



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

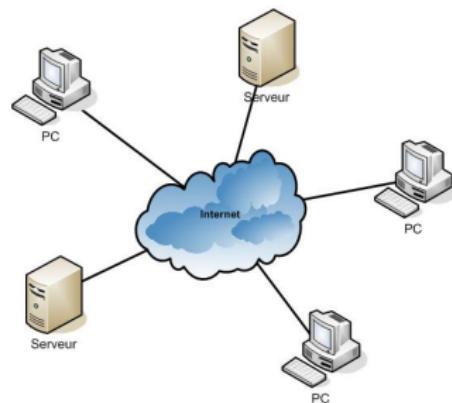
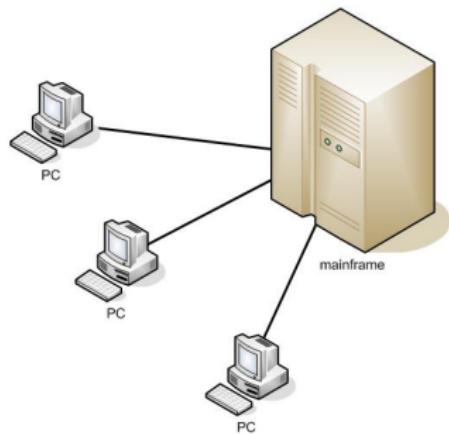


