Module 9:

Google, le plus grand système réparti

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

Infrastructure

- Réseau privé de fibres optiques, connexion directe aux principaux fournisseurs Internet (e.g. Videotron, Bell, Rogers), évite les frais de transfert.
- Noeuds individuels de $1000 avec 2TB disque, 16GB RAM, Linux.
- De 2 à 3% de pannes matérielles par année, 95% reliées aux disques ou à la mémoire vive.
- Problèmes logiciels (i.e. doit redémarrer) rares mais 10 fois plus fréquents que problèmes matériels.
- Chassis avec 80 nœuds, 40 de chaque côté, plus commutateurs réseau.
- Grappes de 30 chassis avec commutateurs redondants vers l'Internet.
- Plus de 200 (?) grappes réparties dans un grand nombre de centres de données un peu partout à travers le monde.
- ½ million de nœuds!?! 1 exabyte? *Exascale computing*!

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

- **Protocol Buffers**
  - Semblable à SUN RPC, encodage binaire plutôt que le XML si populaire.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  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;
}
```

- **Protocol Buffers**

```
Person person;
person.set_name("John Doe");
person.set_id(1234);
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. Ce numéro permet de retrouver les bons champs même si un nouveau champ est inséré au milieu.
- Un message est une séquence de clé, type, valeur. La clé est le numéro de champs, le type est un octet (varint, float, length delimited...).
- 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.
- Il 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).

Publish Subscribe

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

Chubby

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

Election avec Chubby

- Le serveur maître Chubby devient non disponible.
- Les sessions des clients dépassent l'échéance de leur bail et toutes leurs valeurs en cache sont invalidées.
- L'attribut *serveur élu* devient invalide.
- L'élection prend place avec chaque candidat essayant d'accumuler le plus de promesses (de courte durée) de votes.

- Un serveur autre, X, devient non disponible.
- Il perd son verrou sur le fichier Chubby de serveur élu pour le service X.
- Le premier serveur de rechange qui obtient le verrou sur ce fichier s'inscrit comme nouveau seveur élu.

BigTable

- Base de donnée simple optimisée pour la très grande taille.
- Rangée: clé d'accès, contenu de colonnes.
- Colonnes: familles de colonnes avec chacune un nom (nom de famille: nom de colonne).
- Versions de rangées indexées par estampille de temps.

  (row: string; column: string; time: int64) → string

- Exemple: rangée pour une page Web avec clé URL, colonnes contenu, priorité, origine de liens vers cette page, catégorie, accès; rangée pour la session d'un usager et colonnes pour les opérations effectuées.

Opérations supportées

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

Organisation de BigTable

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

Optimisation

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

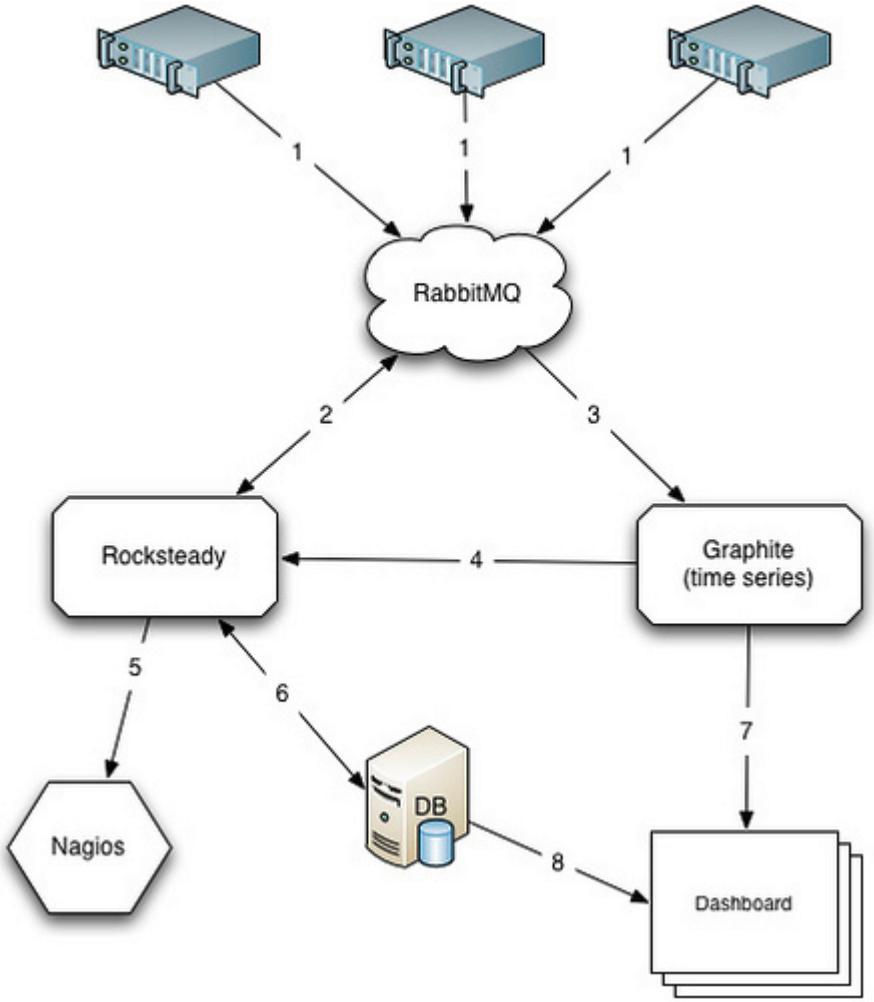
MapReduce

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

Optimisation

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

Monitoring de l'infrastructure: métriques



RockSteady

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

Traces d'exécution au niveau du noyau

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

GWP: Google Wide Profiling

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

Dapper

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

Google Build System

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

Optimisation

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

Conclusion

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