



# Introduction aux systèmes répartis

Module 1

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① Introduction
- ② Historique
- ③ Les défis
- ④ Modèles de systèmes
- ⑤ Retour sur la réseautique et sécurité



# Introduction aux systèmes répartis

- 1 Introduction
- 2 Historique
- 3 Les défis
- 4 Modèles de systèmes
- 5 Retour sur la réseautique et sécurité

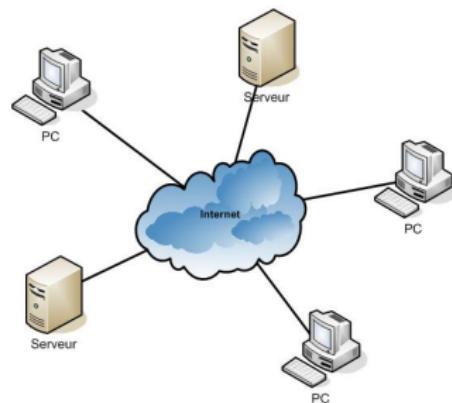
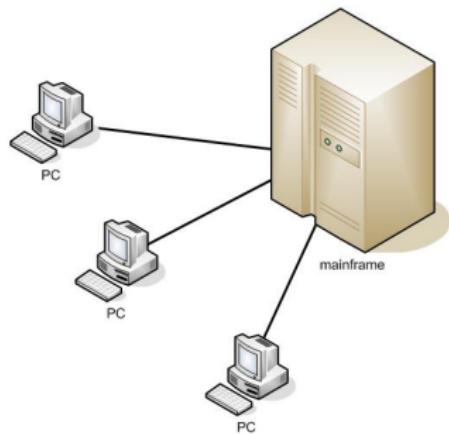


# Évolution des organisations informatiques

- Serveur centralisé avec terminaux
  - Serveur coûteux;
  - Engorgement au serveur.
- Ordinateurs personnels
  - Faible coût d'achat;
  - Grand choix d'applications;
  - Autonomie mais manque de service et de coordination.
- Systèmes répartis
  - Le réseau partout et en continu;
  - Matériel et logiciels modulaires à faible coût;
  - Environnement hétérogène mais protocoles normalisés;
  - Redécoupage des responsabilités client et serveur;
  - Systèmes de plus en plus complexes.



# Système centralisé versus réparti

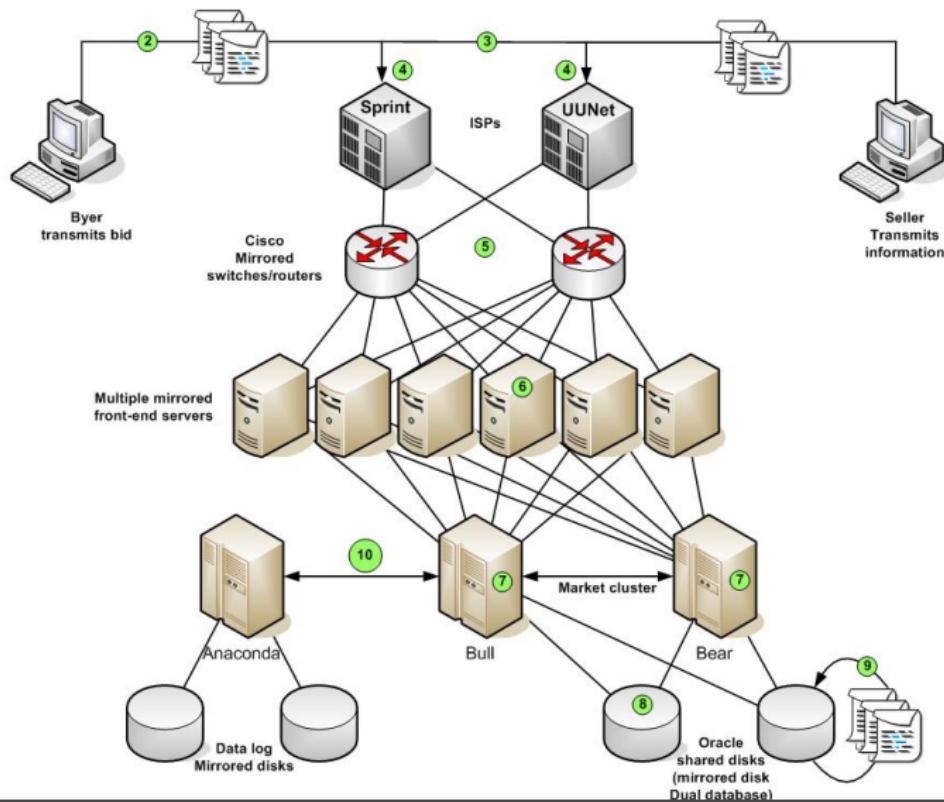


## Système réparti

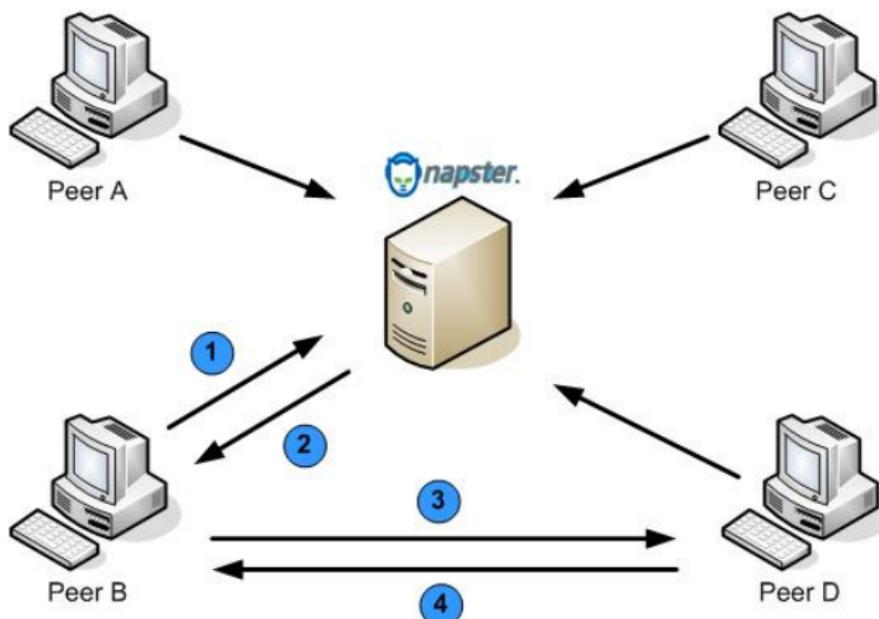
- Système dont les composantes sont réparties sur plusieurs ordinateurs en réseau et qui communiquent entre eux et coordonnent leurs actions uniquement par transmission de messages.
- Un ensemble d'ordinateurs indépendants qui, du point de vue de l'usager, apparaissent comme un système unique et cohérent.
- Une définition alternative par Leslie Lamport (1987):
  - “You know you have one when the crash of a computer you've never heard of stops you from getting any work done.”



# Architecture répartie de eBay

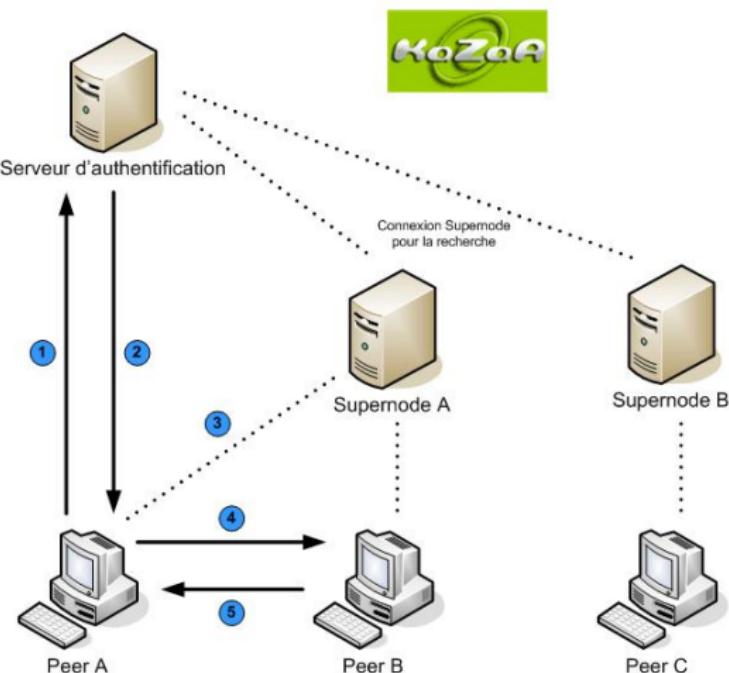


## Réseau P2P centralisé



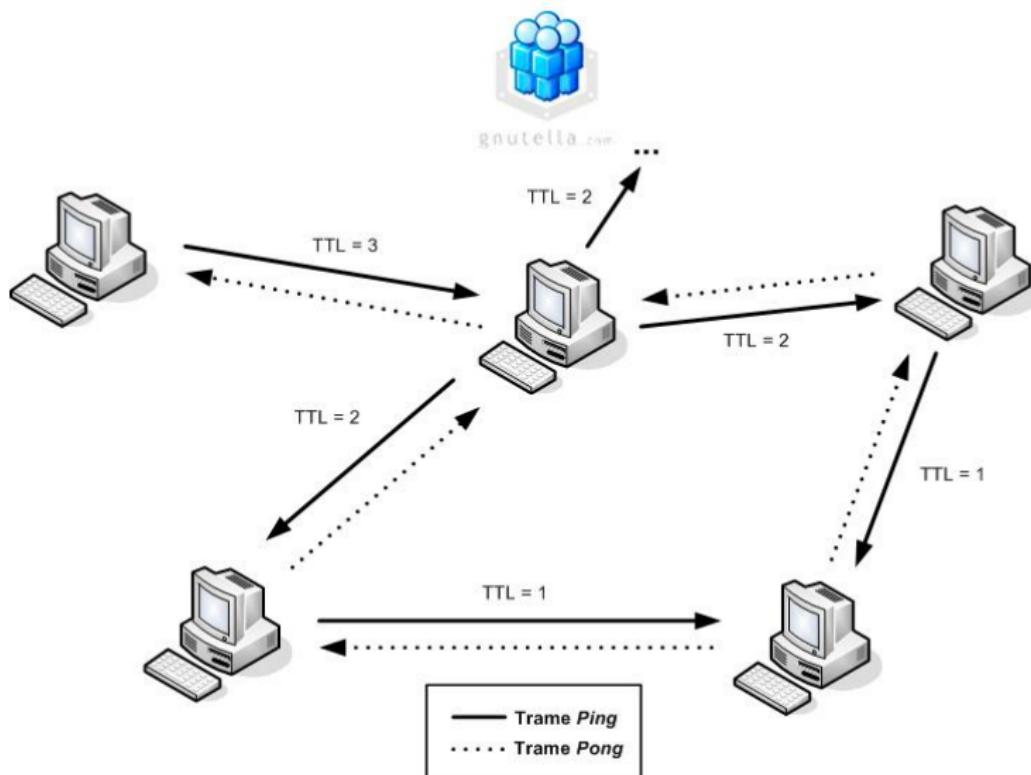
1. Recherche un fichier sur le serveur central
2. Serveur renvoie la liste des peer qui possèdent le fichier
3. Peer B contacte Peer D pour obtenir le fichier
4. Transfert du fichier

# Réseau P2P hybride



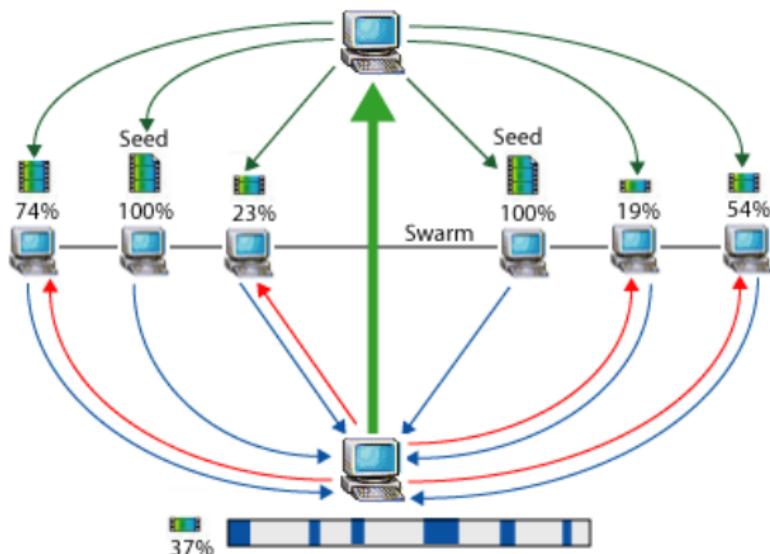
1. Authentification auprès du serveur
2. Serveur renvoie le *Supernode* le plus près
3. Effectue la recherche du fichier sur le *Supernode*
4. Contacte le peer possédant le fichier
5. Transfère le fichier

# Réseau P2P décentralisé



# Réseau P2P autre

BitTorrent tracker identifies the swarm and helps the client software trade pieces of the file you want with other computers.



Computer with BitTorrent client software receives and sends multiple pieces of the file simultaneously.

## Autres exemples

- L'Internet et tous ses protocoles, DNS, SMTP, BGP;
- L'intranet d'une entreprise;
- Le système Interac;
- Google, Facebook...



# Pourquoi les systèmes répartis

- Partage des ressources (données, périphériques...);
- Accès à des ressources distantes;
- Augmentation modulaire de la capacité du système;
- Possibilité de tolérance aux pannes.



# Inconvénients des systèmes répartis

- Plusieurs points de défaillance;
- Sécurité;
- Difficulté pour le système d'avoir un état global;
- Complexité accrue.



# Caractéristiques des systèmes répartis

---

- Les composantes du système:
  - Sont réparties matériellement et/ou géographiquement;
  - Sont autonomes;
  - Sont concurrentes;
  - Peuvent défaillir indépendamment;
  - Possèdent des horloges asynchrones;
  - Communiquent par envoi de message sur le réseau.



## Conséquences

- Nombreux points de défaillance possibles;
- Décalage de temps entre les horloges de chaque système;
- Pas d'état global;
- Pas de garantie que les messages sont reçus;
- Messages peuvent être interceptés, modifiés, ajoutés.



# Introduction aux systèmes répartis

- 1 Introduction
- 2 Historique
- 3 Les défis
- 4 Modèles de systèmes
- 5 Retour sur la réseautique et sécurité



## Historique

- Xerox DFS, Xerox PARC, Xerox Alto, Ethernet, 1977;
- Cambridge Distributed Computing Services (DCS), M68000, Cambridge ring, 1979;
- Système d'exploitation Locus de UCLA, VAX, Ethernet, 1980;
- Apollo Domain, Token Ring, 1980;
- Grapevine, Xerox PARC, Xerox Alto, Ethernet, 1981  
(replicated distributed application-oriented database service);
- Cedar, Xerox PARC, Xerox Dorado, Ethernet, 1982,  
development environment for office and personal systems;
- Amoeba, Vrije University, VAX/M68000..., Ethernet, 1984,  
distributed system based on capabilities;
- Unix BSD 4.2 + SUN RPC/NFS, Vax/SUN, Ethernet, 1985;
- Mach, CMU, VAX/SUN, Ethernet, 1986, système  
d'exploitation réparti basé sur un micro-noyau.



## Historique (suite)

- World Wide Web, HTTP sur TCP/IP, 1992;
- Groupe OMG, CORBA (Common Object Request Broker Architecture) 1992;
- Langage Java, RMI (Remote Method Invocation), 1995;
- Google, 1998;
- VMWare, 1998;
- Langage C#, Remoting, 2001;
- Facebook, 2004;
- Amazon EC2, 2006;
- iPhone, Android, 2007, 2008;
- OpenStack, 2010;
- Docker, 2013;
- Kubernetes et la Cloud Native Computing Foundation, 2015;

# Introduction aux systèmes répartis

- 1 Introduction
- 2 Historique
- 3 Les défis
- 4 Modèles de systèmes
- 5 Retour sur la réseautique et sécurité



# Les principaux problèmes à résoudre

- Répartition de l'application;
- Hétérogénéité des équipements et technologies, besoin d'interopérabilité;
- Ouverture de système;
- Sécurité;
- Évolutivité et mise à l'échelle;
- Tolérance aux fautes et la fiabilité/ Détection et isolation des fautes/défaillances;
- Concurrence, Synchronisation et Interblocage;
- Transparence;
- Validation et tests;



## Répartition de l'application

- Partitionnement de l'application en différents composants;
- Equilibrer la charge de l'application à travers différents composants répartis (client, noeuds de la grappe), statiquement ou dynamiquement;
- Architecture simple, propice à l'évolutivité et au maintien de la sécurité;



# Hétérogénéité

- Réseaux et protocoles utilisés;
- Matériel;
- Systèmes d'exploitation;
- Langages de programmation;
- Implémentations;
- Représentations internes.

## Solutions

- Protocoles et formats de stockage normalisés;
- Intergiciels d'adaptation (e.g. gRPC, CORBA, Java RMI, .NET).



## Systèmes ouverts

---

- Possibilité d'évoluer, de re-développer le système en tout ou en partie;
- Interopérabilité avec des systèmes complémentaires;
- Portabilité vers du nouveau matériel;
- Services développés selon des règles normalisées, formalisées à l'intérieur de protocoles, formats de stockage et interfaces de programmation.



## Evolution vers les systèmes ouverts

- Système unique, homogène;
- Développement interne, en plein contrôle;
- Applications commerciales prêtes à utiliser, plus performantes, moins chères;
- Fournisseur unique, perte de contrôle sur le prix et le cycle de mise à jour;
- Systèmes ouverts, interface de programmation (CORBA), protocoles (IIOP) et formats de stockage normalisés, code source ouvert, implémentation de référence libre (Orbit);
- Mélange de logiciels internes, logiciels libres et logiciels commerciaux



# Sécurité

---

- Transmettre des informations sensibles sur un lien de communication non sécuritaire et non fiable de manière sécuritaire;
- Confidentialité, intégrité, disponibilité.

# Évolutivité et mise à l'échelle

- Taille du système (nombre d'utilisateurs, de requêtes, de ressources);
- Etendue géographique (avec les latences associées);
- Structure administrative (décentralisée, sécuritaire);
- Architecture du logiciel réparti, séparer les politiques des mécanismes;



## Tolérance aux fautes et fiabilité :

- Les fautes et les défaillances sont plus courantes que dans les systèmes centralisés;
- Les défaillances sont habituellement indépendantes;
- Détection des fautes/défaillances;
- Masquage ou tolérance des fautes/défaillances;
- Redondance et réPLICATION;



# Concurrence

---

- Permettre au système de traiter simultanément plusieurs requêtes à une même ressource;
- Les opérations doivent être sérialisées ou donner un résultat cohérent équivalent.



# Transparence

---

- Masquer à l'utilisateur tous les aspects reliés à la répartition du système;
- Accès, localisation, concurrence, réPLICATION, défaillance, mobilité, performance, évolutivité.



## Validation et tests

- Comment tester le système complet? Chaque composante?
- Les fautes lors des tests pourraient être masquées par la tolérance aux pannes?
- Validation formelle de certaines portions.
- SPIN, modelchecker développé par Bell Labs,  
<http://spinroot.com/spin/whatispin.html>.
- UPPAAL est un modelchecker développé par l'Université Uppsala, en Suède et l'Université d'Aalborg en Danemark,  
<http://www.uppaal.com/>. Standard opensource pour la fiabilité et interopérabilité:
- Service Availability Forum (SAF) et Availability Management Framework (AMF), <http://www.saforum.org/>.



# Introduction aux systèmes répartis

- 1 Introduction
- 2 Historique
- 3 Les défis
- 4 Modèles de systèmes
- 5 Retour sur la réseautique et sécurité



# Modèles de systèmes

- Client-serveur (multiples, imbriqués, micro-service...)
- Client-proxy-serveur.
- Collègues (peer to peer).
- Client + code mobile - serveur.
- Agents mobiles.
- Ordinateur réseau ou client minimal (X, VNC, Citrix).
- Réseaux spontanés (découverte de ressources DHCP, réseaux infra-rouge, bluetooth).



# Modèles de pannes

---

- Auto-détection;
- Omission;
- Mauvaise réponse plus ou moins aléatoire;
- Erreur byzantine;
- Erreur de synchronisme.



## Exemple: World Wide Web

- Format HTML, XML, CSS, XSLT, XSL-FO, Javascript;
- Réseau TCP/IP;
- Convention pour les adresses (URL), et protocole pour les requêtes (HTTP);
- Client qui exécute un fureteur: envoi de requêtes par HTTP, affichage du résultat en XML/CSS, exécution d'applet;
- Serveur: sert des requêtes HTTP à partir de fichiers HTML, de fichiers de script (CGI, PHP, jsp), ou de modules spéciaux (XML, XSLT);
- Cette plate-forme est utilisée pour accéder de l'information, rechercher des documents, consulter des annuaires, faire du courriel, interagir avec des groupes de discussion...



# Introduction aux systèmes répartis

- 1 Introduction
- 2 Historique
- 3 Les défis
- 4 Modèles de systèmes
- 5 Retour sur la réseautique et sécurité



# Réseaux logiques

- Réseau logique bâti par-dessus un réseau physique (overlay network);
- Ethernet VLAN;
- Réseau défini par logiciel, par exemple avec OpenFlow;
- Réseau de machines virtuelles en infonuagique;



# Architectures de réseau

- PSTN
- Internet
- Multiprotocol Layer Switching (MPLS)
- Cellulaires (GSM, GPRS, EDGE, UMTS: 3G, LTE et WiMax Mobile :4G)
- Wi-Fi
- WiMAX
- Bluetooth
- Réseaux ad hoc



## Circuits commutés



# Internet

---

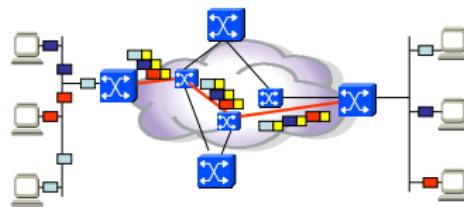
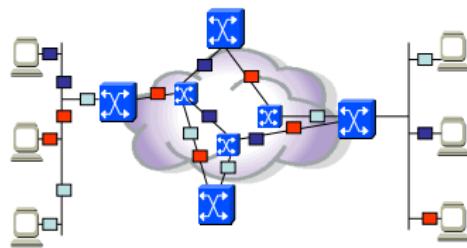
- Réseau mondial basé sur l'envoi de paquets;
- Familles de protocoles IP, UDP, TCP, FTP, SMTP, HTTP...;
- Les aspects techniques et architecturaux sont régis par l'Internet Engineering Task Force (IETF);
- Les protocoles sont documentés dans les Request For Comments (RFC);
- Réseaux locaux avec routage statique et réseau global avec routage dynamique;
- Initialement, rien n'était prévu pour assurer la qualité de service, par exemple afin de transmettre la voix ou le vidéo en temps réel.



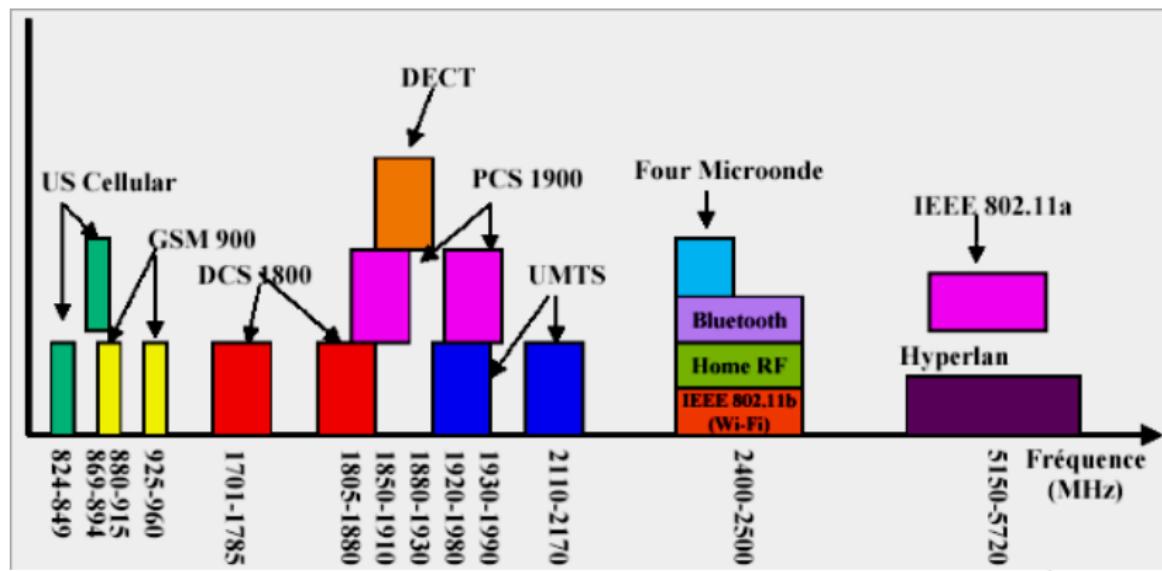
# Multiprotocol Label Switching (MPLS)

- Le principe du MPLS consiste à générer une étiquette courte, d'une longueur fixe, correspondant à un bref résumé de tout l'en-tête du datagramme IP.
- Le premier routeur MPLS rencontré apposera une telle étiquette et le datagramme pourra être envoyé très rapidement dans le réseau MPLS en fonction de cette étiquette.
- De l'autre côté du réseau, le datagramme IP sera de nouveau déballé et acheminé de la manière classique.
- L'étiquette n'est pas seulement créée en fonction de l'adresse de destination, mais aussi à partir de caractéristiques comme la qualité de service.
- Cette méthode peut être comparée à celle utilisée par la Poste. En mettant un code postal sur une lettre, il n'est pas nécessaire d'interpréter toute l'adresse avant d'arriver près de la destination.

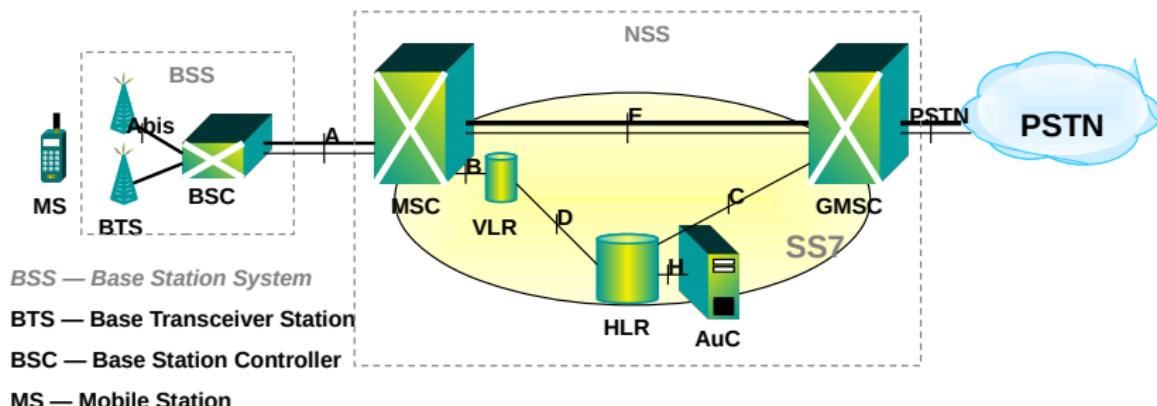
# Routage classique IP versus MPLS



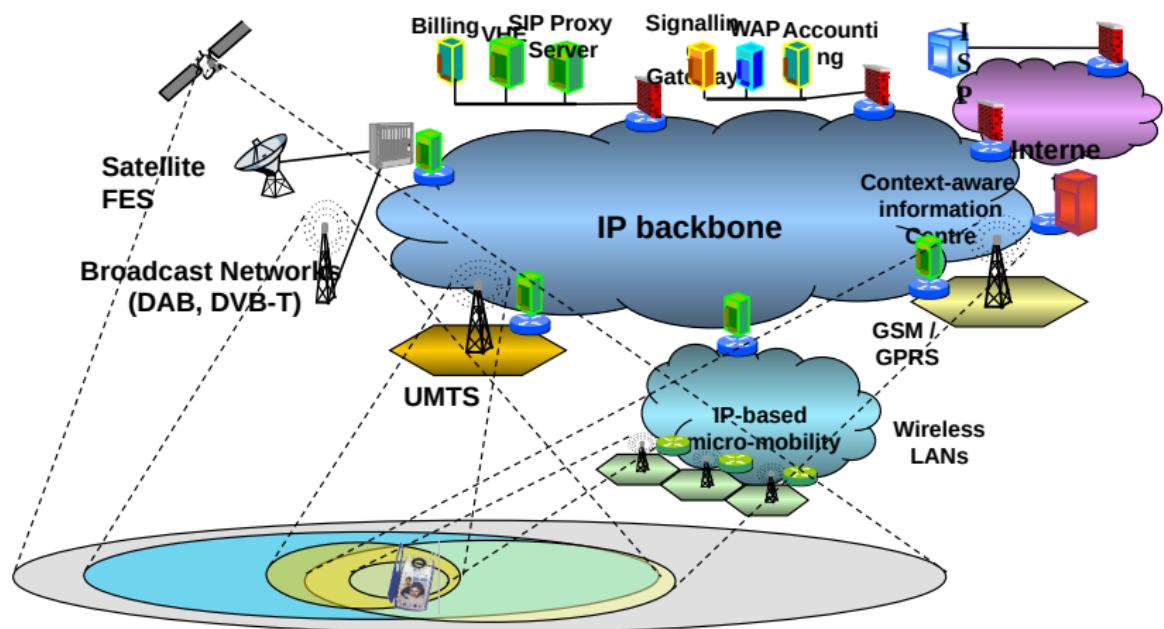
# Les réseaux sans fil, fixes ou mobiles



# Architecture GSM



# Vision tout-IP



# Norme IEEE 802.11 (Wi-Fi)

- Wi-Fi : Wireless Fidelity
- Originellement IEEE 802.11b (11 Mbps) mais il y a eu aussi 802.11a (54 Mbps).
- Maintenant 802.11g (54 Mbps) et 802.11n (600 Mbps) sont très répandus et 802.11ac (6930 Mbps) est aussi disponible.
- Rayon de 50 mètres approximativement pour chaque point d'accès selon les obstacles.



# WiMax

---

- WiMax (Worldwide Interoperability for Microwave Access) est une famille de normes techniques permettant de livrer une connectivité haute vitesse sur le dernier kilomètre;
- Haut débit;
- Grande couverture;
- Alternative à ADSL.



# Équipements WiMax



# Bluetooth

- Technologie ayant évolué des principes de conception des réseaux cellulaires (basé sur 802.11 en mode ad hoc)
- Norme de communication de courte portée (jusqu'à 10 m mais peut être étendue à 100 m)
- Fonctionne à 3.4 GHz, près de la fréquence micro-onde, dans la partie de la bande de fréquence qui ne requiert pas de licence d'opération (ISM - Industrial, Scientific and Medical)
- Effectue des sauts de fréquence rapides (1600 sauts/seconde) entre 79 fréquences de manière à éviter les interférences
- Technologie full-duplex (canal de communication dans les deux sens) en utilisant le TDD (Time Division Duplex)



# Réseaux ad hoc

- La topologie change fréquemment, nœuds entrent, sortent, bougent;
- Découverte des voisins par diffusion de message;
- Capacités réduites en mémoire, calcul et puissance
- Pas d'identificateur global
- Déployés en grand nombre ( $10^3 \dots 10^6$ )
- Réseaux de capteurs
- Exemple: Zigbee et Z-Wave pour la domotique



# La sécurité des réseaux et applications

- Trois volets:
  - Intégrité de l'information;
  - Confidentialité de l'information;
  - Disponibilité du service, coût, réputation;
- Méchanismes:
  - Authentification: garantie de l'identité du correspondant;
  - Somme de contrôle cryptographique: intégrité et non répudiation
  - Encryption: confidentialité;
  - Contrôle d'accès: usager, groupe, administrateur, rôle, délégation...
- Attaques en déni de service, virus, cheval de troie, exploitation de vulnérabilité réseau, clé USB, accès physique...



## La sécurité avec un réseau non fiable

- Un message peut être vu, intercepté, modifié, ajouté, retardé ou rejoué;
- Systèmes de clés publiques pour initier une connexion; des clés symétriques peuvent être communiquées et utilisées par la suite.
  - Paire de clés, une publique, l'autre privée;
  - Chaque utilisateur a deux paires, l'une avec chiffrement publique (déchiffrement secret) permettant à chacun d'écrire un message que seul cet utilisateur peut lire, l'autre avec déchiffrement publique (chiffrement secret) permettant à cet utilisateur d'envoyer un message que lui seul peut avoir envoyé mais tous peuvent lire.
- Avoir un message avec temps, date, numéro de séquence chiffré avec la clé publique du destinataire et la clé privée de l'envoyeur.

## La sécurité d'un système

- Sécurité physique: accès au serveur, à l'ordinateur utilisé par l'administrateur de système, au courrier contenant les logiciels à installer...
- Sécurité humaine: persuader un employé de donner un accès...
- Les vulnérabilités dans les logiciels existent;
- Mise à jour de sécurité fréquentes;
- Multiples lignes de défense, pare-feu, détection d'intrusion, vérification d'intégrité, monitoring réseau, contrôle fin des accès, vérification des log...
- Analyse des risques, plan de contingence.



## Les files d'attente (pour les paquets, requêtes...)

- On suppose que la queue a une capacité très grande et que le taux d'arrivée des requêtes n'est pas influencé par l'attente;
- Taux d'arrivée des requêtes, les requêtes se présentent aléatoirement selon un processus de Poisson:  $\lambda$ ;
- Capacité de traitement des requêtes:  $\mu$  requêtes par seconde;

Utilisation  $U$  d'un service est la fraction de temps occupé

$$U = \frac{\lambda}{\mu}$$

Nombre moyen de requêtes dans le système

$$\bar{N} = \frac{U}{1 - U}$$

# Résumé

---

- ① Introduction
- ② Historique
- ③ Les défis
- ④ Modèles de systèmes
- ⑤ Retour sur la réseautique et sécurité





# Architecture des clients pour l'infonuagique

Module 2

INF8480 Systèmes répartis et infonuagique  
Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① Répartition du travail entre le client et le nuage
- ② Les clients légers
- ③ Les applications Web
- ④ Les clients mobiles
- ⑤ Conclusion



# Architecture des clients pour l'infonuagique

① Répartition du travail entre le client et le nuage

② Les clients légers

③ Les applications Web

④ Les clients mobiles

⑤ Conclusion



## Répartition du travail

- Services offerts de manière transparente par une multitude de serveurs anonymes répartis à travers le monde.
- Le client fait l'affichage graphique simple.
- Le client fait l'affichage graphique spécialisé (décodage vidéo...).
- Le client peut exécuter certaines parties des applications offertes par le serveur.
- Le client peut stocker temporairement certaines données et applications pour des fins de performance et d'autonomie (pour opérer en mode déconnecté), sous une forme de cache.
- Le client est un ordinateur complet avec des applications et des données installées localement et mises à jour ou synchronisées seulement à la demande.



# Les terminaux alphanumériques

- Affichage de caractères avec position adressable.



## Terminaux graphiques

- Caractères alphanumériques
- Dessins vectoriels ou par matrice de pixels.



# Architecture des clients pour l'infonuagique

- ① Répartition du travail entre le client et le nuage
- ② Les clients légers
- ③ Les applications Web
- ④ Les clients mobiles
- ⑤ Conclusion



## Les clients légers

- Ordinateur de faible puissance qui exécute une application d'affichage graphique soit en mémoire morte, soit téléchargée à l'initialisation.
- Aucune gestion des ordinateurs clients qui sont tous identiques, seulement une gestion des comptes usagers.
- Terminaux X11 sous POSIX (Linux, Unix, Solaris...).
- Citrix Independent Computing Architecture (ICA).
- Microsoft Remote Desktop Protocol (RDP) et Remote Desktop Services (RDS) / Terminal Server.
- Virtual Network Computing (VNC).
- Possibilité de créer une session graphique sur le serveur, de s'y connecter avec un client, se déconnecter sans terminer la session et s'y reconnecter avec un autre client!

# Linux Terminal Server Project (LTSP)

- Ordinateurs désuets ou clients légers.
- Installation minimale de Linux localement.
- Démarrage de X11 avec fenêtre de login vers un serveur.
- Possibilité d'exécuter quelques applications localement pour permettre des services locaux ou optimiser la répartition de la charge entre le client et le serveur.
  - Décompression et affichage de vidéo.
  - Imprimante et scanner connectés localement.
  - Application pour jouer le son en parallèle avec X11.



# Les serveurs pour les applications lourdes

- Ordinateur client autonome pour les applications usuelles.
- Applications spécifiques pour lesquelles le traitement est envoyé à un serveur.
  - Calcul scientifique.
  - Rendu graphique par lancer de rayon pour un film.
  - Certains jeux 3D.
  - Reconnaissance de la voix Siri sur iPhone.
  - Calcul de chemin Google Maps sur Android.



## Les ordinateurs ou applications sans stockage

- Ordinateur sans disque avec chargement du système d'exploitation par réseau (Preboot eXecution Environment - PXE Boot) et utilisation de serveurs de fichiers.
- Ordinateur avec disque local SSD et copie temporaire des applications. Mises à jour automatiques et stockage des données sur un serveur de fichiers, par exemple le ChromeBook.
- Fureteur avec toutes les préférences sauveées sur un serveur, par exemple Chrome.



## Les applications téléchargées

- Code binaire qui doit être pour la bonne architecture et la bonne gamme de versions du système d'exploitation et des librairies.
- Code indépendant de l'architecture (e.g. bytecode Java) mais qui doit être pour la bonne gamme de versions de la machine virtuelle et des librairies.
- Applications signées par une autorité digne de confiance. Exécutées comme une application ordinaire avec les priviléges usuels de l'usager.
- Machine virtuelle avec langage sécuritaire (e.g. Java). L'application ne peut utiliser que les API fournis.
- Bac à sable fourni par le système d'exploitation. L'application ne peut faire que des appels systèmes qui sont vérifiés pour les droits d'accès avant d'être exécutés.