Évolution des organisations informatiques

- 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

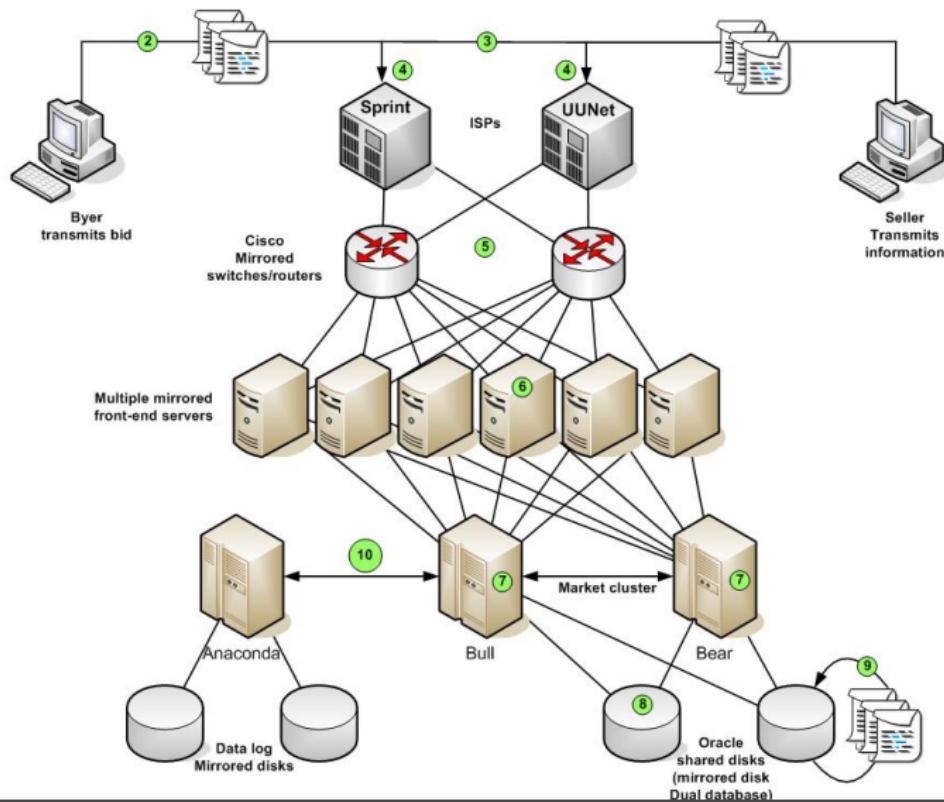


Système réparti

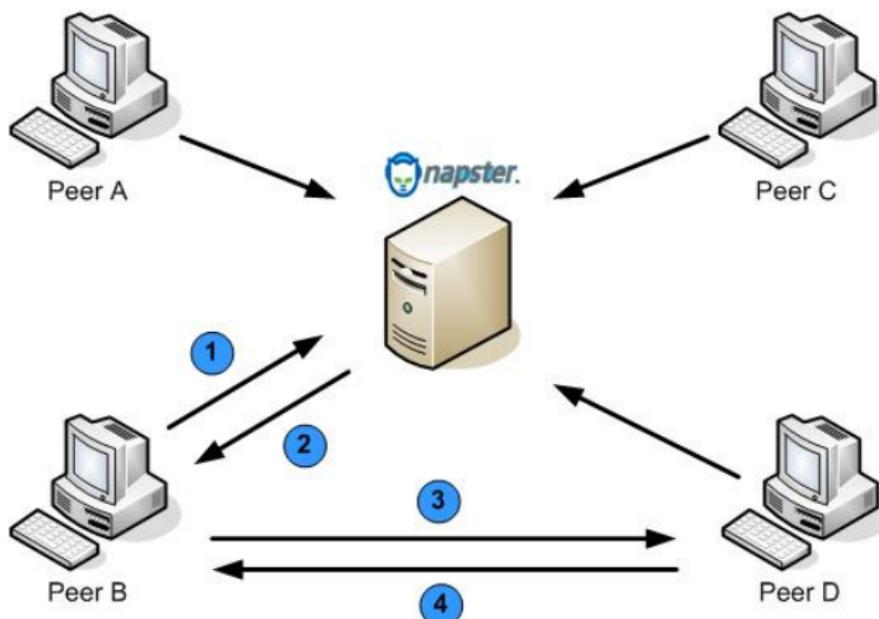
- Système dont les composantes sont réparties sur plusieurs ordinateurs en réseau et qui communiquent entre eux et coordonnent leurs actions uniquement par transmission de messages.
- Un ensemble d'ordinateurs indépendants qui, du point de vue de l'usager, apparaissent comme un système unique et cohérent.
- 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

