ne possède aucun nud central. Les architectures distribuées reposent sur la possibilité d'utiliser des objets qui s'exécutent sur des machines réparties sur le réseau et communiquent par messages au travers du réseau.

3. Qu'est-ce qu'une liaison synchrone ?

   La liaison synchrone, dans laquelle émetteur et récepteur sont cadencés à la même horloge. Le récepteur reçoit de façon continue (même lorsque aucun bit n'est transmis) les informations au rythme où l'émetteur les envoie. C'est pourquoi il est nécessaire qu'émetteur et récepteur soient cadencés à la même vitesse. De plus, des informations supplémentaires sont insérées afin de garantir l'absence d'erreurs lors de la transmission.

   Lors d'une transmission synchrone, les bits sont envoyés de façon successive sans séparation entre deux caractères, il est donc nécessaire d'insérer des éléments de synchronisation, on parle alors de synchronisation au niveau caractère.

   Le principal inconvénient de la transmission synchrone est la reconnaissance des informations au niveau du récepteur, car il peut exister des différences entre les horloges de l'émetteur et du récepteur. C'est pourquoi chaque envoi de données doit se faire sur une période assez longue pour que le récepteur la distingue. Ainsi, la vitesse de transmission ne peut pas être très élevée dans une liaison synchrone.

4. Qu'est-ce qu'une liaison asynchrone ?

   L'émetteur émet le signal d'horloge et l'information en même temps sur la même ligne, la trame de transmission de l'information doit être assez longue. La liaison asynchrone,

dans laquelle chaque caractère est émis de façon irrégulière dans le temps (par exemple un utilisateur envoyant en temps réel des caractères saisis au clavier). Ainsi, imaginons qu'un seul bit soit transmis pendant une longue période de silence... le récepteur ne pourrait savoir s'il s'agit de 00010000, ou 10000000 ou encore 00000100... Afin de remédier à ce problème, chaque caractère est précédé d'une information indiquant le début de la transmission du caractère (l'information de début d'émission est appelée bit START) et terminé par l'envoi d'une information de fin de transmission (appelée bit STOP, il peut éventuellement y avoir plusieurs bits STOP). bit stop arrête l'horloge et bit start la redémarre

5. Quelle est la différence entre une machine virtuelle (par exemple, VMware) et la virtualisation par conteneurs ? Une machine virtuelle traditionnelle exécute un système d'exploitation complet, alors que la virtualisation par conteneurs exécute des processus isolés les uns des autres mais qui utilisent directement le système d'exploitation sous-jacent.

14.2 EXERCICES

Retour sur exercices preparatoires

09h00→09h20 Δ20min

P.E1 Transparence dans le paradigme client-serveur

09h20→09h40 Δ20min

Un message de demande de données unique est envoyé par le client au serveur d'application, qui est en contact avec deux bases de données distinctes. Le serveur d'application utilise le contenu du message pour créer une paire de messages de type requête, un pour chaque base de données spécifique.

Les réponses à ces messages sont renvoyées au serveur d'applications par les deux bases de données. Elles sont ensuite transmises par le serveur d'applications au client, en deux réponses distinctes, l'une par base de données.

1. Faire un schéma pour décrire la situation

   Il faut faire attention au fait qu'il y a une seule demande de données, mais qu'il y aura deux réponses. L'ordre des réponses peut varier. Faire réfléchir les étudiants à cette situation.