## La sécurité dans une machine virtuelle comme la JVM

- Langage sécuritaire sans possibilité de corruption. Pas d'arithmétique de pointeur, de pointeur non initialisé ou avec une valeur incorrecte, de débordement de vecteur...
- Interfaces de programmation restreintes avec accès vérifiés pour toutes les opérations plus dangereuses (entrées-sorties).
- Eviter la lecture de données sensibles et leur envoi à l'extérieur. Restreindre l'affichage pour éviter que l'application puisse prendre le contrôle de l'écran et simuler une autre application (e.g. accès bancaire) et tromper l'usager l'incitant à fournir des informations sensibles.
- La machine virtuelle Java fait appel à de nombreuses librairies natives et plusieurs trous de sécurité exploitables à partir du Java ont été trouvés dans ces librairies au fil du temps.

## La sécurité dans un bac à sable avec Linux

- Applications compilées qui peuvent effectuer n'importe quelle instruction non privilégiée.
- Le système d'exploitation restreint les opérations permises à une application en fonction de son code d'usager, ses groupes et ses paramètres de capacités.
- Le système impose des quotas d'utilisation des ressources du système pour éviter qu'une application puisse monopoliser les ressources.
- Avec le module Security Enhanced Linux (SELinux), ou avec AppArmor, un contrôle beaucoup plus fin des permissions d'accès est possible.
- Plusieurs démons offrent des services aux applications et vérifient les droits d'accès (notification, localisation, téléphonie...).



# Architecture des clients pour l'infonuagique

- 1 Répartition du travail entre le client et le nuage
- 2 Les clients légers
- 3 Les applications Web
- 4 Les clients mobiles
- 5 Conclusion



# Le fureteur Web comme plate-forme pour les applications

- Courriel, agenda, cartographie, édition de document, visionnement de vidéo, potins assistés par ordinateurs (PAO, par exemple Facebook).
- HTML.
- Javascript.
- Applet Java.
- Modules d'extension non normalisés comme Microsoft Silverlight, Flash, Active-X, ...



# HTML5

---

- Langage déclaratif, basé sur XML, pour décrire le contenu d'une page Web.
- Inclut <video>, <audio>, MathML et SVG (Scalable Vector Graphics).
- Langage JavaScript avec interfaces de programmation (API) pour modifier le document (Document Object Model, DOM) et pour informer le fureteur de l'état de la page (qui peut alors être inclus dans un marque-page).



## Javascript

- Langage de script qui ressemble au C et Java mais s'inspire plus sémantiquement de langages comme scheme et self.
- API pour modifier l'affichage via des changements au document avec le Document Object Model.
- Lecture des entrées (clavier et souris) pour la validation des entrées dans les formulaires, interagir avec un jeu ou d'autres applications.
- Interagir avec le fureteur (communiquer l'état pour les marque-pages).
- Communiquer en réseau pour les applications réparties, le suivi des usagers, les statistiques...
- Aucun accès à la machine locale, accès en réseau seulement au serveur d'origine de la page.
- La performance des fureteurs pour exécuter du Javascript est devenue un critère important.

## Applet Java dans le fureteur

- Le code Java arrive sous forme de bytecode prêt à s'exécuter sur le client dans la machine virtuelle Java associée au fureteur.
- Le fureteur exécute la machine virtuelle dans un processus séparé avec peu ou pas d'accès à la machine locale.
- Accès au réseau seulement vers le serveur d'origine.
- Possibilité d'application signée qui demande un accès étendu (caméra, micro, fichiers...).
- Affichage graphique avec la librairie Swing dans un cadre du fureteur ou dans une nouvelle fenêtre.
- Permet d'exécuter efficacement des applications relativement complexes et gourmandes.
- Est-ce que Java est un standard ouvert avec implémentation de référence libre?

# AJAX (Asynchronous JavaScript and XML)

- Pages avec Javascript et API comme DOM.
- Requêtes au serveur par HTTP.
- Encodage XML ou JSON.
- Permet des applications très dynamiques et évite d'attendre de manière synchrone après les requêtes au serveur.
- Exemple: afficher rapidement le début de la page et aller chercher la suite seulement si l'usager fait défiler la page vers la fin (Google images, Amazon...).
- Il est plus difficile de mettre au point ces applications.
- L'indexation est plus difficile.
- Les signets basés seulement sur la URL n'ont pas l'information sur l'état courant.



# Architecture des clients pour l'infonuagique

- 1 Répartition du travail entre le client et le nuage
- 2 Les clients légers
- 3 Les applications Web
- 4 Les clients mobiles
- 5 Conclusion



# Applications Android

- Les applications Android peuvent être écrites en Java ou directement en code binaire (C/C++ et assembleur compilés).
- Doit être pour la bonne architecture si en binaire.
- Doit être pour la bonne version du système (compatibilité vers l'avant).
- Sécurité basée sur l'exécution sous le système d'exploitation Linux avec un numéro d'usager différent pour chaque application.
- Tous les accès vers les ressources passent par un démon qui vérifie les permissions données par l'usager.
- Certaines applications malicieuses peuvent fonctionner si l'usager donne les permissions demandées au moment de l'installation.



## Applications IOS

- Les applications IOS peuvent être écrites en Objective C ou Swift et sont compilées.
- Toutes les applications doivent être signées.
- Les applications s'exécutent sur le système d'exploitation en tant que l'usager mobile et ont un répertoire maison assigné aléatoirement.
- Tout accès à de l'information ou des ressources se fait via des services (démons) qui vérifient si l'accès doit être autorisé.
- Le code est en mode lecture seulement et la pile en mode non exécutable. Les adresses de départ des différentes régions sont randomisées.
- L'exécution en arrière-plan ne se fait que via des fonctions de rappel bien contrôlées.
- Toutes les informations en mémoire permanente sont encryptées.



# Architecture des clients pour l'infonuagique

- 1 Répartition du travail entre le client et le nuage
- 2 Les clients légers
- 3 Les applications Web
- 4 Les clients mobiles
- 5 Conclusion



## Conclusion

- Beaucoup de temps était perdu à une certaine époque pour installer adéquatement un ordinateur client. Toute mise à jour ou remplacement du matériel était un cauchemar.
- Langage de haut niveau, indépendant du matériel, et librairies avec compatibilité vers le haut, permettent une grande portabilité et même des applications mobiles.
- Où est le bon niveau de virtualisation : fureteur, Java, système d'exploitation, matériel?
- Quelles applications et données doivent-elles mieux résider localement?
- Sécurité du client, du réseau et des serveurs?



# Résumé

---

- ① Répartition du travail entre le client et le nuage
- ② Les clients légers
- ③ Les applications Web
- ④ Les clients mobiles
- ⑤ Conclusion





# Processus serveurs pour l'infonuagique

Module 3

INF8480 Systèmes répartis et infonuagique

Michel Dagenais    Raphaël Beamonte

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① L'infonuagique
- ② La virtualisation
- ③ Les services pour l'infonuagique
- ④ Docker et Kubernetes
- ⑤ Conclusion



# Processus serveurs pour l'infonuagique

- 1 L'infonuagique
- 2 La virtualisation
- 3 Les services pour l'infonuagique
- 4 Docker et Kubernetes
- 5 Conclusion



# L'infonuagique, qu'est-ce?

- Distributed and Cloud Computing, from Parallel Processing to the Internet of Things, 2011: virtualization, scalability, SOA, MOM, Security, Discovery, Databases, Grid, Peer-to-peer, Overlay networks, Fault tolerance, Ubiquitous computing, Internet of things, Wireless sensor networks, Social networks...
- John Gage, 1988, Sun Microsystems: the network is the computer!
- Jacques Gélinas, 2001: avec les vserver, chaque service est dans un serveur séparé de configuration plus générique.
- 2013, Quel modèle de serveur: cat or cow, pet versus cattle.
- Découpler les services des serveurs! Eliminer les serveurs individuels au profit de parcs de serveurs qui offrent une grande économie d'échelle.



# Environnements infonuagiques

- VMWare, 1998: consolider plusieurs services sur quelques serveurs redondants grâce à la virtualisation.
- Amazon EC2, 2006: instances d'ordinateurs à louer, "Infrastructure as a service" (IaaS). Virtualisation avec Zen.
- Eucalyptus, 2008: "Elastic Utility Computing Architecture for Linking Your Programs To Useful Systems", clone du logiciel de EC2, initialement ouvert et ensuite à base ouverte.
- OpenStack, 2010: démarré par un large consortium de compagnies de haute technologie, en réponse à Eucalyptus qui n'était plus vraiment ouvert.
- Kubernetes, 2015: les conteneurs sont plus efficaces que les machines virtuelles, pour les infrastructures privées, ou par-dessus des machines virtuelles infonuagiques.

# Le marché de l'infonuagique

- Amazon demeure la référence et le leader avec environ 40% du marché. Tous reprennent ses API pour assurer une compatibilité et une transition facile.
- Microsoft a réussi à percer ce marché, avec une part d'environ 10% pour Azure, grâce à des prix agressifs (gratuit à l'essai), sa force de vente, et ses applications infonuagiquestes (Outlook, Office 365...).
- D'autres joueurs ont une présence importante comme Google, Alibaba et IBM.
- OpenStack (RackSpace) et VMWare (OVH) sont utilisés par plusieurs fournisseurs Internet pour offrir des services infonuagiquestes. Plusieurs compagnies utilisent VMWare mais sont membres du consortium appuyant OpenStack, attendant qu'il soit plus mature.
- Pour leurs systèmes internes, plusieurs compagnies utilisent Kubernetes ou des solutions équivalentes de conteneurs.

# Processus serveurs pour l'infonuagique

- 1 L'infonuagique
- 2 La virtualisation
- 3 Les services pour l'infonuagique
- 4 Docker et Kubernetes
- 5 Conclusion



# Virtualisation

- Conteneurs: plusieurs espaces de nom dans le système d'exploitation, gérés par le noyau (Linux LXC, Docker).
- Virtualisation logicielle: simulateur d'exécution (Bochs), traducteur dynamique (VMWare, Valgrind).
- Virtualisation matérielle: le processeur peut intercepter ou déléguer certaines instructions privilégiées afin de supporter efficacement la virtualisation (VMWare, KVM).
- Paravirtualisation: le système d'exploitation invité collabore en redirigeant ses requêtes vers le système hôte (Xen, VMWare, KVM).
- Hyperviseur (micro-noyau qui gère les interruptions et protections, e.g. Xen) ou système d'exploitation hôte (e.g. KVM).



# Conteneurs

- Un seul noyau avec des espaces de noms séparés par conteneur pour les PID, IPC, usagers, réseau, /proc, hostname, fichiers.
- Chaque conteneur peut rouler une version différente des librairies, applications... mais il n'y a qu'un seul noyau en exécution.
- Aucun coût additionnel en performance, sauf la mémoire non partagée par les versions différentes, si c'est le cas.
- Linux LXC (aussi V-server, OpenVZ, Docker), FreeBSD jails, Solaris containers.



## Simulateurs d'exécution

- Programme qui lit et interprète les instructions en simulant le matériel. L'hôte peut être un Intel et l'ordinateur simulé un ARM.
- Certains simulateurs peuvent aussi calculer le nombre de cycles écoulés et s'interfacer à GDB.
- Différentes techniques: interprétation une instruction à la fois, recompilation dynamique par segments, remplacement de certaines instructions et exécution directe des autres.
- BOCHS, QEMU, VMWare et VirtualBox sans support matériel, Valgrind.
- Environ 2 (remplacement de certaines instructions), 5 (recompilation dynamique) ou 50 (interprétation) fois plus lent.

# Virtualisation matérielle

- Support matériel (Intel VT, AMD V) pour intercepter ou rediriger certaines opérations.
- Assigner un périphérique à une VM (PCI passthrough), démultiplexer par VM les arrivées de paquets dans la carte réseau, déléguer la table de pages...
- Linux KVM, VMWare et VirtualBox avec support matériel.
- Entre même vitesse et 2 fois plus lent selon le degré d'E/S et d'interaction avec le système d'exploitation.



## Paravirtualisation

- Le système d'exploitation est modifié pour faire un appel efficace au système d'exploitation hôte. Pas besoin d'intercepter les opérations d'accès au matériel et d'émuler le matériel.
- Plus d'une centaine d'opérations de bas niveau du noyau Linux (lire CR0, désactiver interruptions, lire bloc, changer table de page...) sont appelées à travers la table paravirt\_ops qui pointe vers la fonction native ou virtualisée.
- Utilisé par Xen mais aussi VMWare, VirtualBox et KVM avec certains pilotes d'interface virtualisés (disque, réseau, affichage).
- Entre même vitesse et deux fois plus lent, selon le type de charge et les opérations qui sont virtualisées ou non.

# Hyperviseur

- Linux virtuel sous Windows réel ou l'inverse?
- Hyperviseur, système d'exploitation minimal qui gère les interruptions et les accès aux périphériques, pour les répartir entre les systèmes d'exploitation des machines virtuelles.
- Xen. Linux domaine 0 qui parle aux périphériques et Linux domaines 1, 2... qui sont les machines virtuelles invitées dont les requêtes sont passées par Xen au domaine 0.
- Xen peut maintenant accepter des invités Windows grâce au support de virtualisation matériel.
- Hyperviseur: solution élégante ou un OS de plus inutilement?



## Bénéfices de la virtualisation

- Image logicielle isolée du matériel, utile lorsque les licences sont attachées au matériel ou pour portabilité.
- Possibilité de cohabitation entre plusieurs systèmes d'exploitation ou versions, plutôt que double amorçage.
- Isolation des services à des fins de sécurité ou de gestion. Plusieurs serveurs virtuels de différents groupes peuvent coexister sur le même serveur physique.
- Modularisation des services: démarrer les serveurs virtuels voulus: base de donnée, courriel, Web...



## Coût de la virtualisation

- Certaines instructions causent des interruptions et sont émulées; moins avec le support matériel.
- Accès indirect aux périphériques; moins avec la paravirtualisation ou la virtualisation des I/O (IOMMU).
- Changements de contexte plus nombreux, application, système d'exploitation invité, hyperviseur; moins avec la délégation de tables de pages aux invités.
- Préallocation de la mémoire à chaque machine virtuelle (Xen), n'est pas toujours requis (Xen balloon, KVM).
- Surcoût d'avoir plusieurs copies en mémoire du noyau et des exécutables courants (libc, bash...); moins avec Kernel Samepage Merging.



# Virtualisation du réseau

- Réseaux et commutateurs virtuels à l'intérieur d'un noeud pour connecter les noeuds virtuels; Linux TUN/TAP (network tunnel, network tap).
- VLAN: réseau local virtuel séparé du reste du réseau local (étiquette ajoutée à chaque paquet Ethernet, gestion des diffusions générales sur le VLAN).
- VPN/VPLS: connexions multi-point virtuelles privées par-dessus le réseau public.
- Le résultat est un réseau dédié virtuel (overlay network); latency, bande passante, qualité de service...



# Migration

- Pour équilibrer la charge ou libérer le matériel qui requiert un entretien.
- Déplacer une image en exécution d'une machine virtuelle à l'autre; revient à migrer une machine virtuelle d'un ordinateur physique à un autre de manière transparente.
- Contraintes de même réseau local, mêmes fichiers accessibles, pas de 64 vers 32 bits, matériel virtuel identique.
- Copier toutes les pages de l'image en traçant celles qui sont remodifiées dans l'intervalle. Faire une seconde et possiblement troisième passe. Tout suspendre, copier les pages encore modifiées et poursuivre sur l'autre ordinateur.



# Processus serveurs pour l'infonuagique

- 1 L'infonuagique
- 2 La virtualisation
- 3 Les services pour l'infonuagique
- 4 Docker et Kubernetes
- 5 Conclusion



# Serveurs virtuels/instances

- Plusieurs catégories de noeuds, et plusieurs “tailles” (CPU, mémoire, GPU, ...)
- Au centre de plusieurs autres services qui travaillent en corrélation:
  - services de stockage;
  - services de bases de données;
  - répartition de requêtes réseaux;
  - service de mise à l'échelle (allocation +/- flexible de noeuds supplémentaires).
- Plusieurs zones de disponibilité.



## Images / modèles

- Fournit des images de systèmes utilisables directement, ou après avoir appliqué des modifications dessus (cliché d'un serveur).
- Possibilité de créer des propres images Linux et de les importer; attention à utiliser les bonnes options selon le service utilisé (pour la paravirtualisation et les pilotes, par exemple).
- Le format accepté pour les images diffère selon le service, mais les formats communs comme vmdk (VMWare), vhd (Hyper-V) et raw (KVM) sont souvent acceptés.



## Enchères de calcul (services commerciaux)

- Il est possible de spécifier un prix de lancement d'instances pour un gros calcul à effectuer à bas prix.
- Lorsque le prix est sous le seuil spécifié, les instances demandées sont démarrées.
- Le prix fluctue sous le prix spécifié. Si le prix remonte, les instances peuvent être arrêtées.
- Requête unique ou persistente.
- Le prix “spot” est souvent moins de 30% du prix régulier.
- Modèle pour utiliser les instances qui vont et viennent (nombre variable de noeuds de travail).



N/A



# Utilisation des instances

- Interface Web pour commander les instances, ligne de commande ou API.
- Définition de règles d'accès pour le groupe de sécurité contenant l'instance.
- Sélection d'une image, d'une taille d'instance, du nombre d'instances, de la zone de disponibilité, et du groupe de sécurité, puis démarrage.
- Des métadonnées sont disponibles pour chaque instance (paramètres, adresses, numéro d'instance...).
- Connexion par SSH à l'instance.



# Services de stockage

**Instance storage:** stockage pour la durée d'une instance, avec l'image comme contenu initial de la partition racine.

**Block Storage:** partition de disque pour stockage permanent qui peut être attachée à une instance à la fois.

**Object Storage:** stockage permanent, extensible, accessible de plusieurs instances en lecture et écriture.

**Shared file storage:** partition montée via le réseau pour stockage permanent, qui peut être attachée à plusieurs instances à la fois.

Block: EBS  
Object: S3  
Shared: EFS



Block: Cinder  
Object: Swift  
Shared: Manila



Block: Page BLOB  
Object: Block BLOB  
Shared: Azure File Storage



# Services de bases de données

- Bases de données relationnelles: généralement MySQL, Oracle ou PostgreSQL.
  - Pour les services commerciaux:
    - Différentes tailles possibles, payées à l'heure et au transfert.
    - Sauvegardes et mises à jour intégrées.
- Bases de données NoSQL:
  - Les services commerciaux ont souvent des solutions propriétaires qui acceptent ou non les requêtes de type SQL;
  - Les services à source ouvert utilisent des solutions à source ouvert existantes (Cassandra, MongoDB, ...)
- Simplifie la conception de systèmes avec répartiteur de charge, instances de service élastiques sans données, et base de données centrale.

 **SQL: RDS** (up to 16TB)  
**NoSQL:** DynamoDB

 **openstack.**  
**SQL:** Trove  
**NoSQL:** Trove

 **SQL:** SQL Database (up to 4TB)  
**NoSQL:** CosmosDB

## Répartiteur de charge

- Surveille un nom de noeud (e.g. www.macompagnie.com) et un numéro de port (e.g. 80).
- Répartit les requêtes reçues sur ce port entre les instances enregistrées pour le servir.
- Répartition entre les instances dans différentes zones de disponibilité.
- Maintien de métriques sur le niveau de service dans le répartiteur de charge: latence, nombre de requêtes, nombre d'instances en santé, etc.
- Envoi de la prochaine requête à l'instance répondant aux critères de répartition (architecture, zone géographique, mémoire vive, nombre de coeurs) la moins chargée (charge actuelle, nombre d'instances, puissance du noeud...)
- Arrêt d'envoi de requêtes aux instances non fonctionnelles.



**ELB (Elastic  
Load Balancer)**



## Service de mise à l'échelle/nuage élastique

- Maintien d'un niveau de service en surveillant le nombre d'instances valides et le taux d'utilisation du CPU.
- Redémarre les instances non valides.
- Démarre de nouvelles instances si le taux d'utilisation du CPU dépasse un certain seuil.
- Arrête des instances si le taux d'utilisation du CPU descend sous un certain seuil.
- Les services commerciaux limitent souvent le nombre d'opérations de réduction pouvant être faites, il faut prendre plus de temps avant d'agir (e.g. CPU à 80% pendant 10mn = ajout d'une instance, CPU à 5% pendant 4h = retrait de X instances).



# Surveillance

- Métriques mesurées régulièrement et conservées pendant un temps donné à propos des instances et volumes de stockage en blocs; certains services permettent de récupérer plus d'informations sur plus de systèmes.
- Instance: taux d'utilisation du CPU, accès en entrée et en sortie (opérations et octets), nombre d'octets envoyés et reçus par réseau.
- Block Storage: accès en entrée et en sortie (opérations et octets), temps de lecture et d'écriture, temps morts, longueur moyenne de la file.
- Émission de notifications/alarmes lorsque certains évènements se produisent.

**Cloudwatch**  
(mesures aux 1 ou 5mn,  
conservées 2 semaines)

**Synaps,**  
**Telemetry**  
(Ceilometer)

**Azure Monitor**

## Gestion d'identité

- Répertoire des usagers.
- Authentification par mot de passe ou par jetons (et parfois de l'authentification en plusieurs temps).
- Permissions, rôles et politiques d'accès, qui permettent notamment un contrôle d'accès détaillé aux ressources.
- Fournit généralement une intégration du répertoire de l'organisation, via Windows Active Directory ou LDAP par exemple.
- Peut fournir une liste des services disponibles pour un utilisateur authentifié.



# NUAGE PRIVÉ DANS LE NUAGE PUBLIC

- Réseau virtuel.
- Adresses IP choisies par l'utilisateur.
- Tables de routage.
- Couche de sécurité supplémentaire avec listes de contrôle des accès.
- Filtrage des sorties en plus des entrées pour chaque instance.
- Possibilité de limiter l'accès de l'Internet à une passerelle IPSec.



## Orchestration de travaux

- Format déclaratif qui peut facilement être mis dans un système de gestion du code source avec version.
- Outils pour activer la configuration définie par la recette.
- Définition des paramètres à spécifier.
- Déclaration de la configuration en utilisant les paramètres qui devront être fournis par l'usager.
- Utile pour organiser des tâches sur l'ensemble du système, faire des actions sur une partie du parc de machines virtuelles, etc.



# Files de messages

- Systèmes de files de messages, avec publication/abonnement (publish/subscribe) ou notification.
- Pour par exemple: distribution de tâches à des noeuds, collecte de données, envoi de commandes à plusieurs destinataires, réception de réponses de multiples destinataires, outils de synchronisation en continu...
- Pour un très grand nombre de files, messages, destinataires...



## Mégadonnées (*Big Data*)

- Services pour facilement déployer et mettre à l'échelle des systèmes de traitement de données comme Hadoop, Apache Spark, HBase, Presto, Hive, Flink. . .
- Réutilise en général les autres services disponibles (images, noeuds, stockage, authentification...), mais sans être gérés directement par l'utilisateur.
- Les services commerciaux rendent “invisible” cette utilisation des autres services, le service de mégadonnées étant facturé à part.
- Permet de gérer des cas communs d'utilisation des mégadonnées: analyse de fichiers journaux, indexage de sites web, transformation de données, apprentissage machine, analyses financières, simulation scientifique. . .



# Les services pour l'infonuagique: Discussion

## Utilisation des services commerciaux (AWS, Azure, ..)

- Permet de mettre sur pied une grappe ou un service Web de grande envergure très rapidement avec un coût initial presque nul.
- Evite les coûts de locaux ou d'équipes d'entretien.
- Attention, les frais d'utilisation et de transferts s'accumulent.
- Plus cher qu'une solution maison si on possède une très grande grappe, gérée efficacement et pleinement utilisée.
- Excellente solution pour les petites ou moyennes entreprises, la capacité excédentaire ou ponctuelle, comme plan de contingence, pour un démarrage rapide...

# Les services pour l'infonuagique: Discussion

Qu'offre OpenStack pour un nuage privé vs AWS ou Azure?

- Convergence d'un grand nombre de joueurs autour de quelques technologies clé (Linux, KVM, API de AWS EC2, Réseaux virtuels).
- Infrastructure infonuagique entièrement libre qui offre des fonctionnalités semblables à ce qui a été mis de l'avant par Amazon/Azure.
- Progrès très rapide. Grandes différences d'une année à l'autre.
- L'essentiel de la fonctionnalité requise est maintenant disponible.



# Processus serveurs pour l'infonuagique

- 1 L'infonuagique
- 2 La virtualisation
- 3 Les services pour l'infonuagique
- 4 Docker et Kubernetes
- 5 Conclusion



# Docker

---

- C'est une image, c'est un conteneur, c'est une compagnie?
- Format d'image pour exécuter un conteneur sur Linux.
- Environnement d'exécution pour rouler une image sur Linux... ou sur d'autres systèmes comme Windows.
- Compagnie qui offre des outils de haut niveau pour gérer les conteneurs, au-delà des fonctions de base de Docker.
- Le format d'image et l'environnement d'exécution ont été transférés au Open Container Initiative de la Linux Foundation qui regroupe de nombreuses entreprises.



# Dockerfile

- Déclaration et recette pour créer et rouler une image.

```
FROM ubuntu:latest
RUN apt-get update
RUN apt-get install -y wget
RUN apt-get install -y build-essential tcl8.5
RUN wget http://download.redis.io/releases/redis.tgz
RUN tar xzf redis.tgz
RUN cd redis-stable && make && make install
RUN ./redis-stable/utils/install_server.sh
EXPOSE 6379
ENTRYPOINT ["redis-server"]
```



# Kubernetes

---

- Contribué par Google au Cloud Native Computing Foundation de la Linux Foundation en 2015.
- Orchestration de conteneurs typiquement avec Docker.
- Rapidement supporté par les fournisseurs d'infonuagique et très populaire pour les nuages internes aussi.
- Peut être déployé par-dessus des noeuds natifs ou des machines virtuelles.
- Très bonne mise à l'échelle, si c'est assez bon pour Google...



## Kubernetes nodes (minions)

- Pod: groupe de conteneurs qui s'exécutent sur un même noeud pour offrir un service.
- Noeud: machine virtuelle ou noeud physique disponible pour rouler des conteneurs, qui exécute:
  - Docker runtime: engin pour exécuter les conteneurs;
  - Kubelet daemon: processus pour gérer, arrêter ou démarrer, les conteneurs sur le noeud;
  - Kube-proxy: processus pour gérer les communications, qui redirige les requêtes au bon conteneur;
  - Cadvisor: agent qui collecte diverses métriques qui peuvent être utilisées pour le monitoring ou pour gérer les pannes et la mise à l'échelle.



# Kubernetes master

- Control plane: orchestration des conteneurs, typiquement avec redondance, sur les noeuds à l'aide de:
  - Etcd: base de donnée clé-valeur, répartie et persistente, avec notification de changement, représentant la configuration désirée;
  - API server: reçoit les requêtes REST et accède etcd;
  - Ordonnanceur: choisit quel “pod” (groupe de conteneur) roule sur quel noeud.
  - Controller manager: collection de modules de commande qui gèrent l'orchestration des conteneurs pour la réPLICATION, la mise à l'échelle...



# Kubernetes service répliqué

```
$ kubectl create -f svc.yaml  
  
# Répartiteur pour le service  
apiVersion: v1  
kind: Service  
metadata:  
  name: simpleservice  
spec:  
  ports:  
    - port: 80  
      targetPort: 9876  
  selector:  
    app: sise
```

```
$ kubectl create -f rc.yaml  
  
# Conteneurs répliqués  
apiVersion: v1  
kind: ReplicationController  
metadata:  
  name: rcsise  
spec:  
  replicas: 2  
  selector:  
    app: sise  
  template:  
    metadata:  
      name: somename  
      labels:  
        app: sise  
    spec:  
      containers:  
        - name: sise  
          image: img/serv:0.5.0  
          ports:  
            - containerPort: 9876
```

## Kubernetes : Discussion

- Essor très rapide car technologie mature qui répond à un besoin très présent.
- Approche typique de Google avec mécanismes simples mais puissants qui se mettent bien à l'échelle.
- Pourquoi avoir la migration de conteneur lorsqu'on a déjà la tolérance aux pannes.
- Bon pour les déploiements où on contrôle bien l'ensemble de l'application car plus efficace mais moins transparent que les VM.
- D'autres outils similaires ont vu le jour comme Docker Swarm ou Apache Mesos
- Peut être déployé via des services pour l'infonuagique:



# Processus serveurs pour l'infonuagique

- 1 L'infonuagique
- 2 La virtualisation
- 3 Les services pour l'infonuagique
- 4 Docker et Kubernetes
- 5 Conclusion



# Conclusion

---

- Les ordinateurs, comme les voitures, deviennent interchangeables; un signe de maturité.
- Le temps où un technicien s'occupait de 1 à 10 ordinateurs est révolu.
- Le gouvernement américain veut réduire le nombre de ses centres de données de plus de 1200 (sur environ 3000).
- Les ventes de serveurs sont en mutation.



# Résumé

---

- ① L'infonuagique
- ② La virtualisation
- ③ Les services pour l'infonuagique
- ④ Docker et Kubernetes
- ⑤ Conclusion





# Communication dans les systèmes répartis

Module 4

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

① Modèles de communication

② Modèle requête-réponse RPC

③ Communication par messages



# Communication dans les systèmes répartis

- 1 Modèles de communication
- 2 Modèle requête-réponse RPC
- 3 Communication par messages



## Avec ou sans connexion

- Sans connexion (UDP):
  - Send ([destinataire], message);
  - Receive ([émetteur], file de stockage);
- Avec connexion (TCP):
  - Connect (destinataire, émetteur);
  - Read, Write...
  - Disconnect (identificateur de connexion);



## Synchrone ou asynchrone:

- Le send et receive sont bloquants; les processus se synchronisent à chaque message. Send attend l'accusé de réception!?!?
- Le send est non-bloquant, si la file du système d'exploitation n'est pas pleine, tandis que receive est généralement bloquant. Si les paquets arrivent trop vite et la file du système d'exploitation est pleine, un message de ralentir est envoyé et les paquets en surplus peuvent être ignorés.
- Le receive aussi est non-bloquant, le processus vérifie de temps en temps, peut demander un signal, ou utilise un appel système de type select ou epoll.



## Un ou plusieurs destinataires:

- Avec un bus, on peut envoyer un message à n destinataires sur le bus en 1 unité de temps au lieu de n unités de temps pour des messages séparés.
- Les protocoles Ethernet et IP supportent la diffusion générale (broadcast) ainsi que la multidiffusion (multicast).
- Gestion des membres d'un groupe de multi-diffusion plus complexe s'ils ne sont pas tous sur le réseau local.
- Messages sans connexion non fiables (UDP), ou séquences de messages fiables (Pragmatic General Multicast, PGM) avec accusés de réception négatifs.



## Messages de groupe par multidiffusion

---

- Tolérance aux fautes basées sur des services répliqués sur plusieurs serveurs.
- Amélioration des performances par des services et données répliqués sur plusieurs serveurs.
- Découvertes de services dans un environnement de réseautage spontané.
- Economie de bande passante par multidiffusion des notifications d'événements, vidéo ou fichiers identiques demandés par de nombreux clients.



## Messages de groupe

- Envoi d'un message aux membres d'un groupe.
- Sur réception, le message est remis au processus destinataire.
- Communication non fiable, un seul message UDP est envoyé, par exemple pour annoncer une adresse Ethernet versus IP.
- Communication fiable, avec chaque message livré au moins une fois (valide) ou au plus une fois (intègre)
- Communication atomique, livré à tous ou à aucun.
- Communication ordonnancée: totalement, causalement, par origine.



## Messages de groupe atomiques

- Tous le reçoivent ou personne ne le reçoit.
- Envoi à tous du message non-confirmé et demande d'accusé de réception. Retransmission au besoin si l'accusé de réception ne vient pas.
- Si tous ont reçu, message de confirmation pour rendre le message maintenant confirmé disponible aux applications.
- Si des accusés de réception manquent après un certain temps, annuler le message.
- Si un client ne reçoit pas de confirmation, il demande l'état à l'envoyeur puis au besoin vérifie avec un autre ordinateur qui peut prendre la relève.



## Messages de groupe totalement ordonnancés

- Chaque message a un numéro de séquence unique, par exemple fourni par un processus séquenceur.
- Chaque récipiendaire attend d'avoir reçu les messages précédents avant de livrer un message à ses applications.
- Un récipiendaire peut devoir laisser tomber des messages plus récents s'il en a trop en attente.
- Un envoyer peut bloquer s'il a trop de messages en attente d'accusés de réception.



## Messages de groupe causalément ordonnancés

- Les messages en provenance d'un même ordinateur arrivent dans leur ordre d'envoi à chaque récipiendaire: numéro de séquence propre à chaque membre du groupe qui envoie des messages.
- Les messages reçus en provenance de plus d'un ordinateur arrivent en ordre causal: chaque processus maintient un vecteur de numéros de séquence de messages de groupe vus venant de chaque processus. Un message avec un certain vecteur n'est pas délivré avant que tous les messages de chaque processus n'aient été reçus, jusqu'au numéro dans le vecteur pour ce processus.



# Communication dans les systèmes répartis

1 Modèles de communication

2 Modèle requête-réponse RPC

3 Communication par messages



## RPC versus appel de fonction

- Message de requête et de réponse; écritures et lectures réseau explicites ou appel de fonction distante (RPC).
- Paramètre d'entrée, de sortie ou les deux.
- Pas de passage de paramètre par pointeur.
- Pas de variable globale accessible à tous.
- Doit spécifier comment sérialiser les structures spéciales qui utilisent des pointeurs (chaîne de caractère, vecteur, liste, arbre...) par un langage de définition d'interface ou des attributs sur les déclarations.
- Si l'appelant et l'appelé sont programmés dans différents langages, il faut adapter les différences (e.g. ramasse-miette, exceptions...).
- Arguments supplémentaires: serveur à accéder, code d'erreur pour la requête réseau.

## Catégories et historique

- Protocoles simples : SMTP (1982), HTTP (1991), Remote GDB...
- RPC basés sur un langage d'interface (multi-langage): Sun RPC (1984), ANSA (1985), MIG (1985), CORBA (1991), gRPC (2015).
- RPC basés sur un langage spécifique (meilleure intégration): Cedar (1981), Argus, Modula-3 (1993), Java (1995).
- RPC basé sur une machine virtuelle multi-langage (Common Language Runtime) C# (2000).



# Langage de définition des interfaces (IDL)

- Déclaration des procédures (nom, et liste et type) des arguments).
- Déclaration des types et des exceptions.
- Générateur de code à partir du IDL pour la librairie client (proxy), et la librairie serveur (squelette), dans le langage de programmation désiré.
- Le IDL et le générateur de code ne sont pas requis si la réflexivité permet d'obtenir cette information à l'exécution.



## Sémantique des appels

- Mécanismes: envoi simple, accusé de réception, retransmission, filtrage des requêtes dupliquées, mémorisation des réponses.
- **Peut-être:** en l'absence de réponse on ne peut savoir si la requête ou la réponse fut perdue.
- **Au moins une fois:** retransmission sans filtrage des requêtes jusqu'à obtention d'une première réponse.
- **Au plus une fois:** avec retransmission, filtrage des requêtes, et mémorisation du résultat, la procédure est exécutée exactement une fois si une réponse est obtenue et au plus une fois en l'absence de réponse.
- **Dernière fois:** pareil que pour "Au moins une fois", mais le client n'accepte la réponse que si elle correspond au dernier appel réalisé (utilise un identifiant)

## Sun RPC

*External calls ??*

- Définition d'interface et de types en XDR.
- Procédures avec un argument (entrée), une valeur de retour (sortie), et possiblement un numéro de version.
- rpcgen produit la librairie client (proxy/client stub) et le squelette (server stub) du programme serveur.
- Compléter le squelette du serveur avec l'implantation.
- Initialiser la connection dans le client et utiliser les fonctions proxy en vérifiant les erreurs.



## Sun RPC

---

- Le DNS effectue la localisation du serveur.
- Portmap effectue la localisation du service.
- Les appels sans valeur de retour peuvent être accumulés et envoyés d'un coup pour plus de performance.
- Authentification.
- Sémantique au moins une fois.
- Sur UDP, chaque requête ou réponse est limitée à 8Koctets.



## Exemple de fichier XDR

```
const MAX=1000;
typedef int FileId;
typedef int FilePointer;
typedef int Length;

struct data {
    int length;
    char buffer[MAX];
};

struct writeargs {
    FileId f;
    FilePointer position;
    Data data;
};

struct readargs {
    FileId f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
    version VERSION {
        void WRITE(writeargs)=1;
        data READ(readargs)=2
        }=2;
        }=9999;
```



# Serveur Sun RPC

```
// Serveur

#include <stdio.h>
#include <rpc/rpc.h>
#include "FileReadWrite.h"

void *write_2(writeargs *a) {
    /* supply implementation */
}

Data *read_2(readargs *a) {
    /* supply implementation */
}
```



## Client Sun RPC

```
// Client

#include <stdio.h>
#include <rpc/rpc.h>
#include "FileReadWrite.h"

main(int argc, char *argv[]) {
    CLIENT cl;
    struct data *d;
    struct readargs r;

    cl = clnt_create("server.polymtl.ca", FILEREADWRITE, VERSION,
                     "tcp");
    if(cl == NULL) {...}
    d = READ(&r,cl);
    if(d == NULL) {...}
}
```

# CORBA

---

- Langage de définition d'interface avec constantes, structures, méthodes, et exceptions.
- Générateur de proxy et de squelette multi-langage.
- Possibilité d'appels asynchrones en l'absence de valeur de retour.
- Mécanisme pour la découverte et l'invocation dynamique de procédures.
- Divers services de notification...
- Semblable à Sun RPC mais plus sophistiqué et objet.



# La norme CORBA

- Common Object Request Broker Architecture.
- L'OMG (Object management group) maintient la norme CORBA.
- C'est une architecture et infrastructure ouverte, indépendante du constructeur.
- CORBA n'est pas un Système Distribué mais sa spécification:
  - spécifications principale de plus de 700 pages.
  - Plus 1200 pages pour spécifier les services naviguant autour
  - 1991: v1: standardisation de IDL et OMA
  - 1996: v2: spécifications plus robustes, IIOP, POA, DynAny.
  - 2002: v3: ajout du component model, QoS (asynchronisme, tolérance de panne, RT Corba- Corba temps réel).