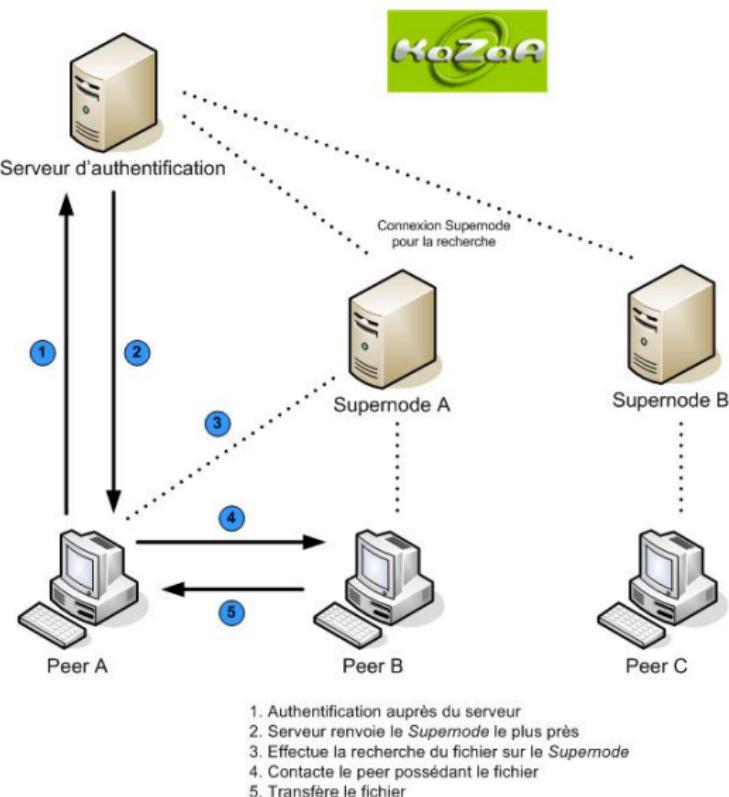


Réseau P2P centralisé

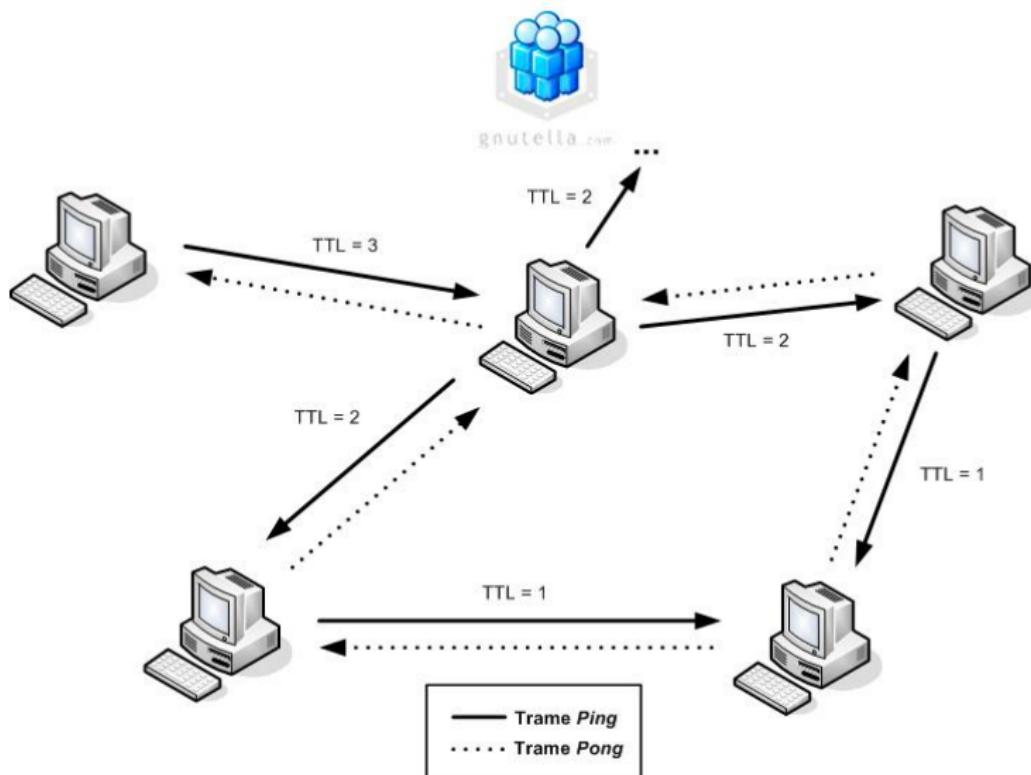


1. Recherche un fichier sur le serveur central
2. Serveur renvoie la liste des peer qui possèdent le fichier
3. Peer B contacte Peer D pour obtenir le fichier
4. Transfert du fichier