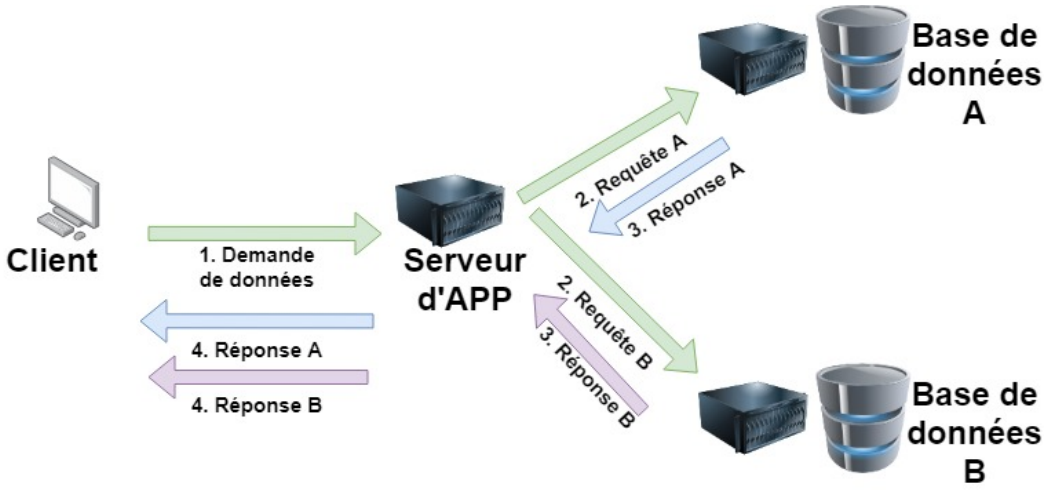

FIGURE 14.1 Solution Question 1.

2. Expliquer les implications en matière de transparence de cette décision de conception.

   Les implications en matière de transparence de la conception sont les suivantes :

   — Envoi de deux réponses pour une seule requête envoyée par le client. Le client doit fournir la logique pour être en mesure de recevoir et traiter les messages sans confusion.

   — L'approche révèle des indices quant à l'architecture interne du service, en particulier dans ce cas, qu'il existe des serveurs de bases de données multiples (rupture de la vue système unique).

   — D'une certaine manière, cette méthode est robuste et dans une certaine mesure assure la transparence des défaillances dans la mesure où elle permet au client de recevoir des données d'une base de données quand l'autre est indisponible.

   — L'approche n'est pas universellement appropriée, étant donné que dans de nombreuses applications de bases de données, il est souhaitable de ne pas recevoir des réponses incomplètes à partir de requêtes.

3. Quelles seraient les conséquences de la combinaison des deux réponses (une réponse en provenance de chacune des bases de données) en une seule réponse transmise au client ?

   Les conséquences sont les suivantes :

— La complexité supplémentaire dans le serveur d'applications. Les deux réponses des serveurs de base de données arrivent de façon asynchrone sur le serveur d'application, donc il y aurait besoin d'avoir un moyen d'attendre jusqu'à ce que les réponses soient disponibles avant de passer un message de réponse unique au client.
— Les aspects de synchronisation pourraient être difficiles, en particulier la détermination du temps d'attente du serveur d'application pour la paire de réponses :
  — Si le serveur attend indéfiniment, le service sera peu fiable dans le cas où l'un des serveurs de base de données tombe en panne.
  — Si au contraire le serveur attend trop longtemps avant d'expirer, le service sera robuste mais avec un temps de latence plus élevé que nécessaire.
  — Si enfin le serveur n'attend assez longtemps avant d'expirer, les réponses valides des bases de données seront perdues.

P.E2 Couplage des composants.

09h40→10h00 Δ20min

Identifier le(s) type(s) de couplage se produisant dans chacun des scénarios suivants :

1. Une application dans une architecture en couche et fonctionnant en utilisant le *middleware*

   Les applications Client-Serveur sont directement couplées (dans le cas fréquent où le client identifie le serveur explicitement) et faiblement couplées lorsque le middleware est un intermédiaire de communication.

   Alors : **Faible, Indirect**

2. Un prototype du système à trois niveaux qui utilise des adresses établies et figées dans le code (*hardcoded*) pour identifier les emplacements des composants.

   L'utilisation des adresses figées dans le code implique la conception de connectivité décidée au préalable. Les composants adjacents se raccordent directement les uns aux autres ; par conséquent, dans ce cas, le couplage est fort et direct.

   Les niveaux non adjacents (l'interface utilisateur et le niveau d'accès aux données) se connectent indirectement l'un à l'autre, par le niveau intermédiaire. Dans ce cas, le couplage est **fort** et *indirect* : **Fort, Indirect**

3. Une demande peer-to-peer dans lequel les composants se connectent les uns aux autres lors de sa découverte dans le réseau local (le processus de découverte est automatique, basé sur l'utilisation de messages de diffusion)