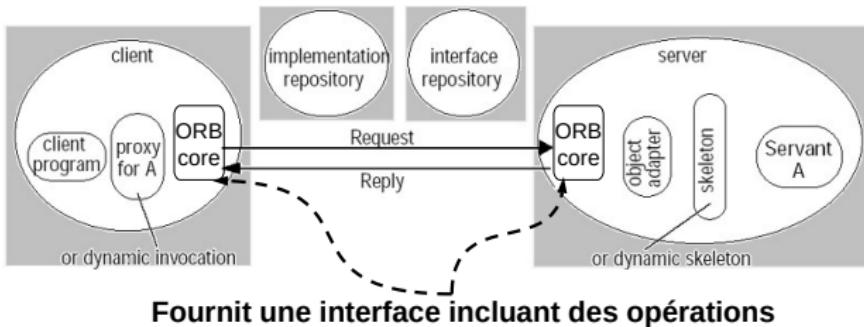


# Composantes de CORBA

- Object Request Broker (ORB)
  - Initialisation et finalisation.
  - Conversion de références aux objets.
  - Envoi de requêtes.
- Portable Object Adapter (POA)
  - Activation des objets.
  - Répartition des requêtes vers les objets via les squelettes.
  - Gestion des adresses réseau.
  - Portable: fonctionne avec des implantations CORBA différentes.



# Composantes de CORBA

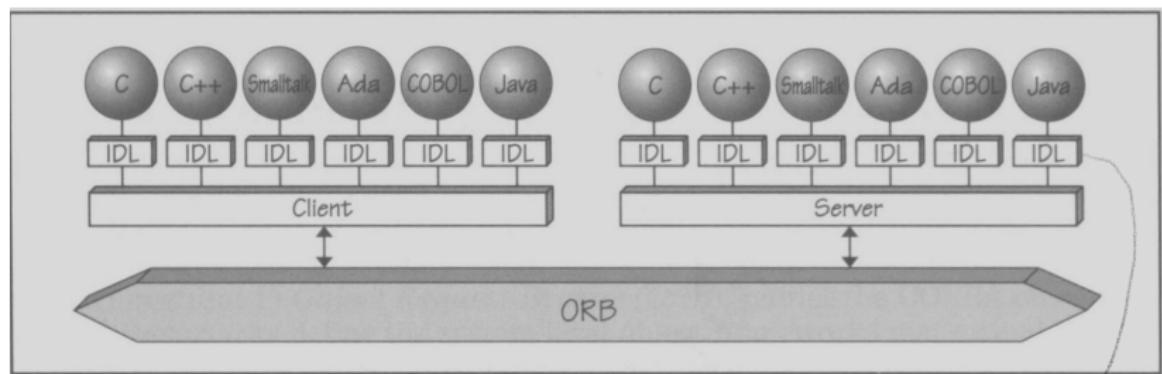


# Langage de définition d'interface

- Module: définit un espace de nom.
- Interface: liste de méthodes, attributs et déclarations de types ou exceptions. Héritage simple ou multiple.
- Méthode: nom de fonction, arguments in/out/inout, valeur de retour, exceptions. Attribut oneway pour invocation asynchrone.
- Types: short, long, unsigned short, unsigned long, float, double, char, boolean, octet, any, struct, enum, union (tagged), array, string, sequence, object.
- Exception: peut contenir des variables.
- Attribut: variable de l'interface. Peut être readonly.



# Le concept de CORBA



Interface Definition Language

## Référence à un objet

- IOR: Interoperable Object Reference.
- Format: identificateur du serveur, IIOP, hostname, port, nom de l'adapteur, nom de l'objet.
- IIOP: protocole particulier, interoperable et basé sur TCP/IP.
- IOR transitoire ou persistent.
- Plusieurs champs serveur/port sont possibles pour les services avec réPLICATION.
- Un serveur peut répondre avec une redirection.



## Support pour différents langages

---

- Chaque langage doit avoir son générateur idl et son interface de programmation CORBA.
- En C, un argument est utilisé pour vérifier les exceptions.
- En Java les struct, enum, et unions sont implantés avec des classes.
- En C++ la relation entre les constructeurs/destructeurs et la gestion de la mémoire est subtile.



## Services CORBA

- Noms: service hiérarchique (contexte, liste de (nom, contexte ou objet)). Part du contexte initial racine.
- Canal d'événement. Fournisseurs et clients qui peuvent chacun être appelant ou appelé (push/pull). Plusieurs fournisseurs et clients peuvent se connecter au même canal. Un canal peut être un fournisseur, ou un client pour un autre.
- Notification: service enrichi pour un canal d'événements. Les clients peuvent spécifier les événements d'intérêt et interroger les types d'événements offerts. Les fournisseurs peuvent spécifier les types qu'ils offrent et savoir lesquels sont d'intérêt pour un ou plusieurs clients.
- Sécurité: authentification de l'envoyeur et de la réponse, journal, liste de contrôle des accès.
- Transactions: transactions et transactions imbriquées (begin/commit/abort).
- Persistence.



# CORBA Common Data Representation (CDR)

- Permet de représenter tous les types de données qui peuvent être utilisées comme arguments ou valeurs de retour dans un appel à distance CORBA.
- Il existe 15 types primitifs: Short (16bit), long(32bit), unsigned short, unsigned long, float, char, ...
- CDR offre la possibilité d'avoir des *constructed types* qui sont des types construits à partir de types primitifs.
- Le type d'un item de données n'est pas donné avec sa représentation dans le message: l'émetteur et le récipiendaire du message connaissent l'ordre et le type des données du message.



## Exemple du CDR

```
Struct Person {  
    string name;  
    string place;  
    long year;  
};
```

index	contenu 4 octets
0-3	5
4-7	"Smit"
8-11	"h__"
12-15	6
16-19	"Lond"
20-23	"on__"
24-27	1934



## Exemple CORBA C: interface en IDL

```
interface DocumentServer
{
    void getServerInfo(out string version, out string date);
};
```



## Exemple CORBA C: initialisation du client

```
#include "stdio.h"
#include "orb/orbit.h"
#include "DocumentServer.h"

DocumentServer server;

int main(int argc, char *argv[]) {
    CORBA_Environment ev; CORBA_ORB orb;
    CORBA_char *version, *date;
    FILE * ifp; char * ior; char filebuffer[1024];

    CORBA_exception_init(&ev);
    orb = CORBA_ORB_init(&argc, argv, "orbit-local-orb", &ev);
    /* Lecture dans un fichier de l'id du serveur à accéder. */
    ifp = fopen("doc.ior", "r");
    if( ifp == NULL ) { g_error("No doc.ior file!"); exit(-1);}
    fgets(filebuffer,1024,ifp);
    ior = g_strdup(filebuffer);
    fclose(ifp);

    server = CORBA_ORB_string_to_object(orb, ior, &ev);
    if (!server) { printf("Cannot bind to %s\n", ior); return 1;}
```

## Exemple CORBA C: appel au serveur par le client

```
DocumentServer_getServerInfo(server, &version, &date, &ev);

/* Vérification du code d'erreur. */
if(ev._major != CORBA_NO_EXCEPTION) {
    printf("we got exception %d from echoString!\n", ev._major); return 1;
} else {
    printf("Version: %s, Date %s\n", version, date);
    CORBA_free(date);
    CORBA_free(version);
}
CORBA_Object_release(server, &ev);
CORBA_Object_release((CORBA_Object)orb, &ev);
return 0;
}
```



## Exemple CORBA C: initialisation du serveur

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "signal.h"
#include "orb/orbit.h"
#include "DocumentServer.h"

DocumentServer server = CORBA_OBJECT_NIL;

static void do_getServerInfo(PortableServer_Servant servant,
    CORBA_char **version, CORBA_char **date, CORBA_Environment *ev);

/* Initialisation des objets CORBA servant à décrire le service. */
PortableServer_ServantBase__epv base_epv = {NULL,NULL,NULL};
POA_DocumentServer__epv documentServer_epv = { NULL, do_getServerInfo };
POA_DocumentServer__vepv poa_documentServer_vepv =
    { &base_epv, &documentServer_epv };
POA_DocumentServer poa_documentServer_servant =
    { NULL, &poa_documentServer_vepv };
```



## Exemple CORBA C: initialisation du serveur

```
int main (int argc, char *argv[])
{ PortableServer_ObjectId objid =
    {0, sizeof("myDocumentServer"), "myDocumentServer"};
PortableServer_POA poa;
CORBA_Environment ev; char *retval; CORBA_ORB orb; FILE * ofp;
/* Sortir proprement en cas d'interruption du programme */
signal(SIGINT, exit); signal(SIGTERM, exit);

/* Initialisation du service CORBA */
CORBA_exception_init(&ev);
orb = CORBA_ORB_init(&argc, argv, "orbit-local-orb", &ev);

/* Initialisation du service de document */
POA_DocumentServer__init(&poa_documentServer_servant, &ev);

/* Contacter le localisateur d'objet pour lui donner une référence*/
poa = (PortableServer_POA)CORBA_ORB_resolve_initial_references(orb,
    "RootPOA", &ev);
PortableServer_POAManager_activate(
    PortableServer_POA__get_the_POAManager(poa, &ev), &ev);
PortableServer_POA_activate_object_with_id(poa,&objid,
    &poa_documentServer_servant, &ev);
```

## Exemple CORBA C: le serveur exporte l'objet

```
/* Obtenir une référence à notre service et l'écrire*/
server = PortableServer_POA_servant_to_reference(poa,
    &poa_documentServer_servant, &ev);
if(!server){printf("Cannot get objref\n"); return 1;}
retval = CORBA_ORB_object_to_string(orb, server, &ev);

/* Ecrire la référence dans un fichier. */
ofp = fopen("doc.ior","w");
fprintf(ofp,"%s", retval); fclose(ofp);
CORBA_free(retval);

/* Impression d'un message et attente des requêtes. */

fprintf(stdout,"En attente de requêtes...\\n"); fflush(stdout);
CORBA_ORB_run(orb, &ev);
return 0;
}
```



## Exemple CORBA C: la fonction de service

```
/* Implantation de la fonction offerte en service CORBA */

static void
do_getServerInfo(PortableServer_Servant servant,
                  CORBA_char **version, CORBA_char **date,
                  CORBA_Environment *ev)
{
    (*version) = CORBA_string_dup("0.99");
    (*date) = CORBA_string_dup("24 janvier 2000");
    return;
}
```



# Exemple CORBA Java: interface en IDL

```
struct Rectangle{  
    long width; long height;  
    long x; long y;  
};  
struct GraphicalObject {  
    string type; boolean isFilled;  
    Rectangle enclosing;  
};  
interface Shape {  
    long getVersion() ;  
    GraphicalObject getAllState() ;  
};  
typedef sequence <Shape, 100> All;  
  
interface ShapeList {  
    exception FullException {};  
    Shape newShape(in GraphicalObject g) raises (FullException);  
    All allShapes();  
    long getVersion();  
};
```



## Exemple CORBA Java: initialisation du serveur

```
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
import java.util.Properties;

public class ShapeListServer
{
    public static void main(String args[]) {
        try{
            // create and initialize the ORB
            Properties props = new Properties();
            props.put("org.omg.CORBA.ORBInitialPort", "1500");
            ORB orb = ORB.init(args, props);

            // create servant and register it with the ORB
            ShapeListServant shapeRef = new ShapeListServant(orb);
            orb.connect(shapeRef);
            // get the root naming context
            org.omg.CORBA.Object objRef =
            orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(objRef);
```



## Exemple CORBA Java: le serveur exporte l'objet

```
// bind the Object Reference in Naming
NameComponent nc = new NameComponent("ShapeList", "");
NameComponent path[] = {nc};
ncRef.rebind(path, shapeRef);
// wait for invocations from clients
java.lang.Object sync = new java.lang.Object();
synchronized (sync) {
    sync.wait();
}
} catch (Exception e) {
    System.err.println("ERROR: " + e);
    e.printStackTrace(System.out);
}
}
```



## Exemple CORBA Java: dessin offert par le serveur

```
import org.omg.CosNaming.*; import org.omg.CORBA.*;
class ShapeListServant extends _ShapeListImplBase
{ private Shape theList[]; private int version;
  private static int n=0; ORB theOrb;

  public ShapeListServant(ORB orb){
    theList = new Shape[4]; version = 0; theOrb = orb;
  }
  public Shape newShape(GraphicalObject g) throws
    ShapeListPackage.FullException {
    version++;
    Shape s = new ShapeServant( g, version );
    if(n >=100) throw new ShapeListPackage.FullException();
    theList[n++] = s;
    theOrb.connect(s);
    return s;
  }
  public Shape[] allShapes(){ return theList; }

  public int getVersion() { return version; }
}
```



# Exemple CORBA Java: forme dans le dessin

```
import org.omg.CosNaming.*; import org.omg.CORBA.*;

class ShapeServant extends _ShapeImplBase {
    int myVersion;
    GraphicalObject myG;

    public ShapeServant(GraphicalObject g, int version) {
        myG = g;
        myVersion = version;
    }

    public int getVersion() {
        return myVersion;
    }

    public GraphicalObject getAllState() {
        return myG;
    }
}
```



## Exemple CORBA Java: initialisation du client

```
import org.omg.CosNaming.*; import org.omg.CORBA.*;  
import org.omg.CosNaming.NamingContextPackage.*;  
import java.util.Properties; import java.util.Vector;  
  
public class ShapeListClient {  
    public static void main(String args[]) {  
        String option = "Read";  
        String shapeType = "Rectangle";  
        if(args.length > 0)  option = args[0]; // read or write  
        if(args.length > 1)  shapeType = args[1];  
        try{  
            // create and initialize the ORB at port number 1500  
            Properties props = new Properties();  
            props.put("org.omg.CORBA.ORBInitialPort", "1500");  
            ORB orb = ORB.init(args, props);  
        }  
    }  
}
```



# Exemple CORBA Java: appel de méthodes par le client

```
try{
    org.omg.CORBA.Object objRef =
    orb.resolve_initial_references("NameService");
    NamingContext ncRef = NamingContextHelper.narrow(objRef);
    NameComponent nc = new NameComponent("ShapeList", "");
    NameComponent path [] = { nc }; try{
        ShapeList shapeListRef = ShapeListHelper.narrow(ncRef.resolve(path));
        if(option.equals("Read")){
            Shape[] sList = shapeListRef.allShapes();
            for(int i=0; i<sList.length; i++){
                GraphicalObject g = sList[i].getAllState(); g.print();
            }
        } else {
            GraphicalObject g = new GraphicalObject(shapeType,
                new Rectangle(50,50,300,400), false);
            try { shapeListRef.newShape(g);}
            catch(ShapeListPackage.FullException e) {}
        }
    } catch(Exception e){}
    } catch(org.omg.CORBA.ORBPackage.InvalidName e){}
} catch(org.omg.CORBA.SystemException e){System.out.println("ERROR: "+e);}
}
```

# SOAP

- Simple Object Activation Protocol.
- Requête HTTP, action POST, contenu XML (nom de la fonction et arguments d'entrée), réponse XML avec arguments de retour.
- Facile à déployer car réutilise toute l'infrastructure Web.
- Permet d'utiliser Web Services Security (WS-Security), spécification qui définit l'implémentation des mesures de sécurité pour protéger un service web d'attaques externes.
  - Comment signer les messages SOAP pour en assurer l'intégrité et la non-répudiation
  - Comment chiffrer les messages SOAP pour en assurer la confidentialité
  - Comment attacher des jetons de sécurité pour garantir l'identité de l'émetteur
- Relativement inefficace.



# Requête SOAP

```
POST /InStock HTTP/1.1
```

```
Host: www.example.org
```

```
Content-Type: application/soap+xml; charset=utf-8
```

```
Content-Length: nnn
```

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Body
        xmlns:m="http://www.example.org/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>

</soap:Envelope>
```



# Réponse SOAP

HTTP/1.1 200 OK

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```



# RESTful API for Web Services

- Requête sans contexte (stateless), pour accéder une ressource représentée par un URL, en utilisant un certain format (e.g. JSON) et les méthodes HTTP (GET, PUT...).
- La méthode GET n'a aucun effet sur la ressource.
- Les méthodes PUT et DELETE sont idempotentes.
- Pas une norme, simplement un style architectural de service Web.
- De plus en plus populaire en raison de sa simplicité.
- Pas toujours bien utilisé (manques au niveau de la récupération d'erreurs, pas complètement sans contexte... ).



# Requête REST

- Requête

GET /api/StockPrice/IBM.json HTTP/1.1

Host: www.example.org

- Réponse

HTTP/1.1 200 OK

Content-Type: application/json

Content-Length: nnn

```
{ \"CompanyName\": \"IBM\",  
  \"Price\": \"34.5\"  
}
```



# gRPC

- Acronyme pour **gRPC Remote Procedure Call**.
- Rendu disponible par Google en 2015: “*A high performance, open-source universal RPC framework*” .
- Communication avec HTTP2 (compression des entêtes).
- Générateurs de code pour une dizaine de langages populaires (C++, Java, Python, Go, Ruby, C#...).
- Permet différents formats pour les messages mais implante initialement Protobuf (Protocol Buffers).
- Permet d'insérer le support pour l'équilibrage de charge, le traçage, le monitoring et l'authentification.
- Utilisé par Google, Netflix, Twitter, Cisco, Juniper...



# Requête gRPC

```
HEADERS (flags = END_HEADERS)
:method = POST
:scheme = http
:path = /google.pubsub.v2.Publisher
    Service/CreateTopic
:authority = pubsub.googleapis.com
grpc-timeout = 1S
content-type = application/grpc+proto
grpc-encoding = gzip
authorization = Bearer y235.
    wef315yfh138vh31hv93hv8h3v
```

```
DATA (flags = END_STREAM)
<Length-Prefixed Message>
```

```
HEADERS (flags = END_HEADERS)
:status = 200
grpc-encoding = gzip
content-type = application/grpc+proto

DATA
<Length-Prefixed Message>

HEADERS (flags = END_STREAM, END_HEADERS)
grpc-status = 0 # OK
trace-proto-bin = jher831yy13JHy3hc
```

# Interface Protocol Buffers

- Semblable à SUN RPC, encodage binaire.

```
message Person {  
    required string name = 1;  
    optional string email = 3;  
    enum PhoneType { MOBILE = 0; HOME = 1; WORK = 2; }  
  
    message PhoneNumber {  
        required string number = 1;  
        optional PhoneType type = 2 [default = HOME];  
    }  
    repeated PhoneNumber phone = 4;  
}  
  
Message PersonInfo { int32 age = 1; int32 salary = 2; }  
  
service Contact {  
    rpc Check (Person) returns (PersonInfo) {}  
}
```



# Librairie d'accès aux objets Protocol Buffers

```
Person person;
person.set_name("John Doe");
person.set_email("jdoe@example.com");
person.SerializeToOstream(&output);

person.ParseFromIstream(&input);
cout << "Name: " << person.name() << endl;
cout << "E-mail: " << person.email() << endl;
```



## Protocol Buffers

- Format binaire compatible vers l'avant et l'arrière.
- Chaque champ peut être obligatoire ou optionnel et vient avec un numéro (tag) pour l'identifier.
- Un message est une séquence de: numéro de champ, type, valeur. Le type est un entier de 3 bits (varint, float, length delimited...). Le numéro de champ et le type sont groupés en un varint et prennent ensemble usuellement 1 octet.
- Les varint sont des entiers de longueur variable (0-127 ou 1b+0-127 +varint). Un premier bit à un indique qu'un octet supplémentaire est utilisé. Petit boutien.
- Fonctionne entre les langages et entre les plates-formes et est possible de traiter un message même s'il contient des champs inconnus.
- Protoc génère le code pour initialiser, sérialiser et lire les types décrits de même que pour faire des RPC (un type pour l'envoi et un pour la réponse).

## Exemple de service de calcul de chemin

```
// route_guide.proto, (tiré de https://grpc.io/docs/tutorials/basic/c.html)

syntax = "proto3";
package routeguide;

service RouteGuide {
    rpc GetFeature(Point) returns (Feature) {}
    rpc ListFeatures(Rectangle) returns (stream Feature) {}
}

message Point {
    int32 latitude = 1;
    int32 longitude = 2;
}

message Rectangle {
    Point lo = 1;
    Point hi = 2;
}

message Feature {
    string name = 1;
    Point location = 2;
}
```



## Fonctions de service de chemin

```
// route_guide_server.cc
...
class RouteGuideImpl final : public RouteGuide::Service {
private:
    std::vector<Feature> feature_list_;
public:
    explicit RouteGuideImpl(const std::string& db) {
        routeguide::ParseDb(db, &feature_list_);
    }
    Status GetFeature(ServerContext* context, const Point* point,
                      Feature* feature) override {
        feature->set_name(GetFeatureName(*point, feature_list_));
        feature->mutable_location()->CopyFrom(*point);
        return Status::OK;
    }
    Status ListFeatures(ServerContext* context, const routeguide::Rectangle*
                        rectangle, ServerWriter<Feature>* writer) override {
        for (const Feature& f : feature_list_) {
            if (inside(f.location(), rectangle)) { writer->Write(f); }
        }
        return Status::OK;
    } };
```



# Activation du serveur de chemins

```
void RunServer(const std::string& db_path) {
    std::string server_address("0.0.0.0:50051");
    RouteGuideImpl service(db_path);

    ServerBuilder builder;
    builder.AddListeningPort(server_address,
        grpc::InsecureServerCredentials());
    builder.RegisterService(&service);
    std::unique_ptr<Server>
        server(builder.BuildAndStart());
    server->Wait();
}

int main(int argc, char** argv) {
    std::string db = routeguide::GetDbFileContent(argc, argv);
    RunServer(db);
    return 0;
}
```



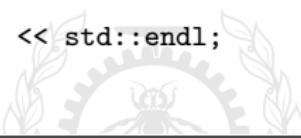
# Client pour le service de chemins

```
class RouteGuideClient {
public:
    RouteGuideClient(std::shared_ptr<Channel>
        channel, const std::string& db)
        : stub_(RouteGuide::NewStub(channel)) {}

    void ListFeatures() {
        routeguide::Rectangle rect;
        Feature feature;
        ClientContext context;

        rect.mutable_lo()->set_latitude(400000000);
        rect.mutable_lo()->set_longitude(-750000000);
        rect.mutable_hi()->set_latitude(420000000);
        rect.mutable_hi()->set_longitude(-730000000);

        std::unique_ptr<ClientReader<Feature> >
            reader(stub_->ListFeatures(&context, rect));
        while (reader->Read(&feature))
            std::cout << "Found feature called " << feature.name() << std::endl;
        Status status = reader->Finish();
    }
}
```



## Client pour le service de chemins (suite)

```
bool GetFeature(const Point& point, Feature* feature) {
    ClientContext context;

    Status status = stub_->GetFeature(&context, point, feature);
    if (!status.ok() || !feature->has_location() ||
        feature->name().empty()) return false;
    std::cout << "Found feature called " << feature->name() << std::endl;
    return true;
}
std::unique_ptr<RouteGuide::Stub> stub_;
std::vector<Feature> feature_list_;
};

int main(int argc, char** argv) {
    RouteGuideClient guide(grpc::CreateChannel("localhost:50051",
        grpc::InsecureChannelCredentials()),db);
    Feature feature;

    guide.GetFeature(MakePoint(0,0), &feature);
    guide.ListFeatures();
    return 0;
}
```

## Exemples d'utilisation des RPC

- Quelques services comme NFS sont basés sur les Sun RPC.
- Les principaux programmes dans le bureau GNOME (chiffrier, fureteur, éditeur, panneau...) offraient une interface CORBA mais se tournent maintenant vers D-BUS.
- CORBA est souvent utilisé par les compagnies de télécommunications pour leurs applications réparties de gestion de réseau.
- Java RMI est souvent utilisé dans des applications réparties internes.
- gRPC est utilisé à l'interne par Google et dans Kubernetes, ainsi que par quelques compagnies comme Netflix, Cisco, Morgan Stanley...



# Communication dans les systèmes répartis

- 1 Modèles de communication
- 2 Modèle requête-réponse RPC
- 3 Communication par messages



## Autres mécanismes

- D-Bus : commun à GNOME et KDE, (en remplacement de CORBA et DCOP). Bus système (e.g., notification de batterie, réseau, clé USB...) et bus de session (e.g., sélection).
- WebSockets : communication duplex entre client et serveur avec entête HTTP.
- AMQP, ZeroMQ : solutions de remplacement plus performantes et plus légères pour la communication inter-processus et l'appel de procédures à distance (Request–reply, Publish–subscribe, Push–pull, Exclusive pair), avec option at-least-once, at-most-once, exactly-once. Utilisé pour le courtage, OpenStack...



## Conclusion

- Les RPC sont un mécanisme intéressant pour formaliser les interfaces de manière à permettre la communication entre composantes possiblement distantes et dans un langage de programmation différent.
- Les Sun RPC sont encore assez répandus pour certains services.
- CORBA est depuis plusieurs années déployé à grande échelle mais des solutions plus simples et performantes sont populaires comme gRPC, AMQP et ZeroMQ.
- SOAP, REST, Ajax... sont très utilisés sur le Web.





# Communication par objets répartis

## Module 5

### INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① Objets et méthodes
- ② Gestion de la mémoire
- ③ Les objets repartis en C#
- ④ Les objets répartis en Java
- ⑤ Java Enterprise Edition
- ⑥ Conclusion



# Communication par objets répartis

- 1 Objets et méthodes
- 2 Gestion de la mémoire
- 3 Les objets repartis en C#
- 4 Les objets répartis en Java
- 5 Java Enterprise Edition
- 6 Conclusion



## Appel de méthodes à distance

- Modèle objet: références aux objets, interfaces, méthodes, exceptions, ramasse-miettes.
- Proxy: se présente comme un objet local, ses méthodes sérialisent les arguments, transmettent la requête au module de communication et retournent la réponse reçue.
- Répartiteur: reçoit les requêtes dans le serveur et les communique au squelette correspondant.
- Squelette: reçoit les requêtes pour un type d'objet, désérialise les arguments, appelle la méthode correspondante de l'objet référencé et sérialise la réponse à retourner.
- Module de communication: envoi de requête client (type, numéro de requête, référence à un objet, numéro de méthode, arguments), la requête reçue par le serveur est envoyée au répartiteur et la réponse est retournée.

## Module des références réseau

- Table des objets importés (pour chaque objet distant utilisé, adresse réseau et adresse du proxy local correspondant),
- Table des objets exportés (pour chaque objet utilisé à l'extérieur, liste des clients, adresse réseau et adresse locale).
- Lorsqu'une référence réseau est reçue, son proxy ou objet local est trouvé ou un nouveau proxy est créé et une entrée ajoutée dans la table.
- Lorsqu'un objet réseau local est passé en argument, son adresse réseau le remplace et il doit se trouver dans la table des objets exportés.



## Services pour l'appel de méthodes distantes

- Service de nom: permet d'obtenir une référence objet réseau à partir d'un identificateur/nom.
- Contextes d'exécution: le serveur peut créer un fil d'exécution pour chaque requête.
- Activation des objets: le serveur peut être démarré automatiquement lorsqu'une requête pour un objet apparaît (message augmentation de salaire pour le budget qui est stocké sur disque dans un fichier de chiffrier).
- Stockage des objets persistents.
- Service de localisation.



# Communication par objets répartis

- 1 Objets et méthodes
- 2 Gestion de la mémoire
- 3 Les objets repartis en C#
- 4 Les objets répartis en Java
- 5 Java Enterprise Edition
- 6 Conclusion



## Gestion de la mémoire en réparti

- Malloc et Free difficiles à appeler au bon moment en réparti!
- Chaque client s'enregistre lorsqu'il utilise un objet et avertit lorsqu'il l'a terminé. Le serveur libère un objet si ni lui ni les clients ne l'utilisent.
- Ramasse-miette conventionnel qui travaille en réparti? Très difficile et inefficace puisqu'il faut tout arrêter en même temps lors du ramassage.
- Ramasse-miette local à chaque processus avec notification automatique au serveur lorsqu'un client cesse d'utiliser un proxy.
  - Une référence à un objet O est envoyée de A à B et A cesse de l'utiliser, le message de A (référence à O inutilisée) peut arriver avant celui de B (utilise O).
  - Cycle de références entre 2 clients ou plus.
  - Si le client disparaît, le serveur doit l'enlever de la liste (limite de validité pour les références, ou besoin de messages de maintien keepalive).

# Communication par objets répartis

- 1 Objets et méthodes
- 2 Gestion de la mémoire
- 3 Les objets repartis en C#
- 4 Les objets répartis en Java
- 5 Java Enterprise Edition
- 6 Conclusion



## Le Remoting en C#

- Fonctionne entre tous les langages supportés par le CLR: C#, C++, VB...
- Utilise la reflexivité pour générer le code client et serveur dynamiquement.
- Choix d'encodage binaire ou XML.
- Choix de canal TCP ou HTTP.
- Les objets qui héritent de MarshalByRef sont passés par référence à travers le réseau, les autres sont passés par valeur et doivent supporter l'interface ISerializable.
- Un objet déjà créé peut être exporté ou on peut enregistrer un type et l'objet est créé au moment de la requête selon un de deux modes (Singleton, Single call).
- Les références aux objets peuvent avoir une date d'expiration pour éviter d'avoir à vérifier pour les références inutilisées; passé cette expiration elles n'existent plus.

# Interface en C# Remoting

```
using System;

abstract public class Server: MarshalByRefObject {
    abstract public string GetInfo();
}
```



# Client en C# Remoting

```
using System;
using System.IO;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

class Client
{
    public static void Main(string[] args) {
        int serverPort = 9090;
        host = "localhost";
        TcpChannel channel = new TcpChannel();
        ChannelServices.RegisterChannel(channel);
        Server server = (Server)RemotingServices.Connect(
            typeof(Server),
            "tcp://" + host + ":" + serverPort.ToString() + "/MyServer");
        string info = server.GetInfo();
    }
}
```



# Serveur en C# Remoting

```
using System; using System.IO;
using System.Threading;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

class AServer: Server {
    public override string GetInfo() {
        return "A Server version 0.01"
    }
    public static void Main(string[] args) {
        int serverPort = 9090;
        TcpChannel channel = new TcpChannel(serverPort);
        ChannelServices.RegisterChannel(channel);
        AServer server = new AServer();
        ObjRef serverRef = RemotingServices.Marshal(server, "MyServer",
            typeof(Server));
        Thread.Sleep(20000);
        RemotingServices.Disconnect(server);
        ChannelServices.UnregisterChannel(channel);
    }
}
```



# Communication par objets répartis

- 1 Objets et méthodes
- 2 Gestion de la mémoire
- 3 Les objets repartis en C#
- 4 Les objets répartis en Java
- 5 Java Enterprise Edition
- 6 Conclusion



## Java RMI

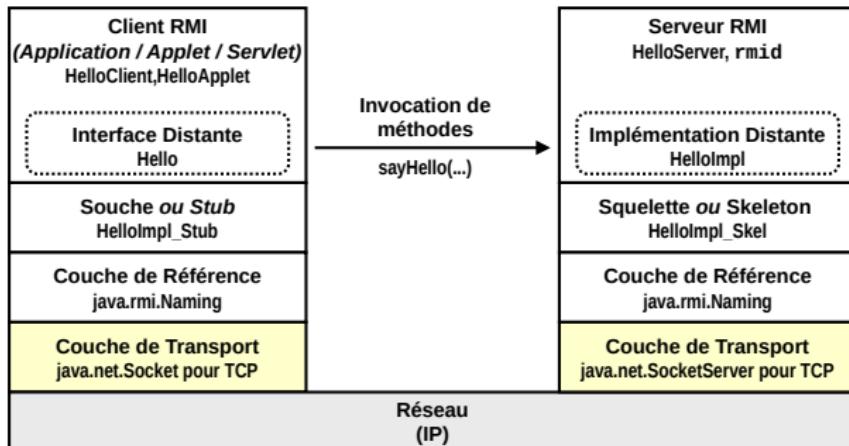
- Ne fonctionne que de Java à Java.
- S'applique à tout type qui implante l'interface Remote.
- Les méthodes doivent accepter l'exception RemoteException.
- Les arguments seront envoyés et la valeur renvoyée sert de réponse.
- Ces arguments doivent être des types primitifs, des objets réseau, ou des objets qui implantent l'interface Serializable (le graphe complet d'objets rejoints par un argument peut être transmis).
- Le type des objets serialisés contient une référence à la classe. La classe peut être téléchargée au besoin par le récepteur.
- Le type des objets réseau contient une référence à leur classe proxy (stub) qui peut être téléchargée au besoin.

## Java RMI

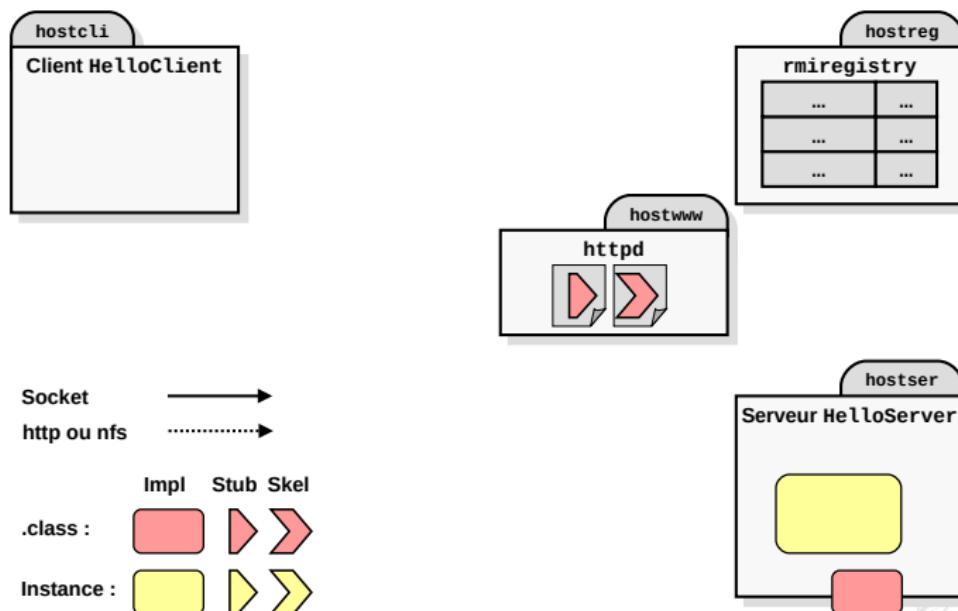
- Le RMIregistry maintient une table (nom, référence réseau) pour obtenir une référence à un objet désiré qui se trouve sur un ordinateur donné. Un nom a la forme:  
`//hostname:port/objectName.`
- rmic peut être utilisé pour créer les proxy à partir du code des classes compilées. Le répartiteur est générique et fourni en librairie.
- Les appels distants peuvent être servis par des fils d'exécution différents, surtout s'il viennent de clients (connexions) différents.