Réseau P2P hybride

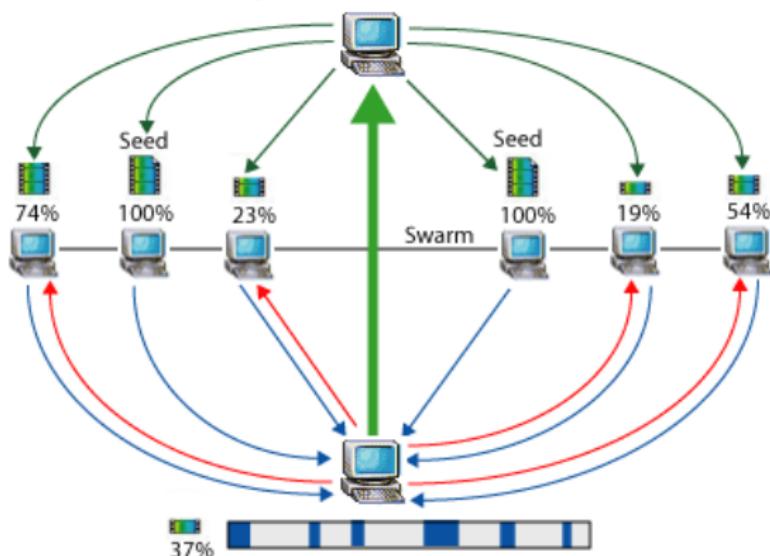


Réseau P2P décentralisé



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

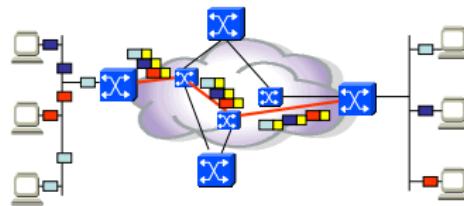
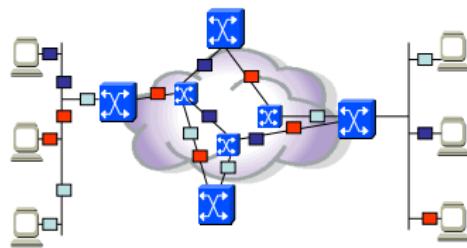
- Réseau mondial basé sur l'envoi de paquets;
- Familles de protocoles IP, UDP, TCP, FTP, SMTP, HTTP...
- Les aspects techniques et architecturaux sont régis par l'Internet Engineering Task Force (IETF);
- Les protocoles sont documentés dans les Request For Comments (RFC);
- Réseaux locaux avec routage 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.



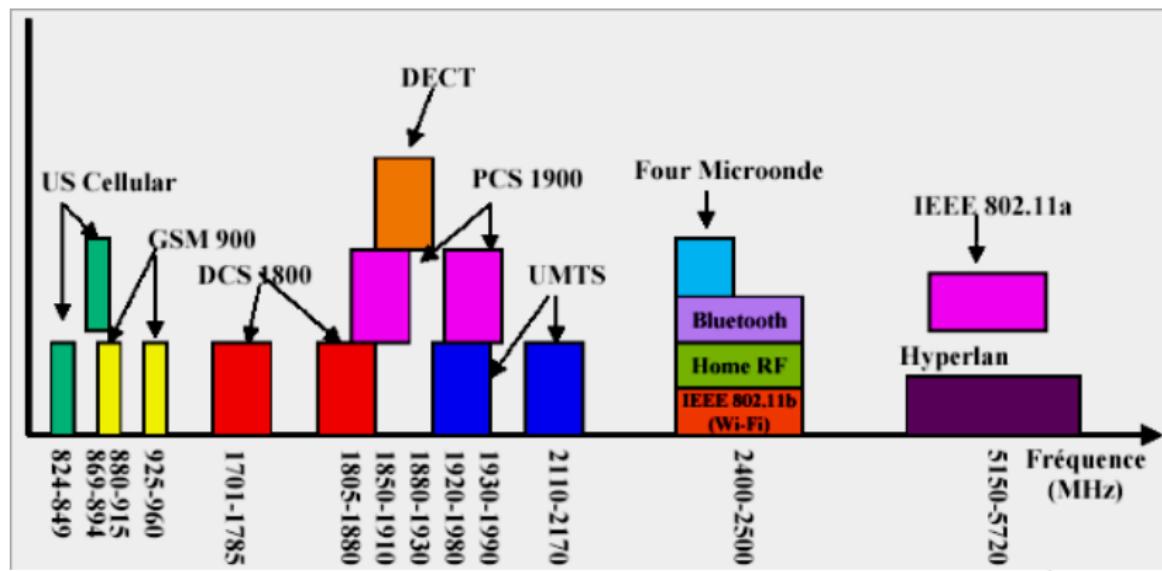
Multiprotocol Label Switching (MPLS)

- 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.