   Faible, Direct

1. **Tight coupling (couplage fort)** Se caractérise par des connexions décidées au moment de la conception entre des composants spécifiques. Les références explicites à d'autres composants créent des dépendances directes entre les composants, ce qui signifie que l'application ne peut fonctionner que si tous les composants requis sont disponibles. Si un composant échoue, les autres composants qui en dépendent échouent ou du moins ne peuvent pas fournir les fonctionnalités auxquelles contribue le composant défaillant. Ainsi, une conception étroitement copiée a tendance à augmenter la sensibilité à l'échec, à réduire la flexibilité, à réduire la scalabilité, à augmenter la difficulté de maintenance et à inhiber la reconfigurabilité du temps d'exécution.
2. **Loosely coupling (couplage faible)** Les composants n'ont pas de connexions avec d'autres composants spécifiques décidés au moment de la conception. Cette forme de couplage est caractérisée par des services intermédiaires, qui fournissent la communication entre les composants (tels que les middleware), et / ou par l'utilisation de mécanismes de découverte de composants dynamiques (tels que la publicité de services). les composants sont couplés à des références indirectes. Il s'agit d'une approche flexible d'exécution car il est possible pour le service intermédiaire de modifier les références à d'autres composants en fonction de la disponibilité d'autres composants au fur et à mesure ; ainsi, les composants ne dépendent pas directement d'instances spécifiques d'autres composants ou processus.
3. **Direct coupling (couplage directe)** Il se caractérise par le fait que les connexions de niveau processus à processus correspondent à la communication au niveau métier de l'application. Les connexions logiques de la couche de transport mappent directement sur la connectivité de niveau supérieur. Par exemple, il peut y avoir une connexion TCP ou UDP directe entre les composants de niveau métier.
4. **Direct coupling (couplage directe)** Décrit la situation dans laquelle les composants interagissent via un intermédiaire. Un système de négociation d'actions fournit un exemple où les clients ont des relations privées mais voient les effets des transactions d'autres clients, ce qui peut conduire à de nouvelles transactions. Un autre exemple est fourni par un jeu multijoueur hébergé sur un serveur central, où les clients du jeu sont conscients de la présence de l'autre au niveau de l'application mais pas directement connectés entre eux en tant que composants. Chaque client est connecté uniquement au serveur et toute communication entre les clients est transmise au serveur et transmise à l'autre client.
5. **Isolated coupling (couplage isolé)** Décrit la situation dans laquelle les composants ne sont pas couplés entre eux et ne communiquent pas entre eux bien qu'ils fassent partie du même système. Par exemple, une paire de clients connectés chacun au même

serveur n'a pas besoin de communiquer directement ni même de savoir que les autres se terminent.

P.E3 Identifier l'hétérogénéité.

10h00→10h20 Δ20min

Identifier les classes d'hétérogénéité qui peuvent se présenter dans chacun des scénarios suivants :

1. Un laboratoire informatique avec des ordinateurs achetés en lots sur une période de deux ans, installés avec le système d'exploitation Windows et la même suite d'applications logicielles sur tous les des ordinateurs.

   Il est possible qu'il n'y ait aucune hétérogénéité présente dans ce cas. Toutefois, l'achat des lots différents d'ordinateurs sur une période de temps prolongée est susceptible de conduire à des différences de ressources, par exemple, les lots ultérieurs peuvent avoir plus grande mémoire, plus grand stockage sur disque dur, ou différentes variantes de la CPU. Ainsi, il est probable qu'il y ait une certaine hétérogénéité de la performance actuelle. En outre, il est possible que les différentes versions du système d'exploitation Windows sont en cours d'utilisation, qui pourrait donner lieu à une hétérogénéité du système d'exploitation.

2. Un petit système de réseau d'entreprise comprenant quelques ordinateurs de bureau différents, exécutant différentes versions du système d'exploitation Windows et quelques ordinateurs portables fonctionnant également avec le système d'exploitation Windows.

   Le matériel des plates-formes est susceptible d'être compatible (les ordinateurs portables devraient fournir la même interface matérielle que le PC), et donc, les plates-formes peuventt offrir les mêmes interfaces avec les processus et les systèmes d'exploitation. Il est probable que les plates-formes offrent différents niveaux de ressources, de sorte que le système exposera l'hétérogénéité des performances. Le système global peut être hétérogène si différentes versions du système d'exploitation Windows sont utilisées en parallèle.

3. Un système qui prend en charge une application commerciale client-serveur. Un puissant ordinateur exécutant le système d'exploitation Linux est utilisé pour héberger le processus serveur. Les utilisateurs accèdent au service par le biais d'ordinateurs de bureau à faible coût de fonctionnement du système d'exploitation Windows.