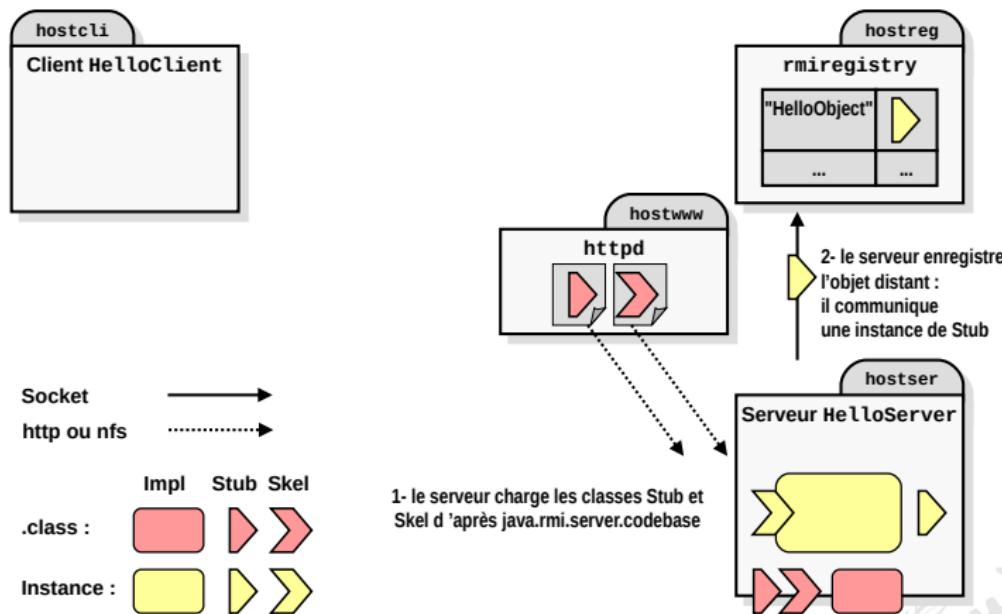
# Structure logique des couches RMI



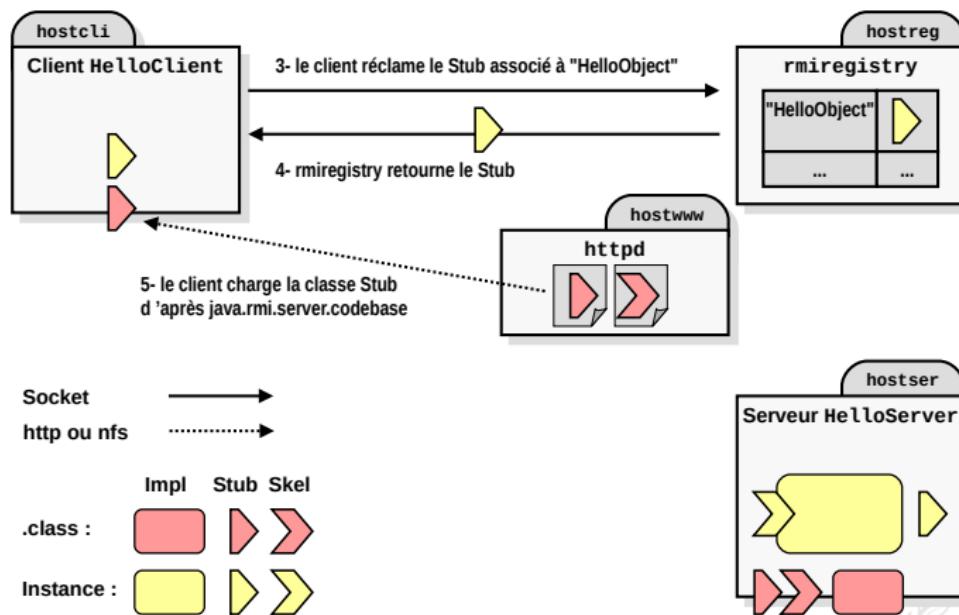
# RMI: la configuration



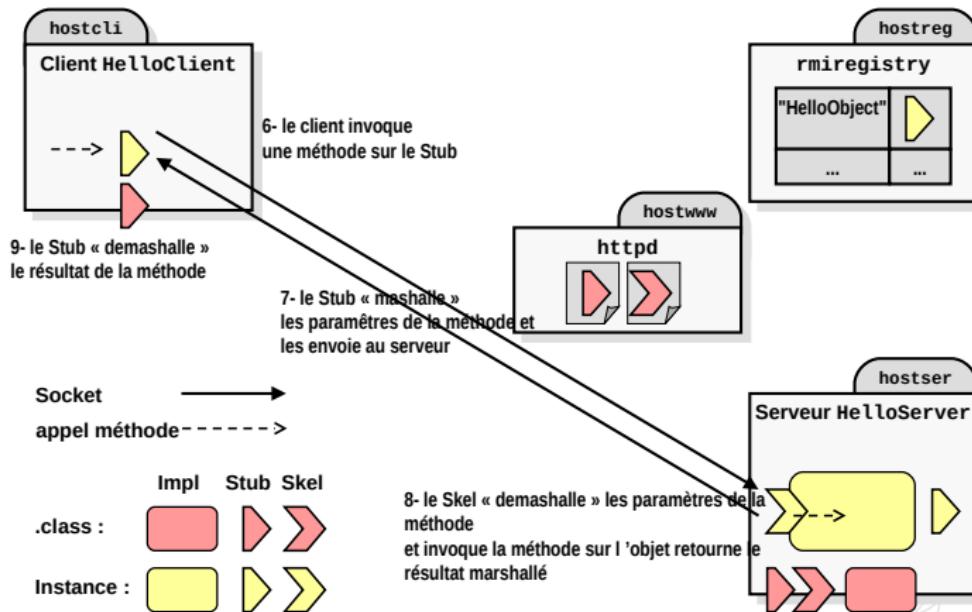
# RMI: l'enregistrement de l'objet



# RMI: la récupération du Stub



# RMI: invocation d'une méthode



# Création et manipulation d'objets distants

- 5 Packages
  - java.rmi : pour accéder à des objets distants
  - java.rmi.server : pour créer des objets distants
  - java.rmi.registry : lié à la localisation et au nommage d'objets distants
  - java.rmi.dgc : ramasse-miettes pour les objets distants
  - java.rmi.activation : support pour l'activation d'objets distants
- Etapes du développement
  - Spécifier et écrire l'interface de l'objet distant.
  - Ecrire l'implémentation de cette interface.
  - Générer les Stub/Skeleton correspondants.
  - Ecrire le serveur qui instancie l'objet implémentant l'interface, exporte son Stub puis attend les requêtes via le Skeleton.
  - Ecrire le client qui réclame l'objet distant, importe le Stub et invoque une méthode de l'objet distant via le Stub.

## Exemple RMI: interfaces réseau

```
// Interface réseau Forme
package examples.RMIShape;
import java.rmi.*;
import java.util.Vector;

public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject getAllState() throws RemoteException;
}

// Interface réseau dessin (liste de Forme)
package examples.RMIShape;
import java.rmi.*;
import java.util.Vector;

public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException;
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```



## Exemple RMI: serveur de dessin

```
package examples.RMIShape;
import java.rmi.*;

public class ShapeListServer {
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        System.out.println("Main OK");
        try{
            ShapeList aShapelist = new ShapeListServant();
            System.out.println("After create");
            Naming.rebind("ShapeList", aShapelist);
            System.out.println("ShapeList server ready");
        }
        catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());
        }
    }
}
```



## Exemple RMI: dessin exporté par le serveur

```
package examples.RMIShape;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.Vector;

public class ShapeListServant extends UnicastRemoteObject
    implements ShapeList{
    private Vector theList;
    private int version;

    public ShapeListServant()throws RemoteException{
        theList = new Vector();
        version = 0;
    }
```



## Exemple RMI: dessin exporté par le serveur (Suite)

```
public Shape newShape(GraphicalObject g) throws RemoteException{
    version++;
    Shape s = new ShapeServant( g, version);
    theList.addElement(s);
    return s;
}

public Vector allShapes()throws RemoteException{
    return theList;
}

public int getVersion() throws RemoteException{
    return version;
}
}
```



## Exemple RMI: forme exportée par le serveur

```
package examples.RMIShape;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class ShapeServant extends
    UnicastRemoteObject implements Shape {
    int myVersion;
    GraphicalObject theG;

    public ShapeServant(GraphicalObject g, int version) throws RemoteException{
        theG = g;
        myVersion = version;
    }

    public int getVersion() throws RemoteException {
        return myVersion;
    }

    public GraphicalObject getAllState() throws RemoteException{
        return theG;
    }
}
```



## Exemple RMI: client

```
// Client
package examples.RMIShape;
import java.rmi.*; import java.rmi.server.*;
import java.util.Vector;
import java.awt.Rectangle; import java.awt.Color;

public class ShapeListClient{
    public static void main(String args[]){
        String option = "Read";
        String shapeType = "Rectangle";
        if(args.length > 0)  option = args[0]; // read or write
        if(args.length > 1)  shapeType = args[1];
        System.out.println("option = " + option + "shape = " + shapeType);

        if(System.getSecurityManager() == null){
            System.setSecurityManager(new RMISecurityManager());
        }
        else System.out.println("Already has a security manager");
        ShapeList aShapeList = null;
        try{
            aShapeList = (ShapeList) Naming.lookup("//test.shapes.net/ShapeList");
            System.out.println("Found server");
        }
```

## Exemple RMI: client (Suite)

```
Vector sList = aShapeList.allShapes();
System.out.println("Got vector");

if(option.equals("Read")){
    for(int i=0; i<sList.size(); i++){
        GraphicalObject g = ((Shape)sList.elementAt(i)).getAllState();
        g.print();
    }
} else {
    GraphicalObject g = new GraphicalObject(shapeType,
        new Rectangle(50,50,300,400),Color.red, Color.blue, false);
    System.out.println("Created graphical object");
    aShapeList.newShape(g);
    System.out.println("Stored shape");
}
}