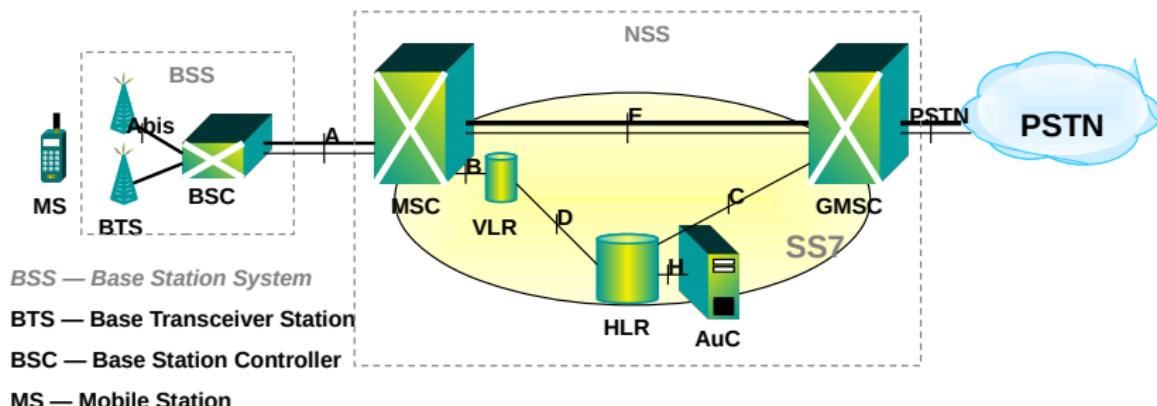
Routage classique IP versus MPLS



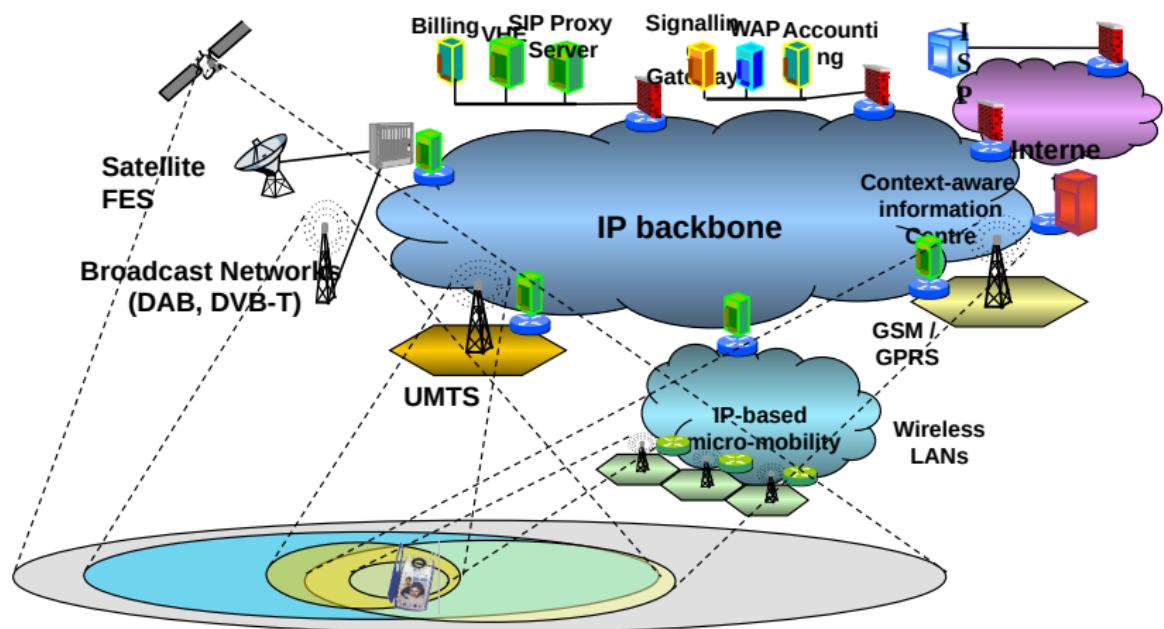
Les réseaux sans fil, fixes ou mobiles



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: λ ;
- Capacité de traitement des requêtes: μ 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

 **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)**



LBaaS



**Azure 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

- 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).