L'hétérogénéité de la performance a été introduite délibérément pour assurer que le processus de serveur dispose de ressources suffisantes pour atteindre la performance appropriée. Utilisation de l'hétérogénéité du système est également présent.

PAUSE

10h20→10h35 Δ15min

P.E4 Transparence de mise en oeuvre

10h35→10h50 Δ15min

1. Dans le cadre de la communication de composantes logicielles, que veut dire IDL ?
   IDL = Interface Definition Language.
2. Comment le IDL contribue à la transparence de la mise en oeuvre ?
   IDL facilite l'interopérabilité lorsque les composants sont développés en utilisant différents langages de programmation. Elle le fait en fournissant une représentation intermédiaire universelle des interfaces méthode d'appel, qui est indépendant du langage. Par exempleă:

```
La définition IDL:

module HelloApp
{
    interface Hello
    {
        string sayHello();
        oneway void shutdown();
    };
};

Le code Java:

//HelloOperations.java
package HelloApp;


/**
* HelloApp/HelloOperations.java
* Generated by the IDL-to-Java compiler (portable), version "3.0"
```

```
* from Hello.idl
*/

public interface HelloOperations
{
    String sayHello ();
    void Shutdown ();
} // interface HelloOperations
```

3. Pourquoi le IDL ne contient que des définitions d'interfaces sans en inclure la mise en oeuvre ?

   Le IDL doit représenter des demandes d'appel d'une façpm indépendante de la langue et est donc seulement concerné par les interfaces des composants. Le IDL ne définit pas le comportement ou la logique interne des composants de communication, il n'y a donc pas besoin d'IDL pour exprimer tous les détails de mise en oeuvre.

P.E5 Transparence de concurrence

10h50→11h10 Δ20min

1. Quel est le principal défi quand on veut faciliter l'accès simultané aux ressources partagées ?

   Le principal défi est de maintenir la cohérence.

2. Comment un système transactionnel contribue à la transparence de la concurrence ?

   Un système transactionnel empêche l'accès à des ressources partagées avec chevauchement. Les propriétés touchées sont l'atomicité, cohérence, isolation, durabilité et assurent collectivement que le système est laissé dans un état cohérent après chaque événement d'accès et/ou mise à jour terminé.

3. Expliquez les proprietés ACID d'une transaction.

   Propriétés ACID d'une transaction :

   Atomicité : Une transaction est une unité atomique de traitement. Elle est réalisée totalement ou elle ne l'est pas du tout (pas d'exécution partielle)

   Consistance : L'exécution correctte d'une transaction doit prendre la base de données d'un état cohérent et la ramener à autre état cohérent.

   Isolation : Une transaction ne laisse pas voir ses changements aux autres transactions alors qu'elles ne sont pas officielles (c'est-à-dire, les transactions qui n'ont pas encore eu de *commit*).

Durabilité : Une fois que la transaction modifie la base de donnes et les changements sont officiels (*commit*), les modifications ne peuvent être perdues à cause d'une défaillance ultérieure.

P.E6 Transparence de localisation

11h10→11h25 Δ15min

1. Pourquoi la transparence de localisation est l'une des exigences les plus courantes des applications distribuées ? Les composants ont besoin de communiquer avec d'autres composants, peu importe où ils se trouvent. Il doit y avoir soit un moyen automatisé pour trouver l'emplacement d'un composant ou encore un moyen d'envoyer un message à un autre composant par un service intermédiaire sans que l'expéditeur connaisse l'emplacement de ce composant.
2. Comment un service de nom contribue à la réalisation de la transparence de localisation ? Un service de nom résout un nom de composant (ou un nom de ressource) dans son adresse. Par conséquent, l'expéditeur d'un message ne doit connaitre que l'identité du composant cible et non là où il se trouve.
3. Expliquez le principe d'un serveur de *DNS* Le *Domain Name System*, généralement abrégé *DNS*, qu'on peut traduire par ń système de noms de domaineż, est le service informatique distribué utilisé pour traduire les noms de domaine Internet en adresse IP ou autres enregistrements. En fournissant dès les premières années d'Internet (autour de 1985) un service distribué de résolution de noms, le DNS a été un composant essentiel du développement du réseau.