catch(RemoteException e) {
    System.out.println("allShapes: " + e.getMessage());
}
catch(Exception e) {
    System.out.println("Lookup: " + e.getMessage());
}
}
```



# Communication par objets répartis

- 1 Objets et méthodes
- 2 Gestion de la mémoire
- 3 Les objets repartis en C#
- 4 Les objets répartis en Java
- 5 Java Enterprise Edition
- 6 Conclusion



# Java Enterprise Edition (EE) étend Standard Edition (SE)

- J2EE 1.2 (1999), 1.3 (2001), 1.4 (2003), Java EE 5 (2006), 6 (2009), 7 (2013), 8 (2017).
- JavaServer Pages (JSP): interfaces pour HTTP.
- Unified Expression Language (EL): langage de script pour les expressions.
- JavaServer Faces (JSF): interface usager.
- Java API for RESTful Web Services (JAX-RS): support pour REST.
- Enterprise JavaBeans (EJB): support de composantes pour les business objects.
- Java Transaction API (JTA): support pour les transactions réparties.
- Java Persistence API (JPA): stockage de l'état dans une base de donnée.
- Bean Validation: annotations de contraintes...

## Enterprise Java Beans

- Programmer la logique de l'application séparément du reste de l'environnement: grappe pour le déploiement, interface usager, base de donnée, RPC, sécurité...
- Suppose une architecture classique à trois tiers (interface usager, logique de l'application, base de donnée).
- EJB 1.0 (1998): architecture de base.
- EJB 1.1 (1999): fichiers de méta-données en XML décrivant l'environnement, composantes (beans) de session et d'entité. Interface d'accès à distance.
- EJB 2.0 (2001): interface par message.
- EJB 2.1 (2003): Minuterie et support Web Service.
- EJB 3.0 (2006): POJO avec annotations remplace les méta-données XML.
- EJB 3.1 (2009): Quelques simplifications à l'architecture.
- EJB 3.2 (2013): Changements mineurs.

## Conteneurs EJB

- Logiciels comme JBoss (Red Hat), WebSphere (IBM), NetWeaver (SAP), WebLogic (Oracle), Geronimo (Apache), GlassFish (Sun).
- Reçoit les requêtes d'objets/clients locaux ou distants (RMI, RMI-IIOP, Web Services, JMS) qui fournissent l'interface usager.
- Les requêtes sont validées et dirigées vers les objets de session qui sont référencés ou créés au besoin (Business logic).
- Les objets entités sont accédés par les objets de session et leur état est géré et mis à jour dans la base de données selon ce qui a été spécifié dans les annotations (Persistence).



## Les rôles selon EJB

- Bean provider: fournisseur des composantes de l'application.
- Application assembler: concepteur de l'application qui assemble les composantes pour obtenir les fonctions désirées.
- Deployer: responsable du déploiement de l'application dans un environnement adéquat.
- Service provider: spécialiste des systèmes répartis qui s'assure du niveau de service désiré.
- Persistence provider: spécialiste des bases de données.
- Container provider: spécialiste de l'environnement d'exécution des composantes Java.
- System administrator: administrateur du système informatique qui s'assure que le système fonctionne selon ce qui a été conçu.

# Communication par objets répartis

- 1 Objets et méthodes
- 2 Gestion de la mémoire
- 3 Les objets repartis en C#
- 4 Les objets répartis en Java
- 5 Java Enterprise Edition
- 6 Conclusion



## Conclusion

- Java RMI est simple d'utilisation et est utilisé dans des systèmes homogènes Java.
- Le Remoting est simple d'utilisation et fonctionne avec plusieurs langages (C#, C++, VB). Il remplace très avantageusement DCOM.





# Services de fichiers

Module 6

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① Introduction
- ② Services poste-à-poste
- ③ Services de fichiers conventionnels



# Services de fichiers

- 1 Introduction
- 2 Services poste-à-poste
- 3 Services de fichiers conventionnels



## Echanges de fichiers poste-à-poste

- Souvent non structuré à l'avance.
- Ne cherche pas à reproduire la sémantique de l'accès à un fichier local (POSIX).
- Se préoccupe de la mise à l'échelle, de la performance globale et individuelle, et de l'équité des usagers (donner versus recevoir la bande passante) et parfois de l'anonymat.
- Pour des gros fichiers publics (e.g. torrent d'image ISO de logiciels libres), la contestation d'un régime dictatorial, ou l'échange de fichiers dont l'usage est restreint par droits d'auteur.



## Services de fichiers conventionnels

- Eviter d'avoir à gérer localement le stockage et les copies de sécurité.
- Partager les fichiers au niveau d'un groupe, département, entreprise ou le monde entier.
- Transparent aux applications, reproduit la sémantique d'un accès local.



# Services de fichiers

- 1 Introduction
- 2 Services poste-à-poste
- 3 Services de fichiers conventionnels



# NAPSTER

- Serveur centralisé pour l'index (liste des pairs offrant un fichier donné).
- Un client se connecte à l'index pour avoir la liste des autres clients offrant un fichier qu'il recherche. Il offre ses propres fichiers qui sont ajoutés à l'index.
- Le client se connecte à un des autres clients pour obtenir le fichier recherché.
- Il se peut que d'autres clients se connectent au premier pour télécharger un des fichiers qu'il offre.



# GNUTELLA

- Pas de serveur central, seulement des pairs parmi lesquels certains sont plus puissants et se distinguent (ultra-pairs).
- Graphe bien connecté avec chaque pair connecté à quelques ultra-pairs et les ultra-pairs connectés à des dizaines d'ultra-pairs.
- Chaque pair envoie la liste de ses fichiers aux ultra-pairs connectés (code de dispersion des mots contenus dans les titres).
- Les ultra-pairs font l'union des tables de leurs pairs et envoient cette information à leurs ultra-pairs.
- Une requête est envoyée de préférence à un nœud ayant le fichier, autrement à un ultra-pair qui a annoncé le fichier, et finalement à un ultra-pair à la fois.
- La requête contient l'adresse de l'ultra-pair qui a initié la recherche et le fichier trouvé lui est envoyé directement.

# BITTORRENT

- Système réparti poste à poste spécialement conçu pour la propagation de gros fichiers (e.g. image iso Ubuntu, contenu de DVD...).
- Fichier .torrent donne le nom et la longueur du fichier, l'adresse du serveur central de suivi pour ce fichier, et une somme de contrôle.
- Le serveur maintient une liste des pairs qui ont une copie complète ou partielle du fichier.
- Un client obtient du serveur une liste de pairs possédant le fichier. Il les contacte pour obtenir les morceaux en parallèle dans n'importe quel ordre.
- Des statistiques sont maintenues aux 10s sur ce que chaque pair contribue. Chaque pair donne normalement priorité aux autres pairs qui contribuent à l'effort.
- Les morceaux de fichiers les plus rares sont offerts en priorité.

# Services de fichiers

- 1 Introduction
- 2 Services poste-à-poste
- 3 Services de fichiers conventionnels



## Module local d'accès aux fichiers

- Pilote d'interface: disque(s) ou partition de disque accessible sous la forme d'un vecteur de blocs.
- Gestion des blocs: tampons d'entrée-sortie, pré-lecture, écriture asynchrone, ordonnancement des accès.
- Système de fichiers avec inode (identificateur de fichier avec permissions d'accès et liste de blocs directs ou indirects qui le composent) et répertoire (fichier contenant une liste de noms versus identificateurs de fichiers permettant de constituer une structure hiérarchique).
- Table de mount : certains chemins mènent vers d'autres partitions ou disques.
- Cache de dentries : cache de chemins versus identificateur de fichier.



## Accès de fichiers par réseau

- Transparence d'accès: un programme peut lire un fichier local ou distant sans distinction.
- Transparence de localisation: le nom ne change pas selon l'ordinateur où se trouve le programme qui veut accéder le fichier.
- Transparence de concurrence: les accès concurrents peuvent se faire de la même manière qu'en local.
- Transparence de défectuosité (messages perdus, serveur réinitialisé, réseau partitionné...).
- Transparence de performance (performance similaire même avec un nombre croissant de clients).
- Disponible sur plusieurs plates-formes.
- Transparence même en cas de réPLICATION et de migration.
- Sécurité

## Modules d'un service de fichiers réparti

- Propriétés d'un fichier: longueur en octets, date de création, dernière lecture/écriture, décompte de référence, propriétaire, liste de contrôle des accès.
- Service de répertoire: convertir le chemin demandé en identificateur de fichier et vérifier les permissions d'accès.  
Retourner une clé appropriée.
- Service de fichiers de base: la clé permet de savoir le fichier à accéder, et les droits que possède le détenteur sur ce fichier.  
Opérations de lecture, écriture sur le fichier.
- Module client qui utilise les services de répertoires et de fichiers et qui utilise des opérations réapplicables (idempotentes) pour tolérer les défaillances (message retransmis). Il peut avoir à se connecter à plus d'un serveur.

## Interface pour le service de répertoire

---

- FID Lookup(FID dir, String name, Mode accessMode, UID user);
- AddName(FID dir, String name, FID file, UID user)  
{NameDuplicate};
- UnName(FID dir, String name) {NotFound};
- ReName(FID dir, String oldName, String newName)  
{NotFound, NameDuplicate};
- StringSequence GetNames(FID dir, String pattern);



## Interface pour le service de fichiers

- CharSequence Read(FID file, unsigned pos, unsigned length) {BadPosition};
- Write(FID file, unsigned pos, CharSequence data) {BadPosition};
- FID Create();
- Truncate(FID file, unsigned length);
- Delete(FID file);
- AttributeSequence GetAttributes(FID file);
- SetAttributes(FID file, AttributeSequence attr);



## Considérations

- Grouper les fichiers par sous-arbre ou autrement pour les répartir sur plus d'un serveur.
- La création de fichiers orphelins s'ils ne sont pas mis dans un répertoire (opération combinée Create/AddName).
- Le FID peut contenir: Identificateur de groupe (32 bits adresse IP + 16 bits temps), 32 bits numéro de fichier, 32 bits nombre aléatoire + 5 bits permission encryptés, 5 bits de permission non encryptés. Il identifie le fichier et les permissions mais est très difficile à contrefaire.
- Serveur de localisation de groupe: numéro de groupe versus ordinateur et numéro de port.
- Le serveur de fichiers utilise une cache comme à l'habitude. Les clients peuvent aussi avoir une cache mais cela pose un problème de cohérence sérieux.

## Sun NFS

- Introduit en 1985, définition placée dans le domaine public en 1989. Un sous-arbre du serveur peut être monté localement.
- Transparence d'accès, et de localisation, migration, et réPLICATION en lecture avec un peu d'effort via autofs.
- Le serveur peut se réinitialiser sans affecter l'accès du client autrement qu'en retardant son accès (stateless).
- Sur un réseau 10Mbits/s peu chargé, la performance pour lire des blocs en accès aléatoire est comparable à un accès local (accès disque de 10ms pour un bloc de 8K).
- Pas de support pour la réPLICATION lecture/écriture, ce qui limite l'adaptabilité à un grand nombre de clients.
- Support mitigé pour le verrouillage des fichiers à des fins de synchronisation.



## Implantation

---

- Le noyau maintient une couche VFS (Virtual File System) pour savoir quel sous-arbre vient de quel système de fichier.
- Le client NFS est un module du noyau qui est utilisé lors d'opérations sur les fichiers qui sont sur des serveurs NFS. Le processus biod s'occupe de pré-lecture et d'écriture asynchrone de blocs.
- Monté en dur, les accès NFS sont bloquants, autrement une expiration de délai cause le retour avec un code d'erreur des opérations sur un fichier. Certains programmes supposent que les accès aux fichiers ne peuvent causer d'erreur!
- La traduction du chemin doit se faire une composante à la fois afin de vérifier si on arrive à une sous-branche montée ailleurs.

## Implantation

---

- Automount/autofs permet de monter les sous-arbres à la demande et de les démonter lorsqu'ils ne sont plus utilisés.
- Le serveur roule portmap, mountd, et nfsd qui peuvent être des processus en mode usager.
- Lors d'un accès pour un fichier, la date de dernière modification est vérifiée et utilisée pour invalider la cache du client. Autrement, on ne force une vérification de la date de dernier accès qu'une fois par 3 secondes pour les fichiers et 30 secondes pour les répertoires.
- Les opérations réseau utilisent des appels Sun RPC, avec optionnellement de l'encryption.
- Les écritures sont synchrones.



## Andrew File System (AFS)

- Version 1 en 1986, et 2 en 1989.
- Développé à l'université CMU, prévu pour des milliers de clients et des dizaines de serveurs.
- Lorsqu'un fichier est ouvert, il est copié sur le disque local du client (par morceaux de 64k ou le fichier complet) s'il ne s'y trouve pas déjà.
- Lors de la fermeture il est mis à jour sur le serveur.
- Efficace pour de nombreux petits fichiers qui sont soit lecture-seulement, soit modifiables par un seul usager, ce qui est typique d'un environnement de laboratoire informatique.
- Distribué commercialement au début puis relâché comme logiciel libre.



## Implantation

- Les appels open et close sont interceptés pour effectuer les accès requis au serveur.
- Le client compte sur le serveur pour l'avertir de toute mise à jour sur les fichiers qu'il contient dans sa cache. Il revérifie après une réinitialisation, ou lorsqu'il ne reçoit rien du serveur pour quelques minutes.
- Conserve un état mais plus efficace, équivaut à 40% de la charge du même service par NFS.
- Le client a la garantie que lors d'un open il a la version la plus récente (à quelques minutes près), et que lors d'un close la mise à jour est propagée au serveur.
- Volumes (groupes de fichiers) lecture-seulement peuvent être répliqués sur plusieurs serveurs.
- Base de donnée de localisation de volumes répliquée sur chaque serveur.



# CODA

---

- Développé à CMU à partir de 1990, première version vers 1995.
- Permettre la réPLICATION de volumes lecture-écriture, et l'opération déconnectée.
- Semblable à AFS sauf pour open lire d'un seul serveur et pour close envoyer la modification à tous les serveurs.
- Une liste permet de spécifier les fichiers dont une copie locale devrait toujours exister. Au moment de la reconnexion, ou après une panne de réseau, la cache est revalidée.
- Lorsqu'un serveur redevient accessible, il faut vérifier ses vecteurs de numéros de version pour les fichiers modifiés. S'il manque des mises à jour, elles lui sont propagées. Si des mises à jour différentes ont été reçues de part et d'autre, il y a un conflit à résoudre.

# Common Internet File System (CIFS)

- Server Message Block (SMB) par-dessus NetBIOS, développé chez IBM pour DOS.
- Intégré à Lan-Manager (OS/2, Windows for Workgroups)
- En 1996, renommé CIFS.
- OpLock Exclusive, aucune autre copie sur un autre client, le détenteur n'a pas à propager chaque modification.
- OpLock Level 2, accès partagé, peut faire les lectures en cache mais doit propager toute modification.
- OpLock Batch, permission de retarder les close et de les annuler en cas de open qui survient peu de temps après.
- OpLock Break, révocation de OpLock car le fichier vient d'être modifié par un autre client.



# Global File System GFS

- Commencé à l'Université du Minnesota, 1995, Sistina Software, 1999, Red Hat 2003.
- Plusieurs serveurs peuvent se connecter au même SAN (disques réseau) par iSCSI, FibreChannel ou AoE (ATA over Ethernet), offrant une certaine redondance.
- Aussi possible de se connecter sur un disque répliqué par logiciel en Linux: GFS2 sur DRDB sur DM.
- Gestionnaire de verrou (Distributed Lock Manager DLM) assure que les accès sur des connexions différentes de SAN ou des réplicats de disque différents ne sont pas en conflit.
- Lorsqu'un nœud est considéré défaillant, il est désactivé par matériel (relais) pour éviter qu'il ne cause des conflits.



# Gluster File System

- La compagnie Gluster (GNU Cluster) a été fondée en 2005 pour offrir un système de fichier libre et performant.
- Acheté en 2011 par Red Hat.
- Fait l'aggrégation de systèmes de fichiers natifs (e.g. xfs, btrfs) et les exporte sous plusieurs protocoles (NFS, CIFS, HTTP...).
- Gère la distribution et la redondance.
- Un des systèmes les plus utilisés avec OpenStack mais qui sera concurrencé par Ceph.



## Le système de fichiers Lustre

- Compagnie démarrée à CMU, achetée par SUN puis SUN par Oracle. Produit discontinué chez Oracle mais qui se poursuit en logiciel libre. Utilisé par 15 des 30 au sommet du Top500.
- Serveur de métadonnées, (Meta Data Server MDS); répertoires, liste des objets de stockage pour le fichier, propriétés des fichiers.
- Plusieurs serveurs d'objets de stockage (Object Storage Server) avec chacun quelques groupes de disques, e.g. 2 à 8, (Object Storage Target). Un fichier peut avoir des morceaux sur plusieurs OSS.
- Gestion des verrous (Distributed Lock Manager) lecture et écriture par sections de fichiers.
- Redondance Active/Passive pour MDS et Active/Active pour OSS.

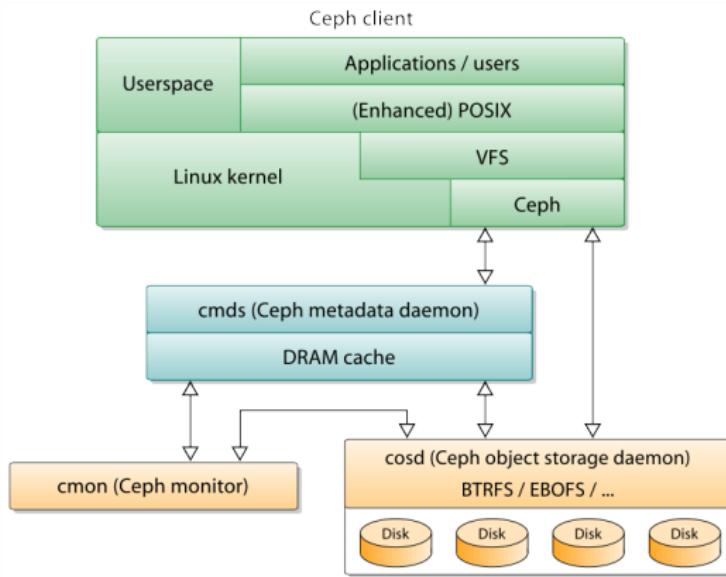
# Google File System GFS

- Serveur maître responsable des métadonnées (attribut, emplacement des morceaux de fichiers avec réplicats), Master Server MS.
- Journal des opérations du MS vers un autre serveur pour récupération en cas de panne.
- Serveurs (centaines) de morceaux (64MB) de fichiers, ChunkServers.
- Lors d'une modification à un fichier, un ChunkServer est identifié comme primaire pour chaque morceau.
- Le client envoie la modification à chaque réplicat puis confirme la modification au primaire qui détermine l'ordre, écrit, puis demande aux réplicats d'écrire dans le même ordre et confirme au client.



# CEPH

- Nouveau service de blocs, fichiers, ou objets de stockage prévu pour la mise à l'échelle et l'infonuagique.
- Première version stable en 2016.



# CEPH

**Ceph-mon** cluster monitor, vérifie l'état des serveurs de la grappe pour noter s'ils sont actifs ou en panne (au moins 2 pour tolérance aux pannes). Maintient une carte des serveurs avec consensus par Paxos.

**Ceph-mds** serveur de métadonnées (répertoires et inodes). Chacun couvre un sous-espace ajusté dynamiquement avec des recouplements pour fins de redondance. Les métadonnées sont stockées dans des fichiers.

**Ceph-osd** serveur pour le contenu des fichiers eux-mêmes, usuellement sur des partitions xfs ou btrfs. (Au moins 2, par défaut 3, pour réPLICATION).

**Ceph-rgw** serveur RESTful pour offrir une interface compatible avec Amazon S3 et OpenStack Swift.

# CEPH

- Un fichier est identifié par un inode number (INO) fourni par le serveur de métadonnées et décomposé selon sa taille en plusieurs objets (ONO) qui ont chacun un OID. (objets)
- Une fonction de hachage simple associe le OID à un groupe de placement (PGID). Le groupe s'occupe de la réplication. (Placement group)
- Le placement d'un groupe (PGID) sur des serveurs de fichiers (ceph-osd) est déterminé par un algorithme : Controlled Replication Under Scalable Hashing (CRUSH). (claw)
- Emphase sur la performance (morceaux de fichiers sur plusieurs serveurs), la mise à l'échelle (serveurs de métadonnées séparés, CRUSH) et la tolérance aux pannes.



# OrangeFS

- Nouvelle génération de PVFS (*Parallel Virtual File System*), qui était développé à Clemson University et ANL pour les grappes MPI.
- Services de métadonnées et de fichiers séparés.
- Les fichiers sont des objets avec des paires clé/valeur et une séquence d'octets (bytestream).
- Les clients peuvent accéder le service directement (Linux, Windows, Mac) ou par WebDAV, S3, Hadoop. . .
- La répartition des objets sur les serveurs est la partie importante et peut être décidée automatiquement ou spécifiée par l'utilisateur.



# HDFS

- Stockage de fichiers (ne satisfait pas POSIX) afin de répartir de gros fichiers entre les nœuds d'une grappe.
- Au moins un name-node pour les métadonnées et une grappe de data-nodes pour la distribution et réPLICATION des blocs de fichiers.
- Les nœuds de données peuvent communiquer entre eux pour équilibrer l'espace occupé et la charge.
- Optimisé pour de très gros fichiers, surtout en lecture, sur lesquels on veut faire du traitement réparti (HADOOP), ce qui permet de conserver les données près de là où elles seront utilisées.



## Conclusion : services de fichiers pour le nuage

- OpenStack utilise principalement NFS, CIFS, GlusterFS, HDFS et bientôt Ceph.
- NFS et CIFS sont les solutions classiques, connues, faciles à installer.
- GlusterFS est la solution la plus établie pour les grandes installations qui nécessitent une certaine mise à l'échelle. Ceph sera un compétiteur sérieux, plus avancé technologiquement.
- HDFS peut être intéressant pour des applications de type Hadoop.
- Plusieurs systèmes de fichiers propriétaires sont aussi mis de l'avant par leurs fournisseurs respectifs (e.g., VMWare).





# Service de répertoire de noms

Module 7

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Problématique

- Nom d'ordinateur, d'utilisateur, de service sous forme textuelle à traduire en identificateur binaire et éventuellement en identificateur de bas niveau (nom/adresse IP/adresse Ethernet, nom de fichier/capacité/serveur-fichier).
- Base de donnée de paires nom-attributs.
- Service de consultation, recherche, découverte, modification, effacement, enregistrement.
- Serveurs hiérarchiques, répliqués.
- Doit fonctionner à l'échelle planétaire (nom des ordinateurs, utilisateurs...).
- Exemples: DNS, X500, CORBA Naming Service, Portmap, LDAP.



## Organisation hiérarchique de noms

- **Domaine de nom**: niveau dans la hiérarchie pour lequel une autorité gère les noms mais peut déléguer un sous-domaine.
- Pour chaque nom une série de paires (type, valeur) est stockée: (usager, nom/téléphone/adresse...), (répertoire, liste de noms)...
- **Services**: AttributeSequence Lookup(String name, AttributeType t), Bind(String name, AttributeSequence attr), UnBind(String name).
- Une composante d'un nom peut mener à un autre contexte. La recherche d'un nom peut donc se faire de manière itérative ou récursive en partant du contexte racine dont la localisation doit être bien connue.
- Cache: (nom, attributs), ou (Préfixe,nom de serveur).
- **Différents espaces de noms**: ordinateurs, usagers, services, fichiers...

## Discussion

- Noms relatifs.
- Fusion d'espaces de noms.
- Alias pour aider restructuration.
- **URI** (*Uniform Resource Identifier*, peut être URL ou URN),  
**URL** (*Uniform Resource Locator*, comme un lien) et  
**URN** (*Uniform Resource Name*, comme un ISBN par exemple).
- Le nom peut être un des attributs avec recherche possible sur tous les attributs: quel est le nom de l'usager dont le numéro de téléphone est X. (Pages jaunes au lieu de simples pages blanches).



# Différents services

- Domain Name Service (**DNS**)
- Hesiod (Projet Athena au MIT)
- NIS (**Sun Yellow Pages**)
- Netinfo (**NeXT**)
- Banyan **VINES**
- **NT Domains** (avant Active Directory)
- X.500 (OSI)
- **LDAP** (simplification de X.500)
- **Active Directory** (Microsoft, basé sur LDAP)
- SLP, Jini, CORBA naming service, Portmap...



# DNS

- Avant 1987, chacun prenait une copie d'un monstrueux fichier /etc/hosts par ftp.
- Convertir les noms en adresses IP.
- Trouver le serveur de courriel pour un domaine.
- Informations sur chaque ordinateur.
- Alias pour services courants (www.polymtl.ca, ntp.polymtl.ca, ftp.polymtl.ca).
- Trouver le nom pour une adresse IP.



# Organisation de DNS

- Serveur avec: attributs pour les noms d'un domaine, noms et adresses des serveurs en autorité pour le domaine et pour les sous-domaines dont l'autorité a été déléguée, paramètres pour la zone comme le TTL (Time To Live).
- Chaque zone doit être servie par au moins deux serveurs en autorité qui présentent des modes de défaillance non coréllés.
- Le logiciel de serveur Bind peut être configuré en serveur primaire, secondaire ou cache seulement.
- La librairie client contacte par UDP les serveurs qui s'occupent de maintenir une cache en utilisant les valeurs de TTL.
- Peut avoir plusieurs IP (avec TTL très court) pour un nom, de manière à répartir des requêtes sur plusieurs ordinateurs.

## Serveurs DNS de départ

- Le service officiel est géré par le Internet Corporation for Assigned Names and Numbers (**ICANN**), autrefois sous le USA Department of Commerce et, depuis octobre 2016, après 18 ans de débats, sous une gouvernance plus neutre.
- **OpenNIC** offre des **serveurs racine alternatifs** avec beaucoup **plus de liberté** sur les **noms de domaines** disponibles (.bbs, .free, .geek, .libre, .neo, .null, .pirate...).
- Il existe d'autres racines DNS alternatives comme New Nations (.ko, .ku, .te, .ti, .uu...).



# Jini / Apache River

- Développé par Sun sous le nom Jini, puis transféré à Apache sous le nom de projet River.
- Message à tous pour trouver le serveur.
- Enregistrement des services offerts par chaque objet avec un TTL.
- Requêtes pour découvrir les services appropriés.
- Appariement des requêtes basée sur la hiérarchie de types Java.



# Service Location Protocol (SLP)

- Message à tous pour chercher un service (avec certains attributs).
- Les serveurs SLP qui connaissent un tel service répondent.
- SLP est souvent utilisé pour localiser des imprimantes et est supporté par des systèmes d'impression comme CUPS



## Le service de noms de OSI: X.500

- Wikipedia cite: *The original X.500 plan is unlikely ever to come to fruition*, mais LDAP s'en inspire.
- **DIT** (Directory Information Tree): hiérarchie de noms répartie sur plusieurs serveurs.
- **DIB** (Directory Information Base): noms et ensembles d'attributs pour chaque nom.
- Agent usager: accédé par les applications.
- Agent serveur: fournit les réponses, possiblement en faisant des requêtes à d'autres serveurs, ou en redirigeant le client vers un autre serveur.
- Les types des attributs sont définis avec ASN.1 (Abstract Syntax Notation). Un des attributs est le nom.
- Opérations: lire (chemin et liste des attributs désirés), chercher (préfixe de chemin, et expression booléenne de tests sur les valeurs des attributs).
- Interface d'administration pour la mise à jour.

# LDAP

- Utilisé dans les grosses entreprises à la place de NIS
- Version allégée de X500 basée sur TCP/IP plutôt que OSI.
- Utilise ASN.1 et BER.
- Connexion TCP port 389.
- Arbre de répertoires contenant des entrées, chaque entrée constituée d'attributs pouvant contenir plusieurs valeurs.
- Requêtes et réponses asynchrones (plusieurs requêtes de suite, réponses non ordonnancées).



# LDAP, opérations

- Start TLS: passer en mode encrypté.
- Bind: s'authentifier et spécifier la version du protocole.
- Search: effectuer une recherche dans le répertoire.
- Compare: vérifier si une entrée a une certaine valeur comme attribut.
- Add: ajouter une nouvelle entrée.
- Delete: effacer une entrée.
- Modify: modifier une entrée.
- Modify Distinguished Name (DN): renommer ou déplacer une entrée.
- Abandon: annuler une requête envoyée.
- Extended Operation: mécanisme pour extension.
- Unbind: fermer la connexion.



# LDAP, recherche

- **baseObject**: chemin absolu de l'entrée à laquelle commencer la recherche.
- **scope**: entrée, répertoire ou sous-arbre à chercher.
- **filter**: critères de recherche, combinaisons (et ou non) sur des relations (égal, commence par...) sur les valeurs des attributs.
- **derefAliases**: suivre ou non les alias.
- **attributes**: quels attributs retourner dans les résultats.
- **sizeLimit, timeLimit**: temps maximum et taille maximum des résultats à retourner.
- **typesOnly**: seulement retourner le type des attributs et non leur valeur.



## Discussion

- Le service DNS demeure le point d'entrée sur l'Internet. C'est un service **facile à offrir mais essentiel**. Il ne sert pratiquement que pour résoudre les adresses IP.
- **Les autres services de répertoires** (LDAP, Active Directory ou NIS) **sont internes à une organisation** et contiennent principalement la base de données des usagers (avec leur mot de passe, localisation du répertoire de fichiers, quotas...).
- Les **requêtes externes** passent souvent par une interface Web ou des Web Services (e.g. bottin du personnel de Polytechnique).
- Les **données** qui alimentent ces services peuvent souvent être **stockées dans les faits** dans une **base de donnée** conventionnelle.





# Services de temps et de coordination

## Module 8

### INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

① Le temps

② Les horloges logiques

③ Coordination

④ Conclusion



# Services de temps et de coordination

---

① Le temps

② Les horloges logiques

③ Coordination

④ Conclusion



## Le temps

- L'heure change d'un fuseau horaire à l'autre, d'une saison à l'autre et d'un pays à l'autre en raison de l'heure avancée.
- La durée des jours change à cause du frottement des marées et des courants du noyau terrestre.
- Le temps universel (UTC): une seconde est 9 192 631 770 périodes du cézium 133, nombre de secondes par jour ajustable, point de référence GMT (Greenwich Mean Time).
- L'horloge des ordinateurs dérive avec le temps comme les montres.
- En cas de retard, avancer d'un coup ou lentement.
- En avance, ralentir pour laisser le temps nous ratrapper... remonter dans le temps est dangereux.



# Problématique du temps dans les systèmes répartis

- Le temps est très important dans les systèmes répartis. On veut savoir avec précision quand un évènement est arrivé.
- Il n'y a pas d'horloge globale pour mesurer le temps à travers le système réparti, étant donné le délai pour envoyer par réseau une lecture de temps.
- Algorithmes de synchronisation servent à maintenir la cohérence des données réparties, éliminer les mises à jour redondantes, vérifier l'authenticité des requêtes envoyées au serveur, coordonner certaines opérations, tracer l'exécution du système...



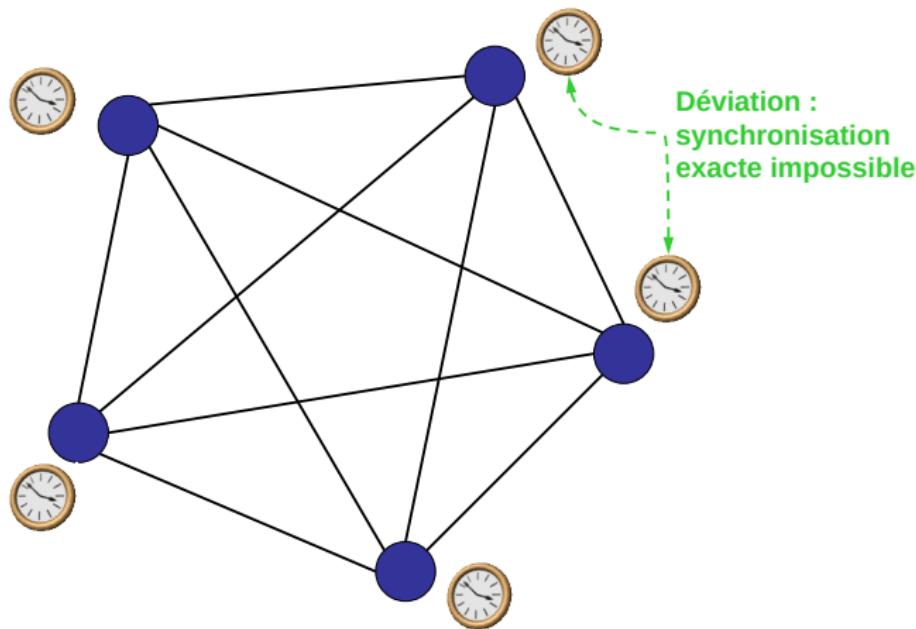
## Les horloges physiques

- Horloge d'ordinateur: cristal au quartz qui vibre à une fréquence précise (e.g. à plus ou moins  $10^{-6}$ ), avec une interruption après un certain nombre d'oscillations, ce qui génère un tic.
- L'horloge est obtenue par un calcul d'un temps initial (heure lue du circuit RTC mémorisé avec une pile): temps lu au démarrage + nombre de tics depuis ce temps initial x durée d'un tic.
- Les cristaux diffèrent d'un ordinateur à l'autre, ce qui cause une désynchronisation entre les systèmes multi-ordinateurs, appelée dérive des horloges (clock skew).



## Systèmes asynchrones

- On ne peut synchroniser parfaitement des systèmes asynchrones.



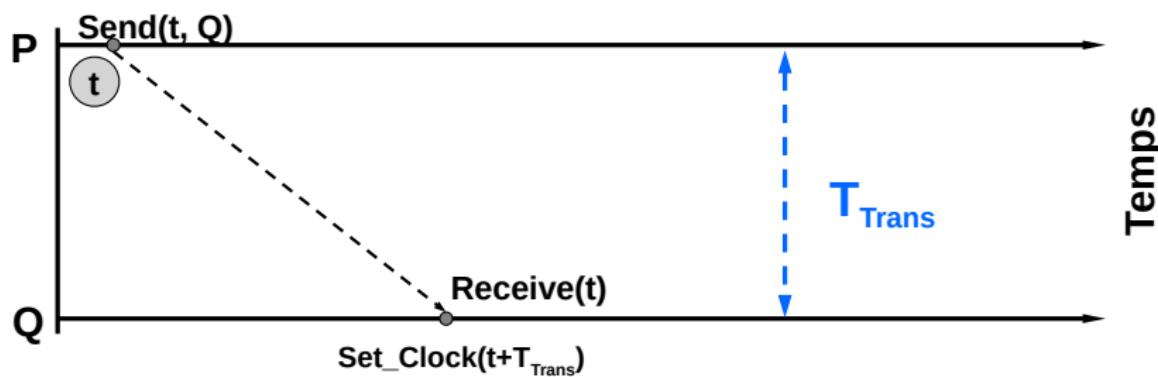
## Synchronisation

- Le temps de propagation est limité par la vitesse de la lumière.
- Délais liés à la préemption et au traitement des interruptions pour recevoir un message de synchronisation.
- Le GPS (Global Positionning System) donne l'heure juste à 1us près à travers le monde. La différence d'heure reçue entre les satellites est causée par la distance et donne la position par triangulation.
- Plusieurs ordinateurs (20 à 30) sont connectés à des horloges atomiques ou GPS et agissent comme serveurs de temps primaires. On compte quelques milliers de serveurs de temps secondaires qui prennent l'heure des serveurs primaires via l'Internet. La précision est typiquement de 30ms ou mieux 99% du temps.
- Avec une horloge calibrable, il est possible d'obtenir une précision de 1ms sur plusieurs heures.



# Synchronisation des systèmes synchrones

- Cas le plus simple:
  - Deux processus P et Q veulent synchroniser leurs horloges.
  - Le système est synchrone (délai fixe de transmission connu).



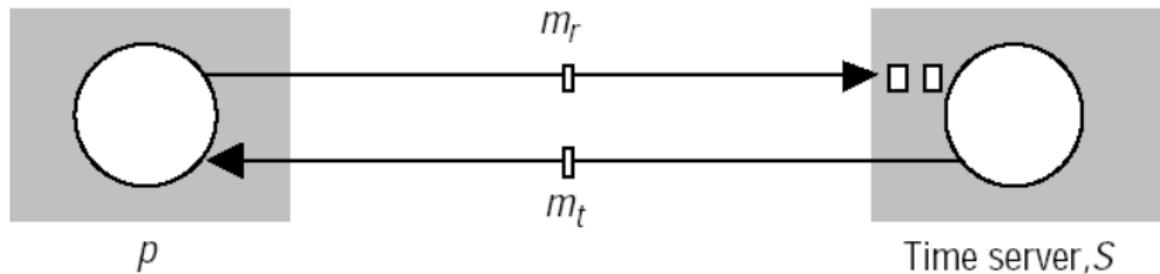
# Synchronisation des systèmes asynchrones

- Si le temps de transmission peut varier dans un intervalle  $U$ , utiliser une valeur à l'intérieur de  $U = [\text{MIN}, \text{MAX}]$ .

scénario	valeur choisie	déviation maximale
1	$t + \text{MIN}$	$U$
2	$t + \text{MAX}$	$U$
3	$t + (\text{MAX} + \text{MIN}) / 2$	$U / 2$

## Synchronisation par la méthode de Cristian

- Méthode basée sur la **synchronisation externe** ;
- Elle utilise un **serveur central de temps** (Time server) qui est connecté à un périphérique recevant des signaux d'une source UTC ;
- Le processus P peut **mesurer le temps de réponse** à sa requête,  $T_r$ , **et affecte à son horloge**  $t + \frac{T_r}{2}$ .



## Synchronisation par l'algorithme de Berkeley

- Un coordonnateur (**maître**) **interroge** les autres ordinateurs (**esclaves**) dont les horloges sont **à synchroniser**.
- Les **esclaves envoient** la valeur de **leur horloge au maître**.
- Le **maître estime** leur temps local **en observant le délai d'un aller-retour**.
- Il **calcule la moyenne des valeurs reçues et de sa propre valeur**.
- Le **maître renvoie** les **ajustements nécessaires aux esclaves**.
- Si le **maître tombe en panne**, **un autre est élu** comme maître.



# Network Time Protocol (NTP)

- Défini en 1995 dans le RFC 958.
- Le service NTP est fourni par un réseau de serveurs localisés à travers l'Internet.
- Serveurs primaires connectés directement à une source de temps UTC.
- Serveurs secondaires synchronisés avec les serveurs primaires.



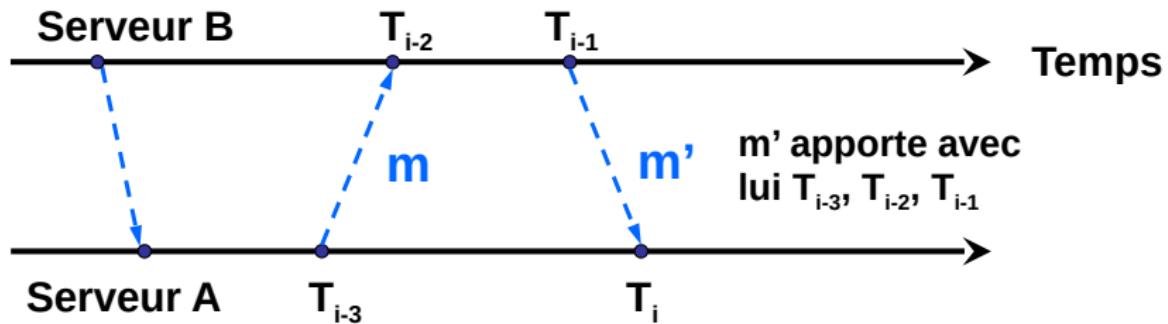
# Modes de synchronisation des serveurs

- Diffusion sélective - multicast:
  - Utilisé dans des LANs à haut débit.
  - Un ou plusieurs serveurs diffusent le temps aux démons s'exécutant sur d'autres ordinateurs connectés au LAN.
  - Les démons qui reçoivent les messages affectent à leur horloge les valeurs reçues en considérant un faible délai.
- Appel de procédure:
  - Un serveur accepte les requêtes des autres ordinateurs ;
  - Renvoie une réponse contenant une estampille (valeur de l'horloge locale).
  - Offre une meilleure précision.



## Calcul du décalage

- Un processeur demande à l'autre le temps par envoi de messages.
- Les temps d'envoi et de réception sont notés pour en tenir compte dans le calcul de décalage.



## Calcul du décalage

- Soit le temps de transmission  $t$  de A vers B et  $t'$  de B vers A.
- $T_{i-2} = T_{i-3} + t + d$  et  $T_i = T_{i-1} + t' - d$
- Soit  $a = T_{i-2} - T_{i-3}$  et  $b = T_{i-1} - T_i$
- $imprecision = t + t' = a - b$  avec  $t > 0$  et  $t' > 0$
- $ajustement = \frac{a+b}{2}$
- $d = ajustement + \frac{t'-t}{2}$
- $\frac{a+b}{2} - \frac{a-b}{2} \leq decalage \leq \frac{a+b}{2} + \frac{a-b}{2}$
- $decalage = \frac{a+b}{2} \pm \frac{a-b}{2}$
- On peut envoyer plusieurs messages et retenir ceux dont  $a$  et  $b$  sont les plus petits. Normalement seuls  $t$  et  $t'$  varient sur une courte période, en les minimisant on réduit l'imprécision.

# Services de temps et de coordination

---

1 Le temps

2 Les horloges logiques

3 Coordination

4 Conclusion



## Etat des processus et systèmes répartis

- Collection de N processus  $P_i$  où  $i = 1, 2, 3, \dots, N$ .
  - Les processus sont indépendants;
  - Il n'y a pas de mémoire partagée entre les processus.
- Si: état du processus  $P_i$ .
  - L'état = valeurs des variables, des ressources ou des objets locaux utilisés par le processus.
- La communication entre les processus ne se fait que par transmission de messages.
- Sur un processus  $P_i$ , le passage de l'état  $S_i$  à l'état  $S'_i$  est directement associé à l'ensemble des actions pouvant modifier l'état d'un processus (envoi et réception de message).



## Evènements dans un système réparti

- Un évènement  $e$  est l'occurrence d'une seule action.
- Un évènement modifie l'état d'un processus.
  - L'ordre total unique des évènements est noté :  $\rightarrow$
  - $e \rightarrow e'$  si l'événement  $e$  est survenu avant  $e'$  dans  $P_i$
- L'historique de  $P_i$  est une série d'évènements survenus dans  $P_i$  et ordonnés par la relation  $\rightarrow$ 
  - $History(P_i) = h_i = < e_i^0, e_i^1, e_i^2, \dots >$

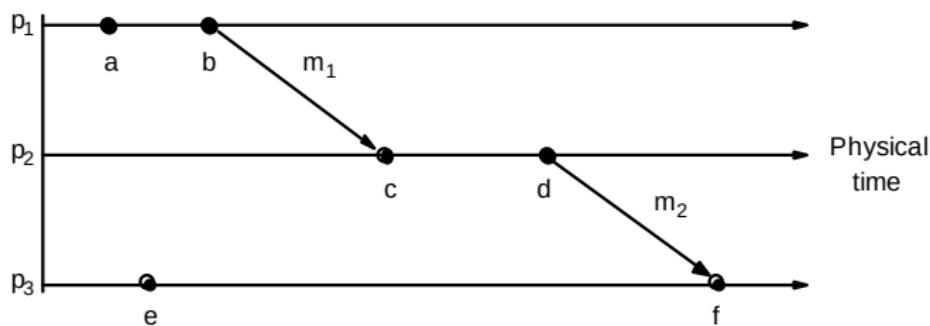


# Horloges logiques

- Souvent, seul l'ordre entre les événements importe.
- Dans un même processus, les événements sont numérotés avec un compteur.
- Lors d'un envoi de message, la correspondance est faite entre le décompte de l'envoyeur et celui du receveur ( $Te < Tr$ ).
- En l'absence de messages, on ne peut rien dire sur l'ordre réel entre deux événements dans des processus différents. On les considère concurrents.



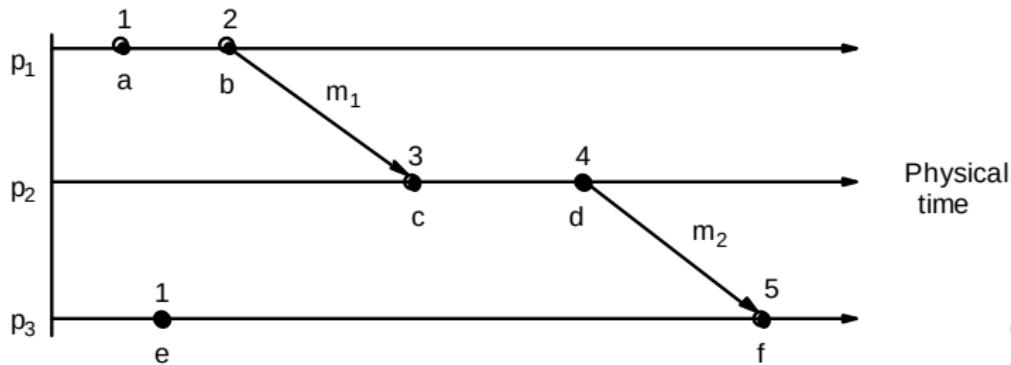
## Exemple de relations entre évènements



- $a \rightarrow b$  (puisque les deux se suivent dans  $p_1$ ).
- $b \rightarrow c$  (puisque  $b = \text{send}(m_1)$  et  $c = \text{receive}(m_1)$ )
- $c \rightarrow d$  (puisque les deux se suivent dans  $p_2$ )
- $d \rightarrow f$  (puisque  $d = \text{send}(m_2)$  et  $f = \text{receive}(m_2)$ )
- $a \rightarrow f$  (puisque  $a \rightarrow b \rightarrow c \rightarrow d \rightarrow f$ )
- $e$  et  $a$  sont des évènements concurrents, sans ordre démontrable

## Exemple d'horloge logiques

- L'horloge logique est incrémentée de 1 par rapport à la valeur précédente à chaque relation (séquence ou réception de message).
- Si un évènement est postérieur à un autre, sa valeur d'horloge logique est plus grande, l'inverse n'est pas nécessairement vrai (événements concurrents).

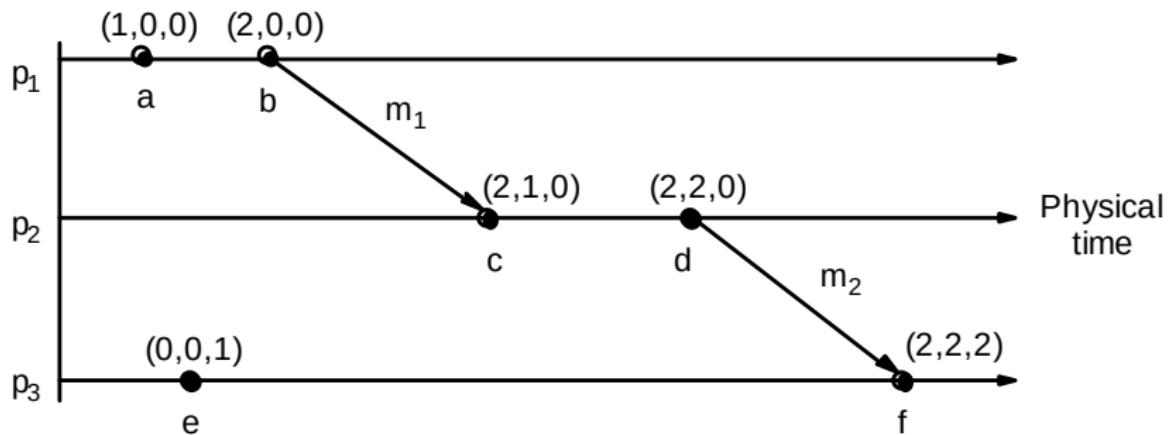


## Vecteurs de compteurs d'évènements

- Vecteur de N entiers initialisés à 0 dans chacun des N processus.
- A chaque évènement dans le processus i,  $Vi[i]$  est incrémenté.
- A chaque message envoyé par le processus i,  $Vi$  est inclus.
- Lorsque le processus i reçoit un vecteur dans un message, il prend pour chaque entrée de son vecteur le maximum entre l'entrée présente et celle reçue (fusion des vecteurs).
- Chaque entrée dans le vecteur du processus i indique le dernier événement dans un autre processus qui peut avoir influencé le processus i (lien de causalité).



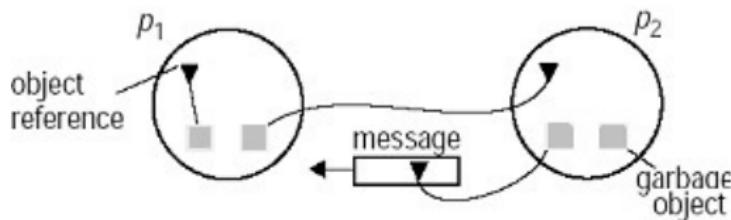
## Exemple de vecteurs de compteurs d'évènements



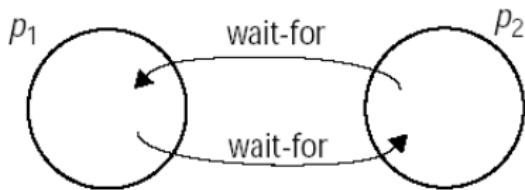
- $V = V'$ ssi  $V[j] = V'[j], \forall j = 1 \dots N$
- $V < V'$ ssi  $V[j] \leq V'[j], \forall j = 1 \dots N$  et  $V \neq V'$
- $e \rightarrow e'$ ssi  $V(e) < V(e')$

## État global d'un système réparti:

- Pour vérifier si une des propriétés particulières est vérifiée ou non durant son exécution.
- Ramasse-miettes (garbage collection), un objet doit être supprimé s'il n'est plus rejoignable.



- Interblocage réparti s'il y a un cycle dans le graphe d'attente de plusieurs processus.



## État global d'un système réparti:

- L'état global est important mais difficile à obtenir sans tout figer.
- Une propriété est stable si une fois atteinte elle demeure vraie (e.g. interblocage).
- Un système est sécuritaire pour une propriété si cette propriété reste vérifiée indépendamment de l'ordre dans lequel les évènements non reliés causalement surviennent (e.g. jamais d'interblocage).
- Un système est vivace pour une certaine propriété si elle devient éventuellement vraie, indépendamment de l'ordre dans lequel les évènements non reliés causalement surviennent (e.g. une demande de verrou doit être satisfaite).



## Etat global d'un système réparti

- Un système E est composé de N processus  $P_i$  où  $i = 1, 2, 3, \dots, N$ ;
- L'historique d'un processus est:  
$$History(P_i) = h_i = < e_i^0, e_i^1, e_i^2, \dots >$$
- L'historique fini est:  $h_i^k = < e_i^0, e_i^1, e_i^2, \dots, e_i^k >$
- L'historique global du système est:  $H = h_1 \cup h_2 \cup \dots \cup h_N$
- L'état global du système E est:  $S = (S_1, S_2, \dots, S_N)$



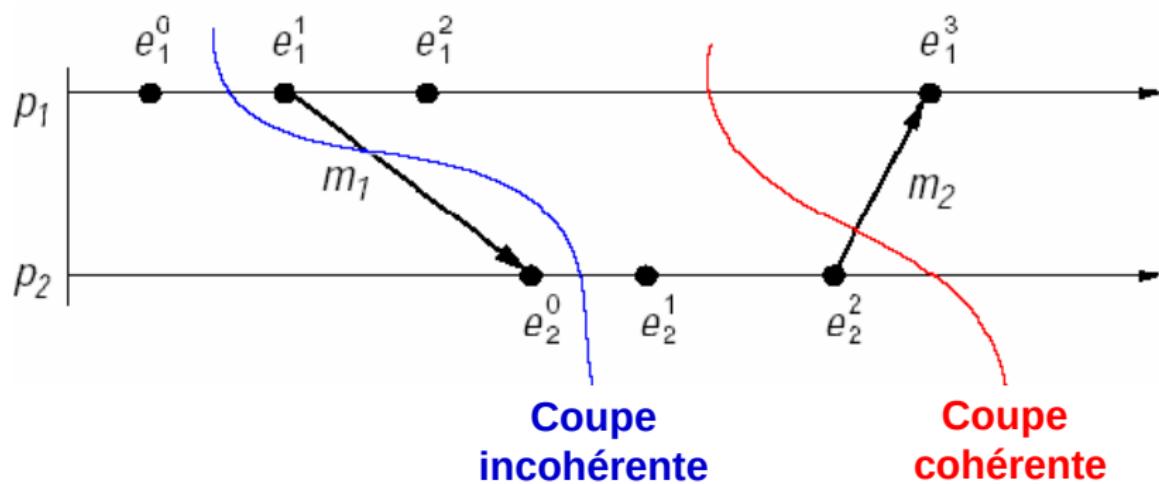
## Cliché cohérent de l'état global

- Une coupe (**cut**) est un **sous-ensemble** de l'état global.
- Une coupe  $C$  est cohérente si pour chaque événement  $e$  qu'elle inclut, tous les événements qui sont survenus avant la coupe sont inclus
- $\forall e \in C, f \rightarrow e \Rightarrow f \in C$



## Exemple de coupe dans l'historique d'état

- Un état global cohérent correspond à une coupe cohérente.



## Algorithme pour l'obtention d'un cliché cohérent

- Un processus mémorise son état, envoie un message demandant la mémorisation aux autres processus connectés en sortie, et enregistre tous les messages reçus d'autres processus qui n'ont pas encore mémorisé leur état.
- Un processus qui reçoit l'ordre de mémorisation fait de même.
- A la fin, l'état global est l'état de chaque processus plus les messages mémorisés (déjà envoyés avant que le processus d'origine n'ait mémorisé son état mais reçus après que le processus courant ait mémorisé son état).



## Calcul d'un cliché cohérent possible

- Il est possible d'enregistrer les événements (changements d'état) de tous les processus. Chaque processus envoie une trace à un moniteur central avec pour chaque événement un vecteur de compteurs d'événements.
- Un état global est défini par un numéro de dernier événement pour chaque processus impliqué. Cet état est cohérent si pour chaque dernier événement, chaque entrée des vecteurs de compteurs d'événements est inférieure ou égale à la même entrée pour le dernier événement inclus pour le processus correspondant.  $V(s_i)[i] \geq V(s_j)[i] \quad i, j = 1, 2, \dots, N$
- Dans un système synchrone, un état est cohérent si les différences entre les temps des événements sont inférieures à l'imprécision des horloges.



## Propriétés d'un cliché cohérent possible

- Construire un treillis d'états globaux possibles (cohérents).
- Traverser le treillis un niveau à la fois.
- Si un des états globaux possibles rejoint a une propriété, cette propriété est possible.
- Si on arrête à chaque état global possible pour lequel la propriété est vraie et qu'on ne puisse ainsi arriver à l'état final, cette propriété est nécessairement vraie.



# Services de temps et de coordination

---

1 Le temps

2 Les horloges logiques

3 Coordination

4 Conclusion

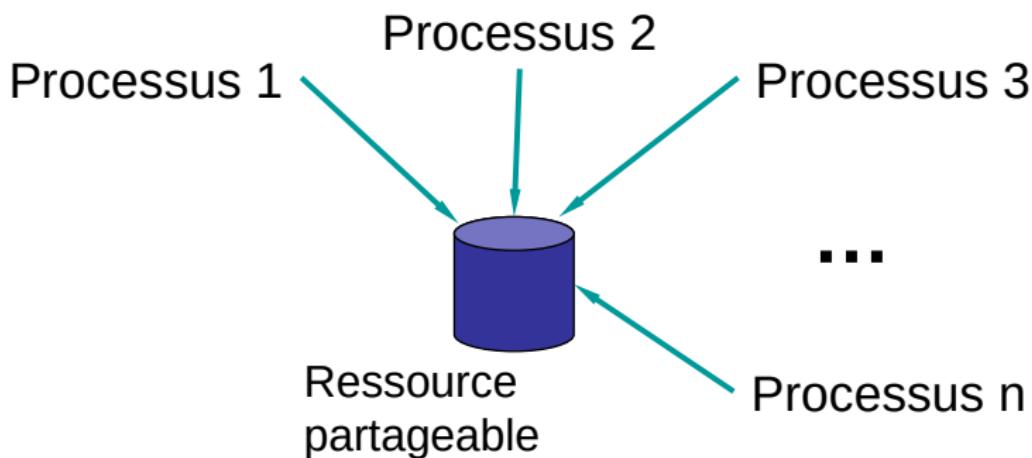


# Coordination et temps

- Un des **problème** fondamentaux des systèmes répartis : le **consensus**!
  - **Comment faire pour mettre d'accord plusieurs processus indépendants?**
  - **Comment faire pour coordonner leurs actions?**
- **Comment faire pour résoudre ce problème?**
  - Une solution naïve est d'**implémenter le modèle maître-esclave**.
- Est-ce que le problème du consensus est le même pour les systèmes synchrones et asynchrones?



## Exclusion mutuelle répartie



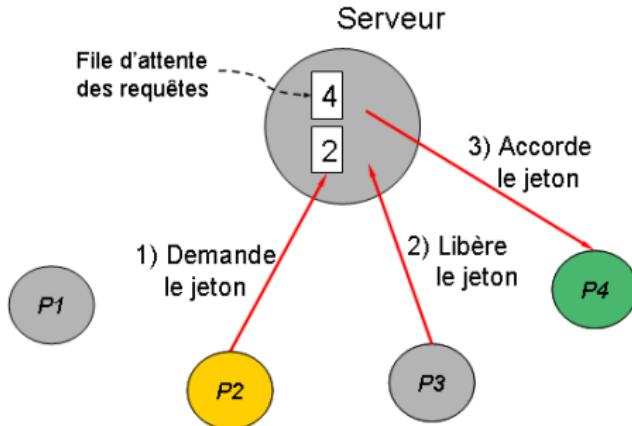
- Exclusion mutuelle est nécessaire pour :
  - Prévenir les interférences ;
  - Assurer la cohérence en cas d'accès simultanés aux ressources.

## Synchronisation par exclusion mutuelle répartie

- **Sûreté:** un seul client à la fois obtient le verrou.
- **Vivacité:** personne n'est laissé pour compte, chaque client voit éventuellement sa requête satisfaite.
- **Ordre:** premier arrivé, premier servi, si un client A fait une demande, envoie un message à B, et B fait une demande, l'ordre logique dit que la demande de A arrive avant celle de B. Elle devrait être servie en premier.

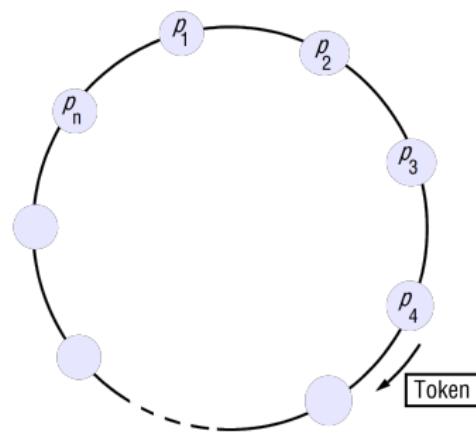


## Serveur central



- Un client envoie une demande de verrou, le serveur attend que le verrou soit libre, le donne au client, et le client le relâche éventuellement, ce qui permet au serveur de le donner à un autre client.
- Si le serveur est en panne, élection d'un nouveau serveur (majorité de clients), et vérification de tous les clients pour voir qui possède des verrous ou a soumis une requête (problème si clients non rejoignables en raison de division du réseau).
- Exemple: serveur lockd sous NFS.

## Anneau à jeton



- Le verrou passe de processus en processus constamment.
- Inutile lorsque personne ne requiert le verrou.
- **Problème dès qu'un client fait défaut.**
- Ne respecte pas premier arrivé, premier servi.

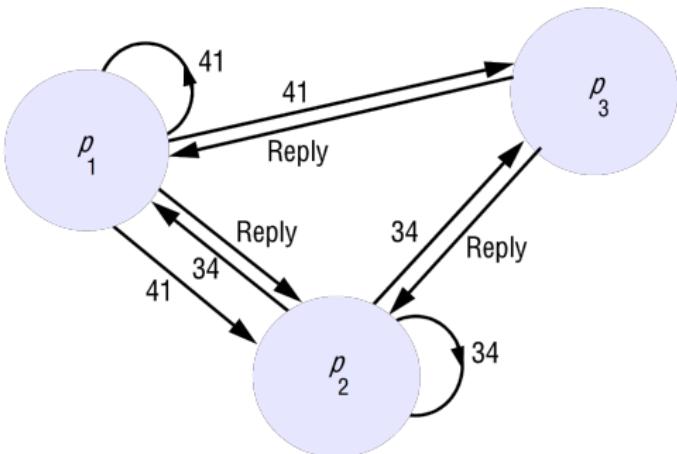
## Exclusion par envoi à tous

- Envoyer une requête à tous.
- Recevoir le O.K. de tous.
- Si on a le verrou, attendre d'en avoir terminé avant de répondre O.K.
- Si on veut le verrou et notre demande est antérieure (estampille de temps), attendre le verrou et d'en avoir fini avant de répondre O.K.
- Demande  $n$  messages (ou  $2n - 2$  sans message à tous).
- Tous les clients doivent être actifs.
- Moins bon que serveur central.

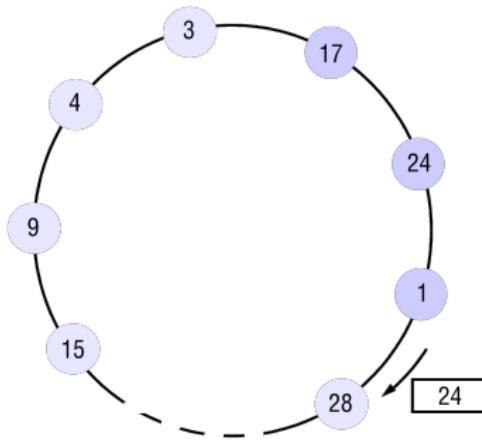


## Exclusion par envoi à tous (suite)

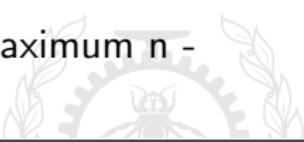
- $p_3$  ne veut pas entrer dans la section critique
- $p_1$  et  $p_2$  demandent accès à la section critique en même temps:
  - requête de  $p_1$  à  $T = 41$
  - requête de  $p_2$  à  $T = 34$
- Processus:
  - $p_3$  répond immédiatement aux requêtes
  - $p_2$  reçoit la requête de  $p_1$ , il voit que  $T(p_2) < T(p_1)$  et ne répond pas (hold  $p_1$ )
  - $p_1$  reçoit la requête de  $p_2$ , il voit que  $T(p_2) < T(p_1)$  et répond immédiatement
  - $p_2$  reçoit la réponse de  $p_1$  et entre dans la section critique; quand il la quitte, il répond à  $p_1$
  - $p_1$  reçoit la réponse de  $p_2$  et entre dans la section critique



## Election en anneau

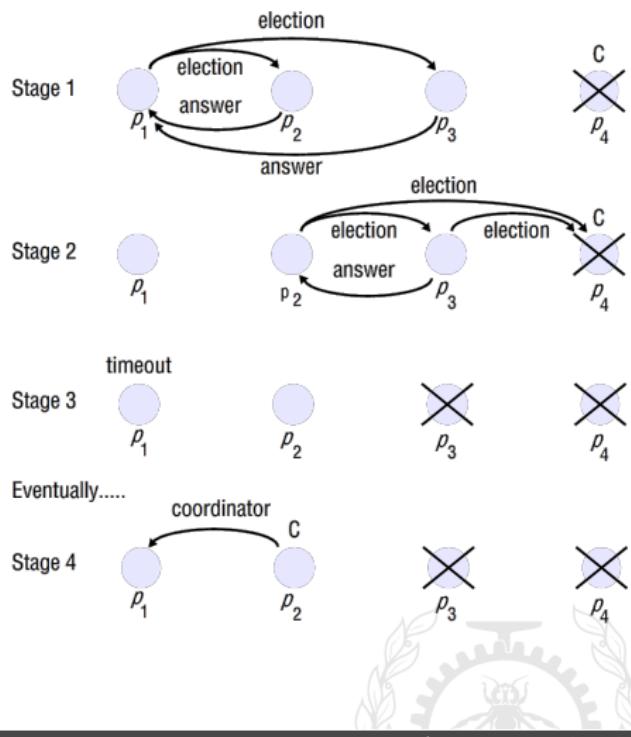


- Un participant envoie son message d'élection avec son ID.
- Le prochain participant envoie  $\max(\text{ID reçu}, \text{ID})$  à son voisin.
- Si un participant reçoit un message avec son ID, il se déclare élu et propage la nouvelle.
- Ceci prend un maximum de  $3n - 1$  messages (ID maximum n - 1, se découvrir élu n, propager la nouvelle n).



## Election hiérarchique (*Bully algorithm*)

- Lorsqu'un nouvel ordinateur est connecté, ou si le coordonnateur ne peut être rejoint, déclencher une élection.
- Envoyer un message d'élection à ceux de plus haute priorité.
- Pas de réponse, il se proclame coordonnateur et le signale à tous.
- Une réponse, il attend un message de proclamation qui devrait suivre.
- Demande  $n - 1$  messages dans le meilleur des cas,  $n \times n$  si tous les processus déclenchent une élection en même temps.



# Services de temps et de coordination

---

- 1 Le temps
- 2 Les horloges logiques
- 3 Coordination
- 4 Conclusion



## Conclusion

- Chaque ordinateur a son horloge asynchrone et la synchronisation de temps est imparfaite.
- Pour plusieurs applications, un ordre relatif, une horloge logique, suffit.
- Exclusion mutuelle, élection ou consensus, les approches les plus distribuées ou les plus "démocratiques" ne sont pas nécessairement les plus efficaces.
- Attention aux pannes multiples, répétées, au partitionnement de réseau...



# Résumé

---

① Le temps

② Les horloges logiques

③ Coordination

④ Conclusion





# Cohérence et réPLICATION pour les données répartIES

Module 9

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① Introduction
- ② Les transactions
- ③ Contrôle de la concurrence
- ④ Transactions imbriquées
- ⑤ Récupération en cas de panne
- ⑥ Les transactions réparties
- ⑦ Conclusion



# Cohérence et réPLICATION pour les données réPARTIES

- ① Introduction
- ② Les transactions
- ③ Contrôle de la concurrence
- ④ Transactions imbriquées
- ⑤ Récupération en cas de panne
- ⑥ Les transactions réparties
- ⑦ Conclusion



# Cohérence et réPLICATION de données réPARTIES

- **Cohérence** (consistency): refléter sur la copie d'une donnée les modifications intervenues sur d'autre copies de cette donnée.
- En anglais:
  - *Coherence*, en cas de plusieurs modifications concurrentes sur une donnée, la même valeur finale rejoindra éventuellement toutes les copies.
  - *Consistency*, l'ordre entre plusieurs modifications concurrentes sur différentes données sera vu de manière cohérente (plus ou moins stricte selon le modèle) par les clients de toutes les copies
- Les modèles de cohérence sont utilisés pour les données réparties ou répliquées: bases de données, systèmes de fichiers, cache...



## Modèles de cohérence

- Cohérence **stricte**: la mise à jour est vue **en même temps** sur toutes les copies (e.g. invalider la valeur actuelle sur toutes les copies, propager la nouvelle valeur sur toutes les copies).
- Cohérence **séquentielle**: l'**ordre** des modifications **est le même** sur toutes les copies.
- Cohérence **causale**: l'**ordre** des modifications **reliées causalement** (e.g. même origine) **est le même** sur toutes les copies.



# Transactions

- Dans la plupart des cas, on doit s'assurer de la cohérence d'un **groupe de modifications** (transaction) plutôt qu'une modification seule (e.g. transactions dans les bases de données, opérations sur les données et métadonnées pour écrire dans un fichier).
- La **cohérence** sera donc étudiée principalement **sous l'angle des transactions**.
- **Transaction:** séquence d'opérations que le serveur exécute d'une manière **atomique**, même en présence d'accès simultanés par plusieurs clients et de pannes.
- **Assurer le partage sécuritaire et cohérent des données des serveurs par plusieurs clients.**



## Synchronisation simple (sans transaction)

- Opérations atomiques au niveau du serveur.
- Utilisation de plusieurs fils d'exécution au niveau du serveur.
- Les opérations des clients peuvent s'exécuter simultanément.
- Un verrou assure qu'un seul thread accède à un objet à un instant donné.



## Exemple: Banque

- Chaque compte est représenté par un objet.
- Interface Account: fournit les opérations pouvant être exécutées par les clients.
  - deposit(amount): deposit amount in the account
  - withdraw(amount): withdraw amount from the account
  - getBalance() → amount: return the balance of the account
  - setBalance(amount): set the balance of the account to amount
- Interface Branch: fournit les opérations pour les succursales de cette banque
  - create(name) → account: create a new account with a given name
  - lookUp(name) → account : return a reference to the account with the given name
  - branchTotal() → amount: return the total of all the balances at the branch

## Exemple: transaction bancaire

- Transaction T:

```
a.withdraw(100);  
b.deposit(100);  
c.withdraw(200);  
b.deposit(200);
```

- Transaction atomique: deux propriétés à satisfaire
  - Tout ou rien: soit la transaction est exécutée (ses effets sont alors permanents), ou bien elle n'a aucun effet (même en cas de panne du serveur)
  - Isolation: chaque transaction est exécutée sans interférence avec d'autres transactions (effets intermédiaires invisibles aux autres transactions)



# Cohérence et réPLICATION pour les données réPARTIES

- 1 Introduction
- 2 Les transactions
- 3 Contrôle de la concurrence
- 4 Transactions imbriquées
- 5 Récupération en cas de panne
- 6 Les transactions réparties
- 7 Conclusion



## Propriétés des transactions (ACID)

**Atomicité:** les effets d'une transaction ne sont rendus visibles que si toutes ses opérations peuvent être réalisées. Sinon, aucun effet n'est produit par la transaction

**Cohérence:** les effets d'une transaction doivent satisfaire les contraintes de cohérences de l'application (la transaction transforme l'état cohérent du système en un autre état cohérent)

**Isolation:** les effets d'une transaction sont identiques à ceux qu'elle pourrait produire si elle s'exécutait seule dans le système

**Durabilité:** les effets d'une transaction terminée ne peuvent être détruits ultérieurement par une quelconque défaillance

## Pannes inévitables mais objets récupérables

---

- Objets pouvant être récupérés suite à une panne de serveur.  
Le serveur doit stocker suffisamment d'information en mémoire stable (disque), possiblement redondante.
- Un journal sur disque permet de lister les éléments d'une transaction avant de commettre ou d'annuler, et permet en cas de panne de savoir ce qui était en train d'être fait.
- La synchronisation des transactions pour les sérialiser est une méthode permettant l'isolation.



## Coordonnateur de transaction

- **Création:** le coordonnateur crée la transaction et lui affecte un identificateur unique qui sera associé à toutes les opérations de cette transaction.
- **Opérations:** le coordonnateur maintient la liste des objets ou processeurs impliqués dans les différentes opérations de la transaction.
- **Fin:** la transaction est confirmée (*commit*) ou annulée (*abort*) par le client, le coordonnateur accepte ou annule une transaction confirmée et annule une transaction annulée.
- Une transaction est complétée suite à la coopération entre le programme du client, les objets récupérables et le coordonnateur.



# Les divers scénarios d'une transaction

Confirmée	Annulée (client)	Annulée (serveur)
Begin Transaction	Begin Transaction	Begin Transaction
opération	opération	opération
opération	opération	opération
opération	opération	opération
:	:	:
Commit Transaction	Abort Transaction	Commit Transaction
<i>Success: accept</i>	<i>Success: abort</i>	<i>Server error: abort</i>



## En cas de panne du client

- Le client redémarre et a oublié toute transaction qu'il avait initiée et qui était en cours.
- Si le serveur n'a plus de nouvelles d'un client après un certain délai (timeout) il annule la transaction.
- Si le client n'était pas en panne mais son réseau était trop lent et le délai expire, le serveur lui refusera la transaction puisqu'elle est annulée (ou répondra que la transaction n'existe plus).



## En cas de panne du serveur

- Le **serveur redémarre** et a **oublié toutes les transactions** en cours (non complétées) qui sont conséquemment annulées.
- Une **procédure de recouvrement** (e.g. lecture du journal des transactions) **permet** au serveur **de revenir à un état cohérent** des objets (valeurs produites par les plus récentes transactions complétées).
- Un client ne recevant **plus de réponse du serveur** après un certain délai **abandonne sa requête**.
- Si le **serveur a redémarré**, le client se fera répondre que la **transaction n'existe plus**.
- Si le **serveur plante avant d'avoir répondu** au client si la transaction est acceptée, le **client reste dans l'incertitude**. Il **demandera le statut** de la transaction après le redémarrage et **se fera confirmer** qu'elle est acceptée ou répondre que la transaction n'existe plus.

## Le problème de la cohérence

- Une banque contient deux comptes avec les soldes  $A = \$200$  et  $B = \$200$ .
- Un client 1 fait une transaction V: le transfert de  $\$100$  de A vers B.
  - Retirer  $\$100$  de A
  - Ajouter  $\$100$  à B
- Un client 2 fait simultanément une transaction W: calculer la somme des comptes de la banque.
- Sans synchronization, la somme peut donner  $\$300$  (incorrect) ou  $\$400$ .



## Sérialisation des transactions

- Une transaction exécutée seule ne présente pas de problème.
- L'exécution sérialisée des transactions, une par une, permet aussi d'assurer les propriétés requises, T<sub>1</sub>, T<sub>2</sub>, ..., T<sub>n</sub>.
- Entrelacement des transactions équivalent à une exécution en série, pour un ordre des transactions quelconque, est aussi acceptable.



## Contraintes pour sérialiser les transactions

---

- Deux opérations sont en conflit si leur effet combiné dépend de l'ordre dans lequel elles sont exécutées.
- Lecture-Lecture, pas de conflit.
- Lecture-Ecriture, conflit, le résultat de la lecture est affecté par l'écriture de la variable à lire.
- Ecriture-Ecriture, conflit, le résultat des lectures subséquentes dépend de l'ordre des deux écritures sur une même variable.



## Sérialisation des transactions: exemple

- Deux transactions sont sérialisables si toutes les paires d'opérations en conflit des deux transactions sont exécutées dans le même ordre sur tous les objets qu'elles accèdent

Transation T	Transaction U
$x = \text{read}(i)$	
$\text{write}(i, 10)$	
	$y = \text{read}(j)$
	$\text{write}(j, 30)$
$\text{write}(j, 20)$	
	$z = \text{read}(i)$

- Cet ordonnancement n'est pas équivalent à une exécution sérielle, il faudrait 1) ou 2):
  - 1) T accède à i avant U et T accède à j avant U
  - 2) U accède à i avant T et U accède à j avant T



# Cohérence et réPLICATION pour les données réPARTIES

- ① Introduction
- ② Les transactions
- ③ Contrôle de la concurrence
- ④ Transactions imbriquées
- ⑤ Récupération en cas de panne
- ⑥ Les transactions réparties
- ⑦ Conclusion



# Approches de contrôle de la concurrence

- Le protocole doit produire un ordonnancement équivalent à une exécution en série.
- Le verrouillage:
  - Utilisé par la plupart des systèmes.
  - Chaque objet est verrouillé par la première opération qui y accède.
  - L'objet est déverrouillé au *commit* ou *abort*.
- Contrôle optimiste de la concurrence:
  - On suppose l'absence de conflit.
  - Chaque transaction est exécutée jusqu'à la fin sans bloquer.
  - Avant d'être complétée, le serveur vérifie si elle n'a pas exécuté des opérations sur des objets qui sont en conflit avec les autres opérations des autres transactions simultanées.
  - Si de tels conflits existent, le serveur abandonne la transaction et le client peut la redémarrer.



# Recouvrement après l'abandon d'une Transaction

- Le serveur doit:
  - Sauvegarder les effets des transactions complétées
  - Assurer qu'aucun effet des transactions abandonnées ne soit stocké
- Problèmes associés à l'abandon de transactions, si mal géré:
  - Problème de lectures contaminées (*dirty reads*)
  - Problème d'écritures prématuées (*premature writes*)



# Problème de lectures contaminées

Transaction T	Transaction U	Compte A
Begin Transaction		\$100
$b = \text{getBalance}(A)$		\$100
$\text{setBalance}(A, b + 10)$		\$110
	Begin Transaction	\$110
	$b = \text{getBalance}(A)$	\$110
	$\text{setBalance}(A, b + 20)$	\$130
Abort Transaction		\$100
	Commit Transaction	\$130



# Problème d'écritures prématurées

Transaction T	Transaction U	Compte A
Begin Transaction		\$100
setBalance(A,\$105)		\$105
	Begin Transaction	\$105
	setBalance(A, \$110)	\$110
Abort Transaction		
(restore A)		\$100
	Abort Transaction	
	(restore A)	\$105



## Se prémunir contre les écritures prématurées

- Chaque transaction possède ses propres versions provisoires(tentative) des objets qu'elle met à jour.
- Les écritures sont faites sur les versions provisoires.
- Les lectures d'autres transactions sont faites à partir des versions provisoires (correct si finit après), ou à partir des versions permanentes (correct si finit avant ou est annulé).
- Les versions provisoires sont transférées dans les versions permanentes des objets seulement lorsque la transaction est complétée, en une seule étape.
- Pendant le transfert, les autres transactions n'ont pas accès aux objets modifiés.
- Si les objets modifiés ne sont pas verrouillés (contrôle optimiste), il faut vérifier la cohérence des transactions concurrentes.



# Cohérence et réPLICATION pour les données réPARTIES

- 1 Introduction
- 2 Les transactions
- 3 Contrôle de la concurrence
- 4 Transactions imbriquées
- 5 Récupération en cas de panne
- 6 Les transactions réparties
- 7 Conclusion



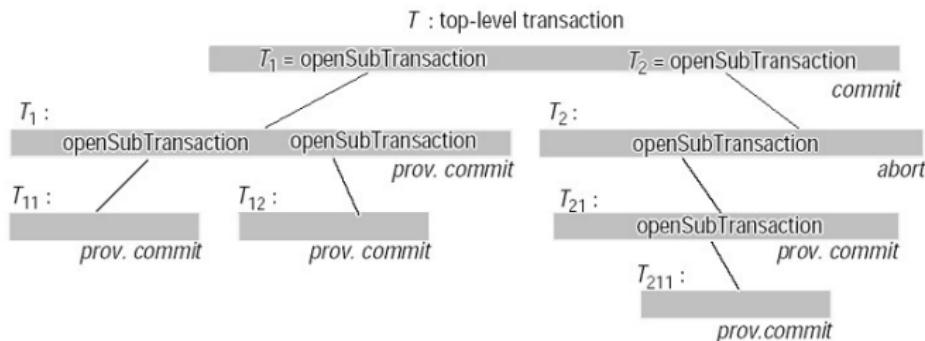
# Transactions Imbriquées

- Transaction contenue dans une autre et qui peut elle-même contenir des transactions imbriquées
- Une transaction imbriquée à un niveau de la hiérarchie n'est complétée que si son parent complète son exécution
- Accroit le niveau de concurrence en permettant aux transactions imbriquées d'un même niveau dans la hiérarchie d'être exécutées simultanément
- Permet aux transactions imbriquées d'être complétées ou abandonnées indépendamment les unes des autres



# Transactions Imbriquées

- Sous-transactions du même niveau, e.g. T1 et T2, peuvent s'exécuter simultanément, mais leurs accès aux objets partagés sont sérialisés (verrouillage)
- Si une sous-transaction est abandonnée: le parent peut choisir une autre sous-transaction pour compléter sa tâche



## Verrouillage

- Un verrou permet de résERVER l'accès unique à un objet (ressource).
- Si un client demande l'accès à un objet déjà verrouillé par une autre transaction, il doit attendre son déverrouillage.
- Verrou de lecture: avant une opération de lecture d'un objet, le serveur pose un verrou (non exclusif) de lecture sur l'objet.
- Verrou d'écriture: avant une opération d'écriture d'un objet, le serveur pose un verrou (exclusif) d'écriture sur l'objet.



## Exemple de verrouillage (suite)

Transaction T	Verrous T	Transaction U	Verrous U
Begin Transaction			
bal = getBalance(B)	Lock B		
setBalance(B, bal*1.1)		Begin Transaction	
withdraw(A, bal/10)	Lock A	bal = getBalance(B)	wait for B
Commit		⋮	⋮
Success	Unlock A, B	⋮	⋮
		bal = getBalance(B)	Lock B
		setBalance(B, bal*1.1)	
		withdraw(C, bal/10)	Lock C
		Commit	
		Success	Unlock B, C

## Granularité de verrouillage

- Verrou **complet ou séparé** lecture et écriture.
- Verrou sur **un seul objet**.
- Verrou sur **un groupe d'objets** (e.g. page sur disque).
- Verrou **global** (sérialiser complètement les transactions).



# Protocoles de verrouillage

- Verrouillage à 2 phases:
  - 1ère phase (*growing phase*): la transaction acquiert de nouveaux verrous.
  - 2ème phase (*shrinking phase*): la transaction libère ses verrous.
  - Aucun verrou ne peut être posé si un verrou a été levé.
- Verrouillage strict à 2 phases:
  - Verrouillage à 2 phases dans lequel les verrous sont libérés seulement lorsque la transaction est complétée ou abandonnée.
  - Prévient les lectures contaminées et les écritures prématurées
  - Il faut attendre le *commit* ou *abort* mais pas nécessairement la fin de la transaction pour libérer les verrous de lecture.



## Le gestionnaire de verrous

- Maintient la table de verrous pour les objets d'un serveur.
- Entrée pour chaque verrou:
  - objet associé au verrou;
  - type de verrou (écriture ou lecture);
  - identificateurs des transactions détenant le verrou (un seul pour écriture);
  - liste des transactions en attente du verrou.



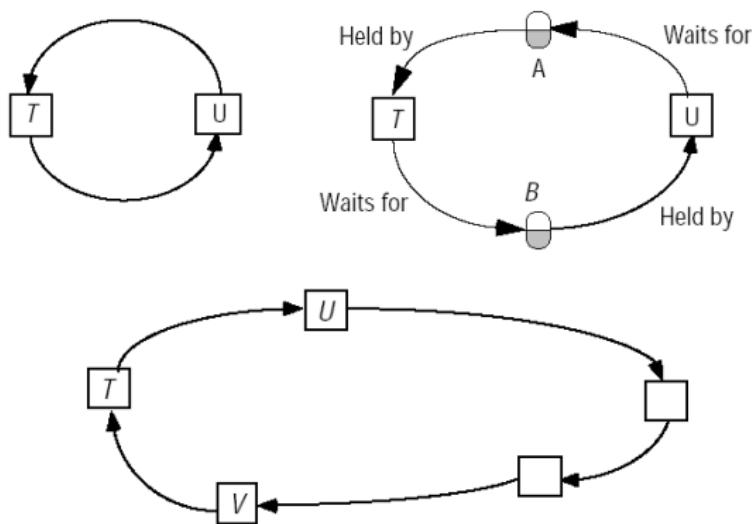
# Interblocages

- Implique au moins deux transactions.
- Chaque transaction est bloquée et ne peut être débloquée que par une autre transaction de l'ensemble.

Transaction T	Verrous T	Transaction U	Verrous U
deposit(A, 100)	Lock A		
		deposit(B, 200)	Lock B
withdraw(B, 100)	Wait for B		
:	:	withdraw(A, 200)	wait for A
:	:	:	:

## Graphes de dépendance

- Dépendance directe: T attend que U libère le verrou et vice-versa.
- Dépendance indirecte: V attend après T qui attend après U...



## Prévention des interblocages

Solution 1: verrouiller tous les objets utilisés par la transaction dès le début.

- Génère un verrouillage prématué, réduit la concurrence.
- Parfois, il est impossible de prédire dès le début quels objets seront utilisés (cas d'applications interactives).

Solution 2: verrouiller les objets dans un ordre prédéfini.

- Pour ordonner, il faut aussi prédire les objets utilisés.



## Détection d'interblocage

- Déetecter les cycles dans le graphe de dépendance.
- Abandonner une transaction appartenant à chacun des cycles.
- Le module responsable de la détection peut faire partie du gestionnaire de verrous:
  - Maintient une représentation du graphe de dépendance.
  - Arcs ajoutés ou supprimés dans le graphe lors des opérations sur les verrous.
  - Vérification périodique de l'existence de cycle.
  - Lorsqu'un cycle est détecté, une des transactions est abandonnée (choisie selon son âge, le nombre de cycles dans lesquels elle apparaît).



## Résolution d'interblocage

- Chaque verrou pris a une période durant laquelle il est non vulnérable.
- A la fin de cette période, si une autre transaction attend après le verrou, la transaction qui le possède est annulée et le verrou libéré.
- La transaction qui attendait peut maintenant acquérir le verrou et poursuivre son travail.



# Contrôle optimiste de la concurrence (COC)

- Approche développée par Kung et Robinson (1981) pour pallier les inconvénients du verrouillage.
- Chaque transaction a une version provisoire des objets qu'elle met à jour (Il est aisément d'abandonner une transaction).
- Fonctionnement:
  - Les lectures sont exécutées sur la version réelle (pas de lectures contaminées). Il serait possible mais plus difficile de lire dans les versions provisoires.
  - Les écritures sont exécutées sur la version provisoire.
  - Deux ensembles sont associés avec chaque transaction, les objets lus et les objets mis à jour.
  - Puis validation et mise à jour



# COC: Principe de validation des transactions

- La validation est une phase atomique, on vérifie si les opérations ont produit des conflits ou pas.
  - Si la transaction est validée, tous les changements seront rendus permanents.
  - Sinon, les versions provisoires sont simplement abandonnées.

Transaction T	Transaction U	Règle
écriture	lecture	U ne doit pas lire un objet écrit par T
lecture	écriture	T ne doit pas lire un objet écrit par U
écriture	écriture	T ne doit pas écrire un object écrit par U U ne doit pas écrire un objet écrit par T



# COC: Approches de validation des transactions

- **Méthode vers l'arrière**, vérifier qu'aucune transaction concurrente terminée n'ait écrit une variable lue par la transaction à valider.
- **Méthode vers l'avant**, vérifier que la transaction courante n'ait pas écrit une variable déjà lue par une transaction concurrente non terminée.
- La phase de validation et d'écriture doit être atomique par rapport aux opérations des autres transactions. Tout le reste est bloqué lorsqu'on valide et termine une transaction.



# Validation des transactions par estampille de temps

- Un **numéro d'ordre (temps)** est affecté à chaque transaction **au moment du début de la transaction**.  $T_i$  précède  $T_j$  si  $i < j$ .
- A **chaque lecture, ou écriture** d'une variable, **on note le temps** de la transaction.
- Pour une **écriture** d'une variable, **si la transaction est avant** la **dernière lecture ou** avant la dernière **écriture commise**, la **transaction est annulée**, autrement l'écriture est acceptée.
- Pour la **lecture** d'une variable, **si la transaction est avant** la **dernière écriture commise**, la **transaction est annulée**, autrement la lecture est acceptée.
- Des **versions plus élaborées** de l'algorithme peuvent lire des anciennes valeurs ou des nouvelles valeurs tentatives et ainsi **éviter plusieurs conflits** et **augmenter la concurrence possible**.

# Cohérence et réPLICATION pour les données réPARTIES

- 1 Introduction
- 2 Les transactions
- 3 Contrôle de la concurrence
- 4 Transactions imbriquées
- 5 Récupération en cas de panne
- 6 Les transactions réparties
- 7 Conclusion



# Gestionnaire de récupération en cas de panne

- Sauver les items de données sur support de stockage permanent.
- Récupérer les données après une panne du processeur.
- Optimiser la performance de la récupération.
- Libérer l'espace de stockage des données intermédiaires.



# Stockage permanent

- Mémoire non volatile avec écriture atomique.
- Disque.
- Disque et cache + pile.
- Disques redondants.
- Disque + ruban.
- Rubans ou disques redondants à distance.



# Principe de la récupération en cas de panne

- Toutes les nouvelles valeurs à commettre doivent être mémorisées pour les appliquer si la transaction est acceptée.
- Toutes les anciennes valeurs doivent être conservées au cas où la transaction sera annulée.
- Il faut éventuellement se débarrasser des nouvelles valeurs de transactions annulées, ou des anciennes valeurs de transactions acceptées.



## Méthode du journal

- Ajouter à la fin du journal les nouvelles valeurs pour une transaction, et les marques pour le début, la fin ou l'annulation d'une transaction.
- Lorsqu'une transaction est acceptée, une marque est écrite dans le journal et les nouvelles valeurs sont propagées éventuellement à la base de donnée.
- Pour récupérer: initialiser les données, lire à l'envers le journal, et ajouter toutes les valeurs provenant de transactions acceptées pour les éléments de données qui n'ont pas encore été lus du journal.
- Si la récupération est arrêtée et recommencée, le résultat n'est pas modifié.
- Lorsque le journal devient trop long, une copie de toutes les valeurs acceptées mais non propagées est écrite dans un nouveau journal, puis les entrées de transactions en cours sont ajoutées, et le nouveau journal remplace l'ancien.

## Méthode des doubles versions

- Un espace de donnée suffisant pour les valeurs acceptées et les valeurs tentatives de toutes les transactions en cours est utilisé.
- Une carte dit pour chaque variable où se trouve la valeur acceptée.
- Pour préparer une transaction, les nouvelles valeurs sont stockées et une carte qui y réfère est créée.
- Pour accepter une transaction, la nouvelle carte remplace l'ancienne.



# Cohérence et réPLICATION pour les données réPARTIES

- 1 Introduction
- 2 Les transactions
- 3 Contrôle de la concurrence
- 4 Transactions imbriquées
- 5 Récupération en cas de panne
- 6 Les transactions réparties
- 7 Conclusion



## Les transactions réparties

- Une transaction répartie peut faire des requêtes à plusieurs serveurs.
- Un serveur peut faire appel à d'autres serveurs pour traiter une requête
- A la fin de la transaction, tous les serveurs doivent soit compléter, soit abandonner (prennent conjointement la même décision).



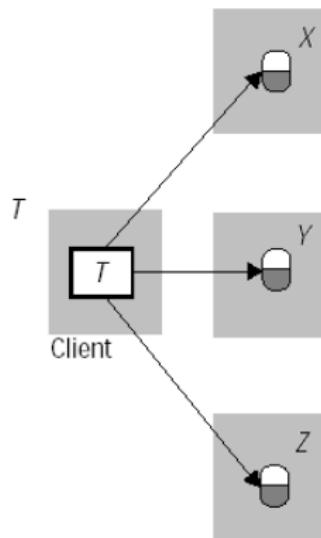
# Défis des transactions réparties

- Contrôle de la concurrence :
  - Les transactions doivent être *sérialisées globalement* sachant que chaque serveur *sérialise localement* les transactions.
  - Des *interblocages répartis* peuvent survenir suite à des cycles de dépendances entre différents serveurs.
- Recouvrement d'abandons de transactions:
  - Chaque serveur doit assurer que ses objets soient récupérables
  - Garantir que le contenu des objets reflète bien tous les effets des transactions complétées et aucun de celles abandonnées
- Recouvrement de panne:
  - Un serveur est désigné comme coordonnateur, il détermine si la transaction est approuvée ou non.
  - Chaque serveur doit finaliser les transactions approuvées, même en cas de panne.



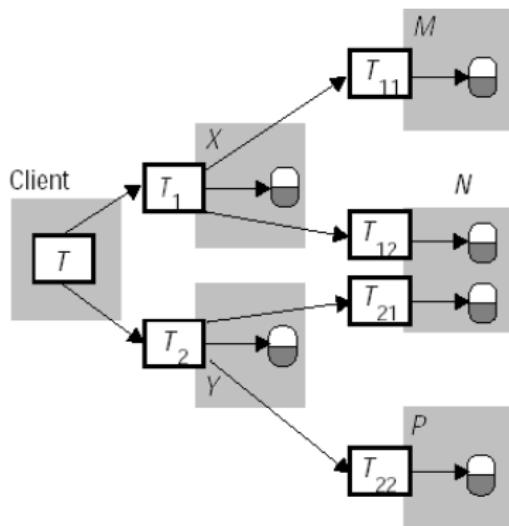
## Transactions Réparties Simples

- Contient des opérations qui s'adressent à plusieurs serveurs, mais un seul à la fois.
- Les requêtes sont envoyées et traitées séquentiellement



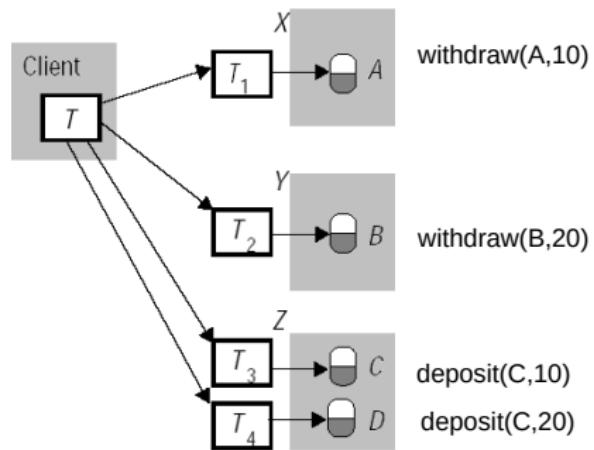
## Transactions Réparties Imbriquées

- Consiste en une hiérarchie de sous-transactions
- Les sous-transactions du même niveau s'exécutent simultanément, s'adressent à plusieurs serveurs et sont elles-mêmes des transactions imbriquées

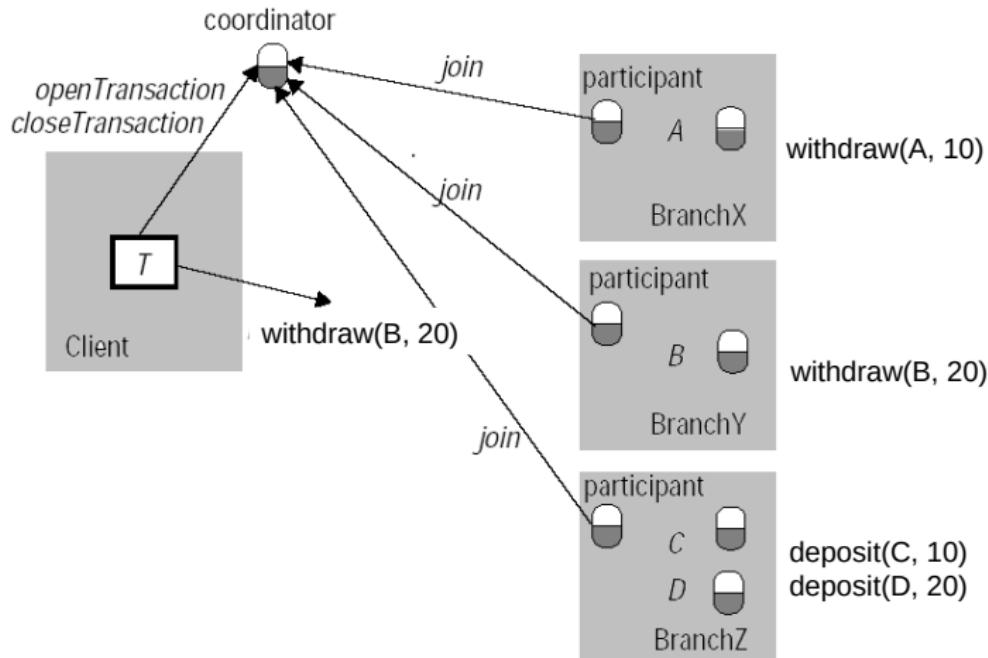


# Transactions Réparties

- Begin Transaction T
- withdraw(A,10)
- withdraw(B,20)
- deposit(C,10)
- deposit(D,20)
- Commit T



# Traitement d'une transaction



## Protocoles de fin de transaction

- Assurer l'atomicité d'une transaction répartie: toutes les opérations sont exécutées correctement, ou aucune opération n'est exécutée (elle est abandonnée).
- Protocole de fin de transaction atomique à *une phase*: le coordonnateur envoie d'une façon répétitive un message de fin de transaction aux participants jusqu'à ce que tous les participants répondent par un accusé de réception. Ne fonctionne que si tous acceptent (ou tous refusent).
- Protocole de fin de transaction atomique à *deux phases*: premier tour pour vérifier si tous les participants peuvent accepter la transaction, deuxième tour pour donner le verdict.



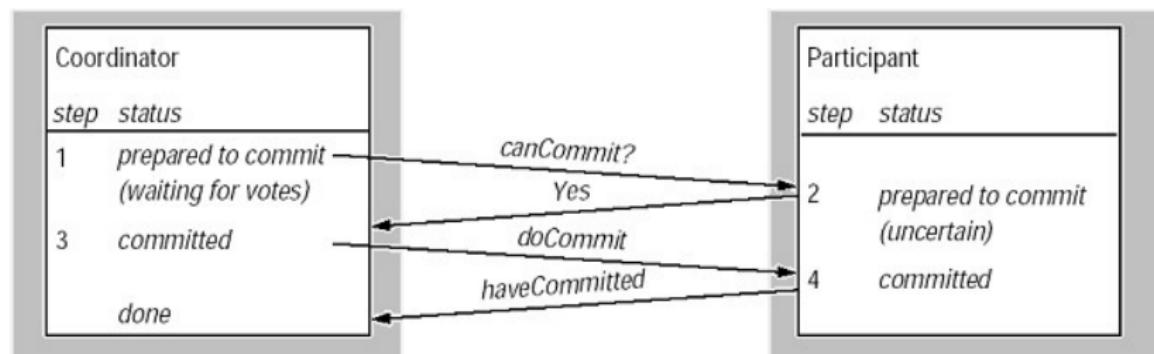
# Protocole de fin de transaction atomique à deux phases

- Phase de **vote**:
  - Chaque serveur (**coordonnateur et participants**) **vote** pour **compléter ou abandonner** la transaction.
- Phase de **décision**:
  - **Si au moins un des serveurs a voté pour abandonner**, la décision sera **d'abandonner** la transaction.
  - Les serveurs exécutent la décision prise par le coordonnateur.



# Interface du protocole à deux phases

- Interface du participant:
  - `canCommit(trans)`: Yes / No
  - `doCommit(trans)`
  - `doAbort(trans)`
- Interface du coordonnateur
  - `haveCommitted(trans, participant)`
  - `getDecision(trans)`: Yes / No



## Problèmes possibles et solutions

- Si la réponse d'un participant (*canCommit*) en première phase tarde trop, le coordonnateur peut simplement annuler la transaction.
- Si un participant est sans nouvelle depuis trop longtemps d'une transaction amorcée mais pour laquelle il ne s'est pas commis, il peut abandonner la transaction.
- Si un participant s'est commis (répondu positivement à *canCommit*) mais ne reçoit pas de confirmation du coordonnateur (*doCommit* ou *doAbort*), il doit obtenir une réponse en relançant le coordonnateur avec *getDecision*.
- Le protocole requiert au minimum  $4N$  messages soit 4 par participant (*canCommit*, réponse oui ou non, *doCommit*, *haveCommitted*). Le message *haveCommitted* est optionnel mais permet de libérer l'information sur la transaction. Si le coordonnateur est un participant, le nombre de messages réseau devient  $4(N - 1)$ .

## Contrôle de la concurrence pour les transactions réparties

- Les serveurs qui traitent des transactions réparties doivent s'assurer que leur exécution est équivalente à une exécution en série: **Si** une transaction T est avant U selon un des serveurs, **alors** elle doit être dans cet ordre pour tous les serveurs dont les objets sont accédés à la fois par T et U.
- Le contrôle optimiste de la concurrence avec sa phase de vérification atomique ne se prête pas bien aux transactions réparties.
- Le verrouillage fonctionne bien en réparti, chaque serveur contrôle la concurrence de ses propres objets avec des verrous posés localement et levés seulement à la fin de la transaction.



## Interblocages dans les transactions réparties

- Les serveurs posent les verrous localement et imposent des ordres possiblement différents sur les transactions.
- Il se produit des interblocages répartis.
- Détecteur d'interblocage global (complexe) ou délai maximal (simple) pour se sortir d'un interblocage.

Transaction T	Verrous T	Transaction U	Verrous U
Write(A) on X	Lock A		
		Write(B) on Y	locks B
Read(B) on Y	Waits for U		
:		Read(A) on X	Wait for T

# Cohérence et réPLICATION pour les données réPARTIES

- 1 Introduction
- 2 Les transactions
- 3 Contrôle de la concurrence
- 4 Transactions imbriquées
- 5 Récupération en cas de panne
- 6 Les transactions réparties
- 7 Conclusion



## Conclusion

- Dans beaucoup de cas, plusieurs étapes d'une opération logique doivent s'exécuter de manière atomique:
  - Transaction;
  - Modification de fichiers répliqués;
  - Installation de tous les fichiers d'une nouvelle version d'une application...
- Il faut assurer la cohérence par le contrôle de la concurrence (e.g. verrous).
- Le principe du protocole à deux phases est utilisé un peu partout pour assurer la validation
  - Transactions réparties possiblement imbriquées;
  - Envoi atomique de message de groupe
  - Réserver un moment pour une réunion à plusieurs personnes (vérifier la disponibilité, ensuite confirmer ou annuler).



# Tolérance aux pannes

## Module 10

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① Terminologie et modèles
- ② Réduction et masquage des fautes
- ③ RéPLICATION
- ④ Le consensus en réparti
- ⑤ Transactions réparties
- ⑥ Calcul de disponibilité
- ⑦ Cas exemples: les limites de la tolérance aux pannes



# Tolérance aux pannes

- ① Terminologie et modèles
- ② Réduction et masquage des fautes
- ③ RéPLICATION
- ④ Le consensus en réparti
- ⑤ Transactions réparties
- ⑥ Calcul de disponibilité
- ⑦ Cas exemples: les limites de la tolérance aux pannes



# Un peu de terminologie

- Sûreté de fonctionnement (*Dependability*)
  - Composantes (*Attributes*)
    - Disponibilité (*Availability*)
    - Fiabilité (*Reliability*)
    - Sécurité (*Security*)
    - Confidentialité (*Confidentiality*)
    - Intégrité (*Integrity*)
    - Maintenabilité (*Maintainability*)
  - Mécanismes (*Means*)
    - Prévention des fautes (*Fault prevention*)
    - Tolérance aux fautes (*Fault tolerance*)
    - Eradication des fautes (*Fault removal*)
    - Prévision des fautes (*Fault forecasting*)
  - Problèmes (*Impairments*)
    - Faute (*fault*)
    - Erreur (*Error*)
    - Panne ou défaillance (*Failure*)



## Systèmes fiables

Disponibilité: être prêt à l'utilisation.

Fiabilité: continuité de service.

Sûreté: pas de conséquences catastrophiques pour l'environnement.

Sécurité: prévention d'accès non autorisés et/ou de la manipulation de l'information (confidentialité, intégrité, disponibilité).

Maintenabilité: réfère à la facilité avec laquelle un système défaillant peut être réparé.



## Définitions

- Défaillance (**panne**): lorsque le comportement d'un système viole sa spécification de service
  - Les défaiillances résultent de problèmes inattendus internes au système qui se manifestent éventuellement dans le comportement externe du système.
- Ces problèmes sont appelés erreurs et leurs causes mécaniques ou algorithmiques sont dites fautes.
- Quatre sources de fautes peuvent causer la défaillance d'un système:
  - Spécification inadéquate.
  - Erreurs de conception dans le logiciel.
  - Défaillance matérielle.
  - Interférence sur le sous-système de communication

## Classification des fautes

**Transitoire:** se **produit de manière isolée** (une fois et disparaît); rayonnement alpha qui perturbe un bit de mémoire.

**Intermittente:** se reproduit sporadiquement (transitoire et survient à plusieurs reprises); problème de bruit, de mauvaise synchronisation, de couplage entre signaux, de chaleur...

**Permanente:** persiste indéfiniment (jusqu'à réparation) après son occurrence; disque brisé, circuit brûlé...

## Classification des défaillances

**déTECTée:** le serveur répond avec un message d'erreur (disque avec **parité incorrecte sur un bloc**).

**plantage:** serveur **s'arrête, mais il fonctionnait correctement** jusqu'alors.

**panne d'omission:** serveur ne répond pas à une requête; bonne réponse ou pas de réponse (**contrôleur de disque brûlé**).

**panne de temporisation:** la réponse survient en dehors de l'intervalle de temps réel spécifié; **délai excessif** (commande de centrale nucléaire, commande de vol, transactions à haute fréquence).

**panne de réponse:** **réponse** est simplement **incorrecte**; capteur bruité, mémoire corrompue.

**panne arbitraire (Byzantine):** **réponses arbitraires voire malicieuses** (testeur qui essaie de déjouer le système, attaque de sécurité).

# Tolérance aux pannes

- 1 Terminologie et modèles
- 2 Réduction et masquage des fautes
- 3 RéPLICATION
- 4 Le consensus en réparti
- 5 Transactions réparties
- 6 Calcul de disponibilité
- 7 Cas exemples: les limites de la tolérance aux pannes



# Approches pour atteindre la Fiabilité

- **Eviter** les fautes
  - **Prévention** de fautes: comment **prévenir l'occurrence ou l'introduction de fautes**
  - **Élimination** de fautes: comment **réduire la présence** (nombre et sévérité) de fautes
- **Accepter** les fautes
  - **Tolérance** aux fautes: comment **fournir un service en dépit des fautes**
  - **Prévision** de fautes: comment **estimer la présence, la création et les conséquences** des fautes



## Prévention de Fautes par évitemen

- Utilisation des composants les plus fiables en respectant les coûts et contraintes de performance.
- Choix des meilleures techniques d'assemblage et d'interconnexion des composants.
- Langages de programmation avec abstraction de donnée, modularité, sécurité.