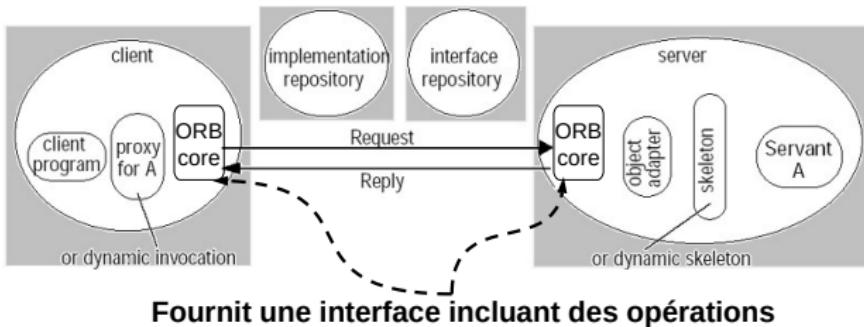


Composantes de CORBA

- 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

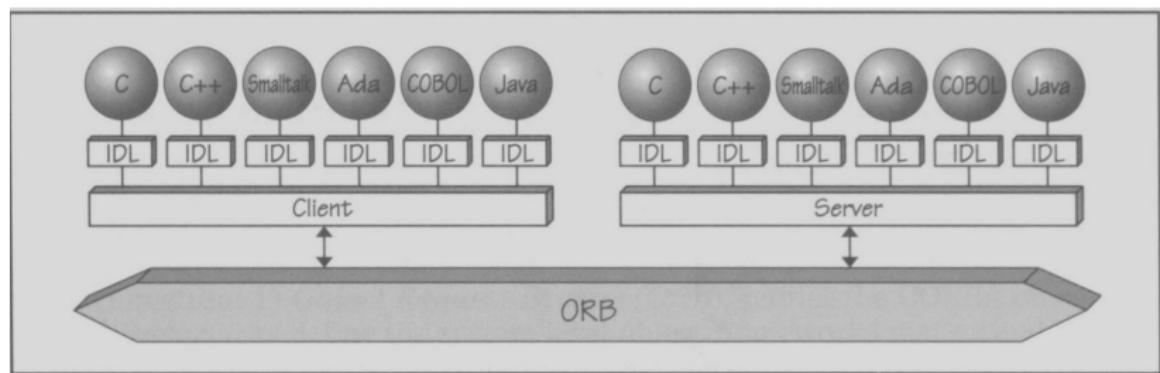


Langage de définition d'interface

- 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



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

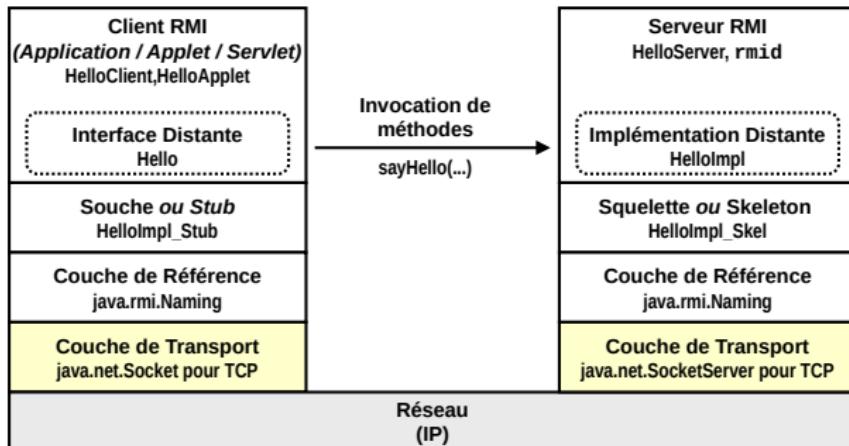
- Ne fonctionne que de Java à Java.
- S'applique à tout type qui implante l'interface Remote.
- Les méthodes doivent accepter l'exception RemoteException.
- Les arguments seront envoyés et la valeur renvoyée sera la réponse.
- Ces arguments doivent être des types primitifs, des objets réseau, ou des objets qui implantent l'interface Serializable (le graphe complet d'objets rejoint 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

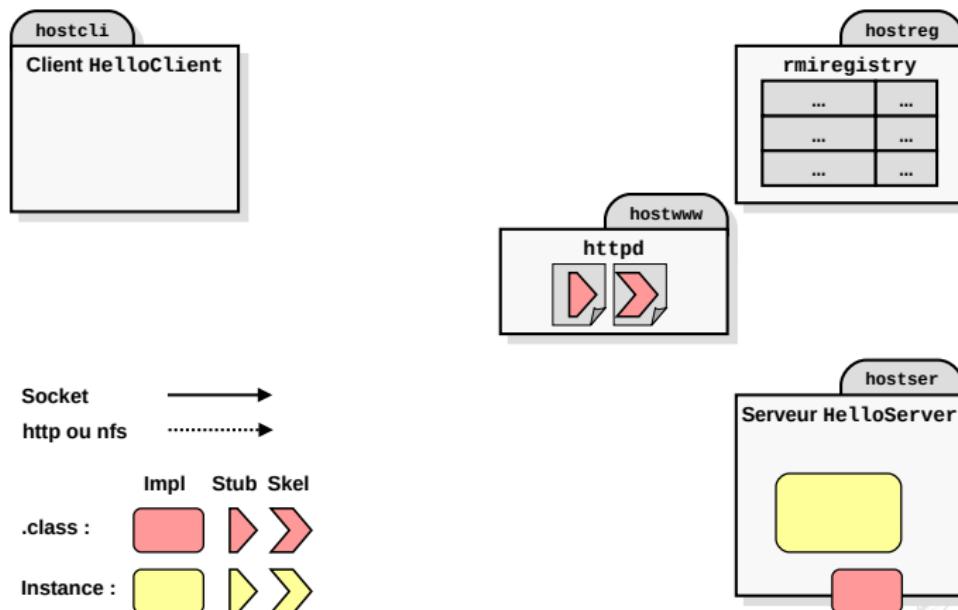
- Le RMIregistry maintient une table (nom, référence réseau) pour obtenir une référence à un objet désiré qui se trouve sur un ordinateur donné. Un nom a la forme:
`//hostname:port/objectName.`
- rmic peut être utilisé pour créer les proxy à partir du code des classes compilées. Le répartiteur est générique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