   Les équipements (hôtes) connectés à un réseau IP, comme Internet, possèdent une adresse IP qui les identifie sur le réseau. Ces adresses sont numériques afin de faciliter leur traitement par les machines. En IPv4, elles sont représentées sous la forme :

   "xxx.xxx.xxx.xxx",

   où "xxx" est un nombre entre 0 et 255 (en représentation décimale). En IPv6, les adresses sont représentées sous la forme :

   "xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx",

   où "xxxx" représente une valeur hexadécimale de 0000 à FFFF.

   Pour faciliter l'accès aux hôtes sur un réseau IP, un mécanisme a été mis en place pour associer un nom à une adresse IP. Ce nom, plus simple à retenir, est appelé "nom de domaine". Résoudre un nom de domaine consiste à trouver l'adresse IP qui lui est associée.

P.E7 Transparence de réplication

11h25→11h40 Δ15min

1. Quelles sont les principales motivations pour la réplication des données et/ou des services ? La réplication des données et/ou des services contribue à la robustesse, la disponibilité, la réactivité et l'évolutivité.
2. Quel est le principal défi lors de l'implémentation de la réplication ? La réplication consiste à créer des copies multiples des ressources de données et des informations d'état. Cela introduit la possibilité pour les différentes instances d'une ressource répliquée de devenir incompatibles. Par conséquent, le maintien de la cohérence est le principal défi lors de la mise en uvre de réplication.
3. Expliquez le fonctionnement du protocole en deux phases *Two Phases Commit*

   Ce protocole permet d'assurer, en cas de pannes, l'atomicité des mises à jour effectuées sur différents sites par une transaction distribuée.

   Première phase : Préparer Le coordinateur demande aux autres sites s'ils sont prêts à faire le *commit* de leurs mises à jour.

   Deuxième phase Deux possibilités :

   Succès : (*Commit*). Tous les participants effectuent leur validation sur ordre du coordinateur.

   Échec : (*Abort*). Si un participant n'est pas prêt, le coordinateur demande à tous les autres sites de défaire la transaction
4. Comment le protocole de la validation en deux phases (*Two Phases Commit*) contribue à la réalisation des mécanismes de réplication robustes ?

   Le protocole de la validation en deux phases garantit que les mises à jour sont effectuées à toutes les répliques des instances d'une ressource de données ou à aucun d'entre eux, maintenant ainsi la cohérence.

P.E8 Microkernel et Microservices

11h40→11h50 Δ10min

Quelle est la différence entre les modèles d'architecture micro-noyau (*microkernel*) et micro-services

Les composants du micro-noyau peuvent être mis à niveau séparément sans nécessiter de redémarrage de la machine hôte, ce qui rend également le système plus robuste. Il y avait aussi l'espoir que le système serait plus facile à écrire. Cependant, les micro-noyaux sont

assortis d'une pénalité de performance, et les progrès dans l'écriture des modules dans les noyaux monolithiques ont annulé le dernier avantage. En effet, écrire les parties d'un micro-noyau est plus complexe car la communication entre les différentes parties est plus difficile, puisqu'elles ne font plus partie d'un seul programme.

Les microservices sont de petits serveurs Web qui gèrent des tâches uniques. (Dans la communauté Python, les microservices sont souvent écrits en utilisant le *framework* Flask.) Cette architecture est très similaire à une architecture à micro-noyau : un *proxy*, comme nginx, agit sur le noyau qui coordonne les nombreux services.

La plupart des avantages sont les mêmes dans les deux cas : chaque service peut échouer sans sortir les autres, ils peuvent être mis à jour séparément, et en théorie chaque service peut être petit et simple.

P.E9 Technologies sous-jacentes aux conteneurs

11h50→12h00 Δ10min

1. Quelles sont les trois éléments sous-jacents du système d'exploitation Linux qui ont permis l'émergence des conteneurs ?

   Il s'agit des *namespaces*, des *cgroups* et des systèmes de fichiers à union.

2. Quels sont leurs rôles respectifs ?

   namespaces Permettent de nommer des ressources et de limiter leur visibilité à un sous-ensemble des processus actifs

   cgroups Permettent de limiter l'accès et les modifications des ressources du système

   systèmes de fichiers à union Permettent de partager des portions du système de fichiers en lecture seule, facilitant la réutilisation des répertoires et fichiers entre les conteneurs

3. Quelle(s) fonctionnalité(s) additionnelle(s) facilite(nt) l'utilisation des conteneurs ?

   Les registres (*registries*) permettent de facilement partager des images, c'est-à-dire des portions en lecture seule de systèmes de fichiers à union.