- Environnement de développement et méthodologie structurés aidant à gérer la complexité et ne rien oublier.
- Examiner les formes d'interférences.
- Spécification rigoureuse des besoins, si pas formelle.



## Elimination de Fautes

- La prévention ne peut éliminer la possibilité de fautes. Il faut détecter et éliminer les fautes. Prévoir un jeu de test le plus complet possible, dans des conditions de charge réalistes.
- Un test peut seulement être utilisé pour démontrer la présence de fautes, pas leur absence complète.
- Il est parfois impossible de tester sous des conditions réelles.
- La plupart des tests sont faits dans un mode de simulation, et il est difficile de garantir que la simulation est exacte.
- Les erreurs introduites durant la spécification des besoins peuvent ne pas se manifester jusqu'à ce que le système tombe en panne.



## Tolérance aux fautes

- **Récupération de faute:** si un délai de récupération est acceptable.
- Il suffit de pouvoir reprendre où on avait laissé, après avoir redémarré (et possiblement réparé) le système.
- Requiert un stockage permanent fiable pour ne perdre aucune information. Copies de sauvegarde, cliché à intervalle régulier...
- Faire attention de bien tout sauver avant d'accepter une transaction.



## Tolérance aux fautes

- Masquer la présence de fautes en utilisant la **redondance**
- **Redondance matérielle:** composants matériels ajoutés pour supporter la tolérance aux fautes à tous les niveaux (alimentation électrique, processeurs, disques, réseau...).
- **Redondance logicielle:** inclut tous les programmes et instructions utilisés pour tolérer les fautes.
- **Redondance temporelle:** temps extra pour exécuter les tâches (**exécuter les instructions plusieurs fois**) pour la tolérance aux fautes.

## Niveaux de tolérance aux fautes

Tolérance aux fautes complète: le système **continue à fonctionner** en présence de fautes, **sans perte** significative de **fonctionnalité** ou de **performance**.

Dégénération graduée: le système **continue à fonctionner** en présence de fautes, **acceptant une dégradation partielle** des **fonctionnalités ou de performance** **durant le recouvrement ou la réparation**.

Arrêt sécuritaire: le système maintient son intégrité tout en acceptant un arrêt temporaire de son fonctionnement.

### Notes:

- Niveau de tolérance aux fautes nécessaire dépend de l'application.
- (La plupart des systèmes critiques nécessitent une tolérance complète, mais plusieurs se contentent d'une dégradation graduée.)

## Phases de la tolérance aux fautes

Détection d'erreurs: la présence des fautes est déduite en détectant une erreur.

Recouvrement d'erreurs: élimine les erreurs de telle façon qu'elles ne se propagent pas par des actions futures.

## Phases de la tolérance aux fautes (suite)

**Recouvrement d'erreurs en aval:** continuer à partir de l'état erroné en faisant des corrections sélectives à l'état du système.  
(Remplacer un disque défectueux dans une unité RAID)

**Recouvrement d'erreurs en amont:** consiste à restaurer le système à un état précédent sûr et en exécutant une section alternative du programme. (Remplacer un disque défectueux et l'initialiser avec une copie de sauvegarde)

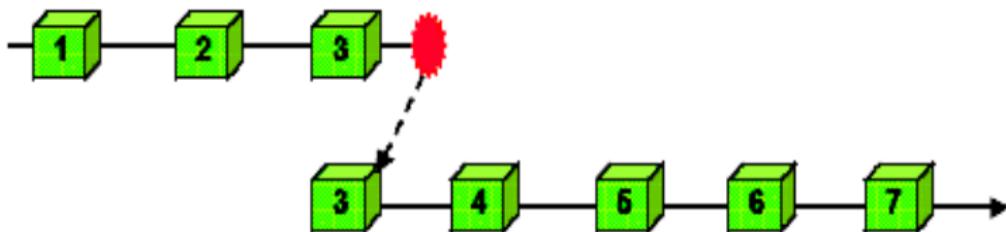
**Point de recouvrement:** point auquel un processus est restauré

**Point de contrôle (checkpoint):** établissement d'un point de recouvrement (en sauvegardant l'état approprié du système)

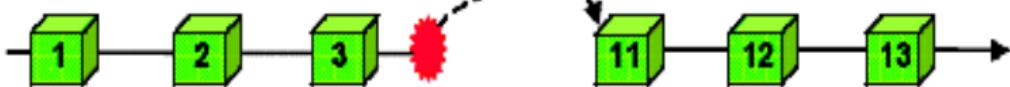


## Méthodes de recouvrement d'erreurs

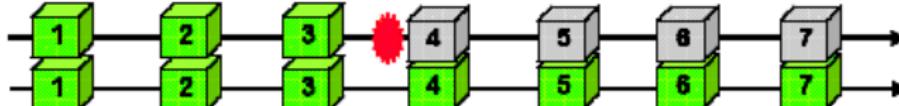
**Backward recovery**



**Forward recovery**



**Compensation-based recovery (fault masking)**



# Traitement de fautes et continuité de service

- Le recouvrement d'erreur retourne le système à un état exempt d'erreur; l'erreur peut survenir à nouveau. La phase finale de la tolérance aux fautes est d'éradiquer la faute du système.
- Une erreur est un symptôme de faute; corriger l'erreur ne répare pas la faute.
- Le composant défaillant (ou la panne) doit être identifié.
- Le traitement automatique des pannes est difficile et spécifique au système.
- Localisation de la panne et réparation du système
- Les techniques de détection d'erreur peuvent aider à retracer la panne
  - Faute matérielle: le composant peut être remplacé
  - Faute logicielle: peut être éliminée dans une nouvelle version du code
  - Dans les applications qui ne doivent pas s'arrêter il est nécessaire de modifier le programme alors qu'il s'exécute!

## Masquage hiérarchique des fautes

- Un niveau détecte, reprend et masque les erreurs au niveau plus bas:
  - Le client demande l'adresse IP d'un ordinateur.
  - Le serveur de nom fait une requête récursive à d'autres serveurs et prend les serveurs alternatifs lorsqu'un serveur ne répond pas.
  - Un serveur interrogé découvre une erreur sur un disque et lit plutôt de son second disque en miroir.
  - Le client a la réponse et ne sait rien des défaillances.



## Masquage par groupe des fautes

- Le client demande à tous les serveurs, prend le premier qui répond. Peut tolérer  $\frac{n-1}{n}$  pannes par omission.
- Le client demande à tous les serveurs et prend un vote sur les réponses. Peut tolérer  $\frac{n-2}{n}$  pannes si les chances d'avoir deux mauvaises réponses identiques sont presque nulles, ou  $\frac{n}{2n+1}$  pannes même si tous les ordinateurs défaillants tentent vicieusement de s'entendre sur une mauvaise réponse.
- **Groupe synchronisé:** tous les serveurs doivent recevoir les mêmes mises à jour dans le même ordre.
- **Groupe non synchronisé:** les serveurs de secours enregistrent les mises à jour dans un journal mais sans les traiter. Au besoin, ils utilisent ce journal pour se mettre à jour et prendre le relais avec un certain délai.

## Tolérance aux fautes matérielle

**Statique:** composants redondants utilisés pour cacher les effets des fautes

- **Exemple:** Triple Modular Redundancy (TMR): 3 composants identiques et un circuit de vote majoritaire; les résultats sont comparés et si un diffère des deux autres il sera masqué.
- Assume que la faute n'est pas commune (telle qu'une erreur de conception) mais elle peut être transitoire ou causée par la détérioration du composant.
- Pour masquer les fautes de plus d'un composant il faut NMR (**N-tuple Modular Redundancy**).

**Dynamique:** redondance utilisée à l'intérieur d'un composant indiquant si le résultat est erroné.

- Fournit une technique de détection d'erreurs; le recouvrement doit être fait par un autre composant

# Tolérance aux fautes logicielle

- Utilisée pour déetecter les erreurs de conception
- **Statique:** programmation N-Versions
- **Dynamique:**
  - Détection et recouvrement
  - Blocs de recouvrement: recouvrement d'erreurs en amont
  - Exceptions: recouvrement d'erreurs en aval

# Tolérance aux pannes

- 1 Terminologie et modèles
- 2 Réduction et masquage des fautes
- 3 Réplication
- 4 Le consensus en réparti
- 5 Transactions réparties
- 6 Calcul de disponibilité
- 7 Cas exemples: les limites de la tolérance aux pannes



# RéPLICATION de données

- Maintien de copies d'un objet sur plusieurs systèmes.
- Peut améliorer les performances du système (2x plus de bande passante de lecture pour deux disques en miroir, même performance qu'un seul disque en écriture).
- Augmente la disponibilité en masquant les fautes et donc minimisant l'impact des pannes.



# Effets positifs de la réPLICATION

## Performance:

- Mise en cache des données accédées fréquemment, ou la réPLICATION des données près du point d'accès pour réduire le temps d'accès.
- Répartition des données sur plusieurs serveurs pour répartir la charge de travail.
- La réPLICATION d'objets statiques est triviale tandis que la réPLICATION d'objets dynamique requiert des mécanismes de contrôle de la concurrence et de maintien de la cohérence (ajoute de la surcharge de traitement et de communication).

## Disponibilité:

- Offrir une disponibilité du service le plus près possible de 100%
- 2 types de pannes:
  - Serveur: réPLICATION de données sur des serveurs qui tombent en panne indépendamment des autres.
  - Partition du réseau ou opérations déconnectées: les utilisateurs dans leurs déplacements se connectent/déconnectent aléatoirement au réseau.

# Requis généraux pour la réPLICATION de données

- **Transparence** de la réPLICATION:

- Les utilisateurs voient des objets logiques, ils n'ont **aucune connaissance des multiples copies physiques qui existent**
- Ils **accèdent une instance de l'objet, peu importe laquelle**, et reçoivent un **résultat unique**

- **Cohérence**:

- Étant donné **plusieurs copies physiques** de l'objet **et des accès concurrents**, les **copies doivent être** maintenues dans un état **cohérent** entre elles

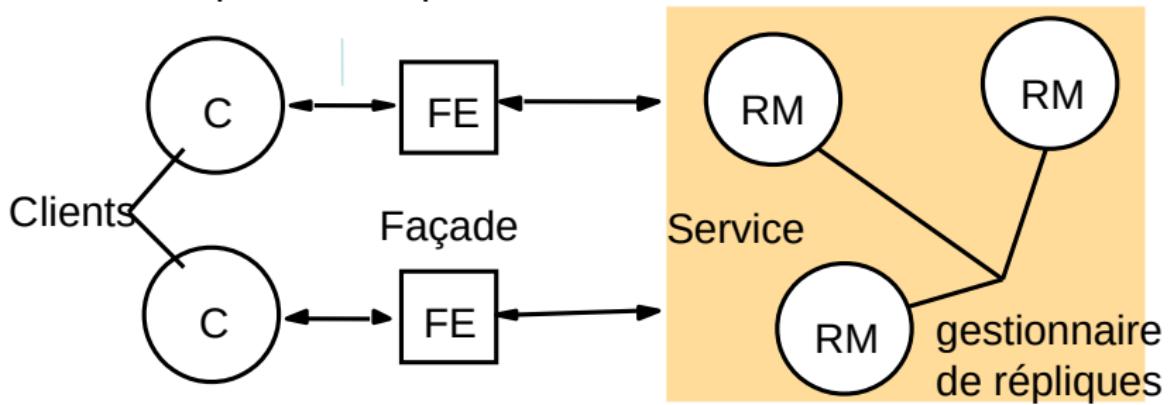


## Modèle du système

- Les données dans le système sont considérées comme un ensemble d'objets logiques.
- Chaque objet logique est implémenté par un ensemble d'objets physiques appelés répliques.
- La cohérence de chacune des répliques dépend des opérations appliquées localement par le système sur lequel elles résident.
- Les répliques peuvent être incohérentes entre elles à un point donné dans le temps.
- Le système en soi est asynchrone et peut tomber en panne.

## Modèle architectural pour la réPLICATION

### Requêtes et réponses



## Modèle architectural de base pour la réplication

- Le système se compose de trois éléments :
  - Les clients effectuent les requêtes sur des objets répliqués à travers les FE (*Front End*).
  - Les FE agissent comme intermédiaire pour les clients et s'occupent de contacter les RMs (*Replica Manager*). Ils assurent la transparence pour les utilisateurs
  - Les RMs maintiennent les répliques d'objets qui sont sous sa responsabilité
    - Les objets peuvent se trouver répliqués sur tout ou sur un sous-ensemble des RMs
    - Les RMs offrent un service d'accès aux objets répliqués au client (accès en lecture ou en mise à jour)



# Les étapes de base pour l'accès aux objets répliqués

## 1 Requête

- La requête d'un client est prise en charge par le FE qui s'occupe de la relayer à un ou plusieurs RM (diffusion sélective)

## 2 Coordination

- Les RMs décident d'accepter ou non la requête, lequel RM va servir la requête et dans quel ordre par rapport aux autres requêtes (FIFO, causal, total)

## 3 Exécution

- Le ou les RMs concernés réalisent la requête (peut-être partiellement pour permettre le recouvrement de l'ancien état de l'objet)

## 4 Accord

- Les RMs se mettent d'accord sur les effets de la requête sur les différentes instances de l'objet répliqué (approche immédiate ou paresseuse)

## 5 Réponse

- Un ou plusieurs RMs renvoient la réponse au FE ayant effectué la requête, qui lui-même la donne au client concerné

## Service tolérant aux fautes

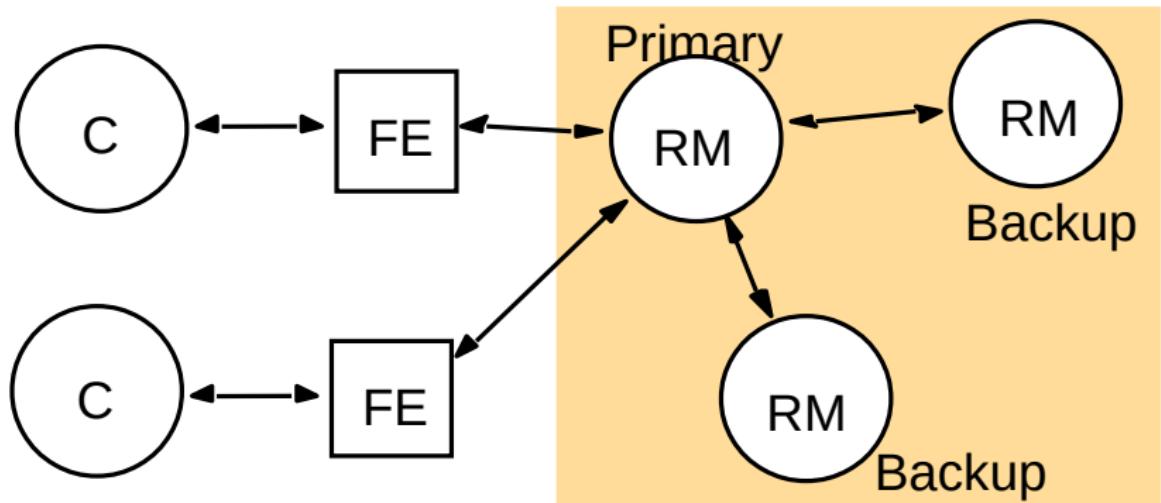
- Un service a un comportement dit correct s'il fonctionne selon les spécifications même en présence de pannes
- Les clients ne peuvent voir la différence entre un service obtenu à partir de données répliquées et celui obtenu à partir d'une copie unique
- On doit s'assurer que l'ensemble des répliques produit le même comportement qu'une seule copie



## Modèle de **réPLICATION PASSIVE**

- Dans ce modèle, à tout moment il y a un RM actif ( primaire) qui répond aux requêtes des clients et des RMs passifs (secondaires) qui servent de RMs de secours en cas de panne.
- Le résultat d'une opération effectuée sur le RM primaire est communiqué à tous les secondaires (ce qui permet de maintenir la cohérence des données).
- Quand le RM primaire tombe en panne, un RM secondaire prend la relève (après une élection, par exemple).
- Le système est linéarisable puisque la séquence des opérations est programmée et contrôlée par le RM primaire.
- Si le RM primaire tombe en panne, le système demeure linéarisable puisqu'un des RMs secondaires reprend exactement là où le RM primaire est tombé en panne.

## Modèle de réPLICATION passive



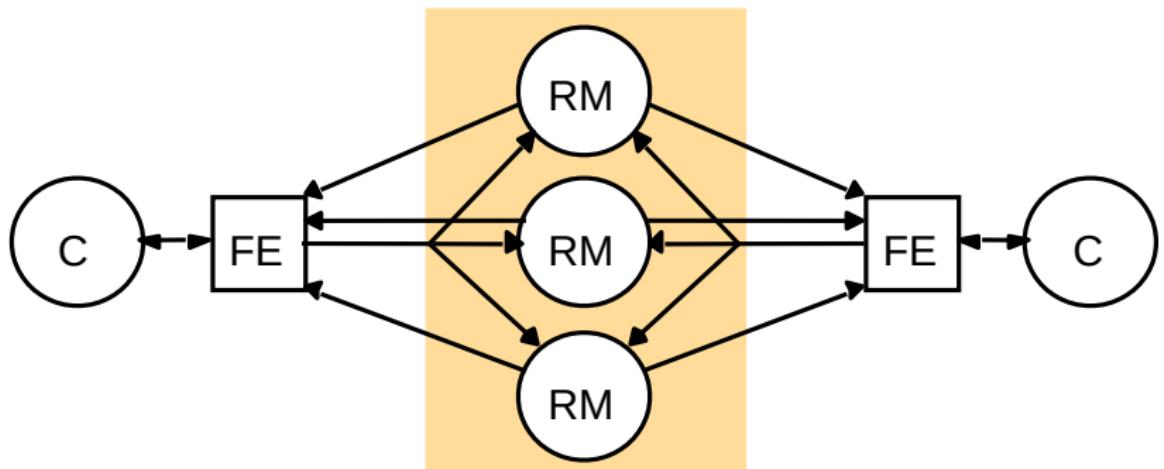
## Exemples de réPLICATION passive

- Serveur NIS: un **serveur primaire** qui **propage** les **modifications** à plusieurs **serveurs secondaires**.
- **Linux virtual server**: un serveur primaire a deux adresses IP, une pour la gestion et une pour le service. Un serveur de rechange reçoit les mises à jour et interroge souvent le serveur primaire. Si le serveur primaire ne répond plus, il le déconnecte, prend son adresse IP de service et commence à s'annoncer sous cette adresse et à servir les requêtes.
- **Deux serveurs DHCP peuvent coexister s'ils offrent des blocs d'adresses différents**. Le second serveur répond s'il voit que le premier ne répond pas aux requêtes à tous pour une adresse dynamique.

## Modèle de réPLICATION active

- Les RMs sont des machines à état qui jouent exactement le même rôle et sont organisés en groupe.
- Les RMs débutent dans le même état, et changent d'état concurremment, de telle manière que leurs états restent identiques.
- Si un RM tombe en panne, cela n'a aucun effet sur les performances du système parce que les autres RMs peuvent prendre la relève de manière transparente.
- Ce modèle requiert un mécanisme de communication à diffusion sélective (*multicast*) fiable et totalement ordonné (pour que les RMs effectuent les mêmes opérations dans le même ordre).
- Supporte la cohérence séquentielle mais pas la linéarisation (car l'ordre total ne correspond pas nécessairement à l'ordre d'exécution temporel des opérations).

## Modèle de réPLICATION active



## Exemple de réPLICATION active

- Serveurs NFS avec automount: la requête est envoyée à tous les serveurs et le premier à répondre est pris.
- CODA: plusieurs serveurs offrent les mêmes fichiers et se propagent les modifications. Très flexible mais ne peut garantir l'absence de conflits (mises à jour concurrentes).



# Tolérance aux pannes

- 1 Terminologie et modèles
- 2 Réduction et masquage des fautes
- 3 Réplication
- 4 Le consensus en réparti
- 5 Transactions réparties
- 6 Calcul de disponibilité
- 7 Cas exemples: les limites de la tolérance aux pannes



## Le consensus en réparti

- Plusieurs processus, corrects ou fautifs, échangent des messages.
- Chaque processus doit prendre une décision (e.g. qui est le serveur primaire).
- Terminaison: éventuellement tous les processus corrects arrivent à une décision.
- Consensus: tous les processus corrects terminent avec la même décision.
- Intégrité: si tous les processus corrects proposent la même décision, cette décision doit l'emporter.
- Chaque processus correct communique sa proposition et celle déjà connue d'autres processus au groupe. Chaque processus accumule les propositions des autres et se soumet à la proposition majoritaire.

## Difficultés

- Le coordonnateur élu importe peu, l'important est d'avoir un consensus!
- L'élection hiérarchique produit un élu dans chaque partition du réseau, ce qui n'est pas acceptable si on doit avoir un coordonnateur unique (verrou, base de donnée).
- Exiger un vote à majorité? Ceci assure qu'un seul coordonnateur peut être élu.
- Chaque candidat va chercher des votes. Que faire en cas de résultat minoritaire? Peut-on changer son vote?



# Algorithme de Paxos

- Le consensus en réparti est très difficile à assurer en présence de défaillances, messages asynchrones, partitionnement de réseau...
- Proposé par Lamport en 1989 et publié dans une revue, après avoir été entièrement revisé, seulement en 1998.
- Complexé, utilisé surtout pour les questions fondamentales (e.g., élire un serveur primaire de manière sécuritaire, ensuite le serveur primaire peut coordonner le reste).

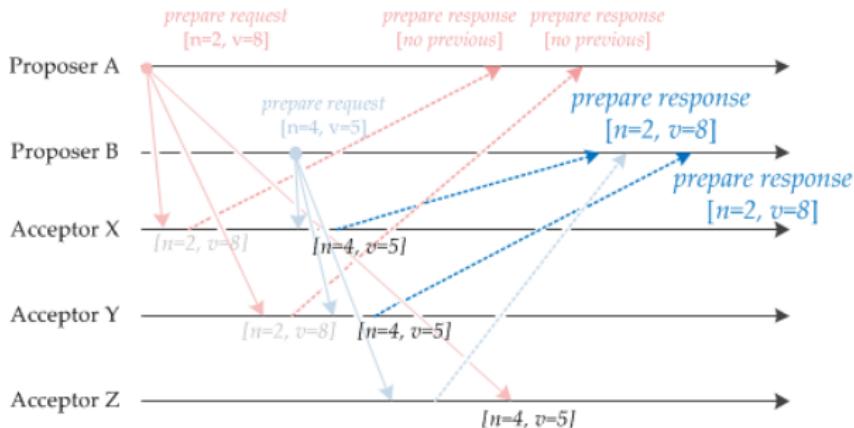


# Algorithme de Paxos

- Par exemple, si on veut élire (établir un consensus) un serveur primaire, il faut obtenir une majorité parmi les accepteurs.
- Chaque proposition a un numéro de séquence et une valeur.
- Première proposition reçue par un accepteur, il promet de ne pas accepter une proposition antérieure et mémorise cette proposition.
- Proposition antérieures reçues par un accepteur, il promet mais retourne la plus récente proposition reçue.
- Le proposeur choisit la proposition la plus récente déjà reçue par un accepteur et demande à chaque accepteur de l'accepter.
- Si une majorité d'accepteurs acceptent, le consensus est obtenu.

## Exemple de l'algorithme de Paxos

- A propose “ $n=2, v=8$ ” qui est accepté par X et Y, et B propose “ $n=4, v=5$ ” qui est accepté par X et Y (en remplacement de  $n=2$ ) et par Z. Ensuite, Z ignore la proposition de A car elle est antérieure ( $n=2$ ).

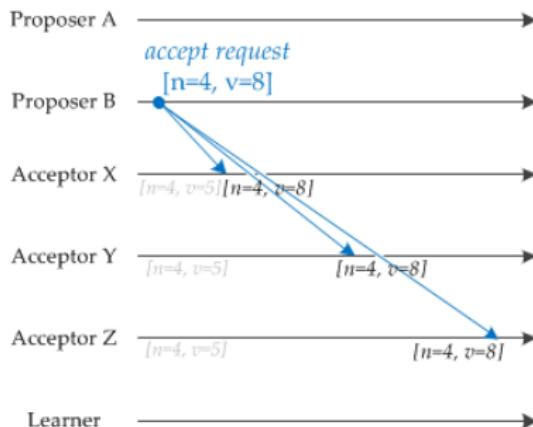


<https://medium.com/@angusmacdonald/paxos-by-example-66d934e18522>

## Exemple de l'algorithme de Paxos (suite)

- A, avec 2 votes sur 3, envoie “accept n=2, v=8” qui est ignoré (désuet) par X, Y et Z; il échoue. B envoie “accept n=4, v=8” (v=8 de proposition antérieure de A) accepté par tous.

Consensus atteint!



<https://medium.com/@angusmacdonald/paxos-by-example-66d934e18522>

# Tolérance aux pannes

- 1 Terminologie et modèles
- 2 Réduction et masquage des fautes
- 3 RéPLICATION
- 4 Le consensus en réparti
- 5 Transactions réparties
- 6 Calcul de disponibilité
- 7 Cas exemples: les limites de la tolérance aux pannes



## Transactions réparties avec réPLICATION

- Le protocole à deux phases pour compléter une transaction doit s'étendre aux données répliquées qui doivent elles aussi changer de manière atomique avec le reste.
- Le coordonnateur de transaction parle au gestionnaire de copies qui agit à son niveau comme coordonnateur.
  - Les messages “*prêt*”, et dans une seconde phase “*compléter*” ou “*annuler*”, sont relayés à toutes les copies avant de produire une réponse au coordonnateur.
  - C'est un peu comme si le coordonnateur de transaction devait contacter toutes les copies.



## Transactions réparties avec réPLICATION (suite)

- Lecture sur un, écriture sur tous: verrou en lecture sur un, ou verrou en écriture sur tous (bloqué si un verrou en lecture existe).
- Copie primaire: tous les verrous sont pris sur la copie primaire, au moment d'accepter la transaction, toutes les mises à jour sont propagées aux copies.
- Il faut faire attention si les copies disponibles pendant la transaction changent, que le tout demeure cohérent.
- Lorsque le réseau est partitionné, on peut continuer dans chaque partition et résoudre les conflits plus tard (optimiste), ou ne continuer que dans la partition qui a le quorum (si une partition a plus que la moitié et est donc la seule ainsi), ou en être réduit à des lectures seulement puisqu'il n'est plus possible d'obtenir des verrous d'écriture sur tous.

# Tolérance aux pannes

- ① Terminologie et modèles
- ② Réduction et masquage des fautes
- ③ RéPLICATION
- ④ Le consensus en réparti
- ⑤ Transactions réparties
- ⑥ Calcul de disponibilité
- ⑦ Cas exemples: les limites de la tolérance aux pannes



## Calcul de disponibilité

- Probabilité qu'un composant soit fonctionnel au temps  $t$ :  
 $\frac{MTTF}{MTTF+MTTR}$  (Mean Time to Fault / Repair).
- Probabilité  $p$  que deux événements indépendants, de probabilité  $p_1$  et  $p_2$ , se produisent en même temps:  $p = p_1 \times p_2$ .
- Probabilité  $p$  que  $p_1$  ou  $p_2$  se produise:  $p = p_1 + p_2 - p_1 \times p_2$
- Les probabilités de cas disjoints ( $p_1 \times p_2 = 0$ ) peuvent s'additionner.
- Probabilité que  $k$  disques sur  $n$  soient fonctionnels ( $n - k$  en panne), si la probabilité pour un disque est  $p_d$ .

$$p = \binom{n}{k} \times p_d^k \times (1 - p_d)^{n-k}$$

- Sommation de  $k$  à  $n$  pour  $k$  disques ou plus fonctionnels.
- Au moins 1 disque fonctionnel dans un miroir:  $1 - (1 - p_d)^2$

## Probabilités: peut-on sommer

- Probabilité que deux événements indépendants de probabilité  $p_1$  et  $p_2$  se produisent au même moment:  $p_1 \times p_2$ . Par exemple, un ordinateur est fonctionnel si la carte mère ( $p_m$ ) et le disque ( $p_d$ ) sont fonctionnels. Il y a une panne sauf si les deux sont fonctionnels.

$$1 - p_m \times p_d$$

- Peut-on plutôt prendre la probabilité de panne de carte + la probabilité de panne de disque?

$$(1 - p_m) + (1 - p_d) \neq 1 - p_m \times p_d$$



## Probabilités: sommer si disjoint

- Quelle est la différence? La panne de disque se produit indépendamment de la carte qui peut être fonctionnelle ou non. On compte donc deux fois le cas où le disque et la carte sont défectueux. Les deux éléments sommés se recoupent alors qu'il faut bien tout séparer. Si  $p_1$  et  $p_2$  sont grands, la partie sommée en double est petite et le résultat est incorrect mais très proche. Si  $p_1$  et  $p_2$  sont petits, le résultat dépasse largement 1!

$$P(p_1 \cup p_2) = p_1 + p_2 - p_1 \times p_2 =$$

$$(1 - p_m) + (1 - p_d) - ((1 - p_m) \times (1 - p_d)) =$$

$$(1 - p_m) + (1 - p_d) - (1 - p_d - p_m + p_m \times p_d) = 1 - p_m \times p_d$$

## Systèmes parfaitement tolérants aux pannes?

- Avec le bon niveau de redondance, on peut diminuer les probabilités de panne pratiquement à 0, sous certaines hypothèses.
- Cinq 9 (0.99999) donne 5.26 minutes par an de panne, six 9 donne 31.5 secondes par an.
- En pratique, les pannes sont plus fréquentes car certaines hypothèses ne tiennent pas, surtout sur les nouveaux systèmes.
- Pour les systèmes plus matures, on analyse les accidents et on apprend des erreurs (e.g., en aviation).



## Disques en miroir

- Un disque tombe en panne environ une fois par 4 ans ( $1\text{h}/4\text{ans} = 0.00003$ ). Avec 2 disques en miroir les probabilités de pannes simultanées sont à peu près nulles ( $.00003^2 = 1\text{h}/150000\text{ans}$ ). Plus besoin de copies de sauvegarde?!?
- Lors d'une panne, il faut être **alerté et effectuer la réparation rapidement** pour revenir à une configuration redondante.
- Le matériel redondant doit être en bon état de marche.
- En fin de vie, les probabilités de panne augmentent rapidement. Deux vieux disques deviennent problématiques.
- Il y a parfois des lots de disques non fiables avec fin de vie prématûrée. Plusieurs vont acheter des disques de lots / modèles différents.
- Vol, feu, foudre, surtension...



# Tolérance aux pannes

- 1 Terminologie et modèles
- 2 Réduction et masquage des fautes
- 3 RéPLICATION
- 4 Le consensus en réparti
- 5 Transactions réparties
- 6 Calcul de disponibilité
- 7 Cas exemples: les limites de la tolérance aux pannes



## Copies de sauvegarde

- Un administrateur système a effectué des copies de sauvegarde, pendant plus d'une année, sans remarquer de problème, avant qu'un usager demande de récupérer un fichier à partir des copies. L'unité de ruban qui prenait les copies était défectueuse et les rubans illisibles.
- Un usager avisé, intégrant un nouveau groupe, feint de perdre un fichier afin de le faire relire et de s'assurer que la procédure de copies de sauvegarde fonctionnait bien. Quelques temps plus tard, son disque tombe en panne. Une partie des fichiers manquants étaient sur le ruban relu qui avait été oublié dans l'unité de ruban et son contenu écrasé.
- Un administrateur conscient faisait des copies de sauvegarde et effectuait régulièrement des tests. Un jour, l'unité de ruban est tombée en panne et il a découvert qu'elle était désalignée car aucune autre unité de ruban compatible ne pouvait relire les rubans.

## Mémoire du Xerox Alto

- Système de mémoire du Maxc avec correction d'un bit en erreur et détection de deux bits en erreur. Aucune erreur rapportée dans les fichiers d'erreur.
- Le même système est repris pour le Alto mais avec détection pour 1 bit seulement. Aucune erreur pour 6 mois. Avec le logiciel d'édition plein écran Bravo, de nombreuses erreurs sont apparues subitement.
- Sur le Maxc, une mauvaise configuration faisait que les erreurs de parité n'étaient pas rapportées. De plus, certains patrons de bits causent beaucoup plus d'erreur dans les circuits de mémoire, ce qui est arrivé avec Bravo sur le Alto.
- Sur le Alto 2, avec des circuits de mémoire plus récents, les concepteurs ont remis la correction et la parité. Le système était très fiable mais ils ont découvert éventuellement que 25% de la mémoire n'était pas protégée. Meilleurs circuits? Erreurs non détectées?

## Ordinateurs dans la navette spatiale

- Quatre ordinateurs redondants qui exécutent le même logiciel de manière synchrone, avec leurs sorties qui sont synchronisées et comparées 400 fois par seconde. Un cinquième ordinateur de relève qui exécute un autre logiciel programmé de manière indépendante.
- Lors du premier lancement, une erreur a été détectée laissant penser que plusieurs ordinateurs étaient défectueux et le lancement a été retardé.
- C'était un problème de synchronisation.
- Le second logiciel développé de manière indépendante ne protège pas contre les erreurs de spécification.

## La fusée Ariane 5

- Le système informatique a été repris tel quel de la fusée Ariane 4. Rien de plus conservateur et sécuritaire!?!
- La valeur d'accélération pour Ariane 4 ne pouvait pas dépasser une certaine valeur maximale.
- Ariane 5 était plus rapide, lorsque la valeur maximale a été dépassée, l'ordinateur a été déclaré fautif et l'ordinateur de relève a été activé.
- L'ordinateur de relève a détecté la même erreur d'accélération trop grande et s'est désactivé.
- Le premier lancement n'a pas bien fonctionné... .



## Commutateurs 4ESS de AT&T, 15 janvier 1990

- Nouvelle version de logiciel installée le décembre précédent sans problème apparent.
- Un problème indépendant en janvier cause une faute qui est détectée sur un commutateur (e.g., A). La procédure corrective est de faire redémarrer le commutateur, ce qui prend 4 à 6 secondes.
- Le système A envoie un message aux autres commutateurs (e.g., B) qu'il n'acceptera plus d'appel car il redémarre.
- Après le redémarrage, l'ancien système envoyait un message pour reprendre les appels, le nouveau recommence directement à initier des appels.
- Le commutateur B, en recevant un appel de A, comprend que A redévient fonctionnel, il réinitialise sa logique interne pour en tenir compte.
- Si B reçoit un autre appel de A pendant qu'il réinitialise sa logique, il devient confus et initie un redémarrage comme A l'avait fait.
- En peu de temps, le réseau de 114 commutateurs n'était plus qu'une cascade sans fin de redémarrages.

## L'appareil Therac 25

- Logiciel de commande pour un appareil à faisceau d'électron qui peut produire une lumière de positionnement, un faisceau d'électron ou un faisceau d'électron très puissant avec filtre spécial pour générer des rayons X.
- Lors d'une entrée du mode rayons X par erreur par l'opérateur, qu'il change tout de suite en mode faisceau d'électrons, la grande puissance était sélectionnée sans qu'il n'y ait de filtre pour rayons X en place.
- Le patient pouvait recevoir 100 fois la dose prévue.
- Le logiciel avait été pris tel quel d'une version antérieure. Le problème n'avait pas eu de conséquence avec l'ancien appareil car des mesures de protection par matériel étaient en place.

## Les pannes sont-elles vraiment indépendantes?

- Ordinateurs redondants avec mêmes logiciel/matériel, vulnérabilités, employés, bâtiment...
- Les trois capteurs de vitesse du vol Air France 447 de Rio à Paris sont pris dans la glace et donnent une vitesse de 0. L'autopilote de désengage et les pilotes réagissent mal.
- Plusieurs moteurs des avions L-1011 ont fait défaut simultanément. La même équipe avait fait l'entretien de ces moteurs et oublié un joint d'étanchéité torique dans le circuit hydraulique.
- Les 4 moteurs du vol British Airways 9 de Londres à Auckland se sont arrêtés presque simultanément près de Jakarta. La poussière d'un nuage volcanique a causé le problème.



# Informatique et développement durable

## Module 11

### INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

- ① L'ingénieur et le développement durable
- ② L'économie selon un modèle de développement durable
- ③ L'informatique et le développement durable
- ④ L'économie d'un centre de données selon un modèle de développement durable



# Informatique et développement durable

- ① L'ingénieur et le développement durable
- ② L'économie selon un modèle de développement durable
- ③ L'informatique et le développement durable
- ④ L'économie d'un centre de données selon un modèle de développement durable



# L'ingénieur

*Un ingénieur est un professionnel concevant des projets, si possible, par des moyens novateurs, et dirigeant la réalisation et la mise en œuvre de l'ensemble : produits, systèmes ou services impliquant de résoudre des problèmes techniques complexes. Il crée, il conçoit, il innove dans plusieurs domaines tout en prenant en compte des facteurs sociaux, environnementaux et économiques. Il lui faut pour cela, non seulement des connaissances techniques, mais aussi économiques, sociales, environnementales et humaines reposant sur une solide culture scientifique et générale.*

(Wikipedia)



# Le développement durable

*Le développement durable est une nouvelle conception de l'intérêt général, appliquée à la croissance économique et reconstruite à l'échelle mondiale afin de prendre en compte les aspects environnementaux et sociaux d'une planète globalisée. Le développement durable est un développement qui répond aux besoins du présent sans compromettre la capacité des générations futures à répondre à leurs propres besoins.*