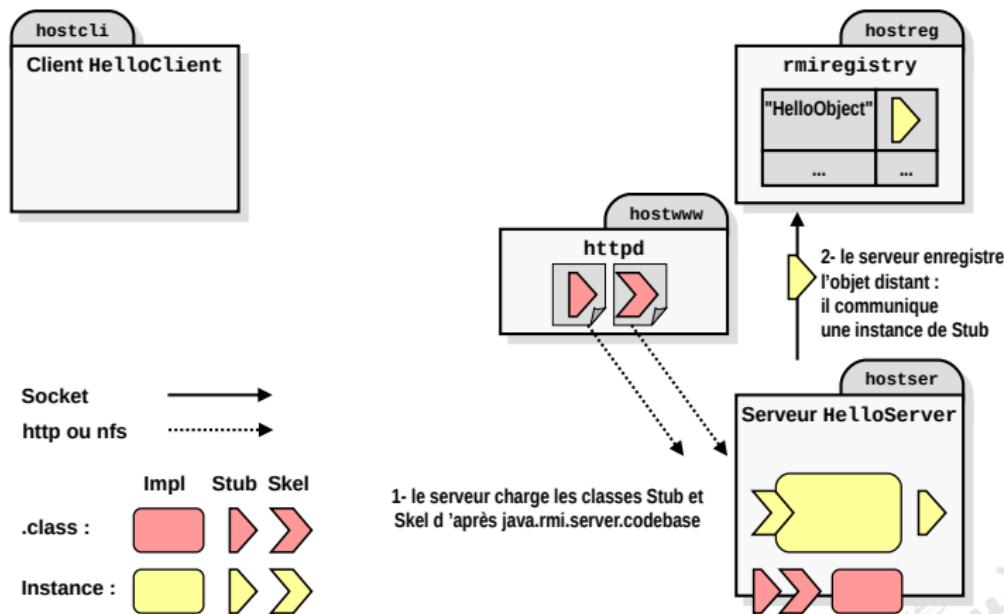
Structure logique des couches RMI



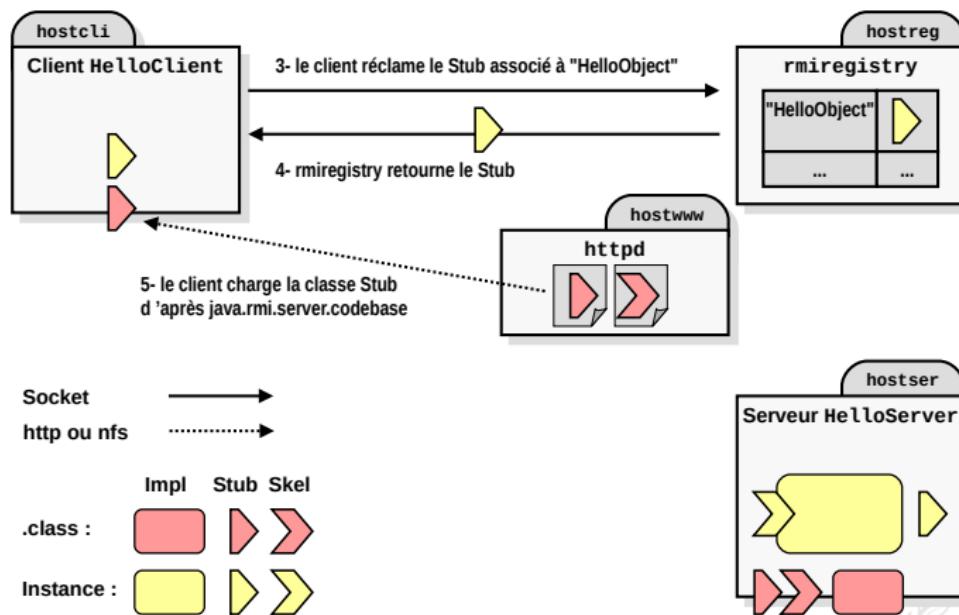
RMI: la configuration



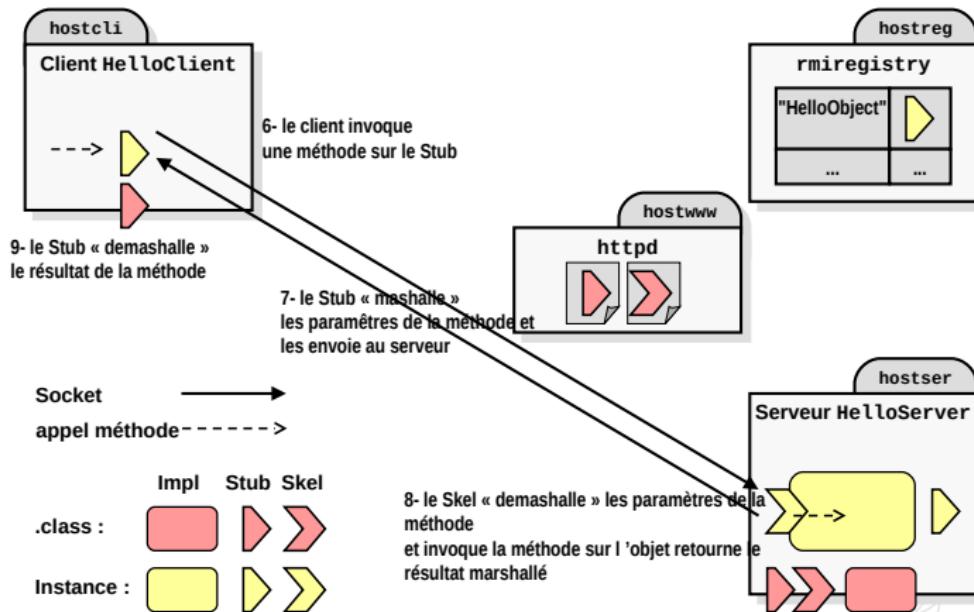
RMI: l'enregistrement de l'objet



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.

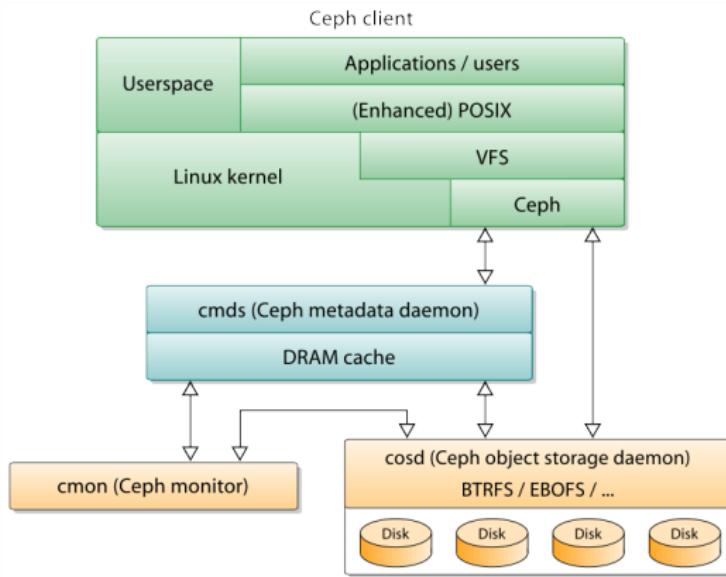
Google File System GFS

- 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.
- Une fonction de hachage simple associe le OID à un groupe de placement (PGID). Le groupe s'occupe de la réPLICATION.
- Le placement d'un groupe (PGID) sur des serveurs de fichiers (ceph-osd) est déterminé par un algorithme : Controlled Replication Under Scalable Hashing (CRUSH).
- 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.