(Wikipedia)



# Loi fédérale sur le développement durable

*Développement qui permet de répondre aux besoins du présent sans compromettre la possibilité pour les générations futures de satisfaire les leurs.*

(Canada 2008)



# Loi provinciale sur le développement durable

*Un développement qui répond aux besoins du présent sans compromettre la capacité des générations futures à répondre aux leurs. Le développement durable s'appuie sur une vision à long terme qui prend en compte le caractère indissociable des dimensions environnementale, sociale et économique des activités de développement.*

(Québec 2006)



# Les 16 principes du développement durable au Québec

- Santé et qualité de vie
- Équité et solidarité sociales
- Protection de l'environnement
- Efficacité économique
- Participation et engagement
- Accès au savoir
- Subsidiarité
- Partenariat et coopération intergouvernementale
- Prévention
- Précaution
- Protection du patrimoine culturel
- Préservation de la biodiversité
- Respect de la capacité de support des écosystèmes
- Production et consommation responsables
- Pollueur payeur
- Internalisation des coûts

# Code de déontologie de l'Ordre des Ingénieurs du Québec

*Dans tous les aspects de son travail, l'ingénieur doit respecter ses obligations envers l'homme et tenir compte des conséquences de l'exécution de ses travaux sur l'environnement et sur la vie, la santé et la propriété de toute personne.*



## Bureau Canadien d'Accréditation des Programmes de Génie (BCAPG)

### Les 12 qualités:

- Investigation
- Conception
- Utilisation d'outils d'ingénierie
- Travail individuel et en équipe
- Communication
- Professionnalisme
- Impact du génie sur la société et l'environnement
- Analyse de problèmes
- Déontologie et équité
- Économie et gestion de projets
- Apprentissage continu
- Connaissances en génie



# Conseil canadien des ingénieurs, 2016

Les lignes directrices du guide national sur le développement durable et la gérance environnementale, les ingénieurs:

- ① devraient maintenir et améliorer continuellement leur connaissance et leur compréhension des principes et des enjeux de la gérance environnementale et de la durabilité dans leur champ d'exercice;
- ② devraient avoir recours à l'expertise de spécialistes pour régler de manière appropriée les questions liées à l'environnement et à la durabilité, de même que pour améliorer leur compréhension de ces questions et les pratiques en la matière;
- ③ devraient tenir compte dans leur travail des valeurs sociétales aux échelons international, régional et local;



## Conseil canadien des ingénieurs, 2016 (suite)

- ④ devraient adopter des indicateurs de durabilité et des critères de gérance environnementale établis d'un commun accord dès la phase initiale des projets, et en évaluer périodiquement la mise en oeuvre par rapport à des cibles de rendement;
- ⑤ devraient déterminer les coûts et les avantages de la protection de l'environnement, des éléments de l'écosystème et de la durabilité lors de l'évaluation de la viabilité économique des travaux;
- ⑥ devraient intégrer la planification de la gérance environnementale et de la durabilité à la planification du cycle de vie et à la gestion des activités qui ont un impact sur l'environnement, et mettre en oeuvre des solutions efficaces et durables;



## Conseil canadien des ingénieurs, 2016 (suite)

- ⑦ devraient rechercher et diffuser des innovations qui permettent d'équilibrer les facteurs environnementaux, sociaux et économiques, tout en contribuant à la santé de l'environnement bâti et naturel;
- ⑧ devraient assumer un rôle de leadership dans les discussions concernant la durabilité et la gérance environnementale, et solliciter l'avis des parties prenantes et des experts accrédités de façon ouverte et transparente;
- ⑨ devraient s'assurer que les projets respectent les exigences réglementaires et législatives, par l'application des technologies et des procédures optimales et les plus économiquement viables;
- ⑩ devraient, s'il existe des risques de dommages graves ou irréversibles, mais pas de certitude scientifique, mettre en oeuvre des mesures d'atténuation des risques en temps voulu pour limiter le plus possible la dégradation de l'environnement.

## Autres lois et principes reliés au développement durable

- La procédure d'évaluation environnementale du BAPE: barrage, dragage, port, gare, aéroport, gazoduc, centrale, raffinerie, mine, incinérateur, nucléaire, certains types d'usines...
- Responsabilité élargie des producteurs (REP), ministère du développement durable, environnement et lutte contre les changements climatiques: récupération et valorisation des produits électroniques, des piles et batteries, des lampes avec mercure, des peintures et leurs contenants, et des huiles.



# Informatique et développement durable

- ① L'ingénieur et le développement durable
- ② L'économie selon un modèle de développement durable
- ③ L'informatique et le développement durable
- ④ L'économie d'un centre de données selon un modèle de développement durable



# L'économie

*L'économie est une discipline qui étudie l'économie en tant qu'activité humaine qui consiste en la production, la distribution, l'échange et la consommation de biens et de services.*

(Wikipedia)

- Un outil pour évaluer les bénéfices d'un scénario de projet et comparer diverses solutions alternatives.



## Scénario A

- Un jeune entrepreneur achète un immeuble à logements à crédit. Selon ses estimés, il escompte les revenus et dépenses suivants pour la prochaine année.
- Hypothèque, entretien, taxes, salaires, électricité... \$350000
- Revenu des loyers, \$300000.
- Est-ce rentable?
- Les données sont-elles complètes?



## Scénario B

- Un jeune entrepreneur hérite d'une ferme et de sa machinerie. Selon ses estimés, en travaillant dur, il escompte les revenus et dépenses suivants pour la prochaine année.
- Engrais, semences, carburant, entretien, taxes, salaires, électricité... \$300000
- Vente des produits agricoles, \$350000.
- Est-ce rentable?
- Les données sont-elles complètes?



## Que vaut notre temps?

- Le prix du salaire à payer pour embaucher quelqu'un qui fait le même travail.
- L'impôt peut biaiser le calcul, le revenu du travail peut générer une augmentation du capital, ce qui est imposé à moitié...
- Peinturer ou cuisiner soi-même, réduction de dépense plutôt que revenu imposé, en partie divertissant versus quelqu'un qui ne fait que cela...
- Le salaire est en relation avec l'offre et la demande, ce qui dépend du nombre de personnes aptes à faire ce travail et de l'intérêt que présente ce travail... avec quelques barrières artificielles qui perturbent cet équilibre.
- Les enquêtes sur le transport qui étudient les choix des usagers permettent d'évaluer le prix que les usagers mettent sur leur temps selon la situation (attendre dans son auto, attendre l'autobus, faire du vélo).

## L'économiste et les confitures

*Le jour que nous reçûmes la visite de l'économiste, nous faisions justement nos confitures de cassis, de groseille et de framboise.*

*L'économiste, aussitôt, commença de m'expliquer avec toutes sortes de mots, de chiffres et de formules, que nous avions le plus grand tort de faire nos confitures nous-mêmes, que c'était une coutume du moyen âge, que, vu le prix du sucre, du feu, des pots et surtout de notre temps, nous avions tout avantage à manger les bonnes conserves qui nous viennent des usines, que la question semblait tranchée, que, bientôt, personne au monde ne commettait plus jamais pareille faute économique.*

## L'économiste et les confitures

*Attendez, monsieur! m'écriai-je. Le marchand me vendra-t-il ce que je tiens pour le meilleur et le principal?*

*Quoi donc? Fit l'économiste.*

*Mais l'odeur, monsieur, l'odeur! Respirez: la maison toute entière est embaumée. Comme le monde serait triste sans l'odeur des confitures!*

*L'économiste, à ces mots, ouvrit des yeux d'herbivore. Je commençais de m'enflammer.*

*Ici, monsieur, lui dis-je, nous faisons nos confitures uniquement pour le parfum. Le reste n'a pas d'importance. Quand les confitures sont faites, eh bien! Monsieur, nous les jetons.*

*J'ai dit cela dans un grand mouvement lyrique et pour éblouir le savant. Ce n'est pas tout à fait vrai. Nous mangeons nos confitures, en souvenir de leur parfum. (Georges Duhamel Fables de mon jardin)*

## Scénario A

- Un jeune entrepreneur achète un immeuble à logements à crédit.
- Est-ce que l'hypothèque inclut une part de remboursement de capital, ce qui est un revenu?
- Combien l'immeuble prend-il de valeur?
- Il faut comparer ce scénario sur une longue période avec ne rien faire, en tenant compte de l'argent à ajouter à chaque année, de l'évolution du prix des dépenses et des loyers et surtout du produit de la vente de l'immeuble, après impôt, à la fin de la période. Il est possible que les pertes annuelles soient déductibles de l'impôt si cet entrepreneur a d'autres revenus.



## Scénario B

- Un jeune entrepreneur hérite d'une ferme et de sa machinerie...
- Combien d'heures devra-t-il travailler?
- Combien vaut la ferme?
- Qu'en est-il de la dépréciation sur la machinerie et les immeubles? En dépit de l'entretien, il faut éventuellement remplacer ces équipements coûteux.
- Est-ce qu'il y a un facteur de risque (récolte perdue)?
- Une étude économique classique comparera ce scénario (avec la vente au bout de 30 ans) avec la vente de la ferme, le placement du produit de la vente, et le salaire obtenu en travaillant un emploi équivalent... .
- Avec l'approche développement durable, il faut aussi regarder les aspects environnementaux (carburant, engrais chimique, pollution... ).

## Les données manquantes

- Le prix d'un bien ne tient généralement pas compte de l'appauvrissement des ressources épuisables (pétrole, minerai...) et de la pollution de l'environnement (accumulation des gaz à effet de serre, empoisonnement graduel du milieu, accumulation de déchets plastiques).
- Les redevances sur l'extraction minière et les taxes sur les produits polluants ne compensent souvent qu'en partie les dommages.
- La société sous sa forme actuelle se dirige vers un mur mais amorce un virage vers l'économie circulaire.
- Dans l'évaluation d'un projet, l'ingénieur doit tenir compte de ces données "manquantes" non reflétées dans les prix affichés.

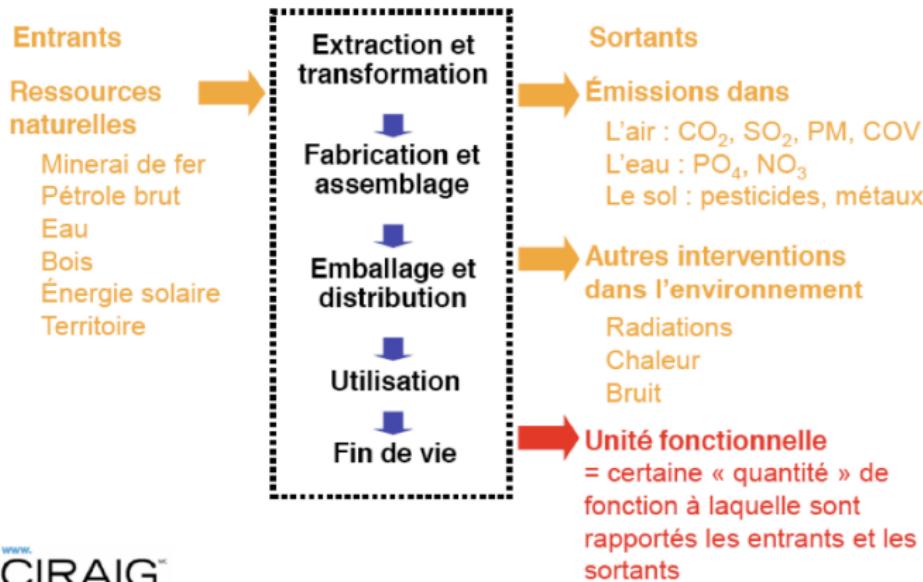
# Analyse du cycle de vie d'un produit ou projet

- Tenir compte de l'impact environnemental
- Considérer toutes les phases : acquisition des ressources, fabrication, distribution, utilisation, gestion en fin de vie.
- Mesurer l'impact environnemental à plusieurs niveaux: santé humaine, écologie, changements climatiques, utilisation des ressources.
- Il est souvent difficile mais pas nécessairement impossible de mettre un prix sur ces différents impacts. Bourse du carbone
- Approche Cycle de Vie



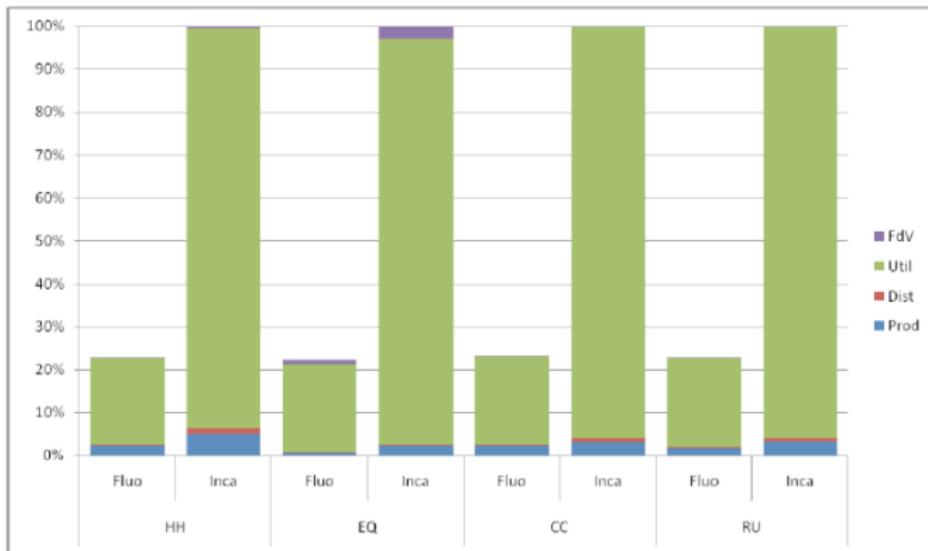
# Intrants et sortants pour un projet

## L'inventaire du cycle de vie



# Cycle de vie des ampoules

## Résultats d'indicateur pour les ampoules



# L'écoblanchiment

*L'écoblanchiment, ou verdissage, aussi nommé greenwashing, est une expression désignant un procédé de marketing ou de relations publiques utilisé par une organisation (entreprise, administration publique nationale ou territoriale, etc.) dans le but de se donner une image écologique responsable. La plupart du temps, l'argent est davantage investi en publicité que pour de réelles actions en faveur de l'environnement.*

(Wikipedia)

# Informatique et développement durable

- ① L'ingénieur et le développement durable
- ② L'économie selon un modèle de développement durable
- ③ L'informatique et le développement durable
- ④ L'économie d'un centre de données selon un modèle de développement durable



# L'informatique et le développement durable

- L'informatique, vecteur de dématérialisation, de procédés plus efficaces et d'optimisation.
- Les coûts de l'informatique, au niveau du client mais plus encore de la face cachée que constituent les centres de données.



## Dématérialisation

- Le papier est remplacé par des versions électroniques (factures, documents, livres, formulaires, cartes d'embarquement...).
- Le travail à distance, usuellement à l'aide d'un ordinateur en réseau.
- Les vidéo-conférences qui remplacent les réunions en personne.
- Les jeux vidéo et systèmes immersifs plutôt que d'aller à l'extérieur ou voyager.
- Paperless office is pure fiction (CBC News 2006)! La consommation de papier dans les bureaux a dans un premier temps augmenté avec la démocratisation de l'impression, connu un sommet en 2015 et commence à baisser depuis.
- Cette baisse a commencé plus tôt (2012) en Amérique du nord et en Europe mais était contrebalancée par l'augmentation en Asie.



## Informatisation de procédés

---

- Epandage d'engrais en fonction de la position, basé sur des analyses de sol en plusieurs lieux.
- Conduite autonome d'avion, de camion, d'auto...
- Découpage laser commandé par ordinateur.
- Moteur à vitesse variable pour les ventilateurs.
- L'électronique pour la commande ne cesse de diminuer de prix.



# Optimisation

- Optimisation du profil d'une aile d'avion, d'une carrosserie de voiture ou des pales d'une turbine.
- Optimisation du chemin pour la livraison de colis pour les conducteurs de camion.
- Modélisation 3D en architecture.
- Calcul des horaires pour le personnel (trains, avions, hopitaux...).
- Prévisions météorologiques.
- Planification de la demande en fonction de divers paramètres (e.g. canicule prévue versus climatiseurs).



# Informatique et développement durable

- ① L'ingénieur et le développement durable
- ② L'économie selon un modèle de développement durable
- ③ L'informatique et le développement durable
- ④ L'économie d'un centre de données selon un modèle de développement durable



## Le client informatique

- Les coûts et problématiques au niveau du client sont semblables et un sous-ensemble de ceux rencontrés dans les centres de données.



## L'économie d'un centre de données

- Construction / fabrication: **construction du centre de données** et de ses infrastructures; **fabrication de l'équipement**.
- **Transport des matériaux et de l'équipement** vers le site.
- Opération du centre de données: **consommation** d'énergie, fournitures, entretien, employés; **renouvellement des équipements** informatiques **aux 3 à 5 ans** typiquement.
- Fin de vie et démantèlement: recyclage des équipements et peut-être du bâtiment, remise en état des lieux.
- Pour chacune de ces étapes, considérer les coûts directs ainsi que l'impact sur la santé humaine, l'écologie, les changements climatiques et l'utilisation des ressources.



## Réaliser une analyse

- Obtenir toutes les données spécifiques aux équipements choisis, aux constructeurs, aux fournisseurs...
- Ou utiliser un mélange de données moyennes de l'industrie (e.g., béton, serveurs) et de données spécifiques (e.g., énergie fournie par Hydro-Québec).
- Mandater un groupe spécialisé en Analyse du Cycle de vie (e.g. CIRAIQ).
- Groupements pour l'informatique durable (Green ICT)  
<https://www.thegreengrid.org/>  
<http://greenict.ieee.org/standards>
- Centre de données Google
- Centre de données Microsoft
- Le projet Kolos



## Exemple d'analyse d'un centre au Royaume-Uni

- Bâtiment de 2 étages avec armature en acier.
- 42500 m<sup>2</sup> dont 8100m<sup>2</sup> pour l'équipement.
- 56600 serveurs 1u à environ 325 Watts (18.4MW).
- Environ 3000 chassis de 20 serveurs.
- Environ 2 à 3 m<sup>2</sup> et 3 à 10 kW par chassis.
- Refroidissement à air (17500 kW).
- UPS pour quelques minutes.
- Génératrices 46000kVA avec redondance.
- Eclairage efficace avec détecteurs de mouvement.



## Calcul de l'impact

- Fabrication et transport des matériaux utilisés (acier, béton, infrastructure électrique...), données typiques utilisées.
- Construction sur le site, impact négligeable.
- Acquisition des équipements informatiques et de réseautique avec un plan de renouvellement (e.g., aux 3 ans), données typiques utilisées.
- Opération pendant la vie utile du centre, principalement l'électricité dont l'impact dépend de la source (le Royaume-Uni utilisait beaucoup de charbon et maintenant du gaz naturel).
- Démolition sur le site, négligeable.
- Recyclage des matériaux et équipements, données typiques utilisées.

<https://link.springer.com/article/10.1007/s11367-014-0838-7>

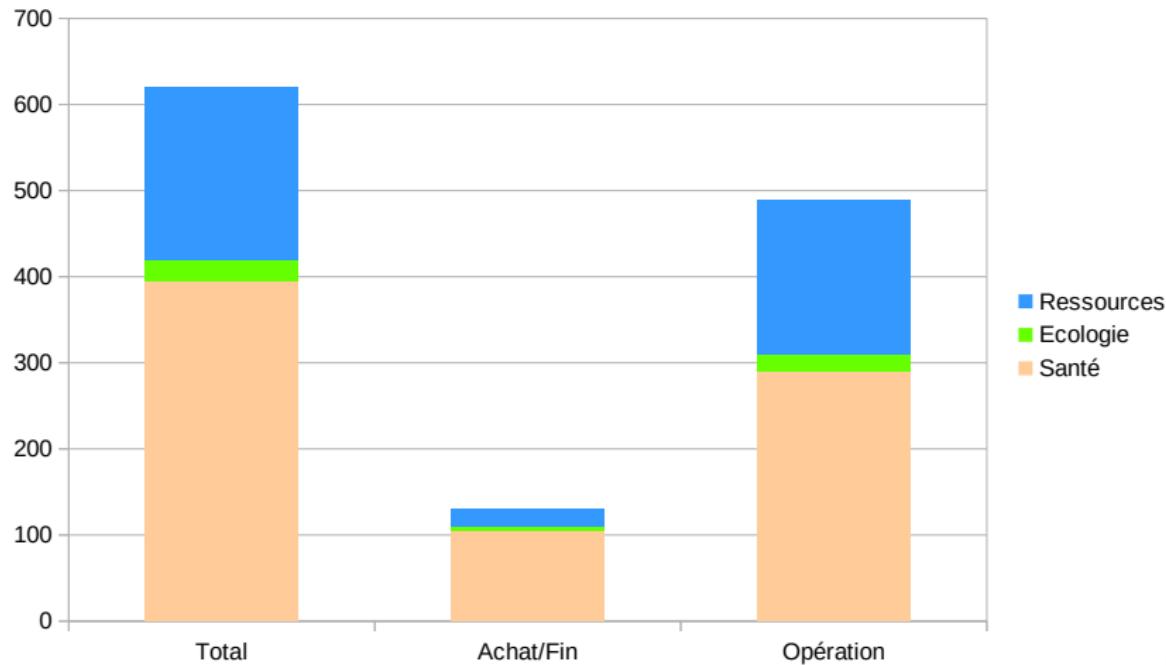
## Consommation électrique

Catégorie	Puissance
Serveurs	9000 kW
Stockage	2600 kW
Réseau	1300 kW
Eclairage	16.8 kW
UPS (pertes)	1096 kW
Distribution (pertes)	258 kW
Ventilateurs	722 kW
Refroidisseurs	51.6 kW
Humidificateurs	12.9 kW
Pompes	12.9 kW

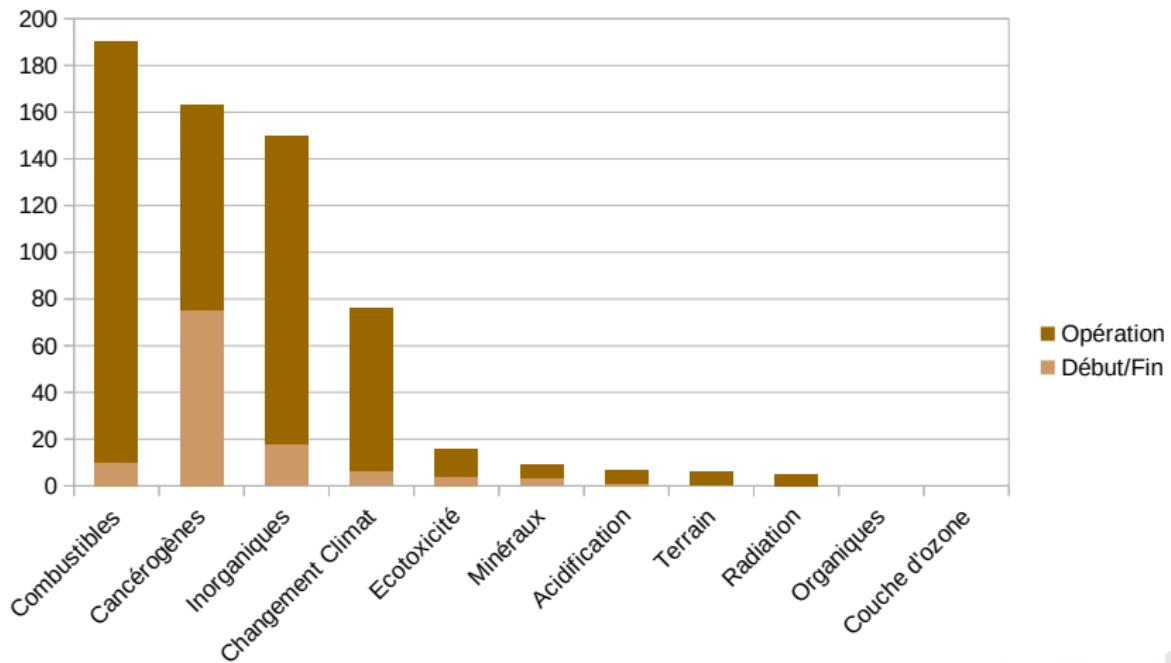
# Destination des matériaux en fin de vie

Matériaux	Enfoui	Recyclé
Asphalte	39%	61%
Terre	53%	47%
Mélange	73%	27%
Bois	20%	80%
Métal	30%	70%
Plastique	36%	64%
Gypse	40%	60%
Isolant	90%	10%
Liquide	2%	98%
Acier	1%	99%
Acier galvanisé	15%	85%
Acier renforcé	8%	92%

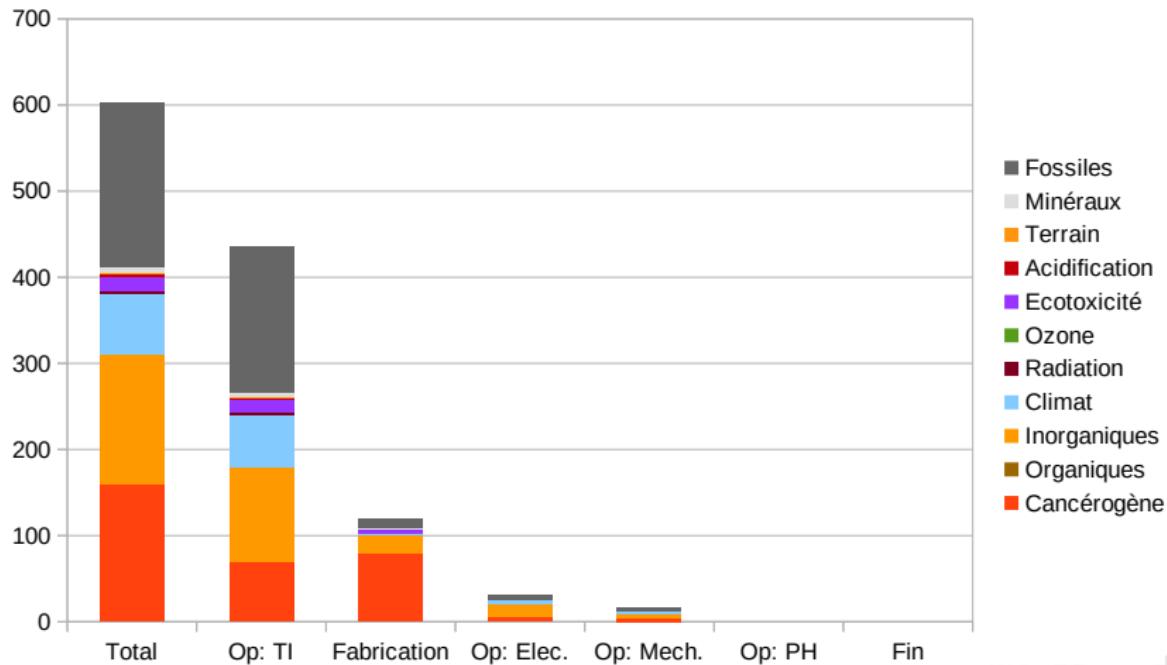
# Cycle de vie du centre de données



# Cycle de vie du centre de données



# Cycle de vie du centre de données



## Principaux impacts

- Utilisation du gas pour produire l'électricité pendant l'opération.
- Utilisation de l'acide sulphurique pour le raffinage du cuivre et de l'or utilisés pour fabriquer les équipements.
- Impact sur la santé dû aux produits cancérogènes comme l'arsenic et le cadmium et aux émissions de gaz carbonique et oxyde d'azote pour la fabrication des équipements.
- Impact sur l'écologie dû au gaz brûlé pour l'électricité et au rejet de l'acide sulphurique pour le raffinage.



## Conclusion

- Le développement durable, basé sur une économie circulaire, ajoute une dimension fondamentale à l'économie et la planification de projet.
- Une comptabilité séparée est souvent utilisée pour les coûts directs (\$) versus les dommages (mauvais points pour la santé, l'environnement, le climat et les ressources).
- Les coûts réels des dommages sont souvent astronomiques et difficiles à estimer.
- L'impact de l'électricité utilisée pendant l'opération varie beaucoup selon sa source.
- Des panneaux solaires diminuent considérablement l'impact en opération mais augmentent beaucoup celui de la fabrication.
- L'indiscipline au niveau mondial rend le problème très difficile: "pas dans ma cour", "laissons les autres faire des efforts", "après moi le déluge".



# Conclusion et exemple

## Google un très grand système réparti

Module 12

INF8480 Systèmes répartis et infonuagique

Michel Dagenais

École Polytechnique de Montréal  
Département de génie informatique et génie logiciel

# Sommaire

---

① Introduction sur Google

② Infrastructure logicielle

③ Conclusion



# Google un très grand système réparti

1 Introduction sur Google

2 Infrastructure logicielle

3 Conclusion



# Google

- Projet de Ph.D. commencé en 1996, compagnie démarlée en 1998.
- Croissance accélérée suivie de plusieurs acquisitions pour développer de nouveaux services.
- Recherche Web, GMail, AddWords, AddSense, Google Maps, Google Earth, Google Docs, Google News, Google Books, Google Translate, Picasa, Youtube, Android, Google+...
- Le plus gros système réparti au monde.
- Services avec des contraintes différentes (recherche vs courriel vs édition de documents partagés).
- Petit nombre de technologies sous-jacentes, simplicité, efficacité et surtout possibilité de mise à l'échelle.
- 40,000 requêtes par seconde, 24 heures sur 24, 3.5 milliards par jour.

## Infrastructure

- Réseau privé de fibres optiques, connexion directe aux principaux fournisseurs Internet (e.g. Videotron, Bell, Rogers), évite les frais de transfert.
- Noeuds individuels de \$1000 avec 2TB disque, 16GB RAM, Linux.
- De 2 à 3% de pannes matérielles par année, 95% reliées aux disques ou à la mémoire vive.
- Problèmes logiciels (i.e. doit redémarrer) rares mais 10 fois plus fréquents que problèmes matériels.
- Chassis avec 80 nœuds, 40 de chaque côté, plus commutateurs réseau.
- Grappes de 30 chassis avec commutateurs redondants vers l'Internet.
- Plus de 200 (?) grappes réparties dans un grand nombre de centres de données un peu partout à travers le monde.
- 1/2 million de noeuds!?! 1 exabyte? Exascale computing!

# Google un très grand système réparti

1 Introduction sur Google

2 Infrastructure logicielle

3 Conclusion



# Infrastructure logicielle

- Communication: encodage et RPC avec Protocol Buffers, communication de groupe avec Publish Subscribe.
- Coordination: Chubby.
- Stockage de données: GFS et BigTable.
- Traitement en réparti: MapReduce, Sawzall.
- Monitoring et analyse de performance: ktrace, ftrace, perf, Ittng; Dapper; RockSteady.
- Système de compilation: Google Build System.



# Publish Subscribe

- Messages formattés avec Protocol Buffers.
- Messages de différentes sources à distribuer en temps réel, de manière fiable, à un grand nombre de récipiendaires.
- Différents canaux sont définis.
- Message: entête, mots-clés, contenu.
- Un client peut s'enregistrer auprès d'un ou plusieurs canaux et spécifier un filtre (basé sur les mots-clés).
- Arbres redondants de distribution des messages, avec union des filtres des abonnés poussés plus haut dans l'arbre, pour minimiser les envois inutiles.



# Chubby

- Service de coordination basé sur Paxos.
- Système de fichiers (lire ou écrire en une seule opération, pas de modification) avec verrous.
- Utilisé pour stocker les métadonnées sur l'environnement (serveur élu pour GFS, BigTable...).
- Service de notification.
- RéPLICATION avec caches cohérentes (demande de modification, invalidation des caches, modification, réponse).
- Système de vérification de session (bail à court terme et demandes de renouvellement).
- Utilisé comme service de nom aussi.
- Un serveur pour des dizaines de milliers de nœuds.



## Election avec Chubby

- Le serveur maître Chubby devient non disponible.
- Les sessions des clients dépassent l'échéance de leur bail et toutes leurs valeurs en cache sont invalidées.
- L'attribut serveur élu devient invalide.
- L'élection prend place avec chaque candidat essayant d'accumuler le plus de promesses (de courte durée) de votes.
- Un serveur autre, X, devient non disponible.
- Il perd son verrou sur le fichier Chubby de serveur élu pour le service X.
- Le premier serveur de rechange qui obtient le verrou sur ce fichier s'inscrit comme nouveau serveur élu.



# BigTable

- Base de donnée simple optimisée pour la très grande taille.
- Rangée: clé d'accès, contenu de colonnes.
- Colonnes: familles de colonnes avec chacune un nom (nom de famille: nom de colonne).
- Versions de rangées indexées par estampille de temps.(row: string; column: string; time: int64): string
- Exemple: rangée pour une page Web avec clé URL, colonnes contenu, priorité, origine de liens vers cette page, catégorie, accès; rangée pour la session d'un usager et colonnes pour les opérations effectuées.



# Opérations supportées

- Création et effacement de tables.
- Création et effacement de colonnes.
- Lectures par rangées.
- Ecriture ou effacement de cellule.
- Remplacement atomique d'une rangée.
- Métadonnées sur les colonnes et tables (e.g. contrôle des accès).
- Ne possède pas toutes les opérations et les transactions d'une base de donnée conventionnelle.



## Organisation de BigTable

- Une table est décomposée en tablettes de 100 à 200MO.
- Librairie chargée avec le client.
- Serveur maître de BigTable, gère les serveurs de tablettes et leur association à des fichiers dans GFS.
- L'association de table à serveurs de tablette est stockée dans Chubby.
- La tablette racine d'une table (Meta0) contient les métadonnées pour les tablettes de métadonnées (Meta1, second niveau). Les tablettes de métadonnées pointent vers les tablettes de données.
- Chaque tablette est constituée de fichiers SSTable non modifiables (table de clés vs valeurs + index en arbre balancé).
- Les modifications sont accumulées dans un log, memtable, dont une copie est conservée en mémoire.
- De temps en temps, le contenu des memtable est utilisé pour mettre à jour et remplacer les SSTable originaux.

# Optimisation

- Les URL utilisés comme clé sont renversés: ca.polymtl.www/bottin pour maintenir ensemble les pages du même site.
- Les rangées consécutives sont stockées dans la même tablette.
- Les familles de colonnes peuvent se retrouver dans des fichiers différents. Ainsi, les colonnes utilisées par une même application sont concentrées dans un même fichier et permettent une meilleure localité des accès.
- Il est possible d'avoir des scripts écrits en Sawzall (lecture de table seulement) exécutés localement sur les serveurs.



# MapReduce

- Fournir les opérations map et reduce qui seront automatiquement réparties sur un grand nombre de nœuds.
- Map: opération à effectuer, ensemble de clés-valeurs en entrée et production d'un ensemble de clés-valeurs en sortie.
- Reduce: combine quelques réponses de map en une réponse agrégée.
- Exemple:
  - entrée: la BigTable du Web,
  - map: fonction qui recherche quelques mots clés,
  - sortie intermédiaire: sections de phrases, URL, priorité
  - reduce: fonction qui compte, et qui retient les résultats les plus prioritaires.



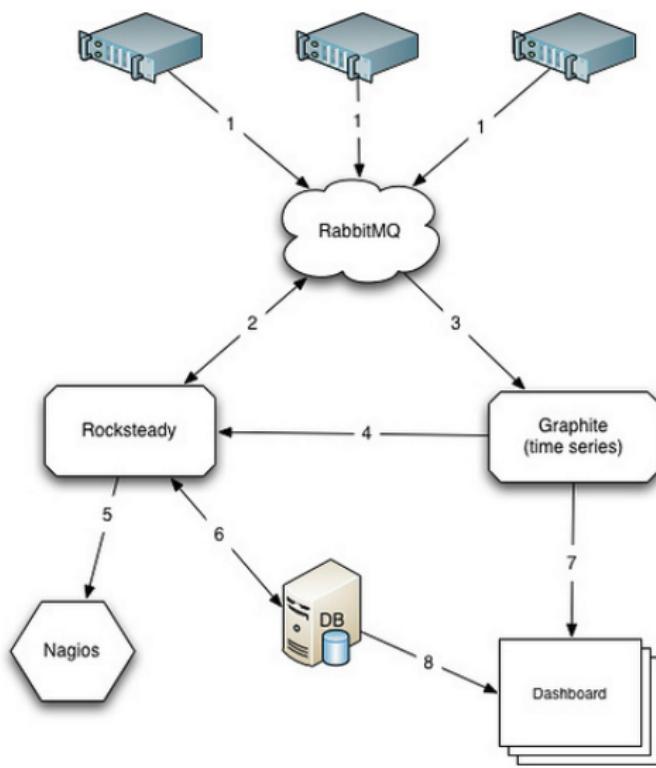
# Optimisation

---

- Découper l'entrée en morceaux de la taille d'un bloc GFS.
- Déetecter les nœuds en panne et redonner leur tâche à un autre noeud.
- DéTECTER les nœuds trop lents et redonner leur tâche à un autre noeud.



# Monitoring de l'infrastructure: métriques



# RockSteady

- Daemon qui collecte les métriques sur chaque nœud et les envoie à un canal RabbitMQ.
- Serveur qui s'abonne aux données de métriques, les agrège et produit au besoin des alertes Nagios.
- Outil de visualisation qui peut aussi s'abonner aux métriques.
- Commandes peuvent être envoyées aux daemon des nœuds pour modifier ce qu'ils doivent collecter.
- Plus efficace que la scrutation normalement effectuée par les outils de monitoring de réseau.



# Traces d'exécution au niveau du noyau

- Outils ktrace, ftrace ou lttng sur Linux.
- Pour des problèmes difficiles à reproduire (en production, non fréquent).
- Trace de tous les appels système, ordonnancements et interruptions sur un nœud écrits dans un tampon circulaire en mémoire (e.g. dernières secondes ou minute).
- Tous les noeuds en production peuvent générer une telle trace.
- Lorsqu'un problème est détecté, une copie de la trace en mémoire est sauvée pour être analysée.



# GWP: Google Wide Profiling

- Collecte d'échantillons de compteurs de performance ou de contenu de la pile d'exécution, par processus ou par nœud.
- Sélection d'un sous-ensemble des nœuds à échantillonner.
- Analyse des informations recueillies: conversion des adresses en symboles ou numéro de ligne de code source, agrégation des données.
- Présentation des résultats.
  - meilleure plate-forme pour chaque application;
  - aide à la compilation (e.g. sauts probables ou improbables);
  - coût global d'exécution par ligne;
  - vitesse relative des versions de chaque fonction ou programme.



# Dapper

- Chaque nouveau travail a un identificateur de contexte.
- Cet identificateur est passé lors de chaque appel RPC synchrone ou chaque requête asynchrone.
- Le début et la fin de chaque requête RPC sont écrits dans une trace avec l'identificateur.
- Il est alors possible de décomposer en sous-intervalles tout le temps requis pour effectuer un travail.
- Il est possible d'activer le traçage de requêtes pour un petit sous-ensemble des requêtes, obtenant ainsi un échantillonage (toutes les requêtes d'une fraction des travaux).



# Google Build System

- Pour chaque paquetage, liste de relations pré-requis, action et résultat, qui forment un graphe acyclique de dépendance.
- Le développeur voit localement les fichiers mais ils ne sont pas copiés sur sa station sauf si c'est vraiment requis.
- Chaque fichier est représenté par identificateur de contenu (somme de contrôle).
- Les fichiers de sortie sont stockés globalement avec la règle qui les a produits et les identificateurs des fichiers en entrée. Les sorties stdout et stderr sont aussi sauveées.
- Le travail de compilation s'effectue en parallèle sur un grand nombre de nœuds.



# Optimisation

- Les fichiers intermédiaires restent sur les nœuds de compilation où ils peuvent resservir à l'étape suivante.
- Si un fichier a déjà été compilé, le résultat est dans la cache globale et est réutilisé. Chaque fichier n'est donc compilé qu'une fois.
- Si un fichier est modifié mais sa version compilée demeure inchangée (e.g. ajout de commentaire à un programme), ceci est détecté et le changement ne se propage pas.



# Google un très grand système réparti

1 Introduction sur Google

2 Infrastructure logicielle

3 Conclusion



## Conclusion

- Le plus gros système réparti sur terre.
- Une compagnie qui connaît un succès phénoménal.
- Petit nombre de fonctions simples, très efficaces et prévues pour la mise à l'échelle.
- Beaucoup de réutilisation de leurs différents mécanismes de base, ce qui simplifie l'architecture, le monitoring et l'analyse de performance.
- Ils ont embauché un grand nombre d'ingénieurs très expérimentés qui les ont aidés à faire de bons choix.
- Les mauvais choix ne survivent pas à des problèmes de cette taille, de toutes manières.

